Neuronale Netzwerke

Feedfoward-Netzwerke (1/2)

Die **Architektur** $\mathcal{A} = (V, E, f)$ eines Feedforward-Netzwerks besteht aus

($*$) einem *azyklischen*, *gerichteten* und *geschichteten* Graphen $G = (V, E)$, wobei V in die Schichten $V_0, \ldots, V_T$ unterteilt ist

1. mit $V_t = \{ (t, j) : 1 \leqslant j \leqslant |V_t| \}$.
2. Alle Kanten (v, w) verlaufen von einem Knoten $v \in V_t$ zu einem Knoten $w \in V_{t+1}$ in der darauffolgenden „Nachbarschicht“.
   - ★ V_0 besteht nur aus Quellen und V_T nur aus Senken.

($*$) einer Zuweisung $f(v) = \gamma_v$ von „**Aktivierungsfunktionen**“

$$\gamma_v : \mathbb{R} \to \mathbb{R}$$

zu Knoten $v \in V$.

($*$) Nicht gezeigt werden Kanten, die einen Knoten mit einer Eingabe versorgen. Diese Kanten sind ebenfalls gewichtet.

Feedforward-Netzwerke (2/2)

Ein **Feedforward-Netzwerk** $\mathcal{N} = (\mathcal{A}, w)$ besteht aus

(a) der Architektur $\mathcal{A}$

(b) sowie einer Zuweisung

$$w : E \to \mathbb{R}$$

von **Gewichten** $w(u, v)$ zu Kanten $(u, v) \in E$.

Wir können *implizite* mit Schwellenwerten arbeiten:

- Verbinde eine neue Quelle $u = (t, 0)$ mit allen Knoten $v \in V_t$.
- Weise der Kante (u, v) den Schwellenwert von v zu.

Wie rechnet ein Feedforward-Netz?

1. Es ist $n := |V_0|$ und $m := |V_T|$. Die einzelnen Komponenten der Eingabe x werden an die Quellen $(0, i) \in V_0$ angelegt: Setze

$$\text{aus}_{0,i}(x) := x_i.$$

2. Für alle Schichten $t = 1, \ldots, T$ in *aufsteigender* Reihenfolge und für alle Knoten $(t, j) \in V_t$:
   1. Setze

$$\text{summe}_{t,j}(x) := \sum_{k:e=((t-1,k),(t,j))\in E} w(e) \cdot \text{aus}_{t-1,k}(x)$$

   2. und anschließend

$$\text{aus}_{t,j}(x) := \gamma_{t,j}\Big(\text{summe}_{t,j}(x)\Big).$$

3. Das Feedforward-Netzwerk berechnet die Funktion

$$h^w_{\mathcal{A}} : \mathbb{R}^n \to \mathbb{R}^m \quad \text{mit} \quad h^w_{\mathcal{A}}(x) := (\text{aus}_{T,1}(x), \ldots, \text{aus}_{T,m}(x)).$$

Architekturen und ihre Hypothesenklasse

Sei $\mathcal{A}$ eine Architektur mit n Quellen, m Senken. Die

Hypothesenklasse $\mathcal{H}_{\mathcal{A}}$

von $\mathcal{A}$ besteht aus allen Funktionen

$$h_{\mathcal{A}}^{w} : \mathbb{R}^n \to \mathbb{R}^m$$

für Gewichtsvektoren $w \in \mathbb{R}^n$.

Eine Zielverteilung p ist zu lernen. Die *typische Lernsituation*:

1. Eine Architektur $\mathcal{A}$ ist vorgegeben.
   - Wissen über p ist in die Konstruktion von $\mathcal{A}$ eingeflossen.
2. Lerne Gewichte:
   - Approximiere p möglichst gut mit einer Hypothese $h \in \mathcal{H}_{\mathcal{A}}$.

Rekurrente Netzwerke (RNNs)

Die Architektur $\mathcal{A}_r = (V, E, f)$ eines rekurrenten Netzwerks (RNN) besteht aus

- $(*)$ einem gerichteten Graphen $G = (V, E)$, der möglicherweise Kreise besitzt und
- $(*)$ einer Zuweisung $f(v) = \gamma_v$ von Aktivierungsfunktionen γ_v zu Knoten $v \in V$.

Ein **rekurrentes neuronales Netzwerk (RNN)** $\mathcal{R} = (\mathcal{A}_r, w)$ besteht aus

(a) der Architektur $\mathcal{A}_r$ und

(b) einer Zuweisung $w : E \to \mathbb{R}$ von Gewichten zu Kanten.

Aktivierungsfunktionen

Aktivierungsfunktionen (1/3)

(a) Die **(binäre) Threshold-Funktion**

$$\gamma(x) := \begin{cases} 1 & x \geqslant 0 \\ 0 & \text{sonst} \end{cases}$$

wurde 1943 von McCulloch und Pitts vorgeschlagen.

- Das erste *Integrate-and-Fire* Modell: Die Ausgaben der Nachbarneuronen u mit $(u, v) \in E$ bestimmen, ob v feuert.
- Hemmende Impulse, bzw. verstärkende Impulse werden gewichtet summiert und mit dem Schwellenwert des Neurons verglichen.

Weitere Modelle in der Computational Neuroscience: *Spikende Neuronen*, *Feuerraten*, ...

(b) Das **Standard-Sigmoid**

$$\sigma(x) := \frac{1}{1 + \exp^{-x}}$$

oder der **hyperbolische Tangens** $\tanh(x) := \frac{\exp^{2x} - 1}{\exp^{2x} + 1}$
(= $2\sigma(2x) - 1$) sind diffbare „Varianten“ der Thresholdfunktion.

(c) Das **ReLU**-Gatter (oder „rectified linear unit“)

$$r(x) := \max\{0, x\}$$

Goodfellow et al: Stückweise lineare Gatterfunktionen gehören zu den wichtigsten algorithmischen Neuerungen.

(d) Die **Softplus**-Funktion

$$s(x) := \ln(1 + \exp^x)$$

kann als eine differenzierbare Variante des Rectifiers angesehen werden. (Beachte, dass $s' = \sigma$.)

- Experimentell: Schlechter als ReLU-Gatter!

(e) Für $c \in \mathbb{R}$ und Vektoren $x, W \in \mathbb{R}^m$ die **Radialbasis**-Funktion

$$r_{c,W}(x) := \exp^{-\frac{1}{c^2}\|W-x\|^2}.$$

x ist der *Vektor* der Eingaben des Gatters.

(f) Wie die Radialbasisfunktion sind auch **Softmax**-Gatter

$$\gamma_i(x_1, \ldots, x_k) := \frac{\exp^{x_i}}{\sum_{j=1}^{k} \exp^{x_j}}$$

keine konventionellen Aktivierungsfunktionen.

- Sie werden häufig als Ausgabegatter eingesetzt und
- produzieren eine Verteilung auf den k Eingabewerten des Gatters.
  - ★ $\gamma_i(x_1, \ldots, x_k) \approx$ Überzeugung hinter Option i.

Neuronale Netzwerke: Das Vorbild

Das zentrale Nervensystem

+ besteht aus $10^{10} - 10^{12}$ Nervenzellen mit insgesamt ca. 10^{12} Schaltvorgängen pro Sekunde
  - Moderne Parallelrechner: Zwischen $10^{11} - 10^{12}$ Schaltvorgängen pro Sekunde
+ mit durchschnittlich 10.000 Verbindungen pro Nervenzelle
  - Moderne Parallelrechner mit Verbindungszahl im kleinen zweistelligen Bereich.
- mit einer „**erbärmlichen**“ Schaltzeit im Millisekundenbereich
  - Moderne Rechner: Im Bereich einer Nanosekunde
+ und benötigt die **Energie einer Glühbirne**.
  - Moderne Supercomputer verbrauchen mehrere Megawatt!

Ein Muß: Löse algorithmische Probleme in **wenigen** Schritten!

Berechnungskraft

Was können Threshold-Funktionen?

Betrachte Threshold-Funktionen über $X = \{0,1\}^n$.

$$\neg x \iff 1 - x \geqslant \frac{1}{2}$$

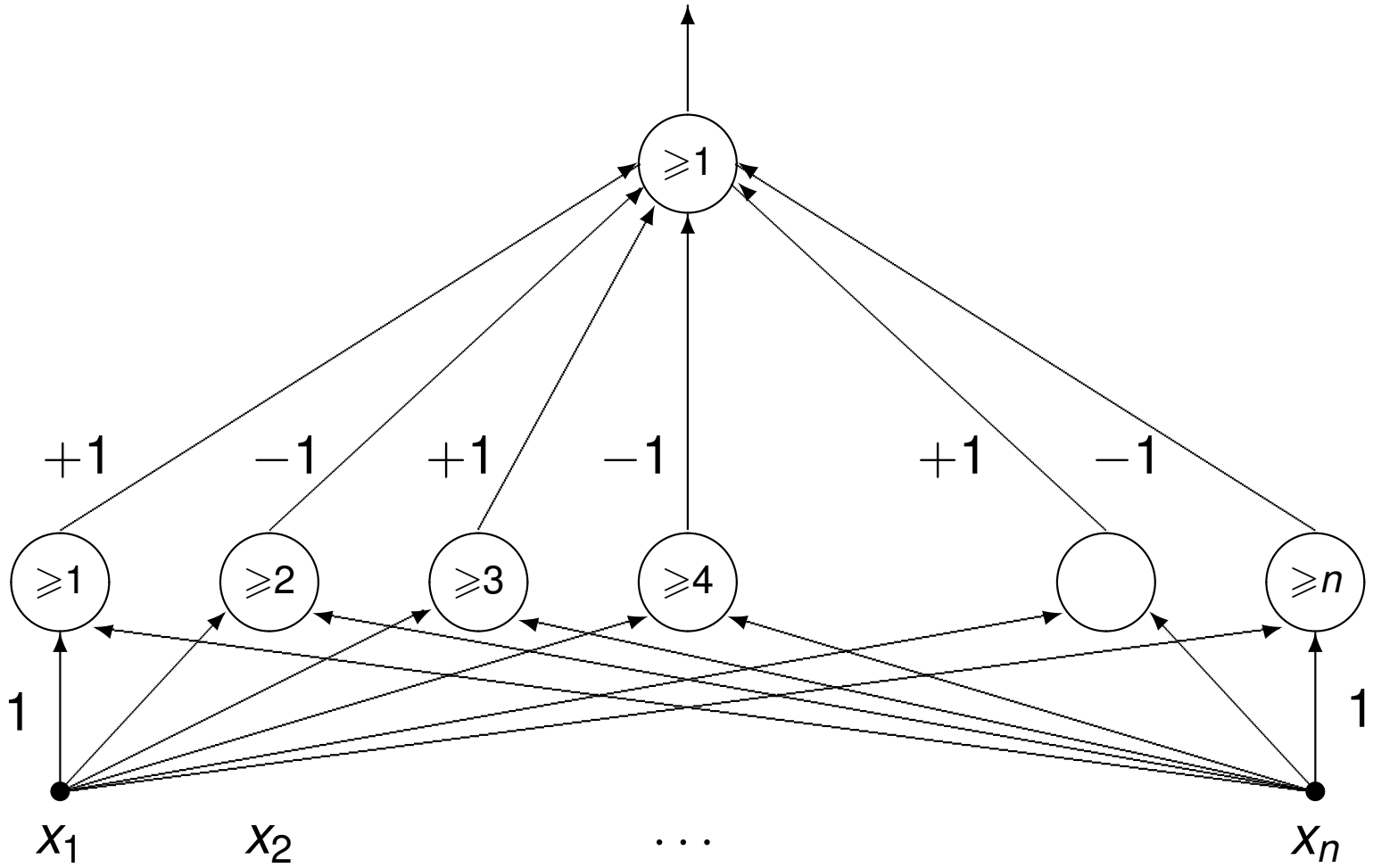
$$x_1 \wedge x_2 \wedge \cdots \wedge x_n \iff \sum_{i=1}^{n} x_i \geqslant n - \frac{1}{2}$$

$$x_1 \vee x_2 \vee \cdots \vee x_n \iff \sum_{i=1}^{n} x_i \geqslant \frac{1}{2}$$

$$\mathrm{Zahl}(x_{n-1}, \ldots, x_0) \geqslant \mathrm{Zahl}(y_{n-1}, \ldots, y_0) \iff \sum_{i=0}^{n-1} x_i \cdot 2^i \geqslant \sum_{i=0}^{n-1} y_i \cdot 2^i$$

Und die Parität $x_1 \oplus \cdots \oplus x_n$?

Die Parität $x_1 \oplus \cdots \oplus x_n$



$n+1$ Knoten und Tiefe zwei genügen $\Longrightarrow$
$\{\neg, \wedge, \vee\}$-Schaltkreise der Tiefe t benötigen $2^{\Omega(n^{1/t})}$ Knoten.

Neuronale Netzwerke und Turingmaschinen

Die Funktion $F : \{0,1\}^* \to \{0,1\}$ werde von einer Turingmaschine mit einem Lese-Schreib-Band in Zeit $T(n)$ berechnet $\Longrightarrow$ Es gibt einen $\{\wedge, \vee, \neg\}$-Schaltkreis der Tiefe $\mathcal{O}(T(N))$ und Größe $\mathcal{O}(T^2)$ für F.

Die Berechnungskraft von Sigmoid-Schaltkreisen

- γ sei 2-fach stetig differenzierbar und es gelte $\gamma''(0) \neq 0$.
- Wir konstruieren einen γ-Schaltkreis der Tiefe 2, der eine Eingabe $x \in [-1, 1]$ approximativ quadriert.

(1) Das quadratische Taylorpolynom von γ ist $\gamma(x) = \gamma(0) + x\gamma'(0) + \frac{x^2}{2}\gamma''(0) + r(x)$ mit $|r(x)| = O(|x^3|)$.

(2) Setze $\gamma^*(x) = \big(\gamma(x) - x\gamma'(0) - \gamma(0)\big) \cdot \frac{2}{\gamma''(0)}$.

- Es ist $\gamma^*(x) = x^2 + \frac{2r(x)}{\gamma''(0)}$.

(3) Berechne $L^2 \cdot \gamma^*(\frac{x}{L})$:

- Es ist $L^2 \cdot \gamma^*(\frac{x}{L}) = x^2 + \frac{2r(\frac{x}{L})L^2}{\gamma''(0)}$.
- Der Approximationsfehler ist höchstens $\frac{2r(\frac{x}{L})L^2}{\gamma''(0)} = O(\frac{x^3}{L})$, denn $| r(\frac{x}{L}) | = O(| \frac{x^3}{L^3} |)$.

Sortieren mit Threshold-Schaltkreisen in 5 Schritten

- n Zahlen $x_1, \ldots x_n$ mit jeweils n Bits sind zu sortieren.
- Ein Thresholdgatter kann zwei Zahlen vergleichen.

(1) Wir vergleichen *alle* Paare (x_i, x_j) mit $i \neq j$ in der ersten Schicht mit $n \cdot (n-1)/2$ Knoten.

(2) $\text{Rang}(x_i)$ ist definiert als die Position, an der der Schlüssel x_i in der sortierten Folge steht.
   - Für jedes i und k teste, ob $\text{Rang}(x_i) = k$ gilt.
   - Dies gelingt in zwei Schichten und $O(n^2)$ Knoten.

(3) Berechnung des r-ten Bits des s-ten Ausgabeschlüssels durch

$$\bigvee_{i=1}^{n} \Big((\text{Rang}(x_i) = s) \quad \wedge \quad x_{i,r} \Big).$$

Dies gelingt in zwei Schichten mit $O(n^3)$-vielen Knoten.

Neuronale Netze als Number Cruncher (1/3)

Addiere und multipliziere n n-Bit Zahlen mit einem Threshold-Schaltkreis **konstanter Tiefe** und **polynomieller Größe**.

Wie funktioniert die Multiplikation?

(1) Arbeite mit kleinen Primzahlen $p_1, \ldots, p_k \leqslant n^2$.

(2) Bestimme die Reste $x_i \mod p_\ell$:

- Da $x_i \equiv \sum_{j=0}^{n-1} 2^j x_{i,j} \mod p_\ell$, wird die Bestimmung der Reste durch die Addition von n Zahlen mit $O(\log n)$ Bits gelöst.

(3) Ein primitives Element g_ℓ modulo p_ℓ erzeugt alle Reste:

- Für jedes i und ℓ bestimme die Potenz $q_{i,\ell}$ mit $x_i \equiv g_k^{q_{i,\ell}} \mod p_\ell$ durch **Table Lookup**.

$$\text{Es ist } \prod_{i=1}^{n} x_i \mod p_\ell \equiv g_\ell^{\sum_{i=1}^{n} q_{i,\ell}} \mod p_\ell.$$

Neuronale Netze als Number Cruncher (2/3)

- Wir kennen $\prod_{i=1}^{n} x_i \equiv r_\ell \mod p_\ell$ für jedes ℓ.
- Rekonstruiere $\prod_{i=1}^{n} x_i$ mit dem *chinesischen Restsatz*.

(1) Kodiere die Zahlen $P_\ell = \frac{p_1 \cdots p_k}{p_\ell}$ wie auch das multiplikative Inverse P_ℓ^{-1} von P_ℓ modulo p_ℓ in den Schaltkreis.

(2) $\prod_{i=1}^{n} x_i \equiv \sum_{\ell=1}^{k} P_\ell \cdot P_\ell^{-1} \cdot r_\ell \mod \prod_{i=1}^{k} p_i$, denn **modulo $\mathbf{p}_\ell$**

$$\sum_{j=1}^{k} P_j P_j^{-1} r_j \equiv P_\ell P_\ell^{-1} r_\ell \equiv r_\ell \mod p_\ell.$$

Die Rekonstruktion des Produkts aus den Resten ist auf eine Addition polynomiell vieler Zahlen zurückgeführt.

Neuronale Netze als Number Cruncher (3/3)

n Zahlen mit *n* Bits können in konstanter Tiefe und polynomieller Größe addiert und multipliziert werden.

(1) $\Longrightarrow$ Threshold-Schaltkreise können schnell und effizient Polynome berechnen.

(2) $\Longrightarrow$ Approximiere analytische Funktionen wie

$$\ln(x), e^x \text{ und } \sigma(x) = \frac{1}{1 + e^{-x}}$$

durch ihr Taylorpolynome:

- Scharfe Approximation analytischer Funktionen in konstanter Tiefe und polynomieller Größe.

(3) Ob Rectifier, Softplus, Standard Sigmoid oder Threshold-Funktion: Berechnungskraft ist unverändert.

TC^0 = alle Sprachen, die in konstanter Tiefe und polynomieller Größe mit einem Threshold-Schaltkreis berechenbar sind.

Die Grenzen der Rechenkraft

Was können „kleine“ neuronale Netzwerke in konstanter Tiefe **nicht**?

(1) $\mathbf{NC^1}$ = alle Sprachen, die $\{\neg, \wedge, \vee\}$-Schaltkreise in logarithmischer Tiefe, Fan-in zwei (und polynomieller Größe) berechnen können.

(2) $\mathbf{TC^0} \subseteq \mathbf{NC^1}$:

- Man kann zeigen, dass ganzzahlige Gewichte und Schwellenwerte in $\{-n^{O(n)}, \ldots, n^{O(n)}\}$ für Eingaben aus $\{0,1\}^n$ ausreichend sind.
- Jetzt simuliere ein Threshold-Gatter durch einen $\{\neg, \wedge, \vee\}$-Schaltkreis in logarithmischer Tiefe.

In $\mathbf{NC^1}$ kann man (wahrscheinlich) nicht feststellen, ob ein gerichteter Graph einen Weg zwischen zwei festgelegten Knoten besitzt.

Beispielkomplexität

Dürfen wir bei fehlerfreiem Training jubeln?

Zumindest wenn die Gatterzahl mit $|S|$ skaliert, besteht selbst bei beliebigen reellwertigen Klassifikationen kein Anlass zum Jubel.

Sei $S \subseteq \mathbb{R}^d$ eine Menge von n Beispielen und sei $f : S \to \mathbb{R}$ eine beliebige Funktion. Dann gibt es Gewichte $a \in \mathbb{R}^d$, $w, b \in \mathbb{R}^n$, so dass

$$f(x) = \sum_{j=1}^{n} w_j \cdot \max\{ \langle a, x \rangle - b_j, 0 \}$$

für alle $x \in S$ gilt.

Ein neuronales Netz aus ReLU-Gattern mit nur $2n + d$ Gewichten genügt.

ReLU-Interpolation

Bestimme $w_1, \ldots, w_n,\ b_1, \ldots, b_n$ und $a_1, \ldots, a_d$ s.d. für alle $x \in S$

$$f(x) = \sum_{j=1}^{n} w_j \cdot \max\{\, \langle a, x \rangle - b_j,\, 0 \,\}$$

- Wähle $a \in \mathbb{R}^d$ zufällig. Es gelte o.B.d.A.

$$\langle a, x_1 \rangle < \langle a, x_2 \rangle < \cdots < \langle a, x_n \rangle.$$

- Wähle b so, dass

$$b_1 < \langle a, x_1 \rangle < b_2 < \langle a, x_2 \rangle < b_3 < \langle a, x_3 \rangle < \cdots < b_n < \langle a, x_n \rangle$$

- Für die $n \times n$ Matrix A mit $A_{i,j} = \begin{cases} 0 & i < j, \\ \langle a, x_i \rangle - b_j & \text{sonst} \end{cases}$
  löse das Gleichungssystem $A \cdot w = \big(f(x_1), \ldots, f(x_n)\big)$.
  - A ist eine obere Dreiecksmatrix ✓

Beispielkomplexität

Wieviele Beispiele müssen angefordert werden, wenn eine Architektur $\mathcal{A}$ aus Threshold-Gattern vorgegeben ist?

1. Für eine Klasse $\mathcal{F}$ von Funktionen $h : X \to Y$ definiere

$$\Pi_{\mathcal{F}}(S) := \{ f|_S : f \in \mathcal{F} \}$$

als die Menge der auf S eingeschränkten Funktionen aus $\mathcal{F}$.

2. Die Definition der **Wachstumsfunktion** $\Pi_{\mathcal{F}}(m)$ ist wie üblich, d.h.

$$\Pi_{\mathcal{F}}(m) := \max\{ |\Pi_{\mathcal{F}}(S)| : |S| = m \}.$$

Wie verhält sich die Wachstumsfunktion $\Pi_{\mathcal{F}}(m)$ bei kartesischen Produkten bzw. bei der Komposition von Funktionen?

Die Wachstumsfunktion

(a) Für Mengen $\mathcal{F}_1, \mathcal{F}_2$ von Funktionen $h : X \to Y$ ist

$$\mathcal{F}_1 \times \mathcal{F}_2 := \{\, h : \; h(x) = (h_1(x), h_2(x)) \text{ für } h_1 \in \mathcal{F}_1, h_2 \in \mathcal{F}_2 \,\}.$$

$$\Longrightarrow$$

$$\Pi_{\mathcal{F}_1 \times \mathcal{F}_2}(m) \leqslant \Pi_{\mathcal{F}_1}(m) \cdot \Pi_{\mathcal{F}_2}(m).$$

(b) Für Mengen $\mathcal{F}_1, \mathcal{F}_2$ von Funktionen $h_1 : X_1 \to X_2$ bzw $h_2 : X_2 \to X_3$ ist

$$\mathcal{F}_1 \circ \mathcal{F}_2 := \{\, h : \; h(x) = f_2(f_1(x)) \text{ für } h_1 \in \mathcal{F}_1, h_2 \in \mathcal{F}_2 \,\}.$$

$$\Longrightarrow$$

$$\Pi_{\mathcal{F}_1 \circ \mathcal{F}_2}(m) \leqslant \Pi_{\mathcal{F}_1}(m) \cdot \Pi_{\mathcal{F}_2}(m).$$

Die VC-Dimension einer Architektur (1/3)

Sei $\mathcal{A} = (V, E, f)$ eine Architektur aus Thresholdgattern $\Longrightarrow$

$$\mathrm{VC}(\mathcal{H}_{\mathcal{A}}) = \mathcal{O}(|E| \log_2 |E|).$$

1. Für einen Knoten $v \in V$ sei $\mathcal{H}_v$ die Menge aller Funktionen

$$h : \mathbb{R}^{\mathrm{ingrad}(v)} \to \mathbb{R},$$

   die von v (als einem Knoten der Tiefe Eins) berechnet werden.
2. Die VC-Dimension von $\mathcal{H}_v$ stimmt überein mit $\mathrm{ingrad}(v)$.
3. Mit Sauers Lemma folgt

$$\Pi_{\mathcal{H}_v}(m) \leqslant (e \cdot m)^{\mathrm{ingrad}(v)}$$

Wie verhält sich die Wachstumsfunktion, wenn der Schaltkreis rechnet?

Die VC-Dimension einer Architektur (2/3)

Sei $V_i \subseteq V$ die Menge aller Knoten $v \in V$ der Tiefe i.

Für den Graphen (V, E) ist $\mathcal{H}_{i+1}$ die Menge aller Funktionen

$$h : \mathbb{R}^{|V_i|} \to \{0,1\}^{|V_{i+1}|},$$

zu allen *möglichen* Rechnungen zwischen Schicht i und Schicht $i+1$.

4. Dann ist $\mathcal{H}_{i+1} = \prod_{v \in V_{i+1}} \mathcal{H}_v \Longrightarrow$

$$\prod_{\mathcal{H}_{i+1}}(m) \leqslant \prod_{v \in V_{i+1}} (e \cdot m)^{\text{ingrad}(v)}.$$

5. Wenn die Architektur aus den Schichten $V_0, \ldots, V_T$ besteht, folgt

$$\mathcal{H}_{\mathcal{A}} = \mathcal{H}_T \circ \cdots \circ \mathcal{H}_1.$$

und damit

$$\prod_{\mathcal{H}}(m) \leqslant \prod_{\mathcal{H}_1}(m) \cdots \prod_{\mathcal{H}_T}(m) \leqslant (e \cdot m)^{|E|}.$$

Die VC-Dimension einer Architektur (3/3)

6. Wenn $\mathrm{VC}(\mathcal{H}_{\mathcal{A}}) = d$, dann gilt

$$2^d \leqslant \prod_{\mathcal{H}_{\mathcal{A}}} (d) \leqslant (e \cdot d)^{|E|}$$

und deshalb

$$d = \mathcal{O}((|E|) \cdot \ln(d))$$

✓

7. Wird die Threshold-Funktion durch das Standard Sigmoid ersetzt:
   - Die VC-Dimension steigt an auf $\Omega(|E|^2)$.
   - Aber, Gewichte sind nur mit beschränkter Präzision B darstellbar:
     - ★ Höchstens $B^{|E|}$ verschiedene Hypothesen
     - ★ $\Longrightarrow$ die „praktische“ VC-Dimension ist höchstens $\mathcal{O}(|E|)$.

Und wenn es zu wenige Beispiele gibt?

Erzeuge künstliche Beispiele – **Data Augmentation** – bzw. erschwere Auswendiglernen „durch Hardware“ oder „durch Software“.

1. Hardware-Methoden:
   - Wende **Dimensionsreduktion** an, um zum Beispiel eine große Schicht von Knoten auf eine kleine Schicht zu komprimieren.
2. Software-Methoden:
   - Wende **Dropout-Verfahren** an, die in jedem Schritt einen konstanten Bruchteil aller Knoten zufällig auswählen und ausschalten.

Wie auch immer: Eine größtmögliche Anzahl qualitativ hochwertiger Beispiele ist entscheidend.

Lernen mit Gradientenabstieg

Lernen = Minimierung einer Loss-Funktion

Sei $\mathcal{H}_{\mathcal{A}}$ die Hypothesenklasse einer Architektur $\mathcal{A}$ und $S = \{ (x_1, y_1), \ldots, (x_s, y_s) \}$ eine Menge klassifizierter Beispiele.

Sei ℓ eine Loss-Funktion. Wir suchen eine ERM-Hypothese $h^w_{\mathcal{A}} \in \mathcal{H}_{\mathcal{A}}$ mit kleinstem Trainingsfehler

$$\begin{aligned} \text{Loss}^S(h^w_{\mathcal{A}}) &= \min_{u \in \mathbb{R}^n} \text{Loss}^S(h^u_{\mathcal{A}}) \\ &= \min_{u \in \mathbb{R}^n} \underbrace{\frac{1}{s} \cdot \sum_{i=1}^{s} \ell(u, x_i, y_i)}_{=:f(u)} . \end{aligned}$$

Ein erfolgreicher Lerner versucht die Funktion $f : \mathbb{R}^N \to \mathbb{R}$ zu minimieren: Ein Minimierungsproblem *ohne Restriktionen*.

Lokale Suche und Gradientenabstieg

Eine zweifach-differenzierbare Zielfunktion

$$f : \mathbb{R}^n \to \mathbb{R}$$

ist über einer kompakten Teilmenge des $\mathbb{R}^n$ zu minimieren.

Wir befinden uns im Punkt $a \in \mathbb{R}^n$:

In welcher Richtung befinden sich kleinere Funktionswerte?

- Approximiere f durch sein lineares Taylor-Polynom im Punkt a:

$$f(a+z) = f(a) + \nabla f(a)^T \cdot z + \mathcal{O}(z^T \cdot z)$$

  gilt, falls $||z||$ genügend klein ist.
- Setze $z = -\eta \cdot \nabla f(a)$ für hinreichend kleines $\eta > 0$:

$$f(a - \eta \cdot \nabla f(a)) = f(a) - \eta \cdot \ \|\nabla f(a)\|^2 + \eta^2 \cdot \mathcal{O}(\|\nabla f(a)\|^2).$$

Die Methode des Gradientenabstiegs

Sei $f : \mathbb{R}^n \to \mathbb{R}$ eine zweimal im Punkt $a \in \mathbb{R}^n$ stetig differenzierbare Funktion mit $\nabla f(a) \neq 0$. Dann gilt für hinreichend kleine $\eta > 0$

$$f\left(a - \eta \cdot \nabla f(a)\right) < f(a),$$

wobei $\nabla f(a)$ der Gradient von f an der Stelle a ist.

Die **Methode des Gradientenabstiegs** verfolgt die Iteration

$$w^{(i+1)} = w^{(i)} - \eta \cdot \nabla f(w^{(i)}).$$

1. Die *Lernrate* η muss in jedem Schritt neu bestimmt werden, da sonst die Gefahr droht, „über das Ziel hinaus zu schießen“.
2. Aber selbst wenn die Schrittweite stets hinreichend klein ist, dann konvergiert $\lim_{n \in \mathbb{N}} w^{(n)}$ mgl. gegen ein *lokales* Minimum.

Gradientenabstieg: Die Fragen

(a) Sollten die Gewichte für alle Beispiele auf einmal oder nacheinander für jedes einzelne Beispiel aktualisiert werden?

(b) Wie ist die Lernrate zu wählen?

(c) Vermeide ein „wildes Gehüpfe“: Sollten auch frühere Richtungen in der Wahl der neuen Richtung eine Rolle spielen?

Um diese Fragen zu beantworten, untersuche den Gradientenabstieg an einem „einfachen“, aber nicht-trivialem Minimierungsproblem,

der Minimierung konvexer Funktionen.

Die Minimierung konvexer Funktionen ist „einfach“

Zur Erinnerung:

1. Eine Teilmenge $K \subseteq \mathbb{R}^n$ heißt genau dann konvex, wenn für alle $x, y \in K$ auch alle Punkte der Geraden von x nach y in K liegen.
2. Die Funktion $f : K \to \mathbb{R}$ über der konvexen Menge K heißt genau dann **konvex**, wenn für alle $\lambda \in [0, 1], x, y \in K$ gilt

$$f(\lambda \cdot x + (1 - \lambda)y) \leqslant \lambda f(x) + (1 - \lambda) \cdot f(y).$$

- Wenn $a, b \in K$ lokale Minima von f sind, dann gilt $f(a) = f(b)$, und alle Punkte auf der Geraden von a nach b sind globale Minima.

  Wenn f differenzierbar ist, dann gilt:

  $(*)$ f ist konvex $\iff f(x) \geqslant f(a) + \nabla f(a)^T \cdot (x - a)$ für alle Punkte $a, x \in K$.
  /* Die Funktion liegt stets oberhalb einer Tangente. */

  $(*)$ Jeder Punkt $a \in K$ mit $\nabla f(a) = 0$ ist ein globales Minimum.
  /* Suche einen Punkt mit verschwindender Gradiente. */

Komplexitätsparameter konvexer Funktionen (1/2)

Die Funktion $f : K \to \mathbb{R}$ heißt genau dann **stark konvex** (mit Parameter mindestens $m > 0$), wenn für alle $a, x \in K$ gilt

$$f(x) \geqslant f(a) + \nabla f(a)^T \cdot (x - a) + \frac{m}{2} \cdot ||x - a||^2.$$

(m ist die **Krümmung** der quadratischen Funktion

$$f(x) = f(a) + \nabla f(a)^T \cdot (x - a) + \frac{m}{2} \cdot ||x - a||^2.$$

)

Die Funktion f sei zweifach stetig differenzierbar.

(a) f ist konvex $\iff$ die Hesse-Matrix $\nabla^2 f(a)$ ist in jedem Punkt a positiv semi-definit, d.h. es gilt $u^T \cdot \nabla^2 f(a) \cdot u \geqslant 0$ für alle $u \in K$.

(b) f ist stark konvex $\iff$ $\nabla^2 f(a)$ ist in jedem Punkt $a \neq 0$ positiv definit und es gilt $u^T \cdot \nabla^2 f(a) \cdot u \geqslant m \cdot ||u||^2$ für alle $u \in K$.

Komplexitätsparameter konvexer Funktionen (2/2)

$f : K \to \mathbb{R}$ heißt **L-glatt**, falls

$$||\nabla f(x) - \nabla f(y)|| \leqslant L \cdot ||x - y||$$

für alle $x, y \in K$ gilt.

(a) Die Funktion f sei L-glatt. Wenn f stark konvex mit Parameter mindestens m ist, dann ist $u^T \cdot \nabla^2 f(a) \cdot u \leqslant L \cdot ||u||$ für alle u.

(b) Definiere $\psi(x) := x - \eta \cdot \nabla f(x)$ als den auf x im Gradientenabstieg folgenden Punkt.

- Wir zeigen, dass ψ mit Kontraktionsfaktor $\beta \leqslant \frac{L-m}{L+m}$ kontrahierend ist, d.h. es gilt
$$||\psi(x) - \psi(z)|| \leqslant \beta \cdot ||x - z||.$$

- Wenn m nicht viel kleiner als L ist (d.h. f ist fast-quadratisch), dann ist β klein und ψ ist „stark“ kontrahierend.

ψ ist kontrahierend: Die Analyse (1/3)

Für eine reellwertige Konstante $\eta > 0$ ist

$$\psi(x) := x - \eta \cdot \nabla f(x)^T$$

Für die Einheitsmatrix E gilt:

$$\begin{aligned} ||\psi(x) - \psi(y)|| &= ||(x-y) - \eta \cdot (\nabla f(x) - \nabla f(y))|| \\ &\overset{(*)}{=} ||(x-y) - \eta \cdot \int_0^1 \nabla^2 f(y + \gamma \cdot (x-y)) \cdot (x-y)\, d\gamma|| \\ &= ||\underbrace{\left(E - \eta \cdot \int_0^1 \nabla^2 f(y + \gamma \cdot (x-y))\, d\gamma \right)}_{=:A} \cdot (x-y)||. \end{aligned}$$

Denn: $\nabla f(x + \gamma(y-x))$ ist Stammfunktion von $\nabla^2 f(x + \gamma \cdot (y-x)) \cdot (y-x)$.

ψ ist kontrahierend: Die Analyse (2/3)

Bestimme Schranken für die Größe von $u^T \cdot A \cdot u$, wobei

$$A := E - \eta \cdot \int_0^1 \nabla^2 f(y + \gamma \cdot (x - y))\, d\gamma$$

(a) Für jedes a ist $u^T \cdot \left(E - \eta \cdot \nabla^2 f(a) \right) \cdot u \leqslant (1 - \eta m) \cdot ||u||^2$, denn f ist stark konvex, und deshalb gilt $u^T \cdot \nabla^2 f(a) \cdot u \geqslant m \cdot ||u||$ für alle u.

(b) f ist L-glatt und deshalb gilt $u^T \cdot \nabla^2 f(a) \cdot u \leqslant L \cdot ||u||$. Als Konsequenz folgt $u^T \cdot \left(E - \eta \cdot \nabla^2 f(a) \right) \cdot u \geqslant (1 - \eta L) \cdot ||u||^2$ für alle u ($\Longrightarrow m \leqslant L$.)

Zusammenfassung: Für $a = y + \gamma(x - y)$ gilt für alle u

$$\underbrace{(1 - \eta \cdot L)}_{\lambda_{\min}(A)} \cdot ||u||^2 \leqslant u^T \cdot \left(\underbrace{E - \eta \cdot \nabla^2 f(a)}_{=A} \right) \cdot u \leqslant \underbrace{(1 - \eta \cdot m)}_{\lambda_{\max}(A)} \cdot ||u||^2.$$

ψ ist kontrahierend: Die Analyse (3/3)

Zur Erinnerung:

$$||\psi(x) - \psi(y) = ||\left(E - \eta \cdot \int_0^1 \nabla^2 f(y + \gamma(x-y))\, d\gamma\right) \cdot (x-y)|| \quad \text{und}$$

$$(1 - \eta \cdot L) \cdot ||u||^2 \leqslant u^T \cdot \left(E - \eta \cdot \nabla^2 f(a)\right) \cdot u \leqslant (1 - \eta \cdot m) \cdot ||u||^2.$$

für jedes a.

Da $||A \cdot z|| \leqslant \max\{\, |\lambda_{\min}(A)|, |\lambda_{\max}(A)| \,\} \cdot ||z||$ folgt

$$||\psi(x) - \psi(y)|| = ||A \cdot (x-y)|| \leqslant \max\{\, |1 - \eta \cdot L|, |1 - \eta \cdot m| \,\} \cdot ||x - y||.$$

Für $\beta := \frac{L-m}{L+m}$ und $\eta := \frac{2}{L+m}$ gilt $-\beta = 1 - \eta \cdot L \leqslant 1 - \eta \cdot m = \beta$. Also ist

$$||\psi(x) - \psi(y)|| \leqslant \beta \cdot ||x - y||.$$

Lineare Konvergenzgeschwindigkeit!

Die Funktion $f : K \to \mathbb{R}$ sei stark konvex mit Parameter mindestens m. Wenn f L-glatt ist, dann gilt für $\beta = \frac{L-m}{L+m}$ und $\eta := \frac{2}{L+m}$

$$||x^{(k)} - x^*|| \leqslant \beta^k \cdot ||x^{(0)} - x^*||.$$

($x^{(k)}$ ist der $k$te von Gradientenabstieg berechnete Vektor.) $\Longrightarrow$

Lineare Konvergenzgeschwindigkeit :-))

Wir wissen: $\psi(x) = x - \eta \nabla f(x)$ ist kontrahierend mit Faktor β.

Sei x^* das globale Minimum. Wegen $\nabla f(x^*) = 0$ folgt

$$\begin{aligned} ||x^{(k)} - x^*|| &= ||\left(x^{(k-1)} - \eta \cdot \nabla f(x^{(k-1)})\right) - \left(x^* - \eta \cdot \nabla f(x^*)\right)|| \\ &= ||\psi(x^{(k-1)}) - \psi(x^*)|| \leqslant \beta \cdot ||x^{(k-1)} - x^*|| \\ &\leqslant \cdots \leqslant \beta^k \cdot ||x^{(0)} - x^*||. \end{aligned}$$

Konvergenzgeschwindigkeit für glatte Funktionen

Die Funktion f sei L-glatt mit globalem Minimum x^*, aber mgl. nicht konvex. Für den $k$ten mit Lernrate $\eta = \frac{1}{L}$ berechneten Vektor $x^{(k)}$ gilt:

(a) $f(x^{(k+1)}) \leqslant f(x^{(k)}) - \frac{||\nabla f(x^{(k)})||^2}{2L}$.

(b) Nach k Schritten ist $\min_{t \leqslant k} ||\nabla f(x^{(t)})|| \leqslant \sqrt{\frac{2L \cdot |f(x^{(0)}) - f(x^*)|}{k}}$.

(Ein Punkt x mit kleinem Gradienten wird gefunden: x ist ein fast stationärer Punkt für den Gradientenabstieg.)

(c) Wenn f zusätzlich konvex ist, dann folgt

$$f(x^{(k)}) - f(x^*) \leqslant \frac{2L \cdot ||x^{(1)}) - x^*||^2}{k-1}.$$

(Langsame Konvergenz, wenn f nicht stark konvex ist.)

Beweis: Siehe Skript.

Lineare First-Order-Methoden

Eine **lineare First-Order-Methode** führt die Aktualisierung

$$x^{(k+1)} := x^{(0)} + \sum_{i=0}^{k} \gamma_i^k \cdot \nabla f(x^i)$$

– für vorgegebene reellen Zahlen γ_i^k – aus.

Beispiele linearer First-order-Methoden sind:

1. *Gradientenabstieg mit Schrittweite* η_k: Für alle $i \leqslant k$ setze $\gamma_i^k := -\eta_i$.
2. *Momentum*: Es gelte $0 \leqslant \beta < 1$ für den „Dämpfungsfaktor“ β. Setze

$$\begin{aligned} z^{(k+1)} &:= \beta \cdot z^{(k)} + \nabla f(x^{(k)}) \text{ und} \\ x^{(k+1)} &:= x^{(k)} - \eta \cdot z^{(k+1)}. \end{aligned}$$

   ($z^{(k)}$ ist das „Gedächtnis“ zum Zeitpunkt k. Ersetze den $(1-\beta)$-Bruchteil von $z^{(k)}$ durch den aktuellen Gradienten.)

3. *Nesterov's Accelerated Gradient*.

Eine optimale lineare First-Order-Methode

f sei konvex und L-glatt, aber mgl. nicht stark konvex.

Nesterov's Accelerated Gradientenverfahren besitzt die Iteration

$$\begin{aligned} z^{(k+1)} &:= x^{(k)} - \eta_k \cdot \nabla f(x^{(k)}) \\ x^{(k+1)} &:= z^{(k+1)} + \gamma_k \cdot (z^{(k+1)} - z^{(k)}). \end{aligned}$$

Unter sehr allgemeinen Annahmen an η_k, γ_k kann man zeigen:

$$f(x^{(k)}) - f(x^*) = \mathcal{O}\left(\frac{2L \cdot ||x^{(1)} - x^*||}{k^2}\right).$$

Schnellste Konvergenzgeschwindigkeit unter linearen First-Oder-Methoden.

Stochastischer Gradientenabstieg (SGD)

Die Situation

Sei $\mathcal{A}$ eine Architektur.

($*$) Für jede Hypothese $g \in \mathcal{H}_{\mathcal{A}}$ gibt es $w \in \mathbb{R}^n$ mit $g = h^w_{\mathcal{A}}$. Setze

$$\ell(w, x, y) := \ell(h^w_{\mathcal{A}}, x, y) \quad \text{und} \quad \text{Loss}_D(w) := \mathbb{E}_{(x,y)\sim D}[\,\ell(w, x, y)\,].$$

($*$) Löse das Minimierungsproblem

$$\min_{w \in \mathbb{R}^n} \text{Loss}_D(w).$$

1. Ziehe ein neues klassifiziertes Beispiel (x, y) gemäß D.
2. Berechne den Gradienten $\nabla\ell(w, x, y) \Longrightarrow$

$$\mathbb{E}_{(x,y)\sim D}[\,\nabla\ell(w, x, y)\,] \overset{!}{=} \nabla\mathbb{E}_{(x,y)\sim D}[\,\ell(w, x, y)\,] = \nabla\text{Loss}_D(w),$$

($\overset{!}{=}$ folgt aus der Linearität des Erwartungswerts.)

SGD: Das Programm

1. w sei ein Gewichtsvektor, η die Lernrate und T ein Parameter.
2. Ziehe ein „**Minibatch**“ $(x_1, y_1), \ldots, (x_T, y_T)$ von T klassifizierten Beispielen gemäß Verteilung D.

   /* *Wenn wenige Minibatches den Gradienten gut approximieren, wird die Laufzeit von Backpropagation drastisch reduziert.* */
3. Berechne

$$G(w) := \frac{1}{T} \cdot \sum_{j=1}^{T} \nabla \ell(w, x_j, y_j)$$

durch Parallelverarbeitung und aktualisiere

$$w := w - \eta \cdot G(w).$$

SGD und Perzeptron

Eine unbekannte lineare Thresholdfunktion $f : K \to \{-1, 1\}$ ist zu lernen.

Für die Minibatch-Größe $T = 1$ und die Loss-Funktion

$$\ell(w, x) := \max(-f(x) \cdot \langle w, x \rangle, 0)$$

stimmt SGD mit dem Perzeptron-Algorithmus überein.

Warum? Für ein beliebiges Beispiel $x \in K$ führt SGD die Aktualisierung

$$\begin{aligned} w^{(k+1)} &= w^{(k)} - \eta \cdot \nabla \ell(w^{(k)}, x) \\ &= w^{(k)} - \eta \cdot \begin{cases} f(x) \cdot x & \text{falls } f(x) \cdot \langle (w^{(k)}), x \rangle < 0 \\ 0 & \text{sonst.} \end{cases} \end{aligned}$$

aus $\Longrightarrow$ SGD = Perzeptron-Lernregel.

Eine Analyse von SGD unter Idealbedingungen

Die Annahmen:

- Die Zielfunktion $f : K \to \mathbb{R}$ sei stark konvex (mit Parameter m).
- Es gilt $||\nabla f(a)|| \leqslant L$ für alle Punkte $a \in K$.
- Die Minibatchgröße ist $T = 1$.

Ein Schritt von SGD

Wenn $x^{(k)}$ der $k$te von SGD für Schrittweite η_k berechnete Vektor und x^* das globale Minimum von f ist, dann folgt

$$\mathbb{E}[\,||x^{(k+1)} - x^*||^2\,] \leqslant (1 - 2m \cdot \eta_k) \cdot \mathbb{E}[\,||x^{(k)} - x^*||^2\,] + \eta_k^2 \cdot L^2.$$

Beweis: Siehe Skript.

Wähle η_k so, dass $\lim_{k\to\infty} \eta_k \ll \frac{1}{2m}, \frac{1}{L}$.

Die mutige und die vorsichtige Strategie

(a) Die **mutige Strategie**: Für die Schrittweite $\eta_k := \frac{1}{k \cdot m}$ gilt

$$\mathbb{E}[\,||x^{(k)} - x^*||^2\,] \leqslant \frac{Q}{k+1},$$

wobei $Q := \max\{\,||x^{(1)} - x^*||^2, \frac{L^2}{m^2}\,\}$.

(b) Die **vorsichtige Strategie**: Für die Schrittweite $\eta := \frac{\varepsilon \cdot m}{L^2}$ gilt

$$\mathbb{E}[\,||x^{(k)} - x^*||^2\,] \leqslant \varepsilon \quad \text{falls} \quad k \geqslant \frac{L^2}{2\varepsilon m^2} \cdot \log\left(\frac{2 \cdot \mathbb{E}[\,||x^{(0)} - x^*||^2\,]}{\varepsilon}\right).$$

Nach k Schritten wird also der Fehler $\varepsilon = \mathcal{O}(\frac{1}{k} \cdot \log k)$ erreicht.

- Konvergenzrate $1/k$ für die mutige Strategie.
- In Anwendungen wird die robustere vorsichtige Strategie bevorzugt: Ein Unterschätzen von m ist nicht tragisch.

Konvergenzrate $\Theta(1/k)$ für SGD

Die Gleichverteilung D möge Zahlen aus dem Intervall $[0, 1]$ erzeugen. Bestimme den Erwartungswert mit Hilfe von SGD möglichst genau.

SGD approximiert das Minimum $w^* = \frac{1}{2}$ der stark konvexen Funktion

$$f(w) := \mathbb{E}_{x \sim D}[(w - x)^2].$$

SGD kann die Schätzung $\frac{1}{k} \cdot \sum_{i=1}^{k} x_i$ nicht verbessern. Allerdings gilt

$$\text{prob}[\,|\frac{1}{k} \cdot \sum_{i=1}^{k} x_i - \frac{1}{2}| \geqslant \frac{c}{\sqrt{k}}\,] \geqslant \frac{1}{2}$$

für eine geeignete Konstante $c \Longrightarrow \mathbb{E}_{x \sim D}[\,(\frac{1}{k} \cdot \sum_{i=1}^{k} x_i - \frac{1}{2})^2\,] = \Omega(\frac{1}{k})$
$\Longrightarrow$ die Konvergenzrate $\Theta(1/k)$ ist optimal.

Gradientenabstieg oder SGD?

Für eine Menge S von s Beispielen und n Gewichten minimiere

$$f(w) := \text{Loss}^S(w) = \frac{1}{s} \cdot \sum_{i=1}^{s} \ell(w, x_i, y_i).$$

Später: Der Gradient $\nabla \ell(w, x_i, y_i)$ kann in Zeit $\mathcal{O}(n)$ bestimmt werden.
$\ell(w, x_i, y_i)$ sei für jedes Paar (x_i, y_i) stark konvex und differenzierbar.

- *Gradientenabstieg* berechnet $\nabla \ell(w, x_1, y_1), \ldots, \nabla \ell(w, x_s, y_s)$ in Zeit $\mathcal{O}(s \cdot n)$ und reduziert die erwartete Distanz zum Optimum um den konstanten Faktor β.
- *SGD* berechnet $\nabla \ell(w, x_i, y_i)$ in Zeit $O(n)$ und reduziert die Distanz zum Optimum nach s-maliger Wiederholung – also in Zeit $\mathcal{O}(s \cdot n)$ – um den Faktor $\frac{1}{s}$.

And the winner is .... SGD!

Wie gut funktioniert SGD?

Zitat von LeCun, Bengio und Hinton:

> *In practice, poor local minima are rarely a problem with large networks. Regardless of the initial conditions, the system nearly always reaches solutions of very similar quality. Theoretical and empirical results strongly suggest that local minima are not a serious issue in general. Instead, the landscape is packed with a combinatorially large number of saddle points where the gradient is zero, and the surface curves up in most dimensions and curves down in the remainder. The analysis seems to show that saddle points with only a few downward curving directions are present in very large numbers, but almost all of them have very similar values of the objective function. Hence, it does not much matter which of these saddle points the algorithm gets stuck at.*

Das Zitat wird sich auf Probleme der Human-Level AI (wie etwa Bilderkennung oder Spracherkennung) beziehen.

Human-Level AI

Zitat von Goodfellow et al.:

Modern deep learning provides a very powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples. Other tasks, that can not be described as associating one vector to another, or that are difficult enough that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now.

SGD: Eine Bestandsaufnahme

- ✓ SGD ist ein **schnelles** Verfahren, das die Bearbeitung großer Beispielmengen in tiefen Netzen erlaubt.
  - Ein entscheidender Vorteil von neuronalen Netzwerken gegenüber SVMs!
  - *Aber wie gelingt eine Berechnung des Gradienten $\nabla \ell(w, x, y)$ in Linearzeit*?
- ? Wird „nur“ die approximative Bestimmung eines lokalen Minimum verlangt, ist SGD möglicherweise geeignet.
- ↯ Ist ein globales Minimum approximativ zu bestimmen, dann ist SGD für nicht-konvexe Optimierungsprobleme i. A. überfordert.

Wie kann man Gradientenstieg bzw. SGD verbessern? Momentum, Nesterov's Accelerated Gradient, Polyak's Heavy-Ball-Methode.

Die Momentum-Methode

Die Momentum-Methode: Warum?

Welche Lernrate produziert den steilsten Abstieg?

- Wir suchen $\eta \in \mathbb{R}$, so dass $g(\eta) := f(x - \eta \nabla f(x))$ minimal ist.
- Bestimme η, so dass $0 \stackrel{!}{=} \frac{dg}{d\eta} = \nabla f(x - \eta \nabla f(x))^T \cdot \nabla f(x)$.

In „*steepest descent*“ stehen aufeinanderfolgende Richtungen senkrecht aufeinander: Unterbinde diese chaotischen Sprünge.

Die *Momentum-Methode*: Die Funktion $f : K \to \mathbb{R}$ sei differenzierbar und es gelte $0 \leqslant \beta < 1$ für den „Dämpfungsfaktor“ β. Setze

$$\begin{aligned} z^{(k+1)} &:= \beta \cdot z^{(k)} + \nabla f(x^{(k)}) \text{ und} \\ x^{(k+1)} &:= x^{(k)} - \eta \cdot z^{(k+1)}. \end{aligned}$$

$z^{(k)}$ ist das Gedächtnis zum Zeitpunkt k. Ersetze den Bruchteil $1 - \beta$ von $z^{(k)}$ durch den aktuellen Gradienten.

Momentum für quadratische Funktionen (1/3)

Die konvexe, quadratische Funktion

$$f(x) := \frac{1}{2} \cdot x^T Ax - b^T x$$

ist gegeben: Die $n \times n$-Matrix A sei symmetrisch und nicht-singulär.

Wann ist das Minimierungsproblem für f schwierig?

Sei A eine symmetrische Matrix mit größtem Eigenwert λ_{max} und kleinstem Eigenwert λ_{min}. Dann ist

$$\kappa(A) := \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$$

die Konditionszahl von A.

Je größer $\kappa(A)$, umso mehr verhält sich A wie eine singuläre Matrix.

Momentum für quadratische Funktionen (2/3)

- Im Vergleich zum konventionellen Gradientenabstieg verbessert sich die Konvergenzgeschwindigkeit von

$$\left(\frac{\kappa-1}{\kappa+1}\right)^k = \left(1-\frac{2}{\kappa+1}\right)^k \approx \exp^{-2k/(\kappa+1)}$$

  auf

$$\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx \exp^{-2k/(\sqrt{\kappa}+1)}.$$

- Gradientenabstieg erreicht den Abstand von ε nach $\Theta((\kappa+1)\ln\frac{1}{\varepsilon})$ Iterationen, Momentum nach nur $\Theta((\sqrt{\kappa}+1)\ln\frac{1}{\varepsilon})$ Iterationen.

Große Vorteile für Momentum bei **großen** Konditionszahlen.

Momentum für quadratische Funktionen (3/3)

(a) Betrachte die Zielfunktion $f_n : \mathbb{R}^n \to \mathbb{R}$ mit

$$f_n(x) = \frac{1}{2} \cdot (x_1 - 1)^2 + \frac{1}{2} \cdot \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2 + \frac{2}{\kappa - 1} \cdot ||x||^2$$

- Zeige, dass x^* mit $x_i^* = (\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^i$ eine optimale Lösung ist.
- Zeige, dass die Konditionszahl von f_n gegen κ konvergiert.

(b) Führe eine First-Order-Methode, beginnend mit $x^{(0)} = 0$, aus.

- Zeige: Für jede First-Order-Methode gilt

$$||x^{(k)} - x^*||_\infty \geqslant (\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1})^k \cdot ||x^{(0)} - x^*||.$$

- Zeige, dass f_n am schnellsten von Momentum minimiert wird.

Momentum ist – unter der Perspektive der Konditionszahl – im Worst-Case eine optimale First-Order-Methode für quadratische Funktionen.

Loss-Funktionen für Backpropagation

Welche Loss-Funktion?

Welche Loss-Funktion $\ell : \mathbb{R}^{|E|} \times X \times Y \to \mathbb{R}$ wird gewählt?

- In den 80'er bis 90'er Jahren: Der **quadratische Loss**

$$\ell(w, x, y) := \frac{1}{2} \cdot (h^w_{\mathcal{A}}(x) - y)^2.$$

- Annahme: Hypothese $h^w_{\mathcal{A}}(x)$ bestimmt $p_w(y \mid x)$ als Wahrscheinlichkeit für die tatsächliche Klassifikation y von Beispiel x.
  - Der **Log-Loss**

$$\ell(w, x, y) := -\log p_w(y \mid x).$$

    Versuche Klassifikationsvorgaben hochwahrscheinlich einzuhalten.
  - Der empirische **empirische Log-Loss** stimmt überein mit

$$\text{Loss}^S(w) = \sum_{i=1}^{s} \ell(w, x_i, y_i) = -\sum_{i=1}^{s} \log p_w(y_i \mid x_i) = -\log \prod_{i=1}^{s} p_w(y_i \mid x_i).$$

    $\Longrightarrow$ ERM-Hypothesen w maximieren $\prod_{i=1}^{s} p_w(y_i \mid x_i)$.

Die Minimierung von $\text{Loss}_D(w)$ (1/2)

Für eine unbekannte Verteilung D erzeugt D das Paar (x, y) mit Wahrscheinlichkeit $p_D(x, y)$. Wann wird $\text{Loss}_D(w)$ minimiert?

Für den Log-Loss $\ell(w, x, y) = -\log p_w(y \mid x)$ und für jede Verteilung D auf $X \times Y$ stimmt $\text{Loss}_D(w)$ überein mit

$$\sum_{x \in X} p_D(x) \cdot \underbrace{\sum_{y \in Y} p_D(y \mid x) \cdot \log \left(\frac{p_D(y \mid x)}{p_w(y \mid x)} \right)}_{\text{Kullback-Leibler Divergenz} \geqslant 0} + \sum_{x \in X} p_D(x) \cdot \underbrace{\sum_{y \in Y} p_D(y \mid x) \cdot \log \left(\frac{1}{p_D(y \mid x}}_{\text{Entropie } H(p_D(* \mid x))}$$

w^* ist eine minimale Lösung von $\text{Loss}_D(w) \iff$
w^* minimiert die „mittlere“ Kullback-Leibler-Divergenz zwischen p_D, p_w.

Die Minimierung von $\text{Loss}_D(w)$ (2/2)

$$\text{Loss}_D(w) = \mathbb{E}_{(x,y)\sim D}[\,\ell(w,x,y)\,] = -\sum_{(x,y)\in X\times Y} p_D(x,y)\cdot \log p_w(y\mid x)$$

$$= \sum_{x\in X} p_D(x)\cdot \sum_{y\in Y} p_D(y\mid x)\cdot \log\left(\frac{1}{p_w(y\mid x)}\right)$$

$$= \sum_{x\in X} p_D(x)\cdot \underbrace{\sum_{y\in Y} p_D(y\mid x)\cdot \log\left(\frac{p_D(y\mid x)}{p_w(y\mid x)}\right)}_{\text{Kullback-Leibler Divergenz}}$$

$$+ \sum_{x\in X} p_D(x)\cdot \underbrace{\sum_{y\in Y} p_D(y\mid x)\cdot \log\left(\frac{1}{p_D(y\mid x)}\right)}_{\text{Entropie } H(p_D(*\mid x))}$$

Die Kreuz-Entropy (1/2)

Die **Kreuz-Entropie** (engl. **Cross Entropy**) zwischen Verteilungen p und q auf einer Zielmenge Ω ist

$$C(p;q) := \sum_{\omega\in\Omega} p(\omega)\cdot\log\frac{1}{q(\omega)}$$

Die Kullback-Leibler Divergenz $D(p\,||\,q) = \sum_{\omega\in\Omega} p_\omega\cdot\log\frac{p_\omega}{q_\omega}$ ist nicht-negativ $\Longrightarrow$

$$C(p,q) = \sum_{\omega\in\Omega} p(\omega)\cdot\log\frac{1}{q(\omega)} \geqslant -\sum_{\omega\in\Omega} p(\omega)\cdot\log p(\omega) = H(p).$$

$C(p,q)$ ist mindestens so groß wie die Entropie $H(p)$ von p und umso kleiner, je ähnlicher p und q sind.

Die Kreuz-Entropy (2/2)

Es ist $C(p; q) := \sum_{\omega \in \Omega} p(\omega) \cdot \log \frac{1}{q(\omega)}$.

Für den Log-Loss $\ell(w, x, y) = -\log p_w(y \mid x)$ gilt

$$\begin{aligned} \text{Loss}_D(w) &= \mathbb{E}_{(x,y)\sim D}[\,\ell(w, x, y)\,] = -\sum_{(x,y)\in X\times Y} p_D(x, y) \cdot \log p_w(y \mid x) \\ &= \sum_{(x,y)\in X\times Y} p_D(x, y) \cdot \log \frac{1}{p_w(y \mid x)} = C(p_D; p_w) \end{aligned}$$

Es ist $\text{Loss}_D(w) = C(p_D, p_w)$.

Reinforcement-Lernen: AlphaZero

Google's AlphaZero spielt Go, Schach und Shogi auf absolutem Spitzenniveau. AlphaZero lernt **nur** durch das Spielen gegen sich selbst.

(a) In Stellung s sagt AlphaZero die Gewinnwahrscheinlichkeit g_s des ziehenden Spielers voraus ebenso wie die vermutete Gewinnwahrscheinlichkeit $p_s(z)$, wenn der ziehende Spieler z wählt.

(b) AlphaZero führt eine Monte-Carlo Suche aus und simuliert für Stellung s ca. 800 Spiele gegen sich selbst:

- Bevorzuge selten gespielte Züge z mit großer Gewinnwahrscheinlichkeit $p_s(z)$.
- In Stellung s sei h_s die *empirisch* festgestellte erwartete Gewinnwahrscheinlichkeit in s und $q_s(z)$ die *empirisch* festgestellte Gewinnwahrscheinlichkeit von Zug z.

(c) Für eine Konstante α benutzt AlphaZero die **Loss-Funktion**

$$\ell(w, s) := (g_s - h_s)^2 - q_s^T \ln p_s + \alpha \cdot ||w||^2$$

- Der Regularisierungsterm $\alpha \cdot ||w||^2$ hält den Gewichtsvektor w klein.
- Die Kreuz-Entropie $-q_s^T \ln p_s$ ist minimal wenn $p_s = q_s$.

Die Bedeutung der Loss-Funktion

Zur Relevanz der Loss-Funktion, bzw. zum Einfluß einer probabilistischen Sichtweise ein Zitat aus Goodfellow et al., Seiten 218-219:

> *Mean squared error was popular in the 1980s and 1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.*
> *The use of cross entropy losses greatly improved the performance of models with sigmoid and softmax outputs, which had previously suffered from saturation and slow learning using the mean squared error.*

Backpropagation

Backpropagation = Schnelle Implementierung von SGD

1. S sei eine „genügend große“ Beispielmenge. Setze $k = 0$. $w^{(0)}$ sei ein zufällig ausgewürfelter Vektor von Gewichten.
2. Wiederhole M Mal:
   (a) Ziehe einen Minibatch $(x_1^{(m)}, y_1^{(m)}), \ldots, (x_T^{(m)}, y_T^{(m)})$ von T klassifizierten Beispielen aus S.
   (b) Berechne den gemittelten Gradienten (*aber wie?*)

$$G(w) := \frac{1}{T} \cdot \sum_{j=1}^{T} \nabla \ell(w^{(m)}, x_j^{(m)}, y_j^{(m)})$$

   durch Parallelverarbeitung mit **Backpropagation** und aktualisiere

$$w^{(k+1)} := w^{(k)} - \eta_k \cdot G(w^{(k)}).$$

   (c) Setze $k := k + 1$.

Die beiden Phasen von Backpropagation

Backpropagation besteht aus zwei Phasen:

1. **Vorwärtsphase**: Bestimme, beginnend mit den Quellen, $\text{summe}_v(x)$, $\text{aus}_v(x) := \gamma_v(\text{summe}_v(x))$ für jeden Knoten v.
2. **Rückwärtsphase**: Berechne $\nabla \ell(w, x, y)$ „rückwärts“, beginnend mit den Senken. y ist die tatsächliche Klassifizierung von Beispiel x.

Wir machen die folgenden **Annahmen**:

(a) Die Architektur $\mathcal{A} = (V, E, f)$ ist vorgegeben.

(b) Unterscheide nicht zwischen Kanten und „Nicht-Kanten“:
- Aber berechne $\frac{\partial \ell(w,x,y)}{\partial e}$ nur für Kanten,
- Fasse Nicht-Kanten nicht an.

(c) Eingaben für Backpropagation: Beispiel **x** und Klassifizierung **y**.

(d) Wir arbeiten der Einfachheit halber mit dem quadratischen Loss $\ell(w, x, y) := \frac{1}{2} \cdot \sum_j \left(\text{aus}_{T,j}(x) - y_j\right)^2$ auf den Ausgabeknoten (T, j).

Die Rückwärtsphase (1/2)

Die Kante $e \in E$ sei in einen Knoten der Tiefe $\leqslant t$ gerichtet.
Bestimme die partielle Ableitung

$$\frac{\partial \ell(w, x, y)}{\partial e}$$

der Loss-Funktion nach dem Gewicht $w(e)$ der Kante e.

$$\frac{\partial \ell(w, x, y)}{\partial e} = \sum_{j=1}^{|V_T|} \underbrace{(\text{aus}_{T,j}(x) - y_j)}_{=:\delta_{T,j}} \cdot \frac{\partial\, \text{aus}_{T,j}}{\partial\, e} = \sum_{j=1}^{|V_T|} \delta_{T,j} \cdot \frac{\partial\, \text{aus}_{T,j}}{\partial\, e}$$

Wir nehmen induktiv an, dass die *Fehlersignale* $\delta_{s,j}$ für alle Knoten (s, j) und alle Tiefen s mit $t + 1 \leqslant s \leqslant T$ definiert wurden, wobei gilt

$$\frac{\partial \ell(w, x, y)}{\partial e} = \sum_{j=1}^{|V_s|} \delta_{s,j} \cdot \frac{\partial\, \text{aus}_{s,j}}{\partial\, e}.$$

Die Rückwärtsphase (2/2)

Es gelte $\frac{\partial \ell(w,x,y)}{\partial e} = \sum_{j=1}^{|V_s|} \delta_{s,j} \cdot \frac{\partial \, \text{aus}_{s,j}}{\partial \, e}$ für $t+1 \leqslant s \leqslant T \Longrightarrow$

$$\frac{\partial \ell(w,x,y)}{\partial e} = \sum_{j=1}^{|V_{t+1}|} \delta_{t+1,j} \cdot \frac{\partial \, \text{aus}_{t+1,j}}{\partial \, e}$$

$$= \sum_{j=1}^{|V_{t+1}|} \delta_{t+1,j} \cdot \frac{\partial \, \gamma_{t+1,j}\Big(\text{summe}_{t+1,j}(x)\Big)}{\partial \, e}$$

$$= \sum_{j=1}^{|V_{t+1}|} \delta_{t+1,j} \cdot \gamma'_{t+1,j}(\text{summe}_{t+1,j}(x)) \cdot \frac{\partial \, \text{summe}_{t+1,j}}{\partial \, e}$$

$$= \sum_{k=1}^{|V_t|} \underbrace{\sum_{j=1}^{|V_{t+1}|} \delta_{t+1,j} \cdot \gamma'_{t+1,j}(\text{summe}_{t+1,j}(x)) \cdot w((t,k),(t+1,j))}_{=:\delta_{t,k},\ \text{das Fehlersignal von Knoten } (t,k)} \cdot \frac{\partial \, \text{aus}_{t,k}}{\partial \, e}$$

Backpropagation: Das Programm

1. Führe die Vorwärtsphase aus.
2. Für alle Knoten $(T, j) \in V_T$: Setze $\delta_{T,j} := \text{aus}_{T,j}(x) - y_j$.
3. Für alle Schichten $t = T - 1, \ldots, 1$ in absteigender Reihenfolge und für alle Knoten $(t, k) \in V_t$:

$$\delta_{t,k} := \sum_{j=1}^{|V_{t+1}|} \delta_{t+1,j} \cdot \gamma'_{t+1,j}(\text{summe}_{t+1,j}(x)) \cdot w((t,k),(t+1,j)).$$

4. Für alle Schichten $t \geqslant 1$, für alle Kanten $e = ((t-1,i),(t,k)) \in E$

$$\frac{\partial \ell(w,x,y)}{\partial e} = \delta_{t,k} \cdot \gamma'_{t,k}(\text{summe}_{t,k}(x)) \cdot \text{aus}_{t-1,i}(x).$$

/* Denn $\frac{\partial \text{aus}_{t,k}}{\partial e} = \gamma'_{t,k}(\text{summe}_{t,k}(x)) \cdot \text{aus}_{t-1,i}(x)$. */

Backpropagation: Der Aufwand

Jede Aktivierungsfunktion der Architektur $\mathcal{A} = (V, E, f)$ sei in $\mathcal{O}(1)$ Schritten auswertbar $\Longrightarrow$

Die Laufzeit von Backpropagation ist durch $\mathcal{O}(|V| + |E|)$ beschränkt.

- Für jede Operation von Backpropagation, ob in Vorwärts- oder Rückwärtsphase, ist eine Kante oder ein Knoten verantwortlich.
- Umgekehrt ist jede Kante oder jeder Knoten in jeder Phase nur für eine Operation verantwortlich.

Backpropagation-Through-Time (BPTT): Lernen für RNNs

Unidirektionale RNNs

Ein RNN $\mathcal{R} = (\mathcal{A}, w)$ besitzt eine Architektur $\mathcal{A} = (V, E, f)$ mit einem *beliebigen* gerichteten Graphen $G = (V, E)$.

Unidirektionales Lernen: „Wickle“ $\mathcal{R}$ in ein Feedforward-Netz $\mathcal{R}^{(t)}$ „ab“.

- Das Feedforward Netz $\mathcal{R}^{(t)}$ besteht aus t Schichten, wobei Schicht i die Knotenmenge $V \times \{i\}$ besitzt.
  - Für jeden Knoten $r \in V$ besitzen alle „Kopien“ (r, i) für $1 \leqslant i \leqslant t$ dieselbe Aktivierungsfunktion f_r.
  - Für jede Kante $e = (r, s) \in E$ und alle $1 \leqslant i < t$ setze die Kante $((r, i), (s, i + 1)$ mit Gewicht $w(e)$ ein.
- **BPTT**: Wende Backpropagation auf $\mathcal{R}^{(t)}$ an.
  - Bestimme die Gradienten g_i für die Gewichte der Kanten, die in einem Knoten aus $V \times \{i\}$ beginnen.
  - Aktualisiere den gegenwärtigen Gewichtsvektor w von $\mathcal{R}$ durch

$$w = w - \eta \cdot \sum_{i=1}^{t} g_i.$$

Bidirektionale RNNs

Bidirektionales Lernen:

- „Wickle“ $\mathcal{R}$ in zwei Feedforward-Netze, nämlich eine Vorwärtsversion $\mathcal{R}_V^{(t)}$ und eine Rückwärtsversion $\mathcal{R}_R^{(t)}$ ab.
  - Die Knoten $r \in V$ besitzen für jedes i ($1 \leqslant i \leqslant t$) eine „Vorwärtskopie“ $(r,i)_V$ und eine „Rückwärtskopie“ $(r,i)_R$.
  - Für jede Kante $e = (r,s) \in E$:
    - Die Vorwärtsversion $\mathcal{R}_V^{(t)}$ benutzt nur Vorwärtskopien der Knoten und setzt die Kanten $((r,i)_V, (s,i+1)_V$ ein.
    - Die Rückwärtsversion $\mathcal{R}_R^{(t)}$ benutzt nur Rückwärtskopien und setzt Kanten $((s,i)_R, (r,i+1)_R$ von Gegenwart in Vergangenheit ein.
- **BPTT**: Wende Backpropagation auf beide Feedforward-Netze an
- und „konkateniere“ die Ergebnisse: Z.B. mittel Ergebnisse aus Vorwärts- und Rückwärtskopie oder behalte beide Ergebnisse bei.

Bidirektionale RNNs: Vergangenheit und Zukunft sind gleichberechtigt.

Anwendungen: Visuelle Objekterkennung

Die CFAR-Benchmarks

- CFAR-10 besteht aus ca. 60.000 zweidimensionalen Bildern, die jeweils durch eine $32 \times 32 \times 3$-Pixelmatrix beschrieben werden.
  - Ein Pixel mit Koordinaten (x, y) wird durch das Tupel (x, y, p) repräsentiert: $p : \{1, 2, 3\} \to \mathbb{R}$ legt die Werte für drei Farbkanäle fest.
  - Jedes Bild ist einer der 10 Kategorien „Flugzeug, Auto, Vogel, Katze, Wild, Hund, Frosch, Pferd, Schiff, Lastwagen“ zugeordnet.
- CFAR-100 besitzt 100 Kategorien, die wiederum in 20 Superkategorien zusammengefasst sind.
  - Jede Kategorie besteht aus ungefähr 600 Bildern.
  - Jedes Bild ist mit seiner Kategorie und seiner Superkategorie versehen.

Die Einteilung der Bilder in (Super-)Kategorien ist zu lernen.

Die Benchmark ImageNet

ImageNet besteht aus ca. 14 Millionen, im Internet veröffentlichten Bildern sowie aus ca. 20.000 Kategorien:

- Bilder werden durch Freiwillige mit einer oder mehreren dieser Kategorien annotiert.
- Pro Kategorie mehrere 100 Bilder.

Der wichtige Wettbewerb **ILSVRC** (ImageNet Large-Scale Visual Recognition Challenge) basiert auf ImageNet.

Convolutional Neural Networks (CNNs)

Warum CNNs?

Zitat von LeCun, Bengio und Hinton:

There was, however, one particular type of deep, feedforward network that was much easier to train and generalized much better than networks with full connectivity between adjacent layers. This was the convolutional neural network (ConvNet). There are four key ideas behind ConvNets that take advantage of the properties of natural signals:

local connections, shared weights,
pooling and the use of many layers.

CNNs: Ein (sehr) kurzer Überblick (1/3)

CNNs sind Feedforward-Netzwerke, die eine Eingabe in (2D oder 3D) **Gitterstruktur** erwarten.

- Die Gitterstruktur wird in nachfolgenden Schichten beibehalten.
- **Faltungsschichten** (Convolutional Layer) und **Pooling-Schichten** wechseln sich mehrmals ab.
- Das Netz endet mit einer oder zwei Schichten, die jeweils vollständig mit ihrer jeweiligen Vorgänger-Schicht verbunden sind.
  - Die Softmax-Funktion wird häufig in der letzten Schicht eingesetzt, um die „Überzeugung“ hinter den jeweiligen Optionen zu messen.

CNNs: Ein (sehr) kurzer Überblick (2/3)

Wir betrachten eine ***Faltungsschicht***.

(a) Derselbe Gitterausschnitt – wie z.B. ein $k \times k$ Teilgitter – wird über das Gitter der Vorgängerschicht verschoben.

- Die „Pixel“ aller Ausschnitte in identischer Position werden dann – jeweils mit ***denselben*** Koeffizienten – gewichtet aufsummiert.
  - ⋆ Die Methode der **shared weights** wird angewandt.
  - ⋆ Bei k Feature Funktionen ist derselbe Ausschnitt mit k unterschiedlichen Gewichtsätzen zu lernen.
- Häufig werden ReLU-Gatter verwandt.

$\Longrightarrow$ **Translationsinvarianz** bei wenigen Freiheitsgraden!

(b) Biologische Plausibilität:

- In den ersten Schichten des **visuellen Cortex** besitzen die Neuronen **kleine rezeptive Felder**,
- die sich nach Hintereinanderausführung in nachfolgenden Schichten entsprechend vergrößern.

CNNs: Ein (sehr) kurzer Überblick (3/3)

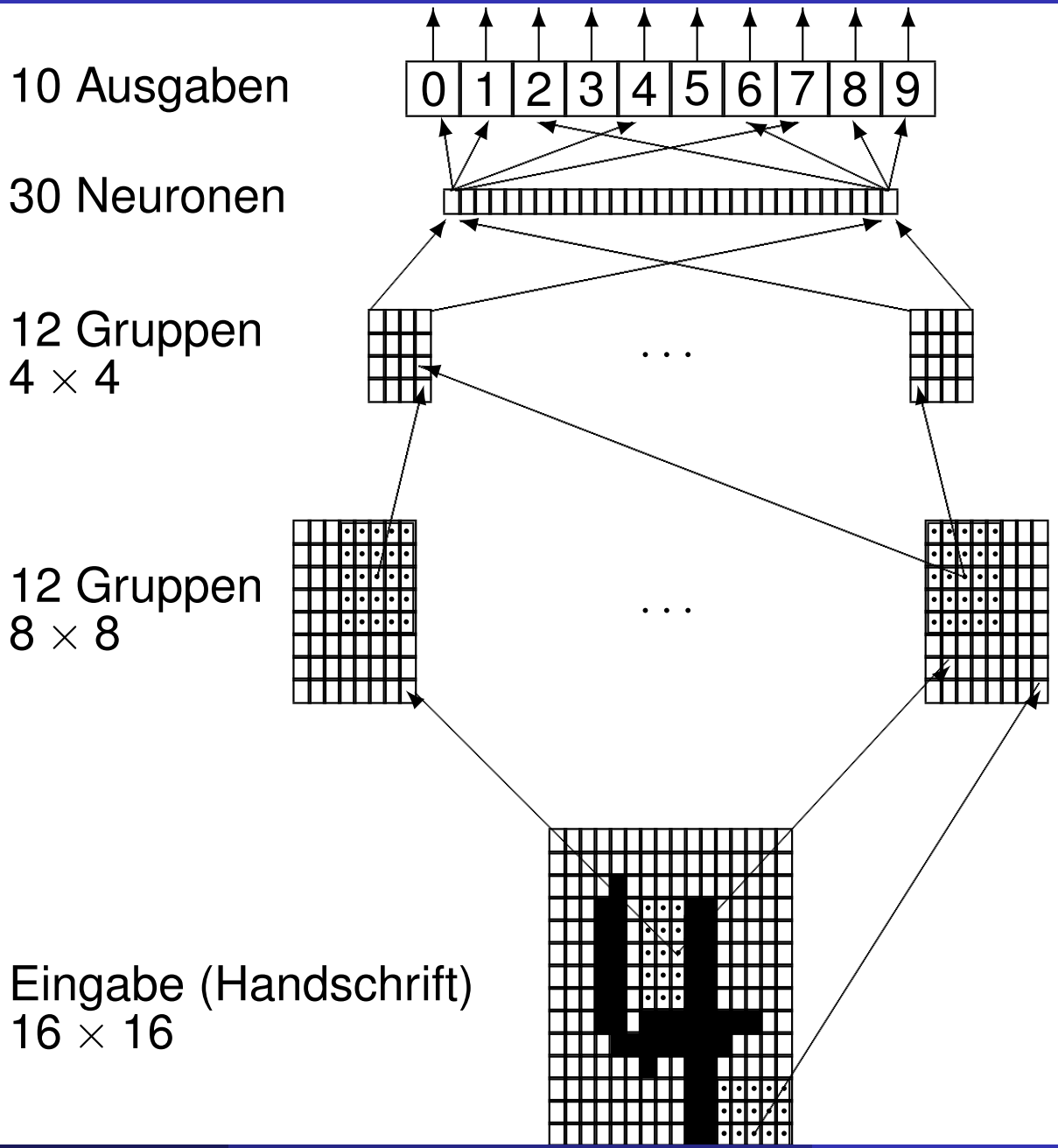
Pooling-Schichten wechseln sich mit Faltungsschichten ab und führen eine Dimensionsreduktion aus.

(a) Häufiges Beispiel ist die **Maximumbildung**:

- Wähle einen Knoten v mit größtem Wert „aus$_v(x)$“ in einer kleinen Nachbarschaft aus.
- Vorbild aus der Biologie: Die „laterale Hemmung“, die möglicherweise der Kontrastverstärkung dient.

(b) Durchschnittbildung tauchen seltener auf.

LeNet: Ziffernerkennung



(a) Die Eingabe ist ein 16×16-Pixel-Bild der Ziffer.

(b) Die erste Schicht besteht aus 12 Gruppen zu je 64 Neuronen.

- Jedes Neuron erhält die Informationen eines 5×5-Ausschnitts
- Jede Gruppe der ersten Schicht besitzt $5^2 = 25$ **geteilte** Gewichte. Jedes der 64 Neuronen erhält einen individuellen Schwellenwert.
  - ★ Insgesamt fallen hier $12 \cdot (25 + 64) = 1068$ Freiheitsgrade an.

(c) Die zweite Schicht besteht aus 12 Gruppen zu je 16 Neuronen.

- Jedes Neuron einer Gruppe betrachtet „5×5-Ausschnitte“ von 8 der 12 Gruppen der vorherigen Stufe.
- Jedes Neuron einer Gruppe erhält einen eigenen Schwellenwert, die Neuronen einer Gruppe **teilen** die $8 \cdot 5^2$ Gewichte.
  - ★ Wir erhalten $12 \cdot (8 \cdot 25 + 16) = 2592$ Freiheitsgrade.

(d) Die dritte Schicht besteht aus 30 Neuronen, die mit mit allen Neuronen der vorherigen Stufe verbunden sind.

- Es wird kein weight-sharing benutzt!
  - ★ $30 \cdot (12 \cdot 4^2 + 1) = 5790$ Freiheitsgrade treten auf.

(e) Die vierte und letzte Schicht besitzt ein Neuron pro Ziffer.

- Es wird kein weight-sharing benutzt!
  - ★ $10 \cdot (30 + 1) = 310$ Freiheitsgrade treten auf.

AlexNet

AlexNet, eine Weiterentwicklung von LeNet, hat die ILSVRC in 2012 mit deutlichem Abstand gewonnen.

(a) Die Methode des Weight Sharing wird übernommen.

(b) Alexnet ist tiefer und besteht aus 5 Faltungsschichten, 2 Max-Pooling Schichten und 3 vollständig verbundenen Schichten.

(c) **Data Augmentation** wird eingesetzt, nämlich Spiegelbild-Transformationen, Bildung von Ausschnitten, Verschieben von Bildern.

(d) Des Weiteren wird SGD in Kombination mit Momentum und dem **Dropout-Verfahren** angewandt.

Die nackten Zahlen:

- AlexNet besteht aus 650.000 Gattern und 60 Millionen zu bestimmenden Gewichten bei ingesamt 630 Millionen Kanten.
- Insgesamt wurden 580 GPUs für ca. sechs Tage trainiert.

GoogleNet

GoogLeNet hat die ILSVRC in 2014 gewonnen.

(a) Statt der 8 Schichten von AlexNet werden 22 Schichten verwandt,

(b) gleichzeitig wird die Parameterzahl von 60 auf 4 Millionen reduziert.

Die Architektur von GoogleNet:

- Die Faltungsschichten von GoogLeNet benutzen parallel 1×1, 3×3 und 5×5-Auflösungen im 3D-Würfel von Position und Farbtiefe.
  - Die 1×1-Auflösung ist eine Dimensionsreduktion: Für jeden „Pixel“ berechne die gewichtete Farbtiefensumme über alle Farbkanäle.
  - Alle drei Faltungen werden an die nächste Schicht weitergeleitet und mit Max-Pooling abgeschlossen.
- GoogLeNet verwendet keine vollständig verbundenen Schichten.

ResNet

ResNet ist Gewinner des ILSVRC-Wettbewerbs in 2015.

(a) ResNet ist wesentlich tiefer als seine Vorgänger und arbeitet mit 152 Schichten und

(b) setzt „**Skip-Verbindungen**“ ein, um *verschwindende* oder *explodierende* Gradienten besser kontrollieren zu können.

- Wiederholte Multiplikationen beim Rückwärtspropagieren erniedrigen bzw. erhöhen den Gradienten $\Longrightarrow$ Die Werte der Aktivierungsfunktionen ändern sich kaum.

Skip-Verbindungen entsprechen Kanten von einem Knoten einer Schicht *i* zu Kopien in höheren Schichten.

Mit Skip-Verbindungen wird versucht, „mit Hardware“ das Vergessen zu verhindern. (Siehe LSTM-Blöcke später.)

Speech-to-Text,
vom Audiosignal zum Text

Automatic Speech Recognition (ASR)

ASR-Systeme erstellen die Transkription eines akustischen Signals.

1. 8.000 - 16.000 Stichproben des Audiosignals x werden pro Sekun- de ermittelt: Repräsentiere eine Stichprobe durch 8 oder 16 Bits.
2. Für *Schrittlänge* S und *Fenstergröße* F konkateniere jeweils F aufeinanderfolgende Stichproben zu den Zeitpunkten $t = S, 2 \cdot S, 3 \cdot S, \ldots$.
   - Wähle $S \ll F$.
   - Wir erhalten überlappende Teilfolgen $(x_t : t)$.
3. Wende eine diskrete Fourier Transformation an: Das Ergebnis dieser Vorverarbeitung sind die Folgen x'_t der Fourierkoeffizienten

Word Error Rate (WER)

Die Word-Error-Rate **WER**(V, T) ist ein Fehlermaß für die vermutete Transkription V im Vergleich zur tatsächlichen Transkription T.

Genauer: Man betrachtet die **Levenshtein-Distanz** $L(V, T)$, wobei

$$L(V, T) \quad := \quad \text{Die minimale Anzahl von Einfüge-, Lösch- und Ersetzungsoperationen, um } V \text{ in } T \text{ zu überführen.}$$

Besteht die Ziel-Transkription T aus n Worten, dann setzt man

$$\text{WER}(V, T) := \frac{L(V, T)}{n}.$$

- Google Home soll eine WER von 4,9% erreichen: Höchstens jedes 20te gesprochene Wort ist zu verbessern.
- IBM vermutet eine WER von ca. 5.1% für menschliche Zuhörer.

Der Aufbau eines traditionellen ASR-Systems

1. Das **akustische Modell** bestimmt die plausibelsten (Phonem- oder Buchstaben-)Folgen.
   - Das *Phonem* (oder der Laut)ist die kleinste bedeutungsunterscheidende Einheit des *Lautsystems* einer Sprache.
   - Das *Graphem* (oder der Buchstabe) ist die kleinste bedeutungsunterscheidende Einheit des *Schriftsystems* einer Sprache.
2. Das **Sprachenmodell** bestimmt eine für die Ausgabe π des akustischen Modell plausibelste Wortfolge.
   - Zum Beispiel werden im *Trigram-Sprachenmodell* die Wahrscheinlichkeiten

     $$\text{prob}[\, w_3 \mid w_1, w_2 \,]$$

     festgehalten, nämlich dass w_3 nach den Worten w_1, w_2 auftritt.

Das akustische Modell ist das „Herz“ eines ASR-Systems.
Mgl. gibt es weitere Modelle, wie etwa ein **Modell** für die **Aussprache**.

Wie lernen ASR-Systeme?

(a) Man arbeitet mit rekurrenten neuronalen Netzen und Backpropagation-Through-Time.

(b) Wegen des Problems *verschwindender* oder *explodierender* Gradienten werden **LSTM-Blöcke** eingeführt.

- LSTM steht für Long-Short-Term-Memory, ein sich langsam änderndes Kurzzeitgedächtnis, dessen aktuell relevanter Teil zum Arbeitsgedächtnis werden soll.
- Ein LSTM-Block soll entscheiden, welche Erinnerungen aufgebaut und welche zu löschen/verändern sind.
  - ⋆ Vergleiche mit den Skip-Verbindungen eines ResNets.

RNNs für die Spracherkennung bestehen aus „konventionellen“ Gattern und LSTM-Blöcken. RNNs ohne sorgfältig ausgewählte

„Datenstrukturen für die Erinnerung“

haben eine deutlich schlechtere Performanz.

Long-Short-Term-Memory

LSTM-Blöcke: Die einzelnen Gatter (1/3)

Ein LSTM-Block besteht aus einer Anzahl N von

- *Input-Gattern*, die dem Systemzustand neue Inhalte hinzufügen,
- *Forget-Gattern*, die den Einfluß des alten Systemzustand einschränken dürfen,
- rekurrenten *Speicher-Gattern*, die den Systemzustand mit Hilfe von Input- und Forget-Gattern aktualisieren sowie
- *Output-Gatter*, die über die Weitergabe des neuen Systemzustands entscheiden.
  - Die Output-Gatter schützen andere Zellen vor störenden Eingaben.

(a) Zum Zeitpunkt t hängt jedes Gatter ab vom *Eingabevektor* x'_t und dem *Systemvektor* h_{t-1}. (x'_t und h_t sind Vektoren gleicher Länge.)

(b) Für jedes Gatter lerne Gewichte zu Eingabe- und Systemvektor.

LSTM-Blöcke: Die einzelnen Gatter (2/3)

- Für Vektoren $u, v \in \mathbb{R}^m$ ist $u \otimes v = (u_i \cdot v_i : i \leqslant m)$ das komponentenweise Produkt der Vektoren u und v.
- σ ist das Standard-Sigmoid, tanh der hyperbolische Tangens.

$$\begin{aligned}
\text{input}^{(t)}(x'_t, h_{t-1}) &:= \sigma(U_{\text{input}} \cdot x'_t + W_{\text{input}} \cdot h_{t-1} + \text{bias}_{\text{input}}) \\
\text{forget}^{(t)}(x'_t, h_{t-1}) &:= \sigma(U_{\text{forget}} \cdot x'_t + W_{\text{forget}} \cdot h_{t-1} + \text{bias}_{\text{forget}}) \\
\text{speicher}^{(t)}(x'_t, h_{t,j}) &:= \text{forget}^{(t)}(x'_t, h_{t-1}) \otimes \text{speicher}^{(t-1)}(x'_{t-1}, h_{t-2}) \\
&+\text{input}^{(t)}(x'_t, h_{t-1}) \otimes \tanh(U_{\text{speicher}} \cdot x_t + W_{\text{speicher}} \cdot h_{t-1} + \text{bias}_{\text{speicher}})
\end{aligned}$$

$$\text{output}^{(t)}(x'_t, h_{t-1}) := \sigma(U_{\text{output}} \cdot x'_t + W_{\text{output}} \cdot h_{t-1} + \text{bias}_{\text{output}}).$$

$\text{output}^{(t)}$ ist der neue Fokus, $\text{speicher}^{(t)}$ das neue Gedächtnis, h_{t-1} das alte Arbeitsgedächtnis. Und das neue Arbeitsgedächntnis?

Lerne alle U, W-Matrizen und bias-Vektoren.

LSTM-Blöcke: Die einzelnen Gatter (3/3)

Der LSTM-Block bestimmt den neuen Systemzustand

$$h_t := \text{output}^{(t)}(x_t, h_{t-1}) \otimes \tanh\left(\text{speicher}^{(t)}(x_t, h_{t-1})\right).$$

h_t ist das neue „Arbeitsgedächtnis“,

Backpropagation für LSTM-Blöcke (1/2)

Vermeide einen verschwindenden oder explodierenden Gradienten.

Was tun, wenn die Rückwärtsphase von Backpropagation einen LSTM-Block mit Ausgabeknoten ℓ erreicht hat?

(a) Die Ausgabe von ℓ ist eine Komponente von

$$h_t := \text{output}^{(t)}(x_t, h_{t-1}) \cdot \tanh\big(\text{speicher}^{(t)}(x_t, h_{t-1})\big).$$

(b) Das Fehlersignal von Knoten ℓ ist

$$\delta_\ell = \sum_{(\ell,u) \text{ ist eine Kante}} \delta_u \cdot \gamma_u'(\text{ summe}_u) \cdot w_{j,u}.$$

(c) Der Ansatz: Bis auf δ_ℓ darf kein Fehlersignal den LSTM-Block verlassen (und dort Schaden anrichten).

Backpropagation für LSTM-Blöcke (2/2)

Eine Version von **Truncated Backpropagation** für LSTM-Blöcke:

- Knoten außerhalb eines LSTM-Blocks erhalten nur Zugriff auf die Fehlersignale δ_ℓ der „Ausgabeknoten“ ℓ.
- Output- und Speicher-Gatter „reichen das Fehlersignal δ_ℓ an die Gatter des Blocks weiter, die es benötigen“.
- Setze das Fehlersignal eines Input-, Forget- oder Speicher-Gatters auf Null, wenn es sein Gatter verlässt.
  - Vor Nullsetzung aktualisiere U, W- oder bias-Gewichte des Gatters.
- Das Speicher-Gatter ist rekurrent!
  - Das Fehlersignal kann im Speicher-Gatter verweilen und bildet ein „Fehlerkarussel“.
  - Unbeschränkt lange, rekurrente Weiterverfolgung des Fehlers ist im *Fehlerkarussel* möglich $\overset{?}{\Longrightarrow}$ gezieltes Erinnern erlernbar.

GRUs: Gated Recurrent Units

Ein GRU-Block besteht aus einem Update- und einem Reset-Gatter.

GRUs unterscheiden nicht zwischen Systemzuständen und Ausgaben.

1. Das **Reset-Gatter** berechnet

$$\text{reset}_t(x_t, h_{t-1}) := \sigma(U_{\text{reset}} \cdot x_t + W_{\text{reset}} \cdot h_{t-1} + \text{bias}_{\text{reset}})$$

$$h_t^* := \tanh(U_{\text{gru}} \cdot x_t + W_{\text{gru}} \cdot \Big(\text{reset}(x_t, h_{t-1}) \otimes h_{t-1}\Big) + \text{bias}_{\text{gru}}).$$

2. Das **Update-Gatter** berechnet

$$\text{update}_t(x_t, h_{t-1}) := \sigma(U_{\text{update}} \cdot x_t + W_{\text{update}} \cdot h_{t-1} + \text{bias}_{\text{update}})$$

$$h_t := \text{update}_t(x_t, h_{t-1}) \otimes h_{t-1} + \Big(1 - \text{update}_t(x_t, h_{t-1})\Big) \otimes h_t^*.$$

LSTMs sind ausdrucksstärker, GRUs kompakter (weniger Gewichte!).

Das ASR-System von Google Home

Vorbereitungen

Wir beschreiben das akustische Modell, das „Herz des Systems“.

(a) Sei $x = (x_i : i)$ der Vektor der Audiosignale.

(b) Für Schrittlänge S und Fenstergröße F zerlege x überlappend in

$$x_t := (x_{t \cdot S}, x_{t \cdot S+1}, \ldots, x_{t \cdot S+F-1}).$$

- Schrittlänge ist 10 ms und Fenstergröße ist 32 ms.
- $\Longrightarrow$ die Vektoren x_t überlappen stark.

(c) Führe eine DFT auf x_t aus:

$$x'_t := \text{Fourier}_F \cdot x_t.$$

- Für $u \in \mathbb{R}^N$ ist $u' := \text{Fourier}_N \cdot u$ mit der DFT-Matrix

$$\text{Fourier}_N = \left(\frac{\omega^{j \cdot k}}{\sqrt{N}} \right)_{0 \leqslant j,k < N}$$

und der primitiven Nten Einheitswurzel $\omega := \exp^{-2\pi \cdot i/N}$.

(d) Unterteile x'_t in „Frequenzblöcke“ $x'_{t,0}, \ldots, x'_{t,L}$.

Ein (Zeit $\times$ Frequenz)-Gitter aus LSTMs (1/3)

x wird in die Vektoren $x'_t = \text{Fourier}_N \cdot x_t$ der Fourier-Koeffizienten überführt.
Unterteile x'_t in Frequenzblöcke $x'_{t,k}$. (t ist die Zeit- und k die Frequenzkomponente.)

(a) Die Vektoren $x'_{t,k}$ werden von einem 2D-Gitter $\mathcal{G}_{\text{Zeit, Frequenz}}$ von LSTMs verarbeitet. Jeder Gitterpunkt (t, k) besteht aus zwei LSTMs:

- Einer Zeit-LSTM $\text{Zeit}_{t,k}$ und einer Frequenz-LSTM $\text{Frequenz}_{t,k}$.
- Beide LSTMs erhalten die Eingabe $x'_{t,k}$ der Fourierkoeffizienten.

(b) Die Zeit-LSTMs werden als ein unidirektionales RNN aufgefasst, um Spracherkennung in Echtzeit liefern zu können.

(c) Die Frequenz-LSTMs bilden ein bidirektionales RNN Frequenz-aufsteigende und Frequenz-absteigende Berechnungen sind als gleichwertig anzusehen.

- $\text{Frequenz}_{t,k}$ besitzt eine Rückwärtskopie $\text{Frequenz}^R_{t,k}$,
- die Ergebnisse der beiden Kopien werden konkateniert.

Ein (Zeit $\times$ Frequenz)-Gitter aus LSTMs (2/3)

Kommunikation zwischen Frequenz- und Zeit-LSTMs:

- $\text{Zeit}_{t,k}$ erhält $h^{\text{Zeit}}_{t-1,k}$ von $\text{Zeit}_{t-1,k}$ und $h^{\text{Frequenz}}_{t,k-1}$ von $\text{Frequenz}_{t,k-1}$.
- *Vorwärtsberechnung*: $\text{Frequenz}_{t,k}$ erhält $h^{\text{Zeit}}_{t-1,k}$ von $\text{Zeit}_{t-1,k}$ und $h^{\text{Frequenz}}_{t,k-1}$ von $\text{Frequenz}_{t,k-1}$.
- *Rückwärtsberechnung*: $\text{Frequenz}^R_{t,k}$ erhält $h^{\text{Zeit}}_{t-1,k}$ von $\text{Zeit}_{t-1,k}$ und $h^{\text{Frequenz}}_{t,k+1}$ von $\text{Frequenz}^R_{t,k+1}$.

Ein (Zeit $\times$ Frequenz)-Gitter aus LSTMs (3/3)

Alle LSTMs bestehen aus Input-, Forget-, Output- und Speichergatter.

Lerne für jedes Gatter G von Zeit$_{t,k}$ die folgenden Gewichtsmatrizen:

- $U_{G,t}$ für Kanten, die G mit der externen Eingabe $x'_{t,k}$ verbinden,
- $W^{\text{Zeit}}_{G,t}$ für Kanten, die G mit den internen Eingaben $h^{\text{Zeit}}_{t-1,k}$ verbinden bzw. $W^{\text{Frequenz}}_{G,k}$ für die interne Eingabe $h^{\text{Frequenz}}_{t,k-1}$.

Die zu lernenden Gewichtsmatrizen für die Vorwärts- und Rückwärtskopien von Frequenz-LSTMs sind analog definiert.

Google Home:
Die Architektur des akustischen Modells

1. Das *Audiosignal* x wird in Frequenzblöcke $x'_{t,k}$ überführt.
2. Das *Grid-LSTM* verarbeitet die Folge $(x'_{t,k} : t, k)$.
   - Für jedes t werden die Ergebnisse der Zeit- und Frequenz-LSTMs zu $(t, 0), (t, 1), \ldots, (t, L)$ konkateniert.
   - Eine lineare *Dimensionsreduktion* mit Ergebnis $(x''_t : t)$ wird ausgeführt.
3. $(x''_t : t)$ wird von vier aufeinanderfolgenden Schichten von jeweils 1024 vollständig verbundenen LSTMs weiter verarbeitet.
4. Eine vollständig verbundene Schicht von 1024 Neuronen gefolgt von einer Softmax-Schicht schließt die Berechnung ab.

Google Home: Backpropagation-Through-Time

1. Der *Log-Loss* wird benutzt.
2. *Truncated Backpropagation-Through-Time* wird eingesetzt, um das „Gradientenproblem“ zu vermeiden:
   - Das rekurrente Netz wird 20 mal abgewickelt und, parallel, für jeweils k_1 Beispiele ausgewertet.
   - Nach jeder Auswertungsphase werden k_2 Iterationen der Rückwärtsphase von BPTT ausgeführt.
3. Die *Beispielmenge* besteht aus gesprochenem, teilweise verrauschtem Text (ca. 18.000 Stunden) sowie Klartext (ca. 4.000 Stunden).

Backpropagation-Through-Time ist sehr aufwändig.

LSTMs: Abschließende Bemerkungen

(a) Google Home benutzt LSTMs, Baidu GRUs.

(b) Aufmerksamkeitsmodelle verfolgen ein ähnliches Ziel wie LSTM-Blöcke: Lerne das Erinnern an relevante Teile der Eingabe.

(c) Die Skip-Verbindungen von ResNet bilden eine elementare Hardware-Lösung.

(d) Das Lernen von RNNs (und damit auch von LSTMs) ist sehr aufwändig.

Connectionist Temporal Classification (CTC)

Connectionist Temporal Classification (CTC) (1/3)

Ist ein ASR-System „aus einem Guß“, also (fast) ohne Vorverarbeitung und ohne ein nachfolgendes Sprachenmodell, möglich?

Das akustische Model überführt die Eingabe x in eine Folge $A \in (\Sigma \cup \{\_\})^*$ von Phonemen (oder Graphemen). (Σ ist das Phonemalphabet, _ das Pausensymbol).

- Unterschiedliche Sprechgeschwindigkeit $\Longrightarrow$ Laute treten wiederholt in A auf und Pausen haben unterschiedliche Länge.
- Hintergrundrauschen erschwert das Verständnis.

Zu x gibt es viele mögliche Phonemfolgen oder **Aligments** $A \in (\Sigma \cup \{\_\})^*$.

Was *bedeutet* ein Alignment? Streiche wiederholt auftretendes Vorkommen (bis auf ein einziges Vorkommen) und entferne dann das Pausensymbol.

$$x \mapsto A \mapsto y = \text{Streichen}(A).$$

Für $A_1 = \text{HHH\_E\_\_L\_L\_O}$ und $A_2 = \text{HH\_\_E\_L\_LL\_O}$ ist

$$\text{Streichen}(A_1) = \text{Streichen}(A_2) = \text{HELLO}$$

Sei $x = (x_t : 1 \leqslant t \leqslant T)$ die Eingabe. Ein neuronales Netz Netz_w bestimmt zu jedem Zeitpunkt t und für jedes $k \in \Sigma \cup \{\_\}$

$$q_t^k := \text{Wahrscheinlichkeit, dass } x_t \text{ die Bedeutung } k \text{ hat.}$$

Die W.keit eines Aligments $A \in (\Sigma \cup \{\_\})^T$ wird definiert durch

$$p_w(A \mid x) := \prod_{t=1}^{T} q_t^{A_t}$$

und die Wahrscheinlichkeit für die Transkription y ist

$$p_w(y \mid x) := \sum_{A,\ \text{Streichen}(A)=y} p_w(A \mid x).$$

Aber die Zuweisung von Buchstaben zu aufeinanderfolgenden Lauten entspricht doch nicht **unabhängigen** Experimenten!

Connectionist Temporal Classification (CTC) (3/3)

Die fragwürdige Annahme der Unabhängigeit wird später aufgehoben.

- Die Aufgabe von CTC: Bestimme die höchstwahrscheinliche(n) Transkription(en) zu Eingabe x.
- Das Verfahren:
  - Fordere eine Beispielmenge $S = \{ (x_i, y_i) : i \}$ an.
  - Für jedes i wende Backpropagation auf (x_i, y_i) an.
    - ★ Was genau soll Backpropagation zurück propagieren?
    - ★ Vergleiche die Ausgabe von Netz$_w$ mit der Zielklassifikation! Aber Netz$_w$ berechnet doch nur die Verteilungen q_t^k für jedes t?

Wie lernt CTC?

(a) Minimiere den *Log-Loss*

$$\ell(w, x, y) := \sum_{(x', y) \in S} -\ln\left(p_w(y \mid x)\right).$$

- Wie bestimmt man $p_w(y \mid x)$?

(b) Die *Netz-Architektur*: Netz_w ist ein bidirektionales RNN mit 2 Schichten aus jeweils vollständig miteinander verbundenen LSTM-Blöcken.

- Eingabe zum Zeitpunkt t ist der Vektor x'_t der Fourier-Koeffizienten.
- Für jedes Symbol in $\Sigma \cup \{\_\}$ gibt es ein Softmax-Ausgabeneuron, das auf die Ergebnisse der beiden Schichten zugreift.

(c) Für die Durchführung des *stochastischen Gradientenabstiegs* sind die partiellen Ableitungen $\frac{\partial p_w(y \mid x)}{\partial e}$ zu bestimmen.

- Wie geht das?

(d) Wenn Gewichte gelernt sind: Wie berechnet man *gute Transkriptionen*?

Wie bestimmt man $p_w(y \mid x)$?

Das Paar (x, y) von gesprochenem Text und Transkription ist gegegeben.

1. Für alle s, t mit $1 \leqslant s \leqslant |y|$ und $1 \leqslant t \leqslant T$ bestimme die Wahrscheinlichkeit $\alpha_t(s)$, dass $y_1 \cdots y_s$ zur Zeit t gesprochen wurde:

$$\alpha_t(s) := \sum_{\substack{A=(A_1,\ldots,A_t),\\ \text{Streichen}(A)=(y_1,\ldots,y_s)}} \prod_{i=1}^{t} q_i^{A_i}.$$

   $p_w(y \mid x) = \alpha_T(|y|)$, aber wie bestimmt man $\alpha_t(s)$?
2. Ebenso bestimme die Wahrscheinlichkeit $\beta_t(s)$, dass $y_s \cdots y_{|y|}$ ab Zeit t gesprochen wurde:

$$\beta_t(s) := \sum_{\substack{A=(A_t,\ldots,A_T),\\ \text{Streichen}(A)=(y_s,\ldots,y_{y|})}} \prod_{i=t}^{T} q_i^{A_i}.$$

Bestimmung der partiellen Ableitungen $\frac{\partial p_w(y \mid x)}{\partial e}$ (1/3)

Bestimme

$$r_t^k(s) := \text{prob}_A \quad [\quad \text{Alignment } A \text{ erzeugt } y \text{ und } A_t = k \text{ und } A \text{ erzeugt } y_1 \cdots y_s \text{ zur Zeit } t \ ]$$

Betrachte die Ereignisse

$$\text{präfix}_t^k(A, s) := \text{Streichen}(A_1 \cdots A_t) = y_1 \cdots y_s, \quad A_t = k$$
$$\text{suffix}_t^k(A, s) := \text{Streichen}(A_t \cdots A_T) = y_s \cdots y_{|y|}, \quad A_t = k$$
$$\Longrightarrow r_t^k(s) = \frac{\text{prob}[\,\exists A: \ \text{präfix}_t^k(A, s)\,] \cdot \text{prob}[\,\exists A: \ \text{suffix}_t^k(A, s)\,]}{q_t^k}$$
$$\Longrightarrow p_w(y \mid x) = r_t^-(s) + r_t^k(s)$$

Aber wie bestimmt man r_t^k? Führe α_t^k, β_t^k ein.

Bestimmung der partiellen Ableitungen $\frac{\partial p_w(y \mid x)}{\partial e}$ (2/3)

Es ist $p_w(y \mid x) = r_t^-(s) + r_t^k(s)$.

$$\Longrightarrow \frac{\partial \ell(w, x, y)}{\partial e} = -\frac{\partial \ln(p_w(y \mid x))}{\partial e} = -\frac{1}{p_w(y \mid x)} \cdot \frac{\partial p_w(y \mid x)}{\partial e}$$

$$= -\frac{1}{p_w(y \mid x)} \cdot \frac{\partial\, r_t^-(s) + \partial\, r_t^k(s)}{\partial e}$$

Und wie berechnet man $\frac{\partial r_t^k(s)}{\partial e}$?

Bestimmung der partiellen Ableitungen $\frac{\partial p_w(y \mid x)}{\partial e}$ (3/3)

1. Mit der Kettenregel folgt für jedes $k \in \Sigma \cup \{\_\}$

$$\frac{\partial r_t^k(s)}{\partial e} = \frac{\partial r_t^k(s)}{\partial q_t^k} \cdot \frac{\partial q_t^k}{\partial e}.$$

2. Die Berechnung von $\frac{\partial q_t^k}{\partial e}$ gelingt mit Backpropagation, denn q_t^k wird in Netz_w von dem Ausgabe-Neuron für k berechnet.

3. Die Berechnung von $\frac{\partial r_t^k(s)}{\partial q_t^k}$ ist effizient möglich: Es ist

$$\begin{aligned} r_t^k(s) &= \frac{\text{prob}[\,\exists A:\ \text{präfix}_t^k(A,s)\,] \cdot \text{prob}[\,\exists A:\ \text{suffix}_t^k(A,s)\,]}{q_t^{y_s}} \\ &= \frac{\alpha_t^k \cdot \beta_t^k}{q_t^{y_s}}. \end{aligned}$$

Wende die Produkt-Regel an.

Bestimme eine gute Transkription: BeamSearch

BeamSearch mit Kapazität K ist eine Breitensuche, die einem Stack der Größe höchstens K benutzt.

Wiederhole für alle Zeitpunkte t:

- Expandiere die höchstens K gespeicherten Alignment-Präfixe um jeweils einen Buchstaben.
- Danach schränke wieder auf die *besten* K Alignment-Präfixe ein.

Zuletzt bestimme die Transkriptionen zu den gefundenen Alignments.

Eine bessere, aber rechenintensivere Alternative: Führe BeamSearch nicht mit Alignments sondern mit Präfixen von Transkriptionen durch:

- Wenn Beam Search aktuell die Präfixe $p_1, \ldots, p_K$ abspeichert, dann betrachte alle 1-Schritt-Erweiterungen $p_{i,j}$ von p_i.
- Wähle die K *höchstwahrscheinlichen* Präfixe aus.

RNN-Transducer

Korrelationen erlaubt!

Ein RNN-Transducer ist eine CTC-Variante, die Korrelationen zwischen aufeinanderfolgenden Buchstaben modellieren kann.

Ein RNN-Transducer besteht aus dem ***akustischen Netz*** A_v und dem ***linguistischen Netz*** L_w. Die Gewichtsvektoren v bzw. w bestehen für das jeweilige Netz aus den Gewichten zwischen Eingabeneuronen und versteckten Neuronen, versteckten Neuronen sowie versteckten Neuronen und Ausgabeneuronen.

1. A_v hat dieselbe Architektur wie Netz$_w$, wird aber anders trainiert.
   - A_v erhält x'_t als Eingabe und gibt die Folge $f_t \in \mathbb{R}^{|\Sigma|+1}$ aus.
     - f_t^k ist die (vermutete) W.keit von Symbol $k \in \Sigma \cup \{\_\}$ zur Zeit t.
     - f_t hängt von Vorwärts- und Rückwärts-Systemzuständen ab.
2. L_w besteht aus einer Schicht vollständig verbundener LSTM-Blöcke.
   - L_w erhält die Transkription y Buchstabe für Buchstabe und gibt für Position u die Folge $g_u \in \mathbb{R}^{|\Sigma|+1}$ aus.
     - g_u^k ist die (vermutete) W.keit von Symbol $k \in \Sigma \cup \{\_\}$ in Position u.
     - g_u hängt von dem Systemzustand in Position u ab.

Wie lernt ein RNN-Transducer?

Trainiere A_v und L_w gemeinsam, um akustische und linguistische Information vermischen zu können: Minimiere den Log-Loss

$$-\ln p^*_{v,w}(\, y \mid x' \,).$$

Was ist $p^*_{v,w}(\, y \mid x' \,)$? Definiere

$$q^k_{t,u} := \frac{f^k_t \cdot g^k_u}{\sum_{k' \in \Sigma \cup \{\_\}} f^{k'}_t \cdot g^{k'}_u}$$

Wenn $q^k_{t,u}$ groß ist, gibt es starke „akustische **und** linguistische Gründe“ für k.

In der Definition von $p^*_{v,w}(\, y \mid x' \,)$ wird deshalb $q^*_{t,u}$ und damit die Verbindung akustischer und linguistischer Gründe eine zentrale Rolle spielen.

Definition von $p^*_{V,w}(\, y \mid x' \,)$ (1/2)

Im Gitter $\mathcal{G}_{T,U} := \{1, \ldots, T\} \times \{0, 1, \ldots, U\}$ wird ein Alignment als ein Weg modelliert, der im Knoten $(1, 0)$ beginnt und im Knoten (T, U) endet.

- Alle horizontalen Kante $(t, u) \to (t + 1, u)$ laufen von „links nach rechts“, alle vertikalen Kanten $(t, u) \to (t, u + 1)$ laufen von „unten nach oben“.
- Eine horizontale Kante $(t, u) \to (t + 1, u)$ entspricht der Aufnahme des Pausensymbols und erhält die Wahrscheinlichkeit $q^{-}_{t,u}$.
- Eine vertikale Kante $(t, u) \to (t, u + 1)$ entspricht der Aufnahme des nächsten Buchstabens y_{u+1} und erhält die Wahrscheinlichkeit $q^{y_{u+1}}_{t,u}$.

Definition von $p^*_{v,w}(\, y \mid x' \,)$ (2/2)

(a) Ein ***Weg*** W in $\mathcal{G}_{T,U}$ beginnt im Knoten $(1, 0)$, endet im Knoten $(T + 1, U)$ und benutzt nur horizontale oder vertikale Kanten.

(b) Die Wahrscheinlichkeit

$$p(\, W \mid x', y \,)$$

eines Weges W ist das Produkt der Wahrscheinlichkeiten seiner Kanten.

(c) Setze

$$p^*_{v,w}(\, y \mid x' \,) := \sum_{W\text{: W ist ein Weg in } \mathcal{G}_{T,U}} p(\, W \mid x', y \,).$$

RNN Transducer: Zusammenfassung

1. Backpropagation muss $-\nabla \ln p^*_{v,w}(y \mid x')$ berechnen.
   - Aber wie? Siehe Skript.
2. Beachte: Das linguistische Netz wird nur für das Training benutzt!
3. Bessere(?!) Definition von $q^k_{t,u}$: Statt des Produkts benutze „tanh“-Gatter mit Zugriff auf die versteckten Neuronen der beiden Netze.
   - Größere Ausdruckskraft.
4. CTC und RNN-Transducer schneiden ähnlich ab, aber der RNN-Transducer braucht weniger Unterstützung durch das Sprachenmodell.
   - Baidu hat mit einem ASR-System begonnen, das auf CTC basiert.
   - Das neue ASR-System basiert auf dem RNN-Transducer.

Baidu: Deep Speech 2 und Deep Speech 3

Deep Speech 2

Als „ganzheitliche“ Systeme besitzen weder **Deep Speech 2** noch **Deep Speech 3** ein akustisches Modell oder ein Modelle für die Aussprache.

Deep Speech 2 benutzt

- bis zu drei Faltungsschichten,
- gefolgt von bis zu 7 bidirektionalen GRU-Schichten und
- darauffolgend eine oder mehrere vollständig verbundene Schichten.

Ein Softmax-Gatter gibt eine Verteilung über Buchstaben aus.

Größenordnungen:

- Deep Speech 2 arbeitet mit ungefähr 100 Millionen Parametern.
- Für Englisch wird mit ca. 12.000 Stunden an Sprachdaten und für Mandarin mit ca. 9.400 Stunden gearbeitet. Die Leistung:
  - *bei klarer Sprache*: Kompetitiv mit menschlichen Zuhörern.
  - Schlechter bei Akzenten bzw. bei verrauschter Spracheingabe.

Neural Machine Translation

Statistische Maschinelle Übersetzung (SMT)

Die statistische maschinelle Übersetzung hat bis vor wenigen Jahren dominiert.

Für einen gegebenen *Satz* e der Ausgangsssprache sind Übersetzungen f in der Zielsprache mit großer Wahrscheinlichkeit

$$\text{prob}[\, f \mid e \,]$$

zu bestimmen.

Die „Kunst“ besteht darin, gute Approximationen von $\text{prob}[\, f \mid e \,]$ mit Hilfe eines

entsprechend großen Textkorpus und ausgefeilter Sprachmodelle

zu bestimmen.

BLEU: Bilingual Evaluation Understudy

BLEU simuliert die Bewertung eines automatischen Übersetzungssystems durch einen menschlichen Experten.

- Sätze T der Ausgangssprache werden durch das Übersetzungssystem in Sätze T' der Zielsprache übersetzt.
- Der Grad der Übereinstimmung zwischen T' und professionellen Übersetzungen $T_1, \ldots, T_m$ wird gemessen:
  - Wie viele n-Gramme kommen sowohl in T' wie auch in den verschiedenen T_i vor?
  - BLEU-Noten im Intervall $[0, 1]$, (bzw. Prozentsätze) werden vergeben.
- BLEU-Werte von 50% oder mehr sind hervorragend, Werte von 15% oder weniger deuten auf schlechte Übersetzungen hin.

BLEU-Werte sind weder für eine Bewertung der Verständlichkeit oder der grammatikalischen Korrektheit der Übersetzung geeignet.

Neural Maschine Translation (NMT)

Im Folgenden ist x ein Satz der Ausgangssprache, dargestellt als Folge

$$x = (x_t \ : \ t)$$

seiner Wörter und Interpunktionszeichen.

Die Architektur eines NMT-Systems besteht aus

- einem **Encoder**, der aus Eingabe x Eigenschaften

$$h = (\ h_t \ : \ t\ )$$

  Wort für Wort ableitet,
- einem **Decoder**, der die Übersetzung produziert
- und dem **Aufmerksamkeitsmodell**, das versucht den relevanten Teil von x für die aktuelle Stelle der Übersetzung zu bestimmen.

Das Aufmerksamkeitsmodell (1/2)

Die (e, d, f, g)-Architektur: Die für die Berechnung der Funktionen d, e, f und g notwendigen Gewichte sind zu lernen.

1. Der Encoder ist ein bidirektionales RNN mit Eingabe $x = (\, x_t \,:\, t \,)$.
   In Iteration t wird die „*Annotation*“ (oder Systemzustand) h_t berechnet mit

   $$h_t := e(x_t, h_{t-1}).$$

2. Der Decoder ist ein unidirektionales RNN mit den Systemzuständen

   $$r_i := d(r_{i-1}, y_{i-1}, s_i).$$

   Der „*Fokus*“ s_i ist ein vom Aufmerksamkeitsmodell berechneter „Mittelwert“ der Annotationen. Die Verteilung

   $$p(y_i \mid y_1, \ldots, y_{i-1}, x) := f(r_i, y_{i-1}, s_i)$$

   gibt Kandidaten für das $i$te Wort y_i der Übersetzung an.

Das Aufmerksamkeitsmodell (2/2)

Das Attention-Netz ist ein Feedforward-Netzwerk, das den Fokus

$$s_i := \sum_t \alpha_{i,t} h_t$$

für jedes t berechnet, wobei $\alpha_{i,*} :=$ softmax$(\beta_{i,*})$ mit

$$\beta_{i,t} := g(r_{i-1}, h_t)$$

die Relevanz wiedergeben soll, die h_t für y_i hat.

- Die Funktion g bewertet den Zusammenhang (oder Alignierung) zwischen y_{i-1} und x_t.

Die Wunschvorstellung: Der Decoder bestimmt y_i in Abhängigkeit von $y_1, \ldots, y_{i-1}$ und von den Teilen der Eingabe, die am meisten zu s_i beitragen.

Google's Neural Machine Translation (GNMT)

GNMT: Die Architektur

Google's Neural Machine Translation (GNMT) übernimmt die obige Architektur mit Encoder, Decoder und Aufmerksamkeitsmodell.

(a) Für den **Encoder** werden 8 LSTM-Schichten übereinander gesetzt. Jede LSTM-Schicht entspricht einem eindimensionalem Gitter.

- Die unterste LSTM-Schicht ist bidirektional, alle weiteren Schichten sind unidirektional.
- Die Ergebnisse der $(i-1)$ten und $i$ten Schicht werden an LSTM-Blöcke der $(i+1)$ten LSTM-Schicht weitergereicht:
  - ⋆ *Residuale Verbindungen* gegen verschwindende Gradienten.
- Die Ausgaben der obersten Schicht sind Eingaben für das Aufmerksamkeitsmodell.

(b) Auch der **Decoder** setzt sich aus 8 LSTM-Schichten zusammen, wobei wieder LSTM-Schichten eindimensionalen Gittern entsprechen.

- Alle LSTM-Schichten sind unidirektional und verwenden residuale Verbindungen.
- Die oberste Schicht des Decoders bestimmt die Funktionen e und f.

GNMT: Die Loss-Funktion

- Für ein Beispielpaar (x, y^*) wird mit der Loss-Funktion

$$\ell(w, x, y^*) := \mathbb{E}_y[\, p(y \mid x) \cdot \mathrm{BLEU}^*(y, y^*)\,]$$

  gearbeitet:
  - Der Erwartungswert wird über alle Ausgabesätze y bis zu einer vorgegebenen Länge berechnet.
  - Die Wahrscheinlichkeit $p(y \mid x)$ eines Ausgabesatzes wird durch das GNMT-System bestimmt.
- $\mathrm{BLEU}^*(y, y^*)$ misst BLEU-ähnlichen Abstandswert zwischen y und y^*.
  - Nicht nur perfekte Übersetzungen, sondern auch genügend wahrscheinliche vernünftige Übersetzungen werden belohnt.
  - Der Erwartungswert wird mit Beam-Search approximativ bestimmt.

GNMT: Die Größenordnungen

- Die Architektur:
  - Alle LSTM-Schichten bestehen aus 1024 LSTM-Blöcken.
  - Das Aufmerksamkeitsmodell ist ein Feedforward-Modell mit einer versteckten Schicht von 1024 Gattern.
- Die Beispiele:
  - Für (Englisch $\rightarrow$ Französisch) werden 36 Millionen Satzpaare, für (Englisch $\rightarrow$ Deutsch) 5 Millionen Satzpaare im Training präsentiert.
  - Dem stehen 255 Millionen Parameter gegenüber.
- Die BLEU-Werte:
  - (Englisch $\rightarrow$ Französisch) hat einen BLEU-Wert von 38,95%, (Englisch $\rightarrow$ Deutsch) einen BLEU-Wert von 24,17%.
  - Verbesserungen um bis zu **60**% gegenüber Google's altem Produktionssystem.

Zero-Shot-Übersetzung

Füge dem Ausgangssatz ein Kürzel für die Zielsprache hinzu.

- Trainiere nacheinander Übersetzungen von U nach V und von V nach W *auf derselben Architektur*.
- Ohne weiteres Training beherrscht das System Übersetzungen von U nach W, allerdings mit deutlich schlechteren BLEU-Werten.
  - Nach zusätzlichem Training für $U \rightarrow W$ mit wenigen Beispielen werden vergleichbare BLEU-Werte zum ausführlichen Training erreicht.

Der Durchschnitt von Halbräumen

Negative Resultate (1/3)

Für einen Gewichtsvektor $w \in \mathbb{R}^n$ ist

$$H_w := \{ x \in \mathbb{R}^n : w^T \cdot x \geqslant 0 \}$$

der von w erzeugte Halbraum.

(a) Der Perzeptron-Algorithmus lernt einen einzigen Halbraum nach höchstens R^2/ρ^2 Gegenbeispielen.

(b) Es gibt Verteilungen, so dass das schwache Konsistenzproblem für den Durchschnitt von zwei Halbräumen NP-vollständig ist,
   - falls Hypothesen einem Durchschnitt von 2 Halbräumen entsprechen müssen.

Gibt es repräsentations-unabhängige Resultate?

Negative Resultate (2/3)

Das Lernen von $k \gg 1$ Halbräumen sollte für jede Hypothesenklasse schwer sein, denn jedes Polytop, also jede von endlich vielen Punkten aufgespannte konvexe Menge, ist ein Durchschnitt von Halbräumen.

Das $f(n)$-**Unique-Shortest-Vector-Problem** (f-uSVP):

- Für Vektoren $v_1, \ldots, v_n \in \mathbb{R}^n$ ist

$$\Gamma(v_1, \ldots, v_n) := \{ \sum_{i=1}^{n} \alpha_i \cdot v_i \ : \ \alpha_1, \ldots, \alpha_n \in \mathbb{Z} \}$$

  das von den Vektoren $v_1, \ldots, v_n$ aufgespannte Gitter.

- Für Vektoren $v_1, \ldots, v_n$, für die es – bis auf Vielfache – genau einen Vektor $v^* \in \Gamma(v_1, \ldots, v_n)$ gibt, der um den Faktor $f(n)$ kürzer als alle anderen von Null verschiedenen Vektoren des Gitters ist,
  ist dieser „mit Abstand“ kürzeste Vektor v^* auszugeben.

Negative Resultate (3/3)

Für jedes $c > 0$ ist der Durchschnitt von $k := n^c$ Halbräumen über $\{0,1\}^n$,

unabhängig von der Wahl der Hypothesenklasse,

nicht effizient PAC-lernbar.

Dieses Ergebnis gilt, falls $f(n) = \Omega(n^{1.5+\epsilon})$ für $\varepsilon > 0$ und falls f-USVP keine effizienten Algorithmen besitzt.

Aber: Für *kleine* Werte von k ist das Lernproblem machbar, falls wir uns auf *einfache* Verteilungen einschränken.

Log-konkave Verteilungen

Eine Wahrscheinlichkeitsverteilung mit Dichtefunktion f ist **log-konkav**, falls

$$f(\alpha \cdot x + (1-\alpha) \cdot y) \geqslant f(x)^{\alpha} \cdot f(y)^{\beta}$$

für eine reelle Zahl α mit $0 < \alpha < 1$ und alle $x, y \in \mathbb{R}$ gilt.

f ist also genau dann log-konkav, wenn $\ln(f)$ konkav ist.

Beispiele log-konkaver Verteilungen:

- Normalverteilung,
- Exponentialverteilung,
- Gleichverteilung.

Es gibt Lernverfahren, die den Durchschnitt von k Halbräumen für log-konkave Verteilungen in Zeit $n^{\mathcal{O}(k)}$ lernen.

Und Backpropagation?

1. Selbst für die Gleichverteilung mit Beispielraum $X = \{-1, 1\}^n$ ist bisher nichts bewiesen worden.
2. Was könnte eine mögliche Situation bei polynomiell vielen Beispielen sein?
   - Erfolgreiches Lernen für $k = \mathcal{O}(\log n)$.
   - Erfolgloses Lernen für $k = \omega(\log n)$.
   - Und für Zwischenwerte von k?