Universität für Bodenkultur Wien

Simulation von Waldbewirtschaftung mittels Deep Neural Networks (DNNs)

Masterarbeit
an der Universität für Bodenkultur, Wien
zur Erlangung des akademischen Titels Diplom-Ingenieur

Matthias Steinparzer

Wien, Februar 2020

Institut für Waldbau
Departement für Wald- und Bodenwissenschaften

Betreuer:
Prof. DI. Dr. Rupert Seidl und DI. Dr. Werner Rammer

INHALTSVERZEICHNIS

DANKSAGUNG

Besonderer Dank gilt meiner Familie und meinen Eltern, die mich immer unterstützen und mir dieses Studium erst ermöglicht haben. Bei stürmischer See seid ihr mein Anker.

Außerdem möchte ich mich bei meinem Betreuer Rupert Seidl für die Chance bedanken, Einblicke in dieses außergewöhnliche Feld der Forstwissenschaften erhalten zu haben, für die initiale Idee, deine Begeisterung und für die einzigartig gute Betreuung, selbst aus der Ferne. Ich wäre sehr bald in den komplexen Tiefen von deep learning und in unzähligen Programmen und Codezeilen versunken, ohne die intensive technische und mentale Betreuung von Werner Rammer. Vielen Dank für die umfassende Anleitung, deine Gelassenheit und Geduld, vor allem, wenn mir wieder einmal der Knopf nicht aufgehen wollte.

Abschließender Dank gilt meinen Freunden, die mich unter anderem daran erinnert haben, dass es noch andere wichtige Dinge außer die Masterarbeit gibt.

ABSTRACT

Anthropogenic influences already led to a global warming of 1 ° C above pre-industrial levels. This trend is likely to continue in the future. Forest ecosystems significantly contribute to slowing down climate change. Future ecosystems will be different from present systems, and it remains uncertain how changing climate conditions will affect forest ecosystems. Modeling is a powerful tool to understand complex interactions in ecosystems, and to study the effect of future climate on forests in scenario analyses. An important element in this context is the faithful representation of forest management in simulation models.

The aim of this master thesis was to test whether different forest management strategies can be simulated using approaches from artificial intelligence. Specifically, deep neural networks (DNNs) were used to learn forest management information. These DNNs were embedded into the meta-model SVD (*scaling vegetation dynamics*) which can efficiently simulate large forest areas.

To generate training data for the DNNs, the landscape model iLand (*the individual-based forest landscape and disturbance model*) was used. An artificial test landscape stocked with a normal forest was created, and forest management was simulated within the agent-based management model ABE (*the agent-based management engine*) in iLand, studying an unmanaged reference variant and six different management strategies (including two clearcut strategies, two shelterwood strategies, and two continuous cover strategies). The simulations were carried out for a period of 1000 years and repeated four times, with only the second half of the simulation period being analyzed (to control for effects of the initialization on the results). Two indicators (top height and leaf area index) were analyzed for all seven management options.

Three different deep neural networks were trained, tested and implemented into the meta-model SVD. Subsequently, the DNN-based simulations of management in SVD were evaluated against process model results from iLand.

The trained DNNs were successfully able to capture forest management (with prediction accuracies ranging from 70.8% to 77.5%). The level of information on forest management provided to the DNN had a clear impact on its performance, with increasing information improving prediction accuracy. SVD was well able to replicate the results obtained from

iLand. However, the spatial patterns and structural developments between SVD and iLand showed only moderate agreement. It was particularly challenging for DNNs to capture management in strategies where a spatial mismatch between the management intervention and the grain of the simulation existed. Overall, this thesis underlines the high potential of deep neural networks for simulating forest management.

ZUSAMMENFASSUNG

Anthropogene Einflüsse haben bereits zu einer globalen Erwärmung von 1°C gegenüber vorindustrieller Zeit geführt. Dieser Trend wird sich mit hoher Wahrscheinlichkeit auch in Zukunft fortsetzen. Waldökosysteme tragen wesentlichen dazu bei, den Klimawandel zu bremsen. Dennoch gibt es große Unsicherheiten, wie veränderte Klimabedingungen die zukünftige Rolle von Waldökosystemen beeinflussen werden. Modellierung ist ein vielseitiges Werkzeug und ermöglicht komplexe Abläufe in Ökosystemen zu verstehen und mittels Szenarioanalysen mögliche zukünftige Entwicklungen abzuschätzen. Ein wichtiges Element ist es dabei, die Effekte von Waldbewirtschaftung detailgetreu in Modellen abzubilden.

Ziel dieser Masterarbeit war es zu testen, ob verschiedene Waldbewirtschaftungsstrategien mittels Künstlicher Intelligenz simuliert werden können. Tiefe neuronale Netzwerke oder *deep neural networks* (DNNs) sollten aus aufbereiteten Daten Strukturen erkennen und somit Waldbewirtschaftung erlernen. Diese DNNs wurden in das Meta-Modell SVD (*scaling vegetation dynamics*) eingebettet, welches in der Lage ist, Walddynamik über große Studiengebiete hinweg zu simulieren.

Um die benötigten Referenzdaten zu generieren, wurde das Prozessmodell iLand (*the individual-based forest landscape and disturbance model*) verwendet. Es wurde eine Testlandschaft mit einem Fichten-Normalwald erstellt, für welche mit dem Agenten-basierten Bewirtschaftungsmodell ABE (*the agent based management engine for iLand*) eine Referenzvariante (keine Bewirtschaftung) und sechs verschiedene Bewirtschaftungsstrategien (je zwei Varianten eines Kahlschlag-, Schirmschlag- und Zielstärkennutzungssystems) simuliert wurden. Die Simulationen wurden für einen Zeitraum von 1000 Jahren durchgeführt und vier Mal wiederholt, wobei nur die zweite Hälfte des Simulationszeitraumes analysiert wurde, um die Effekte des Initialzustandes zu minimieren. Für alle sieben Bewirtschaftungsvarianten wurden die zwei Indikatoren Oberhöhe und Leaf Area Index analysiert.

Anschließend wurden drei verschiedene DNNs trainiert, getestet und in das Meta-Modell SVD implementiert. Die mit SVD und den darunterliegenden DNNs durchgeführten Simulationen wurden in weiterer Folge gegen die Ergebnisse des Prozessmodelles iLand evaluiert.

Die DNNs konnten erfolgreich trainiert werden und waren in der Lage, Waldbewirtschaftung zu abstrahieren (mit Vorhersagegenauigkeiten von 70.8% bis 77.5%). Der Informationsgrad zeigte deutlichen Einfluss auf die Leistung der drei verschiedenen Netzwerkvarianten. Je mehr Information zu Waldbewirtschaftungsstrategien dem DNN zur Verfügung standen, desto genauere Vorhersagen konnten getroffen werden. Die Ergebnisse von iLand konnten in SVD gut repliziert werden, jedoch wiesen die räumlichen Ergebnisse und strukturellen Entwicklungen zwischen den einzelnen DNN-Varianten aus SVD und iLand nur moderate Übereinstimmung auf. Im speziellen war es schwierig für DNNs, jene Waldbewirtschaftungsstrategien zu simulieren bei welchen die Managementeingriffe auf feinerer Skala als die Simulationen durchgeführt wurden. In Summe konnte die Arbeit zeigen, dass Methoden der Künstlichen Intelligenz und im speziellen DNNs großes Potential in der Simulation von bewirtschafteten Wäldern haben.

1 EINLEITUNG

Anthropogene Einflüsse haben bereits zu einer globalen Erwärmung von 1°C gegenüber vorindustriellen Werten geführt. Bei gleichbleibender Entwicklung wird es laut Prognosen zu einem Anstieg um 1,5°C zwischen 2030 und 2052 kommen (IPCC *et al.*, 2018). Laut Klimamodellen werden mit hoher Wahrscheinlichkeit die mittleren Temperaturen in terrestrischen und maritimen Systemen ansteigen und Hitzeextreme zunehmen, regional werden starke Niederschläge, Dürreperioden und Niederschlagsdefizite zunehmen (IPCC *et al.*, 2018). Wälder haben das Potential, das Treibhausgas CO_2 aus der Luft zu binden und gegen den Klimawandel zu wirken. Dieses Potential kann durch Aufforstungen und Wiederherstellung zerstörter Wälder ausgebaut werden (IPCC *et al.*, 2018). Bei fortschreitendem Temperaturanstieg können jedoch Wälder von einer Kohlenstoffsenke zur Kohlenstoffquelle werden (Hadden und Grelle, 2016). Eine der größten Herausforderungen für die Forstwirtschaft sind eben diese Unsicherheiten, welche mit dem Klimawandel einhergehen (Millar *et al*., 2007). Diese Unsicherheiten verlangen eine breite Palette an verschiedenen Herangehensweisen, um Waldökosysteme für kommende unausweichliche Veränderungen vorzubereiten (Millar *et al*., 2007).

Modellierung ist ein vielseitiges Werkzeug und kann neben oder in Kombination mit klassischen empirischen Wissenschaften (Seidl, 2017) dazu genutzt werden, komplexe Vorgänge besser zu verstehen und Lösungen zu erarbeiten. Es können Prognosen über zukünftige Entwicklungen von Ökosystemen erstellt werden und bieten somit einen Anhaltspunkt für BewirtschafterInnen und politische EntscheidungsträgerInnen (Rammer und Seidl, 2019a). Ökosystemforschung wird zunehmend komplexer, besonders im Hinblick auf Klimawandel und menschlichen Einfluss – Modelle können helfen, mit dieser Komplexität umzugehen (Seidl, 2017). Verschiedene Szenarien können simuliert und qualitative Erkenntnisse über mögliche Entwicklungen gewonnen werden (Millar *et al*., 2007). Mit der Verfügbarkeit von riesigen Mengen an Daten und steigender Computerleistung (Goodfellow *et al*., 2016) werden Modellierungen deshalb zunehmend nützlicher (Seidl, 2017). Skalierung und Skalierbarkeit ist in der Ökologie schon seit Jahrzehnten ein zentrales Thema (Schneider, 1994; Wiens, 1989) und um sich mit globalen Fragestellungen und zukünftigen Auswirkungen auseinandersetzen zu können, wird räumliche und zeitliche Skalierbarkeit (Seidl *et al.*, 2013; Seidl, 2017; Rammer und Seidl, 2019a) von Modellen immer wichtiger.

Es gibt bereits Simulationsmodelle, welche Waldbewirtschaftung sehr detailliert beschreiben können, wie das Submodell ABE in iLand (Rammer und Seidl, 2015), PICUS (Lexer und Hönninger, 2001), PROGNAUS (Sterba und Monserud, 1997), SILVA (Pretzsch *et al*., 2002) oder MOSES (Thurnher *et al*., 2017). Die Limitierung dieser Ansätze liegt in der Skalierbarkeit dieser detaillierten Simulationsmodelle, welche Berechnungen auf Ebene von einzelnen Bäumen oder sehr kleinen Einheiten anstellen. Detaillierte Modelle können aber verwendet werden, um daraus sogenannte Meta-Modelle abzuleiten (Urban, 2005), welche auch als Modell eines Modells (Acevedo *et al.*, 2001) oder groß-skalierte Simulationsmodelle (Cipriotti *et al.*, 2016) verstanden werden können. Will man dieselben Vorgänge räumlich und zeitlich nach oben skalieren und gleichzeitig schnelle Modelle entwickeln, kann man sich neuer Technologien bedienen, die unter den Überbegriff der Künstlichen Intelligenz fallen. Künstliche Intelligenz (KI oder *artificial intelligence* AI) ist 2020 wichtiger denn je und ein wachsendes Feld mit vielen praktischen Anwendungen (Tshitoyan *et al.*, 2019) und einer aktiven Forschungslandschaft (Goodfellow *et al*., 2016). Intelligente Software hilft Arbeiten zu automatisieren, Sprachen zu übersetzen und zu verstehen, Bilder zu interpretieren, medizinische Diagnosen zu stellen und die wissenschaftliche Forschung zu unterstützen (Chen *et al*., 2008; Lecun *et al*., 2015; Goodfellow *et al*., 2016). Dies gilt auch für Forschungsfelder der Ökologie (TensorFlow, 2017; Hart *et al.*, 2019; Rammer und Seidl, 2019b; Reichstein *et al.*, 2019). Diese Arbeit bedient sich schon bestehender und getesteter Modelle wie iLand (Seidl *et al*., 2012b; Silva Pedro *et al*., 2015), dem Submodell ABE (Rammer und Seidl, 2015) und dem Meta-Modell SVD (Rammer und Seidl, 2019a) und entwickelt neue tiefe neuronale Netzwerke mit Hilfe von *deep learning* (Lecun *et al*., 2015; Goodfellow *et al*., 2016) für Waldbewirtschaftungssimulationen.

2 FORSCHUNGSFRAGEN UND ZIELE

Nach ausgiebiger Recherche ist diese Diplomarbeit die erste Arbeit, die mit Hilfe von DNNs versucht, Waldbewirtschaftung zu simulieren. Es soll Licht auf die Frage geworfen werden, ob es grundsätzlich möglich ist, diese Technologie für diesen Bereich der Forstwirtschaft einzusetzen. Der ganze Entwicklungsprozess ist daher stark experimentell geprägt. Um die Eignung von *deep learning* und in weiterer Folge DNNs für Waldbewirtschaftungssimulationen zu prüfen, werden verschiedene Waldbewirtschaftungsvarianten mit verschiedenen Modellen umgesetzt und verglichen. Simulationen von Waldbewirtschaftungen können als eine Abfolge von einzelnen waldbaulichen Eingriffen oder als ganze Managementregime umgesetzt werden. Es soll deshalb untersucht werden, ob ein variabler Informationsgrad einen Unterschied in den Vorhersagen eines DNNs verursacht und ob diese Unterschiede über forstliche Parameter analysiert werden können. Des Weiteren soll geklärt werden, ob und mit welcher Genauigkeit *scaling vegetation dynamics* SVD (Rammer und Seidl, 2019a) mit den eingebundenen DNNs die Ergebnisse des prozessbasierten Simulationsmodells iLand reproduzieren kann.

Die Ziele im Speziellen sind:

(i) Kann Waldbewirtschaftung mit Hilfe von DNNs (*deep neural networks*) abstrahiert und in Computersimulationen abgebildet werden? Wie gut ist die Performance relativ zu detaillierten Managementmodellen?

(ii) Gibt es einen signifikanten Unterschied zwischen der Simulation einer Abfolge von expliziten Managementeingriffen mittels DNNs und der Simulation ganzer Managementregimes? Forstliche Parameter dienen als Indikatoren für mögliche Unterschiede.

(iii) Kann SVD strukturelle Unterschiede, welche durch Bewirtschaftungsunterschiede bedingt sind, darstellen? Sind diese Ergebnisse und Strukturen mit Ergebnissen des prozessbasierten Simulationsmodells iLand vergleichbar?

3 MATERIAL UND METHODEN

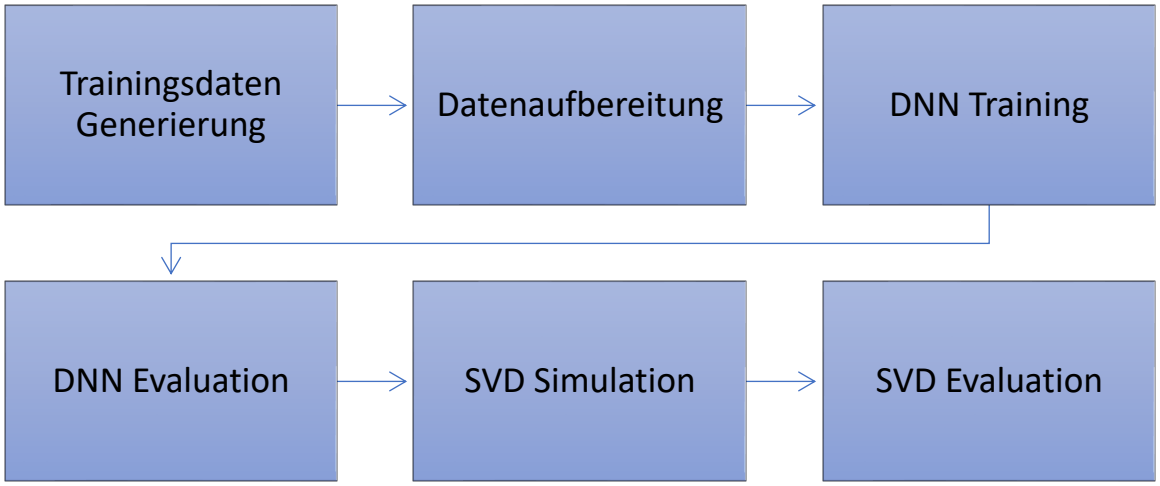


Abbildung 1: Das Flussdiagramm stellt die Vorgangsweise dieser Arbeit dar. Die Generierung von Trainingsdaten ist die Basis für alle weiteren Schritte (Datenaufbereitung, DNN Training, DNN Evaluation, SVD Simulation und SVD Evaluation).

Das Flussdiagramm in Abbildung 1 soll einen kurzen Überblick über den allgemeinen Aufbau und die Vorgehensweise dieser Arbeit geben. Nach Kapitel 3.1, wird beginnend mit der Trainingsdaten Generierung jeder Teil des Flussdiagramms abgehandelt. Die Ergebnisse in Kapitel 4 folgen (exklusive des Punktes Datenaufbereitung) demselben Schema.

3.1 Allgemeines und Werkzeuge

3.1.1 Von Künstlicher Intelligenz (KI) Machine learning (ML), Deep learning und Deep Neural Networks (DNNs)

In den frühen Tagen der Künstlichen Intelligenz wurden vor allem abstrakt und formal komplexe Probleme gelöst, die für den Menschen durchaus kompliziert, für eine Maschine aber relativ einfach zu bewältigen waren. Ein Beispiel wäre das Schachspiel, wo anhand einer relativ kurzen Regelliste ein Programmierer einem Computer alle nötigen Informationen beschreiben kann, um gegen einen Menschen zu siegen (Goodfellow *et al*., 2016). Beim Schach ist ein Computer einem Menschen schon lange überlegen, aber erst kürzlich wurden professionelle Go-Spieler (welches weitaus komplexer als Schach ist) von Googles entwickeltem AlphaGo-System geschlagen (Wick, 2017). Ein Gesicht oder gesprochene Worte zu erkennen, gehören aber zu den wahren Herausforderungen für eine Künstliche Intelligenz. Diese Tätigkeiten und Fähigkeiten sind vom Menschen einfach und intuitiv auszuführen, aber mathematisch und formal schwer zu beschreiben. Um sich als Computer in der alltäglichen Welt intelligent verhalten zu können, sind enorme Mengen an Daten notwendig (Goodfellow *et al*., 2016).

Damit KI-Systeme Aufgaben aus der realen Welt bewältigen können, brauchen KIs die Fähigkeit ihr eigenes Wissen aus Rohdaten zu extrahieren. Es gilt als große Herausforderung, dem Computer dieses oft subjektive und intuitive Wissen zu vermitteln. Diese Fähigkeit ist auch als *machine learning* (ML) bekannt. Mit der Einführung von ML konnten Computer plötzlich Aufgaben bewältigen und scheinbar subjektive Entscheidungen treffen, die Wissen aus der realen Welt voraussetzten. Zwei einfache ML-Algorithmen sind die logistische Regression oder der naive Bayes Klassifikator. Die Leistung von ML-Algorithmen hängt sehr stark von der Repräsentation oder Darstellung der verarbeiteten Daten ab (Goodfellow *et al*., 2016).

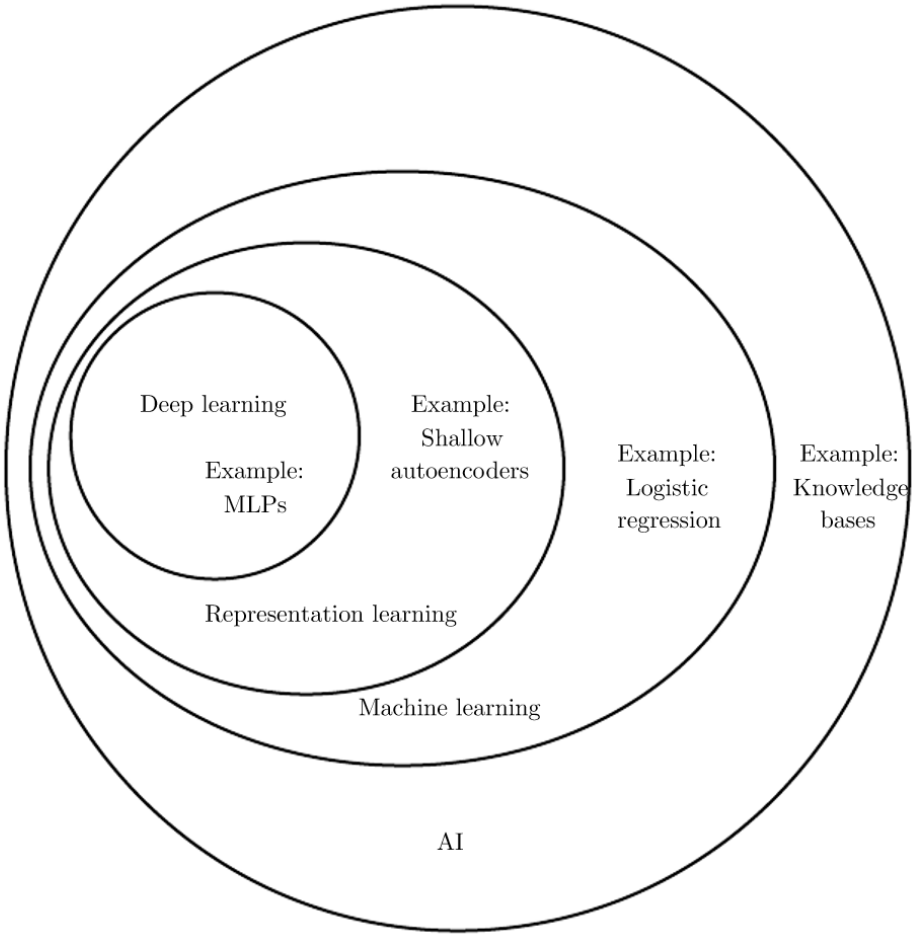


Abbildung 2: Das Venn Diagramm zeigt die Einbettung und Zuordnung von deep learning in dem technologischen Feld der Künstlichen Intelligenz (AI oder KI) inklusive Beispiele. Deep learning ist eine Art von representation learning, wobei representation learning wieder eine Art von machine learning ist, welches wiederrum eine Methode ist, um KI zu realisieren. Aus dem Buch Deep learning von Goodfellow *et al.,* (2016).

Die Lösung liegt darin, dem Computer zu erlauben von Beispielen zu lernen und die Komplexität der Welt durch eine Hierarchie von Konzepten zu verstehen und diese Konzepte wiederum durch vereinfachte Konzepte abzubilden. Dadurch, dass die Maschine Wissen aus gewonnener Erfahrung generiert, muss der Mensch das für die Maschine nötige Wissen nicht mehr manuell festlegen. Die Hierarchie von Konzepten ermöglicht es dem Computer komplexe Konzepte aus einfachen zu bauen. Will man diese Hierarchie von Konzepten und wie sie aufeinander aufbauen grafisch darstellen, erhält man einen tiefen Graphen mit vielen Schichten, woraus sich aus der Thematik Künstliche Intelligenz schlussendlich der Begriff *deep learning* (DL) (Abbildung 2) ableitet (Goodfellow *et al*., 2016). *Deep learning* kann nicht alle Probleme von *machine learning* lösen, es ist vielmehr als ein Werkzeug unter vielen zu sehen, um sehr große Datensätze zu bewältigen (Wick, 2017).

Vorläufer von *deep learning* gibt es unter verschiedenen Synonymen wie Kybernetik oder Konnektionismus bereits seit den 1940ern (Goodfellow *et al*., 2016). In erster Linie waren solche Modelle technische Systeme, die das biologische Gehirn als Vorbild hatten und deshalb auch als künstliches neuronales Netz (*artificial neural network*, ANN) bezeichnet wurden (Goodfellow *et al*., 2016). Die Basis jedes *deep learning* - Ansatzes ist das nach Rosenblatt (1958) benannte Perzeptron (siehe Abbildung 3). Ein Perzeptron ist eine Art eines künstlichen Neurons, es kann aber auch ein Netzwerk von mehreren künstlichen Neuronen sein (siehe Abbildung 4).

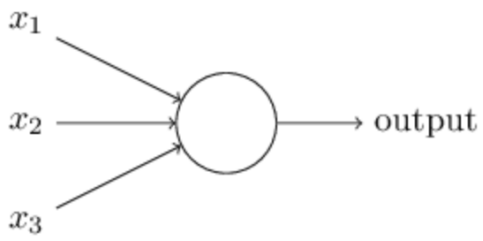


Abbildung 3: Die Grundversion eines Perzeptrons nach Rosenblatt (1958) besteht aus einem künstlichen Neuron mit anpassbaren Gewichtungen und einem Schwellenwert. Ein Perzeptron nimmt mehrere binäre (0 oder 1) Eingänge x1, x2, … und produziert einen binären Ausgang (output) (Nielsen, 2015)

Um einen Ausgangswert oder *output* zu berechnen, führte Rosenblatt Gewichte ein, welche als reale Zahlen die unterschiedliche Relevanz der Eingänge eines künstlichen Neurons bestimmen. Ein Eingang mit niedriger Gewichtung beeinflusst das Ergebnis am Ausgang weniger, als ein Eingang mit höherer Gewichtung (Nielsen, 2015). Bei dem ursprünglichen Perzeptron von Rosenblatt (1958) war der Wert des Ausgangs binär, d.h. entweder 0 oder 1. Moderne neuronale Netzwerke verwenden andere Modelle künstlicher Neuronen wie etwa das Sigmoid Neuron, welches einen beliebigen Ausgangswert des Intervalls [0,1] produzieren kann (Nielsen, 2015). Die grundlegende Funktionsweise soll aber anhand des Perzeptrons erklärt werden. Der Wert, den ein Neuron ausgibt, ist davon abhängig, ob die gewichtete Summe der Eingänge einen festgelegten Schwellenwert übersteigt. Als Eigenschaft eines Perzeptrons kann anstatt eines Schwellenwertes, welcher überschritten werden muss, ein Bias festgelegt werden. Dieser Bias indiziert, wie leicht eine 1 am Ausgang des Perzeptrons erzeugt werden kann. Je größer der Bias, desto leichter wird eine 1 ausgegeben, ist der Bias hingegen stark negativ, ist es relativ schwer für ein Perzeptron eine 1 am Ausgang zu erzeugen (Nielsen, 2015). Abbildung 4 lässt vermuten, dass manche Perzeptronen mehrere Ausgänge haben, faktisch ist es aber noch immer ein Ausgang (Abbildung 3), es wird lediglich angezeigt, dass ein Ausgang eines Perzeptrons als Eingang für mehrere andere Perzeptronen

verwendet werden kann. Jedes Neuron ist somit mit allen Neuronen der vorherigen Schicht verknüpft, wobei eine Schicht aus einer definierten Anzahl von Neuronen besteht.

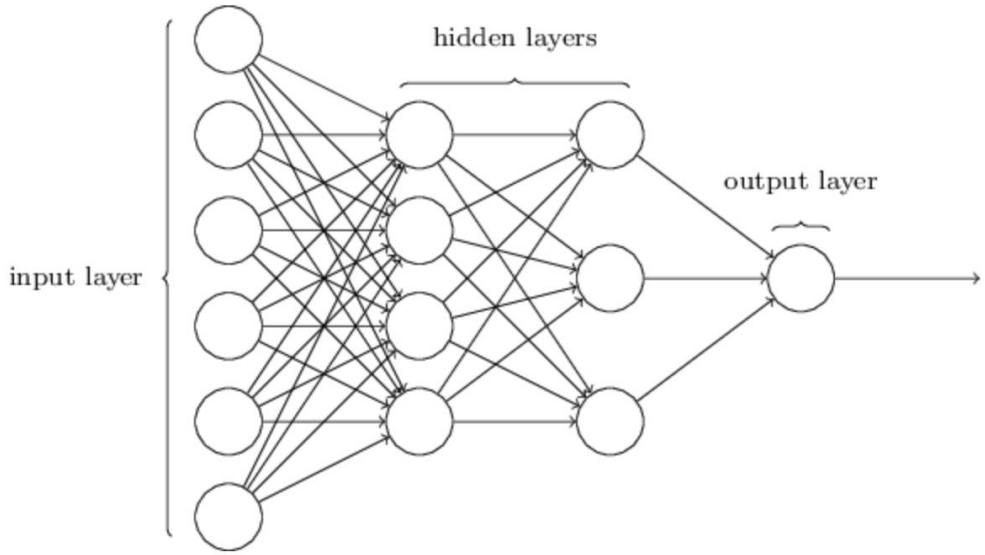


Abbildung 4: Beispielnetzwerk mit zwei versteckten Schichten (hidden layer). Links befindet sich die sichtbare Schicht mit den vorgegebenen Eingängen (input layer), rechts der sichtbare Ausgang (output layer) (Nielsen, 2015).

Es muss aber nicht zwangsläufig nur 0 oder nur 1 am Ausgang als Aktivierungswert ausgegeben werden. Dieser kann zwischen einem definierten Intervall wie [0,1] liegen. Um die Summe der gewichteten Mittel in einem Intervall zwischen 0 und 1 darstellen zu können, ist eine Funktion nötig. Diese sogenannte Aktivierungsfunktion komprimiert die gewichtete Summe der Eingänge in eine reale Zahl zwischen 0 und 1. Neben der vorher erwähnten Schwellenwertfunktion, welche nur 0 oder 1 ausgeben kann, gibt es Funktionen wie die Sigmoid Funktion oder Rectifier (ReLU) (Nielsen, 2015), welche einen beliebigen Wert oder Aktivierung zwischen 0 und 1 liefern können. Die gewählte Aktivierungsfunktion beeinflusst somit die Eigenschaften eines Neurons beziehungsweise eines Netzwerkes. Bei der Sigmoid Funktion (Abbildung 5) werden aufgrund der mathematischen Eigenschaften der Funktion stark negative Eingänge in Ausgänge nahe 0 umgewandelt und stark positive Eingänge in Ausgänge nahe 1 umgewandelt.

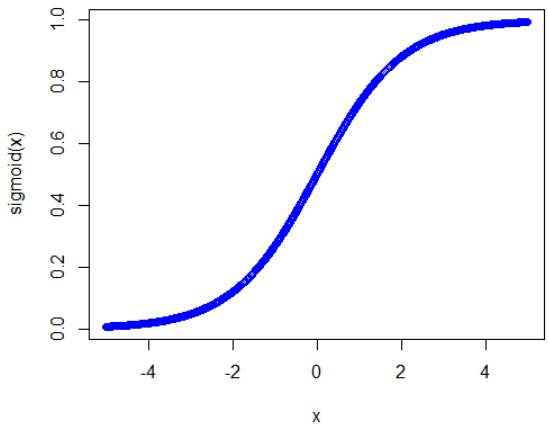


Abbildung 5: Eine Sigmoide Funktion als Aktivierungsfunktion. Die x-Achse stellt die Eingänge dar, die y-Achse den Ausgang des Neurons.

Die Gewichte einzelner Neuronenverbindungen zur vorgelagerten und nachgelagerten Schicht sowie zugehörige Bias und Aktivierungsfunktionen können als Parameter eines Netzwerkes gesehen und kalibriert werden.

Das typische Beispiel für ein DL-Modell ist ein vorwärtsgekoppeltes tiefes Netzwerk (*feedforward deep network*), oder anders ausgedrückt ein mehrschichtiges Perzeptron (*multilayer perceptron* MLP). Ein MLP kann einfach als mathematische Funktion aufgefasst werden, die bestimmte Eingänge in einem Netzwerk in Ausgangswerte wandelt, wobei diese mathematische Funktion aus vielen einfacheren Funktionen besteht. Hier spiegelt sich die Fähigkeit von DL wider, komplexe Vorgänge auf einfache herunterzubrechen, wobei jede Schicht eine einfachere Teilabbildung des Problems als die vorhergehende darstellt. Anders ausgedrückt, es geschieht eine Auflösung komplexer Vorgänge aus der realen Welt durch verschachtelte einfache Konzepte, die in entsprechender Beziehung zueinander stehen (Goodfellow *et al*., 2016). Abbildung 4 zeigt ein vereinfachtes Modell aus zwei sichtbaren Schichten (*input layer, output layer*) und *n* verdeckten (*hidden layer*) Schichten. Sichtbare Schichten deshalb, weil sie beobachtbare Variablen enthalten. Die Bezeichnung „verdeckte Schichten“ erklärt sich daraus, dass die Werte in diesen Schichten nicht in den Eingangsdaten enthalten sind, sondern das Modell selbst Konzepte entwickeln muss, die zur Erklärung der Beziehungen zwischen den Eingangsdaten sinnvoll sind. Diese versteckten Schichten sind in den Rohdaten nicht gegeben, sondern werden eigenständig vom Modell bestimmt inklusive der Beziehungen zueinander (Goodfellow *et al*., 2016).

Damit ein DNN lernen kann, muss es mit Trainingsdaten trainiert werden. Dies setzt einen Lernprozess voraus und verlangt nach einem Algorithmus, welcher die Abweichung zwischen dem erwarteten und dem errechneten Ergebnis minimiert. Eine sogenannte Kostenfunktion

hilft die passenden Gewichte und Bias zu finden, um die Abweichung zwischen dem Ergebnis (der Vorhersage) und dem erwarteten Wert am Ausgang des DNN zu minimieren. Diese Abweichung wird auch Kostenwert genannt und tendiert gegen 0, wenn Vorhersagen und erwartete Werte des DNN über alle Trainingsbeispiele stark miteinander übereinstimmen und gegen 1, wenn die Übereinstimmung gering ist. Der Algorithmus hat also gute Arbeit geleistet, wenn entsprechend gute Gewichte und Bias gefunden wurden und der Kostenwert sich 0 annähern. Das Ziel des Algorithmus ist, die Kosten eines Netzwerkes durch Anpassung von Bias und Gewichten zu minimieren. Dieser Vorgang trägt den Namen *gradient descent*. Das Auffinden des globalen Minimums der Kostenfunktion wäre das Optimum, funktioniert aber nicht nimmer. Eine Art von Kostenfunktion ist die mittlere quadratische Abweichung. Es werden die Quadrate der Abweichungen der berechneten und der gewünschten Aktivierungen der Neuronen aufsummiert (Nielsen, 2015). *Categorical cross entropy* ist eine weitere Kostenfunktion und wurde für diese Arbeit verwendet, um in den DNNs schnellere Lernkurven zu erreichen.

Startet der Trainingsprozess, werden zunächst die Trainingsdaten in kleinere Trainingseinheiten, auch *batches* genannt, aufgeteilt. Jeder *batch* beinhaltet eines, viele oder alle Trainingsbeispiele (=Zeilen) aus dem Trainingsdatensatz. Die Größe eines *batches* kann also variieren und wird vom Anwender festgelegt (Goodfellow *et al*., 2016). Um den Durchschnittswert der Kosten über alle Trainings zu berechnen (dieser gibt an wie schlecht oder gut das DNN funktioniert), wird der Kostenwert separat für jeden einzelnen *batch* berechnet. Mit dieser Information kann das DNN für jeden einzelnen Trainingsdurchlauf die Gewichte und Bias durch Zurückrechnen im Netzwerk selbst anpassen und die Kosten, abhängig von der proportionalen Wichtigkeit einzelner Gewichte und Bias, schrittweise minimieren. Dieser Prozess wird *backpropagation* genannt und ist wesentlich für den Lerneffekt beim Training eines DNN verantwortlich. Der negative Gradient der Kostenfunktion gibt dabei an, wie die Kosten durch gezielte Änderung der proportional unterschiedlichen Gewichte und Bias aller Neuronen und Neuronenverbindungen in einem Netzwerk schnellstmöglich minimiert werden können. Dies ist ein iterativer Prozess und wird so lange fortgesetzt, bis sich der Fehler an das globale Minimum der Kostenfunktion maximal annähert. Lernen im Kontext von DL bedeutet also, Bias und Gewichte so zu wählen, dass eine bestimmte Kostenfunktion minimiert wird (Nielsen, 2015).

Um den gesamten Trainingsprozess weiter zu beschleunigen, wird das Prinzip von *mini-batch gradient descent* angewendet. Die Trainingsdaten werden randomisiert und in kleine Einheiten, sogenannte *mini-batches* aufgeteilt. Eine *mini-batch* enthält per Definition mehr als ein Trainingsbeispiel, aber weniger Beispiele als der gesamte Trainingsdatensatz enthält. Von jeder dieser Einheiten wird mit Hilfe von *backpropagation* der negative Gradient berechnet. Die *mini-batches* stellen nur einen kleinen Teil der Trainingsdaten dar und daher erfolgt eine schnellere Approximation an das Minimum der Kostenfunktion (Nielsen, 2015).

Eine Hauptherausforderung bei ML ist, dass der Algorithmus die Fähigkeit besitzen muss, mit neuen unbekannten, noch nicht beobachteten Daten, gut umgehen zu können – dies wird als Generalisierung bezeichnet (Goodfellow *et al*., 2016). Mit den zur Verfügung stehenden Trainingsdaten kann man den sogenannten Trainingsfehler (siehe Kapitel 4.2.2) berechnen, welchen es zu reduzieren gilt. Damit es bei ML aber auch zu einer Optimierung kommt, wird auch ein geringer Generalisierungsfehler oder auch Testfehler angestrebt. Der Testfehler ist also der Erwartungswert für den Fehler neuer Eingangsdaten für das Netzwerk. Durch die Ermittlung der Leistung eines Netzwerkes für eine neue Testdatenmenge oder Validierungsdatenmenge kann der Generalisierungsfehler geschätzt werden (Goodfellow *et al.*, 2016). Ist der Unterschied zwischen Trainingsfehler und Testfehler zu groß, kommt es zu einer Überanpassung des ML-Modells, d.h. das Modell hat sich sehr gut an die Trainingsdaten angepasst, weist aber eine geringere Leistung gegenüber neuen Eingangsdaten auf. Über die Anzahl der Epochen kann definiert werden, wie oft sich der ML-Algorithmus durch das Trainingsdatenset arbeitet oder iteriert (Goodfellow *et al*., 2016). Eine Epoche ist ein vollständiger Durchlauf durch alle Beispiele im verwendeten Trainingsdatensatz.

Deep learning wird bereits erfolgreich in Sprach-, Text- und Audioverarbeitung, Computervision (Maschinen wird Sehen beigebracht), Bioinformatik und Chemie, Video Spiele, Suchmaschinen, Finanzwesen, Online-Werbung und Robotik eingesetzt (Lecun *et al*., 2015; Nielsen, 2015; Goodfellow *et al*., 2016; Wick, 2017). Durch verbesserte Trainingsalgorithmen und höhere Rechenleistung wurden tiefere und somit leistungsstärkere Netzwerke erst in den vergangenen Jahren möglich (Goodfellow *et al*., 2016).

3.1.2 iLand – the Individual-based forest LANdscape and Disturbance model

Das Individuen-basierte Modell für Waldlandschaften und Störungen iLand (Seidl *et al*., 2012a) wurde für die Simulation von dynamischen Entwicklungen von Waldökosystemen unter sich verändernden Klimabedingungen und Störungsregimen entwickelt. Besonderes Augenmerk wurde auf die Interaktionen und Feedbacks zwischen Klima, Management und Störungsregimen gelegt. iLand arbeitet als Modell auf Landschaftsmaßstab und integriert Populations-Prozesse (z.B. Wachstum, Sterblichkeit, Regeneration und Verteilung von Individuen) und Ökosystem-Prozesse (z.B. Kohlenstoff-, Wasser- und Stickstoffflüsse) (Seidl *et al*., 2012a).

Die kleinste Einheit des Modells sind Baumindividuen. Die Konkurrenz zwischen den Baumindividuen um Licht basiert auf der *ecological field theory*. Dabei wird über die gesamte Landschaft Lichtverfügbarkeit auf einem 2x2 m Raster berechnet. Über die Position des Baumes in der Landschaft und seine physischen Eigenschaften (Höhe, Kronenform, Lichtundurchlässigkeit) kann die relative Konkurrenz zu anderen Individuen errechnet werden.

Auf Bestandesebene wird die *absorbed photosynthetically active radiation* (APAR) in einer Auflösung von 100x100 m berechnet, und die effektive *light use efficiency* (LUE) einer Baumart berechnen. Die Bruttoprimärproduktion (GPP) verhält sich linear zum Anteil der verwendeten APAR und der effektiven LUE, woraus sich durch den Ansatz von Landsberg und Waring (1997) wiederum die Nettoprimärproduktion (NPP) ableiten lässt.

iLand simuliert kurzgesagt die Konkurrenz um Ressourcen auf Basis einzelner Bäume und die generelle Ressourcenverfügbarkeit von Strahlung, Wasser und Nährstoffen auf Bestandesebene (100x100 m Raster). Temperatur, Bodenwasserverfügbarkeit und Feuchtigkeit (auf täglicher Basis) sowie Nährstoffverfügbarkeit und die atmosphärische CO_2-Konzentration (auf monatlicher Basis) werden berücksichtigt (Seidl *et al*., 2012a).

Photosyntheseprodukte werden einmal im Jahr zuerst in Wurzeln und Blattmasse und dann im Stamm und anderen Reservespeichern allokiert. Fein- und Grobwurzelmasse wird bei der Allokation unterschieden, wobei auf ein ausgeglichenes Spross-Wurzelverhältnis geachtet wird. Die Allokation von oberirdischer Biomasse folgt dem Ansatz von Duursma *et al.* (2007). Anhängig von gegebenen Umweltverhältnissen passt jedes Individuum sein Allokationsverhalten an (z.B. steckt bei kargen Umweltbedingungen mehr Energie in die

Ausbildung unterirdischer Biomasse). Bei abnehmenden Lichtverhältnissen wird Höhenwachstum Dickenwachstum vorgezogen, um den kompetitiven Status zu verbessern. Die Entwicklung des H/D-Verhältnisses wird sozusagen der Konkurrenz um Licht angepasst (Seidl *et al*., 2010).

Natürliche Baumsterblichkeit (nicht durch den Menschen oder Störungen verursacht) wird durch das maximale Alter oder durch Stress determiniert. Mit steigendem Alter steigt die Wahrscheinlichkeit eines natürlichen Todes, abhängig von der jeweiligen Baumart. Stress tritt für einen Baum dann auf, wenn die minimalen Anforderungen an Kohlenstoff nicht durch die NPP oder aus Reserven gedeckt werden kann (Gűneralp und Gertner, 2006).

Ein Boden-Modul ermöglicht die Simulation von geschlossenen Kohlenstoff- und Stickstoffkreisläufen. Dazu werden stehendes und liegendes Totholz, Streu sowie die organischen Anteile im Boden berechnet (Seidl *et al*., 2012b).

Das Regenerations-Modul modelliert Samenverbreitung und Verjüngung. Wachstum und Konkurrenz der Verjüngung wird in einer 2x2 m Auflösung in Bezugnahme auf Ressourcenverfügbarkeit sowie limitierende Faktoren (z.B. Klima) für die jeweilige Baumart simuliert. Sobald die Pflanzen eine Höhe von 4m erreicht haben, wird jede Pflanze als Individuum behandelt (Seidl *et al*., 2012b).

Weitere Module für Wind (Seidl *et al*., 2014a), Borkenkäfer (Seidl und Rammer, 2017), Feuer (Seidl *et al*., 2014b) und das für diese Arbeit verwendete Modul für menschliche Eingriffe, genannt ABE (Rammer und Seidl, 2015), wurden für iLand entwickelt und erfolgreich implementiert sowie getestet.

Für die Simulation wurde die iLand Version 1.08 MSVC 64-bit Qt 5.8.0 verwendet (build date: Feb 15 2019).

3.1.3 Testlandschaft

iLand (Seidl *et al*., 2012a) in Kombination mit dem Modul ABE (Rammer und Seidl, 2015) wurde genutzt, um die Datengrundlage für ein DNN zu generieren. Diese Datengrundlage wurde entsprechend aufbereitet und dient dazu, das DNN zu trainieren. Dazu wurde die Waldentwicklung auf einer Testlandschaft (siehe Abbildung 6) mit reiner Fichtenbestockung über einen Zeitraum von 1000 Jahren simuliert. Die Outputs der Simulationen teilen sich in eine SQLLite-Datenbank und in ein speziell erzeugtes CSV-Logfile (Tabelle 4).

Testlandschaft nach Rammer and Seidl (2015):

- 3x4 km (12 km² oder 1200 ha)
- Fichte rein
- jeder Bestand betreffend seines Vegetationszustandes zufällig initialisiert (Normalwald), siehe Tabelle 1
- Geländemorphologie homogen
- keine externen biotischen oder abiotischen Einflüsse (Borkenkäfer, Wind, Feuer, …)
- Klima- und Bodendaten vorhanden

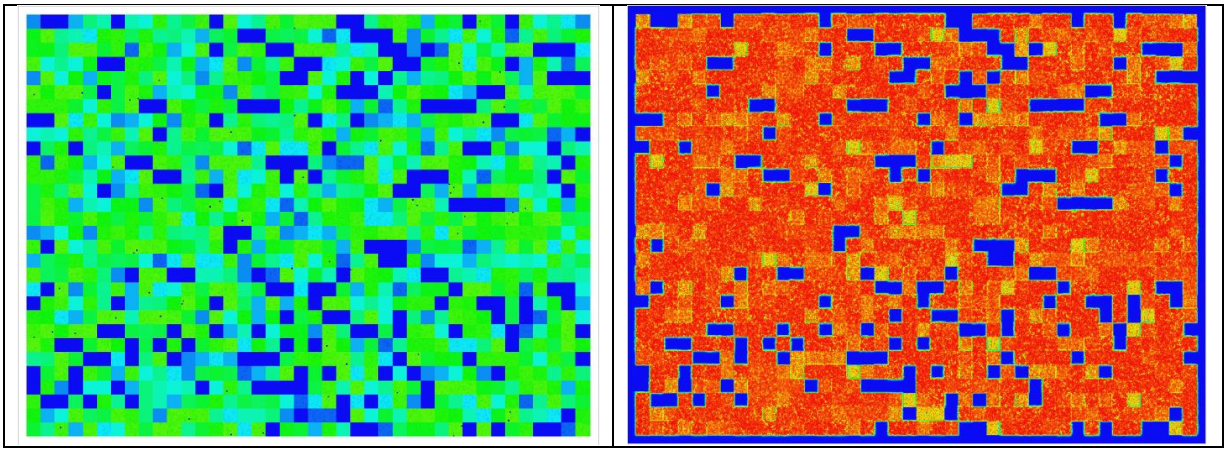


Abbildung 6: Hier wird die Oberhöhe in 10 m Auflösung „Dominance grid" (links) und die Konkurrenz um Licht „Light influence field" (rechts) von einer zufällig initialisierten Testlandschaft in iLand dargestellt.

Tabelle 1: Initialisierungsdatei der Testlandschaft. Beginnend von rechts wird jeder Zelle eine Baumart, die Stammzahl, die BHD-Spreitung, das H/D-Verhältnis und das Alter zugeordnet. Die Initialisierung erfolgte randomisiert.

	stand_id	species	count	dbh_from	dbh_to	hd	age
1	1	piab	1136	19.50	24.50	96	71
2	2	piab	1798	13.10	18.10	99	50
3	3	piab	705	27.20	32.20	87	97
4	4	piab	3300	7.12	10.68	88	30
5	5	piab	646	28.80	33.80	85	102
6	6	piab	2773	8.48	12.72	92	37
7	7	piab	2370	9.84	14.76	95	39
8	8	piab	705	27.20	32.20	87	95
9	9	piab	1264	17.90	22.90	98	65
10	10	piab	546	31.80	36.80	80	110
11	11	piab	646	28.80	33.80	85	98
12	12	piab	1798	13.10	18.10	99	52
13	13	piab	1136	19.50	24.50	96	70
14	14	piab	846	24.20	29.20	91	83

3.1.4 ABE – the Agent Based management Engine for iLand

Um verschiedene Bewirtschaftungsvarianten auf derselben Testlandschaft über definierte Zeiträume simulieren zu können, wurde das Agenten-basierte Submodell ABE eingesetzt.

Das Bewirtschaftungs-Submodell ABE ist in iLand (Seidl *et al*., 2012a) implementiert. Es bietet die Möglichkeit, dynamische Waldbewirtschaftung durch individuelle WaldbesitzerInnen unter sich verändernden Klimabedingungen zu simulieren (Rammer und Seidl, 2015). Dabei werden WaldbesitzerInnen durch Agenten abgebildet, welche einen bestimmten Teil der Landschaft bewirtschaften. Diese WaldbesitzerInnen unterscheiden sich, wie in der Realität, durch deren Verhalten, welches sich im Know-How sowie in den technischen Möglichkeiten zur Waldbewirtschaftung wiederfindet. Diese Agenten werden verschiedenen Agenten-Typen zugeordnet. Agenten-Typen spiegeln typische WaldbesitzerInnen Archetypen wider, wie etwa ein bäuerlicher WaldbesitzerInnen oder einen GroßwaldbesitzerInnen, während Agenten individuelle WaldbesitzerInnen sind (Seidl und Rammer, 2015).

Der Großteil der Eigenschaften und Fähigkeiten dieser Agenten wird über *stand treatment programs* (STPs) definiert. Ein STP ist eine Abfolge von forstlichen Eingriffen, die normalerweise die grundlegenden waldbaulichen Aspekte wie Verjüngung, Bestandespflege, Durchforstung und Ernte umfassen und regeln (Seidl und Rammer, 2015).

Diese forstlichen Eingriffe sind das Herzstück von ABE und bilden das forstliche Management ab, da diese Aktivitäten tatsächlich Bäume in dem simulierten Ökosystem manipulieren. Es gibt eine Bibliothek in ABE, die die wichtigsten forstlichen Eingriffe definiert und beinhaltet. Andere Simulationsszenarien verlangen oft nach adaptierten oder komplexeren Eingriffen. Es ist aber möglich, mit der ABE Javascript API (Seidl und Rammer, 2015) die Bibliothek in ABE selbstständig zu erweitern und individuelle Eingriffe zu programmieren.

3.1.5 Bewirtschaftungsvarianten

In ABE werden Bewirtschaftungsvarianten mithilfe der Programmiersprache JavaScript (JS) definiert. JS ist eine der am häufigsten genutzten Scriptsprache und auch in iLand/ABE verfügbar. Die gesamten nutzerspezifischen Einstellungen für ABE sowie die Einbindung in iLand wurden hiermit umgesetzt.

Im Fall dieser Arbeit wurde lediglich ein Agenten-Typ mit einem Agenten für die Simulation erstellt. Dieser Agent verfügt über sechs STPs. Die Nullvariante oder Referenz (es wird kein forstlicher Eingriff über den Simulationszeitraum durchgeführt) wurde mit deaktiviertem ABE-Modul in iLand realisiert. Für alle anderen Bewirtschaftungsvarianten (siehe Tabelle 2) wurde jeweils ein STP in JS erstellt (siehe Tabelle 3). Die Simulationen liefen sequentiell ab. Zuerst wurde die gesamte Testlandschaft mit einem bestimmten STP initialisiert, danach wurden vier Simulationen über einen Zeitraum von 1000 Jahren durchgeführt, um die Stochastizität der Simulationen abzubilden und als Trainingsdaten für das DNN aufbereitet. Dieser Vorgang wurde für jede der in Tabelle 2 gelisteten Bewirtschaftungsvarianten durchführt.

Tabelle 2: Liste der Bewirtschaftungsvarianten. Zu sehen ist die Strukturierung der Waldbewirtschaftung für ABE in iLand. Die detaillierte Umsetzung der einzelnen Bewirtschaftungsvarianten als STPs ist in Tabelle 3 zu sehen.

Betriebsart: Hochwald Fichte rein	Bewirtschaftungs-Varianten (Abkürzung)	Umtriebszeit [Jahre]	Eingriffe
Nullvariante	REF	-	-
Altersklassenwald			
Kahlschlag	KS100	100	Pflanzung, Dickungspflege 3 x Durchforstung, Kahlschlag
Kahlschlag	KS80	80	Pflanzung, Dickungspflege 2 x Durchforstung, Kahlschlag
Saumschlag	SS100	100	3 x Saumschlag
Saumschlag	SS80	80	3 x Saumschlag
Dauerwald		Bestandesvorrat [fm]	
Zielstärkenutzung	ZSN55	höher	Zielstärke BHD= 55cm; Eingriffe alle 7 – 8 Jahre; maximale Entnahme von 20% des Gesamtvorrats pro Eingriff
Zielstärkenutzung	ZSN45	niedriger	Zielstärke BHD= 45cm; Eingriffe alle 5 Jahre; maximale Entnahme von 20% des Gesamtvorrats pro Eingriff

Die Bewirtschaftungsvarianten wurden als STPs umgesetzt. Diese sind voll funktionstüchtig in iLand mit ABE implementiert worden.

Die waldbaulichen Eingriffe sind im Detail in Tabelle 3 beschrieben und orientieren sich teilweise an vorangegangen Arbeiten mit unterschiedlichen Simulationsmodellen wie iLand (Albrich, 2016) oder PICUS v1.4 (Seidl *et al.*, 2007):

Tabelle 3: Beschreibung der verwendeten Eingriffe je STP im iLand-Modul ABE. Die STPs sind die technische Umsetzung der Bewirtschaftungsvarianten KS100, KS80, SS100, SS80, ZSN55 und ZSN45. Jedes STP setzt sich aus einer Reihe definierter Eingriffe zusammen. Abhängig von der Durchmesserspreitung werden Bestände in iLand in fünf relative Durchmesserklassen unterteilt. Mit einem Prozentanteil pro Durchmesserklasse (kumuliert 100%) kann die Eingriffsstärke je Durchmesserklasse festgelegt werden. Die Unterteilung je relativer Klasse befindet sich in den eckigen Klammern [].

Kahlschlag mit 100 Jahren Umtriebszeit: KS100		
Pflanzung:	100% Fichte, Höhe 30cm, im ersten Jahr nach dem Abtrieb	
Dickungspflege:	zwischen 5-15 Jahren 30% Stammzahlentnahme Durchmesserklassen [100] (Albrich, 2016)	
Durchforstung 1:	zwischen 35-40 Jahren 30% Volumsentnahme Durchmesserklassen: [0, 15, 15, 45, 25] (Seidl *et al.*, 2007) BHD > 10cm	
Durchforstung 2:	zwischen 45-50 Jahren 30% Volumsentnahme Durchmesserklassen [0, 15, 15, 45, 25] (Seidl *et al.*, 2007) BHD > 10cm	
Durchforstung 3:	zwischen 55-60 Jahren 30% Volumsentnahme Durchmesserklassen [0, 15, 15, 45, 25] (Seidl *et al.*, 2007) BHD > 10cm	
Kahlschlag:	Abtrieb des Bestandes	
Kahlschlag mit 80 Jahren Umtriebszeit: KS80		
Pflanzung:	100% Fichte, Höhe 30cm, im ersten Jahr nach dem Abtrieb	
Dickungspflege:	zwischen 5-15 Jahren 30% Stammzahlentnahme Durchmesserklassen [100] (Albrich, 2016)	
Durchforstung 1:	zwischen 35-40 Jahren 30% Volumsentnahme Durchmesserklassen [0, 15, 15, 45, 25] (Seidl *et al.*, 2007) BHD > 10cm	
Durchforstung 2:	zwischen 45-50 Jahren 30% Volumsentnahme Durchmesserklassen [0, 15, 15, 45, 25](Seidl *et al.*, 2007) BHD > 10cm	
Kahlschlag:	Abtrieb des Bestandes	

Saumschlag mit 100 Jahren Umtriebszeit: SS100		
Saumschlag 1:	bei etwa 80% der Umtriebszeit (etwa 20 Jahre vor Abtrieb) ein Streifen von einer Baumlänge (33m) wird abgetrieben vor dem nächsten Saumschlag darf mindestens 10 Jahre kein Eingriff stattfinden	
Saumschlag 2:	bei etwa 90% der Umtriebszeit (etwa 10 Jahre vor Abtrieb) ein Streifen von einer Baumlänge (33m) wird abgetrieben vor dem nächsten Saumschlag darf mindestens 10 Jahre kein Eingriff stattfinden	
Saumschlag 3:	Abtrieb der übriggeblieben 33m Wald des Bestandes	
Saumschlag mit 80 Jahren Umtriebszeit: SS80		
Saumschlag 1:	bei etwa 75% der Umtriebszeit (etwa 20 Jahre vor Abtrieb) ein Streifen von einer Baumlänge (33m) wird abgetrieben vor dem nächsten Saumschlag darf mindestens 10 Jahre kein Eingriff stattfinden	
Saumschlag 2:	bei etwa 88% der Umtriebszeit (etwa 10 Jahre vor Abtrieb) ein Streifen von einer Baumlänge (33m) wird abgetrieben vor dem nächsten Saumschlag darf mindestens 10 Jahre kein Eingriff stattfinden	
Saumschlag 3:	Abtrieb der übriggeblieben 33m Wald des Bestandes	
Zielstärkenutzung: ZSN55 und ZSN45		
Zielstärkenutzung 55:	in einem Intervall von 8 Jahren nur Fichten ab BHD >= 55cm maximal 20% Entnahme des Bestandesvolumens pro Eingriff Entnahme im Bestand räumlich zufällig verteilt	
Zielstärkenutzung 45:	in einem Intervall von 5 Jahren nur Fichten ab BHD >= 45cm maximal 20% Entnahme des Bestandesvolumens pro Eingriff Entnahme im Bestand räumlich zufällig verteilt	

In Tabelle 3 wurde der Aufbau der STPs dargestellt. Jedes STP in ABE ist die Summe der einzelnen zugewiesenen Eingriffe. Die Bewirtschaftungsvariante KS80 entspricht dem gleichnamigen technischen STP KS80, welches sich aus Pflanzung, Dickungspflege, Durchforstung 1, Durchforstung 2 und einem Kahlschlag zusammensetzt. Aus Tabelle 3 wurden die Bewirtschaftungsvarianten KS100, KS80, SS100, SS80, ZSN55 und ZSN45 abgeleitet.

Kahlschlag KS100 & KS80:

Im Unterschied zu allen anderen Bewirtschaftungsvarianten wird hier eine Pflanzung im ersten Jahr nach dem Abtrieb durchgeführt und nicht mit Naturverjüngung gearbeitet. Wie bei Albrich (2016) wird auch hier eine Dickungspflege zwischen Bestandesalter 5 und 15 Jahren und danach drei (bei Umtriebszeit 100 Jahren) beziehungsweise zwei Durchforstungen (bei Umtriebszeit 80 Jahren) durchgeführt. iLand unterteilt jeden Bestand je nach Durchmesserspreitung in fünf Durchmesserklassen. Mit einem Prozentanteil pro Durchmesserklasse, welche in Summe 100% ergeben, kann die Eingriffsstärke je Durchmesserklasse festgelegt werden. Dabei wurde die Entnahme pro Eingriff mit maximal 30% des Gesamtbestandesvolumens festgelegt. Die einzelnen Durchforstungen wurden als Hochdurchforstungen (Reininger, 2000) konzipiert und orientieren sich an den Hochdurchforstungen aus Seidl *et al.* (2007).

Saumschlag SS100 & SS80:

Beide Bewirtschaftungsvarianten orientieren sich an Mayer (1992). Um eine gute Vergleichbarkeit zwischen den Kahl- und Saumschlagformen zu erhalten, wurden auch hier die Umtriebszeiten auf 100 und 80 Jahre festgelegt. Je nach festgelegter Umtriebszeit werden 20 und 10 Jahre vor dem endgültigen Abtrieb ca. eine Baumlänge (hier: 33m) pro Bestand abgetrieben und ergeben unter Berücksichtigung des endgültigen Abtriebs drei Saumschläge. Nach einem erfolgten Saumschlag darf die darauffolgenden 10 Jahre kein weiterer Eingriff stattfinden. Es soll mit Naturverjüngung gearbeitet werden weshalb keine Pflanzung stattfindet. Ebenso findet keine Dickungspflege oder Durchforstung statt.

Zielstärkenutzung ZSN55 & ZSN45:

Die Zielstärkenutzungen können als Überführung eines Altersklassenwald in einen Dauerwald verstanden werden und lehnen sich an die Zielstärkenutzung von Reininger (2000) an. Es werden lediglich zwei unterschiedliche Zeitintervalle und die Ziel-Brusthöhendurchmesser für die räumlich zufälligen Entnahmen festgelegt. Hinzu kommt die Limitierung, dass maximal 20% des Gesamtbestandvolumen pro Eingriff entnommen werden dürfen.

Der gesamte Code der detaillierten Eingriffe befindet sich im Kapitel 11.1.

3.1.6 SVD – Scaling Vegetation Dynamics

SVD ist ein Modell zur Simulation von Vegetationsdynamiken über große räumliche Einheiten. Die Kernkomponente von SVD bildet ein DNN, welches die Übergänge diskreter Vegetationszustände für jede simulierte Zelle (100x100 m) schätzt (Rammer und Seidl, 2019a). Die Vorhersagen bestehen aus Übergangswahrscheinlichkeiten zwischen Vegetationszuständen und der Zeit bis zur Änderung des Vegetationszustandes für jede einzelne Zelle. Das DNN kann mit Daten aus verschiedenen Quellen, wie remote sensing oder anderen Simulationsmodellen, wie in diesem Fall iLand, trainiert werden. Diese Trainingsdaten können aus beobachteten oder simulierten Vegetationszuständen in Kombination mit der Verweilzeit und Umgebungsdaten (Klima, Boden, …) bestehen (Rammer und Seidl, 2019c). Um den Rechenaufwand gering zu halten, bedient sich SVD verschiedener Ansätze. Erstens wird die Komplexität von Vegetation in diskrete Vegetationszustände klassifiziert und zweitens wird stark auf parallele Rechenprozesse gesetzt. Dabei spielen die Umweltbedingungen ebenso eine Rolle wie die nähere Umgebung (Nachbarschaft) einer jeden Einheit beziehungsweise Berechnungszelle. Wichtige ökologische Indikatoren für Kohlenstoffspeicher, Biodiversität oder andere Variablen wie Oberhöhe oder BHD sind über den Vegetationszustand einer Einheit mit einer Datenbank verknüpft und können somit quantifiziert werden. Diese Datenbank wird auch vegetation attribute database oder VAD genannt (siehe Tabelle 11) und enthält entsprechend dem Vegetationszustand alle relevanten Werte (Rammer und Seidl, 2019a).

Der Funktionsablauf von SVD kann grob in zwei parallel laufende Prozesse geteilt werden. Zum einen behält das Kernmodell die Simulationslandschaft im Auge, bereitet Eingangsdaten für das DNN auf und verarbeitet gleichzeitig die laufenden Ausgänge des DNNs. Zum anderen erhält das DNN die aktuellen Daten der Simulationslandschaft vom Kernmodell, stellt die Vorhersagen an und liefert die Zustandsänderungen zurück an das Kernmodell. Wann immer ein neues Simulationsjahr beginnt, evaluiert das Kernmodell für jede Zelle, ob eine Änderung im zehnjährigen Vorhersagehorizont ansteht oder nicht. Tritt dieser Fall ein, werden für die entsprechende Zelle alle nötigen Daten gesammelt und an das DNN geschickt. Ist eine gewisse Menge an Zellendaten erreicht, errechnet das DNN die Übergangswahrscheinlichkeiten für die *Zustandsänderungen* S* und die *Zeit bis zur nächsten Zustandsänderung* ΔR. Abhängig von den Vorhersagen des DNNs führt das Kernmodell die anstehenden Änderungen durch, wie den Vegetationszustand einer Zelle ändern oder den

bestehenden Vegetationszustand beibehalten und die Verweilzeit erhöhen. Die detaillierte Funktionsweise von SVD wird im Appendix von Rammer und Seidl (2019c) erläutert.

Die Verweilzeit gibt an, wie lange sich eine Zelle schon in einem bestimmten Vegetationszustand befindet. Der maximale Vorhersagehorizont für den Zeitpunkt einer Vegetationszustandsänderung einer Einheit beträgt zehn Jahre. Vegetationszustände sind diskrete Zustände auf einer definierten räumlichen Einheit (kleinste Einheit 1 Hektar) und werden durch ihre Struktur, Zusammensetzung und Funktionen in verschiedene Zustandsklassen eingeteilt. Die Struktur eines Bestandes wird am besten durch die vertikale Standraumnutzung der Vegetation repräsentiert und kann durch die Kronendachhöhe abgebildet werden. Die Zusammensetzung wird durch die am Häufigsten vorkommende Baumart (> als 66% Biomasse in der Einheit) oder durch eine Mischung von Baumarten (jede Art hat mindestens 20% der Biomasse in der Einheit) definiert. Die Funktion einer Einheit wird über den *leaf area index* (LAI) festgemacht und spiegelt somit Transpiration, Lichtabsorption und Primärproduktion wider. Der Einfluss durch Nachbarzellen auf eine Einheit wird in zwei Stufen geteilt. Die erste Stufe sind die direkt an eine 100x100 m Einheit angrenzenden acht Nachbarzellen. Die zweite Stufe enthält alle Einheiten in einem 300m Radius um die zentrale Einheit. Um Naturverjüngung und Randeffekte abbilden zu können, haben diese beiden Stufen unterschiedlichen Einfluss auf die jeweils zentrale Einheit. Neben Umweltbedingungen und räumlichen Gegebenheiten bilden Störungen die dritte Gruppe von Einflüssen auf Zustandsänderungen. Sprunghaften Änderungen können durch natürliche oder vom Menschen verursachte Störungen auftreten (Rammer und Seidl, 2019a).

3.2 TRAININGSDATEN GENERIERUNG

iLand in Kombination mit ABE diente bei dieser Arbeit vor allem als Werkzeug für die Generierung von rohen Trainingsdaten für die DNNs. Eine Hauptlimitierung bei DL ist die vorhandene Datenmenge für Trainings. Eine größere und diversere Datenmenge verspricht normalerweise auch eine bessere Leistung der DNNs. Um diesem Anspruch gerecht zu werden, wurde für alle Bewirtschaftungsvarianten, inklusive der nichtbewirtschafteten Variante, ein 1000-jähriger Zeitraum in iLand simuliert. Die insgesamt sieben Bewirtschaftungsvarianten wurden getrennt voneinander auf der zufällig initialisierten Normalwaldlandschaft simuliert. Für jede Variante wurde die Simulation vier Mal wiederholt. In R wurden anschließend die zusätzlichen Informationen von ABE und die Klima- und Bodendaten an die Trainingsdaten angefügt, um somit einen verwendbaren Datensatz zu erhalten. Dieser Vorgang stellte eine Datenmenge von 6 009 907 Trainingsbeispielen für das DNN-Training zur Verfügung. Diese Daten wurden am Ende der Manipulation randomisiert und aufgeteilt in 20% Evaluierungsdaten und 80% Trainingsdaten.

Die Trainingsrohdaten von iLand bildeten die Grundlage für das Training der DNNs und beeinflussten somit wesentlich die spätere Funktionsfähigkeit der DNNs. Die Plausibilität der technischen Implementierung der Waldbewirtschaftung wurde in Kapitel 4.1.1 geprüft. In Kapitel 4.1.2 der Ergebnisse wurde unter anderem die Fähigkeit von iLand getestet, sinnvoll auf die unterschiedlichen Bewirtschaftungsvarianten zu reagieren.

3.3 DATENAUFBEREITUNG

Für die Datenaufbereitung für das DNN sowie die Analysen wurde das R *Project for Statistical Computing Version 3.5.1* verwendet (R Core Team, 2018).

Mit Hilfe von R-Studio (R Core Team, 2018) wurden die Trainingsdaten für die DNNs mit einander verschnitten, neu berechnete Informationen hinzugefügt, Management-Schlüssel erstellt, Klima- und Bodendaten angefügt, randomisiert und schlussendlich in 80/20-Splittung Trainingsdaten und Evaluierungs- bzw. Testdaten für das DNN gewandelt. Bereits vorhandene Klima- und Bodendaten für die Simulationslandschaft wurden von Werner Rammer zur Verfügung gestellt und für einen 1000-jährigen Simulationszeitraum kopiert und iteriert. Mit Datenverschneidung ist die Kombination der Daten aus der SQLLite-Datenbank und dem produzierten Logfile von ABE gemeint. Die Daten aus iLand (SQLLite-Datenbank) wurden über die Ressource ID (RID), welche als eindeutiger Schlüssel fungiert, mit den Daten aus ABE verschnitten bzw. angefügt. Die zusätzlichen Daten aus ABE sind nun also die Eingriffsart (*mgmtactivity, siehe* Tabelle 4), wann innerhalb eines 10-Jahres Horizonts ein spezifischer Eingriff stattfindet, auf welcher Zelle dieses Management passiert und unter welchem waldbaulichen Regime die Zelle bewirtschaftet wird. Die Eingriffsart und die Bewirtschaftungsvariante wurden weiters in einen numerischen Zahlencode gewandelt, damit das DNN später damit arbeiten kann. Außerdem wurden extrem seltene Zustände aus den Trainingsdaten entfernt, welche 0.0001% der Daten ausmachen. Ohne diese seltenen Zustände ist es für das DNN wesentlich einfacher, passende Vorhersagen zu treffen.

Tabelle 4: Das Logfile, das von ABE durch ein JavaScript erstellt wurde. Für jede Zelle wurden alle Eingriffe, die im Simulationszeitraum durchgeführt wurden, protokolliert. Die Eingriffsart wird durch die Spalte mgmtactivity codiert: cc – Kahlschlag, pl – Pflanzung, th1 – Durchforstung 1, th2 – Durchforstung, …. Diese Codierung wird später in einen für das DNN verarbeitbaren numerischen Code gewandelt.

	mgmtactivity	year	standId
1526	cc	37	1
1547	pl	38	1
3527	th1	75	1
4163	th2	85	1
7001	cc	137	1
7031	pl	138	1
8926	th1	174	1
9482	th2	184	1
13033	cc	255	1
13054	pl	256	1

3.4 DNN TRAINING

Das Training der DNNs wurde auf einem Server des Instituts für Waldbau der Universität für Bodenkultur durchgeführt. Das System lief unter Linux mit der Distribution Ubuntu mit folgender Ausstattung:

CPU: AMD Ryzen Threadripper 1950X (16 physical cores)
RAM: 128GB
GPU: 2x Nvidia RTX 2070
Storage: 1x SSD (500GB), 1x HDD (12TB)
Ubuntu 18.04 LTS

Die Programmierung der Netzwerke erfolgte in der Programmiersprache Phyton 3.6 (Guido, 2019), eingebettet in der plattformübergreifenden *open-source* integrierten Entwicklungsumgebung Spyder 3.2.6 (Pierre, 2019). Um die Rechenzeiten möglichst gering zu halten, wurden die Rechenprozesse auf eine leistungsstarke Grafikkarte ausgelagert. Grafikkarten sind für sehr viele kleine parallele Rechenaufgaben ausgelegt und können somit optimal für diese Aufgabe verwendet werden.

3.4.1 Schnittstellen

Keras ist eine *high-level neural networks API (Application-Programming-Interface)* oder zu Deutsch eine Programmierschnittstelle für komplexe neuronale Netzwerke (Chollet *et al*., 2015). Keras ist kompatibel mit Python 2.7-3.6, weshalb das DNN auch in Python3 auf einem Linux Server der Universität für Bodenkultur in Wien programmiert wurde. Der Vorteil von Keras ist, dass CPUs als auch multi-GPUs für die aufwändigen Rechenprozesse verwendet werden können und dies zu einer wesentlichen Zeitersparnis führt. Grundsätzlich gibt es zwei APIs in Keras, die *sequential* API und die *functional* API. Die *functional* API von Keras wurde verwendet, um komplexere Netzwerke zu definieren, wie in diesem Fall multi-output Modelle (Chollet *et al*., 2015).

Keras fungiert als *front-end* und verwendet in diesem Projekt TensorFlow (Abadi *et al.*, 2015) von Google als *back-end,* um Modelle zu entwickeln. *Front-end* deswegen, weil es für den menschlichen Nutzer und nicht für eine Maschine entworfen wurde.

3.4.2 Entwicklung der DNNs

Als *back-end* diente *TensorFlow* (Abadi *et al.*, 2015), eine *open-source* Bibliothek um ML-Modelle zu entwickeln und zu trainieren. Airbnb, AIRBUS, Google, intel, Lenovo, PayPal, Qualcomm (TensorFlow, 2019) sind nur einige wenige der unzähligen Unternehmen, die *TensorFlow* heute nutzen und somit die unglaubliche Bandbreite von ML aufzeigen.

Als erster Schritt wurde in Python ein DNN mit Hilfe von *TensorFlow* über die Schnittstelle Keras programmiert. Eine der Hauptaufgaben, bevor ein Training überhaupt begonnen werden kann, ist die Daten in eine für das neuronale Netzwerk verarbeitbare Form zu bringen. Das bedeutet einerseits die Daten von R in Python in eine entsprechende Form zu bringen und weiters jene Parameter auszuwählen, welche die besten Ergebnisse liefern und für das Netzwerk am besten zu interpretieren sind. Die Inputs für ein DNN mussten so gewählt werden, dass sie qualitativ die besten Indikatoren dafür sind, zu welchem Zeitpunkt (maximal 10 Jahre) eine Zelle in einen anderen Vegetationszustand übergeht oder in dem aktuellen Zustand verbleibt.

Das Logfile, welches unter anderem ein Output eines JavaScripts im ABE Modul ist, beinhaltet die Management-Eingriffe, das Jahr des Eingriffs, die zugehörige Zellen-ID, sowie die entsprechende Bewirtschaftungsvariante. In R wurden die Management-Eingriffe dann in eindeutige numerische Codes transformiert. Das Jahr des Eingriffs wurde in einen Zeitpunkt (maximal 10 Jahre) umgerechnet, relativ bezogen auf das aktuelle Jahr. Somit konnte das DNN mit den Informationen trainiert werden, in wie viel Jahren ein spezifisches Einzelmanagement auf einer bestimmten Zelle vollzogen wird. Eine Übersicht der Inputs und Outputs befindet sich in Tabelle 6.

Als zweiter Schritt wurden aus dem anfänglichen DNN drei verschiedene DNNs mit unterschiedlichem Informationsinput differenziert wie in Kapitel 3.4.3-3.4.5 zu sehen ist. Als dritter Schritt wurden alle drei DNNs einzeln trainiert.

3.4.3 Netzwerk DNN1: Einzelmanagements

Bei DNN1 werden der Zeitpunkt des Bewirtschaftungseingriffs und die Art des Bewirtschaftungseingriff verwendet. Sieht ABE in iLand beispielsweise eine Pflanzung in drei Jahren vom aktuellen Simulationsjahr vor, erhält das Netzwerk den Code für einen Pflanzungseingriff und die Zeitdifferenz von drei Jahren als Information.

3.4.4 Netzwerk DNN2: Bewirtschaftungsvarianten

Bei DNN2 erhält das Netzwerk nur die Information, welche Bewirtschaftungsvariante auf welcher Zelle gerade aktiv ist. Ein eindeutiger Code beschreibt etwa eine Zielstärkenutzung mit einem Zieldurchmesser von 55cm, ein Zeitsignal wann der Eingriff stattfinden wird, fehlt aber.

3.4.5 Netzwerk DNN3: Kombination

DNN3 erhält alle verfügbaren Bewirtschaftungsinformationen zu jeder Zelle, heißt Zeitpunkt und Art des einzelnen Bewirtschaftungseingriffs plus die Bewirtschaftungsvariante. Je mehr Informationen einem Netzwerk zum Training zur Verfügung stehen, umso besser kann der Lernalgorithmus komplexe Beziehungen abstrahieren, vereinfachen und lernen. Zu beachten ist, dass zwischen DNN1 und DNN2 keine direkte Verbindung zwischen den sich unterscheidenden Eingangsinformationen besteht, um Forschungsfrage (ii) prüfen zu können. In der Information, welche Bewirtschaftungsvariante auf welcher Zelle aktiv ist, darf also kein Bezug zu Art oder Zeitpunkt eines einzelnen Bewirtschaftungseingriffes hergestellt werden können. Diese Beziehung oder besser gesagt, das resultierende Ergebnis einer Zustandsänderung der Zelle zu einem bestimmten Zeitpunkt, soll das Netzwerk autonom lernen.

3.5 DNN EVALUATION

Die Trainierten DNNs können auf unterschiedliche Arten evaluiert werden. Die Aufteilung der gesamten Daten auf 80% Trainingsdaten und 20% Testdaten erlaubt eine Evaluierung der DNN Varianten pro Trainingsepoche hinsichtlich ihrer Vorhersagegenauigkeiten. TensorBoard (TensorFlow, 2017) unterstützt diese Validierung grafisch und lässt Vergleich zwischen DNN Varianten zu. Zusätzlich zu den Validierungen durch TensorBoard können in Python die Vorhersagewahrscheinlichkeiten für die abhängigen Variablen *nächster Vegetationszustand S** und *Verweilzeit ΔR* der DNNs ausgelesen werden, welche in R statistisch durch Konfusionsmatrizen berechnet werden können. Vergleiche zwischen den Bewirtschaftungsvarianten je DNN können erstellt werden. Außerdem können Aussagen über Abweichungen der Schätzungen der abhängigen Variablen von den erwarteten Werten getroffen werden (siehe Kapitel 4.2.3).

3.6 SVD SIMULATION

Um der Frage nachgehen zu können, ob Waldbewirtschaftung mit Hilfe von DNNs in Computersimulationen abgebildet werden kann, müssen die trainierten DNNs zuerst in das Modell SVD eingebunden und dann Simulationen durchgeführt werden. Das Besondere an SVD ist seine Skalierbarkeit bei gleichzeitig verhältnismäßig niedrigem Rechenaufwand (Rammer und Seidl, 2019a). Verglichen zu einem prozessbasierten Modell (PBM) wie iLand, ist SVD um drei bis vier Größenordnungen schneller bei der Simulation von dynamischen Vegetationsveränderungen (Rammer und Seidl, 2019a). Ein solcher Ansatz ermöglicht hochauflösende Simulationen von Ökosystemdynamiken auf großen oder globalen Skalen. Simulationspaare mit gleichen Startbedingungen von iLand und SVD werden anschließend über gleiche Zeiträume simuliert und die Endergebnisse einander gegenübergestellt und verglichen.

3.6.1 Allgemeines Setup von SVD

Um ein DNN in SVD erfolgreich zu implementieren, sind einige Schritte erforderlich (Rammer, 2018):

- Konfiguration der Landschaft:
  Jeder Simulationslandschaft in SVD liegt ein Grid zugrunde, welches sich aus 100x100 m Zellen zusammensetzt. Jede Zelle hat neben einer eindeutigen Identifikationsnummer (ID) einen wie oben erwähnten Zustand und eine Verweildauer. Diesem Grid unterliegen nun Umweltdaten, wie Bodeneigenschaften und Klimadaten wie Temperatur und Niederschlag. Diese Umwelt- und Klimadaten sind in Regionen gegliedert. Somit kann jede Zelle einer eindeutigen Umwelt- und Klimaregion zugeordnet werden.
- Liste der Vegetationszustände:
  Das Grundkonzept von SVD ist, dass jede Zelle zu jedem Zeitpunkt einen eindeutigen Vegetationszustand aller vorkommenden Zustände besitzt. Tabelle 5 zeigt einen Ausschnitt aller in SVD auftretenden Vegetationszustände. Jeder ID ist also die Struktur, Zusammensetzung, Funktion und ein zusammengesetzter Name, der den Vegetationszustand beschreibt, zugeordnet. Dieser zusammengesetzte Name kann wie in Abbildung 7 aussehen.

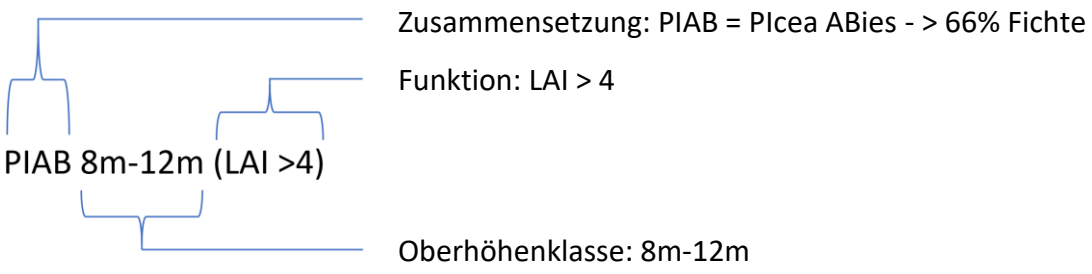


Abbildung 7: Beschreibung und Aufbau einer Beispieldefinition eines Vegetationszustandes in SVD

Tabelle 5: Ausschnitt aus der Liste der Vegetationszustände, die in der Simulation mit SVD vorkommen. Ganz links befindet sich die eindeutige Identifikationsnummer „stateId", dann der String „description". Dieser setzt sich wie in Abbildung 7 skizziert aus „composition", „structure" und „functioning" zusammen. Diese Informationen definieren einen einzigartigen Vegetationszustand.

stateId	description	composition	structure	functioning
1	PIAB 28m-32m (LAI >4)	PIAB	7	2
2	PIAB 24m-28m (LAI >4)	PIAB	6	2
3	PIAB 28m-32m (LAI 2-4)	PIAB	7	1
4	PIAB 20m-24m (LAI >4)	PIAB	5	2
5	PIAB Irr: 0m-12m (LAI <2)	PIAB	21	0
6	PIAB 32m-36m (LAI >4)	PIAB	8	2
7	PIAB 40m-44m (LAI >4)	PIAB	10	2
8	PIAB 36m-40m (LAI >4)	PIAB	9	2
9	PIAB 32m-36m (LAI 2-4)	PIAB	8	1
10	PIAB 0m-4m (LAI <2)	PIAB	0	0
11	PIAB 16m-20m (LAI >4)	PIAB	4	2
12	PIAB 36m-40m (LAI 2-4)	PIAB	9	1
13	PIAB Irr: 12m-24m (LAI <2)	PIAB	22	0

- Einbindung des DNN:

  SVD ist direkt mit *TensorFlow* gekoppelt und ist dadurch in der Lage, jedes beliebige trainierte DNN zu verwenden. Während die Art und Zahl von Eingängen in das DNN flexibel sind, erwartet SVD von jedem implementierten DNN zwei Ausgänge – eine Wahrscheinlichkeitsverteilung für den nächsten Zustand und eine zweite Wahrscheinlichkeitsverteilung für die Zeit bis zum Zustandswechsel. Unabhängig von der Architektur des einzubindenden DNNs müssen in SVD für jeden Informationseingang, die eingespeisten Daten und mögliche Transformationen jener Daten definiert werden.

- Module:

  Neben der Hauptfunktion von SVD Vegetationsübergänge mit Hilfe von DNNs zu simulieren, kann SVD mit Modulen erweitert werden. Dadurch können theoretisch nicht nur Waldflächen sondern Agrarflächen etc. simuliert werden. Prozesse wie Störungen, Waldbewirtschaftung oder andere rasche Änderungen in der Entwicklungsdynamik können mit verschiedensten Modulen implementiert werden.

Für mehr Information über den Aufbau, die Funktionsweise und Anwendung von SVD sei auf die Online-Dokumentation verwiesen: https://svdmodel.github.io/SVD/

Zu Beginn der Simulationen wird die Testlandschaft mit dem exakt selben Normalwald initialisiert, wie jener in iLand (siehe Kapitel 3.1.3), d.h. mit den exakten Vegetationszuständen pro Zelle. Dasselbe gilt für die Klima- und Bodendaten. Die Dauer der Simulation beträgt ebenfalls 1000 Jahre. Die Daten, welche SVD am Ende einer jeden Simulation liefert, bestehen aus dem jeweiligen Simulationsjahr mit der entsprechenden Verteilung der Vegetationszustände über die Landschaft. Es werden Simulationsvergleiche auf Landschaftsebene durchgeführt. Die Vegetationszustände wurden mit den zugehörigen forstlichen Parametern verknüpft und anschließend analysiert.

Die zufällige Initialisierung der Testlandschaft zu Beginn einer jeden Simulation soll die Ausgangssituation vor einem Waldumbau darstellen. Das heißt, dass sich die angewendeten Bewirtschaftungen erst einpendeln müssen und der Waldumbau auch noch ein Mehrfaches an Umtriebszeiten später in den Daten zu sehen ist. Um diesen Umstand zu umgehen, müsste für jede Simulation die entsprechende Bewirtschaftung bereits für hunderte von Jahren ausgeführt werden, um dann einen fließenden Einstieg in eine Bewirtschaftungsvariante zu erzeugen. Diese Tatsache lässt sich sehr schön an den Ganglinien für die Struktur ableiten, weshalb für Vergleiche die letzte Hälfte der Simulationen verwendet wird. Die Analysen exkludieren die Waldumbauphase und umfassen die letzten 500 Jahre der Simulation.

3.7 SVD EVALUATION

Die Ergebnisse werden nach Bewirtschaftungsvarianten gruppiert. Unter jeder Bewirtschaftungsvariante werden die drei in SVD implementierten DNN-Varianten mit den iLand Ergebnissen verglichen. Die Ergebnisse sind unterteilt in quantitative Ergebnisse (siehe Kapitel 4.3.1 und strukturelle Ergebnisse (siehe Kapitel 4.3.2). Unter quantitativen Ergebnissen werden alle forstlichen Parameter (Volumen, Stammzahl, Grundfläche, BHD, …) als Gegenüberstellung von den drei DNNs in SVD und den Ergebnissen aus iLand verstanden. Als strukturelle Ergebnisse sind die Strukturklassen, Oberhöhen oder LAI gemeint. Es wird die landschaftliche Entwicklung der zweiten Simulationshälfte analysiert. Zur besseren Vergleichbarkeit wurden dennoch unter beiden Punkten die Oberhöhe und der LAI gewählt. Da der Umfang der Ergebnisse sehr groß ist, befinden sich alle anderen quantitativen Vergleiche im Kapitel 10 Anhang.

Die Oberhöhe und der LAI dienen als Indikatoren für die Struktur und sollen besonders Forschungsfrage (iii), ob SVD strukturelle Unterschiede bedingt durch Bewirtschaftungsunterschiede darstellen kann, Rechnung tragen. Ein Vergleich einer jeden DNN-Variante mit den Ergebnissen aus iLand soll einen Eindruck der Unterschiede der beiden Ansätze geben. Aufgrund einer Klassifizierung der Vegetationszustände in reguläre und irreguläre Vegetationszustände (bezogen auf die Bestandesstruktur) ergibt sich auch eine unterschiedliche Codierung. Diese Codierung lässt sich unzureichend in einem Diagramm darstellen beziehungsweise müssten die irregulären Vegetationszustände exkludiert werden. Dieser Umstand kann aber mit einer Umrechnung der Strukturklasse in eine mittlere Oberhöhe umgangen und als Oberhöhenverlauf dargestellt werden (siehe Kapitel 4.3.2.1).

4 ERGEBNISSE

4.1 TRAININGSDATEN GENERIERUNG (WALDSIMULATION IN ILAND)

4.1.1 Ganglinien Bewirtschaftungsvarianten

Um die Funktion der mit JS implementierten Bewirtschaftungsvarianten nochmals mit waldbaulich gängigen Methoden zu vergleichen und zu kontrollieren, wurde eine künstliche Ausgangslage in iLand geschaffen. Die Testlandschaft wurde in folgenden initialen Zustand versetzt:

- Fichte rein
- 2500 Pflanzen je Hektar
- BHD: 2cm
- Alter: 5 Jahre
- 1,2m Höhe

Diese generische Ausgangssituation wurde allein aus Gründen der besseren Anschaulichkeit gewählt. Zu beachten ist, dass es sich bei diesen Ganglinien über den Simulationszeitraum auf die Gesamtentwicklung der Testlandschaft bezieht. Diese Vergleiche finden also auf Landschaftsebene statt. Um einen Bezug zu einer einzelnen Zelle der Simulationslandschaft herzustellen, wurde in Abbildung 8c-d und Abbildung 9c-d jeweils nur eine Zelle der Landschaft analysiert.

Um den implementierten JS Code auf Plausibilität prüfen zu können, wurden die Oberhöhenverläufe gesamtlandschaftlich und einzelner Zellen betrachtet. Es wurde angenommen, dass eine Veränderung der Oberhöhe ein guter Indikator für einen erfolgten Einzeleingriff ist. Die Oberhöhen werden aus den Oberhöhenklassen der Vegetationszustände abgeleitet. Weiters konnten die Eingriffe in iLand durch ABE mit waldbaulichen Methoden verglichen und auf Sinnhaftigkeit geprüft werden. Der wellenförmige Gang der Oberhöhen in der zweiten Hälfte des Simulationsraumes spiegelt die Einzelstammentnahmen in einem kurzen Zeitintervall wider.

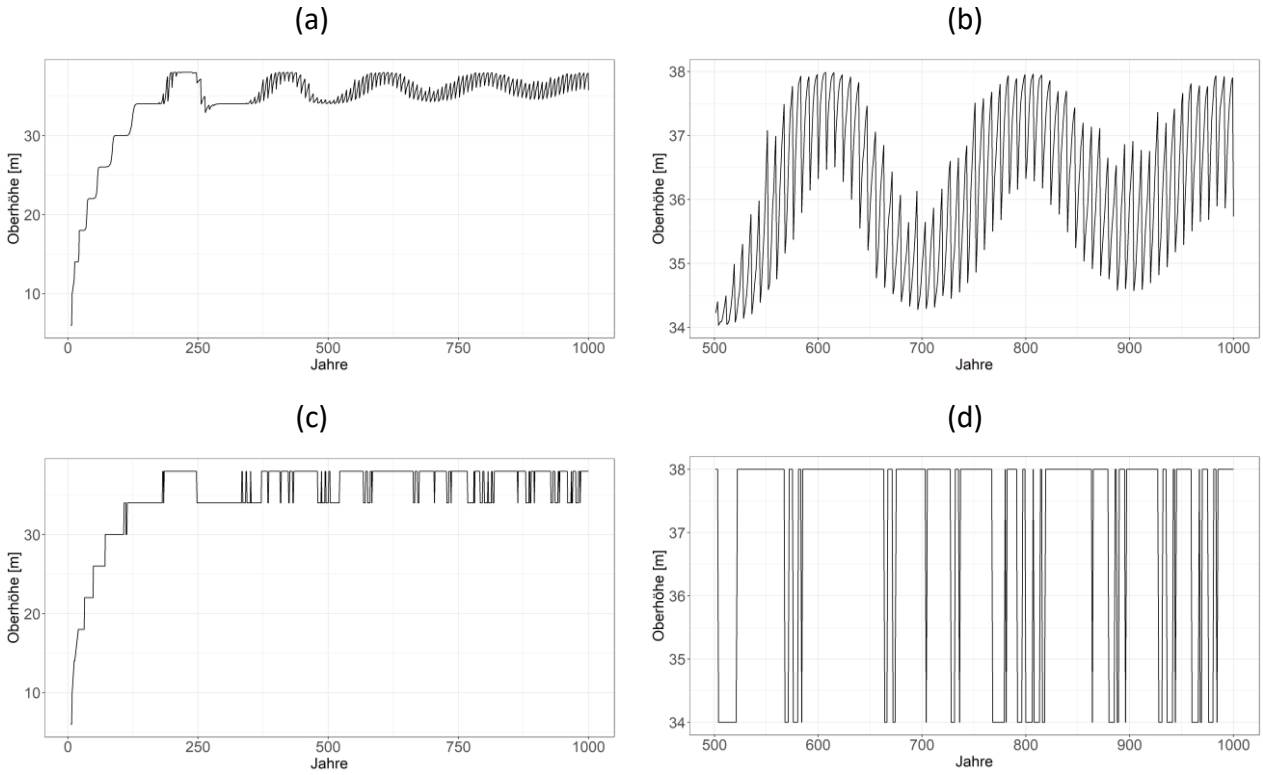


Abbildung 8: Ganglinien der Oberhöhenverläufe der Bewirtschaftungsvariante ZSN55 über 1000 Jahre (a und c) und die letzten 500 Jahre (b und d). Bei (a) und (b) hat sich nach den ersten 250 Jahren die Bewirtschaftungsvariante eingestellt und die Landschaft gerät in eine gleichmäßige Entwicklung. Bei (c) und (d) wurde zum Vergleich nur eine Zelle in der Landschaft ausgewählt und ihre Trajektorien für dieselben Zeiträume (1000 Jahre links und 500 Jahre rechts) dargestellt. Man kann die Einzelstammentnahmen sehr gut erkennen. Die Zielstärkenutzung mit BHD 55cm zeigt eine hohe Oberhöhenverteilung.

Bei der Bewirtschaftungsvariante KS100 ist vor allem die Trajektorie einer einzelnen Zelle interessant. Vor dem Abtrieb sind die einzelnen Eingriffe wie Dickungspflege und Durchforstung 1 – 3 gut zu erkennen Abbildung 9c-d und nachzuvollziehen.

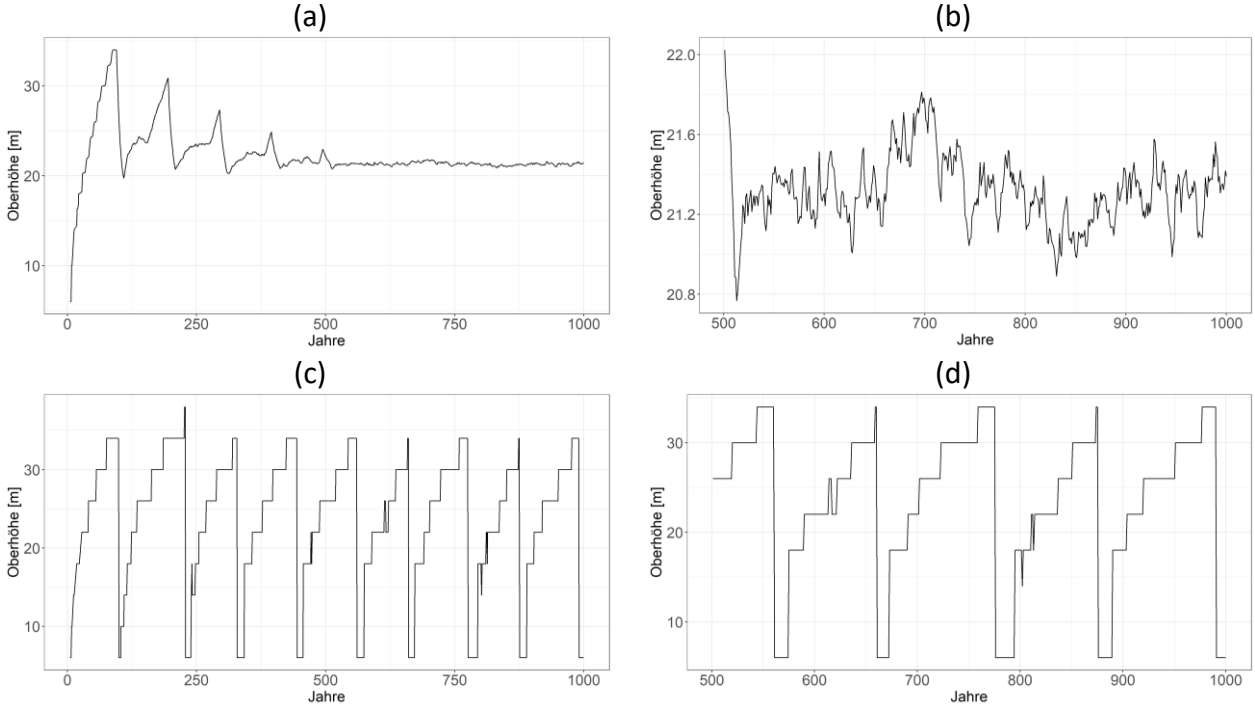


Abbildung 9: Ganglinien der Oberhöhenverläufe der Bewirtschaftungsvariante KS100 über 1000 Jahre (a und c) und 500 Jahre (b und d). Die ersten 500 Jahre hat sich die Bewirtschaftungsvariante eingestellt und die Landschaft gerät in eine gleichmäßige Entwicklung. Wie in Abbildung 8 wird in (c) und (d) zum Vergleich nur eine Zelle ausgewählt und ihre Trajektorie für dieselben Zeiträume (1000 Jahre (c) und 500 Jahre (d)) dargestellt. Die Einzeltrajektorie (c) und (d) zeigt die abrupten Entnahmen aus Dickungspflege und Durchforstungen inklusive Abtrieb nach 100 Jahren Umtriebszeit.

4.1.2 Ergebnisse auf Landschaftsebene

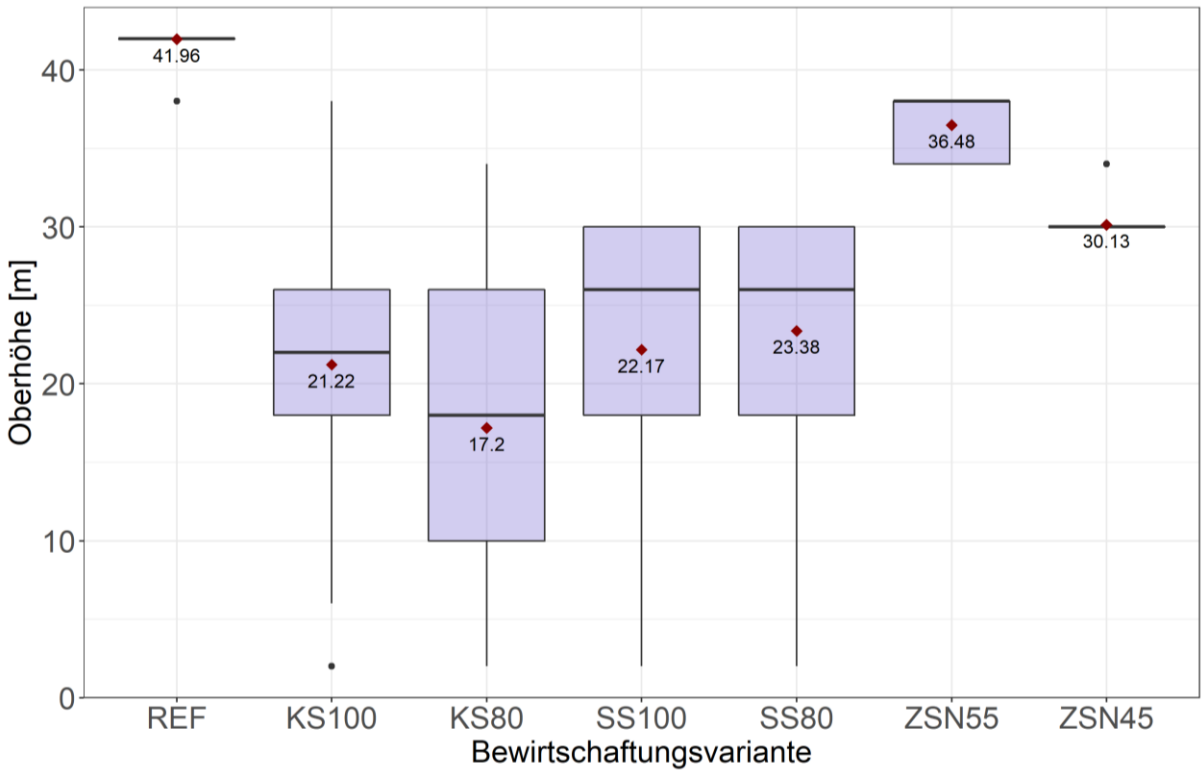


Abbildung 10: Dieses Boxplotdiagramm stellt die Oberhöhenverteilung in der Landschaft über die letzten 500 Jahre des Simulationszeitraumes dar. Die roten Punkte mit den zugehörigen Zahlen geben die Mittelwerte der Oberhöhen der Bewirtschaftungsvarianten an, abgeleitet aus den Oberhöhenklassen der Vegetationszustände.

Da die Simulationen immer mit der gleichen zufällig initialisierten Normalwaldlandschaft begannen, konnte dies mit einem Waldumbau in eine bestimmte Betriebsform gleichgesetzt werden. Um den Einfluss des Waldumbaus auszublenden, wurden deshalb nur die letzten 500 Jahre analysiert. Wenn man die Oberhöhenverteilung der Bewirtschaftungsvarianten in Abbildung 10 betrachtet, stellt man fest, dass die nicht bewirtschaftete Landschaft (REF) die höchste durchschnittliche Oberhöhe mit 42,0m aufweist. Die zweithöchste mittlere Oberhöhe mit 36,5m wird von der Zielstärkenutzung mit Zielstärke BHD 55cm (ZSN55) erreicht, gefolgt von Zielstärkenutzung mit BHD 45cm als Zielstärke (ZSN45) mit 30,1m. Bei den beiden Kahlschlagformen zeigt die verkürzte Umtriebszeit von (KS80) auch einen deutlichen Trend zu einer geringeren mittleren Oberhöhe mit 17,2m im Vergleich zu KS100 mit 21,2m. Die beiden Saumschläge unterscheiden sich minimal was ihre mittleren Oberhöhen betrifft. Beim Saumschlag mit 100 Jahren Umtriebszeit (SS100) beträgt die mittlere Oberhöhe 22,2m und beim Saumschlag mit 80 Jahren Umtriebszeit (SS80) 23,4m.

Deutlicher sind diese Unterschiede in dem Dichtediagramm Abbildung 11 zu sehen. Je dunkler das Feld einer Oberhöhenklasse einer bestimmten Bewirtschaftungsvariante ist, desto häufiger ist die jeweilige Oberhöhenklasse im Simulationszeitraum auf der Landschaft aufgetreten.

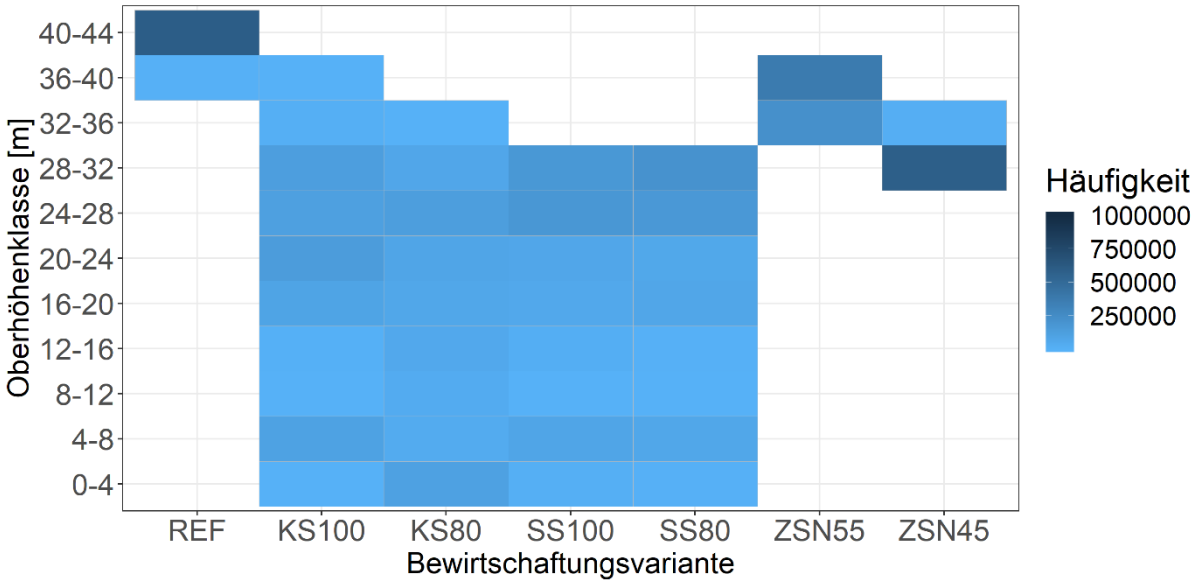


Abbildung 11: Das Dichtediagramm stellt die Oberhöhenverteilung in der Landschaft der letzten 500 Jahre dar. Die Oberhöhen sind auf der Y-Achse in die Oberhöhen in Klassen eingeteilt. Je dunkler ein Feld erscheint, desto häufiger kommt diese Höhenklasse für die betroffene Bewirtschaftungsvariante in der Landschaft vor. Weiße Felder bedeuten, dass diese Oberhöhenklasse nicht vorkommt, vgl. 1000 – jährigen Simulationszeitraum in Abbildung 12.

Um den Effekt des Waldumbaus noch einmal besser hervorheben zu können, wurde derselbe Diagrammtyp in Abbildung 12 für den gesamten Simulationszeitraum von 1000 Jahren angewendet. Durch den Übergang der Waldlandschaften in ein Entwicklungsequilibrium wurden viele Oberhöhenklassen relativ selten durchlaufen und einige wenige sehr oft (siehe REF, ZSN55 und ZSN45).

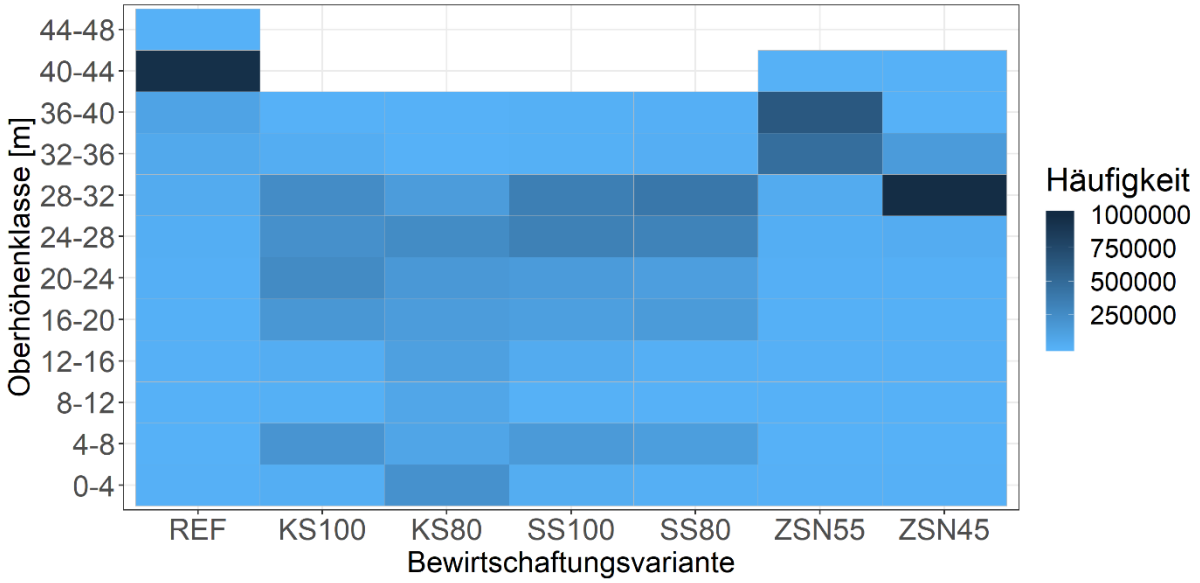
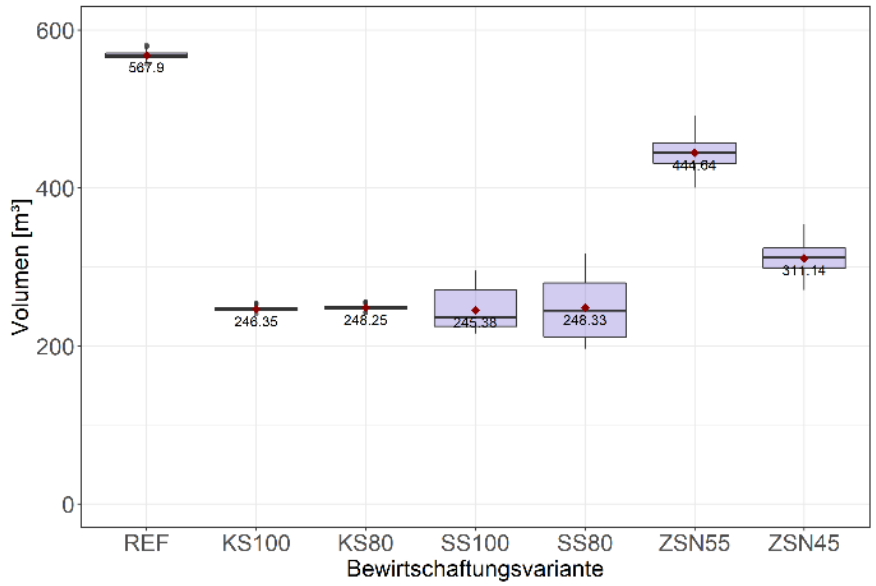


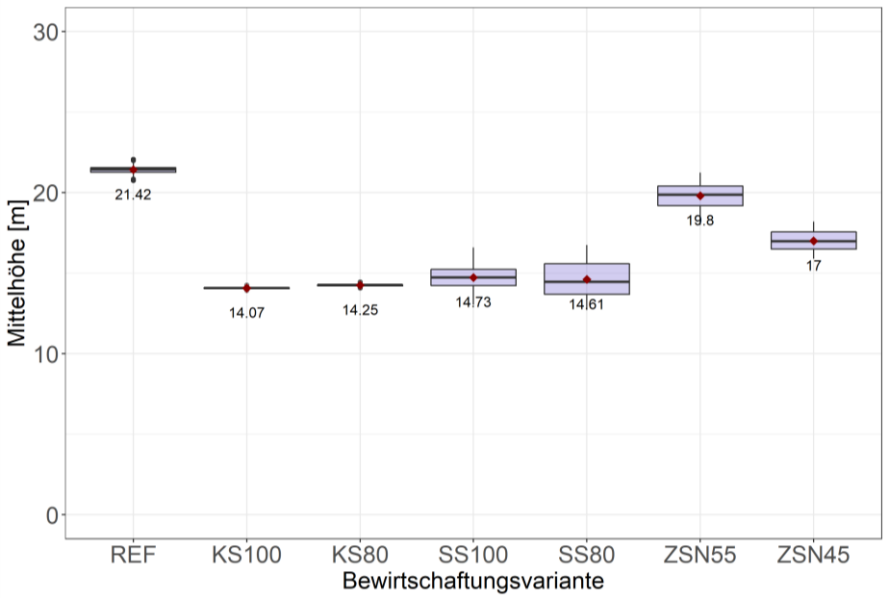
Abbildung 12: Das Dichtediagramm stellt die Oberhöhenverteilung über den Simulationszeitraum von 1000 Jahren in der Landschaft dar. Durch den Waldumbau kommen sehr viele Vegetationszustände vor, die in einem späteren Equilibrium sehr selten bzw. nicht mehr auftreten (vgl. Abbildung 11).

Gesamtlandschaftliche Ergebnisse konnten aus einer bestehenden iLand-Datenbank ausgelesen werden (Abbildung 13). Die jährlich vorhandenen Werte wurden ab dem 500. Jahr aufsummiert, gemittelt und dargestellt. Da die Oberhöhen in der Landschaftsdatenbank nicht mit einem Gesamtwert pro Jahr vorhanden sind, aber einen interessanten Vergleich zwischen den Bewirtschaftungsvarianten zulassen, wurden diese aus den Oberhöhenklassen der Vegetationszustände (VAD) abgeleitet (siehe Abbildung 10). Alle anderen forstlichen Variablen wurden in diesem Kapitel der Landschaftsdatenbank aus iLand entnommen.

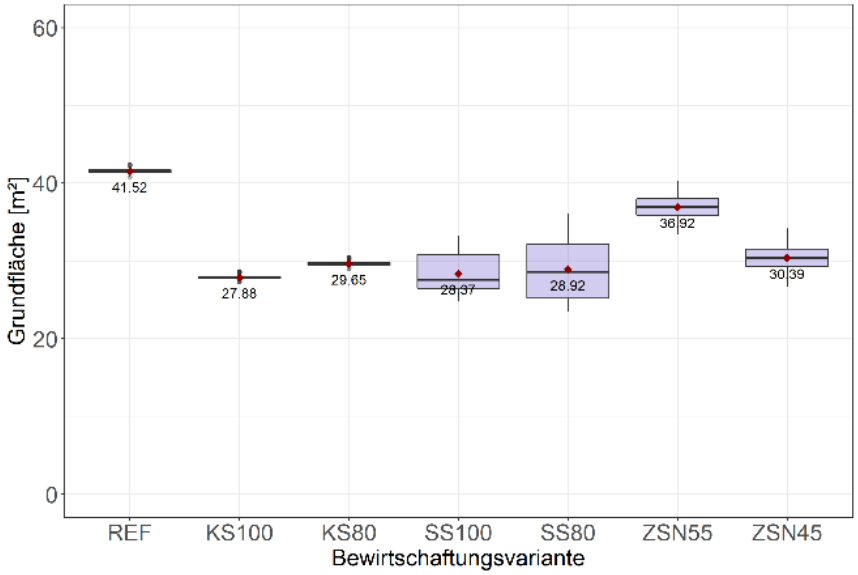
(a)



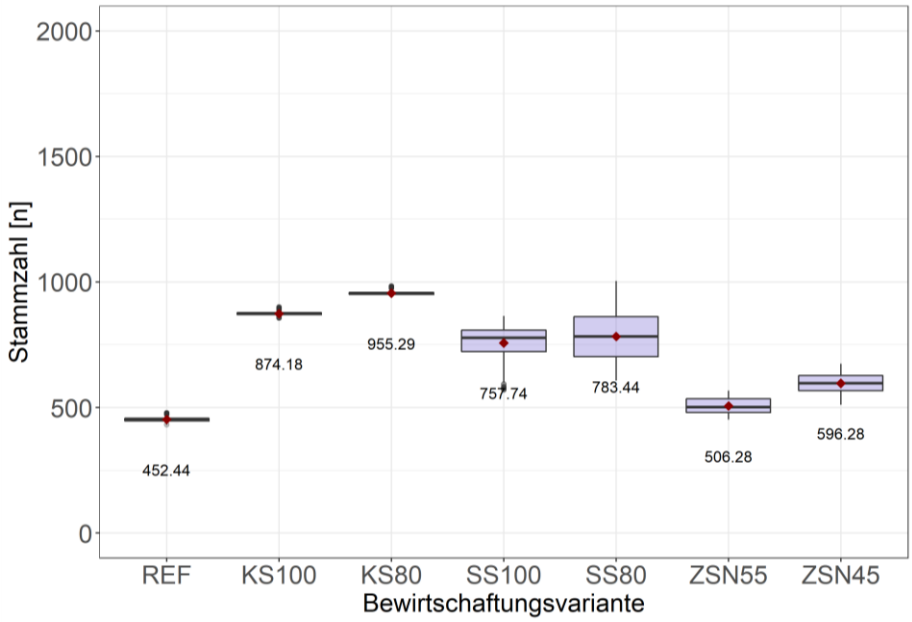
(b)



(c)


60
40
20
0
Grundfläche [m²]
41.52
27.88
29.65
28.37
28.92
36.92
30.39
REF
KS100
KS80
SS100
SS80
ZSN55
ZSN45
Bewirtschaftungsvariante

(d)


2000
1500
1000
500
0
Stammzahl [n]
452.44
874.18
955.29
757.74
783.44
506.28
596.28
REF
KS100
KS80
SS100
SS80
ZSN55
ZSN45
Bewirtschaftungsvariante

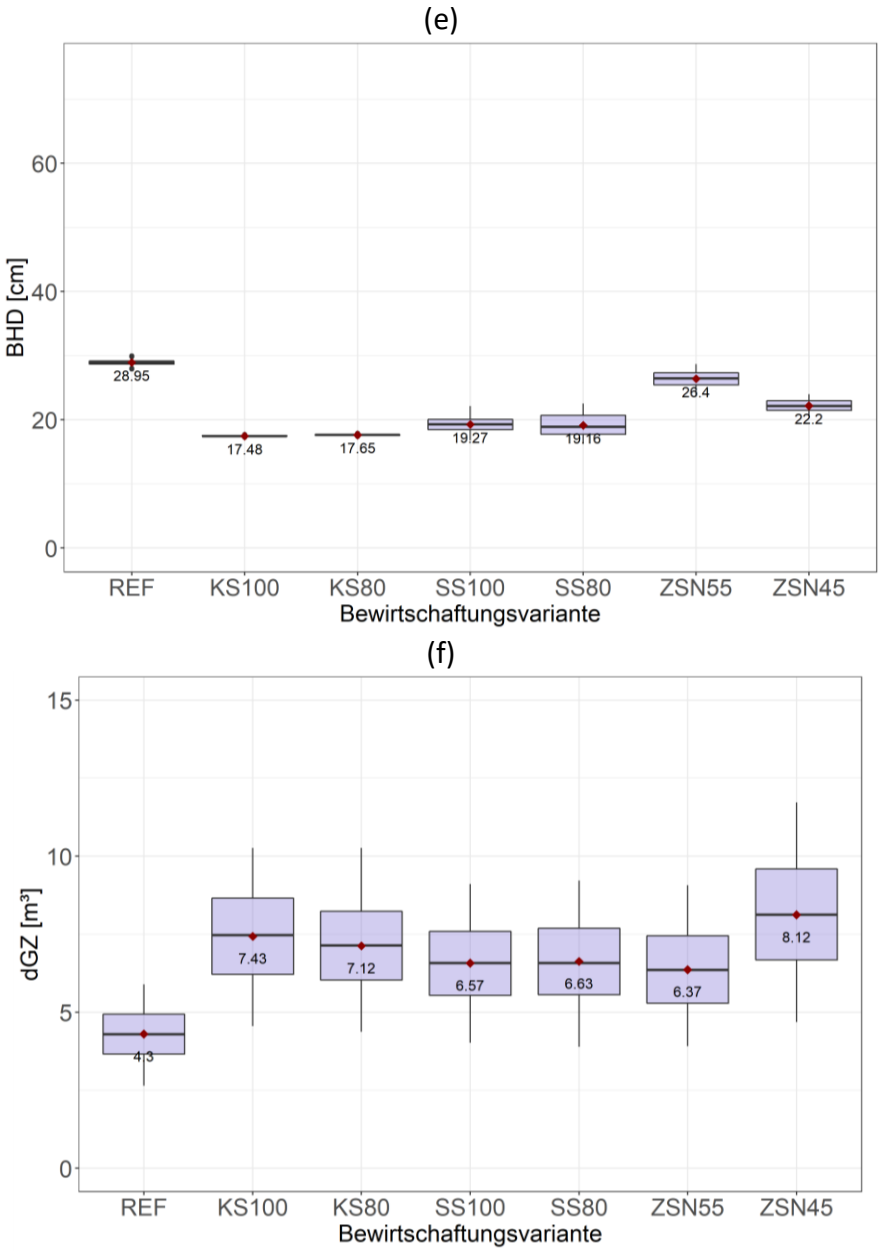


Abbildung 13: Darstellung unterschiedlicher forstlicher Parameter pro Hektar zwischen den Bewirtschaftungsvarianten simuliert mit iLand. Die Diagramme wurden gereiht nach (a) Volumen, (b) Mittehöhe, (c) Grundfläche, (d) Stammzahl, (e) BHD, (f) dGZ. Auf der Y-Achse ist jeweils der forstliche Parameter aufgetragen, auf der X-Achse die entsprechende Bewirtschaftungsvariante. Die roten Punkte mit zugehörigen Werten markieren die Mittelwerte.

Die Boxplot-Diagramme in Abbildung 13 bestätigen die Annahmen gegenüber den waldbaulichen Bewirtschaftungsvarianten. Beispielsweise erwartet man bei den Kahl- und Saumschlagvarianten mit 80 Jahren Umtriebszeit (KS80, SS80) in beiden Fällen höhere Stammzahlen als bei 100 Jahren Umtriebszeit und das bestätigt sich auch. Ebenso, dass bei einer Zielstärkenutzung mit BHD 45cm eine höhere Stammzahl pro Hektar auftritt, als bei einer Zielstärkenutzung ab BHD 55cm und dass eine unbewirtschaftete Landschaft zwar die niedrigste Stammzahl aufweist, aber die höchste durchschnittliche Oberhöhe und den

stärksten durchschnittlichen BHD. Diese und noch viele weitere waldbaulich plausible Ergebnisse lassen sich aus Abbildung 13 schließen.

Die einzige nicht plausible Tatsache ist die höhere mittlere Oberhöhe der Bewirtschaftungsvariante SS80 gegenüber SS100 bei den Oberhöhenverteilungen in Abbildung 10. Dieser Punkt wird später noch einmal genauer beleuchtet. Die Abfolge der Saumschläge jeder Zelle passiert räumlich verteilt auf einem Hektar. Die kleinste räumliche Einheit (selbe Situation in SVD), der ein Vegetationszustand zugewiesen werden kann, liegt aber bei einem Hektar und nicht darunter. Das könnte zu einer Überschätzung in der Strukturklasse führen und letztendlich zu einer verzerrten Ergebnisdarstellung. Szenarien mit größeren räumlichen Einheiten könnten zu einer Reduktion dieser Überschätzung führen und bessere Ergebnisse liefern.

4.2 DNN TRAINING UND EVALUATION

Die Erarbeitung von DNNs für die Simulation von Waldbewirtschaftung in SVD führte im Laufe der Entwicklung zu drei verschiedenen *deep neural networks*:

(1) DNN1 für Einzelmanagements: das Netzwerk erhält Information über einzelne Eingriffe (z.B. „Durchforstung 1“)
(2) DNN2 für Bewirtschaftungsvarianten: das Netzwerk erhält lediglich die Information über die Bewirtschaftungsvariante (z.B. „KS80“)
(3) DNN3 als Kombination aus (1) und (2)

Die Netzwerkarchitektur dieser DNNs ist untereinander weitgehend gleich und unterscheidet sich lediglich im Informationsinput zur Bewirtschaftung (siehe Kapitel 3.4.3-3.4.5). Die Trainings- und Evaluierungsdatensätze, die durch iLand-Simulationen generiert worden sind, werden in alle drei Netzwerke gespeist. Der Unterschied liegt also in den Daten, welche jedes individuelle DNN aus diesen Trainingsdaten zieht.

4.2.1 Finale Netzwerkarchitektur

Die finale Architektur (siehe Abbildung 14 und Abbildung 15) eines DNNs weist 284 775 trainierbare Parameter auf, wobei keine nicht-trainierbaren Parameter existieren. Die abhängigen Variablen *nächster Vegetationstand S** und *Zeit bis zur nächsten Zustandsänderung ΔR* werden als Wahrscheinlichkeitsverteilung über 45 Vegetationsklassen beziehungsweise 10 Klassen für die Zeit an den Ausgang des Netzwerkes geliefert. Das Netzwerk wurde anfänglich mit 45 möglichen vorkommenden Vegetationszuständen entworfen. Schlussendlich sind über alle Bewirtschaftungsvarianten nur 41 Vegetationszustände vorkommen. Der Trainingsdatensatz von iLand lieferte 6.0 Mio. Trainingsbeispiele, wobei jene Zustände entfernt wurden, die eine geringere Frequenz als 0.0001% aufwiesen. 80% der Daten (4.8 Mio.) wurden tatsächlich für das Training verwendet, die anderen 20% (1.2 Mio.) Beispiele wurden als unabhängige Validierungsdaten verwendet. Jedes Netzwerk wurde also gegen 1.2 Mio. Beispiele, welche für das Netzwerk unbekannt waren, getestet und validiert.

Struktur und Aufbau sind dem DNN-Design von Rammer und Seidl (2019a) und Rammer und Seidl (2019c) stark angelehnt und unterscheiden sich durch die genutzten Bewirtschaftungssignale in Tabelle 6.

Tabelle 6: Eingangsvariablen und abhängige Variablen der drei DNN-Designs. Insgesamt stehen dem Netzwerk pro Trainingsbeispiel 249 numerische Variablen zur Verfügung. 247 Eingangsvariablen und 2 abhängige Variablen um einen Übergang beschreiben zu können. Die drei Versionen von DNNs unterscheiden sich in der Verwendung der Information der Bewirtschaftungssignale für das jeweilige Netzwerk (siehe Einfärbung).

Eingangsvariablen	
S	Aktueller Vegetationszustand, 1 Wert
R	Verweilzeit, Anzahl der Jahre in der die Zelle bereits in dem aktuellen Zustand ist, 1 Wert
SSC	Bodenparameter: statische Parameter wie Stickstoff (kg/ha/Jahr) und Sand/Bodentiefe (m), 2 Werte
	Bewirtschaftungssignale:
M/T	Einzeleingriffsart (Code) und Zeit bis zum Eingriff, 2 Werte
REG	Bewirtschaftungsvariante/-regime: Numerischer Zahlencode, 1 Wert
CLIM	Klimadaten: Monatsmittel für Temperatur und Niederschlag für 10 Jahre, 12x2x10=240 Werte
abhängige Variablen	
S*	Nächster Vegetationszustand, kann auch dem aktuellen Zustand entsprechen, 1 Wert
ΔR	Zeit bis zur nächsten Zustandsänderung, kann 10 sein, wenn S*=S, 1 Wert

Die abhängigen Variablen sind jene Variablen, welche ein Netzwerk am Ende seines Trainings gelernt hat, vorherzusagen und werden schon während des Trainings zur Evaluierung verwendet. Sie spiegeln die Daten wider, gegen die die netzeigenen Vorhersagen verglichen werden.

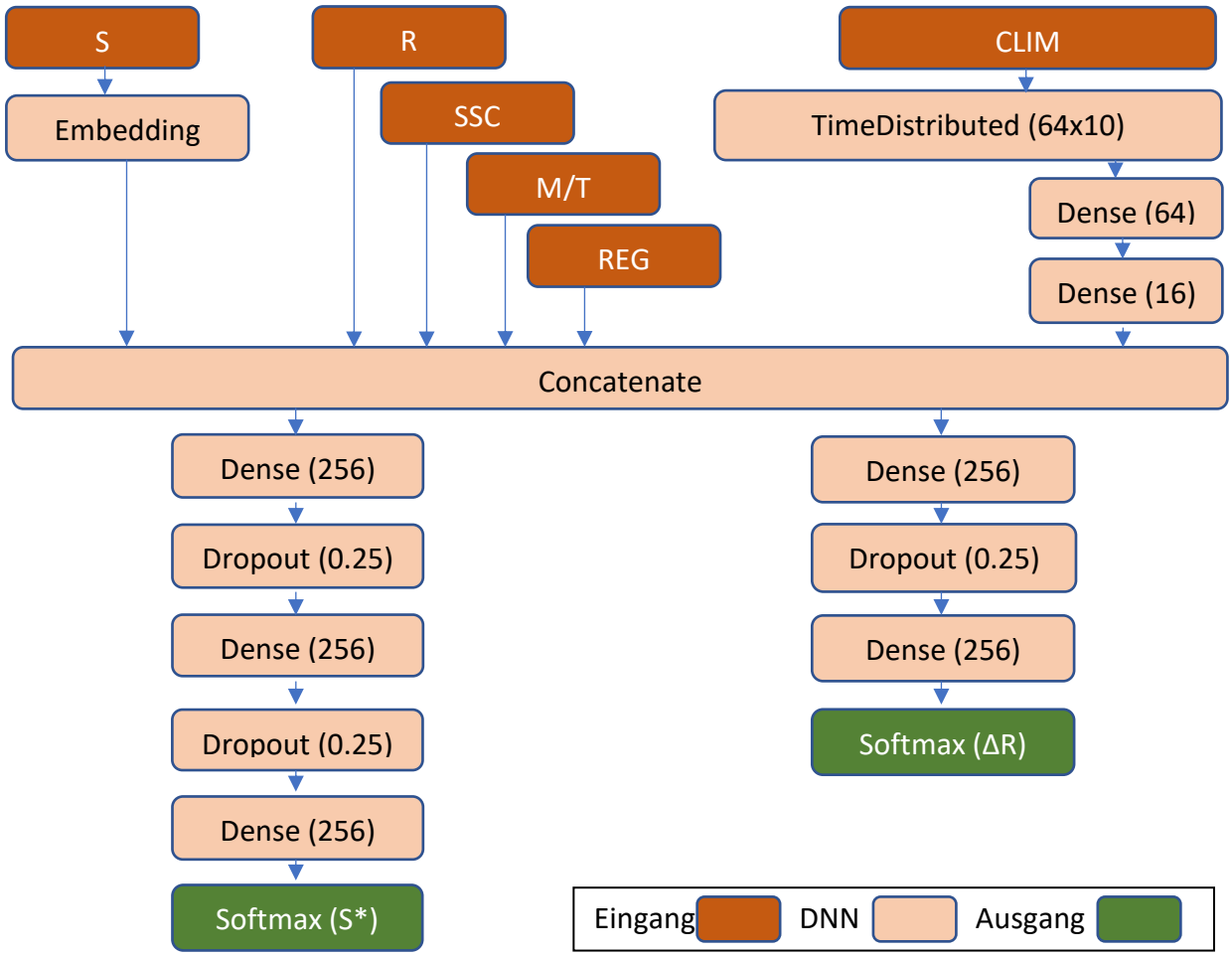


Abbildung 14: Finale Netzwerkarchitektur des trainierten kombinierten Netzwerkes DNN3. Der Eingang des Vegetationszustandes wird durch ein Embedding vorverarbeitet, genauso wie die Klima- und Niederschlagsdaten durch eine spezielle Schicht vorverarbeitet werden, bevor diese beiden Eingänge mit allen anderen Eingängen in einer gemeinsamen concatenate-Schicht zusammengeführt werden. Daraus spalten sich die zwei Ausgänge ab. Die Softmax-Schichten liefern eine Wahrscheinlichkeitsverteilung über die zuvor definierte Anzahl an Klassen pro Ausgang. Dense-Schichten zeigen stark verknüpfte Neuronen Schichten mit der Anzahl an Neuronen (in den Klammern) an. Dropouts geben an, wieviel Prozent der Neuronenverbindungen nach jeder Schicht und jedem Durchlauf gekappt werden, um die Generalisierung des Netzwerkes zu verbessern.

TensorFlow ermöglicht eine zusätzliche detaillierte Darstellung der Netzwerkarchitektur in Abbildung 15. Zu sehen sind die Eingangs- wie Ausgangsschichten, sowie die trainierbaren Parameter je Schicht.

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
climate (InputLayer)            [(None, 10, 24)]     0
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, 10, 64)       1600        climate[0][0]
__________________________________________________________________________________________________
state (InputLayer)              [(None, 1)]          0
__________________________________________________________________________________________________
reshape (Reshape)               (None, 640)          0           time_distributed[0][0]
__________________________________________________________________________________________________
state_em (Embedding)            (None, 1, 32)        1440        state[0][0]
__________________________________________________________________________________________________
envx (Dense)                    (None, 64)           41024       reshape[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 32)           0           state_em[0][0]
__________________________________________________________________________________________________
restime (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
envx2 (Dense)                   (None, 16)           1040        envx[0][0]
__________________________________________________________________________________________________
site (InputLayer)               [(None, 2)]          0
__________________________________________________________________________________________________
mgmt (InputLayer)               [(None, 2)]          0
__________________________________________________________________________________________________
regime (InputLayer)             [(None, 1)]          0
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 54)           0           reshape_1[0][0]
                                                                 restime[0][0]
                                                                 envx2[0][0]
                                                                 site[0][0]
                                                                 mgmt[0][0]
                                                                 regime[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 256)          14080       concatenate[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 256)          0           dense_3[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 256)          65792       dropout_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          14080       concatenate[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 256)          0           dense_4[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 256)          0           dense_1[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 256)          65792       dropout_2[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 256)          65792       dropout[0][0]
__________________________________________________________________________________________________
targetState (Dense)             (None, 45)           11565       dense_5[0][0]
__________________________________________________________________________________________________
targetTime (Dense)              (None, 10)           2570        dense_2[0][0]
==================================================================================================
Total params: 284,775
Trainable params: 284,775
Non-trainable params: 0
__________________________________________________________________________________________________
```

Abbildung 15: Diese Abbildung stellt die Modellarchitektur des DNN3 dar. Insgesamt gibt es 284 775 Parameter, welche während eines Trainings bis zur letzten Trainingsepoche variiert und trainiert werden. Die Modellarchitektur besteht aus 6 Eingangsschichten, 10 versteckte Schichten und 2 Ausgangsschichten. Zu sehen sind die Namen der Schichten, in welche Formen die Eingangsdaten gebracht werden, die trainierbaren Parameter je Schicht sowie die Verbindungen unter den Schichten selbst.

4.2.2 Validierung der DNNs

TensorBoard ist ein Visualisierungswerkzeug, das für DNN-Trainings mit *TensorFlow* verwendet wird, da Netzwerktrainings oft komplex und unübersichtlich sein können. Die grafische Darstellung eines Netzwerktrainings hilft Fehler zu finden, das Netzwerk besser zu verstehen und zu optimieren (TensorFlow, 2017). Außerdem können diverse Versionen eines oder mehrerer Netzwerke übersichtlich dargestellt und miteinander verglichen werden. Das hilft Netzwerke systematisch zu verbessern. Ein Ausschnitt, wie sich DNN1-DNN3 im Trainingsprozess unterscheiden, sieht man in Abbildung 16.

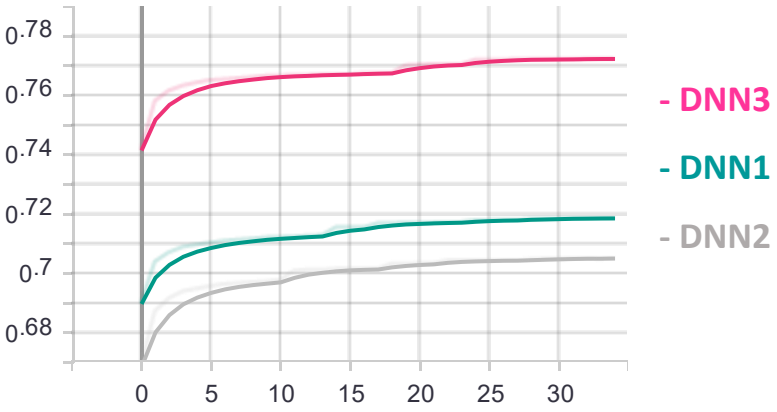


Abbildung 16: Ein Graph aus TensorBoard mit den Lernkurven der drei DNN-Varianten. Auf der X-Achse befinden sich die Trainingsepochen. Auf der Y-Achse die epoch target state accuracy, also die Genauigkeit, wie oft das DNN den richtigen Vegetationszustand in der jeweiligen Epoche vorhersagt. DNN3: 0.772; DNN1: 0.719; DNN2: 0.705

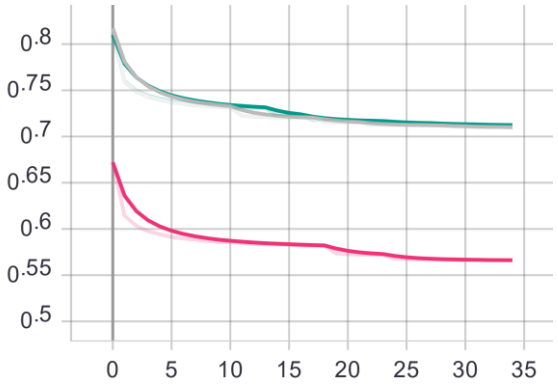
Mit *TensorBoard* erhält man einen genauen Einblick auf verschiedenste Netzwerkmetriken, welche Aufschlüsse über die Effizienz geben, angeführt sind zwei Beispiele:

- Genauigkeit pro Epoche oder *epoch target state accuracy*
- Verlust pro Epoche oder *epoch loss*

Simpel ausgedrückt, beschreibt die *epoch target state accuracy* wie oft die Vorhersagen den Erwartungswerten in der aktuellen Trainingsepoche entsprechen (TensorFlow, 2019). Die allgemeine Definition von Verlust (Eng. *loss*) wird in ML als eine skalare Funktion, die die Differenz zwischen einer DNN-Vorhersage und dem erwarteten Wert beschreibt, bezeichnet (Abadi *et al.*, 2015).

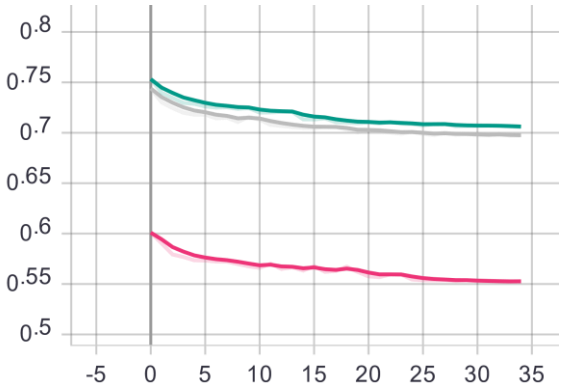
Es wurden die gleichen Netzwerkmetriken für die Zustandsänderung S* und die Zeit bis zur Zustandsänderung ΔR berechnet und jeweils für die Validierungsdurchläufe beider.

Trainingsfehler



- **DNN1**: 0.712
- DNN2: 0.710
- **DNN3**: 0.566

Testfehler



- **DNN1**: 0.706
- DNN2: 0.697
- **DNN3**: 0.553

Abbildung 17: Validierungsdiagramme aus TensorBoard. Auf der Y-Achse ist die prozentuale Fehlerrate und auf der X-Achse die Anzahl der durchlaufenen Epochen zu sehen. Es werden der Trainingsfehler und der Testfehler verglichen. Ist der Testfehler deutlich höher als der Trainingsfehler, kommt es zu einer Überanpassung des ML-Modells, liegt der Testfehler weit unter dem Trainingsfehler, kommt es zu einer Unteranpassung (das ML-Modell ist nicht exakt genug).

Durch die vorangegangene Aufspaltung der erzeugten Trainingsdaten aus iLand in 80% Trainings- und 20% Testdaten, führt *TensorFlow* nach jeder Trainingsepoche sofort eine Validierung mit den Testdaten durch. Diese Ergebnisse werden anschließend in den *TensorBoard* Validierungsdiagrammen dargestellt (siehe Abbildung 17). Trainingsfehler und Testfehler sind fast deckungsgleich – es kommt zu keiner Über- oder Unteranpassung der DNNs, was einer guten Generalisierung entspricht.

In Python kann man das DNN eine Vorhersage über die Verweilzeit, in der eine Zelle noch in ihrem aktuellen Vegetationszustand ist und über einen möglichen geänderten zukünftigen Vegetationszustand erstellen lassen. Diese Vorhersagen können dann in R mit Hilfe von Konfusionsmatrizen gegen die Evaluierungsdaten geprüft werden. Dies sollte einerseits die bereits visualisierten Ergebnisse aus *TensorBoard* widerspiegeln, andererseits können dadurch die Metriken auf die einzelnen Bewirtschaftungsvarianten aufgeteilt werden. Ein Vergleich der *TensorBoard* Diagramme mit den Ergebnissen aus Abbildung 18 und Abbildung 20 macht das sichtbar.

4.2.3 Ergebnisse der DNNs: Vegetationszustand und Verweilzeit

In diesem Abschnitt werden die Gesamtergebnisse sowie Auszüge von Ergebnissen der Netzwerkvariante DNN3 vorgestellt. Diese beziehen sich auf ihre Leistungen relativ zu iLand.

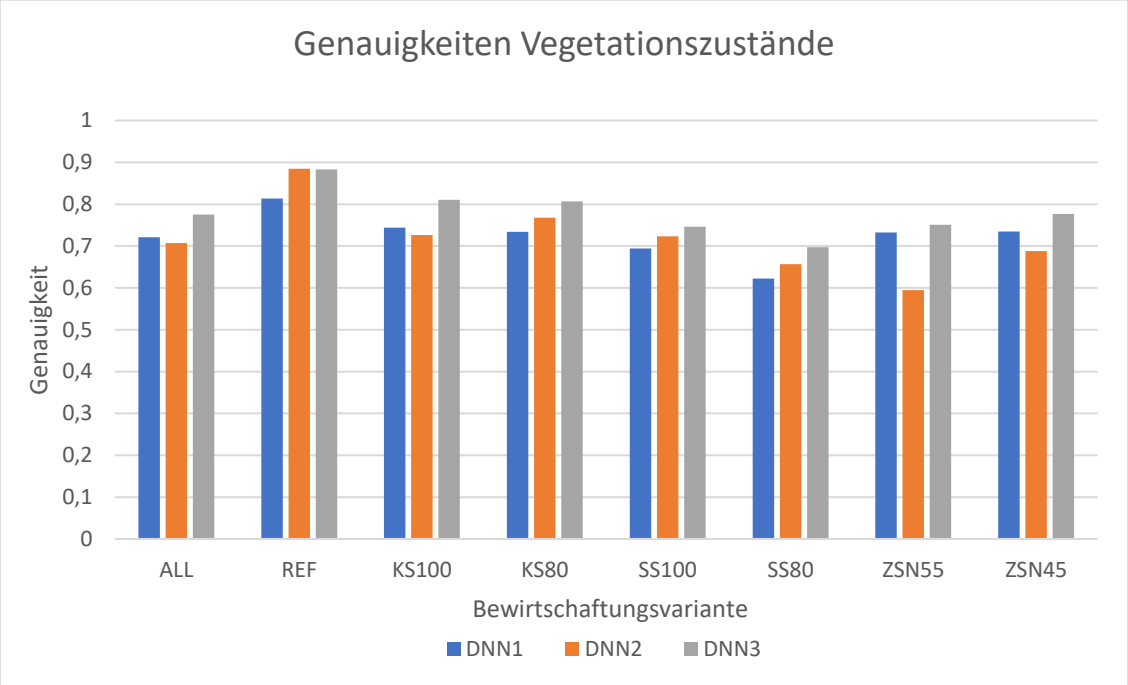


Abbildung 18: Die Genauigkeiten für die Vorhersagen zu den Vegetationszustandsänderungen S für alle drei DNNs. Die Genauigkeit oder auch Accuracy auf der Y-Achse ist eine ML-Metrik und dimensionslos. Sie gibt an wie gut die vorhergesagten Vegetationszustände des DNNs mit den erwarteten Werten der Evaluierungsdaten übereinstimmen. „ALL" ist die Gesamtgenauigkeit des jeweiligen DNNs.*

Tabelle 7: Zugehörige Genauigkeitswerte zu Abbildung 18. Alle statistischen Auswertungen finden sich im Anhang.

	ALL	REF	KS100	KS80	SS100	SS80	ZSN55	ZSN45
DNN (1)	0.7209	0.8136	0.7437	0.7342	0.6944	0.6226	0.7325	0.7346
DNN (2)	0.7075	0.8846	0.7262	0.7675	0.7229	0.6568	0.5947	0.6881
DNN (3)	0.7753	0.883	0.8106	0.807	0.7463	0.6975	0.7509	0.7767

Die Genauigkeiten der Vegetationszustände (siehe Tabelle 7) sowie die der Verweilzeiten (siehe Tabelle 8) wurden durch Konfusionsmatrizen berechnet. Hier wird nicht die aufsummierte Verteilung der Vegetationszustände wie in Abbildung 19 gegen die Verteilung von iLand geprüft, sondern die tatsächliche Genauigkeit (alle statistischen Auswertungen befinden sich in Kapitel 10.1 und 10.2). Das bedeutet, dass ein zukünftiger erwarteter Vegetationszustand mit zugehöriger Verweilzeit für eine Zelle, welcher von iLand berechnet wurde, als Erwartungswert betrachtet wird. Zu jeder Zelle gibt es zu allen Zeitpunkten der Simulation eine Vorhersage zu beiden Variablen vom DNN. Die tatsächlich erwarteten Werte werden mit den Vorhersagen zu allen Zeitpunkten der Simulation und für alle Zellen gegengeprüft und liefern die Ergebnisse der Konfusionsmatrizen. Die Berechnung der

Konfusionsmatrizen erfolgte für alle Netzwerkvarianten DNN1, DNN2 und DNN3 sowie für alle Bewirtschaftungsvarianten inklusive einer Gesamtgenauigkeit. Wie erwartet fällt die Gesamtgenauigkeit bei DNN3 mit 77,5% am höchsten aus und bestätigt die Annahme, dass ein Netzwerk mit höherem Informationsgrad bessere Vorhersagen treffen kann. DNN3 erhält wie in Kapitel 3.4.5 beschrieben, als Eingangsinformation alle verfügbaren Daten, wie Art des Einzeleingriffs, Eingriffszeitpunkt und zugehörige Bewirtschaftungsvariante. Das zweitbeste Ergebnis lieferte DNN1 mit 72,1%. Auch dieses Ergebnis ist über den Informationsgrad zu erklären. DNN1 erhält als Signal die Art des Managementeingriffs und den Zeitpunkt des Eingriffs als Eingangsinformation. DNN2 weiß zwar um welche Bewirtschaftungsvariante es sich handelt, es fehlt aber der genaue Zeitpunkt sowie die Art des Eingriffs bei den Eingangsvariablen. Es muss also rein aus der Information der Bewirtschaftungsvariante , z.B. Kahlschlag mit 80 Jahren Umtriebszeit, zukünftige Eingriffszeitpunkte und -arten vorherzusagen lernen.

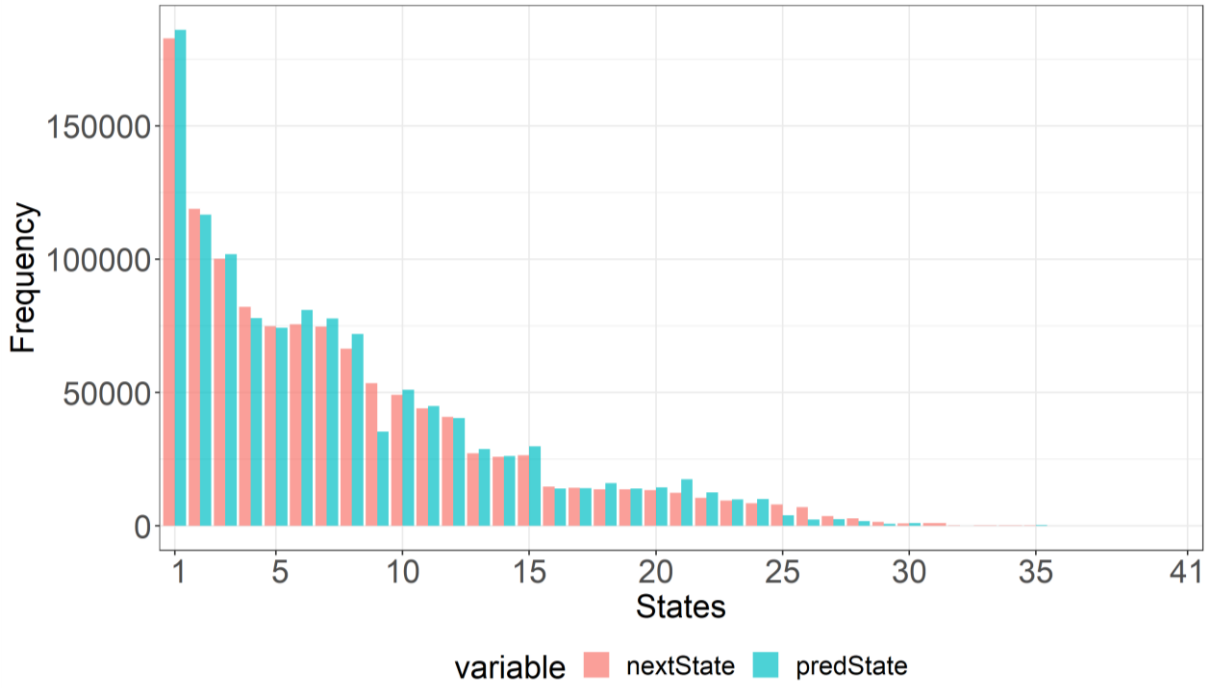


Abbildung 19: Aufsummierte Verteilung des nächsten Vegetationszustandes S von DNN3 über einen 1000 - jährigen Simulationszeitraum über die gesamte Landschaft. Die roten Balken repräsentieren die Summen der erwarteten Zustände vom PBM iLand. Die blauen Balken repräsentieren die vorhergesagten Vegetationszustände des kombinierten DNNs.*

Abbildung 19 ist ein Abbild der aufsummierten Verteilung aller Vegetationszustände. Zu beachten ist, dass dies die Vorhersagen für die Vegetationszustände über den gesamten Simulationszeitraumes sind und über alle Zellen der Landschaft. Daraus lässt sich also nicht ablesen, ob das Netzwerk genau die richtigen Zustände für jede individuelle Zelle für die nächste zukünftige Veränderung der Zelle vorhersagt, sondern, ob die generelle Verteilung

über die Landschaft stimmt. Das lässt zwar nur eine beschränkte Aussage über die Genauigkeit zu, die generelle Performance eines Netzwerkes kann damit aber abgeschätzt werden. Die Vorhersagen über die Vegetationszustände des kombinierten Netzwerkes DNN3 folgt mit den blauen Balken der Verteilung von iLand (rote Balken) sehr gut. Diese Performance spiegelt sich auch später in Abbildung 16 wider.

Abbildung 18 und Abbildung 20 zeigen auch unterschiedliche Performances der Netzwerksvarianten. DNN3 kristallisiert sich als leistungsstärkstes DNN über alle Bewirtschaftungsvarianten sowie der Referenz heraus, es gibt aber durchaus Unterschiede zwischen DNN1 und DNN2. Wie vorher bereits erwähnt, weist DNN1 eine gering höhere Genauigkeit bei den Vegetationszuständen wie auch der Zeit gegenüber DNN2 auf. Diese ist aber vor allem auf die besseren Vorhersagen bei den Bewirtschaftungsvarianten ZSN55 und ZSN45 zurückzuführen. ZSN55 und ZSN45 sind durch sehr kurze Eingriffsintervalle (8 Jahre bei ZSN55 und 5 Jahre bei ZSN45) geprägt. DNN1 erhält im Gegensatz zu DNN2 diesen Zeitimpuls als Eingangsinformation und kann dadurch bei diesen beiden Bewirtschaftungsvarianten bessere Ergebnisse erzielen.

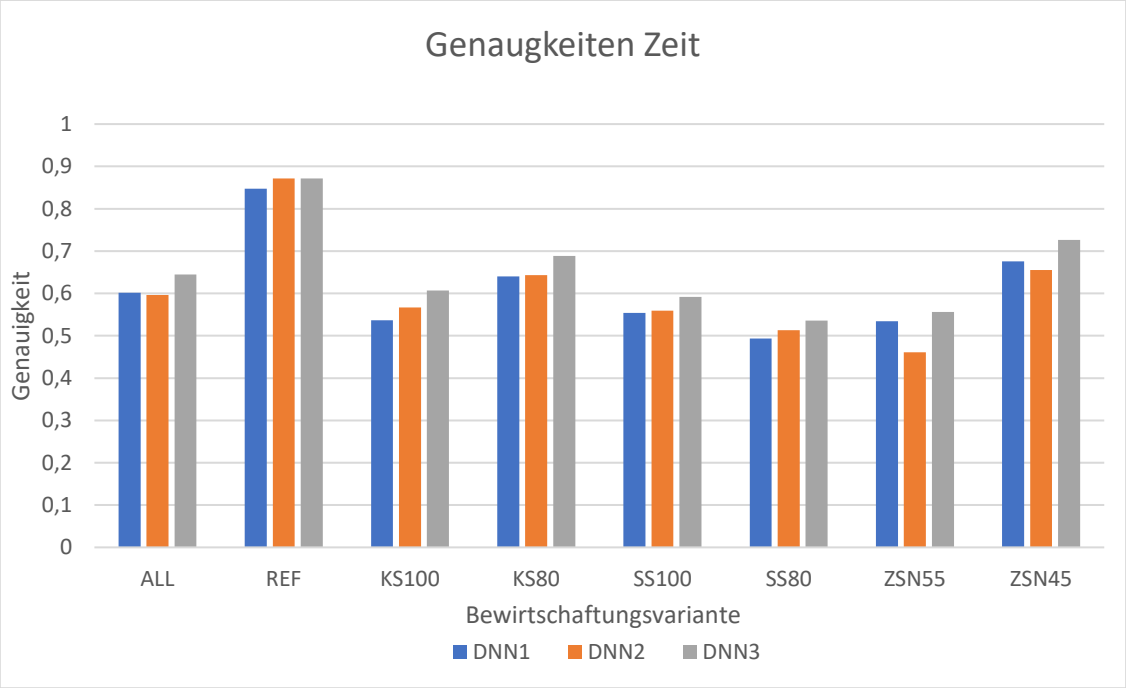


Abbildung 20: Die Genauigkeiten für die Vorhersagen über die Verweilzeit ΔR für alle drei DNNs. Die Genauigkeit oder auch Accuracy auf der Y-Achse ist eine ML-Metrik und dimensionslos. Sie gibt an wie gut die vorhergesagten Verweilzeiten des DNNs mit den erwarteten Werten der Evaluierungsdaten übereinstimmen. „ALL" ist die Gesamtgenauigkeit des jeweiligen Netzwerkes.

Tabelle 8: Zugehörige Genauigkeitswerte zu Abbildung 20. Alle statistischen Auswertungen finden sich im Anhang.

	ALL	REF	KS100	KS80	SS100	SS80	ZSN55	ZSN45
DNN (1)	0.6016	0.8474	0.5363	0.6403	0.554	0.4932	0.5343	0.6754
DNN (2)	0.5962	0.8716	0.5667	0.6431	0.5595	0.5134	0.4608	0.6551
DNN (3)	0.6448	0.8716	0.607	0.6883	0.592	0.536	0.5563	0.7264

Auffällig ist, dass die Variante SS80 (Saumschlag 80 Jahre Umtriebszeit) allen DNNs Probleme bereitet. Die Genauigkeiten für die Vegetationszustände liegen mit zwischen 62,3% für DNN1 und 69,8% für DNN3 deutlich im hinteren Teil des Feldes. SS100 kann nur marginal besser vorhergesagt werden.

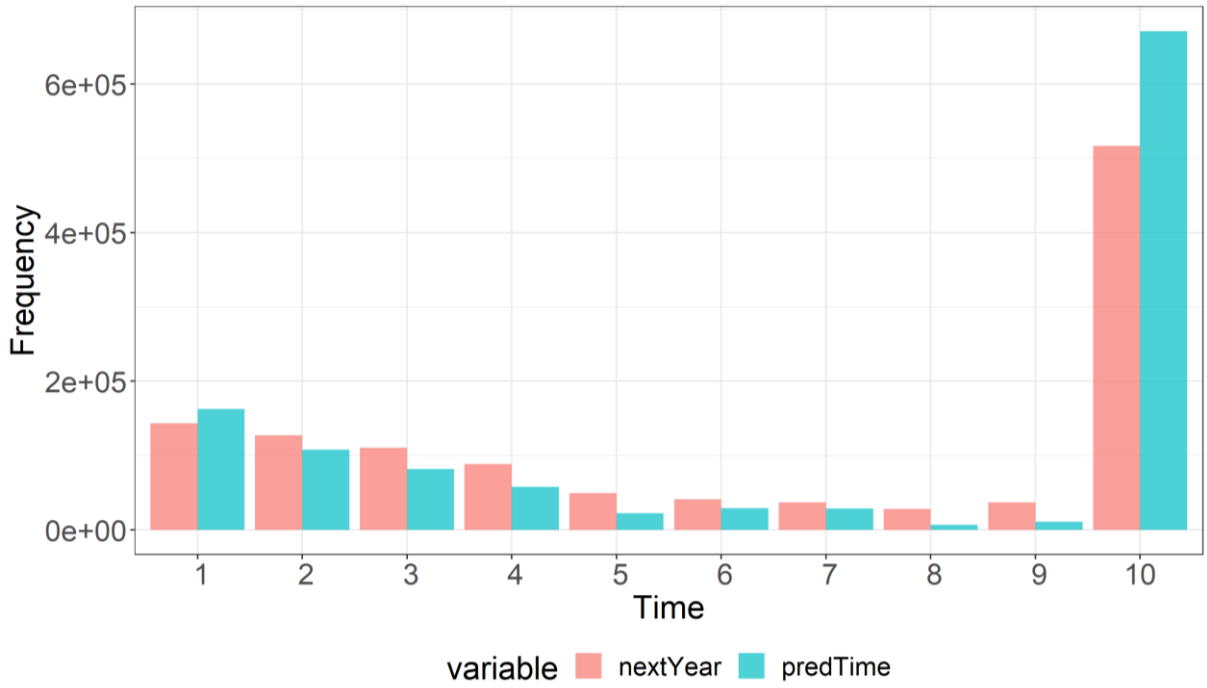


Abbildung 21: DNN3: Aufsummierte Verteilung der Verweilzeit ΔR über einen 1000 - jährigen Simulationszeitraum über die gesamte Landschaft und alle Bewirtschaftungsvarianten inklusive Referenz. Die roten Balken repräsentieren die Summen der errechneten Zeiträume vom PBM iLand. Die blauen Balken repräsentieren die vorhergesagten Zeiträume des kombinierten DNNs.

Obwohl die Gesamtgenauigkeit für die Verweilzeit mit 64,5% nur moderat ausfällt, kann das DNN3 die Verteilung von iLand relativ gut reproduzieren. Es ist festzustellen, dass in den meisten Fällen keine Änderung des Vegetationszustandes innerhalb des 10-jährigen Vorhersagehorizonts passiert und somit hier die Häufigkeiten am größten sind. Das einzelne Netzwerk hat also gelernt, dass der Wert 10 für die Vorhersage der Verweilzeit sehr oft der richtige Wert ist. Dadurch lässt sich aber sehr gut die Überschätzung des Netzwerkes für den Wert 10 bei der Verweilzeit erklären (siehe Abbildung 25).

Tabelle 9: Hier sind die drei verschiedenen Funktionsklassen in ein und derselben Strukturklasse „12m-16m“ des Vegetationszustandes: PIAB 12m-16m () rot markiert. DNNs schätzen oft in einen benachbarten Vegetationszustand, der dem erwarteten Vegetationszustands sehr nahe liegt.

description	composition	structure	functioning
0m-4m (LAI <2)	unforested	0	0
Irr: 0m-12m (LAI <2)	mix	21	0
PIAB 0m-4m (LAI <2)	PIAB	0	0
PIAB 12m-16m (LAI <2)	PIAB	3	0
PIAB 12m-16m (LAI >4)	PIAB	3	2
PIAB 12m-16m (LAI 2-4)	PIAB	3	1
PIAB 16m-20m (LAI <2)	PIAB	4	0
PIAB 16m-20m (LAI >4)	PIAB	4	2
PIAB 16m-20m (LAI 2-4)	PIAB	4	1

Tabelle 9 zeigt rot markiert die drei verschiedenen Funktionsklassen, welche pro Strukturklasse eines Vegetationszustandes vorkommen können. Ein Großteil der Fehlschätzungen der DNNs betreffen eine benachbarte Funktionsklasse gegenüber der erwarteten Funktionsklasse (siehe Abbildung 23).

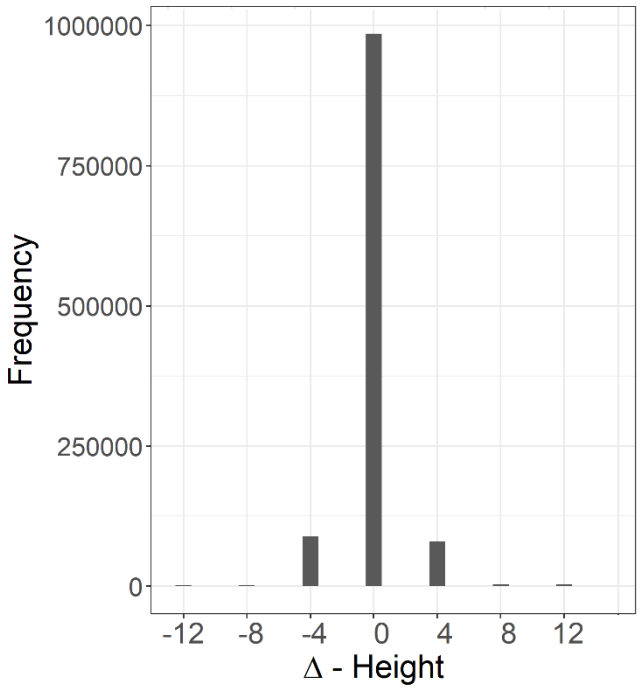


Abbildung 22: Abweichungen der Schätzungen der Höhenklassen des DNN3. Auf der Y-Achse befindet sich die Anzahl an Vorhersagen. Auf der X-Achse befinden sich die Abweichung der Schätzungen der Höhenklassen in Meter von den erwarteten Werten von iLand. Zu sehen ist das Gesamtergebnis des DNN3 über alle Bewirtschaftungsvarianten.

Im Fall der Oberhöheklassen kann man in Abbildung 22 beobachten, dass der Großteil der falschen Vorhersagen die nächste darunter oder darüberliegende Klasse betreffen. Der

Großteil der Fehlvorhersagen des DNN3 bezieht sich auf die strukturelle Nachbarklasse des erwarteten Vegetationszustandes. Abweichungen um zwei (± 8m) oder drei (± 12m) Strukturklassen treten sehr selten auf.

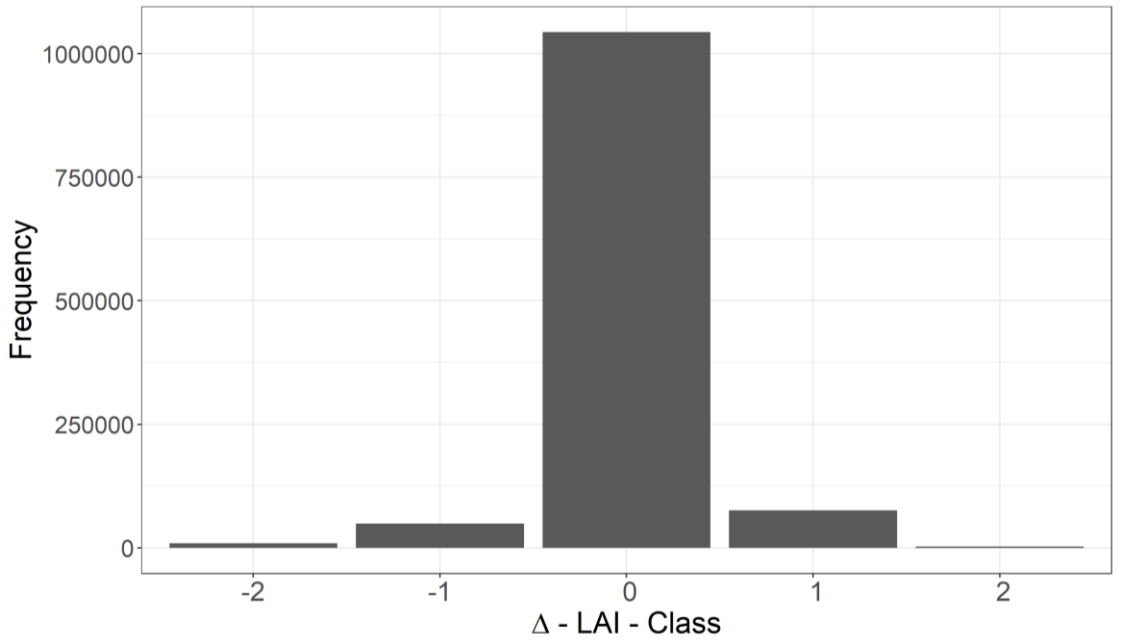


Abbildung 23: Abweichungen der Schätzungen der LAI-Klassen des DNN3. Auf der Y-Achse befindet sich die Anzahl an Vorhersagen. Auf der X-Achse befinden sich die Abweichungen der Schätzungen der Funktionsklassen (LAI) von den erwarteten Werten von iLand. Zu sehen ist das Gesamtergebnis des DNN3 über alle Bewirtschaftungsvarianten.

Auch die Abweichungen der Schätzungen der Funktionsklassen in Abbildung 23 zeigen, dass die meisten Fehlvorhersagen eine Funktionsklasse unter oder über dem Sollwert liegen. Natürlich gibt es nur drei verschiedene Funktionsklassen und eine Fehlschätzung wiegt somit schwerer. Zugleich sollte es für ein DNN leichter zu lernen sein, die richtige Funktionsklasse zu schätzen als beispielsweise die richtige Strukturklasse.

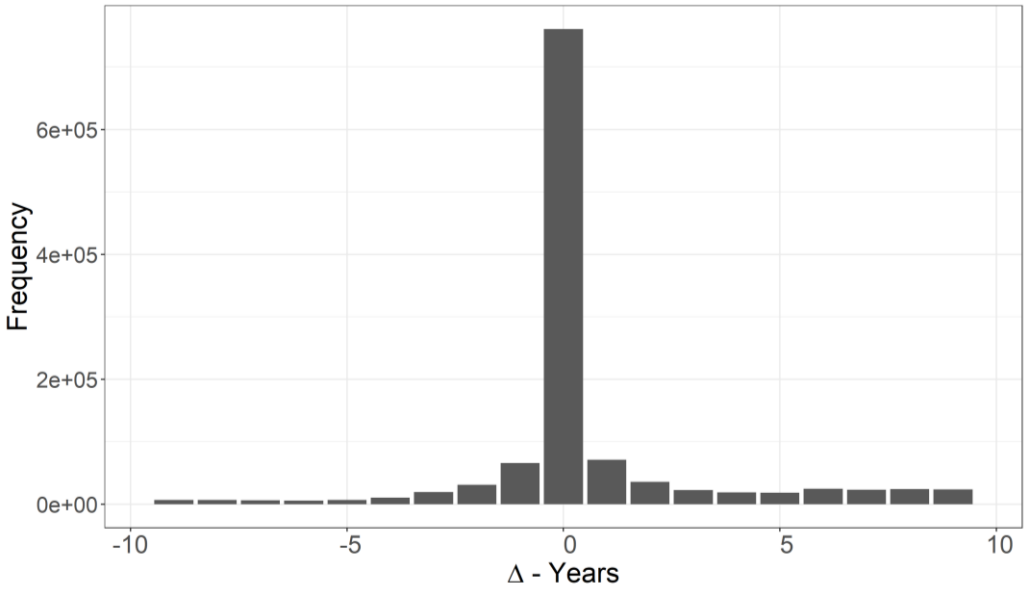


Abbildung 24: Abweichungen der Schätzungen der Verweilzeit ΔR des DNN3. Auf der Y-Achse befindet sich die Anzahl an Vorhersagen. Auf der X-Achse befinden sich die Abweichungen der Schätzungen der Verweilzeit von den erwarteten Werten. Zu sehen ist das Gesamtergebnis des DNN3 über alle Bewirtschaftungsvarianten.

Diese Analyse wurde für jedes DNN und für alle Bewirtschaftungsvarianten ausgeführt sowie eine Gesamtabweichung über alle Varianten errechnet. Dasselbe gilt für die Zeitvariable (siehe Abbildung 24). Die Genauigkeit für die Schätzungen der Zeitvariable waren für alle drei DNNs relativ niedrig (zwischen 59,6% für DNN2 und 64,5% für DNN3). Für ein Netzwerk war die häufigste Schätzung der maximale Zeithorizont 10 Jahre. Das erklärt auch die höheren Häufigkeiten der Abweichungen der Schätzung vom erwarteten Wert zwischen 1 und 10 Jahren. Um diese Tatsache zu veranschaulichen, wurde ein Dichtediagramm für DNN3 und die Schätzungen über die Zeit im Vergleich zu iLand erstellt. In Abbildung 25 sieht man eindeutig, dass der Wert 10 am häufigsten auftritt und das DNN3 diesen Wert verhältnismäßig oft schätzt.

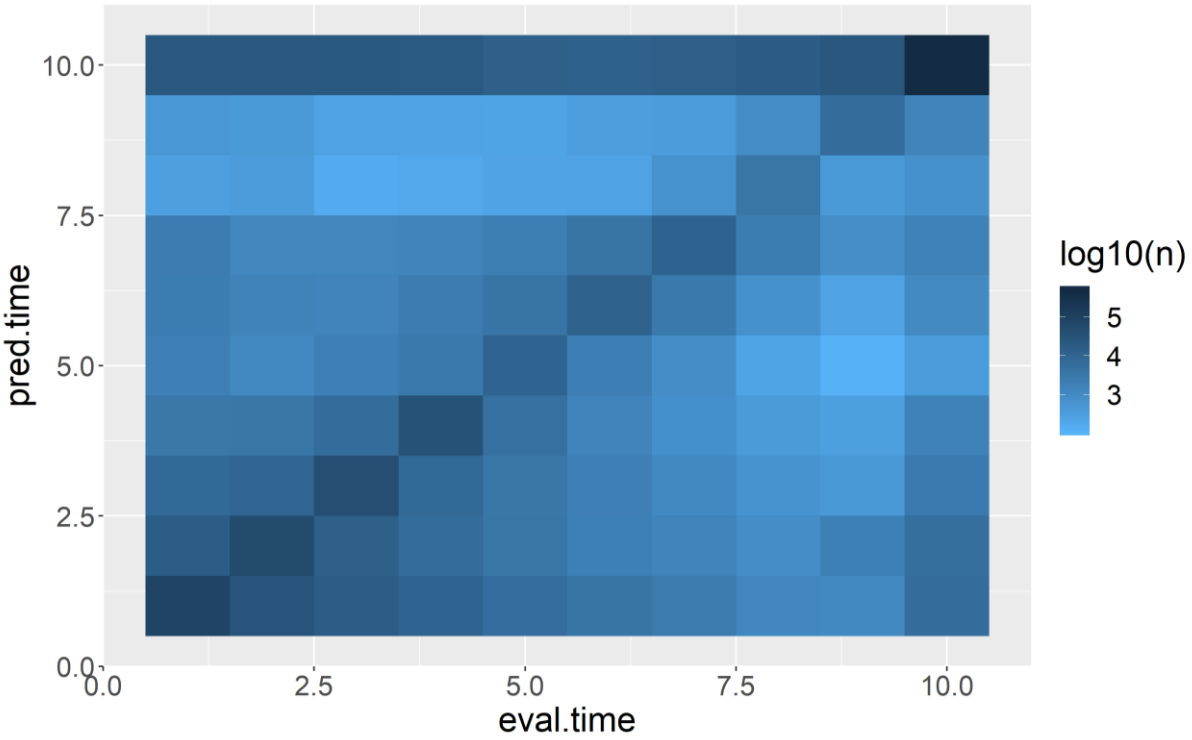


Abbildung 25: Dichtediagramm der geschätzten und erwarteten Verweilzeit ΔR. Auf der Y-Achse befinden sich die Schätzungen für die Verweilzeit des DNN3. Auf der X-Achse befinden sich die tatsächlich erwarteten Zeit-Werte von iLand. Bei 100% Übereinstimmung zwischen den Vorhersagen und den erwarteten Werten, sollten sich eine schwarze Diagonale vom Koordinatenursprung bis in das rechte obere Eck ergeben. Zu sehen ist, dass der Wert 10 am Häufigsten auftritt. Die Häufigkeiten wurden logarithmisch dargestellt. Gesamtergebnis des DNN3 über alle Bewirtschaftungsvarianten inklusive Referenz.

4.3 SVD Simulation und Evaluation

4.3.1 Quantitative Ergebnisse

Die Ergebnisse der SVD Simulationen werden evaluiert indem eine Gegenüberstellung von SVD Ergebnissen (DNN1-DNN3) und iLand Ergebnissen erfolgt. Um Aussagen über verschiedene forstliche Parameter (siehe Tabelle 10) treffen zu können, wurde jeder einzelner Vegetationszustand mit seinen entsprechenden forstlichen Eigenschaften verknüpft.

Tabelle 10: Listung der ausgewerteten forstlichen Parameter.

• Volumen • Oberhöhe • Mittelhöhe • Grundfläche • Stammzahl	• Brusthöhendurchmesser BDH • Durchschnittlicher Gesamtzuwachs dGZ • Kohlenstoffvorrat Total C • Leaf Area Index LAI

Wie in Kapitel 3.1.6 erwähnt wurde, ist jeder Vegetationszustand mit einem Eintrag in einer Datenbank (vegetation attribute database VAD) verknüpft, welche alle aggregierten forstlichen Parameter von iLand enthält (siehe Tabelle 11). Dafür wurden über alle in den Simulationen auftretenden Vegetationszuständen die zugewiesenen forstlichen Parameter aus iLand aufsummiert und nach der auftretenden Frequenz jedes Vegetationszustandes je Simulationslauf gewichtet gemittelt.

Tabelle 11: VAD (vegetation attribute database). In dieser Datenbank ist jeder einzigartige Vegetationszustand mit den aggregierten Werten aus iLand verknüpft. Um die Leserlichkeit zu erhalten, ist hier nur ein Ausschnitt der VAD zu sehen. Neben strukturellen Parametern (composition, structure, functioning) sind auch forstliche Parameter vertreten (volume_mean, basalarea_mean, …) die zur Analyse der SVD Simulationen dienten.

description	composition	structure	functioning	volume_mean	basalarea_mean	dbh_mean	height_mean	stems_m
PIAB 0m-4m (LAI <2)	PIAB	0	0	0.4323445	0.1533285	6.225201	5.670720	73.32065
PIAB 12m-16m (LAI <2)	PIAB	3	0	31.0506676	7.4295815	8.635820	7.791136	1188.420
PIAB 12m-16m (LAI >4)	PIAB	3	2	161.8158027	31.6362981	13.339464	11.524841	2222.744
PIAB 12m-16m (LAI 2-4)	PIAB	3	1	55.0247290	12.3899273	9.776782	8.685668	1547.347
PIAB 16m-20m (LAI <2)	PIAB	4	0	41.6216910	8.0796916	9.818305	8.641701	873.7507
PIAB 16m-20m (LAI >4)	PIAB	4	2	207.7102782	33.3749622	16.292842	13.663230	1608.652
PIAB 16m-20m (LAI 2-4)	PIAB	4	1	87.2990002	16.3186365	11.620683	9.945255	1385.596
PIAB 20m-24m (LAI <2)	PIAB	5	0	55.2551550	9.0619845	10.252095	8.812063	811.6825

Mit den Genauigkeiten der drei DNNs im Hintergrund (siehe Abbildung 18) wurden Boxplot Diagramme für alle drei DNN-Varianten in SVD inklusive der iLand Ergebnisse für jede Bewirtschaftungsvariante und für jeden forstlichen Parameter erstellt. Somit können die iLand Ergebnisse direkt mit jenen der DNN-Varianten in SVD verglichen werden. Diese Boxplots sind über die letzten 500 Jahre des Simulationszeitraumes aufsummiert sowie über alle Zellen. Zusätzlich wurde noch das Mittel mit einer roten Markierung sowie dem zugehörigen Zahlenwert angeführt.

Um der Frage der Strukturunterschiede zwischen den SVD Simulationen und iLand gerecht zu werden, werden Oberhöhe und LAI für die Bewirtschaftungsvarianten näher betrachtet. Die anderen in Tabelle 10 angeführten Größen finden sich im Anhang wieder. Es kann festgehalten werden, dass SVD in der Lage ist, quantitative Ergebnisse aus iLand nachzubilden. Auch wenn hier keine Entwicklung über eine Zeitreihe dargestellt und analysiert wird, sind die Endergebnisse eine gute Annäherung an iLand.

Es konnte festgestellt werden, dass die SVD Ergebnisse bei der Referenzvariante die größten Abweichungen über alle Parameter aufweist, gefolgt von ZSN55. Allein der durchschnittliche Gesamtzuwachs (dGZ) und Grundfläche zeigten bei REF gute Übereinstimmungen zwischen den erwarteten Werten und den Schätzungen. Beim dGZ werden bei allen Bewirtschaftungsvarianten die iLand-Ergebnisse von allen drei DNN Varianten unterschätzt, allein bei REF gibt es eine sehr gute Übereinstimmung zwischen DNN1, DNN2, DNN3 und iLand. Diese Tatsache ist nicht wie erwartet, da bei der Analyse der DNN-Genauigkeiten die Referenz die höchsten Genauigkeitskoeffizienten aufweist. Es treten Probleme bei Bewirtschaftungssimulationen in SVD mit Waldlandschaften auf, welche sich länger ohne starke Eingriffe (Kahlhieb) entwickeln können. Stärkere, sich ähnelnde Abweichungen, treten vorwiegend bei der nicht-bewirtschafteten Referenzlandschaft und ZSN55 auf.

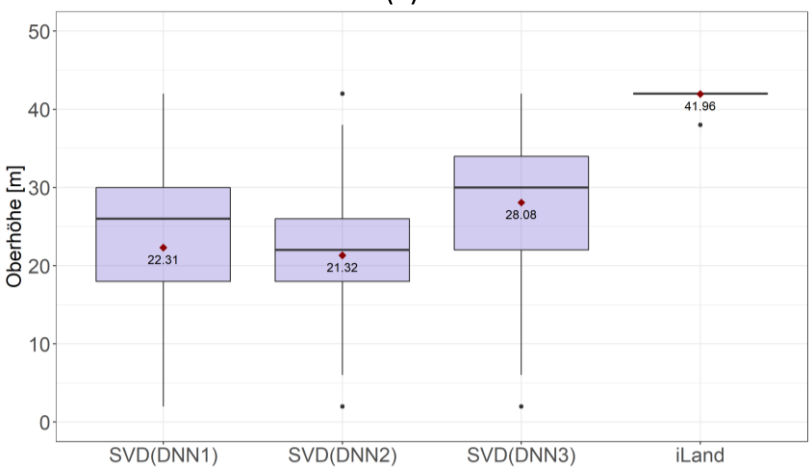
4.3.1.1 Oberhöhe

Wie in Abbildung 26 zu sehen ist, tritt die größte Abweichung zwischen der erwarteten Oberhöhe aus iLand und den DNN - Schätzungen der Referenz (a), gefolgt von der Bewirtschaftungsvariante ZSN55 (f) auf.

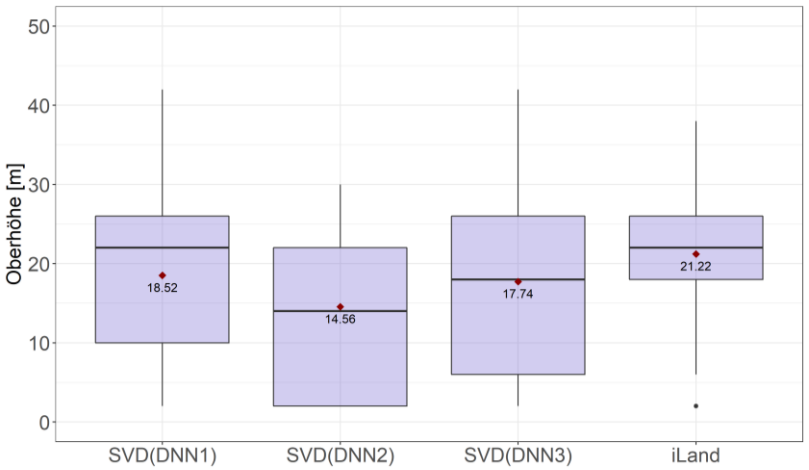
Obwohl die allgemeine Genauigkeit der DNNs für REF am aller höchsten ist, konnte diese Tatsache in Bezug auf die Oberhöhenschätzungen nicht bestätigt werden. Alle drei DNNs unterschätzen die mittlere Oberhöhe der Referenz (42m) stark mit einer maximalen Abweichung von Δ20.6m von DNN2. ZSN55 ähnelt von der Walddynamik REF am meisten unter den Bewirtschaftungsvarianten. Es werden in einem 8 Jahresintervall Bäume mit BHD <= 55cm entnommen und maximal 20% des Bestandesvolumens, ansonsten passiert kein Eingriff. Kein (REF) oder schwacher Eingriff (ZSN55) führt im Lauf der Zeit zu höheren Oberhöhen als stärker bewirtschaftete Varianten. Diese Tatsache wird von den DNNs tendenziell unterschätzt.

Die Vorhersagegenauigkeiten bei KS100, KS80, SS100, SS80 und ZSN45 sind aber durchaus zufriedenstellend.

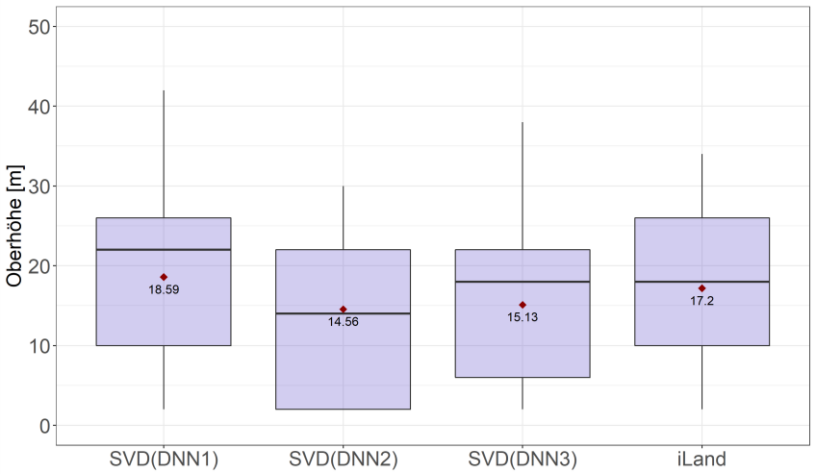
(a)



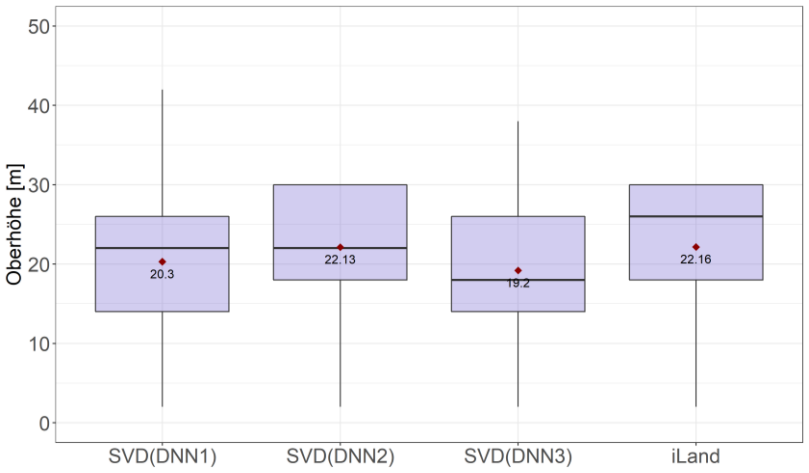
(b)


Oberhöhe [m]
50
40
30
20
10
0
18.52
14.56
17.74
21.22
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

(c)


Oberhöhe [m]
50
40
30
20
10
0
18.59
14.56
15.13
17.2
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

(d)


Oberhöhe [m]
50
40
30
20
10
0
20.3
22.13
19.2
22.16
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

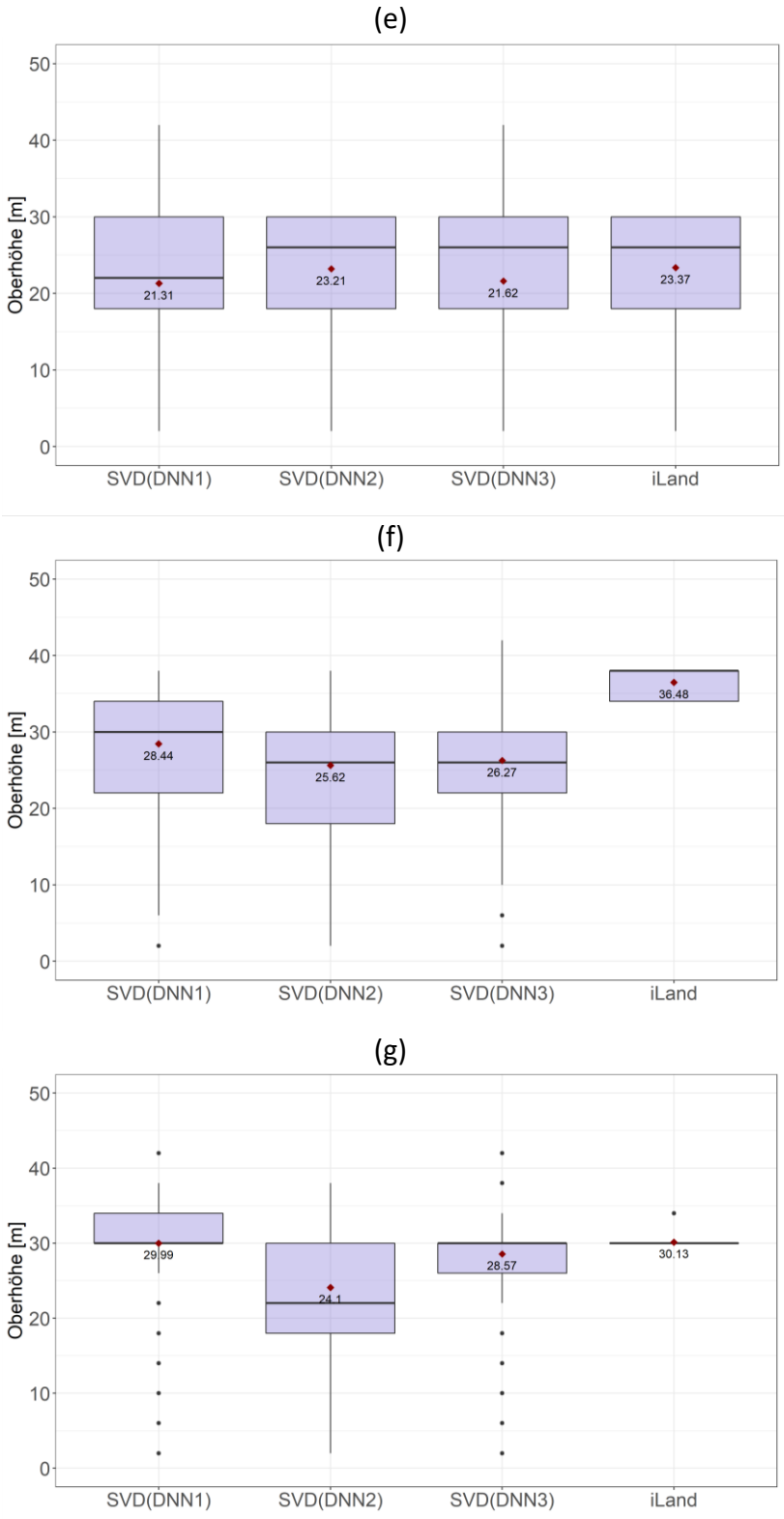
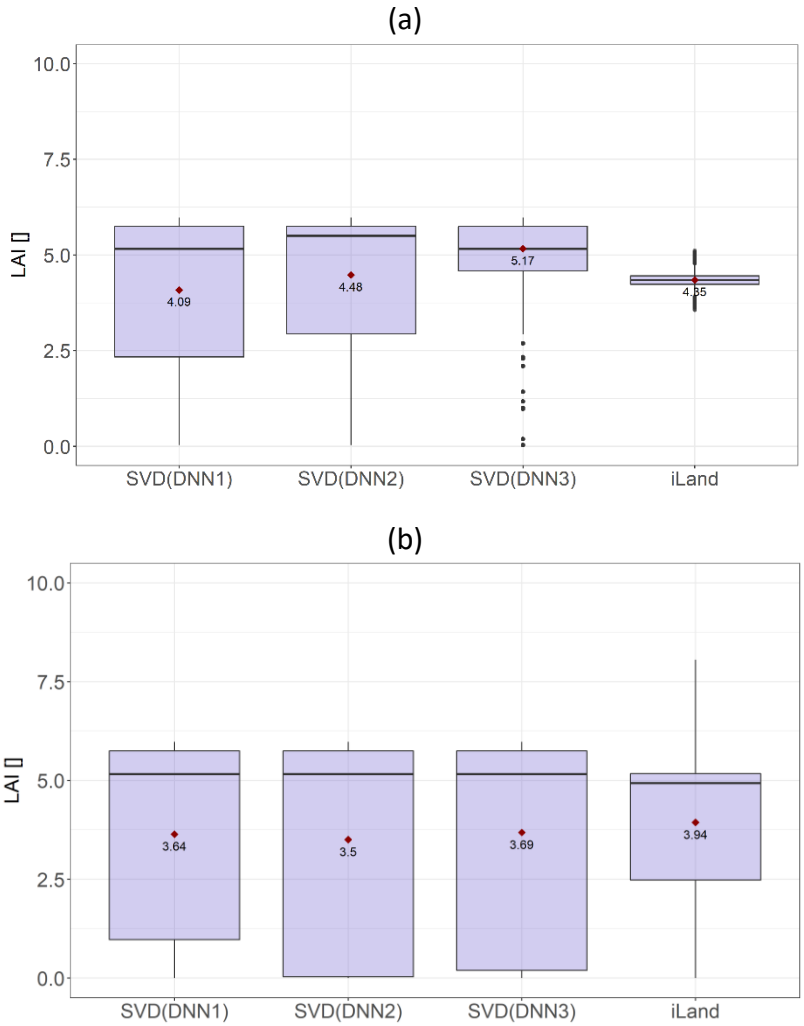


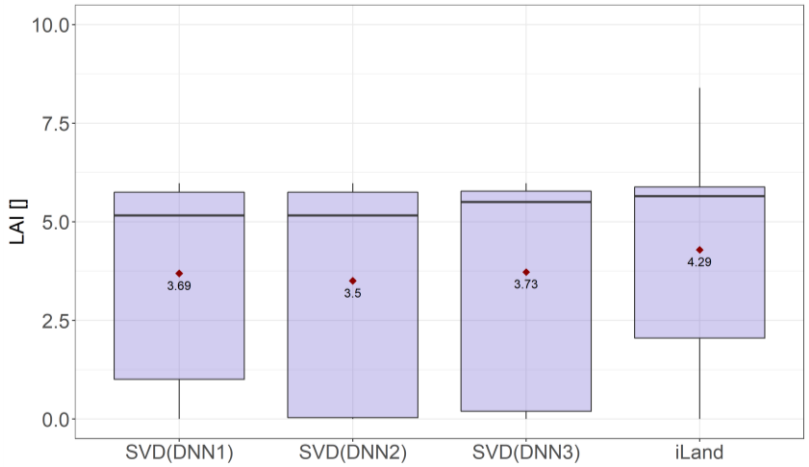
Abbildung 26: Oberhöhen in der Landschaft der einzelnen Bewirtschaftungsvarianten in der zweiten Simulationshälfte. Auf der Y-Achse befindet sich die Oberhöhe. Die Diagramme wurden gereiht nach den Bewirtschaftungsvarianten (a) REF, (b) KS100, (c) KS80, (d) SS100, (e) SS80, (f) ZSN55, (g) ZSN45. Die SVD Ergebnisse der drei DNN-Varianten werden den Ergebnissen aus iLand gegenübergestellt. Die roten Punkte geben die Mittel an.

4.3.1.2 Leaf Area Index

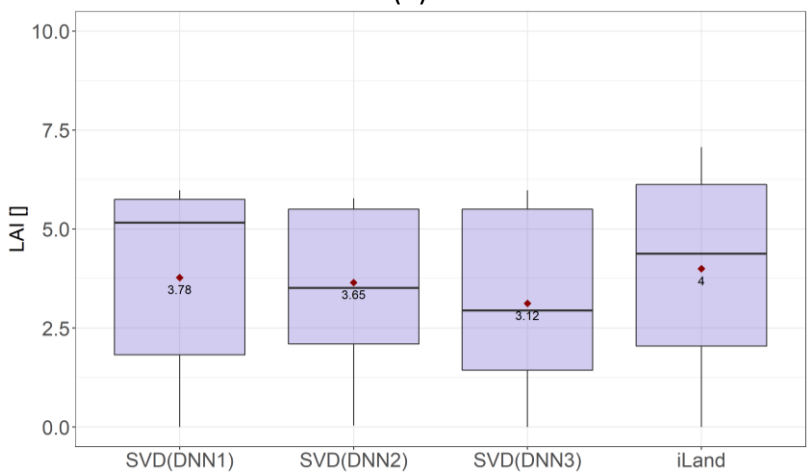
Die Referenz wird von den DNNs in Abbildung 27a tendenziell überschätzt, nur das Mittel von DNN1 liegt unter jenem von iLand. Bei den Bewirtschaftungsvarianten KS100 (b) und KS80 (c) schätzen alle drei DNNs den Median und Mittel sehr gut und liegen nahe den iLand-Ergebnissen. Die Mediane von SS100 und SS80 weichen sichtlich stärker von den iLand Medianen ab, die Differenz der Mittel der DNNs zu iLand bleibt bei den LAI-Schätzungen aber unter dem absoluten Wert von 1 (Δ0.9). Bei den Zielstärkenutzungen ZSN55 (f) und ZSN45 (g) setzt sich dieser Trend von REF fort. Es überschätzten alle DNNs minimal den LAI gegenüber iLand.



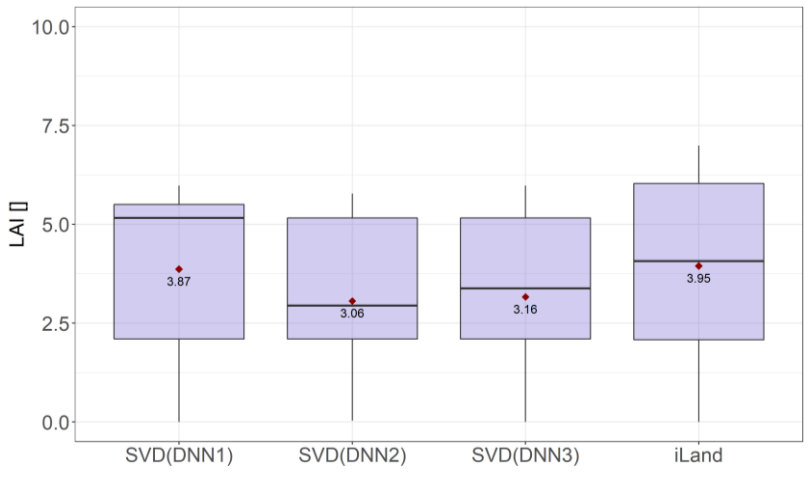
(c)


10.0
7.5
5.0
2.5
0.0
LAI []
3.69
3.5
3.73
4.29
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

(d)


10.0
7.5
5.0
2.5
0.0
LAI []
3.78
3.65
3.12
4
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

(e)


10.0
7.5
5.0
2.5
0.0
LAI []
3.87
3.06
3.16
3.95
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

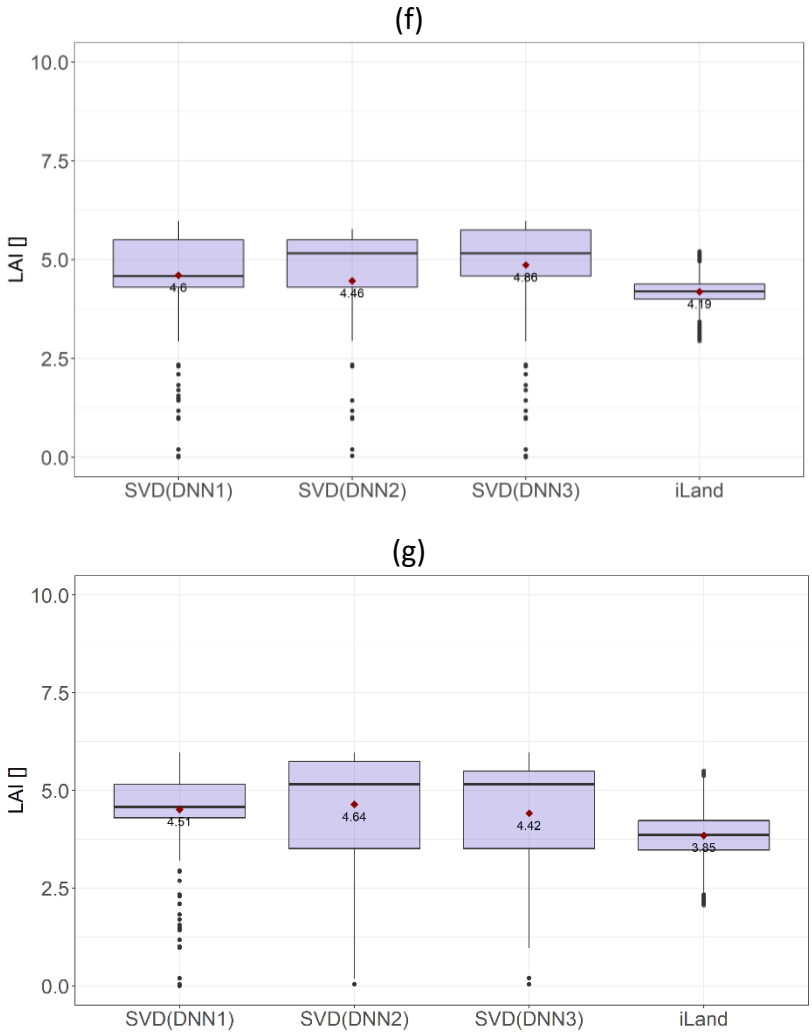


Abbildung 27: LAI in der Landschaft der einzelnen Bewirtschaftungsvarianten in der zweiten Simulationshälfte. Die Diagramme wurden gereiht nach den Bewirtschaftungsvarianten (a) REF, (b) KS100, (c) KS80, (d) SS100, (e) SS80, (f) ZSN55, (g) ZSN45. Die SVD Ergebnisse der drei DNN-Varianten werden den Ergebnissen aus iLand gegenübergestellt. Die roten Punkte geben die Mittel an. Auf der Y-Achse befindet sich der durchschnittliche LAI-Index.

4.3.2 Strukturelle Entwicklung der Landschaft

Die Übereinstimmung des Strukturverlaufs zwischen SVD und iLand-Simulationen ist nur moderat zufriedenstellend. Da Oberhöhe und Strukturklasse miteinander korrelieren, wurde im Folgenden nur die Oberhöhe analysiert.

Es kommt zu minimalen Abweichungen in der Gesamtanzahl (1200 Vegetationszustände pro Jahr) der SVD Ergebnisse, da die Netzwerke von SVD mit 45 Ausgangsklassen erstellt werden. In iLand kommen, aber insgesamt nur 41 unterschiedliche Vegetationszustände vor. Die Klassen 42, 43, 44 und 45 wurden deshalb von den SVD Ausgangsergebnissen exkludiert.

Die Bewirtschaftungsvarianten KS80, SS80 und ZSN55 wurden als repräsentative Beispiele herangezogen. Bezogen auf Abbildung 18 sind die Unterschiede zwischen diesen Varianten am höchsten und deshalb interessant zu untersuchen. Zusätzlich zu den Oberhöhenverläufen, wurden die mittleren Oberhöhen im Laufe der zweiten Simulationshälfte dargestellt und dem entsprechenden Boxplot der Oberhöhenverteilung je Bewirtschaftungsvariante gegenübergestellt. Die gleiche Darstellungsform wurde für den Leaf Area Index gewählt.

4.3.2.1 Verlauf - Oberhöhe

Wie bereits erwähnt, werden in diesem Kapitel die Oberhöhen aus den Strukturklassen abgeleitet und für jedes Simulationsjahr aufsummiert. Die drei DNN-Varianten unter SVD sollen hier mit dem entsprechenden Ergebnis aus iLand verglichen und interpretiert werden.

4.3.2.1.1 KS80

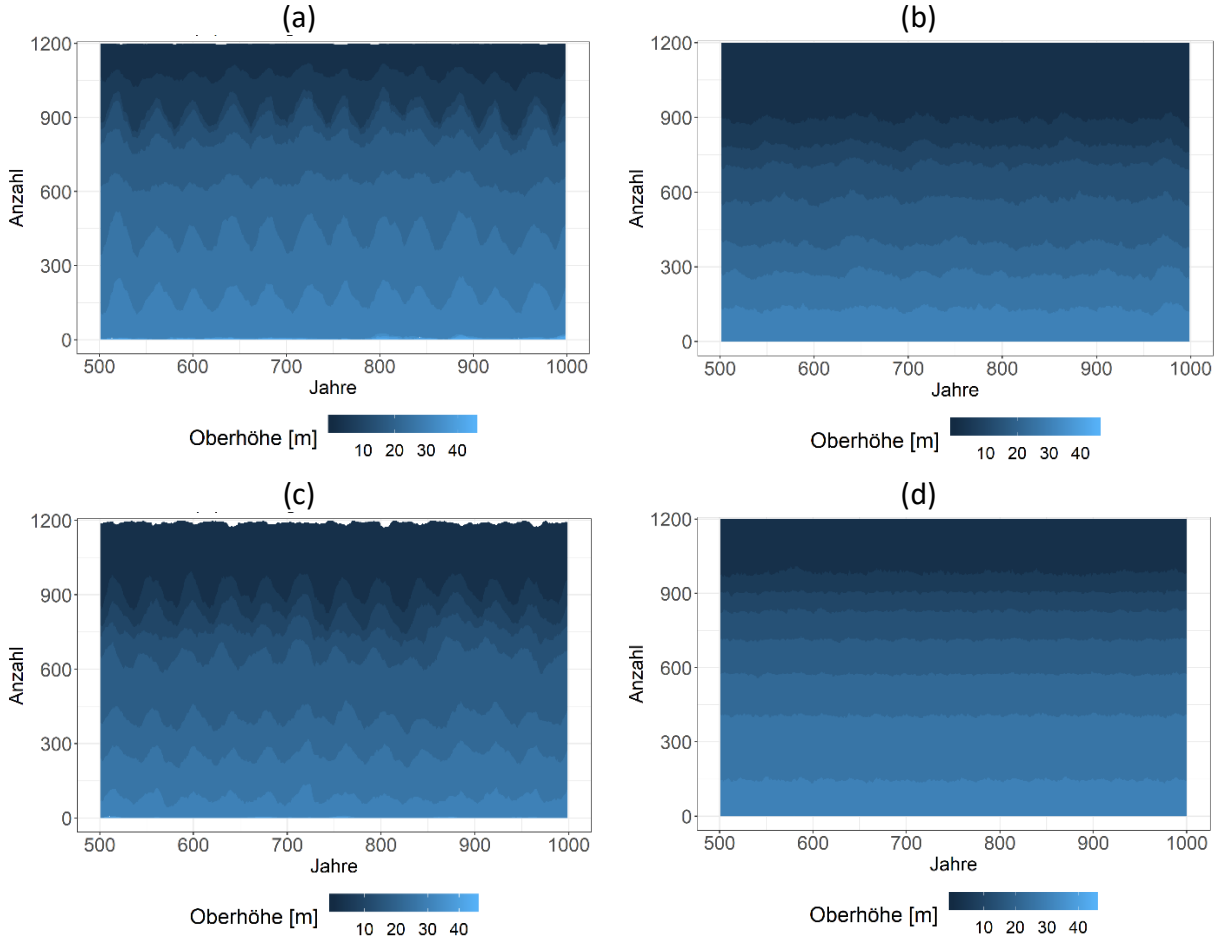


Abbildung 28: Gegenüberstellung des Oberhöhenstrukturverlaufs zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante KS80. Auf der Y-Achse befindet sich die aufsummierte Anzahl der Strukturklassen bzw. der Oberhöhen, auf der X-Achse die Simulationsjahre.

Wie in Abbildung 28 zu sehen, verlaufen die Oberhöhenverläufe in iLand sehr gleichmäßig ohne starke Dynamik. DNN2 kann dies am besten nachbilden (Abbildung 28b). Das wellenförmige Auf und Ab bei DNN1 (Abbildung 28a) und DNN3 (Abbildung 28c) geben den Eindruck von durchgeführten Eingriffen wie Durchforstungen und Abtrieb. Mit einer Genauigkeit von 80.7% sollte DNN3 die besten Übereinstimmungen zu iLand liefern. Diese Tatsache wird weniger durch den Strukturverlauf bestätigt, sondern durch das sehr gute Ergebnis beim Boxplot der Oberhöhenverteilung in Abbildung 26 oder Abbildung 29.

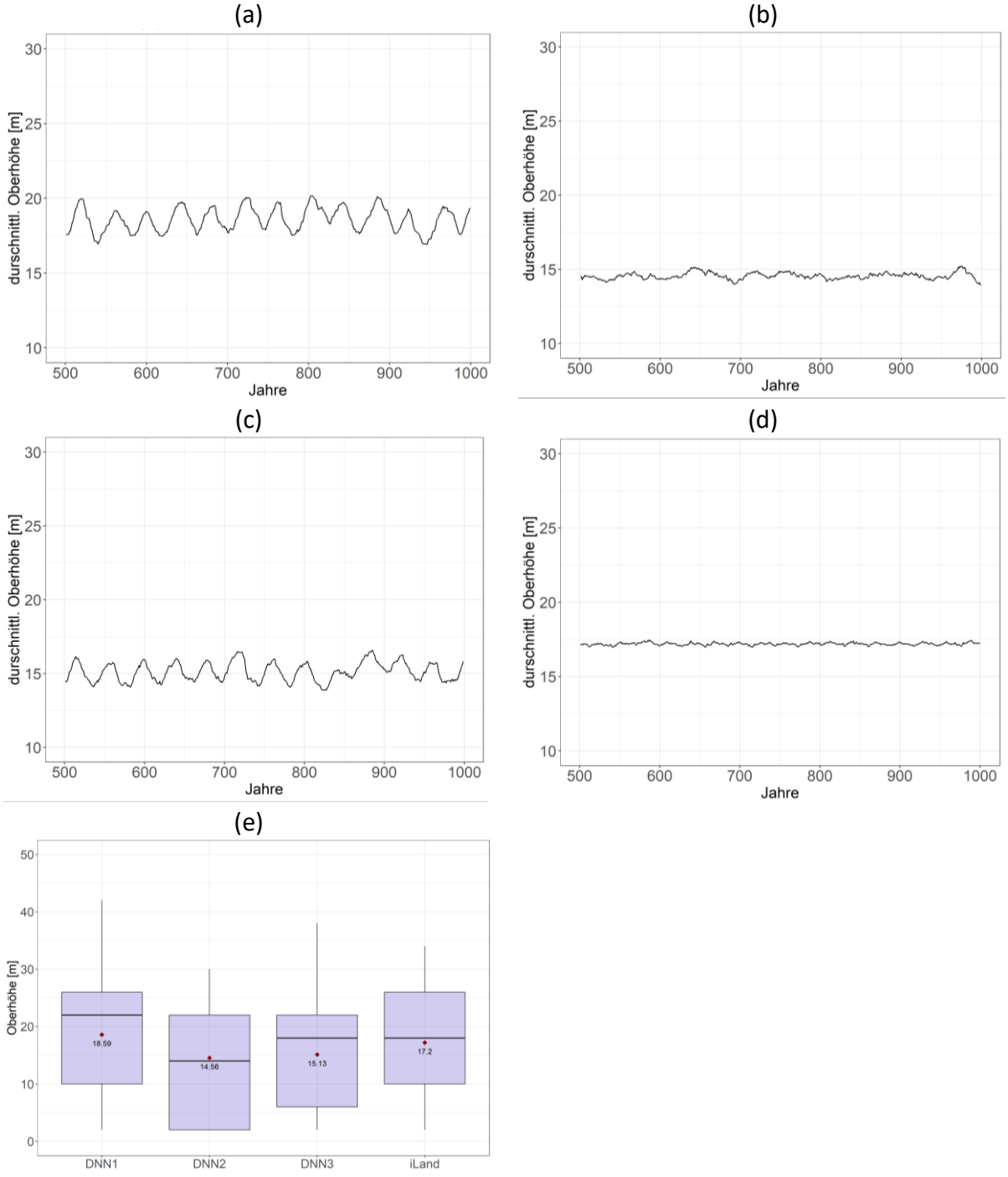


Abbildung 29: Gegenüberstellung des Verlaufs der mittleren Oberhöhe zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante KS80. Auf der Y-Achse befindet sich die durchschnittliche Oberhöhe. Das Boxplot der mittleren Oberhöhen über die gesamte Landschaft der zweiten Simulationshälfte (e) wurde zum Abgleich zwischen den DNNs und iLand ergänzt.

In Abbildung 29 sind die Schwankungen der mittleren Oberhöhe in iLand minimal und werden von DNN2 auch hier am besten nachgebildet, obwohl DNN3 mit dem Mittelwert von 15.1m am nächsten bei dem Wert von iLand (17.2m) liegt als DNN2 (14.6m). DNN1 überschätzt mit 18.6m den Sollwert von iLand.

4.3.2.1.2 SS80

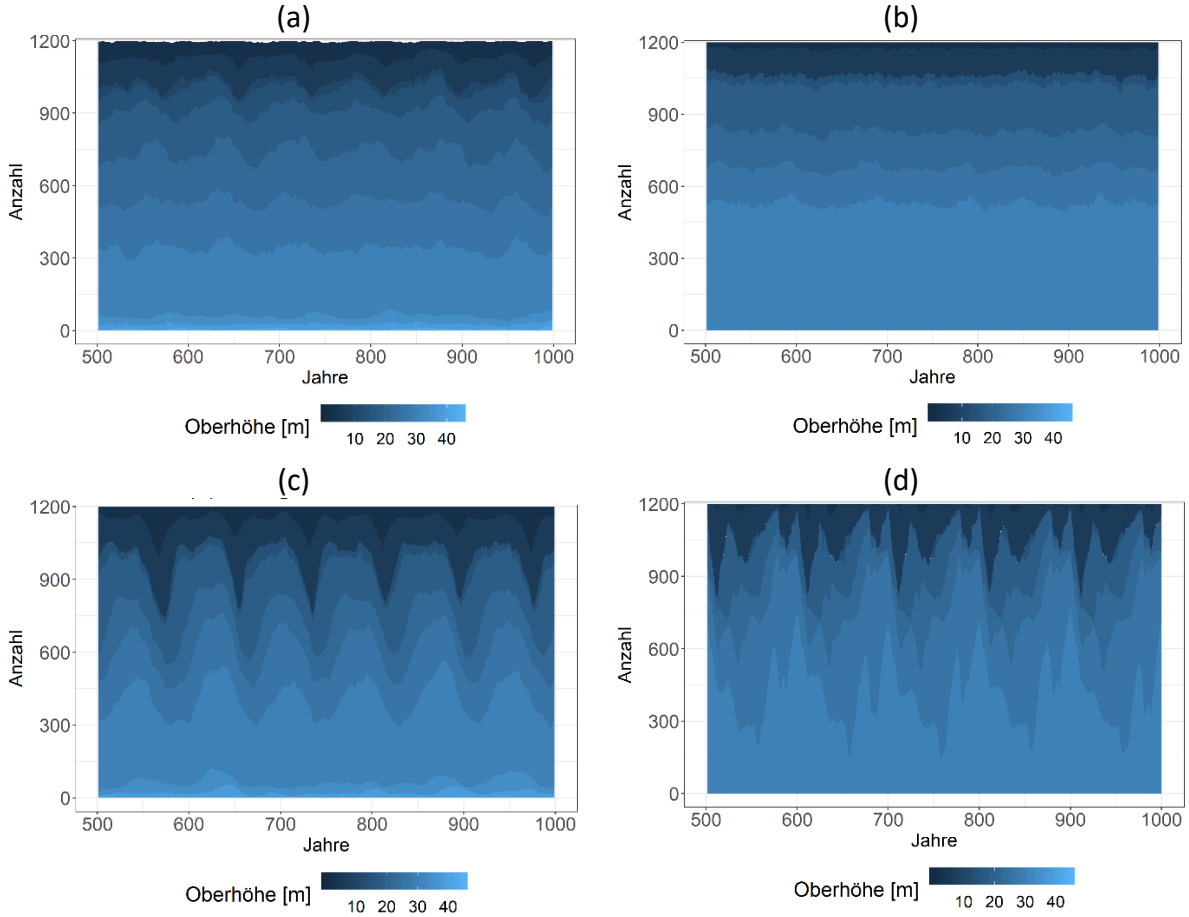


Abbildung 30: Gegenüberstellung des Oberhöhenstrukturverlaufs zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante SS80. Auf der Y-Achse befindet sich die aufsummierte Anzahl der Strukturklassen bzw. der Oberhöhe, auf der X-Achse die Simulationsjahre.

Bewirtschaftungsvariante SS80 stellte für alle DNNs (nur DNN2 schneidet bei ZSN55 noch schlechter ab) im Schnitt die größte Herausforderung dar (siehe Abbildung 30). Die Saumschläge wurden räumlich unter der Auflösung von SVD durchgeführt und waren deshalb womöglich am schwierigsten zu erlernen. Den stark dynamischen Oberhöhenverlauf von iLand bildet DNN3 (Abbildung 30c) am besten ab, ansonsten gibt es keine große Übereinstimmung zwischen DNN1, DNN2 und iLand. Abgesehen vom zeitlichen Oberhöhenverlauf, werden in Abbildung 31e die Oberhöhenmittel sehr gut von allen drei DNNs geschätzt.

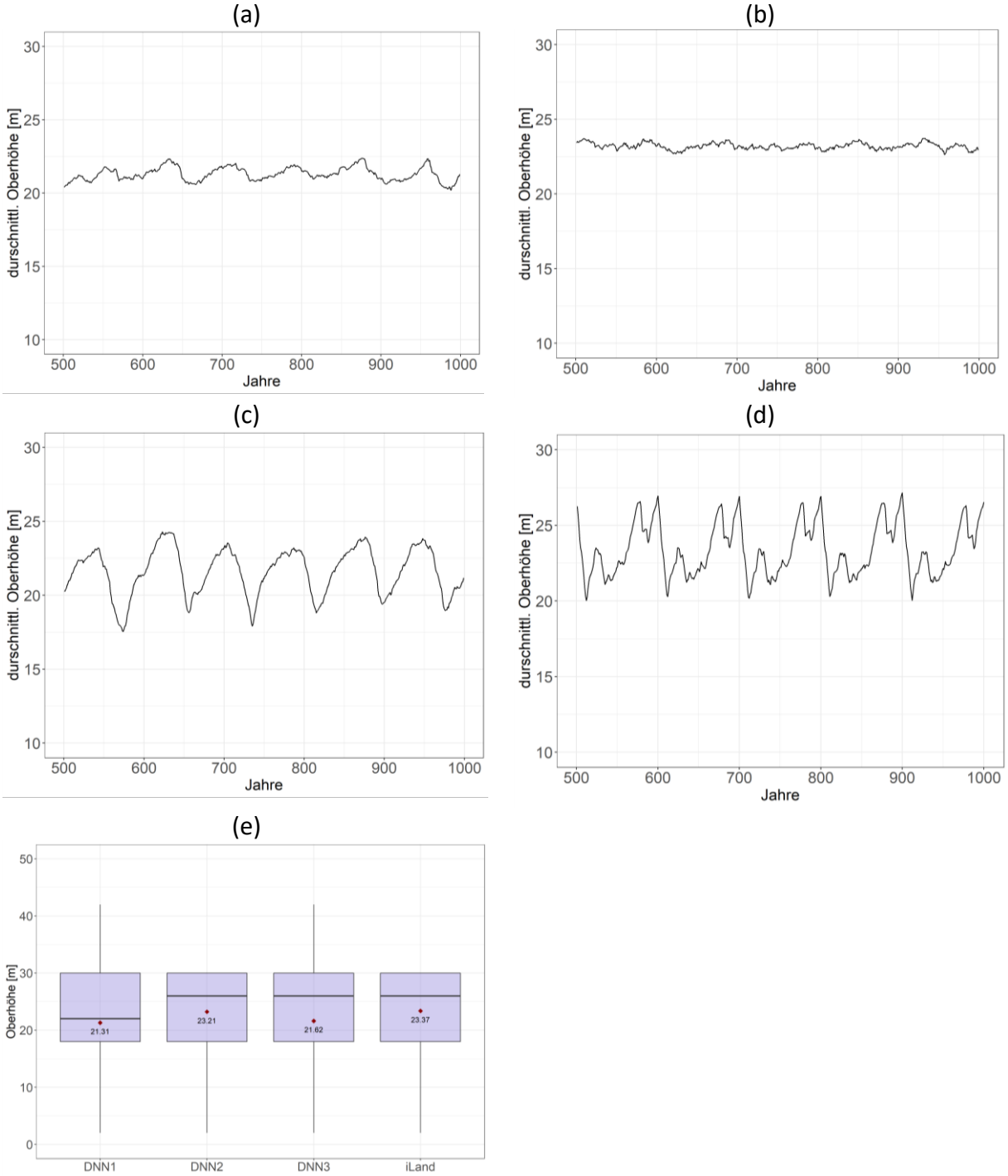


Abbildung 31: Gegenüberstellung des Verlaufs der mittleren Oberhöhe zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante SS80. Auf der Y-Achse befindet sich die durchschnittliche Oberhöhe. Das Boxplot der mittleren Oberhöhen über die gesamte Landschaft für die zweite Simulationshälfte (e) wurde zum Abgleich zwischen den DNNs und iLand ergänzt.

Wie bereits erwähnt, weichen die zeitlichen Entwicklungsverläufe von den SVD relativ stark von jenem in iLand ab, wenn man bedenkt, dass in beiden Fällen die Simulationslandschaften gleiche Ausgangsbedingungen aufwiesen. DNN3 weist eine ähnlich hohe Amplitude (ca. 15m) in der Dynamik auf wie iLand (Abbildung 31c), diese Dynamik ist jedoch bei DNN1 (a) und DNN2 (b) nicht gegeben. Die Mittelwerte sowie Mediane der DNNs

stimmen, gesehen über den gesamten Simulationszeitraum, mit denen von iLand sehr gut überein. Die gemittelte Oberhöhe von iLand (23.4m) wird von DNN2 auf 23.2m geschätzt, gefolgt von DNN3 mit 21.6m und DNN1 mit 21.3m.

4.3.2.1.3 ZSN55

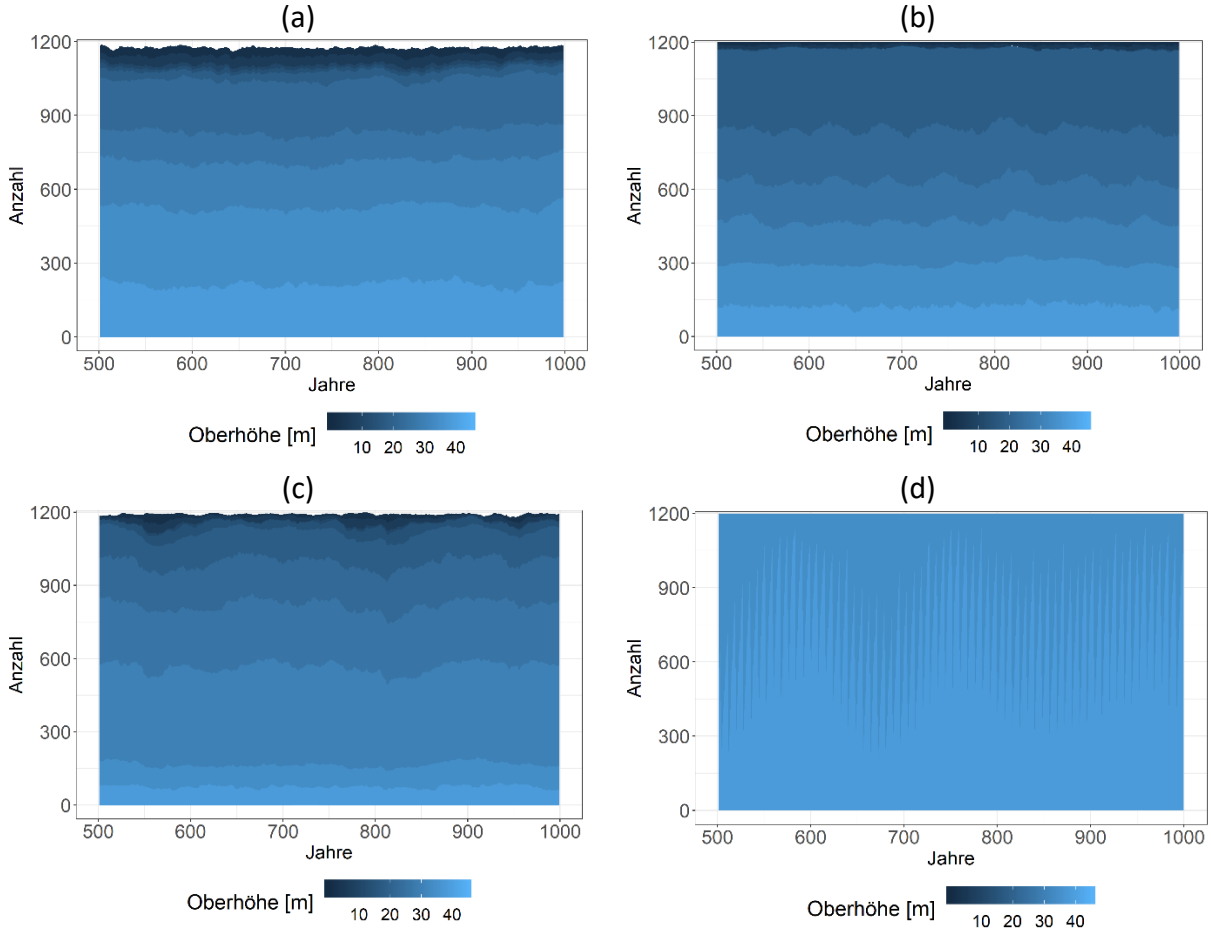


Abbildung 32: Gegenüberstellung des Oberhöhenstrukturverlaufs zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante ZSN55. Auf der Y-Achse befindet sich die aufsummierte Anzahl der Strukturklassen bzw. der Oberhöhe, auf der X-Achse die Simulationsjahre.

ZSN55 zählt neben REF zu der Variante bei der die DNNs den Oberhöhenverlauf sowie in Folge die gemittelte Oberhöhe zu iLand (Abbildung 33e) stark unterschätzen. In Abbildung 33d lässt sich beim Verlauf von iLand in 8-jährigem Abstand die Entnahme ab 55cm BHD auch gut im Oberhöhenverlauf erkennen, welcher sich in der zweiten Simulationshälfte bereits eingependelt hat. Die Oberhöhenverläufe der DNNs unterscheiden sich nicht deutlich voneinander.

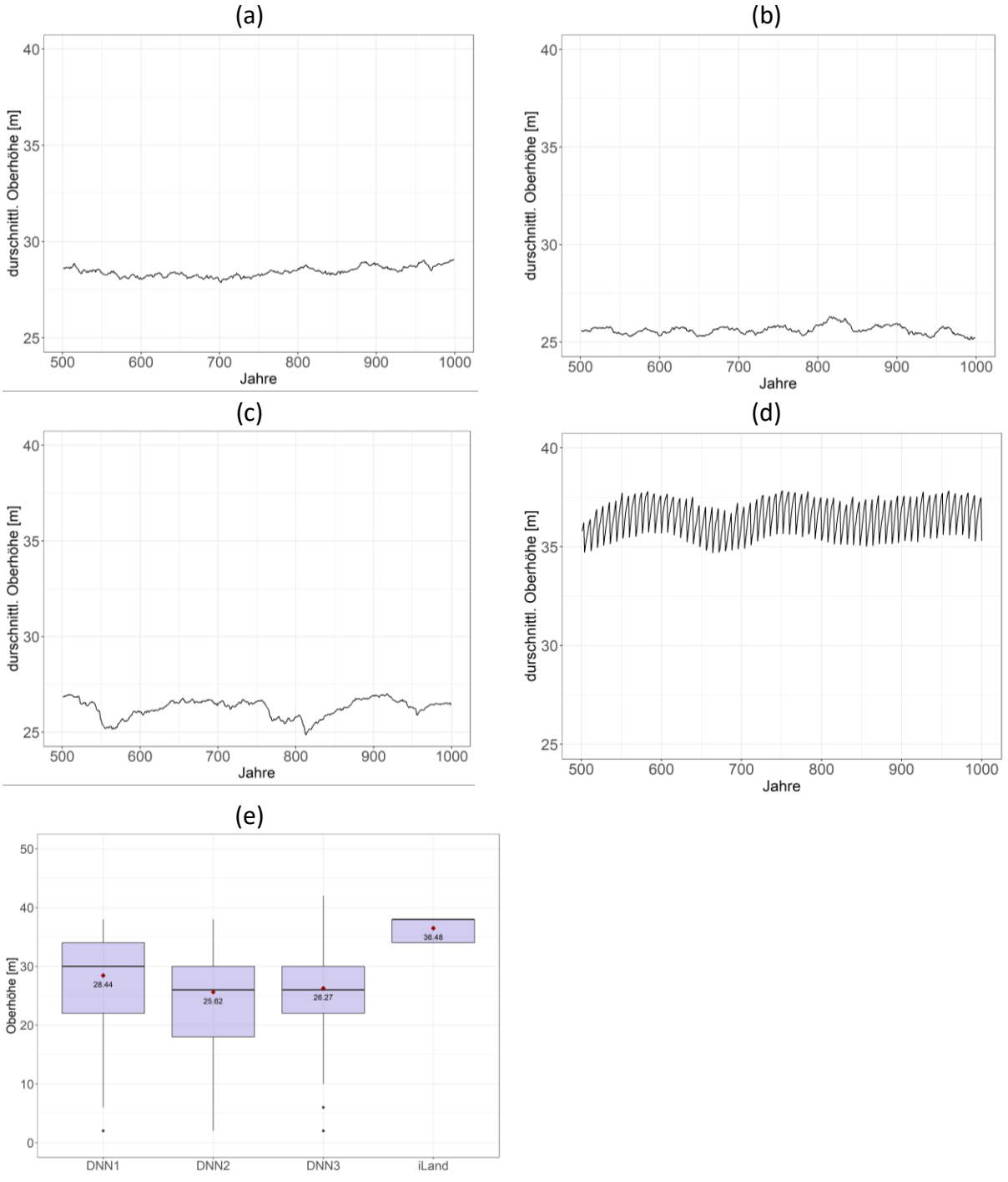


Abbildung 33: Gegenüberstellung des Verlaufs der mittleren Oberhöhe zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante ZSN55. Auf der Y-Achse befindet sich die durchschnittliche Oberhöhe. Das Boxplot der mittleren Oberhöhen über die gesamte Landschaft für die zweite Simulationshälfte (e) wurde zum Abgleich zwischen den DNNs und iLand ergänzt.

Wie auch bei der Referenzvariante REF werden die mittleren Oberhöhen von allen DNN-Varianten in SVD unterschätzt (siehe Abbildung 33e). DNN1 ist mit einem Mittelwert von 28.4m am nächstem bei dem Ergebnis von iLand mit 36.5. Die größte Abweichung weisen DNN3 mit 26.3m und DNN2 25.6m auf. DNN1 erhält als Eingangsinformation die Eingriffsart

sowie Eingriffszeitpunkt, welche zum Erlernen einer Bewirtschaftungsvariante wie ZSN55 essentiell sind.

4.3.2.2 Verlauf – Leaf Area Index

Wie in Kapitel 4.2.3 bereits erwähnt wurde, zeigen die Abweichungen der Schätzungen der Funktionsklassen (siehe Abbildung 23), dass die meisten Fehlvorhersagen eine Funktionsklasse unter oder über dem Sollwert von iLand liegen. Die Funktionsklassen des Leaf Area Index sind in LAI-Intervalle von 0-2, 2-4 und >4 relativ grob unterteilt. Deshalb wurde der Vergleich zwischen SVD und iLand nicht auf Basis der Funktionsklasse jedes Vegetationszustandes, sondern über die gemittelten Werte aus den forstlichen Parametern erstellt. Bezogen auf die Mittelwerte des LAI über den gesamten Simulationszeitraum, liegen die Abweichungen der drei Bewirtschaftungsvarianten KS80, SS80 und ZSN55 alle innerhalb einer Funktionsklasse. Genauer betrachtet, gibt es dennoch Unterschiede im Verlauf des LAI-Index der SVD Simulationen über die Zeit, verglichen zu iLand.

Der LAI-Index-Verlauf lässt auch die Eingriffsintervalle der unterschiedlichen Bewirtschaftungsvarianten in iLand gut nachvollziehen (siehe Abbildung 34 - Abbildung 36).

4.3.2.2.1 KS80

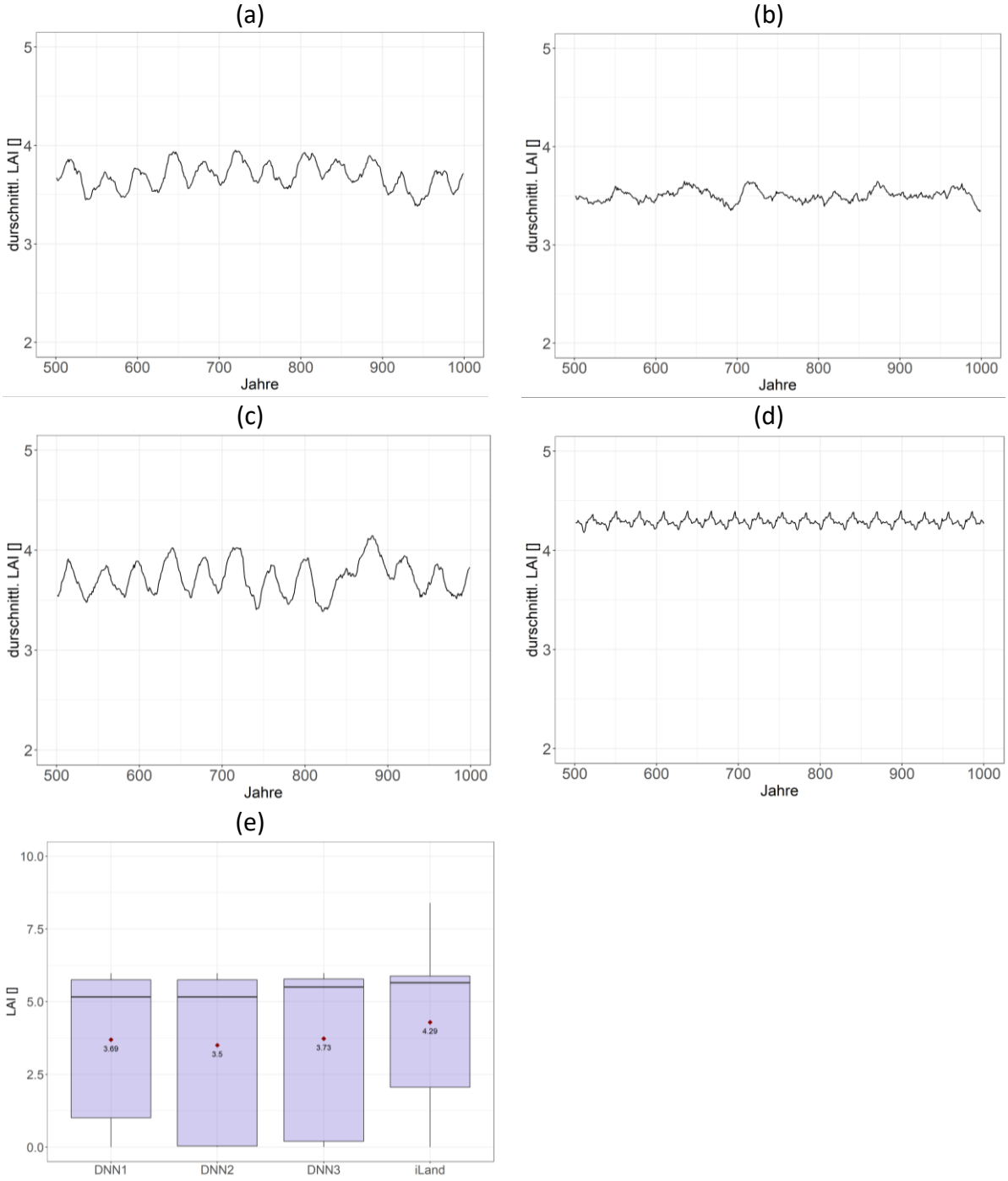


Abbildung 34: Gegenüberstellung des Verlaufs des mittleren LAI-Index zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante KS80. Auf der Y-Achse befindet sich der durchschnittliche LAI-Index, auf der X-Achse die Simulationsjahre. Das Boxplot des LAI über die gesamte Landschaft der zweiten Simulationshälfte (e) wurde zum Abgleich ergänzt.

Tendenziell wird der Soll-LAI (4.3) von allen SVD-Simulationen unterschätzt (siehe Abbildung 34e). DNN2 zeigt die größte Abweichung mit einem LAI von 3.5, gefolgt von DNN1 mit 3.7 und DNN3 mit einem mittleren LAI von 3.7. Eine ähnliche Dynamik im Verlauf wird mit einer

größeren Amplitude von DNN3 und DNN1 nachgebildet. Grundsätzlich sind die Schätzungen hinsichtlich des gemittelten Soll-LAI zufriedenstellend.

4.3.2.2.2 SS80

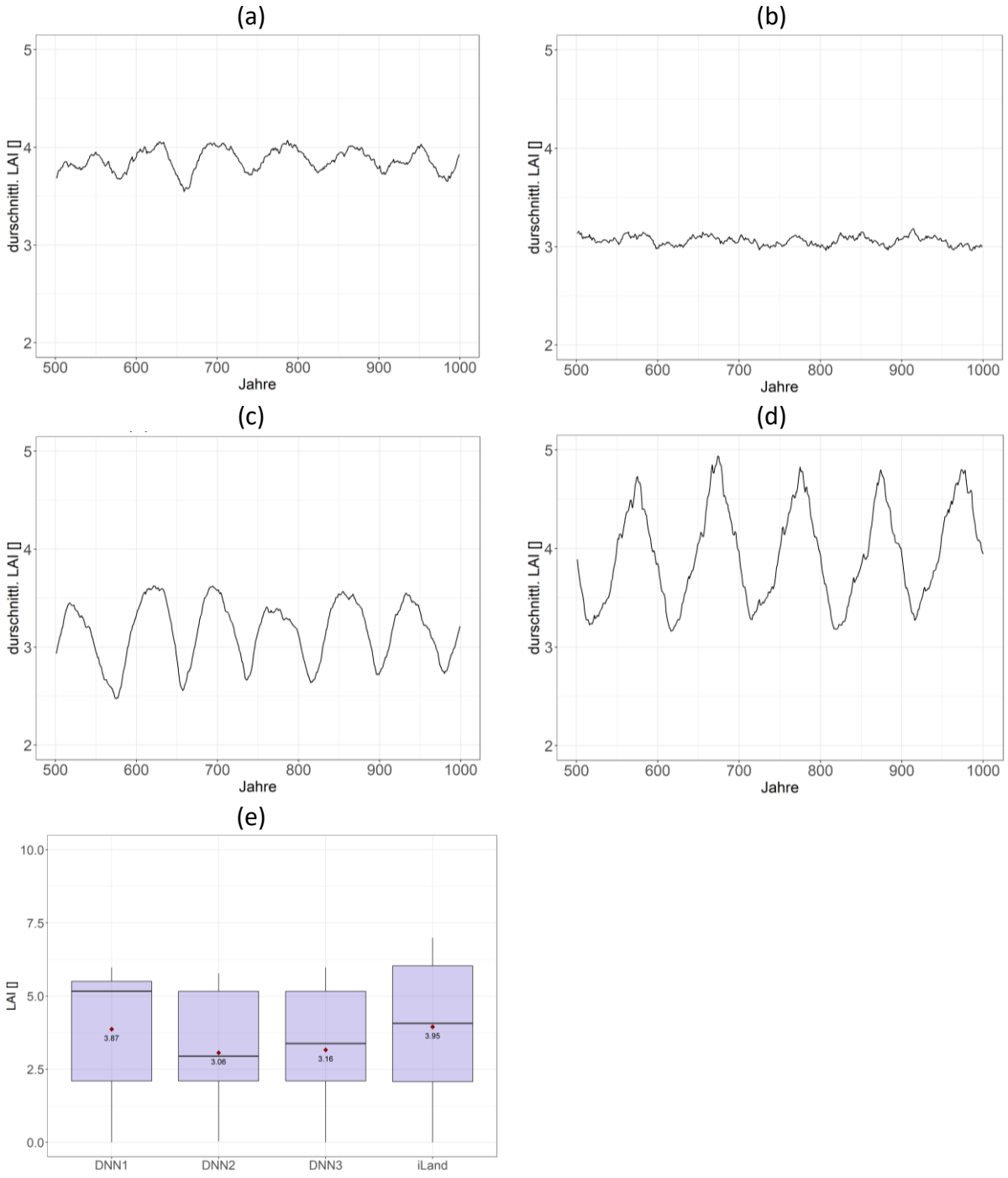


Abbildung 35: Gegenüberstellung des Verlaufs des mittleren LAI-Index zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante SS80. Auf der Y-Achse befindet sich der durchschnittliche LAI-Index, auf der X-Achse die Simulationsjahre. Das Boxplot des LAI über die gesamte Landschaft der zweiten Simulationshälfte (e) wurde zum Abgleich ergänzt.

Auch bei SS80 (siehe Abbildung 35e) wird der Soll-LAI (4) tendenziell von allen SVD-Simulationen unterschätzt. Wieder zeigt DNN2 die größte Abweichung mit einem LAI von

3.1, gefolgt von DNN1 mit 3.2. DNN1 liefert das beste Ergebnis für den gemittelten LAI-Index mit 3.9, aber kann den Verlauf weit schlechter nachbilden als DNN3 (Abbildung 35c). Der Verlauf wird am besten von DNN3, gefolgt von DNN1 (Abbildung 35a) nachgebildet, jedoch mit geringerer Amplitude. Grundsätzlich sind die Schätzungen hinsichtlich des gemittelten Soll-LAI zufriedenstellend.

4.3.2.2.3 ZSN55

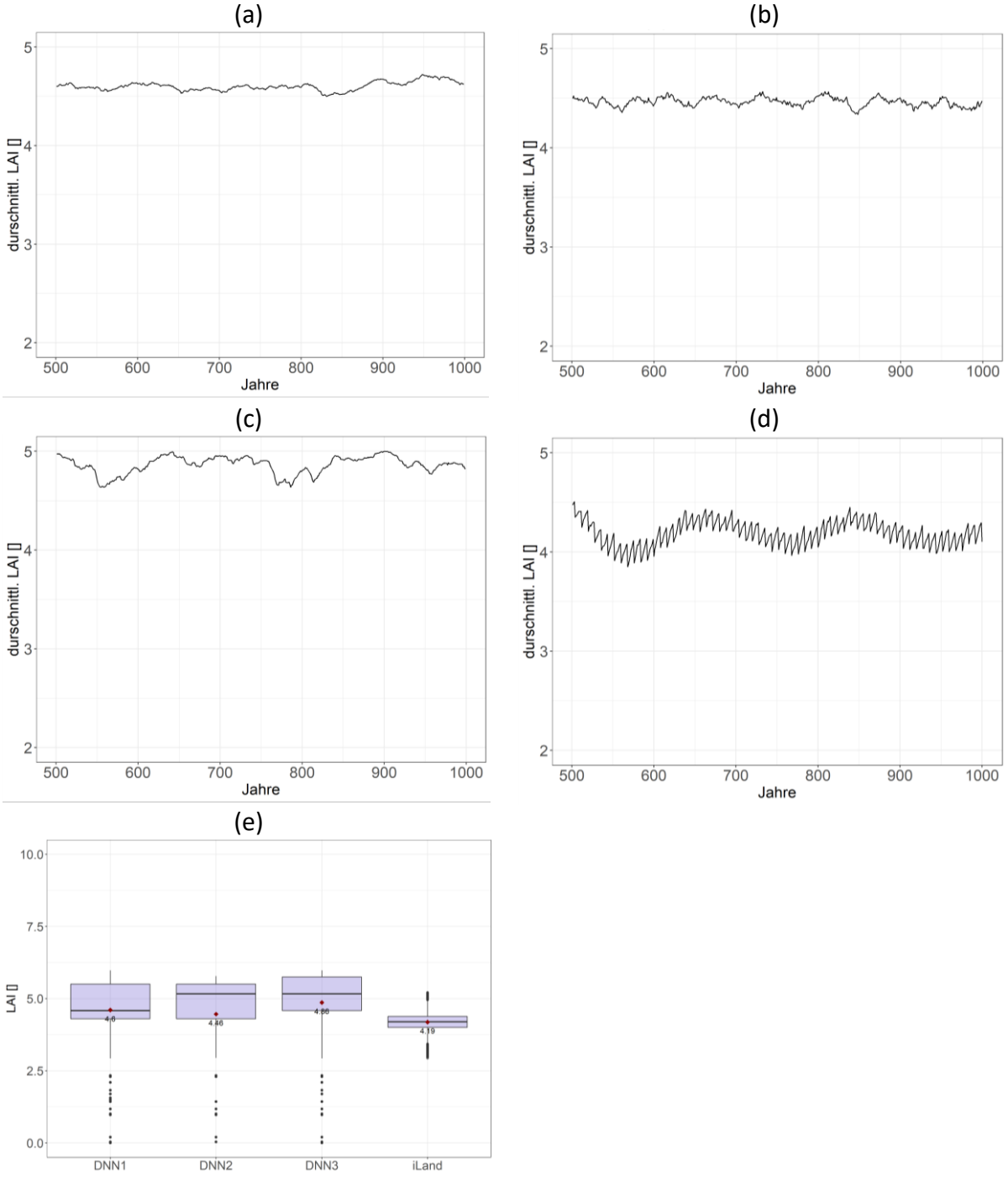


Abbildung 36: Gegenüberstellung des Verlaufs des mittleren LAI-Index zwischen DNN1-DNN3 (a)-(c) in SVD und iLand (d) der Bewirtschaftungsvariante ZSN55. Auf der Y-Achse befindet sich der durchschnittliche LAI-Index, auf der X-Achse die Simulationsjahre. Das Boxplot des LAI über die gesamte Landschaft der zweiten Simulationshälfte (e) wurde zum Abgleich ergänzt.

Im Unterschied zu KS80 und SS80 überschätzen die SVD-Simulationen von ZSN55 in Abbildung 36e das iLand Ergebnis durchgehend mit der größten Abweichung von DNN3 mit einem LAI von 4.9. Der Verlauf in iLand lässt Nutzungen in einem 8 Jahresintervall in einer relativ kleinen LAI Amplitude erkennen (Abbildung 36d), der Mittelwert liegt bei 4.2. Das

Mittel von DNN2 liegt bei einem LAI von 4.5, wobei DNN1 die beste Annäherung mit einem LAI von 4.6 erreicht. Die Verläufe gleichen dem von iLand nicht, wobei die gemittelten LAI-Indizes im Bereich einer Funktionsklasse liegen.

5 DISKUSSION

5.1 iLAND (WALDSIMULATION)

Die Kahlschlag-, Saumschlag- und Zielstärkenutzungen wurden jeweils als Paare mit einer/m längeren und kürzeren Umtriebszeit/Eingriffsintervall ausgeführt, um eine bessere Vergleichbarkeit unter den Varianten zu erhalten. Die Bewirtschaftungsvarianten unterscheiden sich in der Umtriebszeit, Verjüngungsform und den ausgeführten Eingriffen. Agenten-basierten Modelle dienen dazu, Wechselwirkungen von menschlichen und natürlichen System zu simulieren und besser zu verstehen (Leahy *et al.*, 2013; Rammer und Seidl, 2015; Albrich, 2016). In den späteren Analysen wurden die Bewirtschaftungsvarianten hauptsächlich auf Strukturvariablen, wie Oberhöhe oder LAI untersucht. Für die Waldsimulationen wurden alle Bewirtschaftungsvarianten gegenübergestellt und Variablen wie Volumen, Oberhöhe, Mittelhöhe, Grundfläche, Stammzahl, BHD und dGZ auf waldbauliche Plausibilität beurteilt. In Kapitel 4.1.1 bestätigen die Analysen einzelner Zellen der Bewirtschaftungsvarianten ZSN55 (Abbildung 8c-d) und KS100 (Abbildung 9c-d) die Eignung von iLand für Waldbewirtschaftungssimulationen. Einzelstammentnahmen, Dickungspflege, Durchforstungen oder Kahlschläge können gut nachvollzogen werden. Die Anwendbarkeit von iLand wurde schon mehrmals bestätigt (Seidl *et al.*, 2012a; Seidl *et al.*, 2012b; Silva Pedro *et al*., 2015; Albrich, 2016; Thom *et al.*, 2017) und die eingesetzten Bewirtschaftungsvarianten wurden, wenn möglich, jenen von Albrich (2016) und Seidl *et al.* (2007) angepasst. Es gibt in der Modellierungsliteratur unterschiedliche Angaben für Eingriffsstärke sowie Eingriffsintervall für Zielstärkenutzungen (Seidl *et al.*, 2007; Pelyukh *et al.*, 2018). Der Ziel-BHD wurde mit 55cm und 45cm fixiert und nicht wie bei Seidl *et al.* (2007) über relative Durchmesserklassen festgelegt. Es wurde aber automatisch auch in die höchsten BHD-Klassen eingegriffen. Um eine zu intensive Nutzung bei relativ kurzen Eingriffsintervallen zu unterbinden, wurde deshalb eine Entnahmegrenze von 20% des Gesamtbestandvolumens festgelegt, ähnliche Annahmen trafen auch Hanewinkel und Pretzsch (2000). Grundsätzlich konnte iLand die Bewirtschaftungsunterschiede gut abbilden (Abbildung 10a-f und Abbildung 13a-f) und lieferte daher eine gute Grundlage für die Trainings der DNNs. Eine umfangreichere Ausführung der STPs in ABE könnte zu einer besseren Differenzierung der Ergebnisse der einzelnen Bewirtschaftungsvarianten führen.

5.2 DNNs

Um Forschungsfrage (i), ob Waldbewirtschaftung mit Hilfe von DNNs abstrahiert und in Computersimulationen abgebildet werden kann und deren Performance gegenüber detaillierten Managementmodellen zu bewerten, beantworten zu können, wurden drei DNNs mit unterschiedlichem Informationsgrad entwickelt und mit den Daten aus den iLand-Simulationen trainiert. Wie Abbildung 18 und Abbildung 20 zeigen, konnten die DNNs erfolgreich Waldbewirtschaftung abstrahieren und zeigen die relative Performance der DNNs gegenüber dem PBM iLand. Die höchsten allgemeinen Genauigkeiten wurden bei der abhängigen Variable *nächster Vegetationszustand S** erreicht - (1) [72.1%], (2) [70.8%] und (3) [77.5%]. Die allgemeinen Genauigkeiten der abhängigen Variable *Verweilzeit ΔR* waren wie folgt – (1) [60.2%], (2) [59.6%] und (3) [64.5%]. Eine Unterscheidung dieser Performance-Metriken je nach Bewirtschaftungsvariante macht die Stärken und Schwächen der drei einzelnen DNNs sichtbar. Insbesondere DNN2 bleibt bei ZSN55 und ZSN45 in den Genauigkeiten für *S** und *ΔR* deutlich hinter DNN1 und DNN2, mit den größten Differenzen bei ZSN55 (Abbildung 18 und Abbildung 20). Diese Tatsache ist sehr wahrscheinlich auf das sehr kurze Zeitintervall von ZSN55 mit 5 Jahren zurückzuführen. DNN2 bekommt beim Training als einziges Netzwerk keine Information über den Zeitpunkt des Managementeingriffs und schneidet deshalb bei der managementintensiven Bewirtschaftungsvariante ZSN55 [ΔR: 46.1% zu (1) 53.4% und (3) 55.6%] am schlechtesten ab, gefolgt von ZSN45 [ΔR: 65.5% zu (1) 67.5% und (3) 72.6%]. Das gilt auch für den nächsten Vegetationszustand bei ZSN55 [S*: 59.5% zu (1) 73.3% und (3) 75.1%] und ZSN45 [S*: 68.8% zu (1) 73.5% und (3) 77.7%]. Über beide abhängige Variablen und alle Bewirtschaftungsvarianten hinweggesehen, inklusive der Referenzvariante, schnitt DNN3 am besten bei den Genauigkeiten und anderen Netzwerkmetriken ab (Abbildung 16 und Abbildung 17). Die einzige Ausnahme bildet die Referenzvariante bei der Vorhersage für S*, hier erreicht DNN2 eine höhere Genauigkeit mit einer Differenz von 1% gegenüber DNN3.

Ein Punkt, welcher zu einer geringeren Genauigkeit in der Statistik führen könnte, sind die drei verschiedenen Funktionsklassen eines Vegetationszustandes welche in Tabelle 9 rot markiert wurden. Der Bestand kann in der gleichen Strukturklasse mit entsprechender Oberhöhe liegen, unterscheidet sich aber durch den Überschirmungsgrad, also die LAI-Klassen. Sagt ein Netzwerk eine falsche Funktionsklasse, also einen falschen Überschirmungsgrad vorher, aber die richtige Strukturklasse ergibt dies trotzdem ein

negatives Ergebnis für die Genauigkeitsstatistik. Tatsächlich wird der Hauptteil des Vegetationszustandes aber richtig beschrieben. Das heißt eine falsche Vorhersage muss nicht unbedingt bedeuten, dass das Netzwerk komplett daneben lag, sondern einen verwandten oder benachbarten Vegetationszustand geschätzt hat. Dies bestätigen auch Abbildung 22 und Abbildung 23, wo die Oberhöhen-Abweichungen und die LAI-Abweichungen der DNNs mit den erwarteten Werten von iLand verglichen wurden. Die meisten Fehlschätzungen liegen eine Oberhöhen- oder LAI-Klasse ober- oder unterhalb des Sollwertes. Die Fehlschätzungen für die Verweilzeit ΔR (Abbildung 24) zeigen eine stärkere Verteilung unter- und oberhalb des Sollwertes. Die DNNs haben gelernt, dass sich ein Vegetationszustand in den meisten Fällen nicht innerhalb des 10-jährigen Vorhersagehorizontes ändert (Abbildung 25). Es ist also am sichersten für ein DNN diesen Wert zu schätzen, was aber nicht bedeutet, dass das immer richtig ist.

Die Ergebnisse bestätigen die grundsätzliche Annahme bei *deep learning* Anwendungen, dass ein höherer Informationsgrad DNNs bessere Vorhersagen ermöglicht. Dies war schlussendlich auch der Beweggrund, drei verschiedene DNNs mit unterschiedlichem Informationsgrad zu erstellen und gegeneinander zu testen. DNN3 erhält als Managementinformation die Eingriffsart, den Eingriffszeitpunkt sowie die laufende Bewirtschaftungsvariante. Dadurch konnte die Performance deutlich gesteigert werden und erzielte die besten Ergebnisse für die Validierungsdaten bzw. die geringsten Verluste (Abbildung 17). Es kommt bei keinen der drei DNN-Varianten zu einer Überanpassung und sie generalisieren gut. Die Performance von DNN1 liegt, abgesehen von ZSN55 und ZSN45, unter jener von DNN2, insgesamt gibt es aber keine signifikanten Unterschiede zwischen der Simulation einer Abfolge von expliziten Managementeingriffen und der Simulation ganzer Bewirtschaftungsvarianten, was Forschungsfrage (ii) beantwortet. Die nur in den Einzelfällen von ZSN55 und ZSN45 bessere Leistung von DNN1 gegenüber DNN2 würde bedeuten, dass DNN2 mit einer geringeren Informationsdichte über waldbauliche Eingriffe, nämlich nur das Managementregime, vergleichbar gute Ergebnisse liefert wie DNN1, wo jeder Einzeleingriff als Information Eingang in das Netzwerktraining findet. Ein geringerer Informationsinput für vergleichbar gute Vorhersagen kann also durchaus als Vorteil von DNN2 gesehen werden, wenn dies auch nur für bestimmte Arten von Waldbewirtschaftungen zutrifft. Das scheint in gewissem Widerspruch damit zu stehen, dass mehr Information bessere Netzwerkergebnisse liefert, wie die Analyse von DNN3 offensichtlich beweist. Bei Bewirtschaftungsvarianten mit

hohen Eingriffsfrequenzen (ZSN55 und ZSN45) profitieren die Netzwerke wie DNN1 und DNN3 aber deutlich von der höheren Informationsdichte der Einzeleingriffe gegenüber DNN2, dies gilt für beide abhängigen Variablen S^* und ΔR.

5.3 SVD

Forschungsfrage (iii) befasst sich mit dem Thema, ob SVD strukturelle Unterschiede, welche durch Bewirtschaftungsunterschiede bedingt sind, darstellen kann und diese Ergebnisse und Strukturen mit Ergebnissen des PBM iLand vergleichbar sind. Für eine Differenzierung der strukturellen Unterschiede von Bewirtschaftung zwischen SVD und iLand Simulationen wurde zum einen die Oberhöhe und zum anderen der LAI als repräsentative Variablen gewählt. Die Höhe des Kronendachs ist ein guter Indikator für die strukturelle Entwicklungsphase eines Waldbestandes und kann über die Oberhöhe abgeleitet werden (Rammer und Seidl, 2019a). Zusätzlich sind Primärproduktion, Licht Absorption und Transpiration an die Blattorgane eines Baumes gekoppelt (Rammer und Seidl, 2019a), weshalb der LAI als weitere determinierende Variable verwendet wurde. Gemeinsam mit der Baumartenzusammensetzung werden über diese drei Variablen die diskreten Vegetationszustände in SVD gebildet und erklären somit die Wahl von Oberhöhe und LAI für den Vergleich zwischen SVD und iLand. Allein die Baumartenzusammensetzung oder -komposition wurde für Analysen vernachlässigt, weil nur eine Baumart simuliert wurde.

Die zeitlichen Entwicklungen der Oberhöhe und des LAI je Bewirtschaftungsvariante und Simulation wurden gesamtlandschaftlich betrachtet, d. h. die durchschnittliche Entwicklung über alle 1200 Zellen der Testlandschaft. Es konnte bestätigt werden, dass SVD vergleichbare quantitative Aussagen zu iLand nachbilden kann. Die größten Abweichungen traten nicht zwischen den SVD-Simulationen auf, sondern zwischen SVD und iLand, genauer bei einzelnen Bewirtschaftungsvarianten wie REF und ZSN55 (Abbildung 26a und Abbildung 26f). In der Referenzvariante erfolgte kein forstlicher Eingriff, es konnten jedoch durchgehend (Abbildung 26, Abbildung 27 und Anhang 10.3-10.9 exklusive dGZ) die größten Abweichungen zwischen SVD und iLand beobachtet werden. Die in SVD implementierten Netzwerke, welche in erster Linie auf Waldbewirtschaftung trainiert waren, hatten offensichtlich Probleme „no-management“ zu simulieren. Dies widerspricht den Ergebnissen aus Kapitel 4.2.3, wo Abbildung 18 und Abbildung 20 die höchsten Genauigkeiten aller drei DNNs für beide abhängige Variablen S* und ΔR für die Referenzvariante darstellen. Waldlandschaften ohne intensive Managementeingriffe (REF, ZSN55 und ZSN45) stellen die in SVD implementierten DNNs offenbar vor eine Herausforderung, da sich die Abweichungen in Abbildung 26 und Abbildung 27 ähneln.

Die kleinsten von SVD unterstützten räumlichen Simulationseinheiten betragen einen Hektar. Beide Bewirtschaftungsvarianten SS100 und SS80 (Saumschlag-Verjüngungsform) arbeiteten in ABE unter dieser räumlichen Größenordnung. Das bedeutete, auf einem Hektar Wald wurden im Laufe einer Umtriebszeit drei Saumschläge durchgeführt. Es kommen auf einem Hektar also drei verschiedene Entwicklungsphasen eines Bestandes, mit entsprechend starken strukturellen Unterschiede vor, was auch in den iLand Simulationen beobachtet werden konnte. Diese Strukturunterschiede sind schwer in einen eindeutigen Vegetationszustand zu transformieren, weil der Vegetationszustand den Zustand des gesamten Hektars beschreibt und nicht Teile davon. Somit ist es für ein DNN auch schwieriger diese Bewirtschaftungsvarianten gut zu erlernen. Bei SS80 haben die DNNs in SVD Probleme die starke Dynamik des Oberhöhenstrukturverlaufs (Abbildung 30), der mittleren Oberhöhe (Abbildung 31) und des mittleren LAI-Index (Abbildung 35) von iLand nachzubilden.

Obwohl die SVD-Simulationen die strukturellen Entwicklungen von iLand nur moderat nachbilden konnten, so stimmen doch die Gesamtergebnisse gut überein. Für eine Analyse einer exakten Strukturentwicklung zwischen SVD und iLand, müssten Zellen mit der gleichen ID gewählt werden und deren Verläufe oder Ganglinien verglichen werden. Die SVD-Ergebnisse sind jedoch die Verteilung der Vegetationszustände pro Simulationsjahr und lassen in diesem Fall nur gesamtlandschaftliche Aussagen zu. Analysen zu forstlichen Variablen wie Volumen, Mittelhöhe, durchschnittlicher Gesamtzuwachs, Kohlenstoffvorrat, Grundfläche, Stammzahl und BHD wurden nicht näher behandelt, entsprechende Boxplot-Diagramme befinden sich jedoch im Kapitel 10 Anhang. Die guten Ergebnisse der DNN Trainings und Evaluationen aus Kapitel 4.2 stehen nur moderaten Ergebnissen der SVD Simulationen und Evaluationen aus Kapitel 4.3 gegenüber. Daraus lässt sich schlussfolgern, dass allein aus dem erfolgreichen Netzwerktraining noch nicht geschlossen werden kann, wie sich ein implementiertes DNN im Meta-Modell SVD verhält. Gute Übereinstimmungen gesamtlandschaftlicher Ergebnisse zwischen SVD und iLand setzen vergleichsweise gute Übereinstimmung struktureller Entwicklungen zwischen SVD und iLand nicht unbedingt voraus. Die anfänglich ähnliche Entwicklung kann im Laufe des Simulationszeitraumes in eine gänzlich andere Entwicklungsphase abdriften. Eine spannende Frage ist deshalb, wie lange eine Waldentwicklung deckungsgleich verläuft und ab welchem Zeitpunkt es zu Abweichungen kommt.

6 CONCLUSIO

Rammer und Seidl (2019a) erreichten durch die Anwendung des DL-Ansatzes in SVD für verschiedene Klimaszenarien Genauigkeiten der Schätzungen von 86.3% für die Verweilzeit und 85.7% für den Vegetationszustand. Im Vergleich dazu konnten in dieser Arbeit die Genauigkeiten der Schätzungen von 64.5% für die Verweilzeit und 77.5% für den Vegetationszustand für die Simulationen von Waldbewirtschaftung erreicht werden. Die Ergebnisse dieser Arbeit lassen die Schlussfolgerung zu, dass Waldbewirtschaftungen schwieriger zu simulieren sind als beispielsweise Klimaeffekte. Möglicherweise sind größer skalierte Effekte, wie das Klima, einfacher für das Meta-Modell SVD zu erlernen als feiner skalierte Effekte, wie Waldbewirtschaftung. Quantitative Ergebnisse von iLand konnten in SVD gut repliziert werden, jedoch wiesen die strukturellen Ergebnisse und Entwicklungen zwischen den einzelnen DNN-Varianten aus SVD und iLand nur geringe Übereinstimmung auf. Selbst nach einer intensiven Einarbeitungsphase in die Technologie von Künstlicher Intelligenz, *machine learning* und *deep learning,* gibt es unzählige Optimierungsmöglichkeiten des entwickelten Modells. Abhängig von der Problemstellung gibt es unterschiedliche Optima bezüglich Netzwerkart, -architektur und -design (Goodfellow *et al*., 2016) und neue Ansätze könnten zu besseren Ergebnissen führen. Bei *deep learning* gerät man schnell in das Arbeitsgebiet von Data-Scientists (Wikipedia, 2019). Ein besserer Fokus universitärer Ausbildung in den Bereichen Modellierung, Datenanalyse und Programmierung (Seidl, 2017) könnte helfen, ähnliche Arbeiten voranzutreiben.

Limitierungen meiner Aussagen beziehen sich hauptsächlich auf die künstlichen Startbedingungen des Studiendesigns. Ein fehlender Umweltgradient, nur eine vertretene Baumart, keine natürlichen abiotischen oder biotischen Störungen schränken die Aussagekraft ein. Ein weiterer Nachteil dieses Ansatzes ist, dass das verwendete DNN nur die Prozesse abbildet, die auch iLand unterliegen. Dasselbe gilt für die Umwelteinflüsse, welche für iLand spezifiziert sind. Die Anpassung zukünftiger Studiendesigns an reale Umweltbedingungen sind aber durchaus möglich und könnten in zukünftigen Projekten umgesetzt werden. Wie sich verschiedene BewirtschafterInnen mit verschiedenen Managementmöglichkeiten unter veränderten Umweltbedingungen und verschiedenen Klimawandelszenarien verhalten werden, ist ein weiterer Punkt für zukünftige Forschung. Eine allgemeine Limitierung von DNNs ist die riesige Menge an Daten, die benötigt wird, um

komplexe Beziehungen erfolgreich abstrahieren zu können (Rammer und Seidl, 2019a). Dieses Problem konnte aber durch umfangreiche iLand-Simulationen adressiert werden. Neben SVD gibt es noch eine ganze Bandbreite an dynamischen Vegetationsmodellen (Bugmann *et al.*, 2019), wobei keines davon *deep learning* verwendet und vergleichbare Literatur nicht vorhanden ist.

Die Ergebnisse dieser Arbeit konnten zeigen, dass Methoden der Künstlichen Intelligenz und im speziellen DNNs großes Potential in der Simulation von bewirtschafteten Wäldern haben. Das Meta-Modell SVD wird sich in dieser Hinsicht weiter entwickeln. Der Grundstein für Waldbewirtschaftungssimulationen in SVD wurde durch die vorliegende Arbeit gelegt. Da diese Diplomarbeit von einem durchaus experimentellen Charakter geprägt ist und zu Beginn noch nicht abzusehen war (siehe Forschungsfrage (i)), ob Erfolg möglich ist, sind die erreichten Ergebnisse in Summe durchaus positiv zu bewerten. Nicht zu vernachlässigen sind die Erfahrungen und Erkenntnisse, welche für zukünftige Anwendungen in diesem Bereich gesammelt werden konnten.

7 LITERATURVERZEICHNIS

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., … Google Brain. (2016). TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. USENIX Association. https://doi.org/10.1016/0076-6879(83)01039-3

Acevedo, M. F., Ablan, M., Urban, D. L., Pamarti, S. (2001). Estimating parameters of forest patch transition models from gap models. *Environmental Modelling and Software*, *16*, 649–658. https://doi.org/10.1016/S1364-8152(01)00034-2

Albrich, K. (2016). *Effects of forest management on the provisioning of ecosystem services under climate change in a mountain forest landscape to obtain the academic degree Diplom-Ingenieur*. Universität für Bodenkultur, Wien.

Bugmann, H., Seidl, R., Hartig, F., Bohn, F., Brůna, J., Cailleret, M., François, L., Heinke, J., Henrot, A. J., Hickler, T., Hülsmann, L., Huth, A., Jacquemin, I., Kollas, C., Lasch-Born, P., … Reyer, C. P. O. (2019). Tree mortality submodels drive simulated long-term forest dynamics: assessing 15 models from the stand to global scale. *Ecosphere*, *10*(2), 1–22. https://doi.org/10.1002/ecs2.2616

Chen, S. H., Jakeman, A. J., Norton, J. P. (2008). Artificial Intelligence techniques: An introduction to their use for modelling environmental systems. *Mathematics and Computers in Simulation*, *78*, 379–400. https://doi.org/10.1016/j.matcom.2008.01.028

Chollet, F. (2015). Keras. Abgerufen 15. September 2019 von https://keras.io/

Cipriotti, P. A., Wiegand, T., Pütz, S., Bartoloni, N. J., Paruelo, J. M. (2016). Nonparametric upscaling of stochastic simulation models using transition matrices. *Methods in Ecology and Evolution*, *7*, 313–322. https://doi.org/10.1111/2041-210X.12464

Duursma, R. A., Marshall, J. D., Robinson, A. P., Pangle, R. E. (2007). Description and test of a simple process-based model of forest growth for mixed-species stands. *Ecological Modelling*, *203*, 297–311. https://doi.org/10.1016/j.ecolmodel.2006.11.032

Goodfellow, I., Yoshua, B., Courville, A. (2016). *Deep learning* (S. 799). The MIT Press. https://doi.org/10.11432/jpnjvissci.39.75

Guido, van R. (2019). Python. Python Software Foundation. Abgerufen von https://www.python.org/

Gűneralp, B., Gertner, G. (2006). Feedback Loop Dominance Analysis of Two Tree Mortality Models. *Proceedings of the 24th International Conference of the System Dynamics Society*, *27*, 269–280.

Hadden, D., Grelle, A. (2016). Changing temperature response of respiration turns boreal forest from carbon sink into carbon source. *Agricultural and Forest Meteorology*, *223*, 30–38. https://doi.org/10.1016/j.agrformet.2016.03.020

Hanewinkel, M., Pretzsch, H. (2000). Modelling the conversion from even-aged to uneven-aged stands of Norway spruce (Picea abies L. Karst.) with a distance-dependent growth simulator. *Forest Ecology and Management*, *134*, 55–70. https://doi.org/10.1016/S0378-1127(99)00245-5

Hart, E., Sim, K., Kamimura, K., Meredieu, C., Guyon, D., Gardiner, B. (2019). Use of machine learning techniques to model wind damage to forests. *Agricultural and Forest Meteorology*, *265*, 16–29. https://doi.org/10.1016/j.agrformet.2018.10.022

IPCC, Masson-Delmotte, V., P., Zhai, H.-O., Pörtner, D., Roberts, J., Skea, P. R., Shukla, A., Pirani, W., Moufouma-Okia, C. P., R. Pidcock, S., Connors, J. B. R., Matthews, Y., Chen, X., M.I., Z., Gomis, E., … Tignor, T. W. (2018). *Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change,* (S. 616).

Landsberg, J. J., Waring, R. H. (1997). A generalised model of forest productivity using simplified concepts of radiation-use efficiency, carbon balance and partitioning. *Forest Ecology and Management*, *95*, 209–228. https://doi.org/10.1016/S0378-1127(97)00026-1

Leahy, J. E., Reeves, E. G., Bell, K. P., Straub, C. L., Wilson, J. S. (2013). Agent-Based Modeling of Harvest Decisions by Small Scale Forest Landowners in Maine, USA. *International Journal of Forestry Research*, *2013*, 1–12. https://doi.org/10.1155/2013/563068

Lecun, Y., Bengio, Y., Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444. https://doi.org/10.1038/nature14539

Lexer, M. J., Hönninger, K. (2001). A modified 3D-patch model for spatially explicit simulation of vegetation composition in heterogeneous landscapes. *Forest Ecology and Management*, *144*, 43–65. https://doi.org/10.1016/S0378-1127(00)00386-8

Mayer, H. (1992). *Waldbau auf soziologisch-ökologischer Grundlage* (4. Aufl., S. 522). Wien: Gustav Fischer Verlag.

Millar, C. I., Stephenson, N. L., Stephens, S. L. (2007). Climate change and forests of the future: Managing in the face of uncertainty. *Ecological Applications*, *17*(8), 2145–2151. https://doi.org/10.1890/06-1715.1

Nielsen, M. A. (2015). Neural Networks and Deep Learning. Abgerufen 2. Februar 2019 von http://neuralnetworksanddeeplearning.com/index.html

Pelyukh, O., Fabrika, M., Kucbel, S., Valent, P., Zahvoyska, L. (2018). Modelling of secondary even-aged Norway spruce stands conversion using the tree growth simulator SIBYLA: Se „Rakhiv Forestry“ case study. *Bulletin of the Transilvania University of Brasov, Series II: Forestry, Wood Industry, Agricultural Food Engineering*, *11*(60), 29–46.

Pierre, R. (2019). Spyder. MIT. Abgerufen von https://www.spyder-ide.org/

Pretzsch, H., Biber, P., Ďurský, J. (2002). The single tree-based stand simulator SILVA: Construction, application and evaluation. *Forest Ecology and Management*, *162*, 3–21. https://doi.org/10.1016/S0378-1127(02)00047-6

R Core Team. (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing. Abgerufen 15. Januar 2019 von https://www.r-project.org/

Rammer, W., Seidl, R. (2015). ABE - the Agent Based manaegment Engine for iLand. Abgerufen 20. Januar 2019 von http://iland.boku.ac.at/ABE

Rammer, W. (2018). SVD Documentation. Abgerufen 24. September 2019, von https://github.com/SVDmodel/SVD/blob/master/docs/README.md

Rammer, W., Seidl, R. (2015). Coupling human and natural systems: Simulating adaptive management agents in dynamically changing forest landscapes. *Global Environmental Change*, *35*, 475–485. https://doi.org/10.1016/j.gloenvcha.2015.10.003

Rammer, W., Seidl, R. (2019a). A scalable model of vegetation transitions using deep neural networks. *Methods in Ecology and Evolution*, *10*, 879–890. https://doi.org/10.1111/2041-210X.13171

Rammer, W., Seidl, R. (2019b). Harnessing Deep Learning in Ecology: An Example Predicting Bark Beetle Outbreaks. *Frontiers in Plant Science*, *10*(1327), 1–9. https://doi.org/10.3389/fpls.2019.01327

Rammer, W., Seidl, R. (2019c). Supplementary Material: A scalable model of vegetation transitions using deep neural networks. *Methods in Ecology and Evolution*, *10*, 1–16. https://doi.org/10.1111/2041-210X.13171

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., Prabhat. (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, *566*, 195–204. https://doi.org/10.1038/s41586-019-0912-1

Reininger, H. (2000). *Das Plenterprinzip oder die Überführung des Altersklassenwaldes* (S. 238). Graz: Stocker.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408.

Schneider, D. C. (1994). *Quantitativ e Ecology Spatia l and Tempora l Scaling* (S. 399). San Diego: Academic Press, INC.

Seidl, R. (2017). To Model or not to Model, That is no Longer the Question for Ecologists. *Ecosystems*, *20*, 222–228. https://doi.org/10.1007/s10021-016-0068-x

Seidl, R., Eastaugh, C. S., Kramer, K., Maroschek, M., Reyer, C., Socha, J., Vacchiano, G., Zlatanov, T., Hasenauer, H. (2013). Scaling issues in forest ecosystem management and how to address them with models. *European Journal of Forest Research*, *132*, 653–666. https://doi.org/10.1007/s10342-013-0725-y

Seidl, R., Rammer, W. (2015). ABE - the Agent Based management Engine for iLand - online model documentation. Abgerufen 20. Januar 2019 von http://iland.boku.ac.at/ABE

Seidl, R., Rammer, W. (2017). Climate change amplifies the interactions between wind and bark beetle disturbances in forest landscapes. *Landscape Ecology*, *32*(7), 1485–1498. https://doi.org/10.1007/s10980-016-0396-4

Seidl, R., Rammer, W., Bellos, P., Hochbichler, E., Lexer, M. J. (2010). Testing generalized allometries in allocation modeling within an individual-based simulation framework. *Trees - Structure and Function*, *24*, 139–150. https://doi.org/10.1007/s00468-009-0387-z

Seidl, R., Rammer, W., Blennow, K. (2014). Simulating wind disturbance impacts on forest landscapes: Tree-level heterogeneity matters. *Environmental Modelling and Software*, *51*, 1–11. https://doi.org/10.1016/j.envsoft.2013.09.018

Seidl, R., Rammer, W., Jäger, D., Currie, W. S., Lexer, M. J. (2007). Assessing trade-offs between carbon sequestration and timber production within a framework of multi-purpose forestry in Austria. *Forest Ecology and Management*, *248*, 64–79. https://doi.org/10.1016/j.foreco.2007.02.035

Seidl, R., Rammer, W., Scheller, R. M., Spies, T. A. (2012). An individual-based process model to simulate landscape-scale forest ecosystem dynamics. *Ecological Modelling*, *231*, 87–100. https://doi.org/10.1016/j.ecolmodel.2012.02.015

Seidl, R., Rammer, W., Spies, T. A. (2014). Disturbance legacies increase the resilience of forest ecosystem structure, composition, and functioning. *Ecological Applications*, *24*(8), 2063–2077. https://doi.org/10.1890/14-0255.1

Seidl, R., Spies, T. A., Rammer, W., Steel, E. A., Pabst, R. J., Olsen, K. (2012). Multi-scale Drivers of Spatial Variation in Old-Growth Forest Carbon Density Disentangled with Lidar and an Individual-Based Landscape Model. *Ecosystems*, *15*, 1321–1335. https://doi.org/10.1007/s10021-012-9587-2

Silva Pedro, M., Rammer, W., Seidl, R. (2015). Tree species diversity mitigates disturbance impacts on the forest carbon cycle. *Oecologia*, *177*, 619–630. https://doi.org/10.1007/s00442-014-3150-0

Sterba, H., Monserud, R. A. (1997). Applicability of the forest stand growth simulator PROGNAUS for the Austrian part of the Bohemian Massif. *Ecological Modelling*, *98*, 23–34. https://doi.org/10.1016/S0304-3800(96)01934-5

TensorFlow. (2017). TensorBoard: Visualizing Learning. Abgerufen 26. September 2019 von https://www.tensorflow.org/guide/summaries_and_tensorboard

TensorFlow. (2019). Tensorflow. Abgerufen 19. Juli 2019 von https://www.tensorflow.org/

Thom, D., Rammer, W., Dirnböck, T., Müller, J., Kobler, J., Katzensteiner, K., Helm, N., Seidl, R. (2017). The impacts of climate change and disturbance on spatio-temporal trajectories of biodiversity in a temperate forest landscape. *Journal of Applied Ecology*, *54*, 28–38. https://doi.org/10.1111/1365-2664.12644

Thurnher, C., Klopf, M., Hasenauer, H. (2017). MOSES – A tree growth simulator for modelling stand response in Central Europe. *Ecological Modelling*, *352*, 58–76. https://doi.org/10.1016/j.ecolmodel.2017.01.013

Tshitoyan, V., Dagdelen, J., Weston, L., Dunn, A., Rong, Z., Kononova, O., Persson, K. A., Ceder, G., Jain, A. (2019). Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, *571*, 95–98. https://doi.org/10.1038/s41586-019-1335-8

Urban, D. L. (2005). Modeling ecological processes across scales. *Ecology*, *86*(8), 1996–2006.

Wick, C. (2017). Deep learning. *Informatik Spektrum*, *40*(1), 103–107. https://doi.org/DOI 10.1007/s00287-016-1013-2

Wiens, J. A. (1989). Spatial scaling in ecology. *Functional Ecology*, *3*, 385–397.

Wikipedia. (2019). Data Science. Abgerufen 28. November 2019 von https://de.wikipedia.org/w/index.php?title=Data_Science&oldid=193981143

8 ABBILDUNGSVERZEICHNIS

9 TABELLENVERZEICHNIS

10 ANHANG

10.1 STATISTIK GENAUIGKEIT VEGETATIONSZUSTÄNDE

Tabelle 12: Statistische Auswertung der Genauigkeit für die Vegetationszustandsschätzungen von DNN1

DNN1 Einzelmanagements
Konfusions-Matrix der Vegetationszustände

ALL

```
               Accuracy : 0.7209
                 95% CI : (0.7201, 0.7217)
    No Information Rate : 0.1549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.698
```

REF

```
               Accuracy : 0.8136
                 95% CI : (0.8112, 0.816)
    No Information Rate : 0.7165
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5987
```

KS100

```
               Accuracy : 0.7437
                 95% CI : (0.7415, 0.7458)
    No Information Rate : 0.1903
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7075
```

KS80

```
               Accuracy : 0.7342
                 95% CI : (0.732, 0.7363)
    No Information Rate : 0.1963
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7033
```

SS100

```
               Accuracy : 0.6944
                 95% CI : (0.6922, 0.6966)
    No Information Rate : 0.1751
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6656
```

SS80

```
               Accuracy : 0.6226
                 95% CI : (0.6202, 0.6249)
    No Information Rate : 0.1872
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5846
```

ZSN55

```
               Accuracy : 0.7325
                 95% CI : (0.7305, 0.7345)
    No Information Rate : 0.3238
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6336
```

ZSN45

```
               Accuracy : 0.7346
                 95% CI : (0.7328, 0.7364)
    No Information Rate : 0.4074
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6094
```

Tabelle 13: Statistische Auswertung der Genauigkeit für die Vegetationszustandsschätzungen von DNN2

DNN2 Bewirtschaftungsvariante
Konfusions-Matrix der Vegetationszustände

ALL

```
               Accuracy : 0.7075
                 95% CI : (0.7067, 0.7083)
    No Information Rate : 0.1549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.685
```

REF

```
               Accuracy : 0.8846
                 95% CI : (0.8826, 0.8865)
    No Information Rate : 0.7165
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7509
```

KS100

```
               Accuracy : 0.7262
                 95% CI : (0.724, 0.7284)
    No Information Rate : 0.1903
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6855
```

KS80

```
               Accuracy : 0.7675
                 95% CI : (0.7655, 0.7696)
    No Information Rate : 0.1963
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7353
```

SS100

```
               Accuracy : 0.7229
                 95% CI : (0.7208, 0.725)
    No Information Rate : 0.1751
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6961
```

SS80

```
               Accuracy : 0.6568
                 95% CI : (0.6545, 0.6591)
    No Information Rate : 0.1872
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6192
```

ZSN55

```
               Accuracy : 0.5946
                 95% CI : (0.5923, 0.5968)
    No Information Rate : 0.3238
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4367
```

ZSN45

```
               Accuracy : 0.6881
                 95% CI : (0.6862, 0.69)
    No Information Rate : 0.4074
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.534
```

Tabelle 14: Statistische Auswertung der Genauigkeit für die Vegetationszustandsschätzungen von DNN3

DNN3 Kombiniert
Konfusions-Matrix der Vegetationszustände

ALL

```
               Accuracy : 0.7753
                 95% CI : (0.7745, 0.776)
    No Information Rate : 0.1549
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7581
```

REF

```
               Accuracy : 0.883
                 95% CI : (0.8811, 0.885)
    No Information Rate : 0.7165
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.743
```

KS100

```
               Accuracy : 0.8106
                 95% CI : (0.8086, 0.8125)
    No Information Rate : 0.1903
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7828
```

KS80

```
               Accuracy : 0.807
                 95% CI : (0.8051, 0.8089)
    No Information Rate : 0.1963
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7807
```

SS100

```
               Accuracy : 0.7463
                 95% CI : (0.7443, 0.7484)
    No Information Rate : 0.1751
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7221
```

SS80

```
               Accuracy : 0.6975
                 95% CI : (0.6953, 0.6997)
    No Information Rate : 0.1872
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6655
```

ZSN55

```
               Accuracy : 0.7509
                 95% CI : (0.749, 0.7529)
    No Information Rate : 0.3238
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6573
```

ZSN45

```
               Accuracy : 0.7767
                 95% CI : (0.775, 0.7784)
    No Information Rate : 0.4074
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6694
```

10.2 Statistik Genauigkeit Verweilzeit

Tabelle 15: Statistische Auswertung der Genauigkeit für die Verweilzeitschätzungen von DNN1

DNN1 Einzelmanagements
Konfusions-Matrix der Verweilzeit

ALL

```
                Accuracy : 0.6016
                  95% CI : (0.6007, 0.6025)
    No Information Rate : 0.4379
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.4306
```

REF

```
                Accuracy : 0.8474
                  95% CI : (0.8452, 0.8495)
    No Information Rate : 0.8582
    P-Value [Acc > NIR] : 1

                   Kappa : 0.2758
```

KS100

```
                Accuracy : 0.5363
                  95% CI : (0.5338, 0.5387)
    No Information Rate : 0.4115
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.3557
```

KS80

```
                Accuracy : 0.6403
                  95% CI : (0.638, 0.6426)
    No Information Rate : 0.4134
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.5021
```

SS100

```
                Accuracy : 0.554
                  95% CI : (0.5516, 0.5563)
    No Information Rate : 0.2989
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.4456
```

SS80

```
                Accuracy : 0.4932
                  95% CI : (0.4908, 0.4957)
    No Information Rate : 0.3241
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.352
```

ZSN55

```
                Accuracy : 0.5343
                  95% CI : (0.532, 0.5365)
    No Information Rate : 0.3445
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.3946
```

ZSN45

```
                Accuracy : 0.6754
                  95% CI : (0.6735, 0.6774)
    No Information Rate : 0.5445
    P-Value [Acc > NIR] : < 2.2e-16

                   Kappa : 0.4208
```

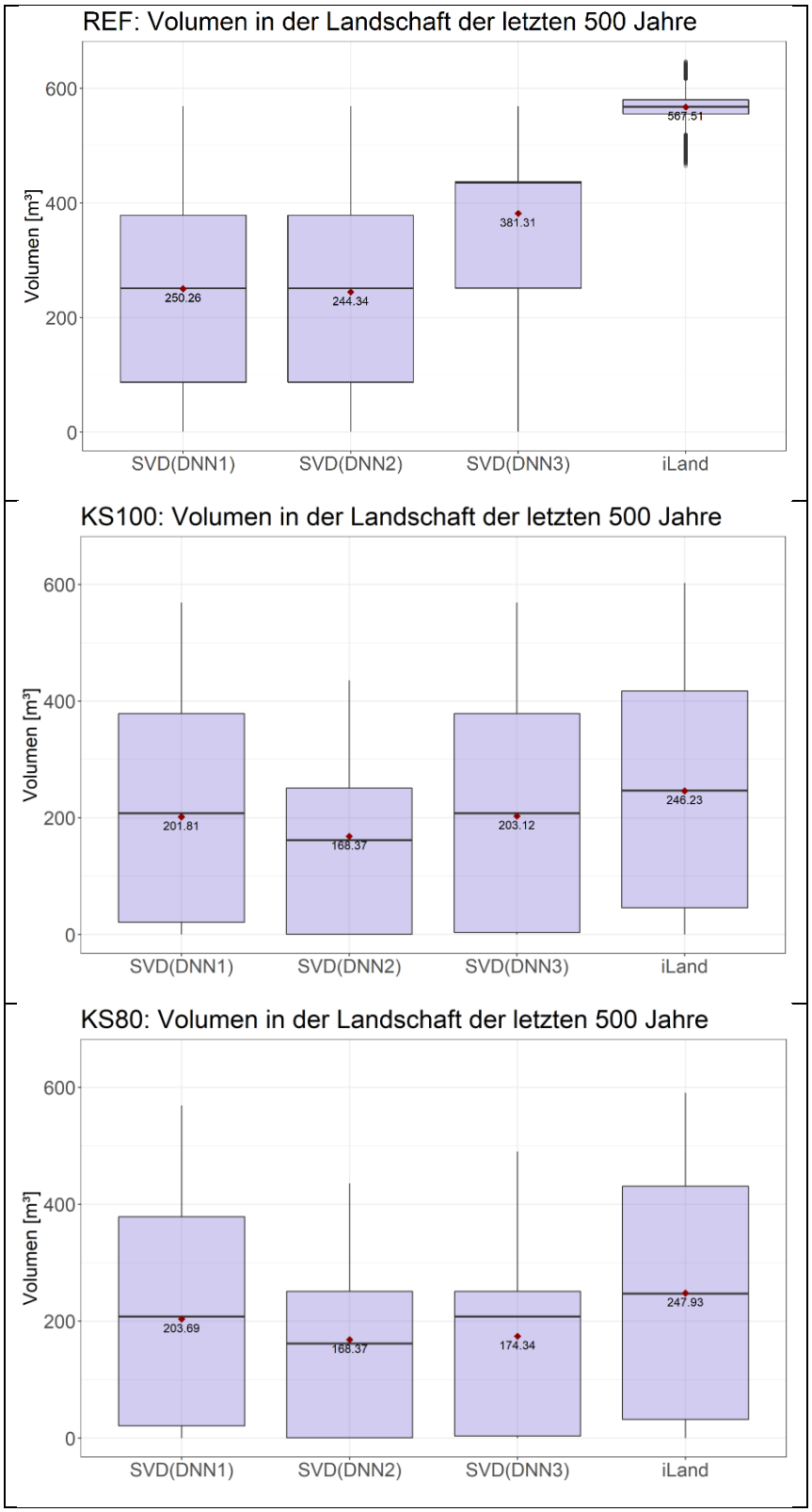
Tabelle 16: Statistische Auswertung der Genauigkeit für die Verweilzeitschätzungen von DNN2

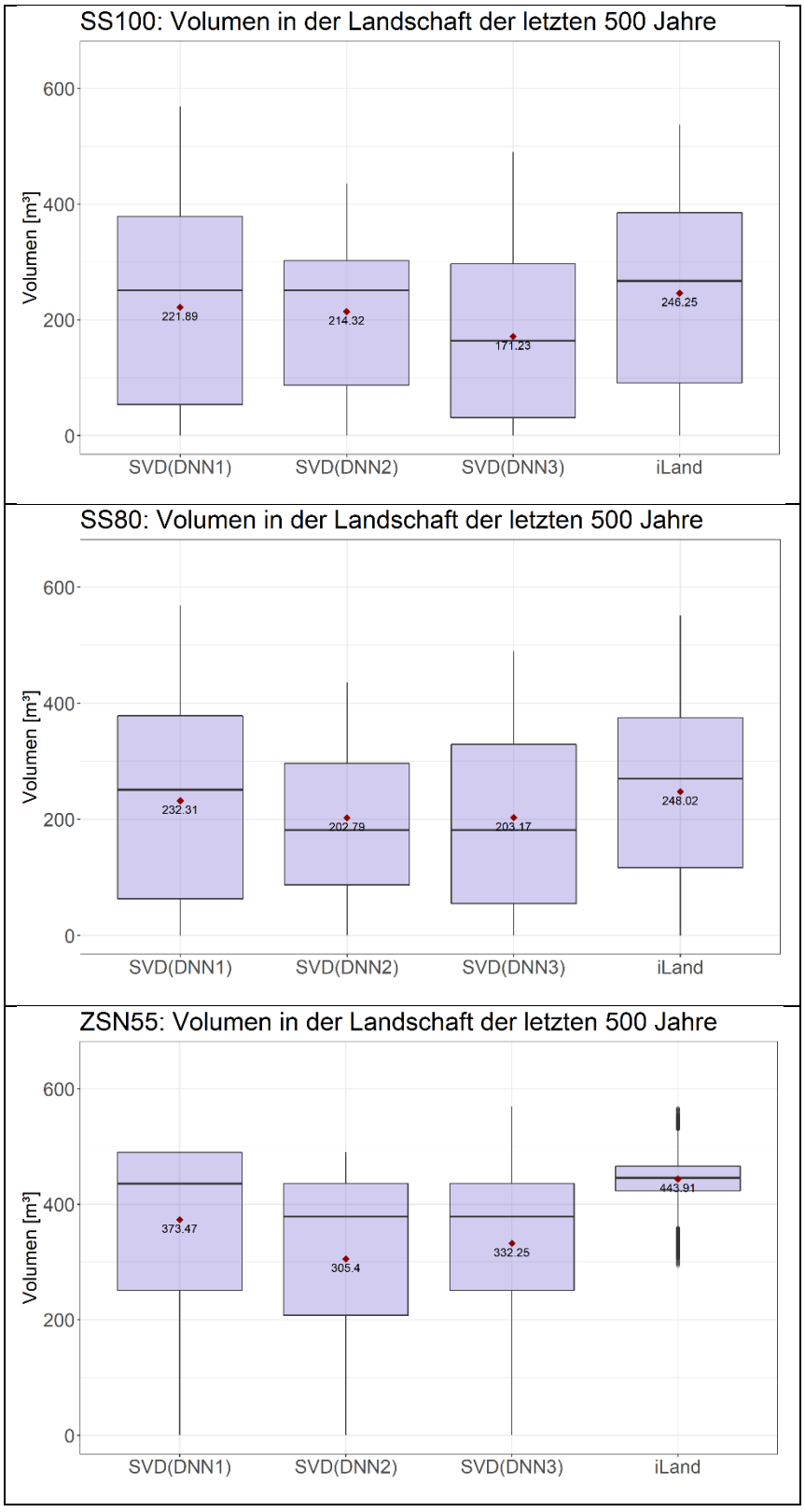
DNN2 Bewirtschaftungsvariante **Konfusions-Matrix der Verweilzeit**	
ALL Accuracy : 0.5962 95% CI : (0.5953, 0.5971) No Information Rate : 0.4379 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.413	REF Accuracy : 0.8716 95% CI : (0.8696, 0.8736) No Information Rate : 0.8582 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.2448
KS100 Accuracy : 0.5667 95% CI : (0.5643, 0.5692) No Information Rate : 0.4115 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.3854	KS80 Accuracy : 0.6431 95% CI : (0.6408, 0.6454) No Information Rate : 0.4134 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.4995
SS100 Accuracy : 0.5595 95% CI : (0.5571, 0.5618) No Information Rate : 0.2989 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.4476	SS80 Accuracy : 0.5134 95% CI : (0.5109, 0.5158) No Information Rate : 0.3241 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.3717
ZSN55 Accuracy : 0.4608 95% CI : (0.4585, 0.463) No Information Rate : 0.3445 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.2891	ZSN45 Accuracy : 0.6551 95% CI : (0.6532, 0.6571) No Information Rate : 0.5445 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.3764

Tabelle 17: Statistische Auswertung der Genauigkeit für die Verweilzeitschätzungen von DNN3

DNN3 Kombiniert **Konfusions-Matrix der Verweilzeit**	
ALL Accuracy : 0.6448 95% CI : (0.644, 0.6457) No Information Rate : 0.4379 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.5006	REF Accuracy : 0.8716 95% CI : (0.8695, 0.8736) No Information Rate : 0.8582 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.257
KS100 Accuracy : 0.607 95% CI : (0.6046, 0.6094) No Information Rate : 0.4115 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.4677	KS80 Accuracy : 0.6883 95% CI : (0.686, 0.6905) No Information Rate : 0.4134 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.5782
SS100 Accuracy : 0.592 95% CI : (0.5896, 0.5943) No Information Rate : 0.2989 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.4925	SS80 Accuracy : 0.536 95% CI : (0.5336, 0.5384) No Information Rate : 0.3241 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.405
ZSN55 Accuracy : 0.5563 95% CI : (0.554, 0.5585) No Information Rate : 0.3445 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.4359	ZSN45 Accuracy : 0.7264 95% CI : (0.7246, 0.7282) No Information Rate : 0.5445 P-Value [Acc > NIR] : < 2.2e-16 Kappa : 0.5349

10.3 VOLUMEN


REF: Volumen in der Landschaft der letzten 500 Jahre
Volumen [m³]
600
400
200
0
250.26
244.34
381.31
567.51
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
KS100: Volumen in der Landschaft der letzten 500 Jahre
Volumen [m³]
600
400
200
0
201.81
168.37
203.12
246.23
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
KS80: Volumen in der Landschaft der letzten 500 Jahre
Volumen [m³]
600
400
200
0
203.69
168.37
174.34
247.93
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand


SS100: Volumen in der Landschaft der letzten 500 Jahre
Volumen [m³]
600
400
200
0
221.89
214.32
171.23
246.25
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: Volumen in der Landschaft der letzten 500 Jahre
Volumen [m³]
600
400
200
0
232.31
202.79
203.17
248.02
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN55: Volumen in der Landschaft der letzten 500 Jahre
Volumen [m³]
600
400
200
0
373.47
305.4
332.25
443.91
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

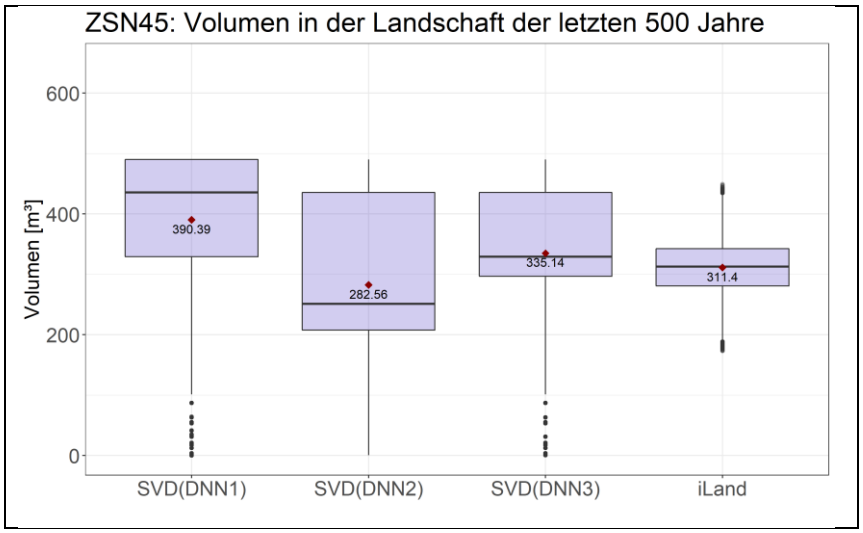
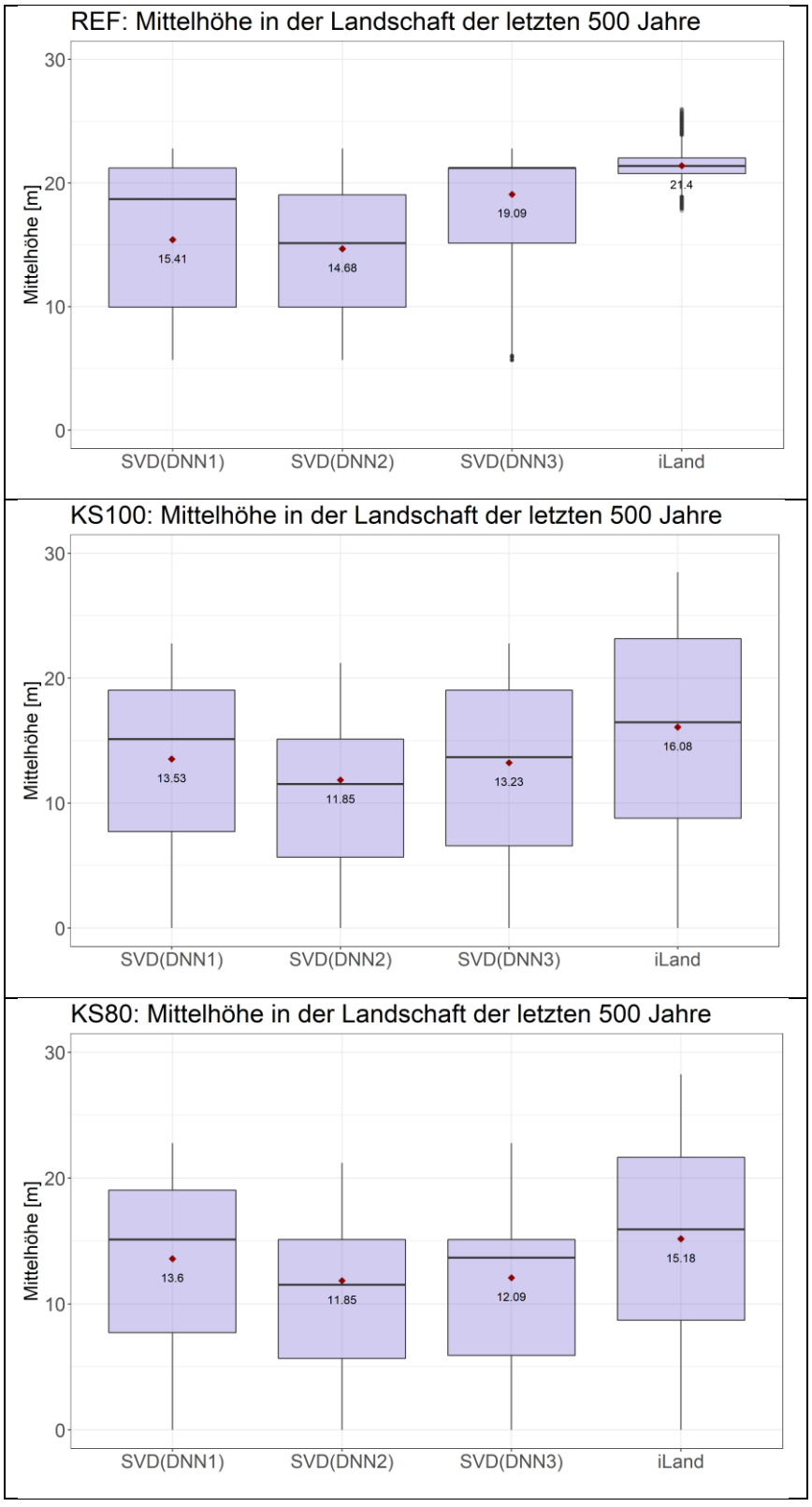
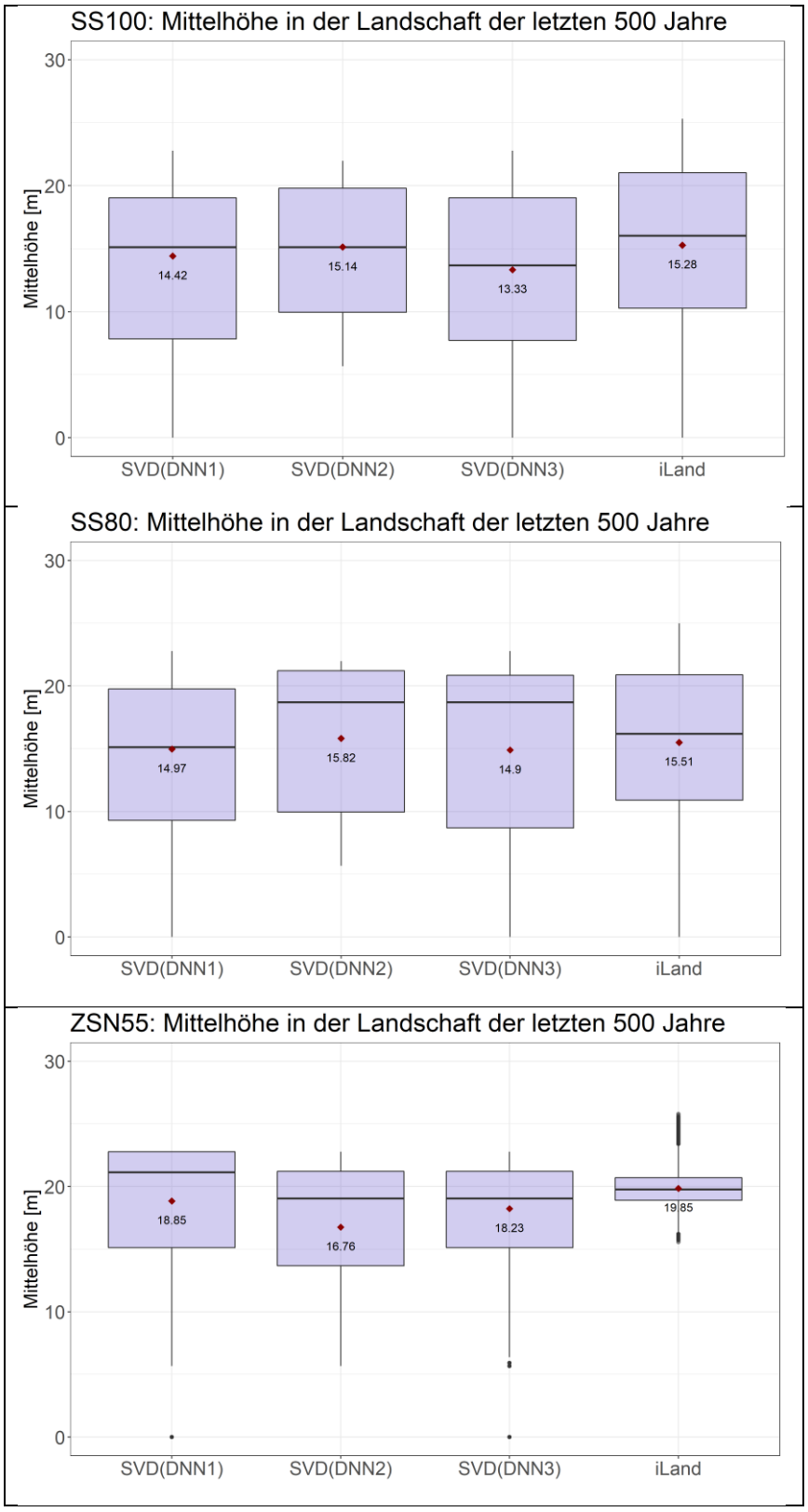


Abbildung 37: Gegenüberstellung SVD/iLand: Volumen

10.4 MITTELHÖHE




SS100: Mittelhöhe in der Landschaft der letzten 500 Jahre
Mittelhöhe [m]
30
20
10
0
14.42
15.14
13.33
15.28
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: Mittelhöhe in der Landschaft der letzten 500 Jahre
Mittelhöhe [m]
30
20
10
0
14.97
15.82
14.9
15.51
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN55: Mittelhöhe in der Landschaft der letzten 500 Jahre
Mittelhöhe [m]
30
20
10
0
18.85
16.76
18.23
19.85
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

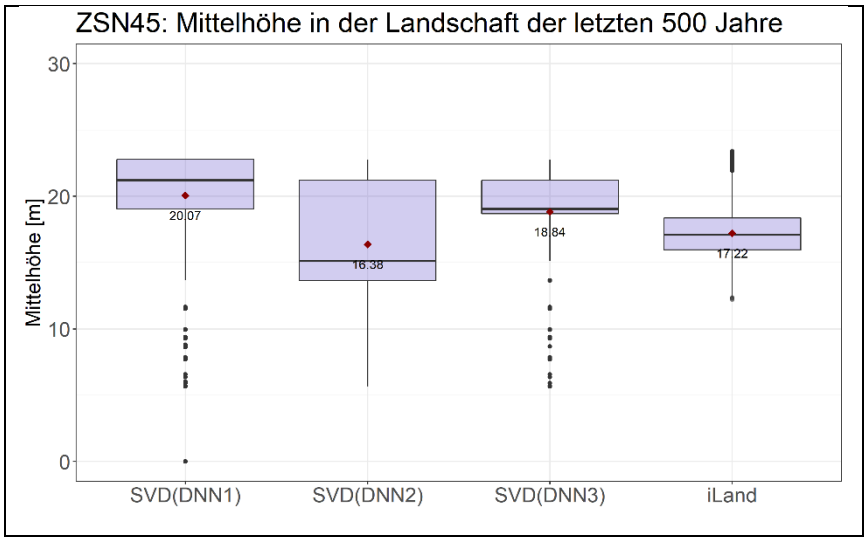
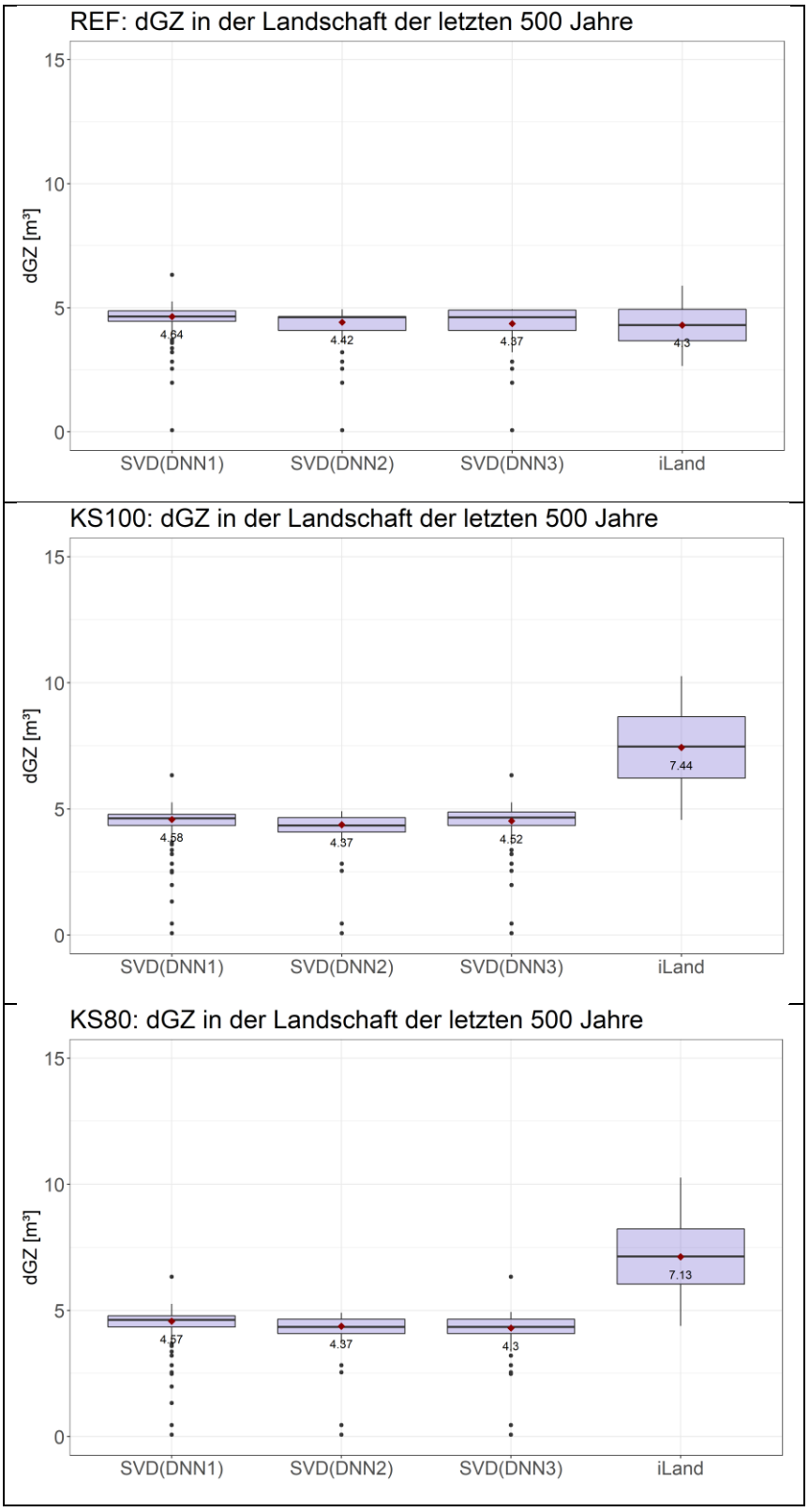
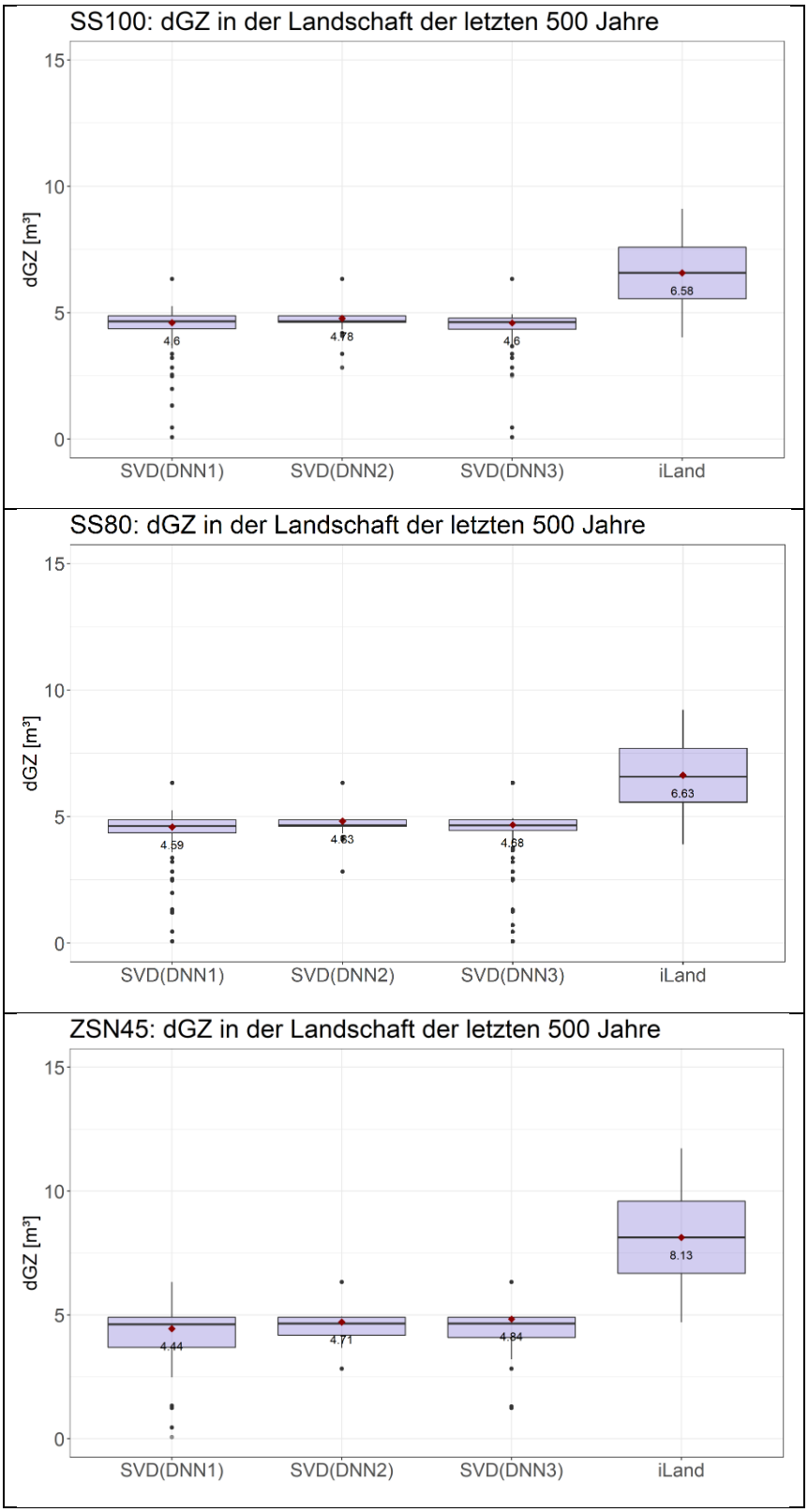


Abbildung 38: Gegenüberstellung SVD/iLand: Mittelhöhe

10.5 Durchschnittlicher Gesamtzuwachs




SS100: dGZ in der Landschaft der letzten 500 Jahre
dGZ [m³]
15
10
5
0
4.6
4.78
4.6
6.58
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: dGZ in der Landschaft der letzten 500 Jahre
dGZ [m³]
15
10
5
0
4.59
4.63
4.58
6.63
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN45: dGZ in der Landschaft der letzten 500 Jahre
dGZ [m³]
15
10
5
0
4.44
4.71
4.84
8.13
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

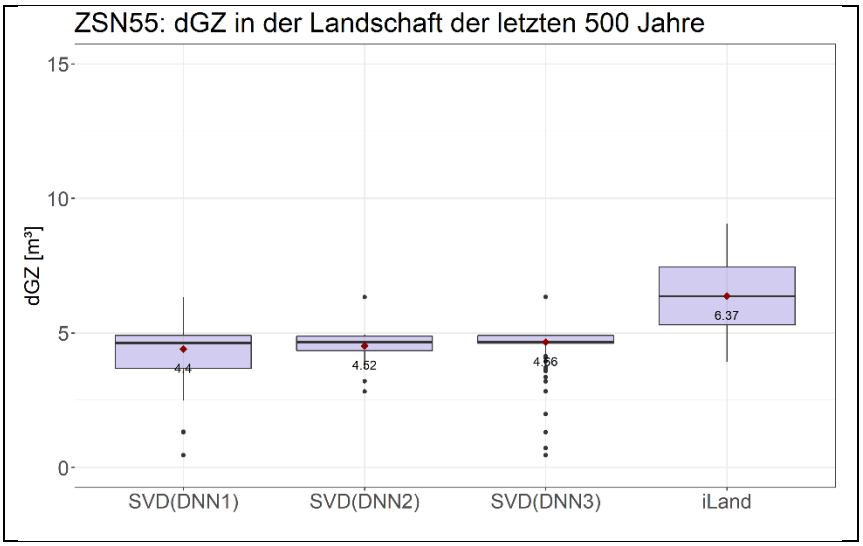
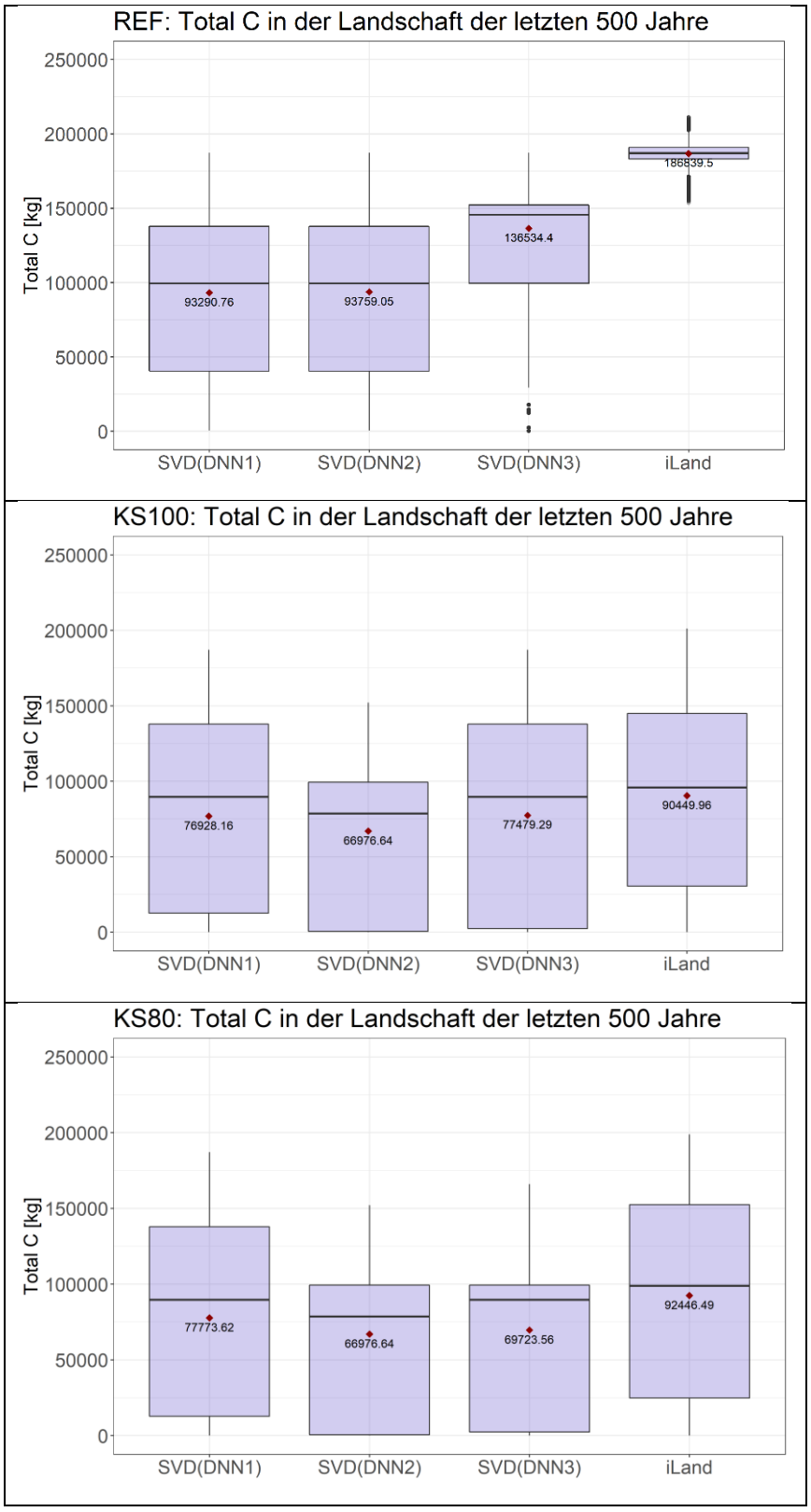
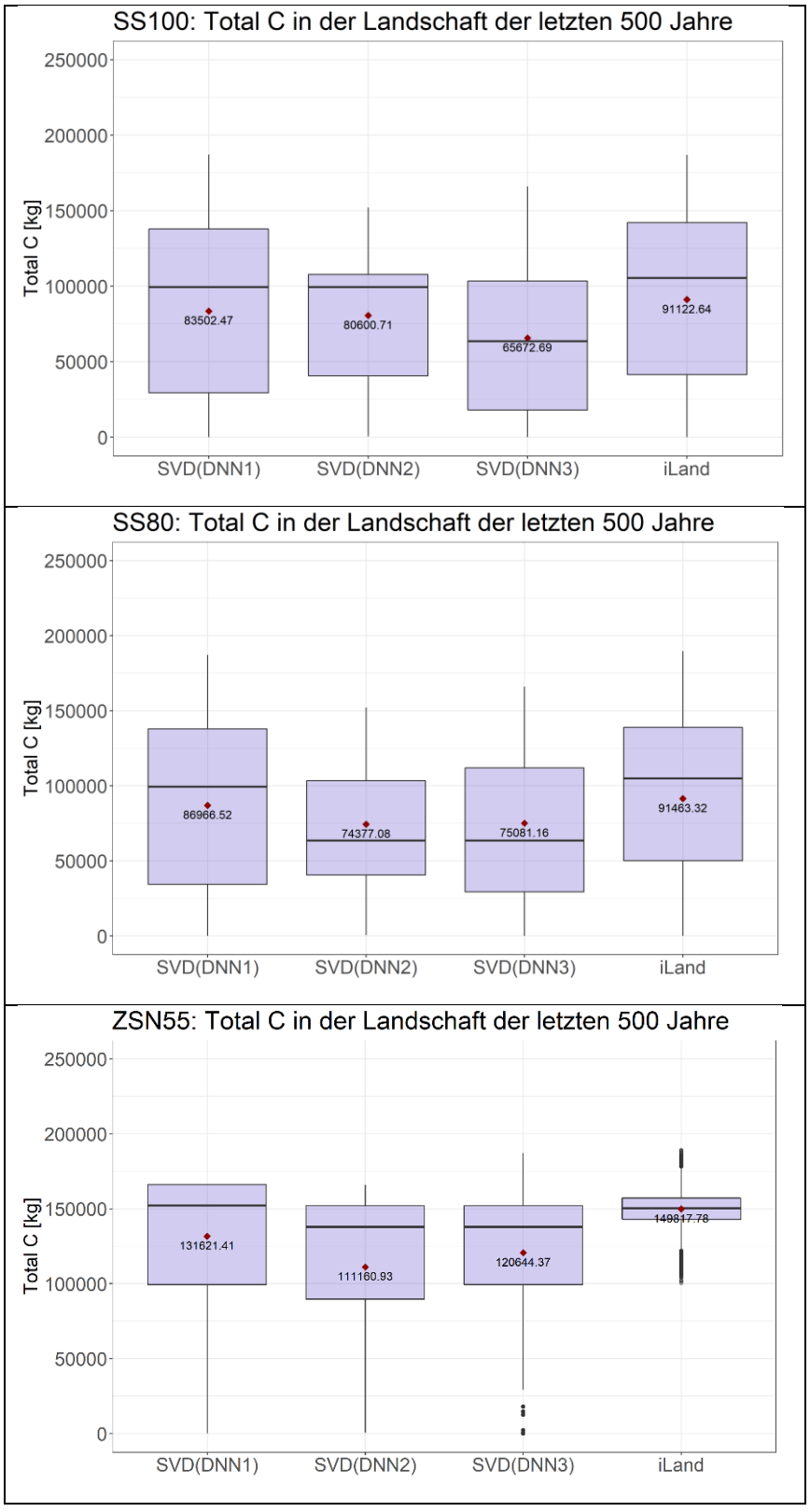


Abbildung 39: Gegenüberstellung SVD/iLand: Durchschnittlicher Gesamtzuwachs

10.6 KOHLENSTOFFVORRAT




SS100: Total C in der Landschaft der letzten 500 Jahre
Total C [kg]
250000
200000
150000
100000
50000
0
83502.47
80600.71
65672.69
91122.64
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: Total C in der Landschaft der letzten 500 Jahre
Total C [kg]
250000
200000
150000
100000
50000
0
86966.52
74377.08
75081.16
91463.32
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN55: Total C in der Landschaft der letzten 500 Jahre
Total C [kg]
250000
200000
150000
100000
50000
0
131621.41
111160.93
120644.37
149817.78
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

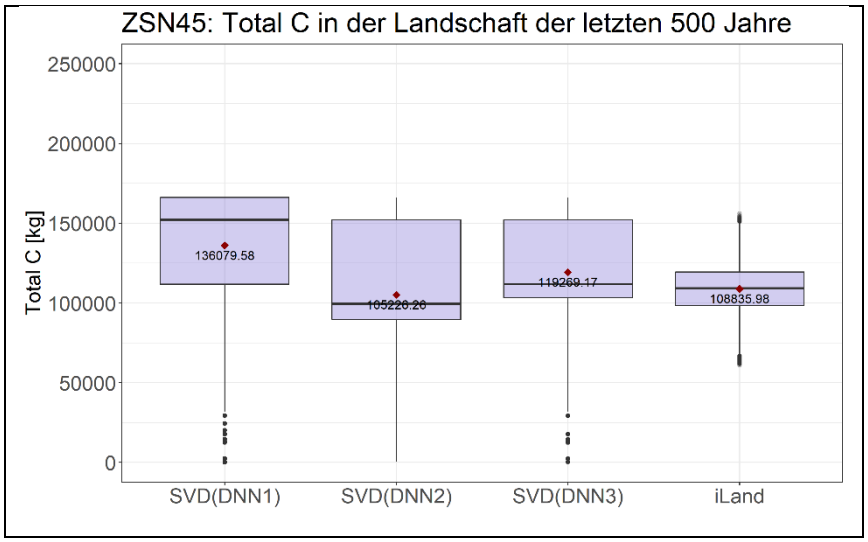
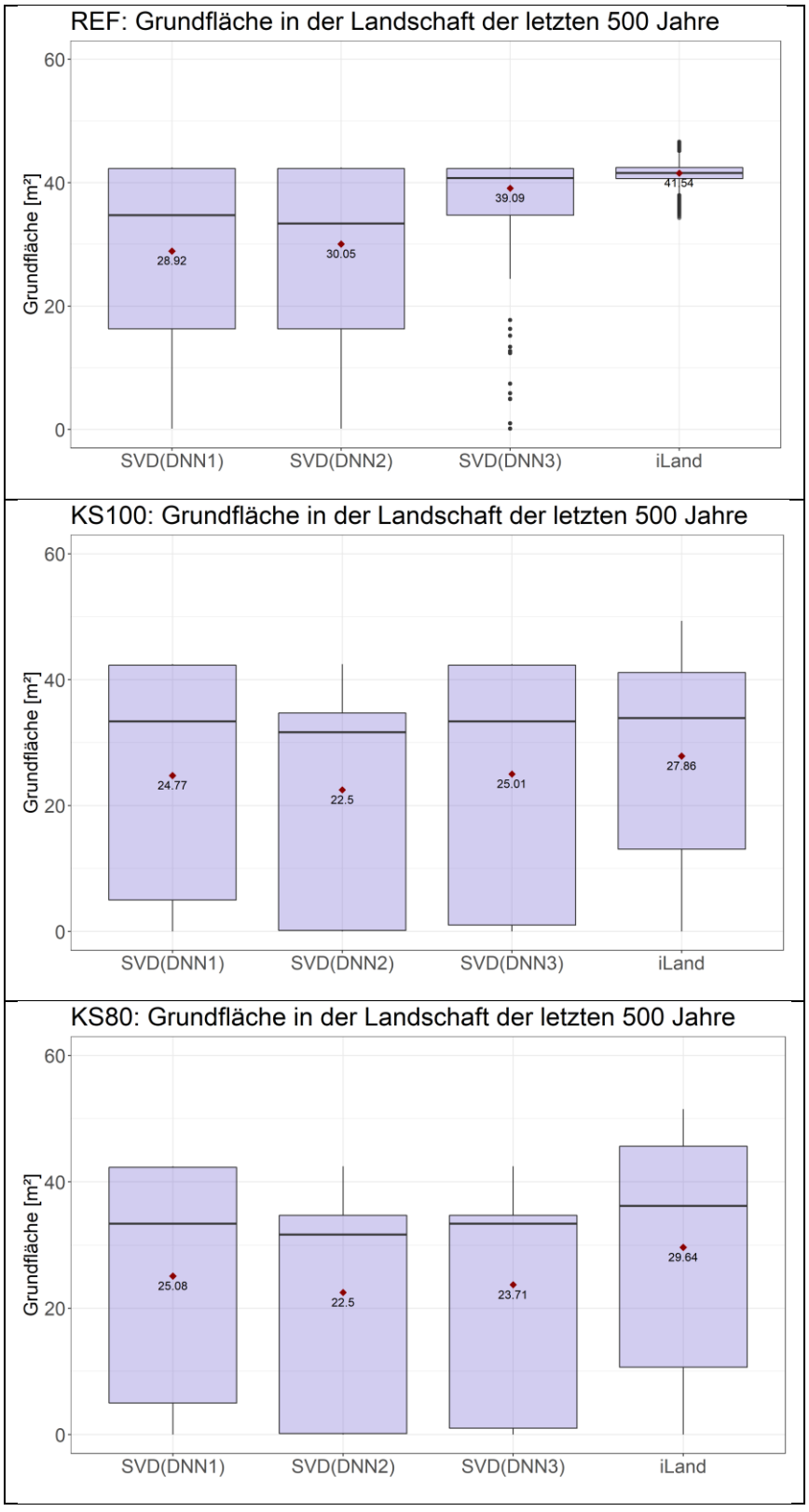
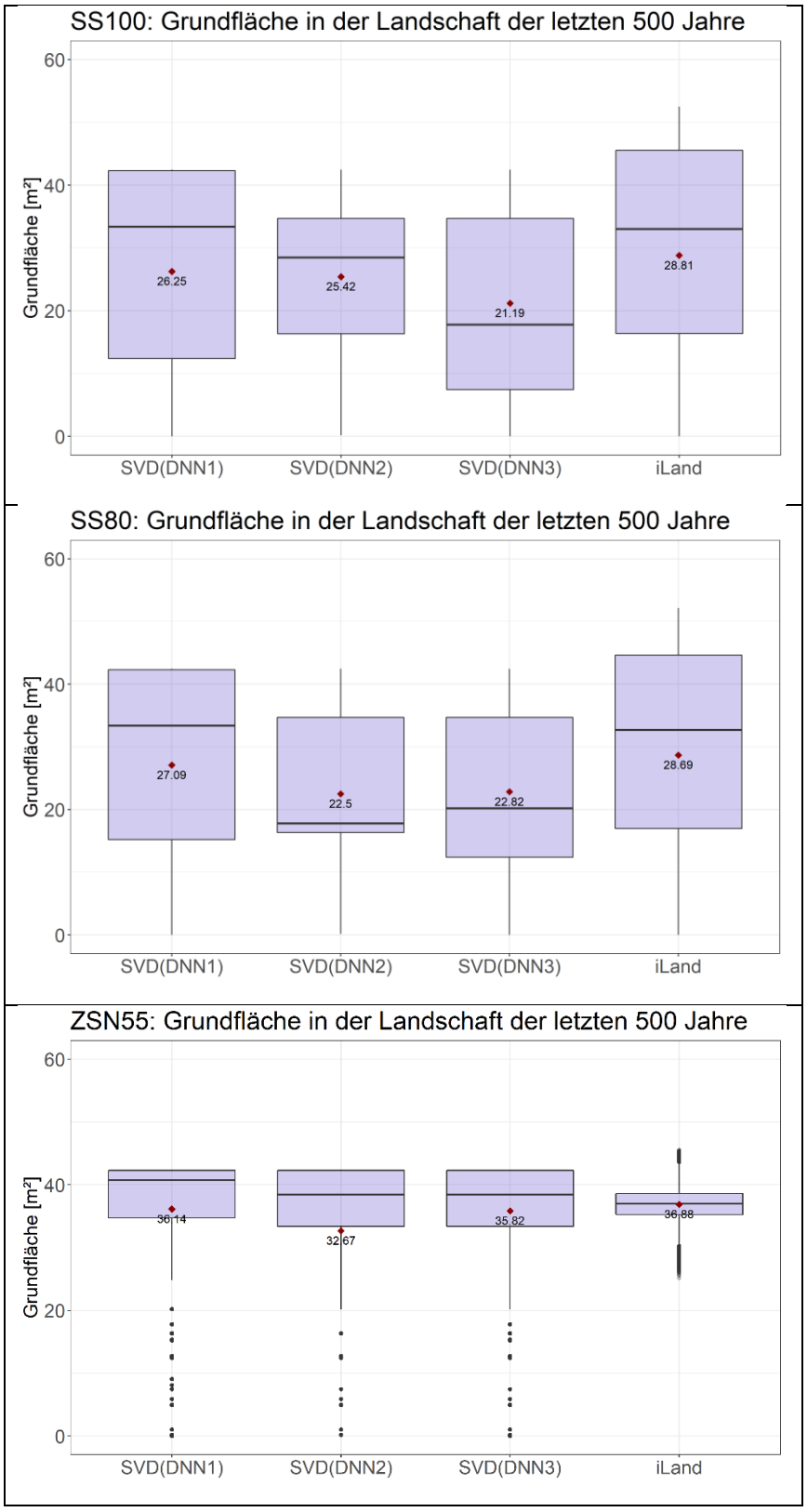


Abbildung 40: Gegenüberstellung SVD/iLand: Kohlenstoffvorrat

10.7 GRUNDFLÄCHE




SS100: Grundfläche in der Landschaft der letzten 500 Jahre
Grundfläche [m²]
60
40
20
0
26.25
25.42
21.19
28.81
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: Grundfläche in der Landschaft der letzten 500 Jahre
Grundfläche [m²]
60
40
20
0
27.09
22.5
22.82
28.69
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN55: Grundfläche in der Landschaft der letzten 500 Jahre
Grundfläche [m²]
60
40
20
0
36.14
32.67
35.82
36.88
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

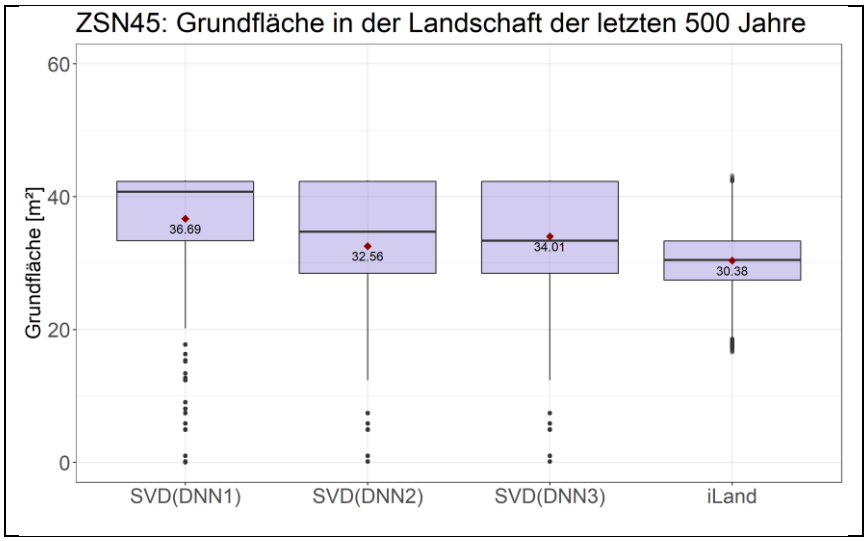
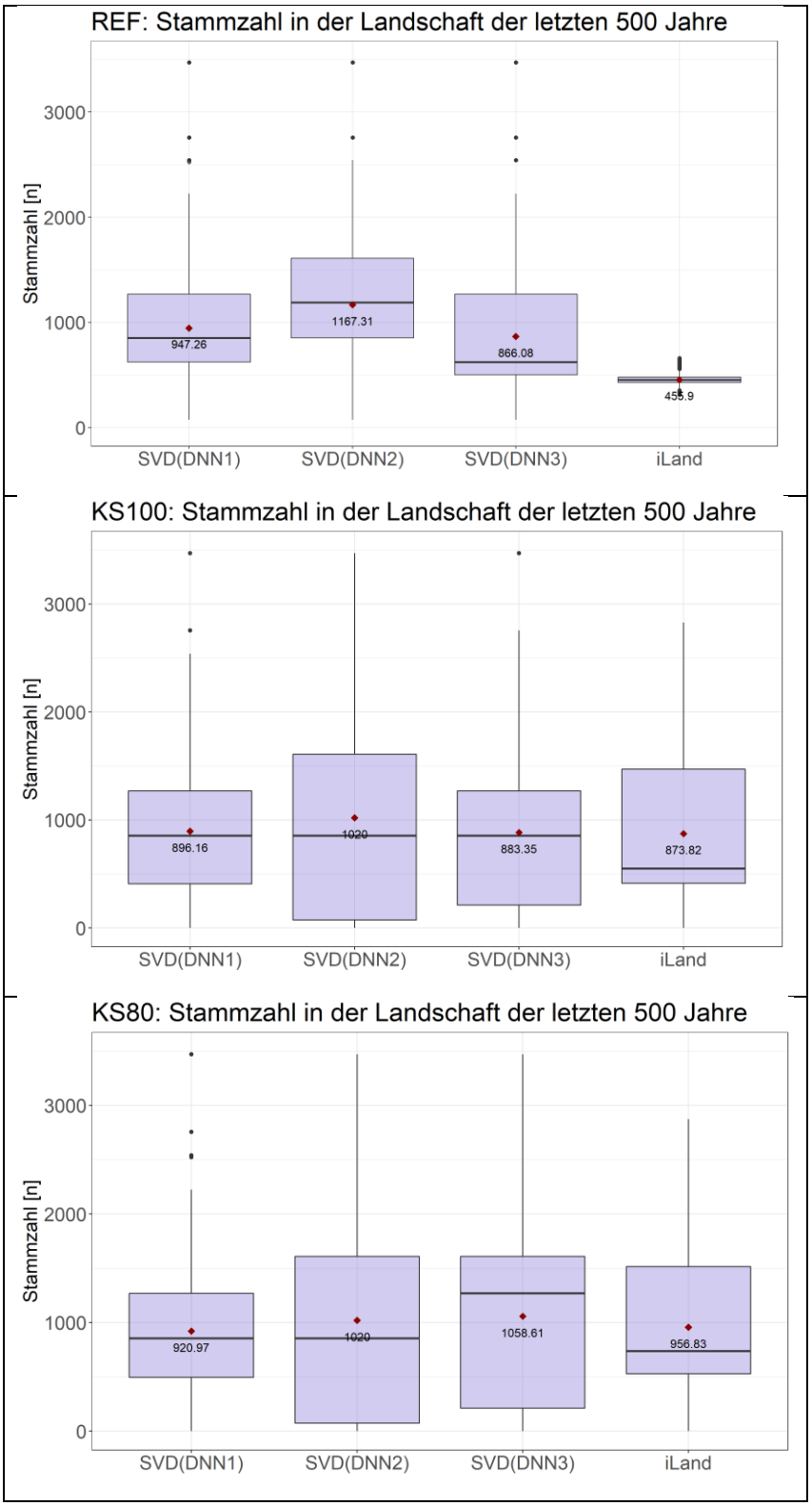
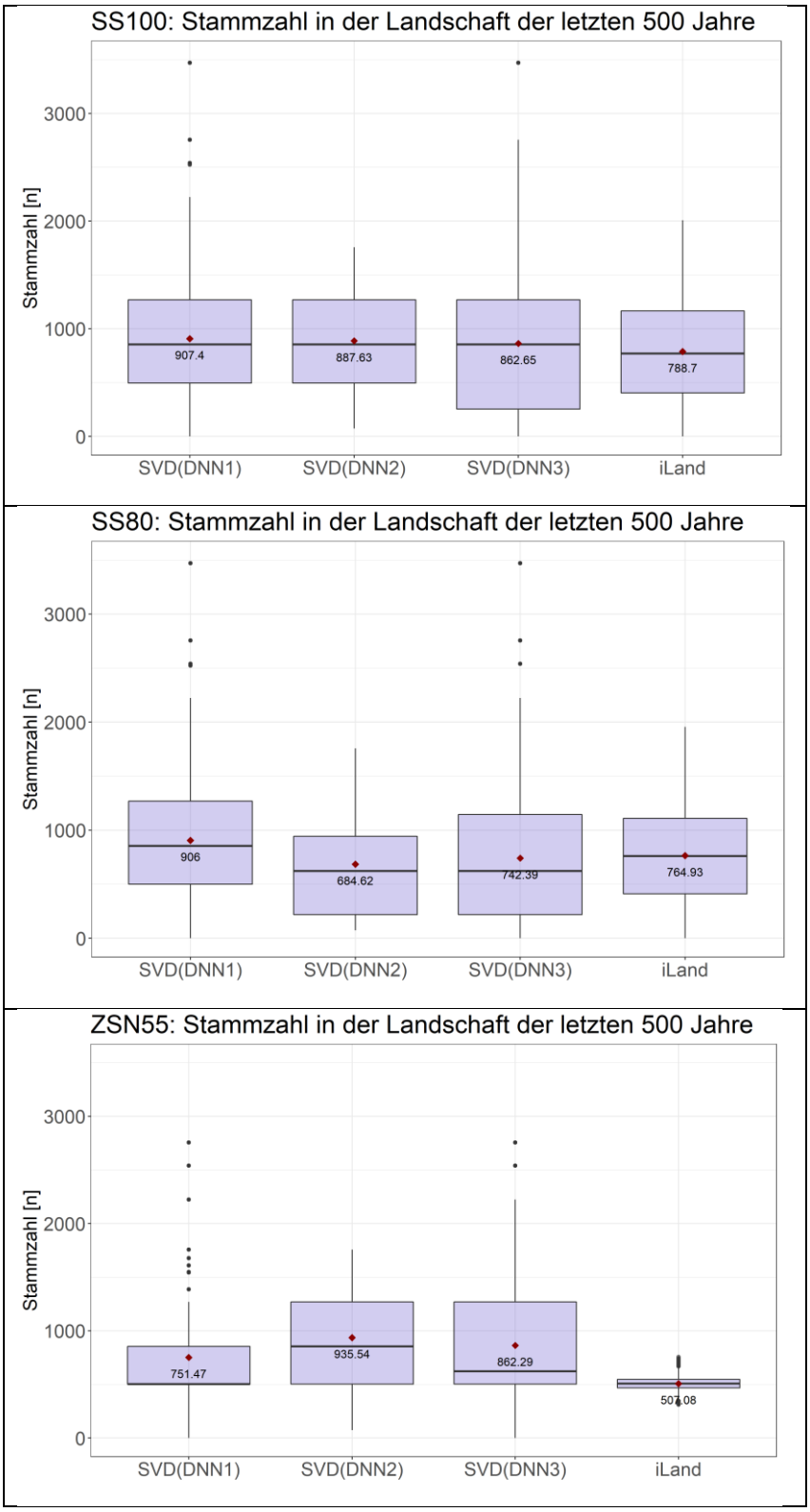


Abbildung 41: Gegenüberstellung SVD/iLand: Grundfläche

10.8 STAMMZAHL




SS100: Stammzahl in der Landschaft der letzten 500 Jahre
Stammzahl [n]
3000
2000
1000
0
907.4
887.63
862.65
788.7
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: Stammzahl in der Landschaft der letzten 500 Jahre
Stammzahl [n]
3000
2000
1000
0
906
684.62
742.39
764.93
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN55: Stammzahl in der Landschaft der letzten 500 Jahre
Stammzahl [n]
3000
2000
1000
0
751.47
935.54
862.29
507.08
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

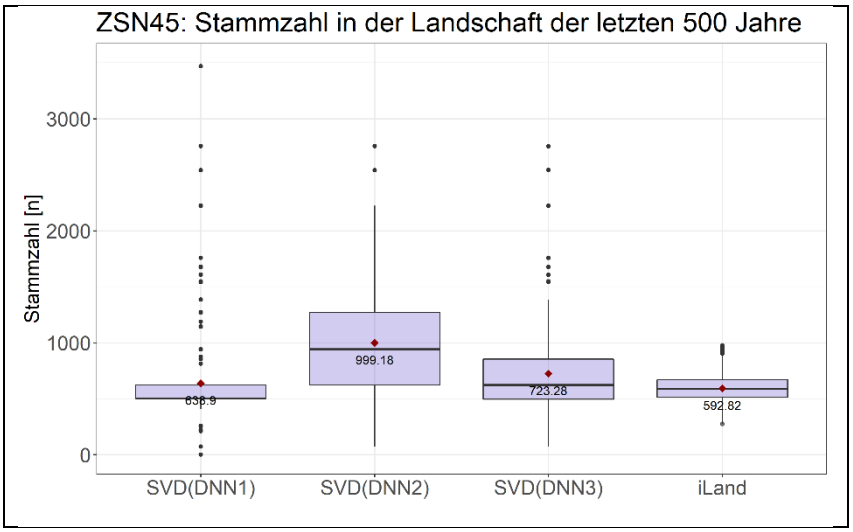
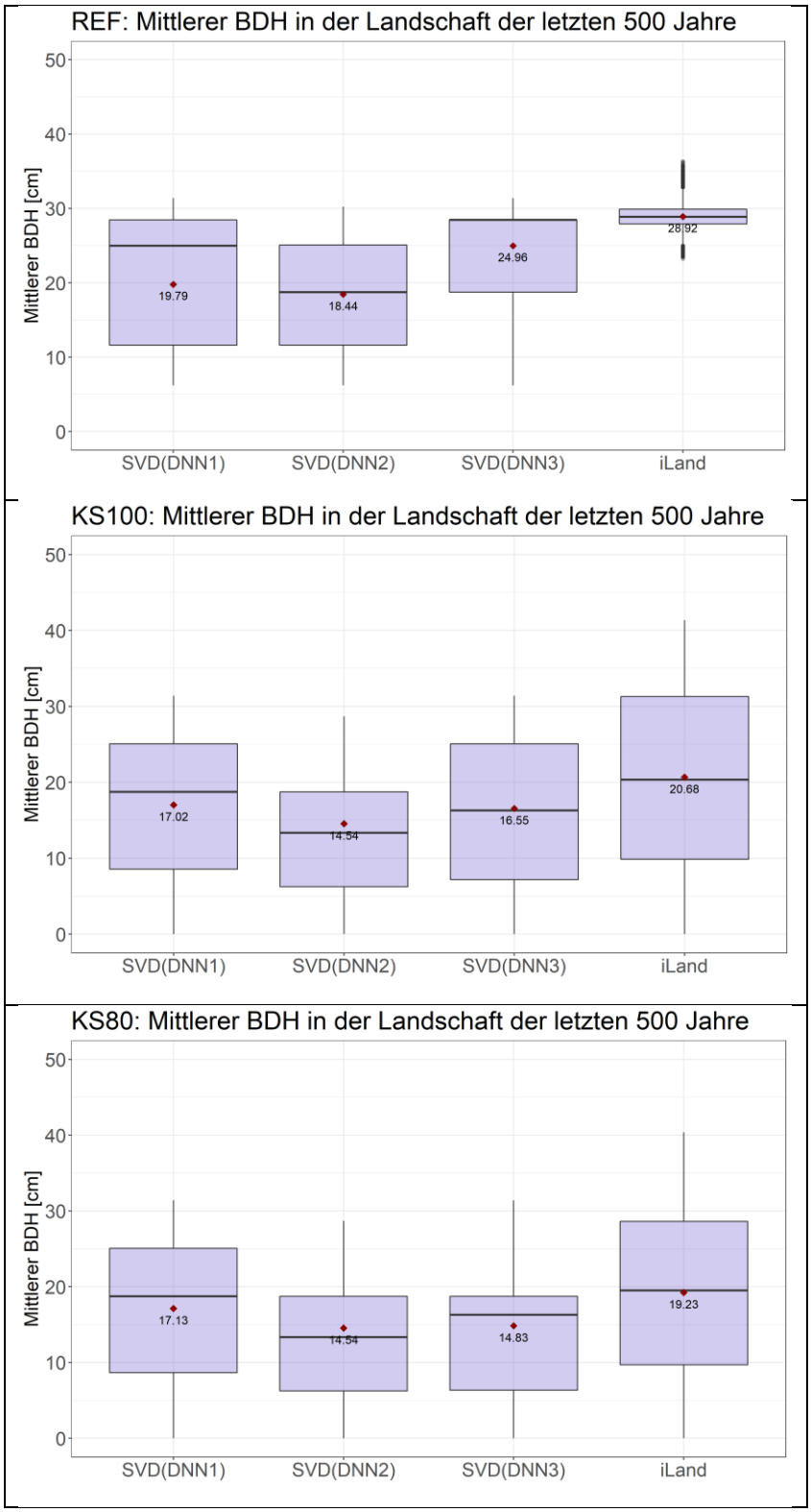
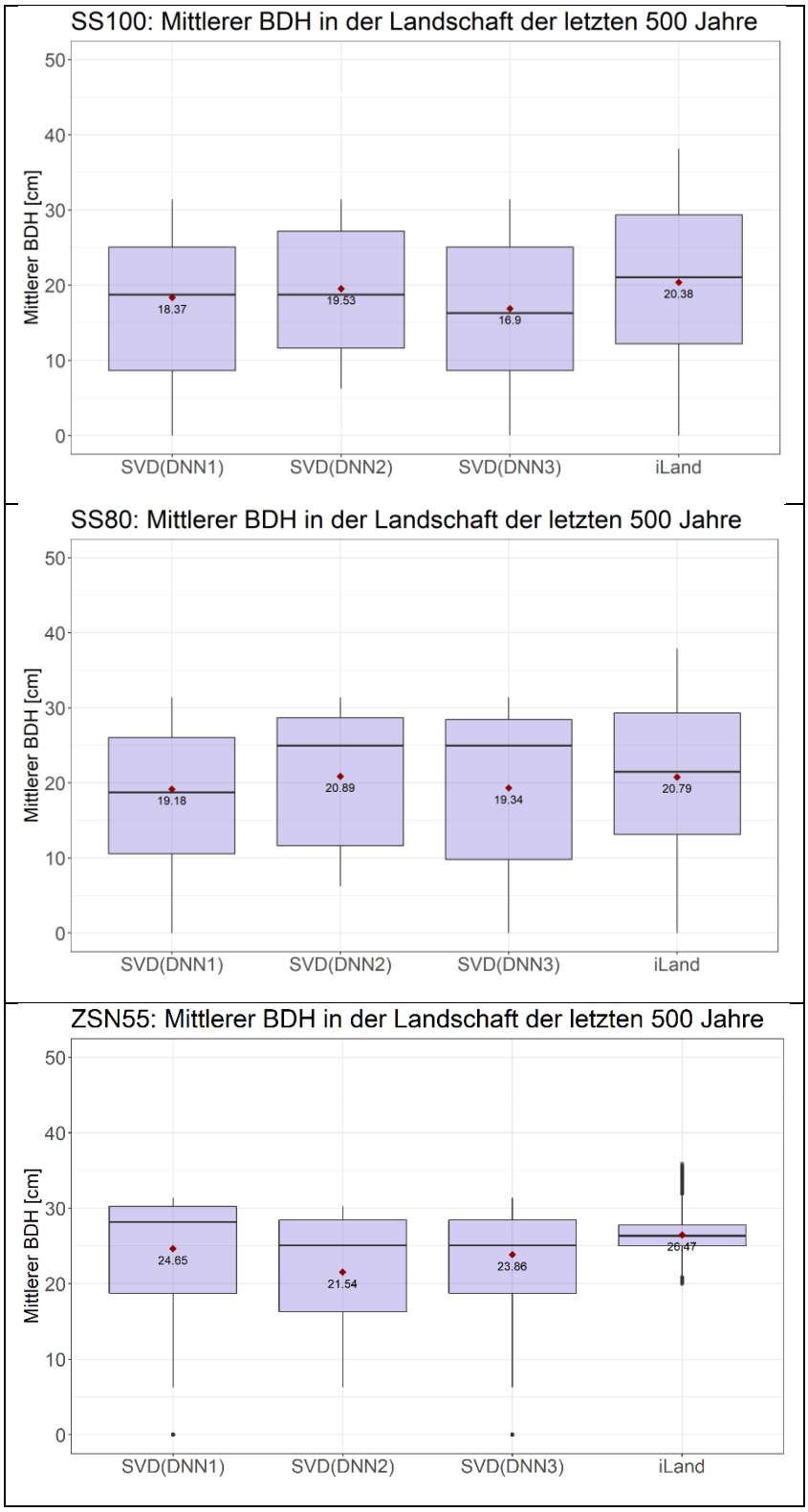


Abbildung 42: Gegenüberstellung SVD/iLand: Stammzahl

10.9 BHD


REF: Mittlerer BDH in der Landschaft der letzten 500 Jahre
Mittlerer BDH [cm]
0
10
20
30
40
50
19.79
18.44
24.96
28.92
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
KS100: Mittlerer BDH in der Landschaft der letzten 500 Jahre
Mittlerer BDH [cm]
17.02
14.54
16.55
20.68
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
KS80: Mittlerer BDH in der Landschaft der letzten 500 Jahre
Mittlerer BDH [cm]
17.13
14.54
14.83
19.23
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand


SS100: Mittlerer BDH in der Landschaft der letzten 500 Jahre
Mittlerer BDH [cm]
0
10
20
30
40
50
18.37
19.53
16.9
20.38
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
SS80: Mittlerer BDH in der Landschaft der letzten 500 Jahre
Mittlerer BDH [cm]
0
10
20
30
40
50
19.18
20.89
19.34
20.79
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand
ZSN55: Mittlerer BDH in der Landschaft der letzten 500 Jahre
Mittlerer BDH [cm]
0
10
20
30
40
50
24.65
21.54
23.86
26.47
SVD(DNN1)
SVD(DNN2)
SVD(DNN3)
iLand

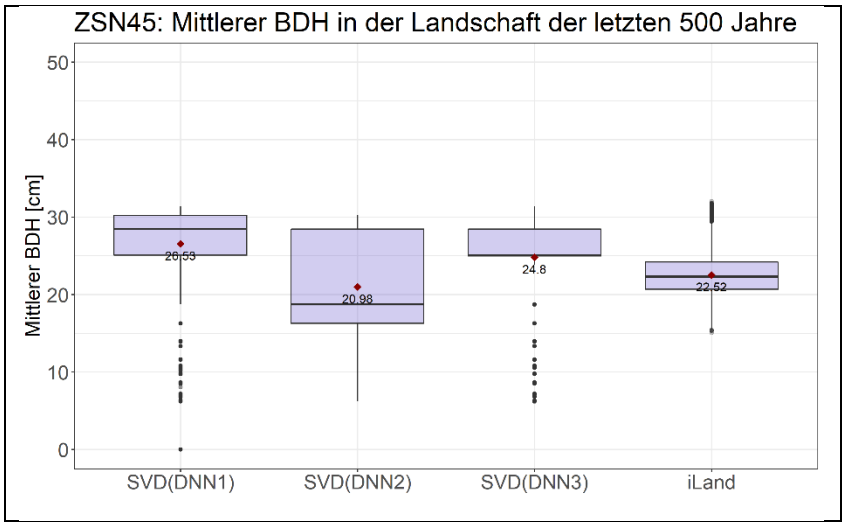


Abbildung 43: Gegenüberstellung SVD/iLand: Mittlerer BHD

11 CODEAUSSCHNITTE

11.1 DIE MANAGEMENTS IN JAVASCRIPT

```
/*
    author: Matthias Steinparzer
    date: July 2019

    Script for the six different stand treatment programms for the testlandscape with 100% spruce
    containing: - clearcut with a rotation age of 100 years
                - clearcut with a rotation age of 80 years
                - strip selection cut with a rotation age of 100 years
                - strip selection cut with a rotation age of 80 years
                - target diameter harvest with a target dbh of 55 cm
                - target diameter harvest with a target dbh of 45 cm
*/

// create a logfile variable
var logfile = "mgmtactivity" + "," + "year" + "," + "standId";

// iLand-logging
function writeLine(activity, year, standId) {
    fmengine.log("SVDLOG: " + activity + "-" + year + "-" + standId);
}

// appending a new line to the logfile for every action happened
function log(activity, year, standId) {
    logfile = logfile + "\n" + activity + "," + year + "," + standId;
}

// global function onYearEnd: called by iLand every year
function onYearEnd() {
    if (Globals.year == 10)
        fmengine.log("year 10 reached");
}

// planting of the new stand
var _planting = { type: 'planting',
    schedule: 1,
    items: [{species: 'piab', fraction: 1.0, height: 0.3}],

    onExecuted: function() { writeLine("pl", Globals.year, stand.id), log("pl", Globals.year, stand.id); }
};

// tending: According to Albrich(2016)
var _tending = { type: 'thinning',
    schedule: { min: 5, opt: 10, max: 15, absolute: false },
    thinning: 'custom',
    //remove 30 % of the stems
    targetValue: 30, targetVariable: 'stems', targetRelative: true,
    classes: [100],

    onExecuted: function() { writeLine("td", Globals.year, stand.id), log("td", Globals.year, stand.id); }
};

```

```
// thinning 1: According to Albrich(2016), Seidl (2007a)
var _thinning1 = { type: 'thinning',
    schedule: { min: 35, opt: 37, max: 40, force: true },
    thinning: 'custom', // constraint: ['stand.topHeight>12'],
    targetValue: 30, targetVariable: 'volume', targetRelative: true,
    minDbh: 10, //
    classes: [0, 15, 15, 45, 25], // thinning from above, considering the 5 dbh classes

    onExecuted: function() { writeLine("th1", Globals.year, stand.id), log("th1", Globals.year, stand.id); }
};

// thinning 2: According to Albrich(2016), Seidl (2007a)
var _thinning2 = { type: 'thinning',
    schedule: { min: 45, opt: 47, max: 50, force: true },
    thinning: 'custom',
    targetValue: 30, targetVariable: 'volume', targetRelative: true,
    minDbh: 10,
    classes: [0, 15, 15, 45, 25], // thinning from above

    onExecuted: function() { writeLine("th2", Globals.year, stand.id), log("th2", Globals.year, stand.id); }
};

// thinning 3: According to Albrich(2016), Seidl (2007a)
var _thinning3 = { type: 'thinning',
    schedule: { min: 55, opt: 57, max: 60, force: true },
    thinning: 'custom',
    targetValue: 30, targetVariable: 'volume', targetRelative: true,
    minDbh: 10,
    classes: [0, 15, 15, 45, 25], // thinning from above

    onExecuted: function() { writeLine("th3", Globals.year, stand.id), log("th3", Globals.year, stand.id); }
};

// clearcut
var _clearcut = { type: 'scheduled',
        schedule: { minRel: 0.90, optRel: 1, maxRel: 1.15, force: true },
        onEvaluate: function(){ // fmengine.log("finalHarvest:" + activity.finalHarvest);
                        return true; },
        onExecute: function() {
            trees.simulate = false;
            trees.loadAll(); trees.harvest("dbh>5");
            trees.simulate = true;
            }, //does this make any difference?
        onCreate: function() { activity.finalHarvest=true; },

        onExecuted: function() { writeLine("cc", Globals.year, stand.id), log("cc", Globals.year, stand.id); }
};

// strip selection cut, rotation age is 100 years: 20 and 10 years before rotation age is reached; 1 tree length
var _ssc1_100 = { type: 'scheduled',
    schedule: { minRel: 0.70, optRel: 0.80, maxRel: 0.90, force: true },
```

```
103         onEvaluate: function(){ // filter tree coordinates here, first third
104             trees.loadAll(); trees.filter('species=piab and mod(x,100) <= 33'); trees.harvest(); return true; },
105         onExecute: function() { trees.removeMarkedTrees(); fmengine.log("calling sleep ssc1_100"); stand.sleep(10);},
106 
107         onExecuted: function() { writeLine("ssc1_100", Globals.year, stand.id), log("ssc1_100", Globals.year, stand.id); }
108     };
109 
110     var _ssc2_100 = { type: 'scheduled',
111         schedule: { minRel: 0.80, optRel: 0.90, maxRel: 1.0, force: true },
112         onEvaluate: function(){
113             trees.loadAll(); trees.harvest('mod(x,100) > 33 and mod(x,100) <= 66'); return true; }, // second third of the unit
                gets selcted
114         onExecute: function() { trees.removeMarkedTrees(); fmengine.log("calling sleep ssc2_100"); stand.sleep(10);},
115 
116         onExecuted: function() { writeLine("ssc2_100", Globals.year, stand.id), log("ssc2_100", Globals.year, stand.id); }
117     };
118 
119     // pseudo-clearcut, eliminating the issue, that in the beginning of the
120     // simulation whole ressource units are being clearcutted, just because the
121     // were initialised with an age between 90 % and 115 % of the nominal
122     // rotation age -> see clearcut variable. This applies not only to saum_u100,
123     // but also saum_u80, which was tested positively in iLand
124     // Pseudo-Clearcuts are following JS variables: _ssc3_100 and _ssc3_80
125     var _ssc3_100 = { type: 'scheduled',
126         schedule: { minRel: 0.90, optRel: 1.0, maxRel: 1.10, force: true },
127         onEvaluate: function(){
128             // third third of the unit gets selcted and cut
129             trees.loadAll(); trees.harvest('mod(x,100) > 66 and mod(x,100) <= 100'); return true; },
130         onExecute: function() { trees.removeMarkedTrees(); fmengine.log("calling sleep ssc3_100"); stand.sleep(10);},
131         onCreate: function() { activity.finalHarvest=true; },
132 
133         onExecuted: function() { writeLine("ssc3_100", Globals.year, stand.id), log("ssc3_100", Globals.year, stand.id); }
134     };
135 
136     // strip selection cut, rotation age is 80 years: 20 and 10 years before rotation age is reached;
137     // 1 tree length
138     var _ssc1_80 = { type: 'scheduled',
139         schedule: { minRel: 0.63, optRel: 0.75, maxRel: 0.87, force: true },
140         onEvaluate: function(){
141             // cut the first tree length of the unit
142             trees.loadAll(); trees.harvest('mod(x,100) <= 33'); return true; },
143         onExecute: function() { trees.removeMarkedTrees(); fmengine.log("calling sleep ssc1_80"); stand.sleep(10);},
144 
145         onExecuted: function() { writeLine("ssc1_80", Globals.year, stand.id), log("ssc1_80", Globals.year, stand.id); }
146     };
147 
148     var _ssc2_80 = { type: 'scheduled',
149         schedule: { minRel: 0.76, optRel: 0.88, maxRel: 1.0, force: true },
150         onEvaluate: function(){
151             // cut the second tree length of the unit
152             trees.loadAll();  trees.harvest('mod(x,100) > 33 and mod(x,100) <= 66'); return true; },
```

```
153      onExecute: function() { trees.removeMarkedTrees(); fmengine.log("calling sleep ssc2_80"); stand.sleep(10);},
154
155      onExecuted: function() { writeLine("ssc2_80", Globals.year, stand.id), log("ssc2_80", Globals.year, stand.id); }
156  };
157
158  // pseudo-clearcut
159  var _ssc3_80 = { type: 'scheduled',
160      schedule: { minRel: 0.88, optRel: 1.0, maxRel: 1.12, force: true },
161      onEvaluate: function(){
162          // cut the third tree length of the unit
163          trees.loadAll();  trees.harvest('mod(x,100) > 66 and mod(x,100) <= 100'); return true; },
164      onExecute: function() { trees.removeMarkedTrees(); fmengine.log("calling sleep ssc3_80"); stand.sleep(10);},
165      onCreate: function() { activity.finalHarvest=true; },
166
167      onExecuted: function() { writeLine("ssc3_80", Globals.year, stand.id), log("ssc3_80", Globals.year, stand.id); }
168  };
169
170  // target diameter harvest dbh 55 cm: 8 year intervall
171  var _tdh55 = { type: 'general',
172      schedule: { repeat: true, repeatInterval: 8 },
173      action: function(){
174          trees.loadAll();
175          var total_volume = trees.sum('volume');
176          trees.filter('species = piab and dbh>=55'); // filter all spruce >= 55 cm dbh
177          trees.randomize(); // get of the selection bias
178          // and take max. 20 % of the stand volume
179          trees.filter('incsum(volume) < ' + total_volume * 0.2 );
180          trees.harvest();
181          writeLine("tdh55", Globals.year, stand.id), log("tdh55", Globals.year, stand.id);
182          return true;},
183  };
184
185  // target diameter harvest dbh 45 cm: 5 year intervall
186  var _tdh45 = { type: 'general',
187      schedule: { repeat: true, repeatInterval: 5 },
188      action: function(){
189          trees.loadAll();
190          var total_volume = trees.sum('volume');
191          // filter all spruce >= 45 cm dbh
192          trees.filter('species = piab and dbh>=45');
193          trees.randomize(); // get of the selection bias
194          // and take max. 20 % of the stand volume
195          trees.filter('incsum(volume) < ' + total_volume * 0.2 );
196          trees.harvest();
197          writeLine("tdh45", Globals.year, stand.id), log("tdh45", Globals.year, stand.id);
198          return true;},
199  };
200
201  // definiton of various STPs
202  // clearcut-system with rotation age 100 years and 3 thinnings
203  fmengine.addManagement({ U: [90, 100, 110],
```

```
204       planting: _planting,
205       thinning1: _tending,
206       thinning2: _thinning1,
207       thinning3: _thinning2,
208       thinning4: _thinning3,
209       clearcut: _clearcut}, 'ks_u100');
210
211   // clearcut-system with rotation age 80 years and 2 thinnings
212   fmengine.addManagement({ U: [70, 80, 90],
213       planting: _planting,
214       thinning1: _tending,
215       thinning2: _thinning1,
216       thinning3: _thinning2,
217       clearcut: _clearcut}, 'ks_u80');
218
219   // strip-selection cut with rotation age 100 years and 3 strip selection cuts
220   // (10 years in between)
221   fmengine.addManagement({ U: [90, 100, 110],
222       ssc1_100: _ssc1_100,
223       ssc2_100: _ssc2_100,
224       ssc3_100: _ssc3_100, // pseudo-clearcut
225       }, 'saum_u100');
226
227   // strip-selection cut with rotation age 80 years and 3 strip selection cuts
228   // (10 years in between)
229   fmengine.addManagement({ U: [70, 80, 90],
230       ssc1_80: _ssc1_80,
231       ssc2_80: _ssc2_80,
232       ssc3_80: _ssc3_80, // pseudo-clearcut
233       }, 'saum_u80');
234
235   // target diameter harvest with dbh 55 cm and a 8 year action intervall
236   fmengine.addManagement({ U: [90, 100, 110],
237       tdh55: _tdh55}, 'ccf1');
238
239   // target diameter harvest with dbh 45 cm and a 5 year action intervall
240   fmengine.addManagement({ U: [90, 100, 110],
241       tdh45: _tdh45}, 'ccf2');
242
243   // create base_agent
244   var base_agent = {
245       scheduler: { enabled: true,
246                    minScheduleHarvest: 1,
247                    maxScheduleHarvest: 20,
248                    scheduleRebounceDuration: 5,
249                    maxHarvestLevel: 1.5,
250                    deviationDecayRate: 0.1,
251                    useSustainableHarvest: 1,
252                    harvestIntensity: 1
253       },
254       // add STPs to the agent
```

```
255       stp: {  'ks_u100': 'ks_u100', 'ks_u80': 'ks_u80', 'saum_u100': 'saum_u100',
256               'saum_u80': 'saum_u80', 'ccf1': 'ccf1', 'ccf2': 'ccf2',
257               'default': 'ks_u100'
258       } ,
259
260       newAgent: function() { var x= {  scheduler: this.scheduler,
261                       agent_updated: false };  return x; }
262   };
263
264   // register the agent type with ABE
265   fmengine.addAgentType(base_agent, 'agent_type');
266   // create a single agent of that type ('bau') with the name 'bau_single'.
267   // Individual stands are assigned to an agent in the agentDataFile.
268   fmengine.addAgent('agent_type', 'test_agent');
269
270   // helper function
271   function printObj( obj) {
272       console.log(JSON.stringify(obj, null, 4));
273   }
274
275   // function triggered at the end of the year by ABE
276   function onYearEnd() {
277       // to do something in a specific year, you can:
278       // save the buffered Logfile, as soon as the given year is reached
279       if (Globals.year==200 ) {
280           Globals.saveTextFile(Globals.path("temp/logfile.csv"), logfile);
281       }
282   }
283
```

11.2 Das DNN in Python mit Tensorflow

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 28 09:34:43 2019
Simulation von Waldbewirtschaftung mittels DNN (Deep Neural Networks)


@author: MatthiasS/WernerR
"""

# import several helper libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Embedding

# The GPU id to use, usually either "0" or "1"; GPU1
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID";
os.environ["CUDA_VISIBLE_DEVICES"]="1";

print(tf.__version__)
tf.test.is_gpu_available()
tf.test.gpu_device_name()

#------------------------------------------------------------------
# allow_soft_placement for erasing new error when running the model with an embedding layer
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
sess = tf.Session(config=tf.ConfigProto(allow_soft_placement=True, log_device_placement=True))

# Choose which network should be trained:
# False: single managments
# True: management regimes
OPT_Regime = True

# set the right path for the data of the data-pipeline
filename =
["/home/msteinparzer/DA/iLand/03_training/generic_lva_mgmt/iland/iland_data/all_examples_mit_Irr/train_data_subset_" +
str(x+1) + ".csv" for x in range(49)]
filename_eval =
["/home/msteinparzer/DA/iLand/03_training/generic_lva_mgmt/iland/iland_data/all_examples_mit_Irr/eval_data_subset_" +
str(x+1) + ".csv" for x in range(49)]


## 250 columns:
# 0..7: state, restime, distance, nitrogen, sand, mgmtIn, mgmtCode, regime
# 248,249: targetState, targetTime
```

```
47  # 8-247: climate 120x temp (8:127), 120x precip (128:247)
48
49  record_defaults = [tf.int32], [tf.float32]*2, [tf.float32]*2, [tf.int32]*3, [tf.float32] * 240, [tf.int32]*2
50  record_defaults = [item for sublist in record_defaults for item in sublist]
51
52  ## use multiple arguments with *
53  # parse e.g. tell the DNN how to translate the provided data from the data-pipeline, select the right columns and convert
    into new dimensions,
54  # shapes and datatypes
55  def _parse_function(*example):
56       ex = ({'state': tf.reshape(example[0]-1,[1]),
57              'restime': tf.cast(tf.reshape(example[1]/10, [1]), dtype=tf.float32),
58              'site': tf.cast(tf.reshape([x/100 for x in example[3:5]], [2]), dtype=tf.float32), # nitrogen, sand
59              'mgmt': tf.cast(tf.reshape([x/10 for x in reversed(example[5:7])], [2]),  dtype=tf.float32), # mgmtCode and mgmtIn
60              'regime': tf.cast(tf.reshape(example[7],[1]), dtype=tf.float32),
61              'climate': tf.concat([tf.reshape(tf.transpose([ x/10. for x in example[8:128] ]), [ 10, 12]), # tempperature
62                                    tf.reshape(tf.transpose([x/20. for x in example[128:248]]), [ 10, 12])], 1), # precipitation
63              },
64              { 'targetState': tf.reshape(example[248]-1, [1]),
65               'targetTime': tf.reshape(example[249]-1, [1])})
66       return ex
67
68
69  ## set up the data-pipeline for the DNN. The input-data is again shuffled and batched in smaller packages to process. One
    dataset is created for the training and
70  # one for the evaluation. The pipeline is also cached for faster processing
71
72  dataset_f = tf.data.experimental.CsvDataset(filename, record_defaults, header = True)
73  dataset = dataset_f.map(_parse_function,
    num_parallel_calls=3).cache("/tmp/msteinp/tfcache").apply(tf.data.experimental.shuffle_and_repeat(25600)).batch(256)
74
75  dataset_eval_f = tf.data.experimental.CsvDataset(filename_eval, record_defaults, header = True)
76  dataset_eval = dataset_eval_f.map(_parse_function,
    num_parallel_calls=2).apply(tf.contrib.data.shuffle_and_repeat(25600)).batch(256)
77
78
79  ################### To test the parse-function ######################
80  ####################################################################
81  sess = tf.Session()
82  next_elem = dataset_f.make_one_shot_iterator().get_next()
83  example = sess.run(next_elem)
84  parsed = _parse_function(*example) # call manually
85  testvar =sess.run(parsed)
86  testvar = sess.run(parsed[0]['mgmt'])
87
88
89  from tensorflow.keras.layers import Input,    concatenate,    TimeDistributed
90  from tensorflow.keras.layers import Dense, Dropout,  Flatten,  RepeatVector, Reshape
91  from tensorflow.keras.callbacks import TensorBoard, ReduceLROnPlateau, ModelCheckpoint
92
```

```
93
94  ###################### Define the DNN ##############################
95  # classes, layers, embeddings, dropouts, inputs, outputs, optimizers
96  # learing rate, tensorboard, modelfitting, ....
97  ####################################################################
98
99  # definition of output classes
100 NClasses = 45 # actually only (41) state classes, but more might follow
101 NTimeSteps = 10 # fixed for 10 different time classes
102
103 # Inputs: state, time, climat, site, mgmt, regime
104 sinput = Input(shape=(1,),dtype="int32",   name='state')
105 timeinput = Input(shape=(1,), name='restime')
106 climinput = Input(shape=(10,24,), name='climate')
107 siteinput = Input(shape=(2,), name='site')
108 mgmtinput = Input(shape=(2,), name='mgmt')
109 regimeinput = Input(shape=(1,), name='regime')
110
111 # layer for environmental parameters
112 clim = TimeDistributed(Dense(64, activation="relu"))(climinput)
113 clim = Reshape( (640,) )(clim)
114
115 # embedding layer for the state input
116 sinput_em = Embedding(output_dim=32, input_dim=NClasses, input_length=1, name="state_em")(sinput)
117 sinput_em = Reshape( (32,) )(sinput_em)
118
119 envx = Dense(64, activation="relu", name="envx")(clim)
120 envx = Dense(16, activation="relu", name="envx2")(envx)
121
122 # depending on the DNN training goal choose wether to concatenate the inputs for single management DNN training
123 # or for management regime DNN training
124 if OPT_Regime:
125     minput = concatenate([sinput_em, timeinput, envx, siteinput, mgmtinput, regimeinput])
126 else:
127     minput = concatenate([sinput_em, timeinput, envx, siteinput, mgmtinput])
128
129 c_time = Dense(256, activation="relu")(minput)
130 c_time = Dropout(0.25)(c_time)
131 c_time = Dense(256, activation="relu")(c_time)
132
133 out_time = Dense(NTimeSteps, name="targetTime", activation="softmax")(c_time)
134
135 x = Dense(256, activation="relu")(minput)
136 x = Dropout(0.25)(x)
137 x = Dense(256, activation="relu")(x)
138 x = Dropout(0.2)(x)
139 x = Dense(256, activation="elu")(x)
140
141 out = Dense(NClasses, activation="softmax", name="targetState")(x)
142
```

```
# depending on the DNN training goal choose to proper list for the input setup
if OPT_Regime:
    input_list = [sinput, climinput, timeinput, siteinput, mgmtinput, regimeinput]
else:
    input_list = [sinput, climinput, timeinput, siteinput, mgmtinput]

# set the distinct inputs and outputs for the model
model = Model(inputs=input_list, outputs=[out, out_time]) #

# set the loss function and loss weights for and the optimizer for the model
model.compile(loss={'targetState':'sparse_categorical_crossentropy', 'targetTime': 'sparse_categorical_crossentropy'},
              loss_weights={'targetState':1, 'targetTime': 0.5},
              optimizer=tf.keras.optimizers.Adam(lr=0.001),  metrics=['accuracy', 'sparse_categorical_accuracy'])

# set up tensorboard for logging the training
tbCallBack = TensorBoard(log_dir='/home/msteinparzer/DA/Python/temp/v36_superNetzwerk')

# reduce the learining rate if a plateau is reached after two epochs
tbReduceLr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
              patience=2, min_lr=0.00001)

## start the training
# Validation steps have been reduced to 1/10 of the original stepsize, due to missing caching in the evaluation dataset.
Randomisation of the evaluation dataset
# should be good enough in terms ofe val-data-noise; after 35 epochs there is no major change in the learning curve
model.fit(dataset, validation_data=dataset_eval, epochs=35, steps_per_epoch=49*400, validation_steps=49*10,
callbacks=[tbCallBack, tbReduceLr])

# get an overview of the model-setup
model.summary()



############## Create predition dataset for analysis ############
################################################################
import numpy as np

dataset_predict = tf.data.experimental.CsvDataset(filename_eval, record_defaults)
dataset_predict = dataset_eval_f.map(_parse_function, num_parallel_calls=2).batch(256)

# eval_data_subset = 24 606 rows, with 256 batch-size equals 96*40  = 3840
x = model.predict(dataset_predict, steps=96*48)

# save the predicted distribution outputs for the states and time
np.savetxt("Predict/eval_result_v36_superNetzwerk_state.txt", x[0]) # save states
np.savetxt("Predict/eval_result_v36_superNetzwerk_time.txt", x[1]) # save time
```

```
191  ################# Save the model  ##################
192  # (1) simple save-approach from the tensorflow API
193  # (2) save in the .h5 format
194  ####################################################
195  ## (1)
196  # Save the tf.keras model in the SavedModel format.
197  saved_to_path = tf.contrib.saved_model.save_keras_model(
198        model, 'simple_keras_saved_model_test_landscape_v36_superNetzwerk')
199
200  # Load the saved keras model back.
201  model_restored = tf.contrib.saved_model.load_keras_model(saved_to_path)
202  model_restored.summary()
203
204  # Run the resored model.
205  model_restored.compile(loss={'targetState':'sparse_categorical_crossentropy', 'targetTime': 'sparse_categorical_crossentropy'},
206                loss_weights={'targetState':1, 'targetTime': 0.5},
207                optimizer=tf.keras.optimizers.Adam(lr=0.001),  metrics=['accuracy', 'sparse_categorical_accuracy'])
208
209
210  ## (2) .h5 format
211  model.save('model_saves/test_landscape_v36_superNetzwerk.h5')
212  # load it again
213  new_model = keras.models.load_model('model_saves/test_landscape_v36_superNetzwerk.h5')
214  new_model.summary()
215
216
217
```