Feedforward-Netzwerke: Die Architektur

Eine **Architektur** $\mathcal{A} = (V, E, f)$ besteht aus

- einem *azyklischen*, *gerichteten* und *geschichteten* Graphen $G = (V, E)$, wobei V in die Schichten $V_0, \ldots, V_T$ unterteilt ist.
  1. Es ist $V_t = \{(t, j) : 1 \leqslant j \leqslant |V_t|\}$.
  2. Alle Kanten (v, w) verlaufen von einem Knoten $v \in V_t$ zu einem Knoten $w \in V_{t+1}$ in der darauf folgenden „Nachbarschicht“.
     - ⋆ V_0 besteht nur aus Quellen und V_T nur aus Senken.
  3. $\mathcal{A}$ verarbeitet Eingabevektoren $x \in \mathbb{R}^n$ mit $n = |V_0|$:
     - ⋆ Die $i$te Komponente x_i wird an die Quelle $(0, i)$ angelegt.
- einer Zuweisung $f(v) = \gamma_v$ von „**Aktivierungsfunktionen**“

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

zu Knoten $v \in V$.

Was ist ein Feedforward-Netzwerk?

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

- der Architektur $\mathcal{A}$
- 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 $u = (t - 1, 1)$ 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

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

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

Ein Konzept c ist zu lernen. Die *typische Lernsituation*:

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

Aktivierungsfunktionen

(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 den 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 **Rectifier**-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 Rectifier!

(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 $X = \{0,1\}^n$.

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

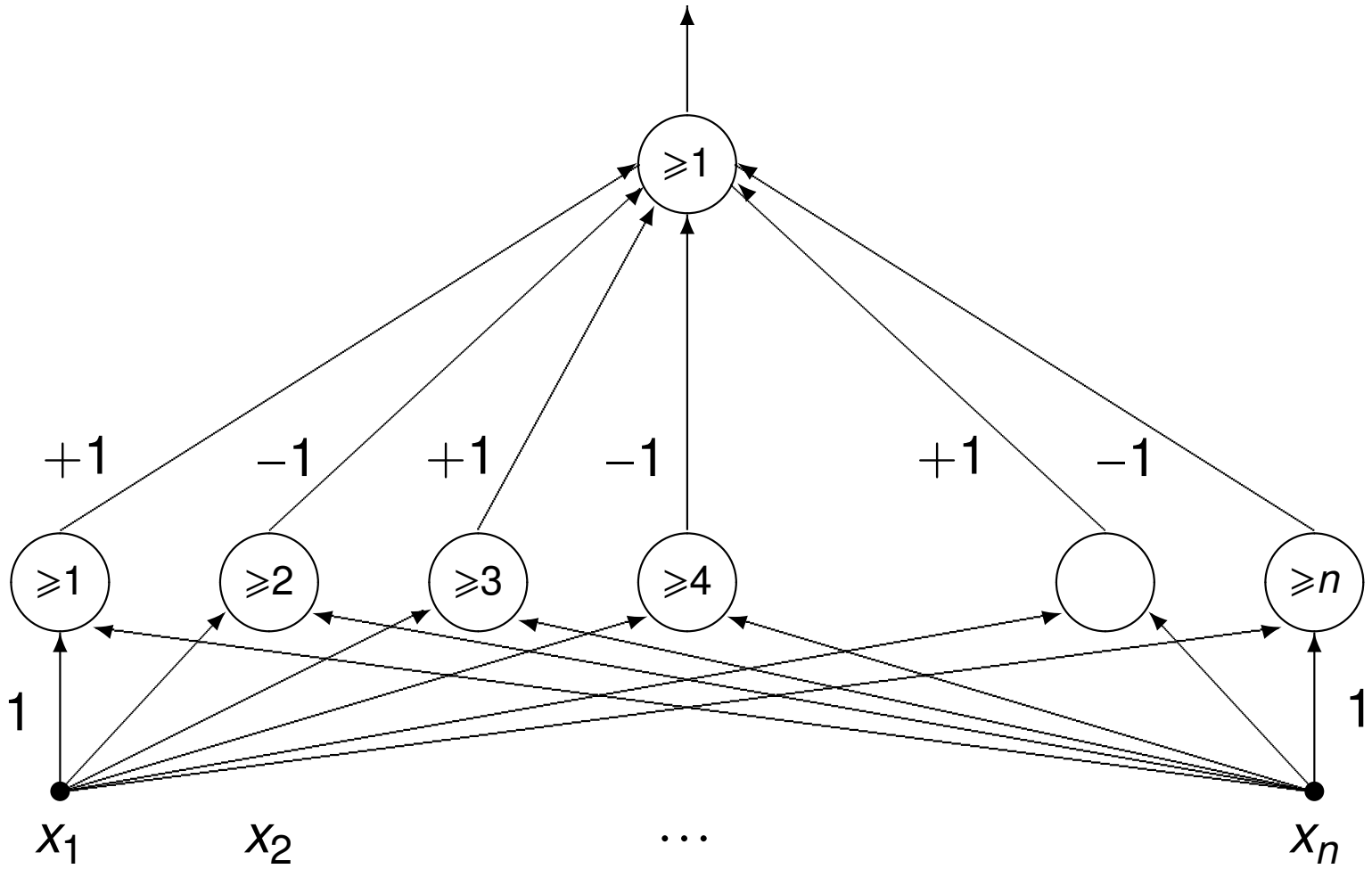
$$x_1 \wedge x_2 \wedge \cdots \wedge x_n \iff \sum_{i=1}^{n} x_i \geqslant n - \frac{1}{2}$$

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

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

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

- 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

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

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

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 $\mathcal{H}_{i+1} = \bigtimes_{v \in V_{i+1}} \mathcal{H}_v \Longrightarrow$

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

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

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

Die VC-Dimension einer Architektur (3/3)

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

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

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 $\Omega(|E|^2)$.
   - Gewichte sind nur mit beschränkter Präzision B darstellbar:
     - ★ Höchstens $B^{|E|}$ verschiedene Hypothesen
     - ★ $\Longrightarrow$ die „praktische“ VC-Dimension ist höchstens $\mathcal{O}(|E|)$.

Die algorithmische Komplexität

Die Komplexität des Lernproblems

- ✓ Das starke Konsistenzproblem für HALBRAUM ist effizient lösbar.
  - Benutze die lineare Programmierung.
- ↯ Das schwache Konsistenzproblem für den Durchschnitt von zwei Halbräumen ist NP-vollständig.
  - Dieses Ergebnis ist repräsentationsabhängig und gilt nur, wenn Hypothesen dem Durchschnitt von zwei Halbräumen entsprechen.
  - Bittere Konsequenz: Feedforward-Netzwerke mit **drei** Threshold-Gattern sind nicht effizient lernbar.
- ↯ Für jede Konstante $c > 0$ und für jede Hypothesenklasse ist der Durchschnitt von n^c Halbräumen über $\{0,1\}^n$ nicht effizient PAC-lernbar, falls das *Shortest-Vector-Problem* keine effizienten Algorithmen besitzt
  - Gegeben sind Vektoren $b_1, \ldots, b_m \in \mathbb{Q}^n$.
  - Gesucht ist ein kürzester ganzzahliger Vektor $v \neq 0$ in dem Gitter

$$G(b_1, \ldots, b_m) = \{\sum_{i=1}^{m} \alpha_i b_i : \alpha_1, \ldots, \alpha_m \in \mathbb{Z}\}.$$

Was sind die Konsequenzen?

Das Lernproblem für neuronale Netzwerke für kleine Netzwerke der Tiefe zwei ist äußerst schwierig:

- Schwierige Lernprobleme
  - wie etwa ein *Reverse Engineering* von Schaltungen der Tiefe zwei

  lassen keine effizienten Lernalgorithmen zu.
- Aber keine Aussagen bei
  - eingeschränkter Architektur,
  - sorgsam ausgewählten Beispielen, ...

  $\Longrightarrow$ Steck Vorwissen in die Wahl von Architektur und Beispielen.

Um Gewichte zu lernen, arbeiten wir mit *Backpropagation*, einer Anwendung der Methode des Gradientenabstiegs.

Gradientenabstieg

Die Minimierung von reellwertigen Funktionen

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 $\eta > 0$

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

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

Stochastischer Gradientenabstieg (SGD)

Die Situation

Löse das Minimierungsproblem

$$\min_{g \in \mathcal{H}} \text{Loss}_D(g).$$

Wir kennen die Trainingsmenge S, aber nicht die Zielfunktion Loss_D.

- Hypothese $h \in \mathcal{H}$ werde durch den Vektor $w \in \mathbb{R}^n$ beschrieben.
- Es ist

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

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

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

    - ★ Die erste Gleichheit folgt aus der Linearität des Erwartungswerts.
    - ★ Hoffentlich approximieren Gradienten bereits für *wenige* Beispiele.

SGD: Das Programm

1. w sei ein Gewichtsvektor, η die Schrittweite und N ein Parameter.
2. Ziehe ein „**Minibatch**“ $(x_1, y_1), \ldots, (x_N, y_N)$ von N klassifizierten Beispielen gemäß Verteilung D.

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

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

durch Parallelverarbeitung und aktualisiere

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

SGD: Minimierung unter Idealbedingungen

Eine **konvexe** Funktion

$$f : \{x \in \mathbb{R}^n : \|x\| \leqslant B\} \to \mathbb{R}$$

ist zu minimieren, wobei die Lipschitz-Schranke gelte:

$$\|f(x) - f(x')\| \leqslant \rho \cdot \|x - x'\|.$$

Die obigen Bedingungen mögen gelten. Wenn mindestens

$$N := \frac{B^2 \rho^2}{\varepsilon^2}$$

Iterationen für die Schrittweite $\eta := \sqrt{\frac{B^2}{\rho^2 \cdot N}}$ ausgeführt werden, dann gilt

$$f(x^{(N)}) \leqslant \min_{x, \|x\| \leqslant B} f(x) + \varepsilon.$$

SGD: Eine Bestandsaufnahme

- ✓ SGD ist ein schnelles Verfahren, das die Bearbeitung großer Beispielmengen erlaubt.
  - Für das Training neuronaler Netzwerke sind manchmal ca. 100 Beispiele pro Parameter des Netzes eine Daumenregel.
  - Beispielzahlen von 100 Millionen und mehr sind nicht unüblich, die größten Netze verbrauchen Beispiele im Milliardenbereich.
- ? Ist „nur“ die approximative Bestimmung eines lokalen Minimum verlangt, erfüllt SGD seinen Zweck.
- ↯ Ist ein globales Minimum approximativ zu bestimmen, dann ist SGD für nicht-konvexe Optimierungsprobleme i. A. überfordert.

SGD für Feedforward-Netzwerke

Welche Verteilung, welche Loss-Funktion?

1. Wie werden Minibatches zusammengestellt?
   - Verschiedene Gesichtspunkte müssen betont werden. In der Schrifterkennung: Translationsinvarianz, Schriftbreite, etc.
2. Welche Loss-Funktion ℓ ist zu wählen?
   - 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: $h_w^{\mathcal{A}}(x)$ bestimmt Wahrscheinlichkeit $p_w(y \mid x)$ für die offengelegte Klassifikation y von Beispiel x.
     - ⋆ Die **Maximum-Likelihood Schätzung**

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

       „belohnt das Modell" w, das die Klassifikationsvorgaben am besten einhält.

Maximum Likelihood für die Loss-Funktion

Der empirische **Maximum-Likelihood-Loss**
(mit der Zielklassifikation y_i für x_i) stimmt überein mit

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

Goodfellow et al.:

> *Die Wahl dieser oder vergleichbarer Loss-Funktionen zählt neben der Verwendung der Rectified Linear Units zu den wichtigsten algorithmischen Fortschritten.*

SGD für Feedforward-Netzwerke

1. S sei eine „genügend große“ Beispielmenge. Setze $m = 0$. $w^{(0)}$ 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

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

   durch Parallelverarbeitung mit **Backpropagation** und aktualisiere

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

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

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.

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.

Backpropagation

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) Wir unterscheiden nicht zwischen Kanten und „Nicht-Kanten“:
- Berechne $\frac{\partial \ell(w,x,y)}{\partial e}$ nur für Kanten,
- $\Longrightarrow$ Nicht-Kanten behalten das Gewicht Null.

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

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

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 „δ-Terme“ $\delta_{s,j}$ für alle Tiefen s mit $t+1 \leqslant s \leqslant T$ definiert wurden, sodass 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 $t+1 \leqslant s \leqslant T \Longrightarrow$

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

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

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

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

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.

Convolutional Neural Networks (CNNs)

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)

(a) CNNs erwarten eine Eingabe in (2D oder 3D) **Gitterstruktur**.
- Die Gitterstruktur wird in nachfolgenden Schichten beibehalten.

(b) **Faltungsschichten** (Convolutional Layer) und **Pooling-Schichten** wechseln sich mehrmals ab.

(c) 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*** und beschränken uns auf *eine* Feature-Funktion.

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

- Die „Pixel“ des Ausschnitts werden dann – jeweils mit ***denselben*** Koeffizienten – gewichtet aufsummiert.
  - ⋆ Die Methode der **shared weights** wird angewandt.
- Die Auswertung erfolgt häufig mit einer Rectified Linear Unit.

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

(b) Biologische Plausibilität:

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

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

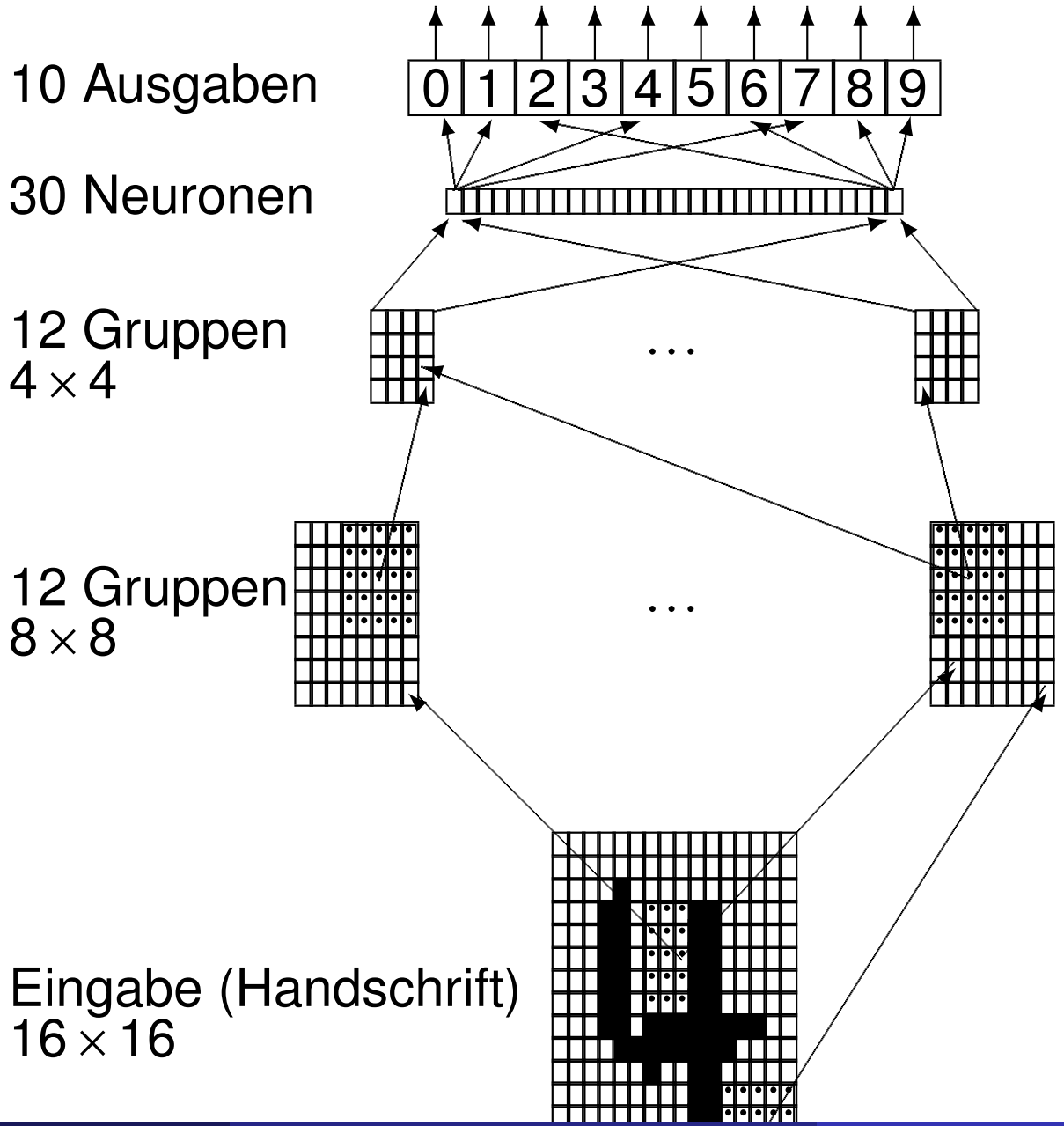
Die ***Pooling-Schicht*** soll die sehr starke Korrelation zwischen benachbarten Knoten in der Vorgänger-Schicht aufweichen.

(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 taucht eher selten auf.

Ein Beispiel: 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.

Zusammenfassung

1. Die Berechnungskraft von Feedforward-Netzwerken ist schon in kleiner Tiefe $\mathcal{O}(1)$ beträchtlich.
   - Viele analytische Funktionen lassen sich beliebig genau approximieren.
   - Die Komplexitätsklasse $\mathbf{TC}^0$ wird allerdings das Erreichbarkeitsproblem wohl nicht enthalten.
2. Das Reverse Engineering von Threshold-Schaltkreisen ist bereits in Tiefe zwei zu schwierig.
   - Viele Probleme der Objekt-Erkennung erfordern kein Reverse Engineering.
3. Backpropagation benutzt SGD, die Methode des stochastischen Gradientenabstiegs.
   - Eine Iteration ist schnell: Zeit $\mathcal{O}(|V|+|E|)$ ist für ein Minibatch der Größe 1 ausreichend.

Wann „funktioniert“ der Gradientenabstieg?