

Neuronale Netze

Die mathematische Struktur

Tom Weinmann

18. Dezember 2021

Inhalt

- 1 Motivation der Modellstruktur
- 2 Neuronale Netze
- 3 Trainieren von neuronalen Netzen

Inhalt

1 Motivation der Modellstruktur

- Regressionsmodell
- Klassifikationsmodell
- Der Fluch der Dimension
- Struktur des Modells

Motivation der Struktur des Modells

Um die mathematische Struktur der neuronalen Netze zu motivieren, erinnern wir zuerst sehr kompakt an lineare Regressions-Modelle und noch kompakter an lineare Klassifikations-Modelle.

Motivation der Struktur des Modells

Um die mathematische Struktur der neuronalen Netze zu motivieren, erinnern wir zuerst sehr kompakt an lineare Regressions-Modelle und noch kompakter an lineare Klassifikations-Modelle.

Anschliessend gehen wir auf eine diesen beiden Modellfamilien gemeinsame strukturelle Schwäche ein.

Motivation der Struktur des Modells

Um die mathematische Struktur der neuronalen Netze zu motivieren, erinnern wir zuerst sehr kompakt an lineare Regressions-Modelle und noch kompakter an lineare Klassifikations-Modelle.

Anschliessend gehen wir auf eine diesen beiden Modellfamilien gemeinsame strukturelle Schwäche ein.

Schliesslich kondensieren wir daraus die mathematische Grundstruktur der neuronalen Netze als eine simultane Erweiterung beider Modellfamilien.

Regression

Lineare Regressionsmodelle sind eine spezielle Familie von diskriminativen, parametrischen, probabilistischen Modellen

$$p(t|\mathbf{x}, \boldsymbol{\theta})$$

für die Verteilung der reellen Ausgabe

$$t \in \mathbb{R}$$

bedingt auf die Eingabe $\mathbf{x} \in \mathbb{R}^d$, wobei wir mit $\boldsymbol{\theta} \in \mathbb{R}^p$ die adaptierbaren Parameter des Modells bezeichnen.

Lineares Regressionsmodell

Ein **lineares Regressionsmodell** wird definiert durch die Wahl einer endlichen Familie

$$\phi(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x}))^T$$

von **Basisfunktionen** und der gaussischen Ausgabeverteilung

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t | \phi^T(\mathbf{x}) \mathbf{w}, \beta^{-1}), \quad (1)$$

deren Mittelwert von der Eingabe \mathbf{x} mittels der Linearkombination

$$\phi^T(\mathbf{x}) \mathbf{w} = \sum_{k=0}^M \phi_k(\mathbf{x}) w_k$$

der an der Eingabe \mathbf{x} ausgewerteten Basisfunktionen ϕ abhängt. (2)

Lineares Regressionsmodell

Ein **lineares Regressionsmodell** wird definiert durch die Wahl einer endlichen Familie

$$\phi(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x}))^T$$

von **Basisfunktionen** und der gaussischen Ausgabeverteilung

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|\phi^T(\mathbf{x})\mathbf{w}, \beta^{-1}), \tag{1}$$

deren Mittelwert von der Eingabe \mathbf{x} mittels der Linearkombination

$$\phi^T(\mathbf{x})\mathbf{w} = \sum_{k=0}^M \phi_k(\mathbf{x}) w_k$$

der an der Eingabe \mathbf{x} ausgewerteten Basisfunktionen ϕ abhängt. (2)

Die adaptiven Parameter θ solch eines Modells bestehen aus dem Vektor $\mathbf{w} \in \mathbb{R}^{M+1}$ der Gewichte der Linearkombination der Basis-Funktionen und der Präzision $\beta > 0$ der Normalverteilung.

Modell anpassen

Wie immer bei überwachten Lernproblemen verwenden wir Trainings-Daten

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq N\}$$

bestehend aus (Ausgabe,Eingabe)-Paaren, um die Parameter θ des Modells

$$p(t|\mathbf{x},\theta)$$

so zu bestimmen, dass damit die Daten D gut erklärt werden.

Modell anpassen

Wie immer bei überwachten Lernproblemen verwenden wir Trainings-Daten

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq N\}$$

bestehend aus (Ausgabe, Eingabe)-Paaren, um die Parameter θ des Modells

$$p(t | \mathbf{x}, \theta)$$

so zu bestimmen, dass damit die Daten D gut erklärt werden.

Dazu können wir zum Beispiel die Maximum-Likelihood-Methode verwenden, indem wir die logarithmierte Likelihood

$$\ln p(\mathbf{t} | \mathbf{X}, \mathbf{w}, \beta) = \sum_{j=1}^N \ln \mathcal{N}(t_j | \phi^T(\mathbf{x}_j) \mathbf{w}, \beta^{-1}) = \frac{N}{2} \ln \beta - \frac{N}{2} \ln 2\pi - \frac{\beta}{2} \sum_{j=1}^N (t_j - \phi^T(\mathbf{x}_j) \mathbf{w})^2 \quad (2)$$

der Daten D in den Parametern $\theta = (\mathbf{w}, \beta)$ des Modells maximieren.

Voraussage berechnen

Schliesslich verwenden wir das an die Daten D angepasste Modell

$$p(t|\mathbf{x}, \boldsymbol{\theta}_D) = \mathcal{N}(t | \boldsymbol{\phi}^T(\mathbf{x}) \mathbf{w}_D, \beta_D^{-1})$$

um für neue Eingaben $\tilde{\mathbf{x}}$ eine Voraussage p für die Ausgabe t zu berechnen.

Voraussage berechnen

Schliesslich verwenden wir das an die Daten D angepasste Modell

$$p(t|\mathbf{x}, \boldsymbol{\theta}_D) = \mathcal{N}(t | \boldsymbol{\phi}^T(\mathbf{x}) \mathbf{w}_D, \beta_D^{-1})$$

um für neue Eingaben $\tilde{\mathbf{x}}$ eine Voraussage p für die Ausgabe t zu berechnen.

Falls die quadratische Abweichung

$$L_Q(t, p) = (t - p)^2$$

unser Mass für den Verlust bei der Voraussage p des wahren Wertes t ist, so würden wir für die Voraussage p zur Eingabe $\tilde{\mathbf{x}}$ schlicht den erwarteten Wert

$$p = E_{p(\cdot|\tilde{\mathbf{x}}, \boldsymbol{\theta}_D)}[t] = \boldsymbol{\phi}^T(\tilde{\mathbf{x}}) \mathbf{w}_D$$

wählen.

Klassifikation

Ein diskriminatives, parametrisches, probabilistisches Klassifikationsmodell

$$p(t|\mathbf{x}, \boldsymbol{\theta})$$

liefert die Verteilung der diskreten Ausgabe

$$t \in \{1, \dots, C\}$$

bedingt auf die Eingabe $\mathbf{x} \in \mathbb{R}^d$, wobei wir mit $\boldsymbol{\theta} \in \mathbb{R}^p$ die adaptierbaren Parameter des Modells bezeichnen.

Klassifikation

Ein diskriminatives, parametrisches, probabilistisches Klassifikationsmodell

$$p(t|\mathbf{x}, \boldsymbol{\theta})$$

liefert die Verteilung der diskreten Ausgabe

$$t \in \{1, \dots, C\}$$

bedingt auf die Eingabe $\mathbf{x} \in \mathbb{R}^d$, wobei wir mit $\boldsymbol{\theta} \in \mathbb{R}^p$ die adaptierbaren Parameter des Modells bezeichnen.

Für die Spezifikation der endlichen Verteilung wird sehr häufig die Softmax-Funktion verwendet.

Softmax-Funktion

Die **Softmax-Funktion**

$$\mathbf{S} : \mathbb{R}^C \longrightarrow \{ \mathbf{p} \in \mathbb{R}_+^C \mid p_1 + \dots + p_C = 1 \}$$

$$\mathbf{a} \mapsto \mathbf{S}(\mathbf{a}) = \begin{pmatrix} S_1(\mathbf{a}) \\ \vdots \\ S_C(\mathbf{a}) \end{pmatrix} \quad (3)$$

ist definiert durch die Komponenten

$$S_k(\mathbf{a}) = \frac{e^{a_k}}{\sum_{j=1}^C e^{a_j}}$$

und generiert aus einem beliebigen Vektor $\mathbf{a} \in \mathbb{R}^C$ den Wahrscheinlichkeitsvektor $\mathbf{S}(\mathbf{a}) \in \mathbb{R}^C$.

Lineares Klassifikationsmodell

Entgegen ihrem Namen ist die **logistische Regression** ein diskriminatives Klassifikationsmodell

$$p(t | \mathbf{x}, \mathbf{W}) = \text{Cat}_{\mathcal{S}(\mathbf{W}\phi(\mathbf{x}))}(t)$$

für Eingaben $\mathbf{x} \in \mathbb{R}^d$ und die Klassen $t \in \{1, \dots, C\}$, welches durch die Wahl der Familie der Basisfunktionen

$$\phi(\mathbf{x}) = (\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x}))^T$$

definiert wird und die Parameter

$$\mathbf{W} = (\mathbf{w}_1 \quad \mathbf{w}_2 \quad \dots \quad \mathbf{w}_C)^T = \begin{pmatrix} w_{1,0} & \dots & w_{1,M} \\ \vdots & & \vdots \\ w_{C,0} & \dots & w_{C,M} \end{pmatrix} \in \text{Mat}(C, M+1)$$

besitzt.

Lineares Klassifikationsmodell

Mit dem logistischen Regressionsmodell erhalten wir für die Voraussage-Wahrscheinlichkeit der Klasse $t = c$ an der Eingabe \mathbf{x} explizit die Formel

$$p(c|\mathbf{x}, \mathbf{W}) = \text{Cat}_{S(\mathbf{W}\phi(\mathbf{x}))}(c) = S_c(\mathbf{W}\phi(\mathbf{x})) = \frac{e^{\mathbf{w}_c^T \phi(\mathbf{x})}}{\sum_{k=1}^C e^{\mathbf{w}_k^T \phi(\mathbf{x})}},$$

wobei wir die Definition der Softmax-Funktion verwendet haben.

Lineares Klassifikationsmodell

Mit dem logistischen Regressionsmodell erhalten wir für die Voraussage-Wahrscheinlichkeit der Klasse $t = c$ an der Eingabe \mathbf{x} explizit die Formel

$$p(c | \mathbf{x}, \mathbf{W}) = \text{Cat}_{S(\mathbf{W}\phi(\mathbf{x}))}(c) = S_c(\mathbf{W}\phi(\mathbf{x})) = \frac{e^{\mathbf{w}_c^T \phi(\mathbf{x})}}{\sum_{k=1}^C e^{\mathbf{w}_k^T \phi(\mathbf{x})}},$$

wobei wir die Definition der Softmax-Funktion verwendet haben.

Bei der logistischen Regression kann die logarithmierte Likelihood

$$\begin{aligned} \ln p(\mathbf{t} | \mathbf{X}, \mathbf{W}) &= \sum_{n=1}^N \ln p(t_n | \mathbf{x}_n, \mathbf{W}) \\ &= \sum_{n=1}^N \ln S_{t_n}(\mathbf{W}\phi(\mathbf{x}_n)) = \sum_{n=1}^N \ln \frac{e^{\mathbf{w}_{t_n}^T \phi(\mathbf{x}_n)}}{\sum_{k=1}^C e^{\mathbf{w}_k^T \phi(\mathbf{x}_n)}} \end{aligned} \quad (4)$$

der Daten \mathbf{X}, \mathbf{t} im Gegensatz zur linearen Regression (2) nicht mehr geschlossen Maximiert werden.

Lineares Klassifikationsmodell

Mit dem logistischen Regressionsmodell erhalten wir für die Voraussage-Wahrscheinlichkeit der Klasse $t = c$ an der Eingabe \mathbf{x} explizit die Formel

$$p(c | \mathbf{x}, \mathbf{W}) = \text{Cat}_{\mathbf{S}(\mathbf{W}\phi(\mathbf{x}))}(c) = S_c(\mathbf{W}\phi(\mathbf{x})) = \frac{e^{\mathbf{w}_c^T \phi(\mathbf{x})}}{\sum_{k=1}^C e^{\mathbf{w}_k^T \phi(\mathbf{x})}},$$

wobei wir die Definition der Softmax-Funktion verwendet haben.

Bei der logistischen Regression kann die logarithmierte Likelihood

$$\begin{aligned} \ln p(\mathbf{t} | \mathbf{X}, \mathbf{W}) &= \sum_{n=1}^N \ln p(t_n | \mathbf{x}_n, \mathbf{W}) \\ &= \sum_{n=1}^N \ln S_{t_n}(\mathbf{W}\phi(\mathbf{x}_n)) = \sum_{n=1}^N \ln \frac{e^{\mathbf{w}_{t_n}^T \phi(\mathbf{x}_n)}}{\sum_{k=1}^C e^{\mathbf{w}_k^T \phi(\mathbf{x}_n)}} \end{aligned} \tag{4}$$

der Daten \mathbf{X}, \mathbf{t} im Gegensatz zur linearen Regression (2) nicht mehr geschlossen Maximiert werden.

Diese logartihmierte Likelihood ist jedoch noch immer eine konkave Funktion.

Inhalt

1 Motivation der Modellstruktur

- Regressionsmodell
- Klassifikationsmodell
- **Der Fluch der Dimension**
- Struktur des Modells

Konstruierte Features

Obwohl die allgemeinen Resultate der linearen Regressions- und Klassifikationsmodelle unabhängig von der Wahl der Basis-Funktionen

$$\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x})$$

sind, hat diese Wahl im allgemeinen grossen Einfluss auf die Verallgemeinerungsqualität des trainierten Modells.

Konstruierte Features

Obwohl die allgemeinen Resultate der linearen Regressions- und Klassifikationsmodelle unabhängig von der Wahl der Basis-Funktionen

$$\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x})$$

sind, hat diese Wahl im allgemeinen grossen Einfluss auf die Verallgemeinerungsqualität des trainierten Modells.

In gewissen Fällen haben wir genügend gesichertes Wissen über den die Daten generierenden Prozess, um eine Familie von Basis-Funktionen

$$\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x})$$

mit guter Verallgemeinerungsqualität zu finden.

Konstruierte Features

Obwohl die allgemeinen Resultate der linearen Regressions- und Klassifikationsmodelle unabhängig von der Wahl der Basis-Funktionen

$$\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x})$$

sind, hat diese Wahl im allgemeinen grossen Einfluss auf die Verallgemeinerungsqualität des trainierten Modells.

In gewissen Fällen haben wir genügend gesichertes Wissen über den die Daten generierenden Prozess, um eine Familie von Basis-Funktionen

$$\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \dots, \phi_M(\mathbf{x})$$

mit guter Verallgemeinerungsqualität zu finden.

Eine solche Familie von Basisfunktionen von Hand zu konstruieren, wird **feature engineering** genannt und ist auch bei guter Kenntnis des Prozesses meistens kein triviales oder mechanisches Unterfangen.

Generische Basisfamilien

Ein anderer Zugang zur Wahl der Basis-Funktionen basiert auf der Verwendung von generischen Basisfamilien wie Spline-Funktionen, Gauss-Kernen oder Ähnlichem.

Generische Basisfamilien

Ein anderer Zugang zur Wahl der Basis-Funktionen basiert auf der Verwendung von generischen Basisfamilien wie Spline-Funktionen, Gauss-Kernen oder Ähnlichem.

Diese generischen Basisfamilien verwenden häufig Basis-Funktionen mit bezüglich der euklidischen Norm lokalisiertem Träger in Zeit/Ort und/oder Frequenz.

Generische Basisfamilien

Ein anderer Zugang zur Wahl der Basis-Funktionen basiert auf der Verwendung von generischen Basisfamilien wie Spline-Funktionen, Gauss-Kernen oder Ähnlichem.

Diese generischen Basisfamilien verwenden häufig Basis-Funktionen mit bezüglich der euklidischen Norm lokalisiertem Träger in Zeit/Ort und/oder Frequenz.

Bei niedrig-dimensionaler Eingabe \mathbf{x} kann dieser Ansatz alleine oder in Kombination mit der Selektion von Basisfunktionen gute Resultate liefern.

Fluch der Dimension

Die Anzahl von in der euklidischen Norm lokalisierten Basisfunktionen steigt jedoch in der Dimension der Eingabe x exponentiell an und damit auch die Menge an Daten, die zur Vermeidung von Überanpassung benötigt werden.

Fluch der Dimension

Die Anzahl von in der euklidischen Norm lokalisierten Basisfunktionen steigt jedoch in der Dimension der Eingabe \mathbf{x} exponentiell an und damit auch die Menge an Daten, die zur Vermeidung von Überanpassung benötigt werden.

Schon für moderate Dimension d der Eingabe \mathbf{x} ist diese benötigte Menge an Daten D realistischerweise nicht beschaffbar.

Fluch der Dimension

Die Anzahl von in der euklidischen Norm lokalisierten Basisfunktionen steigt jedoch in der Dimension der Eingabe \mathbf{x} exponentiell an und damit auch die Menge an Daten, die zur Vermeidung von Überanpassung benötigt werden.

Schon für moderate Dimension d der Eingabe \mathbf{x} ist diese benötigte Menge an Daten D realistischweise nicht beschaffbar.

Die Problematik liegt eigentlich sogar noch etwas tiefer und wird als sogenannter Fluch der Dimension bezeichnet. ([prml, Section 1.4],[esl, Section 2.5])

Fluch der Dimension

Die Anzahl von in der euklidischen Norm lokalisierten Basisfunktionen steigt jedoch in der Dimension der Eingabe x exponentiell an und damit auch die Menge an Daten, die zur Vermeidung von Überanpassung benötigt werden.

Schon für moderate Dimension d der Eingabe x ist diese benötigte Menge an Daten D realistischweise nicht beschaffbar.

Die Problematik liegt eigentlich sogar noch etwas tiefer und wird als sogenannter Fluch der Dimension bezeichnet. ([prml, Section 1.4],[esl, Section 2.5])

Sehr vereinfacht formuliert liegen gleichverteilte Daten in hochdimensionalen Räumen fast immer am Rande der Träger von in der euklidischen Norm lokalisierten Basisfunktionen.

Adaptive Basisfamilie

In realen Anwendungen sind hochdimensionale Eingaben \mathbf{x} natürlich meistens nicht gleichverteilt, sondern konzentrieren sich vielmehr häufig in der Nähe einer im allgemeinen nichtlinearen, immersierten Untermannigfaltigkeit viel kleinerer Dimension als der umgebende Raum.

Adaptive Basisfamilie

In realen Anwendungen sind hochdimensionale Eingaben \mathbf{x} natürlich meistens nicht gleichverteilt, sondern konzentrieren sich vielmehr häufig in der Nähe einer im allgemeinen nichtlinearen, immersierten Untermannigfaltigkeit viel kleinerer Dimension als der umgebende Raum.

Zudem definiert die Sensitivität der Ausgabe t bezüglich der Eingabe \mathbf{x} eine Metrik auf der Datenmannigfaltigkeit, die sich potentiell stark von der induzierten euklidischen Metrik unterscheidet.

Adaptive Basisfamilie

In realen Anwendungen sind hochdimensionale Eingaben \mathbf{x} natürlich meistens nicht gleichverteilt, sondern konzentrieren sich vielmehr häufig in der Nähe einer im allgemeinen nichtlinearen, immersierten Untermannigfaltigkeit viel kleinerer Dimension als der umgebende Raum.

Zudem definiert die Sensitivität der Ausgabe t bezüglich der Eingabe \mathbf{x} eine Metrik auf der Datenmannigfaltigkeit, die sich potentiell stark von der induzierten euklidischen Metrik unterscheidet.

Eine endliche Anzahl statischer und lokalisierter Basisfunktionen wird nicht zufällig eine Nachbarschaft dieser Datenmannigfaltigkeit überdecken und die Abhängigkeit der Ausgabe t von der Eingabe \mathbf{x} lokal genügend gut auflösen.

Adaptive Basisfamilie

In realen Anwendungen sind hochdimensionale Eingaben \mathbf{x} natürlich meistens nicht gleichverteilt, sondern konzentrieren sich vielmehr häufig in der Nähe einer im allgemeinen nichtlinearen, immersierten Untermannigfaltigkeit viel kleinerer Dimension als der umgebende Raum.

Zudem definiert die Sensitivität der Ausgabe t bezüglich der Eingabe \mathbf{x} eine Metrik auf der Datenmannigfaltigkeit, die sich potentiell stark von der induzierten euklidischen Metrik unterscheidet.

Eine endliche Anzahl statischer und lokalisierter Basisfunktionen wird nicht zufällig eine Nachbarschaft dieser Datenmannigfaltigkeit überdecken und die Abhängigkeit der Ausgabe t von der Eingabe \mathbf{x} lokal genügend gut auflösen.

Mit neuronalen Netzen begegnen wir diesem Problem, indem wir endlich viele, jedoch adaptive Basisfunktionen betrachten in der Hoffnung, dass sich diese während des Lernens an die Datenmannigfaltigkeit und die Metrik anpassen.

Inhalt

- 1 Motivation der Modellstruktur
 - Regressionsmodell
 - Klassifikationsmodell
 - Der Fluch der Dimension
 - Struktur des Modells

Vereinheitlichung über die Ausgabeaktivierung

Wir können sowohl bei linearen Regressionsmodellen wie auch bei linearen Klassifikationsmodellen die Parameter der (auf die Eingabe \mathbf{x} bedingten) Ausgabeverteilung als Transformation einer Linearkombination

$$\varphi(\mathbf{W}\phi(\mathbf{x})) \quad (5)$$

von Basisfunktionen

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))$$

mit den Gewichten \mathbf{W} als optimierbare Parameter schreiben.

Vereinheitlichung über die Ausgabeaktivierung

Wir können sowohl bei linearen Regressionsmodellen wie auch bei linearen Klassifikationsmodellen die Parameter der (auf die Eingabe \mathbf{x} bedingten) Ausgabeverteilung als Transformation einer Linearkombination

$$\varphi(\mathbf{W}\phi(\mathbf{x})) \quad (5)$$

von Basisfunktionen

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))$$

mit den Gewichten \mathbf{W} als optimierbare Parameter schreiben.

Für Regressionsmodelle verwenden wir als sogenannte **Ausgabe-Aktivierungsfunktion** φ schlicht die Identitätsfunktion

$$\varphi = \text{id.}$$

und für Klassifikationsmodelle verwenden wir als **Ausgabe-Aktivierungsfunktion** φ natürlich die Softmax-Funktion

$$\varphi = \mathbf{S}.$$

Vereinheitlichung über die Ausgabeaktivierung

Wir können sowohl bei linearen Regressionsmodellen wie auch bei linearen Klassifikationsmodellen die Parameter der (auf die Eingabe \mathbf{x} bedingten) Ausgabeverteilung als Transformation einer Linearkombination

$$\varphi(\mathbf{W}\phi(\mathbf{x})) \quad (5)$$

von Basisfunktionen

$$\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \dots, \phi_m(\mathbf{x}))$$

mit den Gewichten \mathbf{W} als optimierbare Parameter schreiben.

Für Regressionsmodelle verwenden wir als sogenannte **Ausgabe-Aktivierungsfunktion** φ schlicht die Identitätsfunktion

$$\varphi = \text{id.}$$

und für Klassifikationsmodelle verwenden wir als **Ausgabe-Aktivierungsfunktion** φ natürlich die Softmax-Funktion

$$\varphi = \mathbf{S}.$$

Auf diese Art können wir beide wichtigen Anwendungsfälle mit einer universellen Form abdecken.

Adaptive Basisfunktionen

Indem wir die Basisfunktionen adaptierbar machen, also eine Familie parametrisierter Basisfunktionen

$$\phi(\mathbf{x}) = \mathbf{f}_{1,\theta_1}(\mathbf{x})$$

betrachten, erhalten wir für eine beliebig gewählte Ausgabe-Aktivierungsfunktion φ ein flexibleres Modell

$$\varphi(\mathbf{W}\mathbf{f}_{1,\theta_1}(\mathbf{x})),$$

sofern wir zusätzlich zu den Parametern \mathbf{W} auch die Parameter θ_1 der Basisfamilie in den Lernprozess mit einbeziehen.

Adaptive Basisfunktionen

Indem wir die Basisfunktionen adaptierbar machen, also eine Familie parametrisierter Basisfunktionen

$$\phi(\mathbf{x}) = \mathbf{f}_{1,\theta_1}(\mathbf{x})$$

betrachten, erhalten wir für eine beliebig gewählte Ausgabe-Aktivierungsfunktion φ ein flexibleres Modell

$$\varphi(\mathbf{W}\mathbf{f}_{1,\theta_1}(\mathbf{x})),$$

sofern wir zusätzlich zu den Parametern \mathbf{W} auch die Parameter θ_1 der Basisfamilie in den Lernprozess mit einbeziehen.

Natürlich können wir im Prinzip eine beliebig parametrisierte Basisfamilie \mathbf{f}_{1,θ_1} verwenden, bei den neuronalen Netzen behalten wir jedoch die charakteristische Form einer parametrisierten affinen Transformation der Eingabe gefolgt von einer fixen nichtlinearen Transformation bei.

Inhalt

2 Neuronale Netze

- Basisexpansionsschicht
- Zweischichtige neuronale Netze
- Mehrschichtige neuronale Netze
- Aktivierungsfunktionen
- Parametrisierung der affinen Transformation
- Beispielanwendungen

Neuronale Netze

Nachdem wir die charakteristische Struktur der Basis-Expansionen benannt haben, können wir die mathematische Struktur der neuronalen Netze definieren und den Informationsfluss im Schichtmodell erläutern.

Neuronale Netze

Nachdem wir die charakteristische Struktur der Basis-Expansionen benannt haben, können wir die mathematische Struktur der neuronalen Netze definieren und den Informationsfluss im Schichtmodell erläutern.

Danach werden wir häufig verwendete Aktivierungsfunktionen und Parametrisierungen kurz thematisieren und anschliessend je ein Regressions- und ein Klassifikationsbeispiel besprechen.

Inhalt

2 Neuronale Netze

- **Basisexpansionsschicht**
- Zweischichtige neuronale Netze
- Mehrschichtige neuronale Netze
- Aktivierungsfunktionen
- Parametrisierung der affinen Transformation
- Beispielanwendungen

Die Form der Basisexpansion

Bei neuronalen Netzen behalten wir die Form einer affinen Transformation

$$\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1$$

der Eingabe $\mathbf{x} \in \mathbb{R}^{d_1}$ mit einer **Gewichtsmatrix** und einem **Verschiebungsvektor**

$$\mathbf{W}_1 \in \mathbb{R}^{r_1 \times d_1} \text{ und } \mathbf{b}_1 \in \mathbb{R}^{r_1}$$

gefolgt von einer fixen, nichtlinearen Transformation mit einer sogenannten **Aktivierungsfunktion**

$$\varphi_1 : \mathbb{R}^{r_1} \longrightarrow \mathbb{R}^{r_1}$$

auch für die Basisfamilie bei.

Die Basisexpansion

Auf diese Art erhalten wir die charakteristische Form

$$\begin{aligned} \mathbf{f}_{1,\theta_1} : \mathbb{R}^{d_1} &\longrightarrow \mathbb{R}^{r_1} \\ \mathbf{x} \mapsto \mathbf{f}_{1,\theta_1}(\mathbf{x}) &= \varphi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \end{aligned} \tag{6}$$

der **Basisexpansion**, wobei wir die freien Parameter der Gewichtsmatrix \mathbf{W}_1 und des Verschiebungsvektors \mathbf{b}_1 mit $\theta_1 \in \mathbb{R}^{p_1}$ bezeichnen.

Inhalt

- 2 Neuronale Netze
 - Basisexpansionsschicht
 - **Zweischichtige neuronale Netze**
 - Mehrschichtige neuronale Netze
 - Aktivierungsfunktionen
 - Parametrisierung der affinen Transformation
 - Beispielanwendungen

Neuronales Netz mit einer Basisexpansionen

Da die Ausgabeschicht (5) des neuronalen Netzes dieselbe Form

$$\begin{aligned} \mathbf{f}_{2,\theta_2} : \mathbb{R}^{d_2} &\longrightarrow \mathbb{R}^{r_2} \\ \mathbf{x} \mapsto \mathbf{f}_{2,\theta_2}(\mathbf{x}) &= \varphi_2(\mathbf{W}_2\mathbf{x} + \mathbf{b}_2) \end{aligned}$$

hat wie eine Basisexpansion, wobei wir natürlich die zur Aufgabe passende Ausgabe-Aktivierungsfunktion φ_2 wählen, erhalten wir für kompatible Dimensionen $r_1 = d_2$ explizit die Funktion

$$\begin{aligned} \mathbf{f}_{2,\theta_2} \circ \mathbf{f}_{1,\theta_1} : \mathbb{R}^{d_1} &\longrightarrow \mathbb{R}^{r_2} \\ \mathbf{x} \mapsto \mathbf{f}_{2,\theta_2}(\mathbf{f}_{1,\theta_1}(\mathbf{x})) &= \varphi_2(\mathbf{W}_2\varphi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \end{aligned}$$

für die Berechnung der **Ausgabe-Aktivierung** zur Eingabe \mathbf{x} .

Informationsfluss im neuronalen Netz

Der Informationsfluss dieser Berechnung kann mit einem Schichten-Modell veranschaulicht werden, wobei die **Eingabeschicht** die Eingabe

$$\mathbf{z}_0 = \mathbf{x}$$

enthält, die **verborgene Schicht** die Werte

$$\mathbf{z}_1 = \mathbf{f}_{1,\theta_1}(\mathbf{z}_0) = \varphi_1(\mathbf{W}_1\mathbf{z}_0 + \mathbf{b}_1)$$

der ersten Basisexpansion und die **Ausgabeschicht** die Werte

$$\mathbf{z}_2 = \mathbf{f}_{2,\theta_2}(\mathbf{z}_1) = \varphi_2(\mathbf{W}_2\mathbf{z}_1 + \mathbf{b}_2)$$

der Ausgabe-Aktivierung. (1)

Modellierungsstärke

Unser Ziel, ein flexibleres Modell zu konstruieren, hätten wir an dieser Stelle im Prinzip schon erreicht.

Modellierungsstärke

Unser Ziel, ein flexibleres Modell zu konstruieren, hätten wir an dieser Stelle im Prinzip schon erreicht.

Genauer kann ein Modell der eingeführten Art mit einer Basisexpansion (also einer verborgenen Schicht) schon mit einer ziemlich einfachen Aktivierungsfunktion und der linearen Ausgabe-Aktivierung jede stetige Funktion auf einer kompakten Menge gleichmässig approximieren.

Modellierungsstärke

Unser Ziel, ein flexibleres Modell zu konstruieren, hätten wir an dieser Stelle im Prinzip schon erreicht.

Genauer kann ein Modell der eingeführten Art mit einer Basisexpansion (also einer verborgenen Schicht) schon mit einer ziemlich einfachen Aktivierungsfunktion und der linearen Ausgabe-Aktivierung jede stetige Funktion auf einer kompakten Menge gleichmässig approximieren.

Dazu muss natürlich die Dimension $r_1 = d_2$ der verborgenen Schicht genügend gross gemacht werden. (Siehe [prml, Seite 230],[pml1, 13.2.5])

Modellierungsstärke

Unser Ziel, ein flexibleres Modell zu konstruieren, hätten wir an dieser Stelle im Prinzip schon erreicht.

Genauer kann ein Modell der eingeführten Art mit einer Basisexpansion (also einer verborgenen Schicht) schon mit einer ziemlich einfachen Aktivierungsfunktion und der linearen Ausgabe-Aktivierung jede stetige Funktion auf einer kompakten Menge gleichmässig approximieren.

Dazu muss natürlich die Dimension $r_1 = d_2$ der verborgenen Schicht genügend gross gemacht werden. (Siehe [prml, Seite 230],[pml1, 13.2.5])

In zahlreichen Anwendungen hat es sich jedoch als lohnend erwiesen, mehrere verborgene Schichten, also iterierte Basisexpansion, zu verwenden.

Inhalt

- 2 Neuronale Netze
 - Basisexpansionsschicht
 - Zweischichtige neuronale Netze
 - **Mehrschichtige neuronale Netze**
 - Aktivierungsfunktionen
 - Parametrisierung der affinen Transformation
 - Beispielanwendungen

Definition (K-Schichtiges neuronales Netz)

Indem wir mehrere Schichten

$$\begin{aligned} \mathbf{f}_{j,\theta_j} &: \mathbb{R}^{d_j} \longrightarrow \mathbb{R}^{r_j} \\ \mathbf{x} \mapsto \mathbf{f}_{j,\theta_j}(\mathbf{x}) &= \varphi_j(\mathbf{W}_j\mathbf{x} + \mathbf{b}_j) \end{aligned}$$

der Basisexpansionsform (6) mit $r_j = d_{j+1}$ komponieren, erhalten wir ein Modell

$$\begin{aligned} \mathbf{f}_\theta &: \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{r_K} \\ \mathbf{x} \mapsto \mathbf{f}_\theta(\mathbf{x}) &= \mathbf{f}_{K,\theta_K} \circ \dots \circ \mathbf{f}_{1,\theta_1}(\mathbf{x}), \end{aligned} \tag{7}$$

welches als **mehrschichtiges neuronales Netz** mit $K - 1$ verborgenen Schichten bezeichnet wird .

Definition (K-Schichtiges neuronales Netz)

Indem wir mehrere Schichten

$$\begin{aligned} \mathbf{f}_{j,\theta_j} &: \mathbb{R}^{d_j} \longrightarrow \mathbb{R}^{r_j} \\ \mathbf{x} \mapsto \mathbf{f}_{j,\theta_j}(\mathbf{x}) &= \varphi_j(\mathbf{W}_j\mathbf{x} + \mathbf{b}_j) \end{aligned}$$

der Basisexpansionsform (6) mit $r_j = d_{j+1}$ komponieren, erhalten wir ein Modell

$$\begin{aligned} \mathbf{f}_\theta &: \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{r_K} \\ \mathbf{x} \mapsto \mathbf{f}_\theta(\mathbf{x}) &= \mathbf{f}_{K,\theta_K} \circ \dots \circ \mathbf{f}_{1,\theta_1}(\mathbf{x}), \end{aligned} \tag{7}$$

welches als **mehrschichtiges neuronales Netz** mit $K - 1$ verborgenen Schichten bezeichnet wird .

Dieses Modell besitzt die akkumulierten Parameter

$$\theta = (\theta_1, \dots, \theta_K) \in \mathbb{R}^{p_1} \times \dots \times \mathbb{R}^{p_K}$$

der affinen Transformationen aller Schichten.

Inhalt

- 2 Neuronale Netze
 - Basisexpansionsschicht
 - Zweischichtige neuronale Netze
 - Mehrschichtige neuronale Netze
 - **Aktivierungsfunktionen**
 - Parametrisierung der affinen Transformation
 - Beispielanwendungen

Ausgabe-Aktivierungsfunktion

Wir haben schon erwähnt, dass die Ausgabe-Aktivierungsfunktion φ_K eines K-schichtigen neuronalen Netzes durch die Anwendung bestimmt wird.

Ausgabe-Aktivierungsfunktion

Wir haben schon erwähnt, dass die Ausgabe-Aktivierungsfunktion φ_K eines K-schichtigen neuronalen Netzes durch die Anwendung bestimmt wird.

Für Regressionsanwendungen werden wir meistens die triviale Aktivierung in der Form der Identitätsfunktion

$$\varphi_K(\mathbf{a}) = \mathbf{a}$$

für den Mittelwertparameter des probabilistischen Regressionsmodells verwenden.

Ausgabe-Aktivierungsfunktion

Wir haben schon erwähnt, dass die Ausgabe-Aktivierungsfunktion φ_K eines K-schichtigen neuronalen Netzes durch die Anwendung bestimmt wird.

Für Regressionsanwendungen werden wir meistens die triviale Aktivierung in der Form der Identitätsfunktion

$$\varphi_K(\mathbf{a}) = \mathbf{a}$$

für den Mittelwertparameter des probabilistischen Regressionsmodells verwenden.

Für eine Klassifikationsanwendung mit C-Klassen wird hingegen die Softmax-Funktion

$$\varphi_K(\mathbf{a}) = \mathbf{S}(\mathbf{a})$$

als Ausgabe-Aktivierungsfunktion für die Wahrscheinlichkeiten der kategorischen Verteilung über die Klassen eingesetzt.

Ausgabe-Aktivierungsfunktion

Natürlich können wir auch mehrere verwandte Klassifikationsanwendungen simultan modellieren, also mehrere Softmax-Funktionen auf Teile der Ausgabe-Schicht anwenden.

Ausgabe-Aktivierungsfunktion

Natürlich können wir auch mehrere verwandte Klassifikationsanwendungen simultan modellieren, also mehrere Softmax-Funktionen auf Teile der Ausgabe-Schicht anwenden.

Bei Regressionsanwendungen können wir zudem auch die Ausgabepräzision des probabilistischen Modells eingabeabhängig modellieren (heteroskedastische Regression), was auch mittels einer geeigneten Ausgabe-Aktivierungsfunktion erreicht werden kann.

Aktivierungsfunktion für verborgene Schichten

Für die Aktivierungsfunktionen der verborgenen Schichten wird häufig ein und dieselbe eindimensionale Aktivierungsfunktion

$$\varphi : \mathbb{R} \longrightarrow \mathbb{R}$$

auf alle Komponenten

$$\varphi(\mathbf{a}) = \begin{pmatrix} \varphi(a_1) \\ \vdots \\ \varphi(a_r) \end{pmatrix}$$

der affin transformierten Eingabe

$$\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^r$$

angewendet.

Peceptron

Das ursprüngliche Perceptron¹ hat die Heavyside-Aktivierungsfunktion

$$\varphi(a) = h(a) = \begin{cases} 1 & a > 0 \\ 0 & \text{sonst} \end{cases}$$

verwendet, welche jedoch mangels Gradienteninformation für das Lernen der Gewichte und der Verschiebungsvektoren der Schichten nicht geeignet ist.

¹Rosenblatt 1958.

Sigmoidale Aktivierungsfunktion

Später wurden Funktionen wie die logistische Sigmoid-Funktion (3)

$$\varphi(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$

oder der Tangens Hyperbolicus als glatte Approximation der Heavyside-Aktivierungsfunktion $h(a)$ verwendet.

Sigmoidale Aktivierungsfunktion

Später wurden Funktionen wie die logistische Sigmoid-Funktion (3)

$$\varphi(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$

oder der Tangens Hyperbolicus als glatte Approximation der Heavyside-Aktivierungsfunktion $h(a)$ verwendet.

Obwohl diese Funktionen Gradienteninformation liefern, haben sie den Nachteil, dass die Ableitung mit zunehmender Distanz zu 0 schnell sehr klein wird und folglich die Sensitivität auf Veränderung in der affinen Transformation in diesem Bereich minimal ist.

Aktuelle Aktivierungsfunktionen

Zwei noch heute sehr verbreitete Aktivierungsfunktionen sind die **Rectified Linear Unit (ReLU)** und die **leaky Rectified Linear Unit (LReLU)**

$$\text{ReLU}(a) = \max(a, 0) \quad \text{und} \quad \text{LReLU}_\alpha(a) = \max(a, \alpha a)$$

mit $0 < \alpha < 1$, wobei nur letztere das Problem der verschwindenden Gradienteninformation überall löst. (4)

Aktuelle Aktivierungsfunktionen

Zwei noch heute sehr verbreitete Aktivierungsfunktionen sind die **Rectified Linear Unit** (ReLU) und die **leaky Rectified Linear Unit** (LReLU)

$$\text{ReLU}(a) = \max(a, 0) \quad \text{und} \quad \text{LReLU}_\alpha(a) = \max(a, \alpha a)$$

mit $0 < \alpha < 1$, wobei nur letztere das Problem der verschwindenden Gradienteninformation überall löst. (4)

Ein weitere populäre Aktivierungsfunktion ist die **Exponential Linear Unit** (ELU)

$$\text{ELU}(a) = \begin{cases} a & a > 0 \\ \alpha(e^a - 1) & \text{sonst} \end{cases}$$

mit dem Parameter $\alpha > 0$.

Aktuelle Aktivierungsfunktionen

Zwei noch heute sehr verbreitete Aktivierungsfunktionen sind die **Rectified Linear Unit (ReLU)** und die **leaky Rectified Linear Unit (LReLU)**

$$\text{ReLU}(a) = \max(a, 0) \text{ und } \text{LReLU}_\alpha(a) = \max(a, \alpha a)$$

mit $0 < \alpha < 1$, wobei nur letztere das Problem der verschwindenden Gradienteninformation überall löst. (4)

Ein weitere populäre Aktivierungsfunktion ist die **Exponential Linear Unit (ELU)**

$$\text{ELU}(a) = \begin{cases} a & a > 0 \\ \alpha(e^a - 1) & \text{sonst} \end{cases}$$

mit dem Parameter $\alpha > 0$.

Für weitere Informationen über Aktivierungsfunktionen verweisen wir auf [dl, 6.3].

Inhalt

- 2 Neuronale Netze
 - Basisexpansionsschicht
 - Zweischichtige neuronale Netze
 - Mehrschichtige neuronale Netze
 - Aktivierungsfunktionen
 - **Parametrisierung der affinen Transformation**
 - Beispielanwendungen

Vollbesetzte und unbesetzte Schichten

Wenn die adaptiven Parameter θ_1 einer Basisexpansions-Schicht

$$\begin{aligned} \mathbf{f}_{1,\theta_1} : \mathbb{R}^{d_1} &\longrightarrow \mathbb{R}^{r_1} \\ \mathbf{x} &\mapsto \mathbf{f}_{1,\theta_1}(\mathbf{x}) = \varphi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \end{aligned}$$

eines neuronalen Netzwerkes die volle Matrix und den vollen Verschiebungsvektor umfassen

$$\theta_1 = (\mathbf{W}_1, \mathbf{b}_1) \in \mathbb{R}^{r_1 \times d_1} \times \mathbb{R}^{r_1},$$

dann nennen wir dies eine **vollbesetzte** Schicht (**dense layer**) .

Vollbesetzte und unbesetzte Schichten

Wenn die adaptiven Parameter θ_1 einer Basisexpansions-Schicht

$$\begin{aligned} \mathbf{f}_{1,\theta_1} : \mathbb{R}^{d_1} &\longrightarrow \mathbb{R}^{r_1} \\ \mathbf{x} &\mapsto \mathbf{f}_{1,\theta_1}(\mathbf{x}) = \varphi_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \end{aligned}$$

eines neuronalen Netzwerkes die volle Matrix und den vollen Verschiebungsvektor umfassen

$$\theta_1 = (\mathbf{W}_1, \mathbf{b}_1) \in \mathbb{R}^{r_1 \times d_1} \times \mathbb{R}^{r_1},$$

dann nennen wir dies eine **vollbesetzte** Schicht (**dense layer**) .

Das andere Extrem sind Schichten, die gar keine adaptiven Parameter haben, wie zum Beispiel sogenannte pooling Schichten, die Mittelwerte bilden, um die Dimension zu reduzieren.

Strukturierte Parametrisierungen

Natürlich werden auch feiner strukturierte Parametrisierungen mit weniger Freiheitsgraden eingesetzt.

Strukturierte Parametrisierungen

Natürlich werden auch feiner strukturierte Parametrisierungen mit weniger Freiheitsgraden eingesetzt.

Ein bekanntes Beispiel dieses Ansatzes sind neuronale Netze mit Schichten, die Faltungen der Eingabe mit einem parametrisierten Kern berechnen (CNN).

Strukturierte Parametrisierungen

Natürlich werden auch feiner strukturierte Parametrisierungen mit weniger Freiheitsgraden eingesetzt.

Ein bekanntes Beispiel dieses Ansatzes sind neuronale Netze mit Schichten, die Faltungen der Eingabe mit einem parametrisierten Kern berechnen (CNN).

Dazu werden schicht strukturierte Matrizen wie zum Beispiel die Matrix

$$W_1 = \begin{pmatrix} \theta_1 & 0 & \dots & & & 0 \\ \theta_2 & \theta_1 & 0 & & & \\ \theta_1 & \theta_2 & \theta_1 & 0 & & \\ 0 & \theta_1 & \theta_2 & \theta_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ & & 0 & \theta_1 & \theta_2 & \theta_1 \\ & & & 0 & \theta_1 & \theta_2 \\ 0 & & \dots & 0 & \theta_1 & \end{pmatrix}$$

eingesetzt.

Freiheiten des sequentiellen Schichtmodells

Indem wir Direktverbindungen von gewissen Eingaben zu gewissen Ausgaben mit der Identität als affine Transformation und mit der Identität als Aktivierungsfunktion auf dieser Komponente verwenden, können wir auch sogenannte «skip connections», die gewisse Schichten umgehen, in unserem sequentiellen Schichtmodell abbilden.

Freiheiten des sequentiellen Schichtmodells

Indem wir Direktverbindungen von gewissen Eingaben zu gewissen Ausgaben mit der Identität als affine Transformation und mit der Identität als Aktivierungsfunktion auf dieser Komponente verwenden, können wir auch sogenannte «skip connections», die gewisse Schichten umgehen, in unserem sequentiellen Schichtmodell abbilden.

Durch die Verwendung partitionierter Matrizen können wir zudem das Netz bei beliebigen Tiefen in mehrere Teile aufspalten, die sich auf Teilprobleme spezialisieren. (5)

Freiheiten des sequentiellen Schichtmodells

Indem wir Direktverbindungen von gewissen Eingaben zu gewissen Ausgaben mit der Identität als affine Transformation und mit der Identität als Aktivierungsfunktion auf dieser Komponente verwenden, können wir auch sogenannte «skip connections», die gewisse Schichten umgehen, in unserem sequentiellen Schichtmodell abbilden.

Durch die Verwendung partitionierter Matrizen können wir zudem das Netz bei beliebigen Tiefen in mehrere Teile aufspalten, die sich auf Teilprobleme spezialisieren. (5)

Obwohl wir das sequentielle Schichtmodell auf diese Art auch für komplexere Netzarchitekturen beibehalten können, ist es irgendwann für die Modellierung natürlicher, das Netz als gerichteten, azyklischen Graphen von Basis-Expansionen (6) aufzufassen. (6)

Inhalt

- 2 Neuronale Netze
 - Basisexpansionsschicht
 - Zweischichtige neuronale Netze
 - Mehrschichtige neuronale Netze
 - Aktivierungsfunktionen
 - Parametrisierung der affinen Transformation
 - Beispielanwendungen

Eine Klassifikationsanwendung

Wir betrachten das bekannte Iris-Datenset, bei welchem wir drei verschiedenen Blumenarten $t \in \{1, 2, 3\}$ basierend auf vier gemessenen Blütenmerkmalen $\mathbf{x} \in \mathbb{R}^4$ voraussagen müssen, wobei wir 150 Trainingsdaten

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq n_D = 150\}$$

zur Verfügung haben.

Eine Klassifikationsanwendung

Wir betrachten das bekannte Iris-Datenset, bei welchem wir drei verschiedenen Blumenarten $t \in \{1, 2, 3\}$ basierend auf vier gemessenen Blütenmerkmalen $\mathbf{x} \in \mathbb{R}^4$ voraussagen müssen, wobei wir 150 Trainingsdaten

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq n_D = 150\}$$

zur Verfügung haben.

Wir wählen eine Netzarchitektur mit zwei verborgenen, vollbesetzten Schichten

$$\mathbf{f}_{1, \theta_1} : \mathbb{R}^4 \longrightarrow \mathbb{R}^{r_1}$$

$$\mathbf{f}_{2, \theta_2} : \mathbb{R}^{r_1} \longrightarrow \mathbb{R}^{r_1}$$

der wählbaren Breite r_1 und mit einer komponentenweisen Aktivierungsfunktion wie zum Beispiel ReLU.

Eine Klassifikationsanwendung

Wir betrachten das bekannte Iris-Datenset, bei welchem wir drei verschiedenen Blumenarten $t \in \{1, 2, 3\}$ basierend auf vier gemessenen Blütenmerkmalen $\mathbf{x} \in \mathbb{R}^4$ voraussagen müssen, wobei wir 150 Trainingsdaten

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq n_D = 150\}$$

zur Verfügung haben.

Wir wählen eine Netzarchitektur mit zwei verborgenen, vollbesetzten Schichten

$$\mathbf{f}_{1, \theta_1} : \mathbb{R}^4 \longrightarrow \mathbb{R}^{r_1}$$

$$\mathbf{f}_{2, \theta_2} : \mathbb{R}^{r_1} \longrightarrow \mathbb{R}^{r_1}$$

der wählbaren Breite r_1 und mit einer komponentenweisen Aktivierungsfunktion wie zum Beispiel ReLU.

Darüber legen wir eine vollbesetzte Aktivierungsschicht

$$\mathbf{f}_{3, \theta_3} : \mathbb{R}^{r_1} \longrightarrow \mathbb{R}^3$$

mit der Softmax-Funktion $\varphi_3 = \mathbf{S}$ als Ausgabe-Aktivierungsfunktion.

Eine Klassifikationsanwendung

Damit erhalten wir insgesamt das diskriminative, probabilistische Klassifikations-Modell

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = \text{Cat}_{\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})}(t),$$

wobei wir mit

$$\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}_{3, \boldsymbol{\theta}_3} \circ \mathbf{f}_{2, \boldsymbol{\theta}_2} \circ \mathbf{f}_{1, \boldsymbol{\theta}_1}(\mathbf{x})$$

das neuronale Netz mit den Parametern

$$\boldsymbol{\theta} = (\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_3, \mathbf{b}_3) \in (\mathbb{R}^{r_1 \times 4} \times \mathbb{R}^{r_1}) \times (\mathbb{R}^{r_1 \times r_1} \times \mathbb{R}^{r_1}) \times (\mathbb{R}^{r_1 \times 3} \times \mathbb{R}^3)$$

bezeichnen. (7)

Eine Regressionsanwendung

In einigen Regressionsanwendungen ist die Varianz der Ausgabeverteilung offensichtlich von der Eingabe \mathbf{x} abhängig (8) und ein diskriminatives Regressionsmodell

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(t | f_{\mu, \boldsymbol{\theta}}(\mathbf{x}), f_{\sigma, \boldsymbol{\theta}}^2(\mathbf{x})),$$

bei welchem sowohl der Erwartungswert als auch die Standard-Abweichung von der Eingabe \mathbf{x} abhängen, ist prinzipiell geeignet, diese Variabilität zu erfassen.

Eine Regressionsanwendung

In einigen Regressionsanwendungen ist die Varianz der Ausgabeverteilung offensichtlich von der Eingabe \mathbf{x} abhängig (8) und ein diskriminatives Regressionsmodell

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(t | f_{\mu, \boldsymbol{\theta}}(\mathbf{x}), f_{\sigma, \boldsymbol{\theta}}^2(\mathbf{x})),$$

bei welchem sowohl der Erwartungswert als auch die Standard-Abweichung von der Eingabe \mathbf{x} abhängen, ist prinzipiell geeignet, diese Variabilität zu erfassen.

Als Erweiterung des linearen Regressionsmodells könnten wir ein lineares Modell für den Erwartungswert und ein mit der sogenannten Softplus-Funktion

$$\sigma_+(a) = \ln(1 + e^a)$$

transformiertes lineares Modell für die Standard-Abweichung ansetzen.

Eine Regressionsanwendung

Allgemeiner können wir auch ein neuronales Netz für diese beiden Ausgabeverteilungsparameter verwenden.

Eine Regressionsanwendung

Allgemeiner können wir auch ein neuronales Netz für diese beiden Ausgabeverteilungsparameter verwenden.

Um die Flexibilität des Netzes richtig zu dosieren, fassen wir die untersten Schichten des Netzes als gelernte Features auf, welche sowohl für die Berechnung des Erwartungswertes als auch für die Berechnung der Standard-Abweichung eingesetzt werden können und darum gemeinsam verwendet werden.

Eine Regressionsanwendung

Allgemeiner können wir auch ein neuronales Netz für diese beiden Ausgabeverteilungsparameter verwenden.

Um die Flexibilität des Netzes richtig zu dosieren, fassen wir die untersten Schichten des Netzes als gelernte Features auf, welche sowohl für die Berechnung des Erwartungswertes als auch für die Berechnung der Standard-Abweichung eingesetzt werden können und darum gemeinsam verwendet werden.

Danach spalten wir das Netz in die zwei Ausgabeteile (sogenannte heads)

$$f_{\mu, \theta}(\mathbf{x}) \text{ und } f_{\sigma, \theta}(\mathbf{x})$$

auf (5).

Eine Regressionsanwendung

Allgemeiner können wir auch ein neuronales Netz für diese beiden Ausgabeverteilungsparameter verwenden.

Um die Flexibilität des Netzes richtig zu dosieren, fassen wir die untersten Schichten des Netzes als gelernte Features auf, welche sowohl für die Berechnung des Erwartungswertes als auch für die Berechnung der Standard-Abweichung eingesetzt werden können und darum gemeinsam verwendet werden.

Danach spalten wir das Netz in die zwei Ausgabeteile (sogenannte heads)

$$f_{\mu,\theta}(\mathbf{x}) \text{ und } f_{\sigma,\theta}(\mathbf{x})$$

auf (5).

Für den Erwartungswert $f_{\mu,\theta}(\mathbf{x})$ verwenden wir dabei die Identität als Ausgabe-Aktivierung und für die Standard-Abweichung $f_{\sigma,\theta}(\mathbf{x})$ verwenden wir die Soft-Plus-Funktion.

Inhalt

- 3 Trainieren von neuronalen Netzen
 - Die Ableitung der logarithmierten Likelihood
 - Die Ableitung einer Basisexpansions-Schicht
 - Die Ableitung des Netzes nach den Parametern (Backpropagation)
 - Stochastischer Gradienten-Abstieg
 - Regularisierung neuronaler Netze

Trainieren von neuronalen Netzen

Neuronale Netze werden sehr häufig eingesetzt, um die Parameter der Ausgabeverteilung eines diskriminativen probabilistischen Modells

$$p(t|\mathbf{x}, \theta)$$

aus der Eingabe \mathbf{x} zu berechnen.

Trainieren von neuronalen Netzen

Neuronale Netze werden sehr häufig eingesetzt, um die Parameter der Ausgabeverteilung eines diskriminativen probabilistischen Modells

$$p(t|\mathbf{x}, \theta)$$

aus der Eingabe \mathbf{x} zu berechnen.

In diesen Anwendungen wird das probabilistische Modell meistens mit der Maximum-Likelihood-Methode oder der Maximum-Posterior-Methode an die Trainingsdaten

$$D = \{(t_k, \mathbf{x}_k) | 1 \leq k \leq n_D\}$$

angepasst.

Trainieren von neuronalen Netzen

Neuronale Netze werden sehr häufig eingesetzt, um die Parameter der Ausgabeverteilung eines diskriminativen probabilistischen Modells

$$p(t|\mathbf{x}, \theta)$$

aus der Eingabe \mathbf{x} zu berechnen.

In diesen Anwendungen wird das probabilistische Modell meistens mit der Maximum-Likelihood-Methode oder der Maximum-Posterior-Methode an die Trainingsdaten

$$D = \{(t_k, \mathbf{x}_k) | 1 \leq k \leq n_D\}$$

angepasst.

Im Kontext von neuronalen Netzen ist es üblich, die negative logarithmierte Likelihood

$$\text{NLL}(\theta) = -\ln p(\mathbf{t}|\mathbf{X}, \theta) = -\sum_{k=1}^{n_D} \ln p(t_k|\mathbf{x}_k, \theta) \quad (8)$$

in den Parametern θ des Netzes zu minimieren, wobei natürlich allfällige Regularisierungsterme (MAP-Methode) zur logarithmierten Likelihood addiert werden.

Trainieren von neuronalen Netzen

Im Gegensatz zu den linearen Regressionsmodellen (2) kann diese Minimierung nicht geschlossen analytisch berechnet werden.

Trainieren von neuronalen Netzen

Im Gegensatz zu den linearen Regressionsmodellen (2) kann diese Minimierung nicht geschlossen analytisch berechnet werden.

Im Gegensatz zur logistischen Regression (4) ist diese negative Likelihood (8) nicht mehr konvex und hat deshalb in der Regel viele lokale nicht optimale Minimalstellen.

Trainieren von neuronalen Netzen

Im Gegensatz zu den linearen Regressionsmodellen (2) kann diese Minimierung nicht geschlossen analytisch berechnet werden.

Im Gegensatz zur logistischen Regression (4) ist diese negative Likelihood (8) nicht mehr konvex und hat deshalb in der Regel viele lokale nicht optimale Minimalstellen.

Wie nahe am globalen Minimum eine numerisch gefundene lokale Minimalstelle liegt, hängt natürlich stark vom verwendeten Optimierungs-Algorithmus und auch den dazu benötigten Startparametern θ_0 ab.

Trainieren von neuronalen Netzen

Im Gegensatz zu den linearen Regressionsmodellen (2) kann diese Minimierung nicht geschlossen analytisch berechnet werden.

Im Gegensatz zur logistischen Regression (4) ist diese negative Likelihood (8) nicht mehr konvex und hat deshalb in der Regel viele lokale nicht optimale Minimalstellen.

Wie nahe am globalen Minimum eine numerisch gefundene lokale Minimalstelle liegt, hängt natürlich stark vom verwendeten Optimierungs-Algorithmus und auch den dazu benötigten Startparametern θ_0 ab.

Da die neuronalen Netze sehr flexible Modelle sind, ist insbesondere die Verwendung von Regularisierungen zur Verhinderung von Überanpassung und bestenfalls eine sehr grosse Menge D an Trainingsdaten ziemlich wichtig.

Trainieren von neuronalen Netzen

Die meisten numerischen Optimierungs-Verfahren verwenden Gradienteninformation und wir werden als nächstes darlegen, wie diese Gradienten berechnet werden können.

Trainieren von neuronalen Netzen

Die meisten numerischen Optimierungs-Verfahren verwenden Gradienteninformation und wir werden als nächstes darlegen, wie diese Gradienten berechnet werden können.

Danach wenden wir uns dem am häufigsten verwendeten Optimierungsverfahren zu.

Trainieren von neuronalen Netzen

Die meisten numerischen Optimierungs-Verfahren verwenden Gradienteninformation und wir werden als nächstes darlegen, wie diese Gradienten berechnet werden können.

Danach wenden wir uns dem am häufigsten verwendeten Optimierungsverfahren zu.

Schliesslich besprechen wir Techniken zur Regularisierung von neuronalen Netzen.

Inhalt

- 3 Trainieren von neuronalen Netzen
 - Die Ableitung der logarithmierten Likelihood
 - Die Ableitung einer Basisexpansions-Schicht
 - Die Ableitung des Netzes nach den Parametern (Backpropagation)
 - Stochastischer Gradienten-Abstieg
 - Regularisierung neuronaler Netze

Die Ableitung

Da die negative logarithmierte Likelihood (8) eine Summe über alle Datenpunkte und die Ableitung linear ist, bestimmen wir nur die Ableitung

$$-\ln p(t|\mathbf{x}, \boldsymbol{\theta})$$

der negativen logarithmierten Likelihood eines Datenpunktes (t, \mathbf{x}) und summieren anschliessend schlicht diese Ableitungen.

Die Ableitung

Da die negative logarithmierte Likelihood (8) eine Summe über alle Datenpunkte und die Ableitung linear ist, bestimmen wir nur die Ableitung

$$-\ln p(t|\mathbf{x}, \boldsymbol{\theta})$$

der negativen logarithmierten Likelihood eines Datenpunktes (t, \mathbf{x}) und summieren anschliessend schlicht diese Ableitungen.

Da wir vielfach mit einem neuronalen Netz die Parameter der Ausgabeverteilung eines diskriminativen probabilistischen Modells berechnen, benötigen wir zur Berechnung dieser Ableitung die Ableitung der Ausgabe des neuronalen Netzes

$$f_{\boldsymbol{\theta}}$$

nach den Parametern $\boldsymbol{\theta}$.

Die Ableitung bei Regressionsanwendungen

Wir betrachten ein diskriminatives Regressionsmodell

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(t | f_{\boldsymbol{\theta}}(\mathbf{x}), \beta^{-1}),$$

bei welchem der Erwartungswert der gaussischen Ausgabeverteilung durch ein neuronales Netz

$$f(\mathbf{x}, \boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathbf{x})$$

mit der Identität aus Ausgabe-Aktivierungsfunktion berechnet wird.

Die Ableitung bei Regressionsanwendungen

Wir betrachten ein diskriminatives Regressionsmodell

$$p(t|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(t | f_{\boldsymbol{\theta}}(\mathbf{x}), \beta^{-1}),$$

bei welchem der Erwartungswert der gaussischen Ausgabeverteilung durch ein neuronales Netz

$$f(\mathbf{x}, \boldsymbol{\theta}) = f_{\boldsymbol{\theta}}(\mathbf{x})$$

mit der Identität aus Ausgabe-Aktivierungsfunktion berechnet wird.

Damit erhalten wir die negative logarithmierte Likelihood

$$\begin{aligned} -\ln p(t|\mathbf{x}, \boldsymbol{\theta}) &= -\ln \mathcal{N}(t | f_{\boldsymbol{\theta}}(\mathbf{x}), \beta^{-1}) \\ &= \frac{-\ln \beta}{2} + \frac{\ln 2\pi}{2} + \frac{\beta}{2} (f(\mathbf{x}, \boldsymbol{\theta}) - t)^2 \end{aligned}$$

und folglich mit Hilfe der Kettenregel die Ableitung

$$-\frac{\partial \ln p(t|\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \beta (f(\mathbf{x}, \boldsymbol{\theta}) - t) \frac{\partial f}{\partial \boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\theta})$$

nach den Parametern $\boldsymbol{\theta}$ des Modells.

Die Ableitung bei Klassifikationsanwendungen

Wir betrachten das diskriminative Klassifikations-Modell

$$p(t | \mathbf{x}, \boldsymbol{\theta}) = \text{Cat}_{\tilde{f}_{\boldsymbol{\theta}}(\mathbf{x})}(t),$$

mit einem neuronalen Netz mit der Softmax Ausgabe-Aktivierungsfunktion.

Die Ableitung bei Klassifikationsanwendungen

Wir betrachten das diskriminative Klassifikations-Modell

$$p(t | \mathbf{x}, \boldsymbol{\theta}) = \text{Cat}_{\tilde{\mathbf{f}}_{\boldsymbol{\theta}}(\mathbf{x})}(t),$$

mit einem neuronalen Netz mit der Softmax Ausgabe-Aktivierungsfunktion.

Um die Ableitungsformel einfacher zu gestalten, schreiben wir das Modell äquivalent in der Form

$$p(t | \mathbf{x}, \boldsymbol{\theta}) = \text{Cat}_{\mathbf{S}(\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}))}(t)$$

mit der Identität als Ausgabe-Aktivierungsfunktion des neuronalen Netzes $\mathbf{f}_{\boldsymbol{\theta}}$ und erhalten damit die negative logarithmierte Likelihood

$$-\ln p(t | \mathbf{x}, \boldsymbol{\theta}) = -\ln S_t(\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})).$$

Die Ableitung bei Klassifikationsanwendungen

Mit der partiellen Ableitung

$$\frac{\partial \ln S_k(\mathbf{a})}{\partial a_j} = \frac{S_k(\mathbf{a}) (\delta_{k,j} - S_j(\mathbf{a}))}{S_k(\mathbf{a})} = (\delta_{k,j} - S_j(\mathbf{a}))$$

des Logarithmus der k-ten Komponente der Softmax-Funktion (3) erhalten wir die Ableitung

$$\frac{\partial \ln S_k}{\partial \mathbf{a}}(\mathbf{a}) = (\mathbf{e}_k - \mathbf{S}(\mathbf{a}))^T.$$

Die Ableitung bei Klassifikationsanwendungen

Mit der partiellen Ableitung

$$\frac{\partial \ln S_k(\mathbf{a})}{\partial a_j} = \frac{S_k(\mathbf{a}) (\delta_{k,j} - S_j(\mathbf{a}))}{S_k(\mathbf{a})} = (\delta_{k,j} - S_j(\mathbf{a}))$$

des Logarithmus der k-ten Komponente der Softmax-Funktion (3) erhalten wir die Ableitung

$$\frac{\partial \ln S_k}{\partial \mathbf{a}}(\mathbf{a}) = (\mathbf{e}_k - \mathbf{S}(\mathbf{a}))^T.$$

Damit berechnen wir schliesslich die Ableitung

$$-\frac{\partial \ln p(t|\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{\partial \ln S_t(\mathbf{f}(\mathbf{x}, \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} = (\mathbf{S}(\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})) - \mathbf{e}_t)^T \frac{\partial \mathbf{f}}{\partial \boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\theta})$$

der negativen logarithmierten Likelihood nach den Parametern $\boldsymbol{\theta}$ des Modells.

Inhalt

- 3 Trainieren von neuronalen Netzen
 - Die Ableitung der logarithmierten Likelihood
 - Die Ableitung einer Basisexpansions-Schicht
 - Die Ableitung des Netzes nach den Parametern (Backpropagation)
 - Stochastischer Gradienten-Abstieg
 - Regularisierung neuronaler Netze

Die Ableitung

Die für das Training von neuronalen Netzen unentbehrliche Ableitung der Ausgabe des K -schichtigen neuronalen Netzwerkes (7)

$$\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{r_K}$$

$$\mathbf{x} \mapsto \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}_{K, \boldsymbol{\theta}_K} \circ \cdots \circ \mathbf{f}_{1, \boldsymbol{\theta}_1}(\mathbf{x}_0),$$

an der Eingabe \mathbf{x} nach den Parametern $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K)$ des Modells berechnen wir auch schlicht mit Hilfe der Kettenregel.

Die Ableitung

Die für das Training von neuronalen Netzen unentbehrliche Ableitung der Ausgabe des K -schichtigen neuronalen Netzwerkes (7)

$$\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{r_K}$$

$$\mathbf{x} \mapsto \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}_{K, \boldsymbol{\theta}_K} \circ \cdots \circ \mathbf{f}_{1, \boldsymbol{\theta}_1}(\mathbf{x}_0),$$

an der Eingabe \mathbf{x} nach den Parametern $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K)$ des Modells berechnen wir auch schlicht mit Hilfe der Kettenregel.

Dazu benötigen wir insbesondere die Ableitungen

$$\frac{\partial \mathbf{f}_k}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\theta}) \in \mathbb{R}^{r_k \times d_k} \quad \text{und} \quad \frac{\partial \mathbf{f}_k}{\partial \boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\theta}) \in \mathbb{R}^{r_k \times p_k}$$

der einzelnen Schichten

$$\mathbf{f}_{k, \boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}_k(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R}^{d_k} \times \mathbb{R}^{p_k} \longrightarrow \mathbb{R}^{r_k}$$

nach der Eingabe \mathbf{x} respektive den Parametern $\boldsymbol{\theta}$ der Schicht.

Die Ableitung

Die für das Training von neuronalen Netzen unentbehrliche Ableitung der Ausgabe des K -schichtigen neuronalen Netzwerkes (7)

$$\mathbf{f}_{\boldsymbol{\theta}} : \mathbb{R}^{d_1} \longrightarrow \mathbb{R}^{r_K}$$

$$\mathbf{x} \mapsto \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}_{K, \boldsymbol{\theta}_K} \circ \cdots \circ \mathbf{f}_{1, \boldsymbol{\theta}_1}(\mathbf{x}_0),$$

an der Eingabe \mathbf{x} nach den Parametern $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K)$ des Modells berechnen wir auch schicht mit Hilfe der Kettenregel.

Dazu benötigen wir insbesondere die Ableitungen

$$\frac{\partial \mathbf{f}_k}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\theta}) \in \mathbb{R}^{r_k \times d_k} \quad \text{und} \quad \frac{\partial \mathbf{f}_k}{\partial \boldsymbol{\theta}}(\mathbf{x}, \boldsymbol{\theta}) \in \mathbb{R}^{r_k \times p_k}$$

der einzelnen Schichten

$$\mathbf{f}_{k, \boldsymbol{\theta}}(\mathbf{x}) = \mathbf{f}_k(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R}^{d_k} \times \mathbb{R}^{p_k} \longrightarrow \mathbb{R}^{r_k}$$

nach der Eingabe \mathbf{x} respektive den Parametern $\boldsymbol{\theta}$ der Schicht.

Diese Ableitungen lassen sich glücklicherweise für viele verwendeten Schichten leicht mit expliziten Formeln darstellen.

Die Ableitung einer Schicht

Wir betrachten eine voll besetzte Schicht

$$\mathbf{f}_\theta : \mathbb{R}^d \longrightarrow \mathbb{R}^r$$

$$\mathbf{x} \mapsto \mathbf{f}_\theta(\mathbf{x}) = \mathbf{f}(\mathbf{x}, \theta) = \varphi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

mit den Parametern

$$\theta = (\mathbf{W}, \mathbf{b}) \in \mathbb{R}^{r \times d} \times \mathbb{R}^r,$$

und einer komponentenweisen Aktivierungsfunktion

$$\varphi : \mathbb{R}^r \longrightarrow \mathbb{R}^r$$

$$\mathbf{a} \mapsto \varphi(\mathbf{a}) = \begin{pmatrix} \varphi(a_1) \\ \vdots \\ \varphi(a_r) \end{pmatrix}.$$

Die Ableitung einer Schicht

Diese Aktivierungsfunktion hat die diagonale Ableitung

$$\frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) = \begin{pmatrix} \varphi'(a_1) & & 0 \\ & \ddots & \\ 0 & & \varphi'(a_r) \end{pmatrix} = \text{diag}(\varphi'(a_1), \dots, \varphi'(a_r))$$

und damit berechnen wir die Ableitung

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}, \boldsymbol{\theta}) = \frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) \mathbf{W},$$

der Schicht $\mathbf{f}_{\boldsymbol{\theta}}$ nach der Eingabe \mathbf{x} , wobei natürlich die Aktivierung

$$\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

verwendet wird.

Die Ableitung einer Schicht

Für die partiellen Ableitungen nach den Einträgen $w_{i,j}$ der Gewichtsmatrix \mathbf{W} erhalten wir

$$\frac{\partial \mathbf{f}}{\partial w_{i,j}}(\mathbf{x}, \boldsymbol{\theta}) = \frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) \mathbf{E}_{i,j} \mathbf{x} = \varphi'(a_i) \mathbf{e}_i x_j,$$

wobei wir die Einheits-Matrix $\mathbf{E}_{i,j}$ mit dem i, j -Eintrag 1 und Nulleinträgen sonst sowie den Einheitsvektor \mathbf{e}_i mit 1 an i -ter Stelle und null sonst verwendet haben.

Die Ableitung einer Schicht

Für die partiellen Ableitungen nach den Einträgen $w_{i,j}$ der Gewichtsmatrix \mathbf{W} erhalten wir

$$\frac{\partial \mathbf{f}}{\partial w_{i,j}}(\mathbf{x}, \boldsymbol{\theta}) = \frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) \mathbf{E}_{i,j} \mathbf{x} = \varphi'(a_i) \mathbf{e}_i x_j,$$

wobei wir die Einheits-Matrix $\mathbf{E}_{i,j}$ mit dem i, j -Eintrag 1 und Nulleinträgen sonst sowie den Einheitsvektor \mathbf{e}_i mit 1 an i -ter Stelle und null sonst verwendet haben.

Analog bestimmen wir die partiellen Ableitungen

$$\frac{\partial \mathbf{f}}{\partial b_i}(\mathbf{x}, \boldsymbol{\theta}) = \frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) \mathbf{e}_i = \varphi'(a_i) \mathbf{e}_i$$

nach den Verschiebungsvektoren b_i .

Die Ableitung einer Schicht

Für die partiellen Ableitungen nach den Einträgen $w_{i,j}$ der Gewichtsmatrix \mathbf{W} erhalten wir

$$\frac{\partial \mathbf{f}}{\partial w_{i,j}}(\mathbf{x}, \boldsymbol{\theta}) = \frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) \mathbf{E}_{i,j} \mathbf{x} = \varphi'(a_i) \mathbf{e}_i x_j,$$

wobei wir die Einheits-Matrix $\mathbf{E}_{i,j}$ mit dem i, j -Eintrag 1 und Nulleinträgen sonst sowie den Einheitsvektor \mathbf{e}_i mit 1 an i -ter Stelle und null sonst verwendet haben.

Analog bestimmen wir die partiellen Ableitungen

$$\frac{\partial \mathbf{f}}{\partial b_i}(\mathbf{x}, \boldsymbol{\theta}) = \frac{\partial \varphi}{\partial \mathbf{a}}(\mathbf{a}) \mathbf{e}_i = \varphi'(a_i) \mathbf{e}_i$$

nach den Verschiebungsvektoren b_i .

Für nicht voll besetzte Schichten können wir im Prinzip einfach diese Ableitungen zusammen mit der Kettenregel verwenden.

Inhalt

- 3 Trainieren von neuronalen Netzen
 - Die Ableitung der logarithmierten Likelihood
 - Die Ableitung einer Basisexpansions-Schicht
 - Die Ableitung des Netzes nach den Parametern (Backpropagation)
 - Stochastischer Gradienten-Abstieg
 - Regularisierung neuronaler Netze

Die Ableitung des Netzes nach den Parametern

Um die Anwendung der Kettenregel leichter einzusehen, schreiben wir die Abhängigkeit des neuronalen Netzes f_{θ} von den Parametern expliziter

$$f(x, \theta_1, \dots, \theta_K) = f_K \left(\underbrace{f_{K-1} \left(\dots f_2 \left(\underbrace{f_1(x, \theta_1), \theta_2}_{z_1} \right) \dots, \theta_{K-1} \right)}_{z_2} \right), \theta_K$$

$$\underbrace{\left(\dots f_2 \left(\underbrace{f_1(x, \theta_1), \theta_2}_{z_1} \right) \dots, \theta_{K-1} \right)}_{z_{K-1}}$$

aus.

Die Ableitung des Netzes nach den Parametern

Damit ergibt sich für die Parameter θ_j der verborgenen Schichten $1 \leq j < K$ gemäss der Kettenregel die Ableitung

$$\frac{\partial \mathbf{f}}{\partial \theta_j}(\mathbf{x}, \theta_1, \dots, \theta_K) = \frac{\partial \mathbf{f}_K}{\partial \mathbf{x}}(\mathbf{z}_{K-1}, \theta_K) \cdots \frac{\partial \mathbf{f}_{j+1}}{\partial \mathbf{x}}(\mathbf{z}_j, \theta_{j+1}) \frac{\partial \mathbf{f}_j}{\partial \theta}(\mathbf{z}_{j-1}, \theta_j) \quad (9)$$

an der Eingabe \mathbf{x} , wobei die Produkte natürlich Matrixprodukte sind und wir die Ausgaben

$$\mathbf{z}_k = \mathbf{f}_{k, \theta_k} \circ \cdots \circ \mathbf{f}_{1, \theta_1}(\mathbf{z}_0) \quad \text{und} \quad \mathbf{z}_0 = \mathbf{x}$$

der verborgenen Schichten verwendet haben.

Die Ableitung des Netzes nach den Parametern

Damit ergibt sich für die Parameter θ_j der verborgenen Schichten $1 \leq j < K$ gemäss der Kettenregel die Ableitung

$$\frac{\partial \mathbf{f}}{\partial \theta_j}(\mathbf{x}, \theta_1, \dots, \theta_K) = \frac{\partial \mathbf{f}_K}{\partial \mathbf{x}}(\mathbf{z}_{K-1}, \theta_K) \cdots \frac{\partial \mathbf{f}_{j+1}}{\partial \mathbf{x}}(\mathbf{z}_j, \theta_{j+1}) \frac{\partial \mathbf{f}_j}{\partial \theta}(\mathbf{z}_{j-1}, \theta_j) \quad (9)$$

an der Eingabe \mathbf{x} , wobei die Produkte natürlich Matrixprodukte sind und wir die Ausgaben

$$\mathbf{z}_k = \mathbf{f}_{k, \theta_k} \circ \cdots \circ \mathbf{f}_{1, \theta_1}(\mathbf{z}_0) \quad \text{und} \quad \mathbf{z}_0 = \mathbf{x}$$

der verborgenen Schichten verwendet haben.

Für die Ausgabeschicht erhalten wir zudem direkt die Ableitung

$$\frac{\partial \mathbf{f}}{\partial \theta_K}(\mathbf{x}, \theta_1, \dots, \theta_K) = \frac{\partial \mathbf{f}_K}{\partial \theta}(\mathbf{z}_{K-1}, \theta_K).$$

Auswertung der Ableitung

Die Matrixprodukte der Ableitung (9) können prozedural von links nach rechts oder von rechts nach links berechnet werden, wobei der Aufwand nicht derselbe ist.

Auswertung der Ableitung

Die Matrixprodukte der Ableitung (9) können prozedural von links nach rechts oder von rechts nach links berechnet werden, wobei der Aufwand nicht derselbe ist.

Da die Dimension der Ausgabe im allgemeinen wesentlich kleiner als die Dimension der Eingabe ist, erweist es sich als effizienter, das Produkt (9) prozedural von links nach rechts, also mit der Ausgabe-Schicht startend rückwärts ins Netz fortschreitend zu berechnen.

Auswertung der Ableitung

Die Matrixprodukte der Ableitung (9) können prozedural von links nach rechts oder von rechts nach links berechnet werden, wobei der Aufwand nicht derselbe ist.

Da die Dimension der Ausgabe im allgemeinen wesentlich kleiner als die Dimension der Eingabe ist, erweist es sich als effizienter, das Produkt (9) prozedural von links nach rechts, also mit der Ausgabe-Schicht startend rückwärts ins Netz fortschreitend zu berechnen.

Diese Art der Berechnung erlaubt zudem, Zwischenprodukte optimal wiederverwenden zu können.

Backpropagation

Dieser Ableitungsinformationsfluss rückwärts ins Netz erklärt die Bezeichnung **Backpropagation**² für diese Ableitungsberechnung (9).

²Bryson, Ho 1969

Backpropagation

Dieser Ableitungsinformationsfluss rückwärts ins Netz erklärt die Bezeichnung **Backpropagation**² für diese Ableitungsberechnung (9).

Die für die Auswertung der Jakobi-Matrizen benötigten Ausgaben

$$\mathbf{z}_k$$

der verborgenen Schichten werden dabei natürlich zuerst bestimmt, indem mit der Eingabe \mathbf{x} die Ausgabe des Netzes vorwärts berechnet wird.

²Bryson, Ho 1969

Backpropagation

Dieser Ableitungsinformationsfluss rückwärts ins Netz erklärt die Bezeichnung **Backpropagation**² für diese Ableitungsberechnung (9).

Die für die Auswertung der Jakobi-Matrizen benötigten Ausgaben

$$\mathbf{z}_k$$

der verborgenen Schichten werden dabei natürlich zuerst bestimmt, indem mit der Eingabe \mathbf{x} die Ausgabe des Netzes vorwärts berechnet wird.

Die Verallgemeinerung dieser Ableitungsberechnung auf beliebige gerichtete, azyklische Berechnungs-Graphen ist unter der Bezeichnung automatische Differentiation bekannt.

²Bryson, Ho 1969

Inhalt

- 3 Trainieren von neuronalen Netzen
 - Die Ableitung der logarithmierten Likelihood
 - Die Ableitung einer Basisexpansions-Schicht
 - Die Ableitung des Netzes nach den Parametern (Backpropagation)
 - **Stochastischer Gradienten-Abstieg**
 - Regularisierung neuronaler Netze

NLL Batch-Ableitung

Für die numerische Minimierung der negativen logarithmierten Likelihood

$$\text{NLL}(\boldsymbol{\theta}) = -\ln p(\mathbf{t}|\mathbf{X}, \boldsymbol{\theta}) = -\sum_{k=1}^{n_D} \ln p(t_k | \mathbf{x}_k, \boldsymbol{\theta})$$

gemäss (8) benötigen wir die Ableitung

$$\frac{d\text{NLL}}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) = -\sum_{k=1}^{n_D} \frac{\partial \ln p(t_k | \mathbf{x}_k, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad (10)$$

nach den Parametern $\boldsymbol{\theta}$.

NLL Batch-Ableitung

Für die numerische Minimierung der negativen logarithmierten Likelihood

$$\text{NLL}(\boldsymbol{\theta}) = -\ln p(\mathbf{t}|\mathbf{X}, \boldsymbol{\theta}) = -\sum_{k=1}^{n_D} \ln p(t_k | \mathbf{x}_k, \boldsymbol{\theta})$$

gemäss (8) benötigen wir die Ableitung

$$\frac{d\text{NLL}}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) = -\sum_{k=1}^{n_D} \frac{\partial \ln p(t_k | \mathbf{x}_k, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad (10)$$

nach den Parametern $\boldsymbol{\theta}$.

Die einzelnen Summanden der Ableitung berechnen wir natürlich mit der Backpropagation (9), was jedoch für die grossen Datenmengen n_D , die zum Training von flexiblen neuronalen Netzen benötigt werden, einen erheblichen Aufwand darstellt.

NLL Mini-Batch-Ableitung

Indem wir für die Berechnung dieser Ableitung nicht alle Datenpunkte

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq n_D\},$$

sondern nur eine zufällige Auswahl (**Mini-Batch**)

$$B = \{(t_{k_j}, \mathbf{x}_{k_j}) \mid 1 \leq j \leq n_B\} \subset D$$

wesentlich kleineren Umfangs $n_B = |B| \ll n_D$ verwenden, erhalten wir eine Annäherung

$$\frac{d\text{NLL}}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) \cong \frac{n_D}{n_B} \frac{d\text{NLL}_B}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) = -\frac{n_D}{n_B} \sum_{k=1}^{n_B} \frac{\partial \ln p(t_{k_j} \mid \mathbf{x}_{k_j}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad (11)$$

an die exakte Ableitung (10).

NLL Mini-Batch-Ableitung

Indem wir für die Berechnung dieser Ableitung nicht alle Datenpunkte

$$D = \{(t_k, \mathbf{x}_k) \mid 1 \leq k \leq n_D\},$$

sondern nur eine zufällige Auswahl (**Mini-Batch**)

$$B = \{(t_{k_j}, \mathbf{x}_{k_j}) \mid 1 \leq j \leq n_B\} \subset D$$

wesentlich kleineren Umfangs $n_B = |B| \ll n_D$ verwenden, erhalten wir eine Annäherung

$$\frac{d\text{NLL}}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) \cong \frac{n_D}{n_B} \frac{d\text{NLL}_B}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) = -\frac{n_D}{n_B} \sum_{k=1}^{n_B} \frac{\partial \ln p(t_{k_j} \mid \mathbf{x}_{k_j}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad (11)$$

an die exakte Ableitung (10).

Diese Annäherung ist natürlich entsprechend schneller berechnet und liefert bei fixem Umfang n_B und zufälliger Wahl der Datenpunkte in den Mini-Batches B eine unverzerrte Schätzung

$$\frac{d\text{NLL}}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) = \frac{n_D}{n_B} E_B \left[\frac{d\text{NLL}_B}{d\boldsymbol{\theta}}(\boldsymbol{\theta}) \right]$$

der Ableitung der logarithmierten Likelihood (10).

Stochastischer Gradienten-Abstieg

Die Verwendung von stochastischen, unverzerrten Schätzungen

$$\nabla_B f(\boldsymbol{\theta})$$

des Gradienten

$$\nabla f(\boldsymbol{\theta})$$

einer Funktion f zusammen mit dem simplen, iterativen Gradienten-Abstiegsverfahren

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_B f(\boldsymbol{\theta}_t)$$

ist unter der Bezeichnung **stochastischer Gradienten-Abstieg (stochastic gradient descent (SGD))** bekannt.

Stochastischer Gradienten-Abstieg

Die Verwendung von stochastischen, unverzerrten Schätzungen

$$\nabla_B f(\theta)$$

des Gradienten

$$\nabla f(\theta)$$

einer Funktion f zusammen mit dem simplen, iterativen Gradienten-Abstiegsverfahren

$$\theta_{t+1} = \theta_t - \eta_t \nabla_B f(\theta_t)$$

ist unter der Bezeichnung **stochastischer Gradienten-Abstieg (stochastic gradient descent (SGD))** bekannt.

Für die Garantie der lokalen Konvergenz dieser Methode gegen eine isolierte lokale Minimumsstelle muss die Funktion f gewisse Regularitätseigenschaften haben und die **Schrittweiten** $\eta_t > 0$ müssen in geeigneter Geschwindigkeit gegen null konvergieren

$$\eta_t \rightarrow 0.$$

SDG-Verfeinerungen

Obwohl die theoretische Konvergenzrate von SGD gegenüber der Verwendung des exakten Gradienten kleiner ist, also theoretisch mehr Iterationen bis zur Konvergenz benötigt werden, können die einzelnen Iterationen wesentlich schneller berechnet werden und dieser Vorteil überwiegt häufig.

SDG-Verfeinerungen

Obwohl die theoretische Konvergenzrate von SGD gegenüber der Verwendung des exakten Gradienten kleiner ist, also theoretisch mehr Iterationen bis zur Konvergenz benötigt werden, können die einzelnen Iterationen wesentlich schneller berechnet werden und dieser Vorteil überwiegt häufig.

Um die SGD-Methode zu verbessern, werden einerseits Techniken zur Verringerung der Varianz der Ableitungsschätzung sowie andererseits Vorkonditionierungen und die adaptive Schrittweitensteuerung eingesetzt.

SDG-Verfeinerungen

Obwohl die theoretische Konvergenzrate von SGD gegenüber der Verwendung des exakten Gradienten kleiner ist, also theoretisch mehr Iterationen bis zur Konvergenz benötigt werden, können die einzelnen Iterationen wesentlich schneller berechnet werden und dieser Vorteil überwiegt häufig.

Um die SGD-Methode zu verbessern, werden einerseits Techniken zur Verringerung der Varianz der Ableitungsschätzung sowie andererseits Vorkonditionierungen und die adaptive Schrittweitensteuerung eingesetzt.

Ein prominenter Vertreter ist der ADAM-Minimierer (Adaptive Moment Estimation). ([adam])

SDG auf DNN

Für die Minimierung der negativen logarithmierten Likelihood von neuronalen Netzen werden aktuell fast ausschliesslich Versionen der SGD-Methode mit Mini-Batch- Gradienten (11) eingesetzt, wobei die Schrittweite η_t in diesem Kontext die **Lernrate** genannt wird.

SDG auf DNN

Für die Minimierung der negativen logarithmierten Likelihood von neuronalen Netzen werden aktuell fast ausschliesslich Versionen der SGD-Methode mit Mini-Batch- Gradienten (11) eingesetzt, wobei die Schrittweite η_t in diesem Kontext die **Lernrate** genannt wird.

Nebst der theoretischen Schwierigkeit, dass die negative logarithmierte Likelihood nicht konvex ist und folglich die Initialisierung der Gewichte des Netzes wichtig ist, müssen für den erfolgreichen Einsatz auch noch einige spezifische Besonderheiten beachtet werden, für die wir auf [pml1, 13.4] und [dl, 8] verweisen.

Inhalt

- 3 Trainieren von neuronalen Netzen
 - Die Ableitung der logarithmierten Likelihood
 - Die Ableitung einer Basisexpansions-Schicht
 - Die Ableitung des Netzes nach den Parametern (Backpropagation)
 - Stochastischer Gradienten-Abstieg
 - Regularisierung neuronaler Netze

MAP-Regularisierung

Da neuronale Netze sehr flexible Modelle mit enorm vielen Parametern sind, ist die Überanpassung ein sehr ernstzunehmendes Problem.

MAP-Regularisierung

Da neuronale Netze sehr flexible Modelle mit enorm vielen Parametern sind, ist die Überanpassung ein sehr ernstzunehmendes Problem.

Ein naheliegender Ansatz ist natürlich die Verwendung einer Prior-Verteilung über die Parameter des neuronalen Netzes (**MAP-Methode**), wobei für diese Form der Regularisierung häufig unkorrelierte, in null zentrierte Gaussverteilungen

$$\mathcal{N}(\mathbf{W} | \mathbf{0}, \alpha_{\mathbf{W}}^{-1} I) \text{ und } \mathcal{N}(\mathbf{b} | \mathbf{0}, \alpha_{\mathbf{b}}^{-1} I)$$

für die Gewichtsmatrix und den Verschiebungsvektor verwendet werden.

Early stopping

Eine andere sehr verbreitete Methode ist das sogenannte **early stopping**, bei welchem im SGD-Algorithmus periodisch die Verallgemeinerungsqualität des aktuellen Netzes auf unabhängigen Testdaten mittels des gewählten Verlustmasses evaluiert wird und die Minimierung abgebrochen wird, sobald der Verlust auf den Testdaten anzusteigen beginnt.

Early stopping

Eine andere sehr verbreitete Methode ist das sogenannte **early stopping**, bei welchem im SGD-Algorithmus periodisch die Verallgemeinerungsqualität des aktuellen Netzes auf unabhängigen Testdaten mittels des gewählten Verlustmasses evaluiert wird und die Minimierung abgebrochen wird, sobald der Verlust auf den Testdaten anzusteigen beginnt.

Damit wird versucht, eine Überanpassung an die Trainingsdaten rechtzeitig zu verhindern.

Parameter-Sharing

Auch die Verwendung von Faltungsschichten anstelle von vollbesetzten Schichten (CNNs), was als eine Form von **Parameter-Sharing** interpretiert werden kann, hat sich in Bildverarbeitungsanwendungen als sehr lohnend erwiesen.

Parameter-Sharing

Auch die Verwendung von Faltungsschichten anstelle von vollbesetzten Schichten (CNNs), was als eine Form von **Parameter-Sharing** interpretiert werden kann, hat sich in Bildverarbeitungsanwendungen als sehr lohnend erwiesen.

Dieser Ansatz reduziert die Flexibilität gezielt unter Einbezug von struktureller Einsicht in die Problemstellung.

Parameter-Sharing

Auch die Verwendung von Faltungsschichten anstelle von vollbesetzten Schichten (CNNs), was als eine Form von **Parameter-Sharing** interpretiert werden kann, hat sich in Bildverarbeitungsanwendungen als sehr lohnend erwiesen.

Dieser Ansatz reduziert die Flexibilität gezielt unter Einbezug von struktureller Einsicht in die Problemstellung.

Natürlich werden noch viele andere Techniken eingesetzt, welche die inhärente Flexibilität von neuronalen Netzen zähmen sollen, um die Verallgemeinerungs-Qualität auf nicht gigantisch grossen Trainingsdaten zu verbessern [pml1, 13.5],[dl, 7].

Bayes-Methode

Die **Bayessche-Methode**, bei welcher die Posterior-Verteilung nicht mittels eines Punktmasses (unter Vernachlässigung aller verbleibender Unsicherheit in den Parametern) approximiert wird, sondern mit dieser Posterior-Verteilung die Parameter marginalisiert, hat sich in vielen Anwendungen als wesentlich robuster gegenüber der Überanpassung gezeigt.

Bayes-Methode

Die **Bayessche-Methode**, bei welcher die Posterior-Verteilung nicht mittels eines Punktmasses (unter Vernachlässigung aller verbleibender Unsicherheit in den Parametern) approximiert wird, sondern mit dieser Posterior-Verteilung die Parameter marginalisiert, hat sich in vielen Anwendungen als wesentlich robuster gegenüber der Überanpassung gezeigt.






Die Bayes-Methode ist im Kontext der neuronalen Netze aktuell noch nicht so stark verbreitet, aber das Potential ist nicht unerkannt. [bdI]

Vielen Dank für die Aufmerksamkeit.

References I

-  Bishop C.: Pattern Recognition and Machine Learning. Springer.
-  Durbin J., Koopman S. J.: Time Series Analysis by State Space Methods. Oxford University Press. 2nd edition.
-  Gareth J., Witten D., Hastie T., Tibshirani R.: Introduction to Statistical Learning. Springer.
-  Gelman A., Hill J.: Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press.
-  Gelman A., Carlin J., Stern H., Dunson D., Vehtari A., Rubin D.: Bayesian Data Analysis. CRC Press. 3rd edition.
-  Godfellow I., Bengio J., Courville A.: Deep Learning. MIT Press.
-  Hastie T., Tibshirani R., Friedman J.: The Elements of Statistical Learning. Springer. 2nd edition.
-  Kingma D., Lei Ba J.: Adam: A method for stochastic optimization.

References II

-  Koller, D., Friedman N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press.
-  Murphy K.P.: Machine Learning: a Probabilistic Perspective. MIT Press.
-  Murphy K.P.: Probabilistic Machine Learning. MIT Press.
-  Wilson A. G., Izmailov P.; Bayesian Deep Learning and a Probabilistic Perspective of Generalization.
-  https://keras.io/img/guides/functional_api/

Schichtmodell

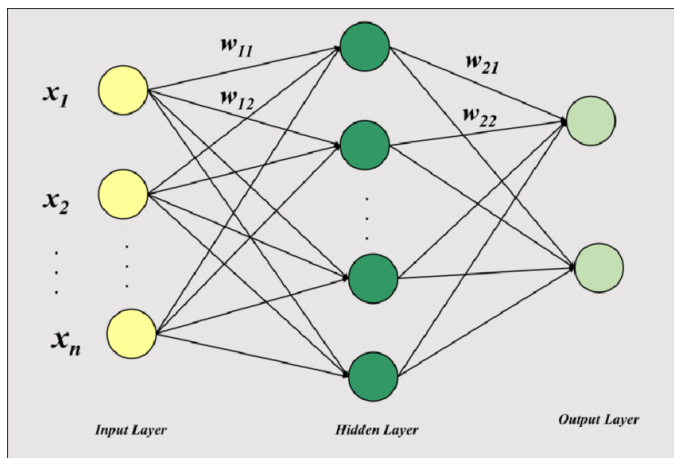


Abbildung: Eine verborgene Schicht mit $d_1 = n$ und $r_2 = 2$ mit schematischer Darstellung der Gewichte \mathbf{W}_1 und \mathbf{W}_2 ohne Darstellung der Verschiebungsvektoren \mathbf{b}_1 und \mathbf{b}_2

Regressionsmodell

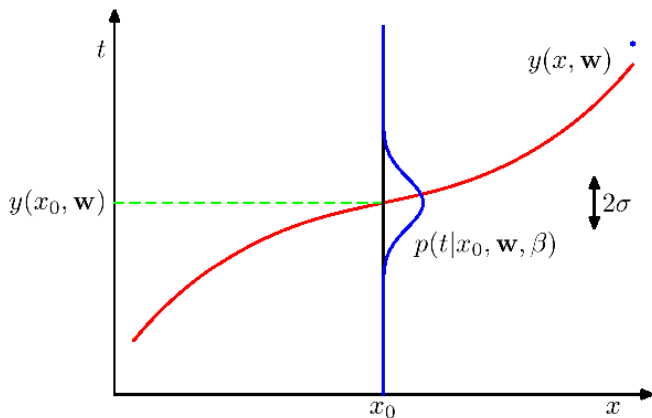


Abbildung: [prml, Figure 1.16] Lineares Regressionsmodell (Gausssche Ausgabeverteilung)

Logistische-Sigmoidfunktion

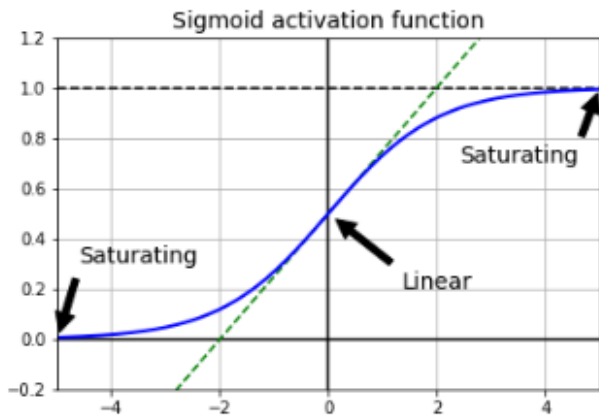


Abbildung: Logistische Sigmoid-Funktion [pml1, Figure 13.2]

Aktivierungsfunktionen

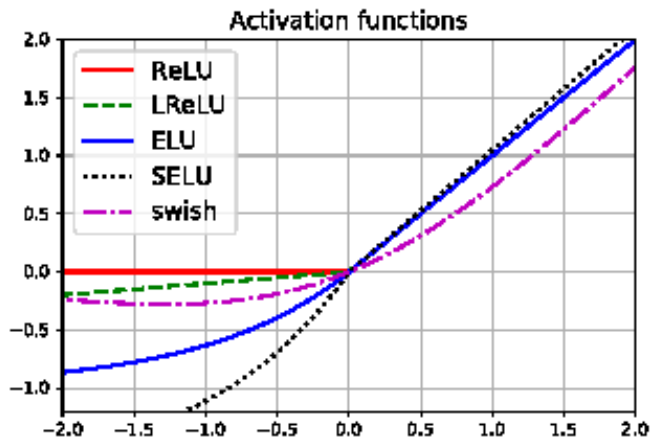


Abbildung: Aktivierungsfunktionen [pml1, Figure 13.2]

Zwei Köpfe

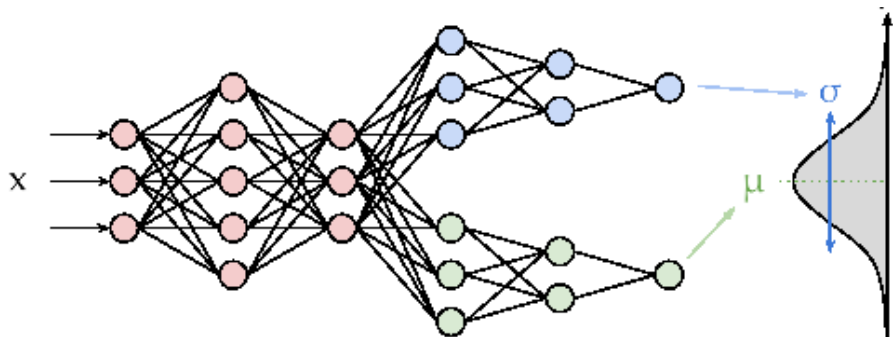


Abbildung: Neuronales Netz für heteroskedastische Regression.[pml1, Figure 13.7]

Schichtmodell

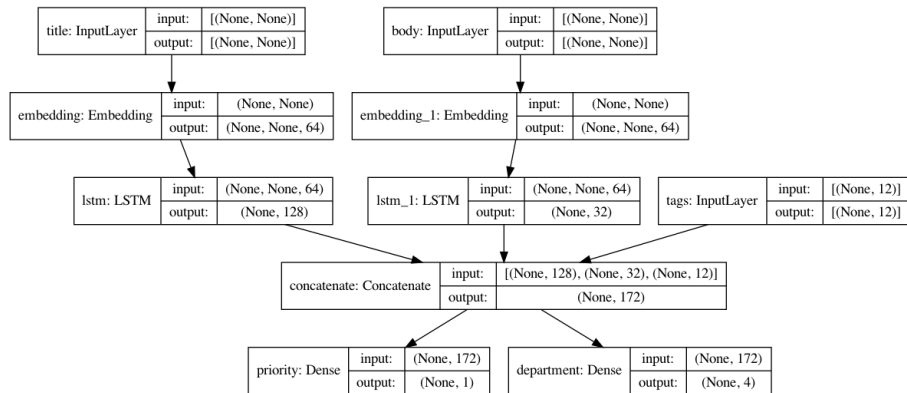


Abbildung: Neuronales Netz als gerichteter azyklischer Graph. [keras, functional_api_40_0.png]

Iris-Klassifikation

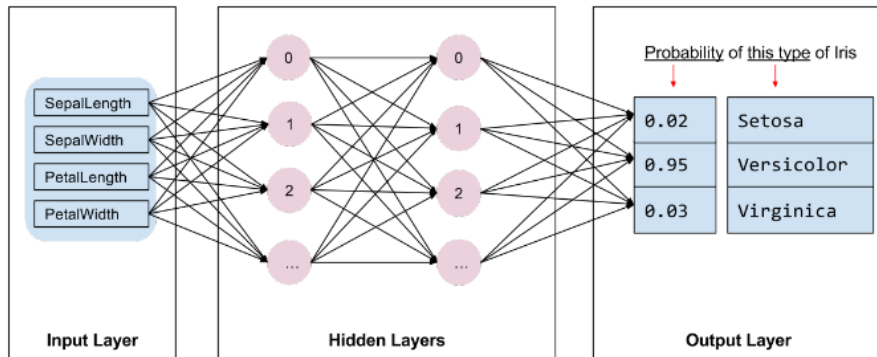


Abbildung: Klassifikationsmodell für Iris-Data[pml1, Figure 13_3]

Heteroskedastische Regression

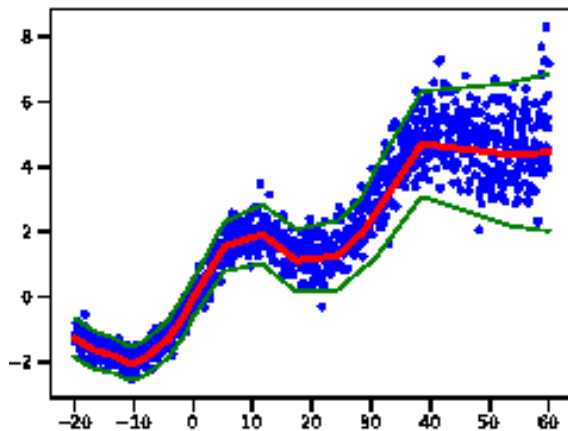


Abbildung: Daten für heteroskedastische Regression [pml1, Figure 13.8]