Techniken zur Standardisierung der Schicht-Inputs Neuronaler Netze

Moritz Kronberger IA4

Studienarbeit für "Neuronale Netze und Deep Learning" (Sommersemester 2021)

1 Einleitung

Tiefe Architekturen Neuronaler Netze (NNs) führen heutzutage beinahe alle Wettbewerbe der Bilderkennung oder sequentiellen Aufgaben wie Sprachverarbeitung an, jedoch ist das Training dieser Netze mit den herkömmlichen Methoden geplagt von langen Trainingszeiten, hoher Sensitivität gegenüber der Lernrate und dem Phänomen des Vanishing und Exploding Gradient.
Abhilfe schaffen hierbei verschiedenen Methoden zur Standardisierung der Inputs der versteckten Schichten, allen voran die wohl bekannteste Standardisierungstechnik *Batch Normalization* (Batch Norm oder BN) (Ioffe & Szegedy, 2015).
In dieser Arbeit soll zunächst die Motivation hinter diesen Techniken unter dem Begriff des *Internal Covariate Shift* (ICS) genauer definiert und erläutert werden. Anschließend soll eine kurze Erklärung von Batch Normalization eine Intuition für den Ablauf und die Grundgedanken hinter den verschiedenen Standardisierungsverfahren geben.
Nach einem kurzen Anriss einer der ersten Nachfolgetechniken von BN, *Layer Normalization* (Layer Norm) (Ba, Kiros & Hinton, 2016), widmet sich der Hauptteil der Arbeit einem recht speziellen Verfahren zur Standardisierung der Neuronen-Inputs: *Self Normalizing Neural Networks* (SNNs) (Klambauer, Unterthiner, Mayr & Hochreiter, 2017). Hierbei soll zunächst eine Intuition zur Funktionsweise dieser Netze gegeben werden, bevor anschließend auf einige der mathematischen Hintergründe von SNNs eingegangen wird.
Abschließend sollen einige neuere Erkenntnisse über den *Zusammenhang von ICS und den beschriebenen Techniken* (Santurkar, Tsipras, Ilyas & Mądry, 2018) Erwähnung finden.

2 Grundproblematik

2.1 Covariate Shift

Bei NNs sind bekanntermaßen die Aktivierungen der versteckten Schichten stark von den Aktivierungen ihrer Vorgängerschichten abhängig. Diese Abhängigkeit kann etwas vereinfacht werden, indem die Ebene des gesamten Netzes betrachtet wird:
Das Netz ist hierbei eine Abbildung h, die einen Input X auf einen Output Y abbildet $h : X \mapsto Y$. Ziel des Trainings ist es, diese an die ideale Abbildung h^* anzunähern, welche jeden Input $x \in X$ auf eine korrekte Ausgabe $y \in Y$ abbildet. Weicht beim Training nun die Verteilung der Features in den Trainingsdaten $p_{train}(x|y)$ zu stark von der Verteilung der Features der Testdaten $p_{test}(x|y)$ ab, ist die vom Netz erlernte Abbildung h, welche lediglich auf Basis der Trainingsdaten ermittelt wurde, kaum noch auf die Testdaten anwendbar.
Dieses Problem wird als *Covariate Shift* bezeichnet (Shimodaira, 2000).
Abbildung 1 veranschaulicht (hypothetisch) wie $p_{train}(x|y) \neq p_{test}(x|y)$ für eine erhebliche Abweichung der gelernten Abbildung h von der Idealfunktion h^* sorgen kann.

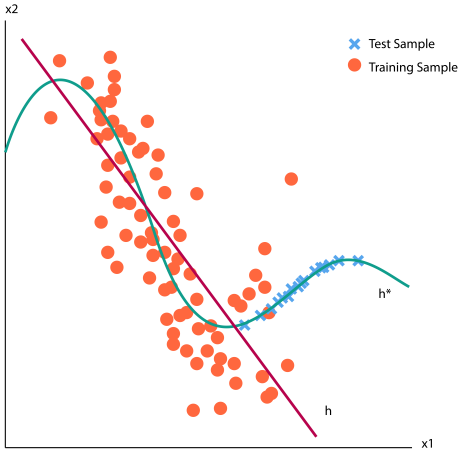

Abbildung 1:
Quelle: Masashi Sugiyama (S.22)

2.2 Internal Covariate Shift

Das Konzept des Covariate Shift, lässt sich auch auf die einzelnen Schichten eines NNs übertragen, wenn man sich eine versteckte Schicht als eine Art "Subnetz" mit ihrer Aktivierung als Output und der Aktivierung der Vorgängerschicht als Input vorstellt.
Es ist also zu erwarten, dass für jede Schicht ebenfalls eine Art Covariate Shift auftritt, wenn die Aktivierungen der Vorgängerschicht zu ungleich verteilt sind. Betrachtet man nun analog zur Test-/Trainingsdaten-Problematik den Lernprozess, kann man sich vorstellen, dass einer Schicht das Training enorm erschwert wird, wenn sie ständig auf die, durch das Training, veränderte Verteilung ihrer Inputs reagieren muss.
Dies könnte das Training nicht nur verlangsamen, sondern im schlimmsten Fall sogar das Finden eines Optimums verhindern. Je tiefer sich eine Schicht im Netz befindet, von desto mehr Vorgängerschichten sind ihre Rohinputs abhängig und desto größer sind vermutlich die Schwankungen in der Input-Verteilung. Für diese Problematik hat sich der Begriff des *Internal Covariate Shift (ICS)* herausgebildet (Ioffe & Szegedy, 2015).

3 Batch Normalization

3.1 Motivation

Besonders bei tiefen NNs erscheint es also lohnend den ICS zu reduzieren. Wenn ICS dabei für zu starke Schwankungen der Verteilungen der Rohinputs sorgt, könnte man dem entgegenwirken, indem die Inputs der versteckten Schichten während des Trainings laufend standardisiert werden.
Genau dies ist die Hauptmotivation hinter der Technik der *Batch Normalization* (Ioffe & Szegedy, 2015).
Einschränkend sei hier bereits zu erwähnen, dass auch wenn ICS sehr intuitiv erscheint und im originalen Paper als Hauptgrund für die Performance Verbesserungen durch Batch Norm angeführt wird, inzwischen Zweifel daran herrschen, in wieweit ICS tatsächlich ausschlaggebend für die Performance eines NNs ist (Santurkar et al., 2018). Nachdem ICS jedoch auch in den Weiterentwicklungen von BN weiterhin oft angeführt wird, ist eine Intuition für ICS dennoch für das Verständnis der Standardisierungstechniken hilfreich.

3.2 Herleitung

Um bei der Standardisierung der Rohinputs nicht die Repräsentationsfähigkeiten des NNs zu schmälern, sondern möglichst viele der Informationen im Netz, wie beispielsweise das Verhältnis der absoluten Werte, zu erhalten, werden die Rohinputs für ein Trainingsbeispiel relativ zu den Statistiken der gesamten Trainingsdaten standardisiert.
Nachdem eine solche Normalisierung über alle Rohinputs einer Schicht l nicht besonders performant wäre, wird jede Komponente $z_i^{(l)}$ des Rohinputs $z^{(l)}$ eines einzelnen Trainigsbeispiels (x^k, y^k) für sich standardisiert, sodass sie einen Mittelwert von 0 und eine Varianz von 1, also die Standardnormalverteilung, hat.
Für eines von N Trainingsbeispielen (x^k, y^k), $i = 1, ..., N$ im Datensatz X, berechnet sich die standardisierte Rohinput-Komponente $\tilde{z}_i^{(l),k}$, $i = 1, ..., d$ eines d-dimensionalen Rohinput-Vektor $z^{(l),k}$ in Schicht l durch:

$$\tilde{z}_i^{(l),k} = \frac{z_i^{(l),k} - E[z_i^{(l)}]}{\sqrt{Var[z_i^{(l)}] + \epsilon}}$$

mit dem Erwartungswert $E[z_i^{(l)}]$ und der Varianz $Var[z_i^{(l)}]$, ermittelt über alle N Trainingsbeispiele (x^k, y^k).
Die obige Formel ist in ihrer Grundfunktion bereits aus den *Feature-Scaling* bekannt. Die sehr kleine Konstante $\epsilon \neq 0$ fängt dabei den Fall $Var[z_i^{(l)}] = 0$ ab.
Tatsächlich werden $E[z_i^{(l)}]$ und $Var[z_i^{(l)}]$ nicht über die gesamten Trainigsdaten sondern nur näherungsweise über den jeweiligen Minibatch berechnet, da ersteres klassische Batchverarbeitung voraussetzen würde, welche bei modernen NNs kaum noch Anwendung findet.
Würden diese standardisierten Rohinputs nun bei der Berechnung der Aktivierung beispielsweise durch die Logistische Funktion (bzw. Sigmoidfunktion) verarbeitet, wäre die resultierende Menge der Aktivierungen aufgrund der Standardabweichung 1 und des Mittelwerts 0 größtenteils auf den linearen Teil der Funktion beschränkt. (vgl. Abbildung 2)

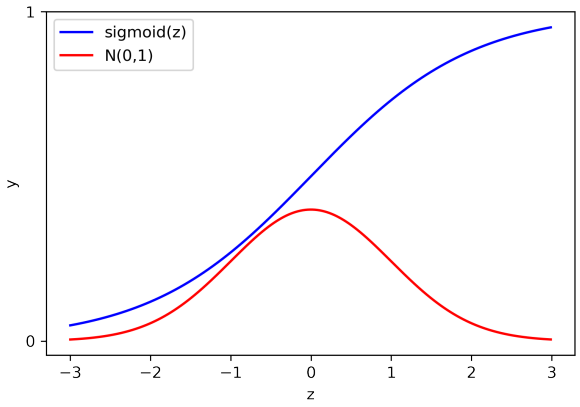

Abbildung 2

Um dies zu beheben wird für jeden Rohinput $\tilde{z}_i^{(l),k}$ ein Parameterpaar $\gamma_i^{(l)}$ und $\beta_i^{(l)}$ eingeführt, welche den standardisierten Rohinput nach dem bekannten Geradengleichungsschema beliebig skalieren und verschieben:

$$\hat{z}_i^{(l),k} = \gamma_i^{(l)} \tilde{z}_i^{(l),k} + \beta_i^{(l)}$$

Würden $\gamma_i^{(l)} = \sqrt{Var[z_i^{(l)}]}$ und $\beta_i^{(l)} = E[z_i^{(l)}]$ gesetzt, ließe sich sogar der ursprüngliche Rohinput $z_i^{(l),k}$ wiederherstellen.
Mit dem fertig transformierten Rohinput $\hat{z}_i^{(l),k}$ und der Aktivierungsfunktion $g(z)$ wird nun wie gewohnt die Aktivierung $a_i^{(l),k}$ berechnet.
Die Parameter $\gamma_i^{(l)}$ und $\beta_i^{(l)}$ werden dabei genau wie die Gewichte oder Biasneuronen durch Backpropagation mit Gradientenabstieg oder beliebigen Optimierungsmethoden gelernt.

Wird in einem NN Batch Norm verwendet, können die Biasneuronen allerdings vernachlässigt werden. Die Begründung hierfür ist relativ simpel:
Wie bisher üblich berechnet sich die Rohinput-Komponente $z_i^{(l),k}$ mit:

$$z_i^{(l),k} = \sum_{j=1}^{n_{l-1}} w_{i,j}^{(l-1)} a_j^{(l-1),k} + b_i^{(l-1)}$$

Das Biasneuron $b_i^{(l-1)}$ ist hierbei eine Konstante, die für alle Trainingsbeispiele in einer Epoche des Minibatches den gleichen Wert hat.
Bildet man nun den standardisierten Rohinput $\tilde{z}_i^{(l),k}$ mit:

$$\tilde{z}_i^{(l),k} = \frac{z_i^{(l),k} - E[z_i^{(l)}]}{\sqrt{Var[z_i^{(l)}] + \epsilon}},$$

so wird dabei der Mittelwert $E[z_i^{(l)}]$ berechnet:

$$E[z_i^{(l)}] = \frac{1}{N}\sum_{k=1}^{N} z_i^{(l),k} = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{n_{n-1}} w_{i,j}^{(l-1)} a_j^{(l-1),k} + b_i^{(l-1)}$$

Die Konstante $b_i^{(l-1)}$ wird dabei also herausgemittelt. Man kann sich vorstellen, dass deren Funktion bei BN durch den β-Parameter übernommen wird.

3.3 Batch Normalization Algorithmus

Für einen Mini-Batch $B \in X$ der Größe M existieren für die Rohinput-Komponente $z_i^{(l)}$ M Werte $z_i^{(l),k}$, $k = 1, ..., M$.
Deren standardisierte Werte $\tilde{z}_i^{(l),k}$ und deren Transformationen $\hat{z}_i^{(l),k}$ sollen nun mithilfe des Batch Normalization Algorithmus berechnet werden:

Für alle Rohinput-Komponenten $z_i^{(l)}$:
(l wird zur Übersicht vernachlässigt)

$$\mu_i := \frac{1}{M}\sum_{k=1}^{M} z_i^k$$

$$\sigma_i^2 := \frac{1}{M}\sum_{k=1}^{M}(z_i^k - \mu_i)^2$$

$$\tilde{z}_i^k = \frac{z_i^k - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$\hat{z}_i^k = \gamma_i\, \tilde{z}_i^k + \beta_i$$

3.4 Forward Propagation mit Batch Normalization

Einen groben Überblick über den Ablauf der Forward Propagation mit Batch Norm soll folgende Darstellung bieten:

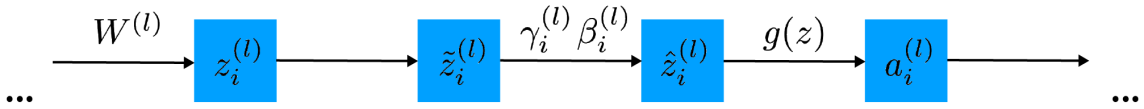

Abbildung 3:
Grafik vom Skript inspiriert

Obiger Batch-Norm-Algorithmus ist also als eine Art Zwischenschicht zwischen Rohinput und Aktivierung implementiert.

3.5 Vorhersagen mit Batch Norm

Eine erste Limitation von Batch Norm lässt sich bereits beim Berechnen der Vorhersagen erkennen.
Nachdem bei BN die Standardisierung immer über den jeweiligen Batch oder Mini-Batch erfolgt, ist eine Batch-Size $M > 1$ zwingend erforderlich um überhaupt standardisieren zu können. Für sehr kleine M liegt es nahe, dass die Standardisierung über den Minibatch nur sehr wenig repräsentativ für den Datensatz ist und damit hohen Schwankungen ausgesetzt ist.
Nun liegen jedoch bei den meisten Vorhersageszenarien die Daten nicht in großen Batches vor. Selbst wenn es je nach Anwendung möglich wäre die Vorhersage-Inputs in Batches zu sammeln, ist es nur sehr selten sinnvoll, bei der Vorhersage durch die Standardisierung einen einzelnen Input in Abhängigkeit anderer Inputs zu betrachten.
Die Standardisierung zur Vorhersage muss also angepasst werden:
Wurden beim Training K Minibatches B der Größe M verwendet gilt für die Standardisierung einer Rohinput-Komponente $z_i^{(l),k}$ weiterhin

$$\tilde{z}_i^{(l),k} = \frac{z_i^{(l),k} - E[z_i^{(l)}]}{\sqrt{Var[z_i^{(l)}] + \epsilon}}$$

jedoch nun mit
(l wird zur Übersicht vernachlässigt)

$$E[z_i] = E_B[\mu_{B,i}] = \frac{1}{K} \sum_{B=1}^{K} \mu_{B,i}$$

und

$$Var[z_i] = \frac{m}{m-1} E_B[\sigma_{B,i}^2]$$

mit

$$E_B[\sigma_{B,i}^2] = \frac{1}{K} \sum_{B=1}^{K} \sigma_{B,i}^2$$

Anstatt also Mittelwert und Standardabweichung über die Trainingsbeispiele des Batches zu berechnen, werden die finalen Mittelwerte und Standardabweichungen über die Minibatches gemittelt und als Konstanten gleichermaßen für alle Vorhersagen eingesetzt.
Der fertig standardisierte Rohinput $\hat{z}_i^{(l),k}$ kann so auch direkt berechnet werden:

$$\hat{z}_i^{(l),k} = \frac{\gamma}{\sqrt{Var[z_i^{(l)}] + \epsilon}} z_i^{(l),k} + (\beta - \frac{\gamma E[z_i^{(l)}]}{\sqrt{Var[z_i^{(l)}] + \epsilon}})$$

3.6 Vor- und Nachteile von Batch Normalization

Neben teilweise verbesserter Performance im Bezug auf die Vorhersagegenauigkeit des NNs auf den Testdaten, bringt Batch Norm weitere ausschlaggebende Verbesserungen mit sich:
Nachdem BN die Rohinputs jeder Schicht standardisiert, wird der Effekt von Vanishing oder Exploding Gradients, also dem Effekt, dass kleine Änderungen der Parameter über das Netz und den Trainigsverlauf hinweg in immer größere Abweichungen der Gradienten resultieren, eingedämmt. Bei Backpropagation mit BN scheint der Gradient sogar für zu große Parameter kleiner zu werden.
All das erlaubt beim Einsatz von Batch Norm das Wählen erheblich größerer Lernraten und beschleunigt damit den Lernprozess enorm. Nachdem das Netz mit BN deutlich schneller konvergiert, können auch bei Techniken wie L2-Regularisierung wesentlich aggressivere Einstellungen gewählt werden.
Dass durch die Standardisierung jedes Trainingsbeispiel vom Netz nicht mehr allein für sich bearbeitet, sondern in eine Abhängigkeit zu allen andern Trainigsbeispielen des Datensatzes gestellt wird, scheint zudem einen positiven, regualrisierenden Effekt auf das Netz zu haben, was es erlaubt, bei der Verwendung von BN Dropout zu reduzieren oder sogar ganz darauf zu verzichten.

Auch wenn Batch Norm inzwischen zum beinahe Standard beim Training tiefer NNs geworden ist, besitzt BN durchaus einige Limitierungen, welche von zahlreichen Nachfolgetechniken aufgegriffen werden.
Der wohl größte Nachteil von Batch Norm, ist die bereits erwähnte, direkte Abhängigkeit der Standardisierung von der Mini-Batch-Größe. Dies führt nicht nur dazu, dass der Effekt von BN für verschiedene Batchgrößen sehr unterschiedlich ausfallen kann, sondern macht BN mit einer Batch-Size $M = 1$ unmöglich. Bei der Vorhersage lässt sich dieser Umstand wie bereits beschrieben durch die Abwandlung der Standardisierung umgehen, bei reinem Stochastic Gradient Descent (SGD), beziehungsweise Online-Learning-Anwendungen bleibt der Einsatz von BN allerdings trotzdem ausgeschlossen. Zusätzlich gibt es keine klare Herangehensweise um Batch Norm über die Zeitschritte von Recurrent Neural Networks (RNNs) zu implementieren.

4 Layer Normalization

Vor allem letztere Limitierung ist eine ausschlaggebende Motivation hinter der Technik der *Layer Normalization* (Ba et al., 2016). Auch wenn RNNs in dieser Arbeit keine große Rolle spielen sollen, ist es dennoch interessant, die Layer Norm Technik zumindest oberflächlich zu betrachten, da sie sehr verständlich zeigt, dass der Vorgang der Standardisierung auch völlig losgelöst von der klassischen Herangehensweise über die Trainingsbeispiele erfolgen kann.
Layer Normalization hat gleichermaßen zum Ziel ICS durch die Standardisierung der Inputs der versteckten Schichten zu reduzieren. Um die Standardisierung der Rohinputs nun allerdings unabhängig von der Batch-Größe durchführen zu können trifft Layer Normalization eine entscheidende Veränderung gegenüber Batch Norm:

Die Standardisierung wird nicht für jede Komponente der Rohinput-Vektoren über die (Teil-) Menge der Trainingsbeispiele, sondern über alle Komponenten des Rohinputs einer versteckten Schicht berechnet.
Die Berechnung von Mittelwert und Standardabweichung für eine Schicht l mit n_l Rohinput-Komponenten $z_i,\ i = 1, ..., n_l$ geschieht also wie folgt:

$$\mu^{(l)} = \frac{1}{n_l} \sum_{i=1}^{n_l} z_i^{(l)}$$

$$\sigma^{2(l)} = \sqrt{\frac{1}{n_l} \sum_{i=1}^{n_l} (z_i^{(l)} - \mu^{(l)})^2}$$

Es teilen sich somit alle Rohinput-Komponenten einer Schicht die selbe Standardisierung, diese ist aber für verschiedene Trainingsbeispiele unterschiedlich. Layer Norm ist somit von etwaigen Batch-Größen unabhängig und kann daher problemlos mit reinem SGD, für Online Learning oder auch RNNs genutzt werden.
Insgesamt zeigt Layer Norm, dass zur Standardisierung der Schicht-Inputs keine Abhängigkeit zu den Trainigsbeispielen hergestellt werden muss, sondern diese auch lediglich über die zu standardisierenden Komponenten selbst erfolgen kann.

5 Self Normalizing Neural Networks

5.1 Motivation

Die anfängliche Motivation hinter *Self Normalizing Neural Networks (SNNs)* (Klambauer et al., 2017) war die schlechte Kompatibilität der klassichen Standardisierungstechniken wie Batch oder Layer Norm mit Feedforward Neural Networks (FNNs).
Die Kombination aus SGD, Regularisierungstechniken wie Dropout und dem Lernen der Standardisierungsparameter γ und β scheint dabei zu Schwankungen der Parameter beim Training zu führen, welche bei Convolutional Neural Networks (CNNs) oder RNNs durch *weight sharing* aufgefangen werden können, bei FNNs allerdings zu deutlich weniger effektivem Lernen führen.

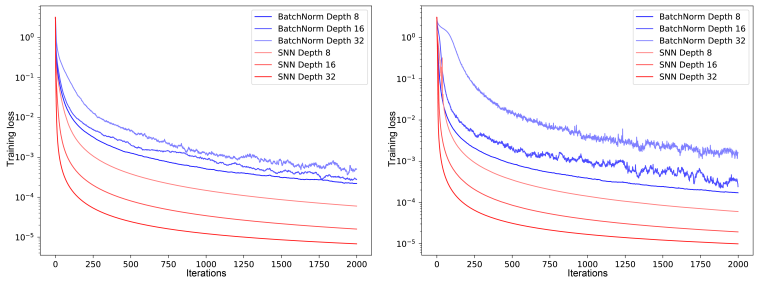

Abbildung 4: Vergleich des Trainings-Loss' von FNNs mit Batch Norm (blau) und SNNs (rot) bei MNIST (links) und CIFAR10 (rechts);
Die SNNs zeigen dabei nicht nur deutlich weniger Loss-Schwankungen sondern konvergieren zudem erheblich schneller.
(Klambauer et al., 2017)

Auch wenn FNNs kaum noch zu den populärsten Vertretern Neuronaler Netze zählen, scheinen sie eine der wenigen Deep-Learning-Methoden zu sein, welche abseits von Bilderkennungs- oder sequentiellen Aufgaben mit Techniken wie Support Vector Machines (SVMs), Random Forests o.ä. schritthalten können. Die kompetitiven FNN-Architekturen sind dabei wie zu erwarten nicht besonders tief, es ließe sich also vermuten, dass sich mit einer geeigneten Standardisierungstechnik tiefere, performantere FNNS trainieren ließen.
Neben diesem, doch etwas speziellen, Einsatzgebiet haben SNNs jedoch noch einen weiteren Vorzug:
Sowohl für Batch als auch Layer Norm werden zwangsläufig bei der Forward- und Backpropagation zusätzliche Verarbeitungsschritte und Parameter benötigt. Besonders elegant wäre es also, wenn ein NN automatisch durch die reguläre Vorwärtsverarbeitung dafür sorgen würde, dass die Inputs der versteckten Schichten die gewünschte Normalverteilung erlangen würden. Genau das ist das Ziel von Self Normalizing Neural Networks.

5.2 Intuition

Nachdem die Herleitung von SNNs alles andere als trivial ist, soll an dieser Stelle die tatsächliche Umsetzung von SNNs vorgegriffen und eine Intuition zu deren Funktionsweise gegeben werden, bevor etwas genauer auf

einige der die zugrundeliegenden mathematischen Prinzipien eingegangen wird.
Ausschlaggebend für SNNs ist der Einsatz der sogenannten *Scaled Exponential Linear Unit (SELU)* als Aktivierungsfunktion. Diese ist gegeben als:

$$selu(z) = \lambda \begin{cases} z & \text{wenn } z > 0 \\ \alpha e^z - \alpha & \text{wenn } z \leq 0 \end{cases}$$

mit den Konstanten

$$\alpha \approx 1.6733$$

und

$$\lambda \approx 1.0507.$$

Der Grund für diese sehr spezifischen Konstanten ergibt sich später aus der Herleitung.
In Keras ist diese Funktion direkt als `'selu'`-Aktivierung verfügbar.
Grundsätzlich ist die SELU, wie ihr Name bereits vermuten lässt, nichts anderes als eine *skalierte* Version der bekannten *Exponential Linear Unit* (ELU):

$$elu(z) = \begin{cases} z & \text{wenn } z > 0 \\ \alpha(e^z - 1) & \text{wenn } z \leq 0 \end{cases}$$

Die Funktionsgraphen der obigen SELU und deren ersten Ableitung sehen folgendermaßen aus:

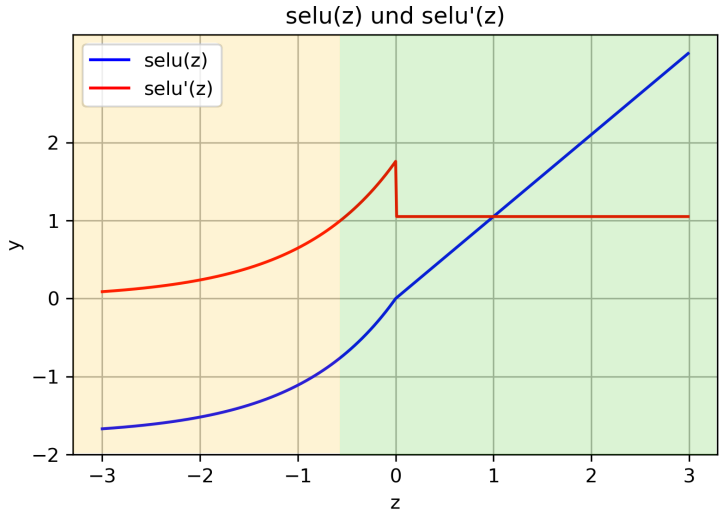

Abbildung 5

Bereits durch die Betrachtung des Graphen der SELU lässt sich eine erste Intuition zur Funktionsweise von SNNs entwickeln:
Bei den bereits vorgestellten Techniken wurde bisher immer von der Standardisierung der Rohinputs ausgegangen, da dies die in der Praxis gängigere Herangehensweise zu sein scheint, es wäre allerdings genauso denkbar stattdessen die Aktivierungen der versteckten Schichten zu standardisieren.
Bei SNNs werden nun zwangsweise die Aktivierungen standardisiert, da die Standardisierung nicht durch einen expliziten Algorithmus sondern hauptsächlich durch die Anwendung der Aktivierungsfunktion erfolgt. Es ergeben sich also einige Anforderungen an die Aktivierungsfunktion eines SNN:
Möchte man zunächst den Mittelwert der Aktivierungen kontrollieren, sollte der Wertebereich der Funktion sowohl positive als auch negative Werte enthalten um jeweils zu hohe oder zu niedrige Werte ausgleichen zu können.
Außerdem sollte sie sogenannte *saturation regions*, also Gradienten die gegen Null laufen, besitzen (gelber Teil in Abbildung 5). Nachdem die Steigung dort gleich oder nahe Null ist, weichen die y-Werte der Funktion selbst bei großen Schritten in x-Richtung kaum bis gar nicht mehr voneinander ab, was genutzt werden kann um die Standardabweichung zu senken, sollte sie zu groß werden. Ebenso sollten Abschnitte mit einem Gradienten > 1 existieren (grüner Teil in Abbildung 5), an denen selbst für kleine Schritte in x-Richtung große Änderungen der y-Werte erfolgen und somit die Standardabweichung erhöht werden kann, sollte sie zu klein werden. Letzteres gibt bereits eine Intuition für die Herkunft des Skalierungsfaktors λ.

Die Senkung der Varianz ist dabei stärker, je weiter sich die Werte von Null entfernen. Die Varianz wird umso mehr gesteigert, je näher sich die Werte Null nähern.
Liegt also bei den Aktivierungen einer Schicht eine hohe Varianz vor, ergibt sich für die Rohinputs der Folgeschicht, also die (gewichtete) Summe dieser Aktivierungen, vermutlich ein weiter von Null entfernter Wert, die Varianz wird also bei der Berechnung der Aktivierungen dieser Schicht gesenkt. Im gegenteiligen Fall liegen die Werte der Rohinputs bei einer niedrigen Varianz der Aktivierungen der Vorgängerschicht vermutlich nahe Null, die Varianz wird also für die Aktivierungen dieser Schicht erhöht.
Die Varianz sollte sich so bei einem gewünschten Wert ausbalancieren. Außerdem soll es sich bei der Funktion um eine kontinuierliche Kurve handeln, was sicherstellt, dass ein Punkt existiert, an dem sich die Senkung der Standardabweichung mit der Erhöhung der Standardabweichung ausgleicht, die Standardabweichung also gleich bleibt.
Im Gegensatz zu den bekannten Aktivierungsfunktionen wie der Rectified Linear Unit (ReLU), ELU oder Sigmoidfunktion, erfüllt die obige SELU alle diese Eigenschaften.

5.3 Mathematische Hintergründe

Die letzte Eigenschaft beschreibt bereits einen Teil des wortwörtlich ausschlaggebenden Punktes bei der Herleitung von SNNs:
Es existiert ein Punkt, an welchem Mittelwert und Standardabweichung nach Anwendung der SELU gleich bleiben. Mathematisch ausgedrückt, handelt es sich bei diesem Punkt um einen Fixpunkt. Dies bedeutet nichts weiter, als dass der Fixpunkt x_0 einer Abbildung h durch diese immer auf sich selbst abgebildet wird. Es gilt also:

$$h(x_0) = x_0.$$

Idealerweise liegt dieser Punkt natürlich genau bei den gewünschten Werten für Mittelwert und Standardabweichung. Allerdings sollen Mittelwert und Standardabweichung der Komponenten einer Aktivierung nicht nur erhalten bleiben, wenn sie das gewünschte Maß haben, sondern davon abweichende Verteilungen automatisch standardisiert werden.
Dies soll geschehen, indem mit jeder Anwendung der Aktivierungsfunktion Mittelwert und Standardabweichung weiter zum Fixpunkt hin konvergieren. Mathematisch korrekter ausgedrückt soll die Aktivierungsfunktion g sich also *kontraktiv* zum Fixpunkt verhalten, bzw. ist g eine *Kontraktion*. Bei einer Kontraktion handelt es sich sehr vereinfacht gesprochen um eine Abbildung einer Menge auf sich selbst, bei der der Abstand zweier Punkte nach Anwendung der Kontraktion auf diese Punkte immer geringer ist als zuvor.
Die Normalverteilung wird also durch iteratives Anwenden der Aktivierungsfunktion auf die Rohinputs einer versteckten Schicht hergestellt. Der Standardisierungsprozess kann dabei aus zwei Sichtweisen betrachtet werden: Zum einen konvergiert in einem tiefen NN die Verteilung mit jeder Schicht durch die jeweilige Anwendung der Aktivierungsfunktion weiter Richtung Fixpunkt, zum anderen kann auch bei der Betrachtung einer Schicht über die Lernschritte hinweg festgestellt werden, dass die Verteilung der Aktivierungen mit jedem Schritt näher an die Normalverteilung rückt.
Um nachzuweisen, dass eine Funktion die beiden obigen Eigenschaften besitzt kann der Banach'sche Fixpunktsatz verwendet werden. Dieser besagt, dass, wenn es sich bei der Abbildung h einer Teilmenge M eines geschlossenen metrischen Raumes auf sich selbst um eine Kontraktion handelt, diese genau einen einzigartigen, anziehenden Fixpunkt x_0 besitzt, für den gilt:

$$h(x_0) = x_0$$

und für alle Punkte $a \in M$

$$\lim_{n\to\infty} h^n(a) = x_0.$$

Jeder Punkt in M läuft also durch die Anwendung von h immer weiter auf den Fixpunkt zu.
Das Ziel ist nun also die allgemeine SELU so zu skalieren, dass sich der Fixpunkt sozusagen an der gewünschten Stelle bezüglich Mittelwert und Standardabweichung befindet und anschließend nachzuweisen, dass für diese Funktion der Banach'sche Fixpunktsatz gilt.
Zur weiteren Herleitung wird ein Beispielnetz, in Form eines kleinen FNN mit Aktivierungsfunktion $g(z)$ und lediglich zwei Schichten, welche durch die Gewichtsmatrix $W^{(1)}$ verbunden sind, betrachtet.
Nachdem die Input-Samples x des Netzes in der Regel nach einer Zufallsverteilung in das Netz gegeben werden, handelt es sich bei den Komponenten des Inputvektors x_i also um Zufallsvariablen.
Die Aktivierungen der ersten Schicht $a_i^{(1)}$ entsprechen wie immer den Komponenten des Input-Vektors x ($a_i^{(1)} = x_i$) und sind somit ebenfalls Zufallsvariablen. Für diese kann nun der Mittelwert

$$\mu := E(a_i^{(1)})$$

und die Varianz

$$\nu := Var(a_i^{(1)})$$

definiert werden.
Eine Komponente des Rohinputs von Schicht Zwei $z_i^{(2)}$ berechnet sich wie gewohnt mit $z_i^{(2)} = w^T a^{(1)}$ und ist somit ebenfalls als Zufallsvariable anzusehen. An dieser Stelle und in der weiteren Herleitung wird der Bias-Vektor vernachlässigt. In der Praxis wird dieser für SNNs in der Regel mit 0 initialisiert und bleibt während des Trainings weitgehend unverändert.
Nachdem für eine eine Komponente der Aktivierung in Schicht Zwei $a_i^{(2)} = g(z_i^{(2)})$ gilt, sind die Aktivierungen in Schicht Zwei ebenfalls Zufallsvariablen mit einem Mittelwert

$$\tilde{\mu} := E(a_i^{(2)})$$

und einer Varianz

$$\tilde{\nu} := Var(a_i^{(2)})$$

Anstatt der gesamten Gewichtsmatrix $W^{(1)}$ lassen sich auch die einzelnen Gewichtsvektoren w der jeweiligen Aktivierungen $a_i^{(1)}$ betrachten. Für n Neuronen in der ersten Schicht kann nun n-mal der Mittelwert der n Gewichtsvektoren w als

$$\omega := \sum_{i=1}^{n} w_i$$

und n-mal die Varianz der Gewichtsvektoren als

$$\tau := \sum_{i=1}^{n} w_i^2$$

definiert werden.
Das gesamte Netz kann nun als Abbildung f betrachtet werden, welche den Mittelwert und die Varianz der Aktivierung aus Schicht Eins auf Mittelwert und Varianz der Aktivierung von Schicht Zwei abbildet:

$$f : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu}).$$

Nun muss für den nächsten Schritt die etwas zweifelhafte Annahme getroffen werden, dass es sich bei den Komponenten des Inputvektors um *unabhängige* Zufallsvariablen handelt. Ist dies der Fall gilt für die x_i und die von ihnen abhängigen Zufallsvariablen der *Zentrale Grenzwertsatz*. Diese ist in der Realität natürlich kaum erfüllt, es existieren allerdings Verallgemeinerungen des Zentralen Grenzwertsatzes, wie beispielsweise die *Ljapunow-Bedingung*, welche anstatt völliger Unabhängigkeit lediglich voraussetzt, dass eine einzelne Zufallsvariable keinen zu großen Einfluss auf das Gesamtergebnis haben darf. Zur Vereinfachung der Herleitung kann also angenommen werden, dass der Zentrale Grenzwertsatz weithin gilt.
Der Zentrale Grenzwertsatz besagt, dass bei ausreichend vielen Zufallsvariablen, diese eine Normalverteilung annehmen. Es kann also angenommen werden, dass die Rohinputs z_i einer breiten Schicht mit einer Normalverteilung $z \sim N(\mu\omega, \sqrt{\nu\tau})$ mit der Dichte $p_N(z; \mu\omega, \sqrt{\nu\tau})$ vorliegen.
Ist dies erfüllt kann der Mittelwert $\tilde{\mu}$ der Abbildung $g : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu})$ mit der allgemeinen SELU-Funktion berechnet werden als:

$$\tilde{\mu}(\mu, \omega, \nu, \tau) = \int_{-\infty}^{\infty} \mathsf{selu}(z) p_n(z; \mu\omega, \sqrt{\nu\tau}) \; dz$$

Die Standardabweichung wird analog mit

$$\tilde{\nu}(\mu, \omega, \nu, \tau) = \int_{-\infty}^{\infty} \mathsf{selu}(z)^2 p_n(z; \mu\omega, \sqrt{\nu\tau}) \; dz - \tilde{\mu}^2$$

berechnet.
Wichtiger als die genaue Herleitung oder Bedeutung dieser Integrale ist für das Verständnis von SNNs, die Parameter der obigen Gleichungen zu betrachten.
Mittelwert und Varianz der zweiten Schicht sind also neben Mittelwert und Varianz der Vorgängerschicht, sowie der Aktivierungsfunktion, abhängig von Mittelwert und Varianz der Gewichte. Nun sollen die genauen Werte für λ und α der allgemeinen SELU mithilfe der obigen Gleichungen so bestimmt werden, dass der Fixpunkt bei der gewünschten Normalverteilung, also der Standardnormalverteilung liegt.
Aufgrund des Fixpunktes gilt:

$$f(\mu, \nu) = (\tilde{\mu}, \tilde{\nu}) = (\mu, \nu)$$

Es können also $\mu = \tilde{\mu} = 0$ und $\nu = \tilde{\nu} = 1$ gesetzt werden.

Wie aus den Funktionsparametern hervorgeht ist auch die Verteilung der Gewichte ausschlaggebend für die Position des Fixpunktes. Diese werden deshalb ebenfalls mit Mittelwert 0 und Standardabweichung 1, also

$$\omega = 0 \ , \ \tau = 1$$

initialisiert. In Keras kann dies durch die Wal des `'lecun_normal'`-Initializers für eine Schicht realisiert werden. Diese Normalverteilung der Gewichte kann während des Trainings selbstverständlich nicht garantiert werden, es wird also zunächst sozusagen nur der erste Lernschritt betrachtet.
Mit den eben festgelegten Werten für μ, ν, ω und τ können die obigen Integrale nun nach α und λ gelöst werden. Man erhält die bekannten $\alpha_{01} \approx 1.6733$ und $\lambda_{01} \approx 1.0507$.

Mit der nun erfolgreich skalierten SELU-Funktion muss nur noch bewiesen werden, dass die obige Abbildung einen stabilen anziehenden Fixpunkt besitzt, also der Banch'sche Fixpunktsatz gilt.
Dies ist der Fall, wenn die Norm der *Jacobi-Matrix* von f kleiner als 1 ist.
Berechnet man die 2x2 Jacobi-Matrix $J(\mu, \nu)$ von $f : (\mu, \nu) \mapsto (\tilde{\mu}, \tilde{\nu})$ für den Punkt $(0, 1)$ mit den Konstanten α_{01} und β_{01} erhält man $J(0, 1) = ((0.0, 0.088834), (0.0, 0.782648))$. Die Spektralnorm von J, also der Wert der größten einzelnen Komponente, ist somit kleiner als 1. Es ist also gezeigt, dass $f_{\alpha_{01}, \lambda_{01}}$ kontraktiv bezüglich des Fixpunktes $(0, 1)$ ist.

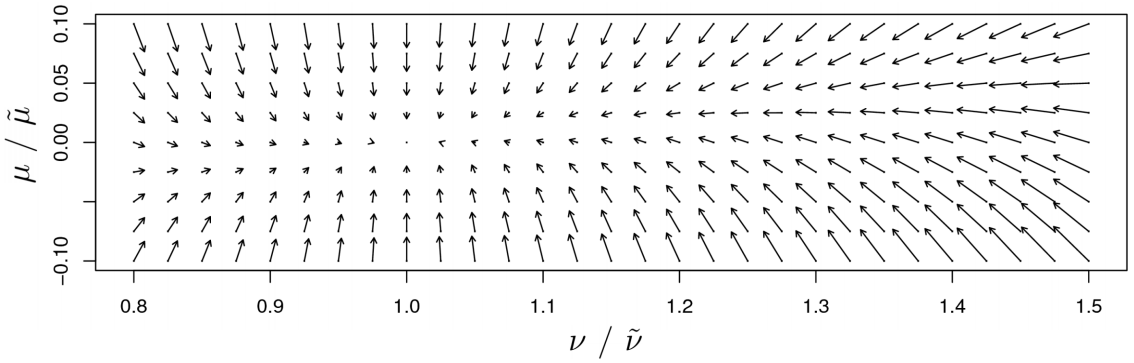

Abbildung 6: Richtung der Abbildung einiger möglicher (μ, ν) auf $(\tilde{\mu}, \tilde{\nu})$ durch die Abbildung g mit den bekannten Parametern; Die Konvergenz zum Punkt $(0, 1)$ ist hierbei sehr gut ersichtlich.
(Klambauer et al., 2017)

Allerdings gilt all dies nur, wenn die Gewichte mit $\omega = 0$, $\tau = 1$ normal verteilt sind, was während des Trainings, wie bereits erwähnt, nicht garantiert werden kann.
Auch während des Trainings einen anziehenden Fixpunkt bei $(0, 1)$ zu erhalten, ist somit nahezu unmöglich. In der Praxis ist es allerdings unerheblich, ob die Verteilung der Aktivierungen nun exakt oder nur annähernd der Standardnormalverteilung entspricht.
Hierzu kann gezeigt werden, dass für ein bestimmtes Intervall von μ, ν, ω und τ nahe der Normalverteilung $(0, 1)$ weiterhin ein anziehender Fixpunkt nahe $(0, 1)$ existiert. Konkret gibt es für $\mu \in [-0.1, 0.1]$, $\omega \in [-0.1, 0.1]$, $\nu \in [-0.8, 1.5]$ und $\tau \in [-0.95, 1.1]$ einen anziehenden Fixpunkt im Intervall $\mu \in [-0.03106, 0.6773]$, $\nu \in [0.80009, 1.48617]$. Bleibt die Verteilung der Gewichte also ausreichend nahe an der Standardnormalverteilung, konvergiert Normalverteilung der Aktivierungen weiterhin extrem nahe an den Fixpunkt $(0, 1)$.
Allerdings gibt es weiterhin keine Garantie, dass μ und ν auch in obigem Intervall verbleiben. Besonders kritisch sind hierbei die Grenzwerte von ν:
Es sei an die anfängliche Intuition über den Funktionsgraphen der SELU erinnert:
Würde die Varianz durch die Anwendung der Abbildung, also in einem tiefen NN mit jeder Schicht, immer weiter gesenkt, würden die Werte der Aktivierungen immer weiter in den Bereich mit Gradienten Null gedrückt, dies entspräche also dem Problem des Vanishing Gradient. Umgekehrt würde der Gradient der Aktivierungsfunktion immer weiter ansteigen, würde die Varianz mit jeder Abbildung weiter erhöht, was einem Exploding Gradient gleichkommt.
Es lässt sich allerdings für ein etwas großzügigeres Intervall von ν zeigen, dass die Varianz innerhalb dieses Intervalls auf einen Minimal- und Maximalwert abgebildet wird.
Konkret beträgt der Maximalwert der Varianz im Intervall $[3, 16]$ den Wert 3, der Minimalwert im Intervall $[0.02, 0.24]$ beträgt 0.24.

Wichtiger als die genauen Werte dieser Intervalle ist die allgemeinere Bedeutung dahinter:

Self Normalizing Neural Networks sorgen durch die Kombination der SELU Funktion und zu Beginn mit entsprechend der Standardnormalverteilung initialisierten Gewichtsvektoren dafür, dass die Aktivierungen der versteckten Schichten über die Schichten und Lernschritte hinweg entweder zum Mittelwert 0 und Varianz 1 oder zu einer Normalverteilung nahe dieser Werte konvergieren. Selbst wenn sich die Verteilung der Gewichte während des Trainings ändert, konvergieren die Aktivierungen weiterhin zu einer Normalverteilung. Außerdem kann durch die Existenz eines Minimal- und Maximalwerts für die Varianz der Aktivierungen ein Vanishing und Exploding Gradient ausgeschlossen werden.

5.4 Regularisierung bei SNNs

Anders als beispielsweise Batch Norm besitzen SNNs keine nennenswerten regularisierenden Eigenschaften. Da eines der Ziele von SNNs allerdings bekanntlich das Erzeugen tiefer, performanter FNNs war, ist der Wunsch nach Regularisierung angesichts der hohen Parameterzahl diese Netze durchaus nachvollziehbar.
Dazu würde nun typischerweise die *Dropout*-Technik eingesetzt.
Diese Technik ist allerdings nicht ohne einige Anpassungen, in Form der sogenannten *Alpha Dropout*-Variante, auf SNNs anwendbar. Für den Unterschied von regulärem und Alpha Dropout soll im Folgenden eine knappe Intuition gegeben werden:
Das zufällige Ein- und Ausschalten von Aktivierungen hat offensichtlich einen großen Einfluss auf deren Mittelwert und Varianz. Um also die Eigenschaften des SNNs auch in Kombination mit Dropout zu erhalten, sollte die Verteilung der Aktivierungen durch diesen möglichst unberührt bleiben.
Hierzu erscheint es sinnvoll die ausgeschalteten Aktivierungen mit dem niedrigst-möglichen Wert der Varianz, also dem Grenzwert der SELU im negativ-Unendlichen, zu ersetzen:

$$\lim_{z \to -\infty} \text{selu}(z) = -\lambda\alpha$$

Dieser Grenzwert wird auch als α' bezeichnet.
Anstatt also ausgeschaltete Aktivierungen wie beim regulären Dropout gleich Null zu setzten, wird beim Alpha Dropout der namensgebende α'-Grenzwert benutzt. Alpha Dropout ist inzwischen eine wählbare Schicht in Keras und scheint mit Werten von 0.05 oder 0.1 für $p - 1$ sehr gute Regularisierungsergebnisse bei SNNs zu erzielen.

5.5 SNNs in der Praxis

Nachdem eine Hauptmotivation hinter der Entwicklung von SNNs war, sehr tiefe Feedforward Netze effizient trainierbar zu machen, widmet sich der experimentelle Teil des Papers sehr intensiv eben jenen FNNs. Es ist zwar sehr eindrucksvoll zu sehen, dass Self-Normalizing-FNNs mit Architekturen von bis zu acht Schichten so effektiv trainiert werden können, dass diese in Challenges, welche typischerweise von SVMs oder Random Forests dominiert werden, neue Rekorde aufstellen, jedoch sind die Datensätze und Ergebnisse kaum mit den bekannten Netzen und Szenarien aus der Vorlesung vergleichbar.
Auch wenn hier der Fokus stark auf FNNs gelegt wurde sind die SNN-Techniken genauso für andere Architekturen einsetzbar. So erweisen sich beispielsweise Self-Normalizing Convolutional Neural Networks (SCNNs) als effektive Alternative zu herkömmlichen CNNs bei Aufgaben der Textklassifizierung (Madasu & Rao, 2019).
Der Vergleich dieser SCNNs mit CNNs illustriert dabei schön die Vor- und Nachteile von SNNs gegenüber den üblichen Standardisierungstechniken wie Batch Norm:
SNNs sind bei der Erstellung eines NNs deutlich restriktiver, da sie eine spezielle Aktivierungsfunktion und Gewichtsinitialisierung voraussetzten und nicht ohne Weiteres mit jedem Optimizer kompatibel sind, während BN praktisch in beinahe jedes bereits existierende Netz eingefügt werden kann.
Dafür erzielen sie, aufgrund des Wegfalls der gesonderten Verarbeitung der Standardisierung, bei geringerer Parameterzahl vergleichbare, oder bei gleicher Parameterzahl, verbesserte Ergebnisse.

Ein interessantes Ergebnis aus den praktischen Experimenten mit SNNs ist folgender Vergleich (Abbildung 7) der Verteilung der Aktivierungen der zweiten Schicht eines SNN direkt nach der Initialisierung (links, rot) und nach 40 Epochen (rechts, rot):

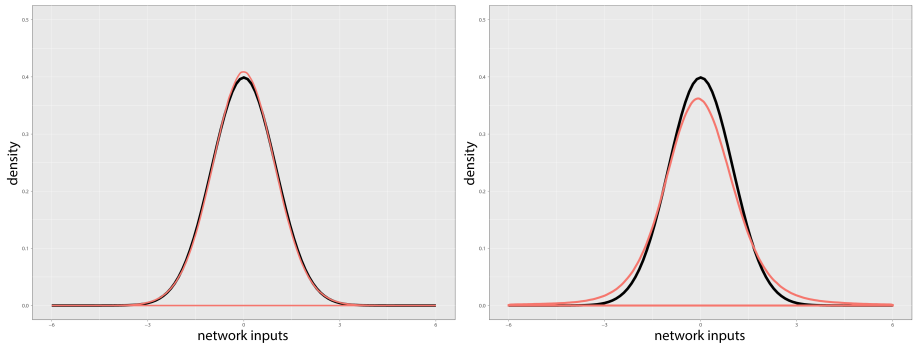

Abbildung 7:
(Klambauer et al., 2017)

SNNs scheinen also auch in der Praxis tatsächlich dafür sorgen, dass die versteckten Aktivierungen über die Lernschritte und Schichten hinweg auf eine Verteilung nahe der Standardnormalverteilung (schwarz) zulaufen.

6 Einschränkung zu Internal Covariate Shift

Auch wenn die Effektivität von Batch Norm und dessen Nachfolgetechniken in der Praxis unumstritten ist, scheint die ursprüngliche Motivation hinter der Entwicklung dieser Verfahren, das Reduzieren von Internal Covariate Shift, nicht der tatsächlich ausschlaggebende Grund für den Erfolg der Standardisierungstechniken zu sein (Santurkar et al., 2018).
Da sich diese Intuition bis heute hartnäckig als Erklärung für die meisten Standardisierungstechniken hält, sollen zum Abschluss dieser Arbeit zumindest oberflächlich die Probleme der ICS-Begründung beleuchtet und eine alternative Erklärung angeboten werden:
Am anschaulichsten wird die Thematik durch ein Experiment. Es werden zwei nahezu identische VGG-Netze erstellt, welche sich nur darin unterscheiden, dass bei einem von beiden Batch Norm eingesetzt wird, während das Andere ohne BN auskommen muss.
Der Clou dieses Experiments ist nun das Erstellen eines dritten VGG, welches ebenfalls die selbe Architektur in Kombination mit Batch Norm besitzt, jedoch jede Aktivierung in diesem Netz für jedes Trainingsbeispiel mit einem zufälligen Noise verzerrt wird. Das Netz sollte also erheblichen ICS aufweisen.

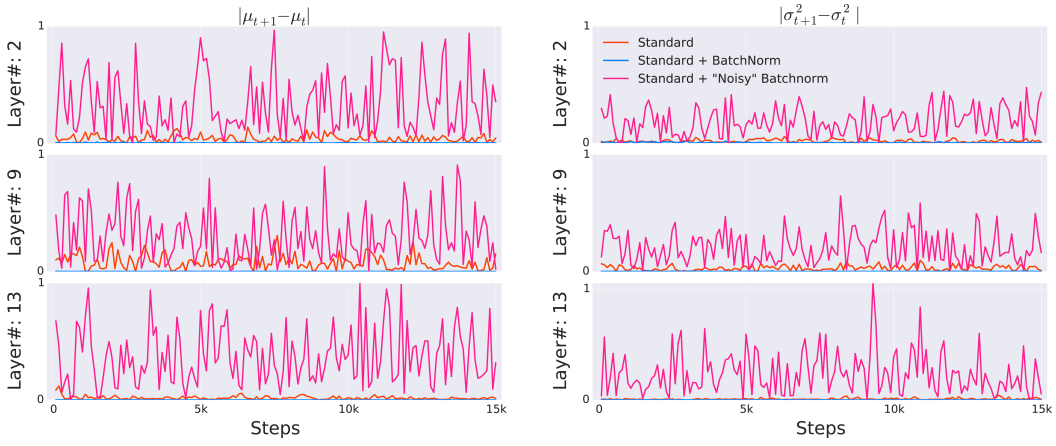

Abbildung 8:
(Santurkar et al., 2018)

Abbildung 8 veranschaulicht den Grad des ICS über das Training hinweg als die jeweilige Differenz von Mittelwert beziehungsweise Standardabweichung zwischen dem jeweils aktuellen und vorhergehenden Zeitschritt. In der Tat gelingt es dem Batch-Norm-VGG (blau) über das Training hinweg die Verteilung gleich zu halten, während das "Noisy"-Batch-Norm Netz (pink) sogar einen enorm höheren ICS aufweist als das Standardnetz (orange).
Betrachtet man nun aber die Performance der Netze in Bezug auf die Schnelle der Konvergenz, stellt man fest, dass "Noisy"-Batch-Norm nicht nur besser abschneidet als das Standardnetz, sondern sogar mit dem ungestörten Bach-Norm-Netz vergleichbare Performance liefert. (vgl. Abbildung 9)

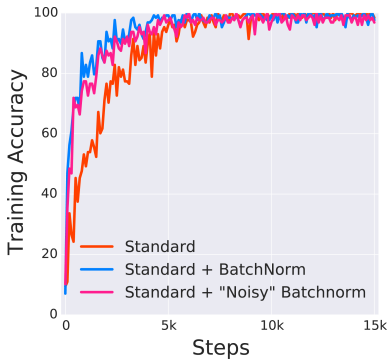

Abbildung 9:
(Santurkar et al., 2018)

Der Grad des ICS nach der gängigen Definition scheint also keine klar nachweisbare Verbindung zur Performance eines Netzes zu haben.

Batch Norm oder andere Standardisierungsverfahren wie SNNs verbessern aber in der Praxis signifikant die Performance eines Netzes, sowohl in Bezug auf die Geschwindigkeit des Trainings, als auch in Bezug auf die erzielte Vorhersagegenauigkeit.
Wenn ICS nicht der Grund für diese Verbesserung ist, was dann?
Ein ausschlaggebender Grund könnte sein, dass diese Techniken sowohl die Fehlerlandschaft als auch deren Gradienten enorm glätten.
Vergleicht man in Abbildung 10 die Vorhersehbarkeit des Gradienten zu zwei verschiedenen Zeitpunkten des Trainings, gemessen als "Distanz", genauer der L2-Norm, zwischen dem Gradienten vor und nach einer bestimmten Schrittweite, stellt man fest, dass bei einem Standard-VGG-Netz ohne Batch Norm (links), die Gradienten bei zu großen Schrittweiten extrem unberechenbar werden, während die Gradienten für ein VGG mit Batch Norm (rechts) auch für große Schrittweiten sehr vorhersehbar bleiben.

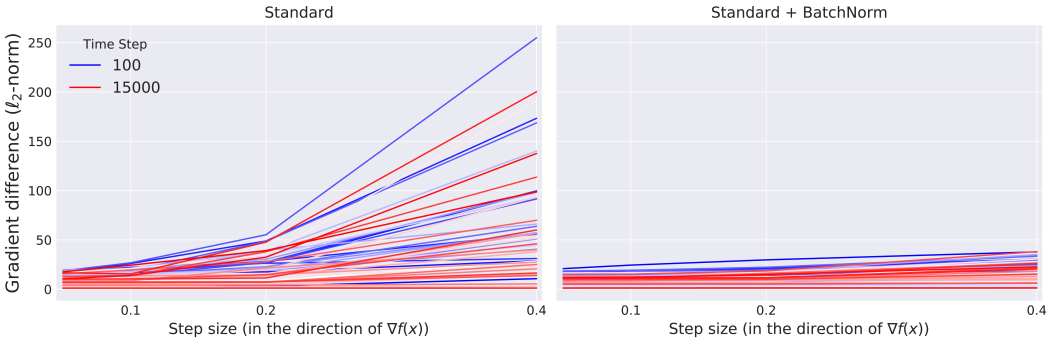

Abbildung 10:
(Santurkar et al., 2018)

Dies ermöglicht beim Lernen in größeren Schritten voranzuschreiten ohne Gefahr zu laufen, dass der Gradient sich innerhalb dieses Schrittes wesentlich in der Richtung ändert. Somit könnte das Training problemlos durch die Wahl einer größeren Lernrate beschleunigt werden, was sich mit den beobachteten Vorteilen aller Standardisierungstechniken deckt. Auch die in allen Techniken beobachtete Reduzierung von Vanishing und Exploding Gradients ließe sich dadurch erklären.
Der Grund für die Wirksamkeit der Standardisierungstechniken scheint also eher im Einfluss der Standardisierung auf den Gradienten, also Verbesserungen bei der Backpropagation, zu liegen, als in verbesserter Forward-Propagation durch die standardisierten Aktivierungen oder Rohinputs selbst.

7 Konklusion

Beim Training tiefer Neuronaler Netze sind Techniken zur Standardisierung der Inputs der versteckten Schichten zu Recht zum Standard geworden. Die verschiedenen Herangehensweisen wie Batch Normalization, Layer Normalization oder auch Self Normalizing Neural Networks ermöglichen allesamt ein schnelleres Training der

Netze durch höhere Lernraten und reduzieren dabei die Problematik des Vanishing oder Exploding Gradient, bzw. eliminieren diese sogar.
Bei Batch Norm werden die Komponenten der Rohinput-Vektoren der versteckten Schichten jeweils für sich über die Trainingsbeispiele des jeweiligen Batches oder Minibatches standardisiert. Layer Norm standardisiert stattdessen die Rohinputs über ihre Komponenten und ist somit unabhängig von Trainingsbeispielen und etwaigen Batchgrößen.
Bei SNNs konvergieren die Aktivierungen der versteckten Schichten automatisch durch die Kombination aus SELU-Aktivierungsfunktion und gemäß der Standardnormalverteilung initialisierten Gewichten über die Schichten und Lernschritte hinweg zu einer Normalverteilung nahe Mittelwert 0 und Varianz 1.
Keine der Techniken ist dabei pauschal besser oder schlechter, sie bringen alle ihre individuellen Stärken und Schwächen mit sich. Während Batch Norm beispielsweise mit nahezu allen Aktivierungsfunktionen, Optimizern oder Netzarchitekturen kompatibel ist, ist dennoch die Kombination von BN mit SGD bzw. Online Learning unmöglich, was wiederum durch Layer Norm ermöglicht wird.
Die beiden Techniken führen allerdings durch die separate Berechnung der Standardisierungen zu einer erhöhten Parameterzahl im Vergleich zu Self Normalizing Neural Networks, welche für ihre Funktion eine spezielle Aktivierungsfunktion und Gewichtsinitialisierung erfordern und somit etwas restriktiver bei der Zusammenstellung des NNs sind. Die Wahl der Standardisierungstechnik sollte daher in der Praxis sehr spezifisch für die jeweilige Aufgabenstellung getroffen werden.
Alle dieser Techniken scheinen ihre Performance-Verbesserungen daraus zu ziehen, dass sie die Gradienten der Fehlerfunktion glätten und diese somit selbst bei großen Lernschritten vorhersehbar bleiben. Auch das Problem des Vanishing und Exploding Gradients wird dadurch behoben.

Literatur

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of machine learning research* (S. 37:448-456).

Klambauer, G., Unterthiner, T., Mayr, A. & Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in neural information processing systems 30 (nips 2017)*.

Madasu, A. & Rao, V. A. (2019). Effectiveness of self normalizing neural networks for text classification. *arXiv preprint arXiv:1905.01338*.

Santurkar, S., Tsipras, D., Ilyas, A. & Mądry, A. (2018). How does batch normalization help optimization? In *Proceedings of the 32nd international conference on neural information processing systems* (S. 2488–2498).

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, *90*, 227-244.