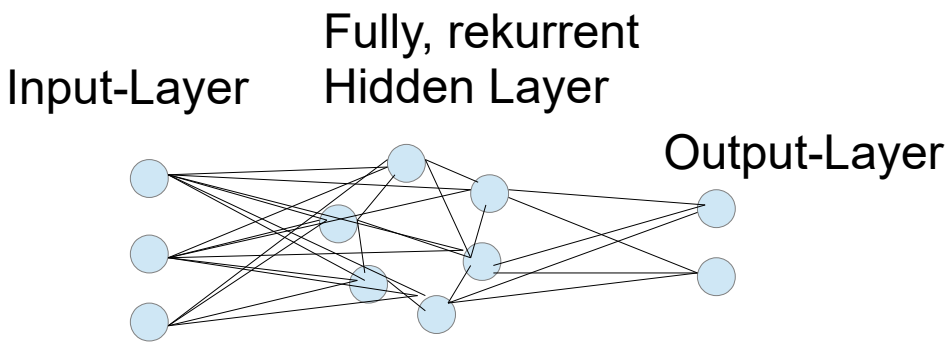
Neuronale Netze trainieren mit lokalem Verfahren ohne Backpropagation oder Fehlerkorrektur

Inhaltsverzeichniss

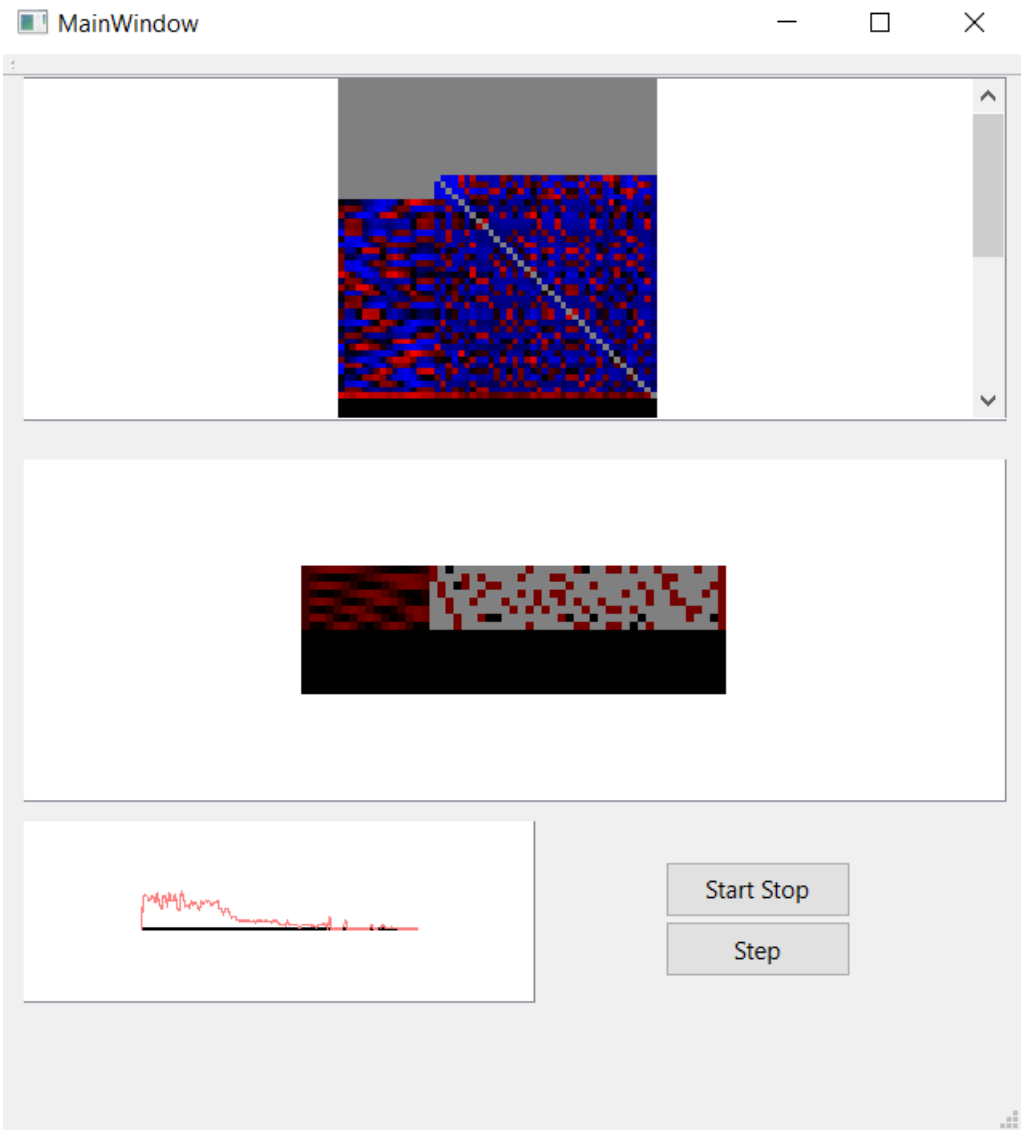
1. Einleitung

Im folgenden möchte ich ein Verfahren beschreiben was die lokale Anpassung der Gewichte eines Neuronalen Netzes mit rekurrenten Verknüpfungen ohne Fehlerrückberechnung (Backpropagation) alleine mit den Signalen die einem Neuron für sich zu Verfügung stehen ermöglicht.
Der Algorithmus den ich dafür entwickelt habe war nicht im Internet zu finden und ermöglicht eine hohe Parallelität in der Berechnung deswegen möchte ich im Folgenden eine schnelle Einführung ermöglichen.

2. Netztopologie, Vorwärts-pass und Aktivierungsfunktion



Die oben dargestellte Netztopologie ist die welche ich im folgenden benutzen möchte.
Sie besteht aus einem Input Layer in die die Daten für das Netz eingegeben werden. Ein „fully-rekurrent-hidden Layer“ welches den Hauptteil der Verarbeitung übernimmt und ein Output-Layer welches aus welchem die Verarbeiteten Daten entnommen werden können.
Untenstehend ist die Netztopologie mit einer kleinen Zahl an Neuronen als fertig trainierte Gewichtsmatrix zu sehen, sowie die Fehlerausgabe und die Neutronenaktivität am Ende des Berechnungszyklus für ein Beispiel. Die momentane Aufgabe des Modells ist die Frequenz einer Sinuswelle zu schätzen.



Im Fenster sind von oben nach unten die Gewichtsmatrix des Neuronalen Netzes zu sehen, darunter die Neuronale-Aktivität und unten links die Fehlerentwicklung über die Zeit des Trainings. Als Aktivierungsfunktion habe ich die Sigmoide-Aktivierungsfunktion genutzt welche für meinen Trainingsalgorithmus bis jetzt auch die einzige neben ihrem inversen ist welche funktioniert:

$$o_i(x_i) = \frac{1.0}{1.0 + e^{-x_i}}$$

Die Berechnung des Eingangssignals ist wie bei allen Neuronalen Netzen durch die Summe der Aktivierungen auf den Eingangsgewichten multipliziert mit diesen gegeben:

$$x_i = \sum_{j=0}^{N} w_{ij} \cdot o_j$$

3. Trainingsalgorithmus

Den Trainingsalgorithmus den ich im folgenden beschreibe ermöglicht ein quasi lokales trainieren des obigen Neuronalen Netzes ohne Fehlerrückführung, Gradient oder Fehlersignal. Die dafür benötigten Variablen berechnen sich aus den Aktivitäten des Neuronalen Netz und werden im folgenden Vorgestellt.

1. Korrektursignalberechnung

Für die Berechnung der Gewichtskorrektur sind im wesentlichen zwei Signale im Neuronalem Netz erforderlich.
Während eines davon, das Eingangssignal bereits vorgestellt wurde wird dem anderen selten Beachtung geschenkt, nämlich dem eigentlichen Ausgangssignal des einzelnen Neuron. Hierbei meine ich nicht den Output welcher durch die Aktivierungsfunktion berechnet wird sondern die Summe der Signale welche auf den Ausgangsgewichten des jeweiligen Neuron zu finden sind. Untenstehend sind Beide mathematisch beschreiben.

a) Eingangssignal des Neuron i

$$x_i = \sum_{j=0}^{N} w_{ij} \cdot o_j$$

b) Ausgangssignal des Neuron i

$$y_i = \sum_{j=0}^{N} w_{ji} \cdot o_i$$

2. Gewichtskorrektur

Für die Gewichtskorrektur sind die beiden obigen Signale sowie die tatsächlichen Aktivierungen o(x)_i der Neuronen, deren Anzahl und eine Lernrate notwendig, mehr nicht:

a) Für die Korrektur der Eingangsgewichte

$$w_{ij} = w_{ij} - o_i \cdot x_i \cdot \eta$$

b) Für die Korrektur der Ausgangsgewichte

$$w_{ji} = w_{ji} - o_j \cdot y_i \cdot \eta$$

3. Gewichtsnormalisierung

Der letzte Schritt besteht in der Normierung der Gewichte mittels der Summe ihres Absolutwertes und zwar jeweils die Eingabegewichte als auch die Ausgabegewichte:

a) Für die Normierung der Eingangsgewichte

$$w_{ij} = \frac{w_{ij}}{(\sum_{j=0}^{N} |w_{ij}|)}$$

b) Für die Normierung der Ausgabegewichte

$$w_{ji} = \frac{w_{ji}}{(\sum_{i=0}^{N} |w_{ji}|)}$$

4. Zusammenfassung und weiterführende Hinweise

Der Algorithmus den ich hier skizziert habe ist voll funktionsfähig auf meinem Github-Repository (https://github.com/SebastianSchwank/Indirect-Backpropergation) zu finden. Leider ist mir noch nicht völlig klar warum er zur Minimierung des Ausgabefehlers führt. Des weiteren habe ich bemerkt dass er auch richtig trainiert wenn ich für jedes Neuron die Aktivierungsfunktion invertiere allerdings nicht mit gemischt poligen Aktivierungsfunktionen im gleichen Netzwerk. Andere Aktivierungsfunktionen sind in dieser Form noch nicht unterstützt.
Die jetzige Implementierung ist noch auf der CPU und ohne jegliche Parallelisierung.

Verfasst am 07.07.2022 von,

Sebastian Schwank