Wissensentdeckung in Datenbanken

Deep Learning (II)

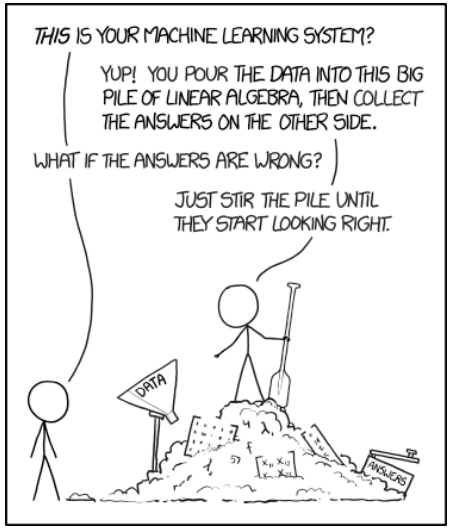
Nico Piatkowski und Uwe Ligges

Informatik—Künstliche Intelligenz
Computergestützte Statistik
Technische Universität Dortmund

25.07.2017

Überblick

- Faltungsnetze
- Dropout
- Autoencoder
- Generative Adversarial Networks



Neuronale Netze

- Eignen sich um beliebige Funktionen zu approximieren
- Können sowohl zur Regression als auch zur Klassifikation eingesetzt werden
- Werden mittels stochastischem Gradientenabstieg trainiert (Gradient wird in jeder Iteration nur auf einer Teilmenge der Daten berechnet)

Überblick

- Faltungsnetze
- Dropout
- Autoencoder
- Generative Adversarial Networks

Neuronale Netze

- Eignen sich um beliebige Funktionen zu approximieren
- Können sowohl zur Regression als auch zur Klassifikation eingesetzt werden
- Werden mittels stochastischem Gradientenabstieg trainiert (Gradient wird in jeder Iteration nur auf einer Teilmenge der Daten berechnet)

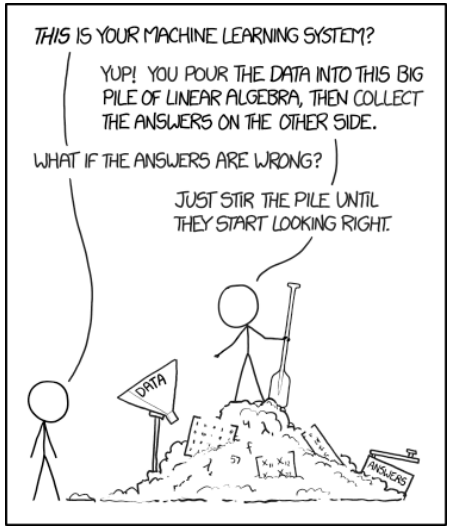
Überblick

- Faltungsnetze
- Dropout
- Autoencoder
- Generative Adversarial Networks

Neuronale Netze

- Eignen sich um beliebige Funktionen zu approximieren
- Können sowohl zur Regression als auch zur Klassifikation eingesetzt werden
- Werden mittels stochastischem Gradientenabstieg trainiert (Gradient wird in jeder Iteration nur auf einer Teilmenge der Daten berechnet)

Überblick



- Faltungsnetze
- Dropout
- Autoencoder
- Generative Adversarial Networks

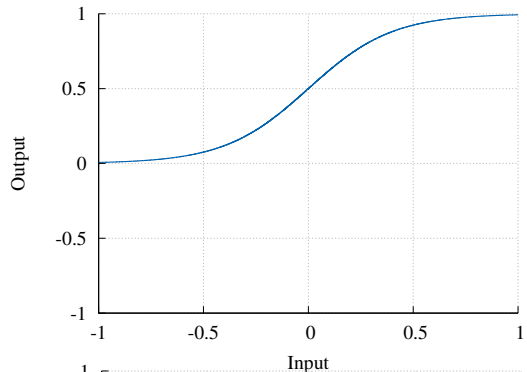
Neuronale Netze

- Eignen sich um beliebige Funktionen zu approximieren
- Können sowohl zur Regression als auch zur Klassifikation eingesetzt werden
- Werden mittels stochastischem Gradientenabstieg trainiert (Gradient wird in jeder Iteration nur auf einer Teilmenge der Daten berechnet)

Aktivierungsfunktionen an Neuron $v \in V$

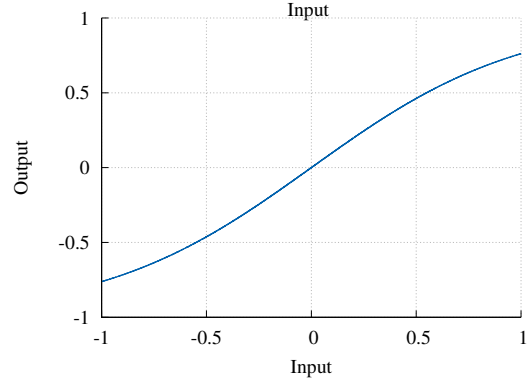
Sigmoid:

$$a_v^{\text{sig}}(z) = \frac{1}{1 + \exp(-z)}$$



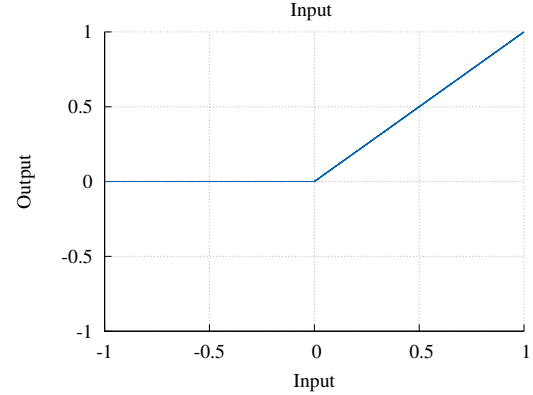
Tangens Hyperbolicus (tanh):

$$a_v^{\text{tanh}}(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



Rectified linear unit (ReLU):

$$a_v^{\text{ReLU}}(z) = \max\{0, z\}$$

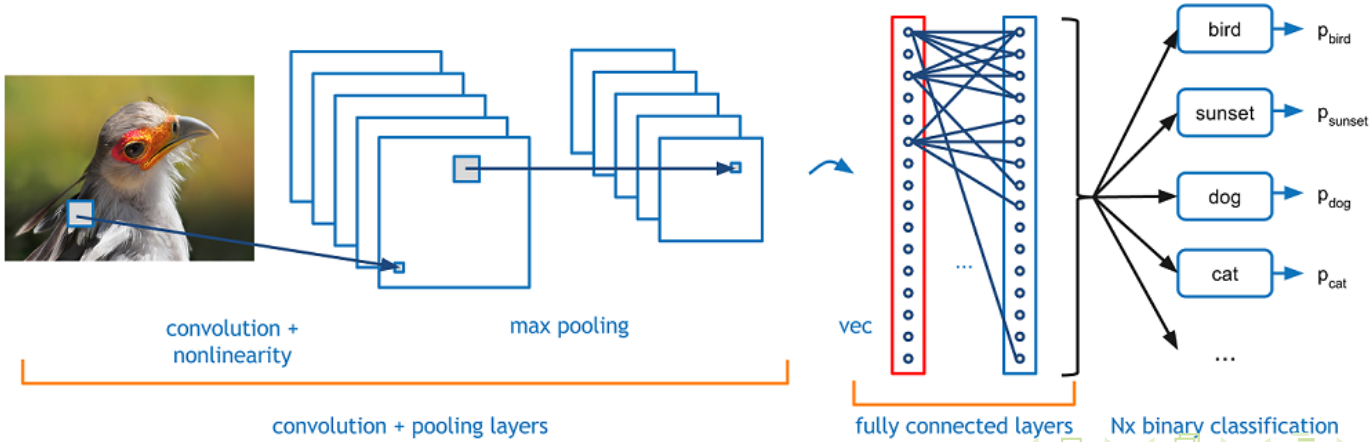


Faltungsnetze und Translationsinvarianz

Beobachtung: Sollen Objekte in Bildern erkannt werden, müssen klassische Netze lernen dass Objekt überall zu erkennen ⇒ künstliche Verkomplizierung des Problems!

Besser: Ausnutzung “lokaler rezeptiver Felder”
Lerne kleine Schablonen die über das Bild geschoben werden

Beispiel: Original Bild: 1024x768. Jedes Neuron auf Schicht 1 ist nur mit 10x10 Pixeln verbunden, und alle Neuronen auf Schicht 1 verwenden **den gleichen Gewichtsvektor**!

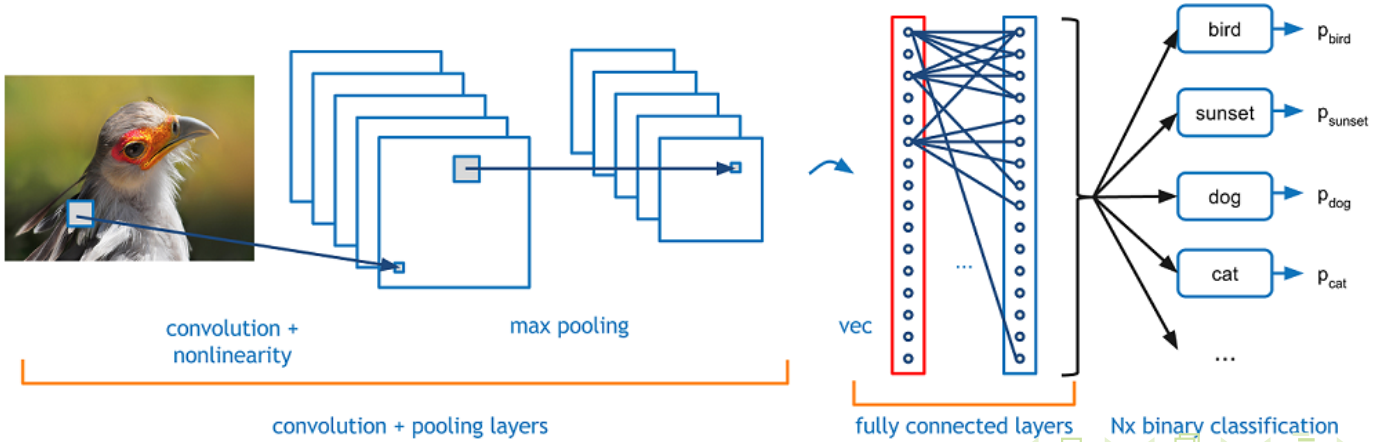


Faltungsnetze und Translationsinvarianz

Beobachtung: Sollen Objekte in Bildern erkannt werden, müssen klassische Netze lernen dass Objekt überall zu erkennen $\Rightarrow$ künstliche Verkomplizierung des Problems!

Besser: Ausnutzung “lokaler rezeptiver Felder”
Lerne kleine Schablonen die über das Bild geschoben werden

Beispiel: Original Bild: 1024x768. Jedes Neuron auf Schicht 1 ist nur mit 10x10 Pixeln verbunden, und alle Neuronen auf Schicht 1 verwenden **den gleichen Gewichtsvektor**!

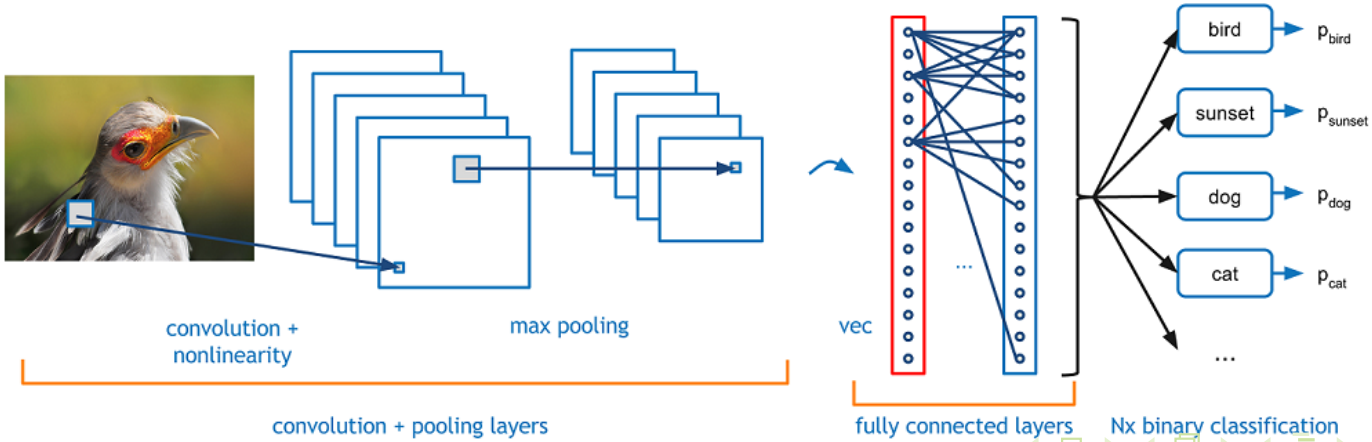


Faltungsnetze und Translationsinvarianz

Beobachtung: Sollen Objekte in Bildern erkannt werden, müssen klassische Netze lernen dass Objekt überall zu erkennen $\Rightarrow$ künstliche Verkomplizierung des Problems!

Besser: Ausnutzung “lokaler rezeptiver Felder”
Lerne kleine Schablonen die über das Bild geschoben werden

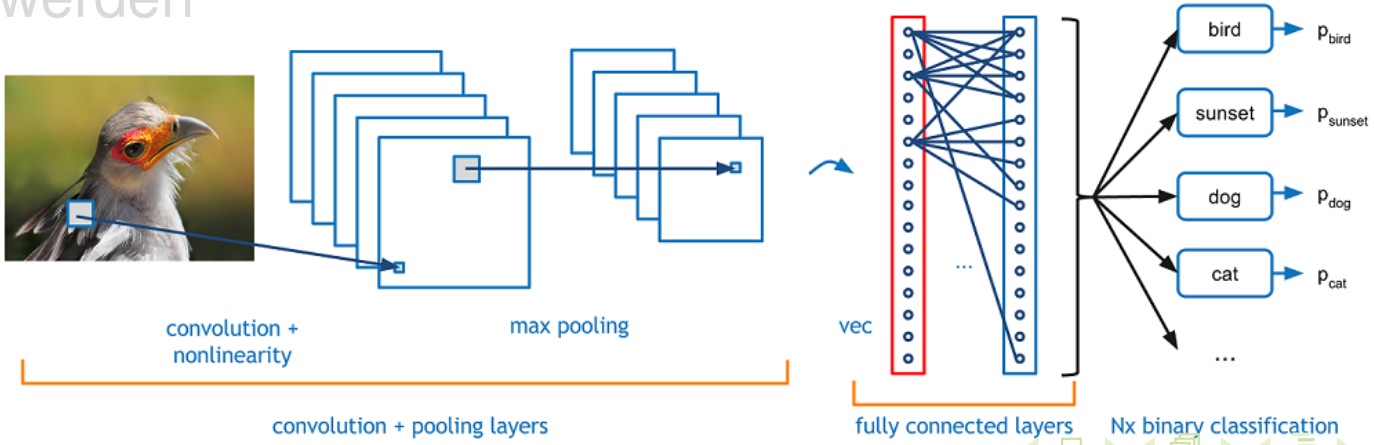
Beispiel: Original Bild: 1024x768. Jedes Neuron auf Schicht 1 ist nur mit 10x10 Pixeln verbunden, und alle Neuronen auf Schicht 1 verwenden **den gleichen Gewichtsvektor**!



Faltungsnetze und Translationsinvarianz (II)

Verallgemeinerung (Faltung):

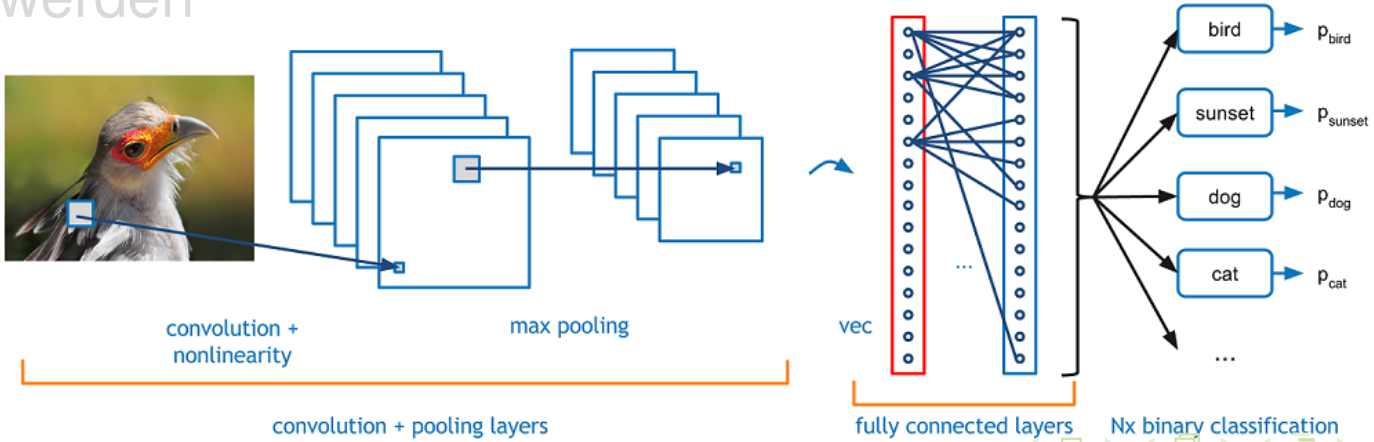
- Es gibt m Klassen von Neuronen—alle Neuronen einer Klasse c verwenden den gleichen Gewichtsvektor β_c
- Jedes Neuron jeder Klasse (=Filter) ist mit einem anderen $h \times h$ Ausschnitt des Bildes verbunden
- **Oft**: Es wird nur das Maximum einer Menge von Neuronen an die nächste Schicht weitergeleitet (Max-Pooling)
- Es können mehrere Faltungen hintereinander ausgeführt werden



Faltungsnetze und Translationsinvarianz (II)

Verallgemeinerung (Faltung):

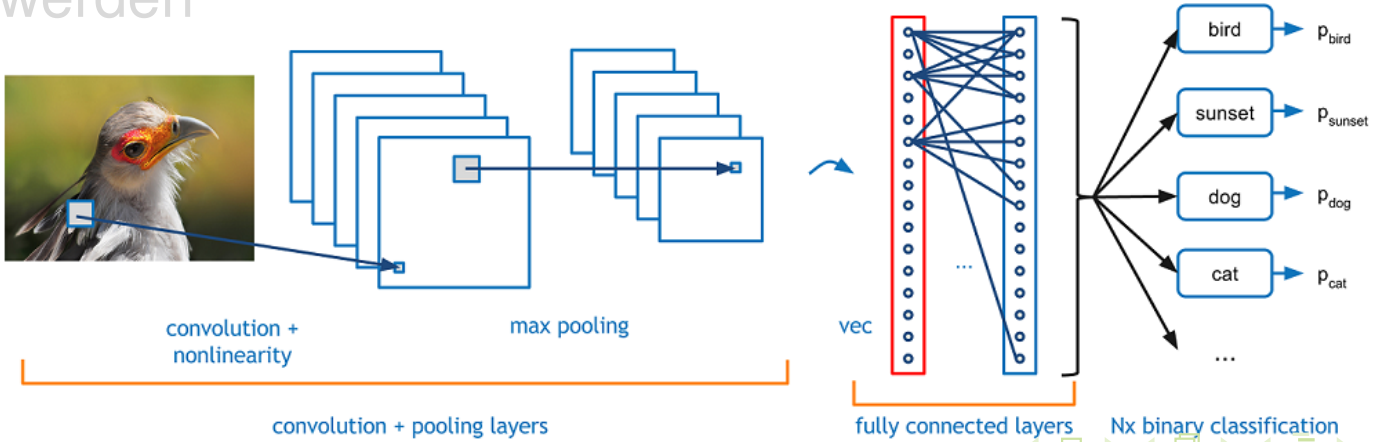
- Es gibt m Klassen von Neuronen—alle Neuronen einer Klasse c verwenden den gleichen Gewichtsvektor β_c
- Jedes Neuron jeder Klasse (=Filter) ist mit einem anderen $h \times h$ Ausschnitt des Bildes verbunden
- **Oft**: Es wird nur das Maximum einer Menge von Neuronen an die nächste Schicht weitergeleitet (Max-Pooling)
- Es können mehrere Faltungen hintereinander ausgeführt werden



Faltungsnetze und Translationsinvarianz (II)

Verallgemeinerung (Faltung):

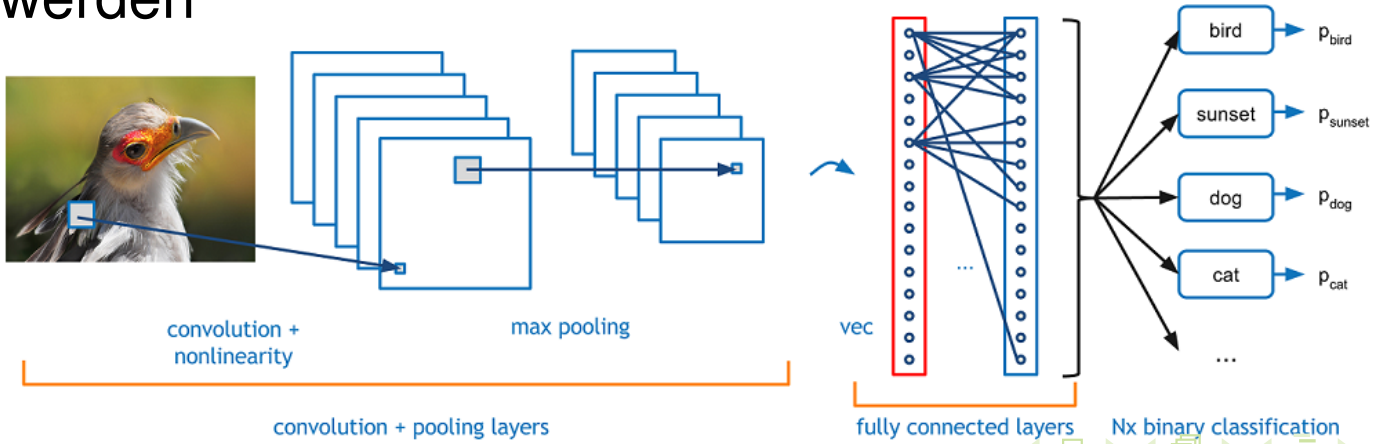
- Es gibt m Klassen von Neuronen—alle Neuronen einer Klasse c verwenden den gleichen Gewichtsvektor β_c
- Jedes Neuron jeder Klasse (=Filter) ist mit einem anderen $h \times h$ Ausschnitt des Bildes verbunden
- **Oft**: Es wird nur das Maximum einer Menge von Neuronen an die nächste Schicht weitergeleitet (Max-Pooling)
- Es können mehrere Faltungen hintereinander ausgeführt werden



Faltungsnetze und Translationsinvarianz (II)

Verallgemeinerung (Faltung):

- Es gibt m Klassen von Neuronen—alle Neuronen einer Klasse c verwenden den gleichen Gewichtsvektor β_c
- Jedes Neuron jeder Klasse (=Filter) ist mit einem anderen $h \times h$ Ausschnitt des Bildes verbunden
- **Oft**: Es wird nur das Maximum einer Menge von Neuronen an die nächste Schicht weitergeleitet (Max-Pooling)
- Es können mehrere Faltungen hintereinander ausgeführt werden



Überanpassung

Jedes Neuron auf Schicht k hat $|\{v \mid L(v) = k-1\}|$ Parameter (weniger bei Faltungsnetzen)

Aufgrund hoher Anzahl an Parametern: Neuronale Netze neigen zur Überanpassung!

- Netz lernt Daten “auswendig”
- Vorhersagefehler ist auf Trainingsdaten viel kleiner als aus Testdaten

Strategien zur Vermeidung von Überanpassung:

- Norm-basierte Regularisierung $\ell(\beta; \mathcal{D}) + \lambda \|\beta\|_q$ (z.B. $q = 1$ oder $q = 2$)
- Dropout
- “Pretraining” mittels Autoencoder

Überanpassung

Jedes Neuron auf Schicht k hat $|\{v \mid L(v) = k-1\}|$ Parameter (weniger bei Faltungsnetzen)

Aufgrund hoher Anzahl an Parametern: Neuronale Netze neigen zur Überanpassung!

- Netz lernt Daten “auswendig”
- Vorhersagefehler ist auf Trainingsdaten viel kleiner als aus Testdaten

Strategien zur Vermeidung von Überanpassung:

- Norm-basierte Regularisierung $\ell(\beta; \mathcal{D}) + \lambda \|\beta\|_q$ (z.B. $q = 1$ oder $q = 2$)
- Dropout
- “Pretraining” mittels Autoencoder

Überanpassung

Jedes Neuron auf Schicht k hat $|\{v \mid L(v) = k-1\}|$ Parameter (weniger bei Faltungsnetzen)

Aufgrund hoher Anzahl an Parametern: Neuronale Netze neigen zur Überanpassung!

- Netz lernt Daten “auswendig”
- Vorhersagefehler ist auf Trainingsdaten viel kleiner als aus Testdaten

Strategien zur Vermeidung von Überanpassung:

- Norm-basierte Regularisierung $\ell(\boldsymbol{\beta}; \mathcal{D}) + \lambda \|\boldsymbol{\beta}\|_q$ (z.B. $q = 1$ oder $q = 2$)
- Dropout
- “Pretraining” mittels Autoencoder

Dropout

Neuron $v \in V$ mit Parametervektor $\boldsymbol{\beta}_v$

Idee: Parameterupdate $\boldsymbol{\beta}_v^{t+1} = \boldsymbol{\beta}_v^{t+1} + \eta \nabla_v \ell(\boldsymbol{\beta}; \mathcal{D})$ wird zufallsbasiert ausgeführt, z.B.

$$p = \mathbb{P}(\text{Parameterupdate and Neuron } v) = \frac{1}{2}$$

Man kann zeigen: Dropout ist equivalent zu einem Ensemble von Netzen

Wichtig: Bei der Anwendung des Models muss Dropout berücksichtigt werden! Zwei Möglichkeiten:

- Skalierung aller Gewichte jedes Neurons mit p
- Berechnung der erwarteten Netzausgabe (d.h. Netz mehrfach auswerten mit Dropout!) ← “richtig”, aber teuer

Dropout

Neuron $v \in V$ mit Parametervektor $\boldsymbol{\beta}_v$

Idee: Parameterupdate $\boldsymbol{\beta}_v^{t+1} = \boldsymbol{\beta}_v^{t+1} + \eta \nabla_v \ell(\boldsymbol{\beta}; \mathcal{D})$ wird zufallsbasiert ausgeführt, z.B.

$$p = \mathbb{P}(\text{Parameterupdate and Neuron } v) = \frac{1}{2}$$

Man kann zeigen: Dropout ist equivalent zu einem Ensemble von Netzen

Wichtig: Bei der Anwendung des Models muss Dropout berücksichtigt werden! Zwei Möglichkeiten:

- Skalierung aller Gewichte jedes Neurons mit p
- Berechnung der erwarteten Netzausgabe (d.h. Netz mehrfach auswerten mit Dropout!) ← “richtig”, aber teuer

Dropout

Neuron $v \in V$ mit Parametervektor $\boldsymbol{\beta}_v$

Idee: Parameterupdate $\boldsymbol{\beta}_v^{t+1} = \boldsymbol{\beta}_v^{t+1} + \eta \nabla_v \ell(\boldsymbol{\beta}; \mathcal{D})$ wird zufallsbasiert ausgeführt, z.B.

$$p = \mathbb{P}(\text{Parameterupdate and Neuron } v) = \frac{1}{2}$$

Man kann zeigen: Dropout ist equivalent zu einem Ensemble von Netzen

Wichtig: Bei der Anwendung des Models muss Dropout berücksichtigt werden! Zwei Möglichkeiten:

- Skalierung aller Gewichte jedes Neurons mit p
- Berechnung der erwarteten Netzausgabe (d.h. Netz mehrfach auswerten mit Dropout!) ← “richtig”, aber teuer

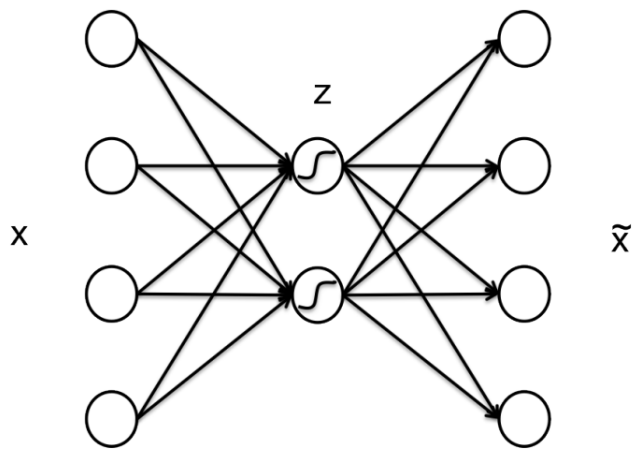
Autoencoder

Allgemeine Dimensionsreduktion: **Gegeben**: d-dimensionaler Datenpunkt $\boldsymbol{x}$
Gesucht: Funktionen $f : \mathbb{R}^d \to \mathbb{R}^h$ und $g : \mathbb{R}^h \to \mathbb{R}^d$ mit $h < d$ und

$$g(f(\boldsymbol{x})) \approx \boldsymbol{x}$$

Erinnerung: Tiefes Neuronales Netz ist Folge von Funktionskompositionen: $f^{\cdots}(f^2(f^1(f^0(\boldsymbol{x}))))$

$\Rightarrow$ Nutze tiefes Netz um f und g zu lernen! (=Autoencoder)



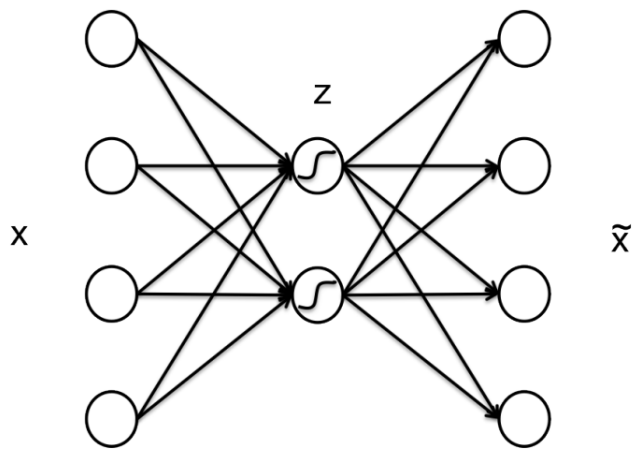
Autoencoder

Allgemeine Dimensionsreduktion: **Gegeben**: d-dimensionaler Datenpunkt $\boldsymbol{x}$
Gesucht: Funktionen $f : \mathbb{R}^d \to \mathbb{R}^h$ und $g : \mathbb{R}^h \to \mathbb{R}^d$ mit $h < d$ und

$$g(f(\boldsymbol{x})) \approx \boldsymbol{x}$$

Erinnerung: Tiefes Neuronales Netz ist Folge von Funktionskompositionen: $f^{\cdots}(f^2(f^1(f^0(\boldsymbol{x}))))$

$\Rightarrow$ Nutze tiefes Netz um f und g zu lernen! (=Autoencoder)



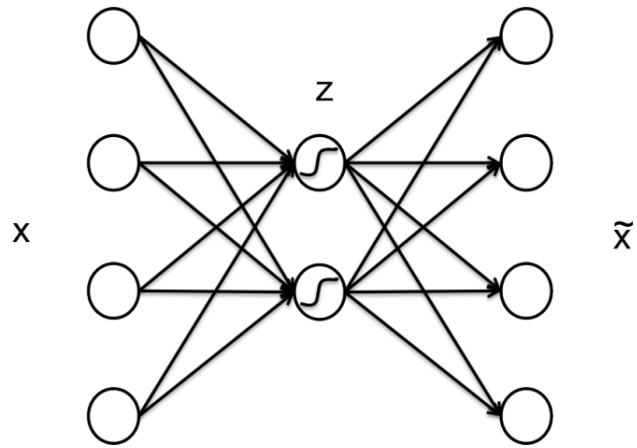
Autoencoder

Allgemeine Dimensionsreduktion: **Gegeben**: d-dimensionaler Datenpunkt $\boldsymbol{x}$
Gesucht: Funktionen $f : \mathbb{R}^d \to \mathbb{R}^h$ und $g : \mathbb{R}^h \to \mathbb{R}^d$ mit $h < d$ und

$$g(f(\boldsymbol{x})) \approx \boldsymbol{x}$$

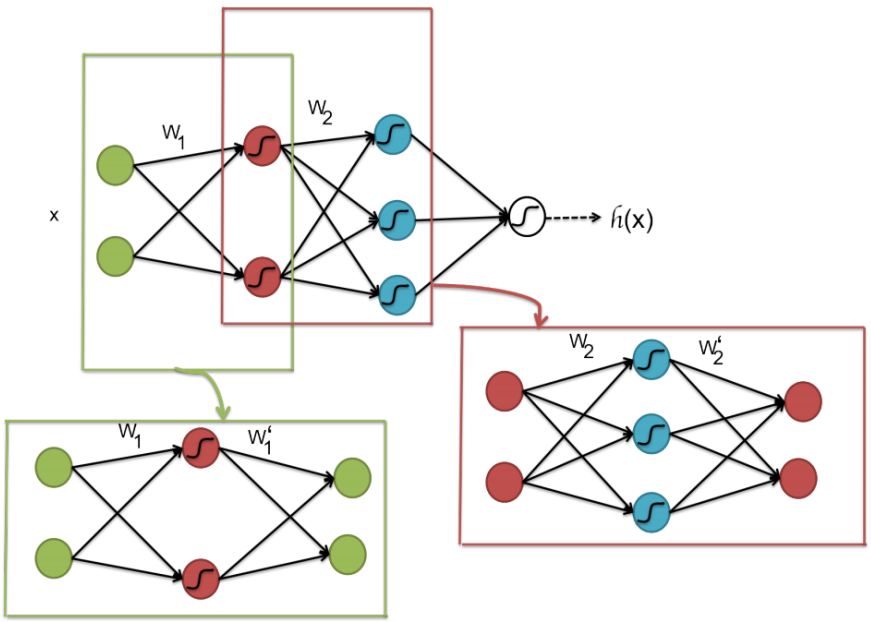
Erinnerung: Tiefes Neuronales Netz ist Folge von Funktionskompositionen: $f^{\cdots}(f^2(f^1(f^0(\boldsymbol{x}))))$

$\Rightarrow$ Nutze tiefes Netz um f und g zu lernen! (=Autoencoder)



Pretraining/Stacked Autoencoder

- Mehrere autoencoder können hintereinander “geschaltet” werden
- Jede Schicht kann einzeln trainiert werden
- Ermöglicht **unüberwachtes** Lernen von high-Level Merkmalen
- Kontruktion von Klassifikator durch anfügen weiterer Schichten und Fine-Tuning der Netzparameter



Datengenerierung mit Neuronalen Netzen

Idee: Nutze gelerntes Netz um Daten zu erzeugen!

Vorgehensweise (ImageNet Klassifikation):

- Trainiere Neuronales Netz auf ImageNet Datensatz
- Lege Zufallsvektor als Eingabe an
- Wähle eine beliebige Klasse (z.B. Blumentopf) und ändere Zufallsvektor, so dass die Aktivierung des entsprechenden Ausgabeneurons erhöht wird (z.B. durch Umformulierung als Maximierungsproblem)
- Wiederhole den vorherigen Schritt so oft wie möglich

Auch mit Autoencodern (unüberwacht) möglich!

Datengenerierung mit Neuronalen Netzen

Idee: Nutze gelerntes Netz um Daten zu erzeugen!

Vorgehensweise (ImageNet Klassifikation):

- Trainiere Neuronales Netz auf ImageNet Datensatz
- Lege Zufallsvektor als Eingabe an
- Wähle eine beliebige Klasse (z.B. Blumentopf) und ändere Zufallsvektor, so dass die Aktivierung des entsprechenden Ausgabeneurons erhöht wird (z.B. durch Umformulierung als Maximierungsproblem)
- Wiederhole den vorherigen Schritt so oft wie möglich

Auch mit Autoencodern (unüberwacht) möglich!

Datengenerierung mit Neuronalen Netzen

Idee: Nutze gelerntes Netz um Daten zu erzeugen!

Vorgehensweise (ImageNet Klassifikation):

- Trainiere Neuronales Netz auf ImageNet Datensatz
- Lege Zufallsvektor als Eingabe an
- Wähle eine beliebige Klasse (z.B. Blumentopf) und ändere Zufallsvektor, so dass die Aktivierung des entsprechenden Ausgabeneurons erhöht wird (z.B. durch Umformulierung als Maximierungsproblem)
- Wiederhole den vorherigen Schritt so oft wie möglich

Auch mit Autoencodern (unüberwacht) möglich!

Datengenerierung mit Neuronalen Netzen

Idee: Nutze gelerntes Netz um Daten zu erzeugen!

Vorgehensweise (ImageNet Klassifikation):

- Trainiere Neuronales Netz auf ImageNet Datensatz
- Lege Zufallsvektor als Eingabe an
- Wähle eine beliebige Klasse (z.B. Blumentopf) und ändere Zufallsvektor, so dass die Aktivierung des entsprechenden Ausgabeneurons erhöht wird (z.B. durch Umformulierung als Maximierungsproblem)
- Wiederhole den vorherigen Schritt so oft wie möglich

Auch mit Autoencodern (unüberwacht) möglich!

Label: Blumentopf

Label: Huhn

Generative Adversarial Networks

- Erzeugte Daten sehen “künstlich” aus
- **Aber**: Daten sollen möglichst echt aussehen?!

Idee: Trainiere zwei Netze!

- Erstes Netz versucht Daten zu erzeugen (z.B. Bilder von Hühnern)
- Zweites Netz versucht “echte” Bilder von künstlichen Bildern zu unterscheiden

GAN Verlustfunktion:

$$\ell(\boldsymbol{\beta}^D, \boldsymbol{\beta}^G; \mathcal{D}) = -\log \mathbb{P}_{\boldsymbol{\beta}^D}(\mathcal{D}) - \mathbb{E}_Z \log(1 - \mathbb{P}_{\boldsymbol{\beta}^D}(G(Z)))$$

Superresolution GAN