Machine Learning

B. Supervised Learning: Nonlinear Models
B.2. Neural Networks

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Fri. 25.10.	(1)	0. Introduction
		A. Supervised Learning: Linear Models & Fundamentals
Fri. 1.11.	(2)	A.1 Linear Regression
Fri. 8.11.	(3)	A.2 Linear Classification
Fri. 15.11.	(4)	A.3 Regularization
Fri. 22.11.	(5)	A.4 High-dimensional Data
		B. Supervised Learning: Nonlinear Models
Fri. 29.11.	(6)	B.1 Nearest-Neighbor Models
Fri. 6.12.	(7)	B.2 Neural Networks
Fri. 13.12.	(8)	B.3 Decision Trees
Fri. 20.12.	(9)	B.4 Support Vector Machines
		— *Christmas Break* —
Fri. 10.1.	(10)	B.5 A First Look at Bayesian and Markov Networks
		C. Unsupervised Learning
Fri. 17.1.	(11)	C.1 Clustering
Fri. 24.1.	(12)	C.2 Dimensionality Reduction
Fri. 31.1.	(13)	C.3 Frequent Pattern Mining
Fri. 7.2.	(14)	Q&A

Outline

Outline

Logistic Regression

logistic regression:

$$\begin{aligned}\hat{y}(x) := \hat{p}(y = 1 \mid x) &= \text{logistic}(\beta^T x) \\ &= 1/(1 + e^{-\beta^T x}) = 1/(1 + e^{-\sum_{m=1}^{M} \beta_m x_m}), \quad x \in \mathbb{R}^M\end{aligned}$$

Logistic Regression (0 hidden layers)

$$\begin{tikzcd} x_1 \arrow[dr] & x_2 \arrow[d] & \cdots & x_M \arrow[dll] & \text{input layer} \\ & y & & & \text{output layer} \end{tikzcd}$$

logistic regression:

$$\begin{aligned} \hat{y}(x) &:= \hat{p}(y = 1 \mid x) = \text{logistic}(\beta^T x) \\ &= 1/(1 + e^{-\beta^T x}) = 1/(1 + e^{-\sum_{m=1}^{M} \beta_m x_m}), \quad x \in \mathbb{R}^M \end{aligned}$$

Feedforward Neural Network (1 hidden layer)

$$\begin{tikzcd}
 & x_1 \arrow[dl] \arrow[d] \arrow[drrr] & x_2 \arrow[dll] \arrow[dl] \arrow[drr] & \cdots & x_M \arrow[dllll] \arrow[dlll] \arrow[d] & \text{input layer} \\
z_1 \arrow[drr] & z_2 \arrow[dr] & \cdots & & z_K \arrow[dll] & \text{hidden layer} \\
 & & y & & & \text{output layer}
\end{tikzcd}$$

feedforward neural network (1 hidden layer):

$$
\begin{aligned}
z_k(x) :=& \operatorname{logistic}(\beta_{1,k}^T x), \quad k = 1, \ldots, K, x \in \mathbb{R}^M \\
\hat{y}(x) :=& \operatorname{logistic}(\beta_2^T z(x))
\end{aligned}
$$

Feedforward Neural Network (2 hidden layers)

$$\begin{tikzcd}
 & x_1 \arrow[dl] \arrow[d] \arrow[drrrr] & x_2 \arrow[dll] \arrow[dl] \arrow[drrr] & \cdots & x_M \arrow[dllll] \arrow[dlll] \arrow[dr] & \\
z_{1,1} \arrow[d] \arrow[dr] \arrow[drrrr] & z_{1,2} \arrow[dl] \arrow[d] \arrow[drrr] & \cdots & & & z_{1,K_1} \arrow[dlllll] \arrow[dllll] \arrow[dl] \\
z_{2,1} \arrow[drr] & z_{2,2} \arrow[dr] & \cdots & & z_{2,K_2} \arrow[dll] & \\
 & & y & & &
\end{tikzcd}$$

input layer

1st hidden layer

2nd hidden layer

output layer

feedforward neural network (2 hidden layers):

$$\begin{aligned}
z_{1,k}(x) :=& \text{logistic}(\beta_{1,k}^T x), & k = 1, \ldots, K_1, \quad x \in \mathbb{R}^M \\
z_{2,k}(x) :=& \text{logistic}(\beta_{2,k}^T z_1(x)), & k = 1, \ldots, K_2 \\
\hat{y}(x) :=& \text{logistic}(\beta_3^T z_2(x))
\end{aligned}$$

Feedforward Neural Network (L hidden layers)

$$\begin{tikzcd}
 & x_1 \arrow[dl] \arrow[d] \arrow[drrr] & x_2 \arrow[dll] \arrow[dl] \arrow[drr] & \cdots & x_M \arrow[dllll] \arrow[dlll] \arrow[d] & \text{input layer} \\
z_{1,1} \arrow[d] \arrow[dr] \arrow[drrrr] & z_{1,2} \arrow[dl] \arrow[d] \arrow[drrr] & \cdots & & z_{1,K_1} \arrow[dllll] \arrow[dlll] \arrow[d] & \text{1st hidden layer} \\
z_{2,1} & z_{2,2} & \cdots & & z_{2,K_2} & \text{2nd hidden layer} \\
 & & \vdots & & & \vdots \\
z_{L,1} \arrow[drr] & z_{L,2} \arrow[dr] & \cdots & & z_{L,K_L} \arrow[dll] & L\text{th hidden layer} \\
 & & y & & & \text{output layer}
\end{tikzcd}$$

$$z_{1,k}(x) := \text{logistic}(\beta_{1,k}^T\, x), \qquad k = 1, \ldots, K_1, x \in \mathbb{R}^M$$

$$z_{\ell,k}(x) := \text{logistic}(\beta_{\ell,k}^T\, z_{\ell-1}(x)), \qquad \ell = 2, \ldots, L, k = 1, \ldots, K_\ell$$

$$\hat{y}(x) := \text{logistic}(\beta_{L+1}^T\, z_L(x))$$

Different Targets y

Binary classification:

$$\hat{y}(x) := \hat{p}(y = 1 \mid x) = \text{logistic}(\beta_{L+1}^T z_L(x)), \qquad \beta_{L+1} \in \mathbb{R}^{K_L}$$

Regression:

$$\hat{y}(x) := \beta_{L+1}^T z_L(x), \qquad \beta_{L+1} \in \mathbb{R}^{K_L}$$

Regression with multiple outputs:

$$\hat{y}(x) := \beta_{L+1} z_L(x), \qquad \beta_{L+1} \in \mathbb{R}^{T \times K_L} \text{ a matrix!}$$

Multi-class classification:

$$\hat{y}(x) := \hat{p}(y \mid x) = \text{softmax}(\beta_{L+1} z_L(x)), \qquad \beta_{L+1} \in \mathbb{R}^{T \times K_L}$$

Notes:

- L hidden layers
- at hidden nodes always are logistic/sigmoid functions (**activation function**, **transfer function**).

Softmax

$$
\begin{aligned}
\text{softmax} : \ & \mathbb{R}^T \rightarrow \mathbb{R}^T \\
\text{softmax}(u) := & \left(\frac{e^{u_t}}{\sum_{s=1}^T e^{u_s}} \right)_{t=1:T}, \quad u \in \mathbb{R}^T \\
= & \begin{pmatrix} \frac{e^{u_1}}{\sum_{s=1}^T e^{u_s}} \\ \frac{e^{u_2}}{\sum_{s=1}^T e^{u_s}} \\ \vdots \\ \frac{e^{u_T}}{\sum_{s=1}^T e^{u_s}} \end{pmatrix}
\end{aligned}
$$

Softmax

binary classification:

$$\begin{aligned}\hat{y}(x) &:= \hat{p}(y=1 \mid x) = \text{logistic}(\beta_{L+1}^T z_L(x)) \\ &= \text{logistic}(u_{L+1}(x)), \quad u_{L+1}(x) := \beta_{L+1}^T z_L(x), \beta_{L+1} \in \mathbb{R}^{K_L}\end{aligned}$$

$$\text{logistic}(u) := \frac{1}{1+e^{-u}}$$

multi-class classification:

$$\begin{aligned}\hat{y}(x) &:= \hat{p}(y \mid x) = (\hat{p}(y=t \mid x))_{t=1:T} = \text{softmax}(\beta_{L+1} z_L(x)) \\ &= \text{softmax}(u_{L+1}(x)), \quad u_{L+1}(x) := \beta_{L+1} z_L(x), \beta_{L+1} \in \mathbb{R}^{T\times}\end{aligned}$$

$$\text{softmax}(u) := \left(\frac{e^{u_t}}{\sum_{s=1}^T e^{u_s}}\right)_{t=1:T}, \quad u \in \mathbb{R}^T$$

Softmax / Generalization of the Logistic

binary classification:

$$\hat{y}(x) := \hat{p}(y = 1 \mid x) = \text{logistic}(\beta_{L+1}^T z_L(x))$$
$$= \text{logistic}(u_{L+1}(x)), \quad u_{L+1}(x) := \beta_{L+1}^T z_L(x), \beta_{L+1} \in \mathbb{R}^{K_L}$$

$$\text{logistic}(u) := \frac{1}{1+e^{-u}} = \frac{e^u}{1+e^u} = \begin{pmatrix} \frac{e^0}{e^0+e^u} \\ \frac{e^u}{e^0+e^u} \end{pmatrix}_2 = (\text{softmax}(\begin{pmatrix} 0 \\ u \end{pmatrix}))_2$$

$$\begin{pmatrix} \hat{p}(y=0 \mid x) \\ \hat{p}(y=1 \mid x) \end{pmatrix} = \text{softmax}(\begin{pmatrix} 0 \\ u_{L+1}(x) \end{pmatrix})$$

multi-class classification:

$$\hat{y}(x) := \hat{p}(y \mid x) = (\hat{p}(y = t \mid x))_{t=1:T} = \text{softmax}(\beta_{L+1} z_L(x))$$
$$= \text{softmax}(u_{L+1}(x)), \quad u_{L+1}(x) := \beta_{L+1} z_L(x), \beta_{L+1} \in \mathbb{R}^{T\times}$$

$$\text{softmax}(u) := \left(\frac{e^{u_t}}{\sum_{s=1}^T e^{u_s}}\right)_{t=1:T}, \quad u \in \mathbb{R}^T$$

Softmax Properties

$$\text{softmax}(u) := \left(\frac{e^{u_t}}{\sum_{s=1}^{T} e^{u_s}} \right)_{t=1:T}, \quad u \in \mathbb{R}^T$$

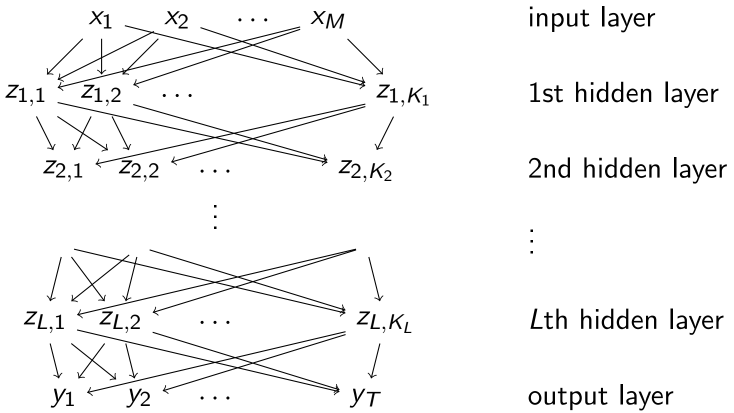
- softmax is a generalization of the logistic function from 2 to T classes.
- softmax is continuous and differentiable.
- softmax components sum to one:

$$\sum_{t=1}^{T} (\text{softmax}(u))_t = 1$$

- softmax in the limit approaches the maximum indicator:

$$\lim_{a \to \infty} \text{softmax}(a \cdot u) = (\mathbb{I}(u_t = u_{\max}))_{t=1:T}, \quad u_{\max} := \max_{s \in 1:T} u(s)$$

Feedforward Neural Network (L hidden layers, T outputs)


x_1 x_2 $\cdots$ x_M
input layer
$z_{1,1}$ $z_{1,2}$ $\cdots$ z_{1,K_1}
1st hidden layer
$z_{2,1}$ $z_{2,2}$ $\cdots$ z_{2,K_2}
2nd hidden layer
$\vdots$
$z_{L,1}$ $z_{L,2}$ $\cdots$ z_{L,K_L}
Lth hidden layer
y_1 y_2 $\cdots$ y_T
output layer

$$z_{1,k}(x) := s_1(\beta_{1,k}^T x), \qquad k = 1, \ldots, K_1, \quad x \in \mathbb{R}^M$$

$$z_{\ell,k}(x) := s_\ell(\beta_{\ell,k}^T z_{\ell-1}(x)), \quad \ell = 2, \ldots, L, \quad k = 1, \ldots, K_\ell$$

$$\hat{y}_k(x) := s_{L+1,k}(\beta_{L+1,.}^T z_L(x)), \quad k = 1, \ldots, T$$

Beware: for softmax s_{L+1} is **not** elementwise!

Feedforward Neural Network (L hidden layers, T outputs)

make it simple 1: each layer a vector

$$\begin{tikzcd}
x \arrow[d] & \text{input layer} \\
z_1 \arrow[d] & \text{1st hidden layer} \\
z_2 & \text{2nd hidden layer} \\
\vdots \arrow[d] & \vdots \\
z_L \arrow[d] & L\text{th hidden layer} \\
y & \text{output layer}
\end{tikzcd}$$

$$z_1(x) := s_1^\circ(\beta_1 x) \qquad \beta_1 \in \mathbb{R}^{K_1 \times M}$$

$$z_\ell(x) := s_\ell^\circ(\beta_\ell z_{\ell-1}(x)), \quad \ell = 2, \ldots, L, \qquad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$$

$$\hat{y}(x) := s_{L+1}(\beta_{L+1} z_L(x)) \qquad \beta_{L+1} \in \mathbb{R}^{T \times K_L}$$

Beware: for softmax s_{L+1} is **not** elementwise!

Feedforward Neural Network (L hidden layers, T outputs)

make it simple 2: rename x and $\hat{y}$ to z_0 and z_{L+1}

$$\begin{tikzcd}
z_0 \arrow[d] & \text{input layer} \\
z_1 \arrow[d] & \text{1st hidden layer} \\
z_2 & \text{2nd hidden layer} \\
\vdots \arrow[d] & \vdots \\
z_L \arrow[d] & L\text{th hidden layer} \\
z_{L+1} & \text{output layer}
\end{tikzcd}$$

$$z_\ell(x) := s_\ell(\beta_\ell \, z_{\ell-1}(x)), \quad \ell = 1, \ldots, L+1, \qquad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$$

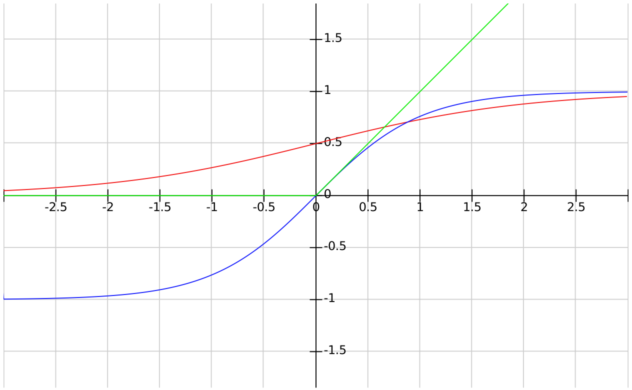
with $z_0 := x$, $\hat{y}(x) := z_{L+1}(x)$, $K_0 := M$, $K_{L+1} := T$, $s_\ell = s_\ell^\circ$ for $\ell \neq L+1$

Beware: for softmax s_{L+1} is **not** elementwise!

Activation Functions

Nowadays, usually the **rectifier** is used as activation function s (such nodes are called **ReLU**: **rectified linear unit**):

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



red: logistic, blue: tanh, green: rect

Parameters & Structure

- model:

$$z_\ell(x) := s_\ell(\beta_\ell \, z_{\ell-1}(x)), \quad \ell = 1, \ldots, L+1 \qquad \beta_\ell \in$$

with $z_0 := x$, $\hat{y}(x) := z_{L+1}(x)$, $K_0 := M$, $K_{L+1} := T$, $s_\ell = s^\circ_\ell$ for $\ell \neq L+1$

- parameters:

$$\beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}, \quad \ell = 1, \ldots, L+1$$

- structure:
  - $L \in \mathbb{N}_0$: number of hidden layers.
  - $K_\ell \in \mathbb{N}, \quad \ell = 1, \ldots, L$: size of hidden layer ℓ.
  - $s_\ell = s^\circ, \quad \ell = 1, \ldots, L$: activation function of hidden layer ℓ.
    (applied elementwise and usually not dependent on the layer ℓ)
  - s_{L+1}: output function.

Predicting with a Neural Network

```
**predict-nn**$(x, \beta, s)$ :
    $z_0 := x$
    for $\ell := 1, \ldots, L+1$ :
        $z_\ell := s_\ell^\circ(\beta_\ell \, z_{\ell-1})$
    return $z_{L+1}$
```

Network Topologies

- **feedforward neural network** (aka **multilayer perceptron**, MLP)
    - often just a single hidden layer is used
        - NN with single hidden layer is already a **universal approximator**
    - **skip arcs** can be used to connect layers skipping a hidden layer
    - usually layers are connected completely (**fully connected layer**), but sometimes **sparse connections** are used.
    - nodes & connections always form a DAG
- **recurrent neural network**
    - neural networks with backward connections / not a DAG.
    - used in language modeling
    - no simple probabilistic interpretation
    - nowadays usually rolled out to a feedforward net with tied weights
- **Hopfield networks** / **associative memory**:
    - symmetric connections between hidden units

Outline

Vector Calculus Refresh – Gradients & Jacobians

function with N inputs, single output:

$$\begin{aligned} f : \mathbb{R}^N &\to \mathbb{R} \\ x &\mapsto f(x_1, \ldots, x_N) \end{aligned}$$

gradient (vector):

$$\nabla f(x) := \left(\frac{\partial f}{\partial x_n}(x)\right)_{n=1:N}$$

function/map with N inputs, M outputs:

$$\begin{aligned} f : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto (f_m(x_1, \ldots, x_N))_{m=1:M} \end{aligned}$$

Jacobian (matrix) / **total derivative**:

$$Df(x) := \left(\frac{\partial f_m}{\partial x_n}(x)\right)_{m=1:M,n=1:N}$$

Vector Calculus Refresh – Gradients & Jacobians

function with N inputs, single output:

$$\begin{aligned} f : \mathbb{R}^N &\to \mathbb{R} \\ x &\mapsto f(x_1, \ldots, x_N) \end{aligned}$$

gradient (vector):

$$\nabla f(x) := \left(\frac{\partial f}{\partial x_n}(x) \right)_{n=1:N} = (Df(x))^T$$

function/map with N inputs, M outputs:

$$\begin{aligned} f : \mathbb{R}^N &\to \mathbb{R}^M \\ x &\mapsto (f_m(x_1, \ldots, x_N))_{m=1:M} \end{aligned}$$

Jacobian (matrix) / **total derivative**:

$$Df(x) := \left(\frac{\partial f_m}{\partial x_n}(x) \right)_{m=1:M, n=1:N} = \begin{pmatrix} \nabla f_1(x)^T \\ \nabla f_2(x)^T \\ \vdots \\ \nabla f_M(x)^T \end{pmatrix}$$

Vector Calculus Refresh – Chain Rule

function composition:

$$\begin{array}{rccc} & X := \mathbb{R}^N & \xrightarrow{f} & Y := \mathbb{R}^M \\ f: & x & \mapsto & f(x) \end{array}$$

Vector Calculus Refresh – Chain Rule

function composition:

$$\begin{array}{llllll} & X := \mathbb{R}^N & \xrightarrow{f} & Y := \mathbb{R}^M & \xrightarrow{g} & Z := \mathbb{R} \\ f : & x & \mapsto & f(x) & & \\ g : & & & y & \mapsto & g(y) \end{array}$$

Vector Calculus Refresh – Chain Rule

function composition:

$$\begin{array}{rcccccc} & X := \mathbb{R}^N & \xrightarrow{f} & Y := \mathbb{R}^M & \xrightarrow{g} & Z := \mathbb{R} & \\ f: & x & \mapsto & f(x) & & & \\ g: & & & y & \mapsto & g(y) & \\ g \circ f: & x & & & \mapsto & (g \circ f)(x) & := g(f(x)) \end{array}$$

chain rule:

$$\begin{aligned} D(g \circ f)(x) &= Dg(f(x))\, Df(x) \\ \nabla(g \circ f)(x) &= Df(x)^T (\nabla g)(f(x)) \end{aligned}$$

Vector Calculus Refresh – Elementwise Function Application

function with single input, single output:

$$\begin{aligned} f : \mathbb{R} &\rightarrow \mathbb{R} \\ x &\mapsto f(x) \end{aligned}$$

elementwise function application:

$$\begin{aligned} f^\circ : \mathbb{R}^N &\rightarrow \mathbb{R}^N \\ x &\mapsto (f(x_n))_{n=1:N} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_N) \end{pmatrix} \end{aligned}$$

its Jacobian:

$$Df^\circ(x) = \begin{pmatrix} f'(x_1) & & & \\ & f'(x_2) & & \\ & & \ddots & \\ & & & f'(x_N) \end{pmatrix} = \operatorname{diag}(f'^\circ(x))$$

Vector Calculus Refresh – Partial Gradients & Jacobians

function with N inputs, single output:

$$\begin{aligned} f : \mathbb{R}^N &\rightarrow \mathbb{R} \\ x &\mapsto f(x_1, \ldots, x_N) \end{aligned}$$

partial gradient (vector):

$$\nabla_I f(x) := \left(\frac{\partial f}{\partial x_n}(x) \right)_{n \in I}, \quad I \subseteq \{1, \ldots, N\}$$

function/map with N inputs, M outputs:

$$\begin{aligned} f : \mathbb{R}^N &\rightarrow \mathbb{R}^M \\ x &\mapsto (f_m(x_1, \ldots, x_N))_{m=1:M} \end{aligned}$$

partial Jacobian (matrix):

$$D_I f(x) := \left(\frac{\partial f_m}{\partial x_n}(x) \right)_{m=1:M, n \in I} \qquad I \subseteq \{1, \ldots, N\}$$

Objective Function

feedforward neural network, L hidden layers with $K_1, \ldots, K_L$ nodes each:

$$z_\ell(x) := s_\ell(\beta_\ell \, z_{\ell-1}(x)), \quad \ell = 1, \ldots, L+1, \qquad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$$

with $z_0 := x, \quad \hat{y}(x) := z_{L+1}(x), \quad K_0 := M, \quad K_{L+1} := T, \quad s_\ell = s^\circ_\ell$ for $\ell \neq L+1$

objective function:

$$f(\beta) := \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y_n, \hat{y}(x_n)) + \frac{\lambda}{2} ||\beta||^2 = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\beta; x_n, y_n) + \frac{\lambda}{2} ||\beta||^2$$

loss for single sample:

$$\mathcal{L}(\beta; x, y) := \mathcal{L}(y, z_{L+1}(x))$$

Objective Function

feedforward neural network, L hidden layers with $K_1, \ldots, K_L$ nodes each:

$$\begin{aligned} u_\ell(x) &:= \beta_\ell \, z_{\ell-1}(x), \quad \ell = 1, \ldots, L+1, \quad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}} \\ z_\ell(x) &:= s_\ell^\circ(u_\ell(x)), \quad \ell = 1, \ldots, L \\ z_{L+1}(x) &:= s_{L+1}(u_\ell(x)) \end{aligned}$$

with $z_0 := x, \quad \hat{y}(x) := z_{L+1}(x), \quad K_0 := M, \quad K_{L+1} := T$

loss for single sample:

$\mathcal{L}(\beta; x, y) := \mathcal{L}(y, z_{L+1}(x))$

Objective Function

feedforward neural network, L hidden layers with $K_1, \ldots, K_L$ nodes each:

$$u_\ell(z_{\ell-1}) := \beta_\ell \, z_{\ell-1}, \quad \ell = 1, \ldots, L+1, \quad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$$

$$z_\ell(u_\ell) := s^\circ(u_\ell), \qquad z_{L+1}(u_{L+1}) := s_{L+1}(u_{L+1})$$

with $z_0 := x, \quad \hat{y}(x) := (z_{L+1} \circ u_{L+1} \circ z_L \circ u_L \circ \cdots \circ z_1 \circ u_1)(x), \quad K_0 := M, \quad K_{L+1} := T$

loss for single sample:

$$\mathcal{L}(\beta; x, y) := \mathcal{L}(y, z_{L+1}(x)) = (\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_\ell \circ u_\ell \circ \cdots z_1 \circ u_1)(x)$$

with pair loss $\mathcal{L}_y(z_{L+1}) := \text{loss}(y, z_{L+1})$

Objective Function

feedforward neural network, L hidden layers with $K_1, \ldots, K_L$ nodes each:

$$u_\ell(z_{\ell-1}) := \beta_\ell \, z_{\ell-1}, \quad \ell = 1, \ldots, L+1, \quad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$$

$$z_\ell(u_\ell) := s^\circ(u_\ell), \qquad z_{L+1}(u_{L+1}) := s_{L+1}(u_{L+1})$$

with $z_0 := x, \quad \hat{y}(x) := (z_{L+1} \circ u_{L+1} \circ z_L \circ u_L \circ \cdots \circ z_1 \circ u_1)(x), \quad K_0 := M, \quad K_{L+1} := T$

loss for single sample:

$$\mathcal{L}(\beta; x, y) := \mathcal{L}(y, z_{L+1}(x)) = (\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_\ell \circ u_\ell \circ \cdots z_1 \circ u_1)(x)$$

with pair loss $\mathcal{L}_y(z_{L+1}) := \text{loss}(y, z_{L+1})$

its gradients:

$$\nabla_{\beta_{\ell,k}} \mathcal{L}(\beta) = D_{\beta_{\ell,k}} u_\ell(z_{\ell-1})^T \nabla(\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_{\ell+1} \circ u_{\ell+1} \circ z_\ell)(u_\ell)$$

$$\nabla(\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_{\ell+1} \circ u_{\ell+1} \circ z_\ell)(u_\ell)$$

$$= D z_\ell^T D u_{\ell+1}^T \nabla(\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_{\ell+2} \circ u_{\ell+2} \circ z_{\ell+1})(u_{\ell+1})$$

Gradients / Recursion Scheme

single sample loss gradients:

$$\nabla_{\beta_{\ell,k}}\mathcal{L}(\beta) = D_{\beta_{\ell,k}} u_\ell(z_{\ell-1})^T \nabla(\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_{\ell+1} \circ u_{\ell+1} \circ z_\ell)(u_\ell)$$

$$\nabla(\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_{\ell+1} \circ u_{\ell+1} \circ z_\ell)(u_\ell)$$
$$= Dz_\ell^T Du_{\ell+1}^T \nabla(\mathcal{L}_y \circ z_{L+1} \circ u_{L+1} \circ \cdots z_{\ell+2} \circ u_{\ell+2} \circ z_{\ell+1})(u_{\ell+1})$$

establishes a recursive computation scheme:

$$\begin{aligned}\nabla_{\beta_{\ell,k}}\mathcal{L}(\beta) &= D_{\beta_{\ell,k}} u_\ell(z_{\ell-1})^T g_\ell(u_\ell)\\ g_\ell(u_\ell) &:= Dz_\ell^T Du_{\ell+1}^T\, g_{\ell+1}(u_{\ell+1})\\ g_{L+1}(u_{L+1}) &:= Dz_{L+1}^T \nabla \mathcal{L}_y(z_{L+1})\end{aligned}$$

Gradients / Components

$$u_\ell(z_{\ell-1}) := \beta_\ell\, z_{\ell-1}, \quad \ell = 1, \ldots, L+1, \quad \beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$$
$$z_\ell(u_\ell) := s^\circ(u_\ell), \qquad z_{L+1}(u_{L+1}) := s_{L+1}(u_{L+1})$$

single sample loss gradients:

$$\nabla_{\beta_{\ell,k}} \mathcal{L}(\beta) = D_{\beta_{\ell,k}} u_\ell(z_{\ell-1})^T g_\ell(u_\ell)$$
$$g_\ell(u_\ell) := Dz_\ell^T\, Du_{\ell+1}^T\, g_{\ell+1}(u_{\ell+1})$$
$$g_{L+1}(u_{L+1}) := Dz_{L+1}^T \nabla \mathcal{L}_y(z_{L+1})$$

components:

$$Du_\ell = \beta_\ell$$
$$Dz_\ell = \operatorname{diag}(s'^\circ(u_\ell))$$
$$D_{\beta_{\ell,k}} u_\ell = e_k\, z_{\ell-1}^T \qquad \rightsquigarrow \qquad \begin{aligned} \nabla_{\beta_{\ell,k}} \mathcal{L}(\beta) &= z_{\ell-1}\, e_k^T\, g_\ell(u_\ell) \\ \nabla_{\beta_\ell} \mathcal{L}(\beta) &= g_\ell(u_\ell)\, z_{\ell-1}^T \end{aligned}$$

Note: e_k denotes the k-th unit vector: $(e_k)_j := \mathbb{I}(k = j)$.
β_ℓ is a parameter matrix, thus $\nabla_{\beta_\ell} \mathcal{L}(\beta)$ is a matrix-shaped gradient!

Gradients / Sticking Everything Together

parameters:

$$\beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}, \quad \ell = 1 : L+1$$

feed forward:

$$\begin{aligned} z_0 &:= x \\ u_\ell &:= \beta_\ell \, z_{\ell-1}, \quad \ell = 1 : L+1 \\ z_\ell &:= s^\circ(u_\ell) \end{aligned}$$

back propagation:

$$\begin{aligned} g_{L+1}(u_{L+1}) &:= Ds_{L+1}(u_{L+1}) \, \nabla \mathcal{L}_y(z_{L+1}) \\ g_\ell(u_\ell) &:= \text{diag}(s'^\circ(u_\ell)) \, \beta_{\ell+1}^T \, g_{\ell+1}(u_{\ell+1}) \\ \nabla_{\beta_\ell} \mathcal{L}(\beta) &= g_\ell(u_\ell) \, z_{\ell-1}^T, \quad \ell = L+1 : 1 \text{ backwards} \\ \beta_\ell^{\text{next}} &:= \beta_\ell - \eta(\nabla_{\beta_\ell} \mathcal{L}(\beta) + \lambda \beta_\ell) \end{aligned}$$

SGD / Backpropagation

```
learn-nn-sgd($\mathcal{D}^{\text{train}} := \{(x_1, y_1), \ldots, (x_N, y_N)\}, L, K, s, \nabla\mathcal{L}, \lambda, \eta, I$):
  randomly initialize $\beta_\ell \in \mathbb{R}^{K_\ell \times K_{\ell-1}}$, $\ell = 1 : L+1$
  for $i := 1, \ldots, I$:
    for $(x_n, y_n) \in \mathcal{D}^{\text{train}}$ in random order:
      $z_0 := x_n$                                                [feed forward]
      for $\ell := 1 : L+1$:
        $u_\ell := \beta_\ell \, z_{\ell-1}$
        $z_\ell := s^\circ(u_\ell)$
      $g_{L+1} := \text{diag}(s'^\circ(u_{L+1}))\nabla\mathcal{L}_{y_n}(z_{L+1})$      [back propagation]
      for $\ell := L+1 : 2$ backwards:
        $g_{\ell-1} := \text{diag}(s'^\circ(u_{\ell-1}))\, \beta_\ell^T \, g_\ell$
        $\beta_\ell := \beta_\ell - \eta_i(g_\ell \, z_{\ell-1}^T + \lambda\beta_\ell)$
      $\beta_1 := \beta_1 - \eta_i(g_1 \, z_0^T + \lambda\beta_1)$
    if converged(...):
      return $\beta$
  raise exception "not converged in $I$ iterations"
```

where

- L number of layers
- K layer sizes
- s activation function
- $\nabla\mathcal{L}$ loss gradient
- λ regularization weight
- η step length schedule
- I number of iterations

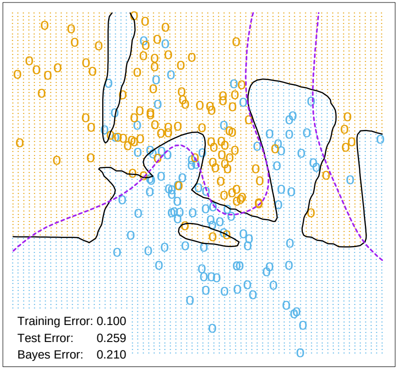
Outline

Regularization of Neural Networks

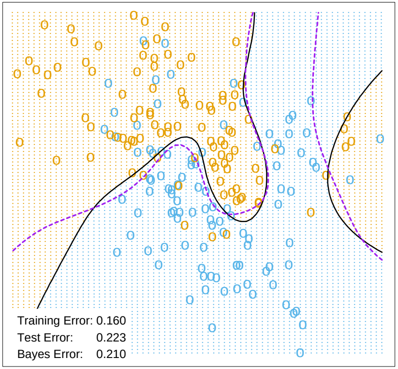
- generic, working with any model:
  - L2 regularization
    - aka **weight decay**
    - most frequently used method
  - L1 regularization
  - number of iterations as hyperparameter (**early stopping**)
- specific for neural networks:
  - **structural regularization**:
    - sufficiently small number of layers and sizes of layers
    - compare number of parameters with sample size!
  - **dropout** [Srivastava et al., 2014]
    - use random sample of input nodes and hidden nodes for each instance during training
  - **Batch normalization** [Ioffe and Szegedy, 2015]
    - standardize the values $z_{\ell,k}$ for each layer (for a minibatch).
  - **self-normalizing neural networks** [Klambauer et al., 2017]

L2 regularization / Example

Neural Network - 10 Units, No Weight Decay

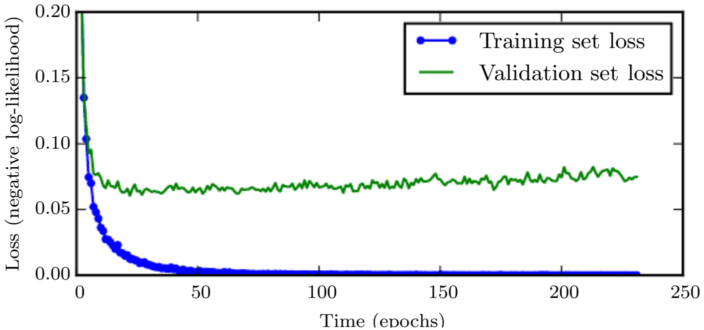


Neural Network - 10 Units, Weight Decay=0.02



[Hastie et al., 2005, p. 39

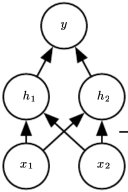
Early Stopping



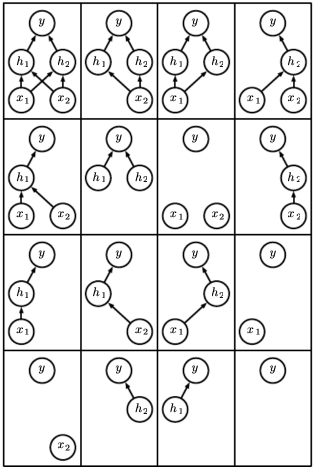
[source: Goodfellow et al. 2016, p. 239]

Early stopping works with any iterative learning algorithm.

Dropout



Base network



Ensemble of subnetworks

[source: Goodfellow et al. 2016, p. 252]

Summary (1/4)

- (Feedforward) Neural networks are parametrized models for supervised learning problems
  - arranged in several layers,
    - with the first layer being the inputs,
    - the last layer being the outputs,
    - intermediate/**hidden layers** representing subexpressions of the prediction function
      (not to be confused with latent variables!)
  - each layer composed of a **linear combination of the previous one**, with weights being parameters of the model,
  - and a **nonlinear activation function**,
    - usually the linear rectifier $\max(0, x)$
    - or a sigmoid function (logistic, tanh)
- Neural networks are learnt through Stochastic Gradient Descent
  - computation of the gradients in reverse order of computations of predictions (**backpropagation**)
  - [illegible]

Summary (2/4)

- The loss function of a neural network is **non-convex**.
- Neural networks can be learnt via Stochastic Gradient Descent
  - computation of the gradients in reverse order of computations of predictions (**backpropagation**)
  - usually using **minibatches** for a few ten or hundred instances.
  - starting from a **random initialization**.

Summary (3/4)

- As any other model, neural networks have to be regularized.
    - **structural regularization**:
        - number of nodes/layer and number of layers.
    - early stopping
    - L2 regularization (**weight decay**)
    - **dropout**: use a random sample of input and hidden nodes per example
- Neural networks can be extended in a rather straightforward way to work with sequential/time series, image data and any other kind of array data.
    - **convolutional neural networks**
    - **recurrent neural networks** (including LSTM, GRU)
    - these models belong to the most powerful models currently used in ML

Summary (4/4)

- A neural network with a single hidden layer can already approximate any function arbitrarily well.
  - **universal approximator**
  - if one adds arbitrarily many hidden nodes in that layer as necessary
  - but deeper networks with more than one hidden layer have shown to generalize better
    - esp. for structured data
    - make better use of a given number of parameters
    - **deep learning**

Further Readings

- See Murphy 2012, chapter 16.5 and Hastie et al. 2005, chapter 11.
- More detailed introduction in Goodfellow et al. 2016, chapter 6 and 7.

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The Mit Press, Cambridge, Massachusetts, November 2016. ISBN 978-0-262-03561-3.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, volume 27. Springer, 2005.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-Normalizing Neural Networks. *arXiv preprint arXiv:1706.02515*, 2017.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.