



FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Feed Forward Neural Networks

A. Maier, K. Breininger, L. Mill, N. Ravikumar, T. Würfl

Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg

October 23, 2018



Outline

Model

Perceptrons as Universal Function Approximators

From Activations to Classifications: Softmax Function

Optimization

Layer Abstraction



FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Model

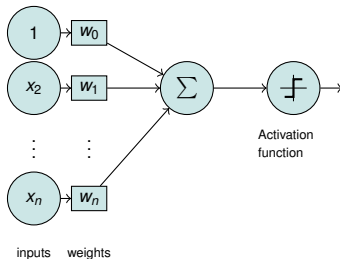


Recap: Perceptrons

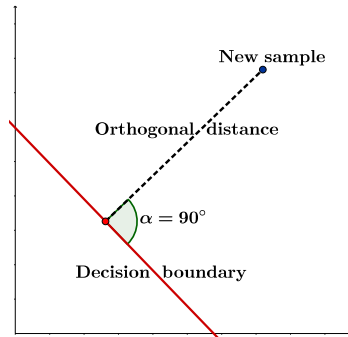
- Perceptron's decision rule:

$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$$

- Classification only depends on sign of distance

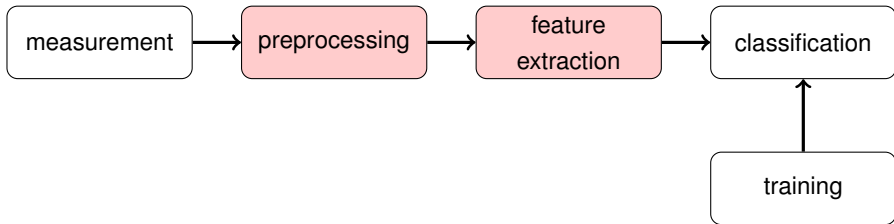


Perceptron



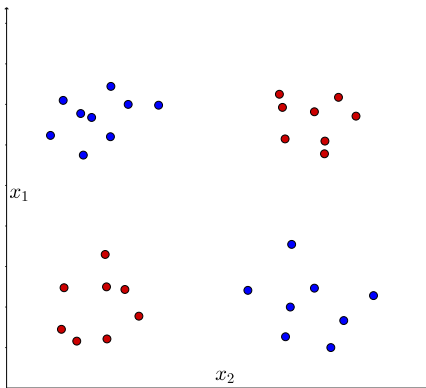
Linear decision boundary

Recap: Pattern Recognition Pipeline



- (Multi-layer) perceptron (today's lecture) still uses predefined features
- “Hand-crafted” feature design is replaced by **data driven feature learning** in state-of-the-art architectures (upcoming lectures)
- Most concepts are important across architectures!

XOR Problem

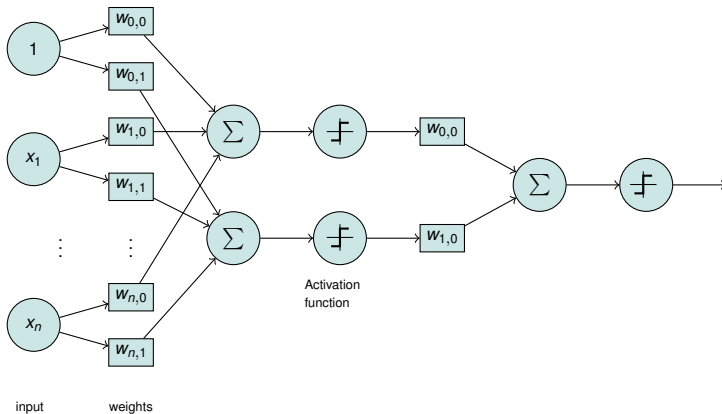


Samples from a XOR problem

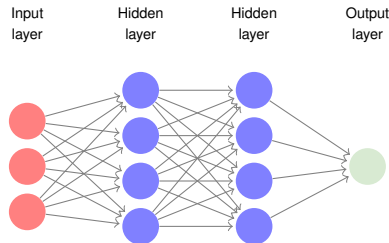
- XOR can't be solved with a line
- 1969: "Perceptrons" described limitations of neural networks
- AI funding was cut heavily
- This became known as "AI winter"

Multi-Layer Perceptron

- A perceptron resembles a single neuron
- Idea: Use multiple neurons!
- This enables non-linear decision boundaries



Terminology



- Terms: Input layer, hidden layers, output layer
- A single hidden layer (of arbitrary width) can already be shown to be a **universal function approximator**
- Non-linear functions
 - are called activation functions in hidden layers
 - predict in the output layer and are used for the loss function



FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Perceptrons as Universal Function Approximators



Universal Approximation Theorem

- Let $\varphi(\cdot)$ be a non-constant, bounded and monotonically increasing function.
- For any $\varepsilon > 0$ and any continuous function f defined on a compact subset of \mathbb{R}^m , there exist an integer N , real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ where $i = 1, \dots, N$, such that

$$F(\mathbf{x}) = \sum_{i=1}^N v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i) \quad \text{with}$$
$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

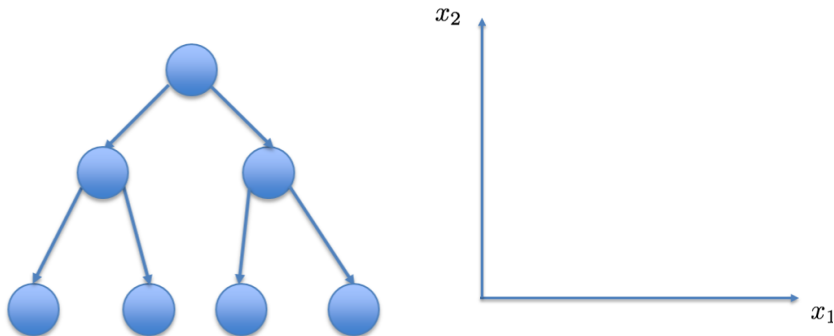
Universal Approximation Theorem

- Let $\varphi(\cdot)$ be a non-constant, bounded and monotonically increasing function.
- For any $\varepsilon > 0$ and any continuous function f defined on a compact subset of \mathbb{R}^m , there exist an integer N , real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ where $i = 1, \dots, N$, such that

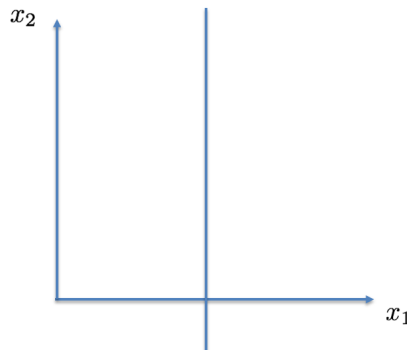
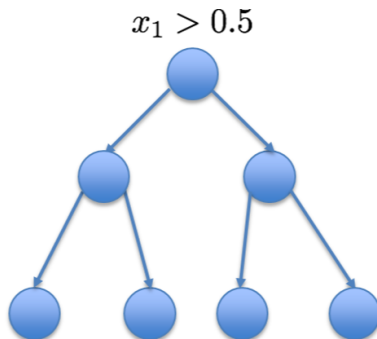
$$F(\mathbf{x}) = \sum_{i=1}^N v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i) \quad \text{with}$$
$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

→ We can approximate *any function with just one hidden layer* with a sensible activation function.

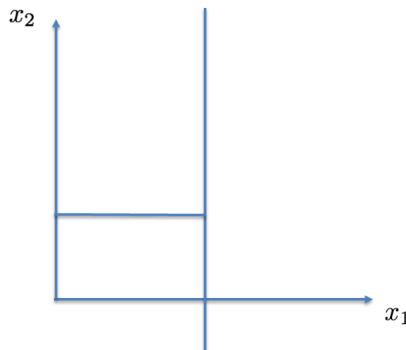
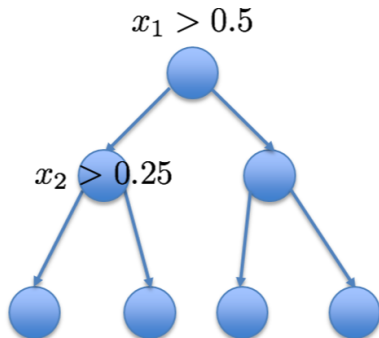
Classification Trees: Why do we need a universal approximator?



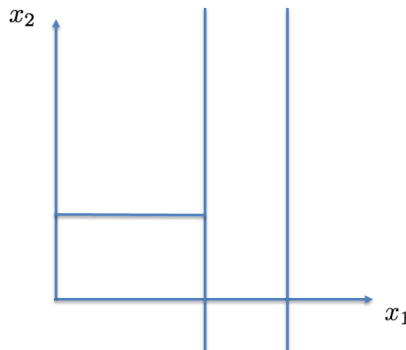
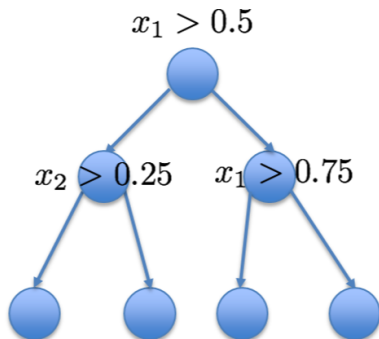
Classification Trees: Why do we need a universal approximator?



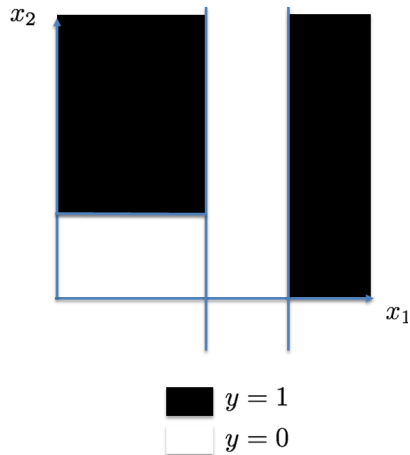
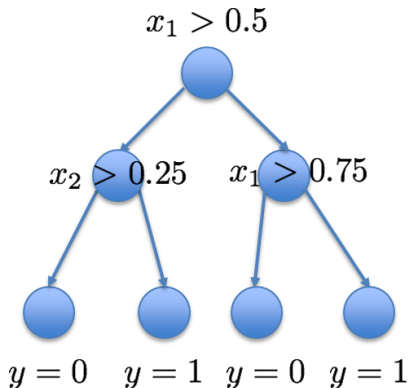
Classification Trees: Why do we need a universal approximator?



Classification Trees: Why do we need a universal approximator?



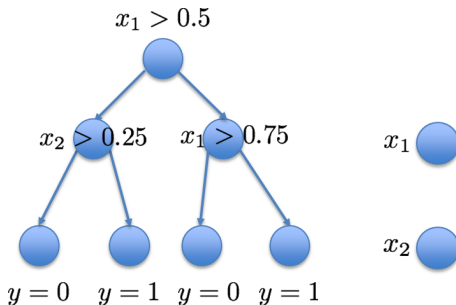
Classification Trees: Why do we need a universal approximator?



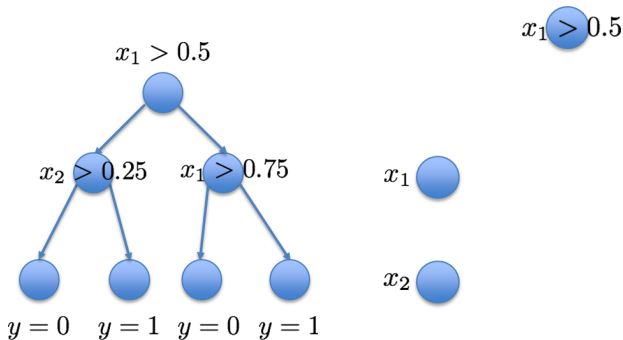
Classification Trees: Why do we need a universal approximator?

We can transform this into a network!

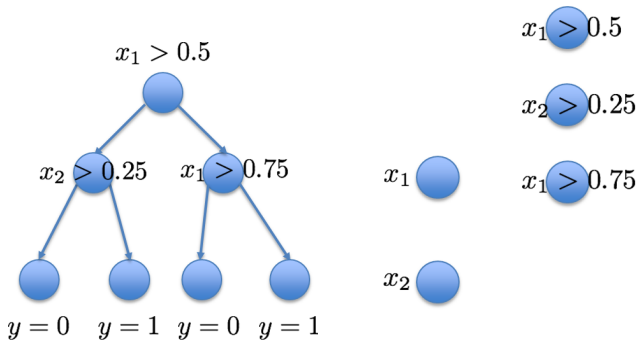
Classification Trees: Why do we need a universal approximator?



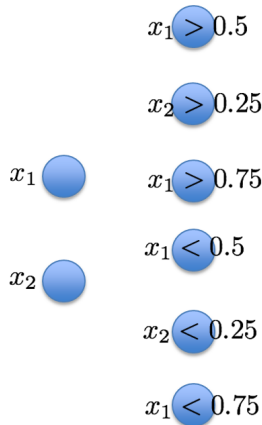
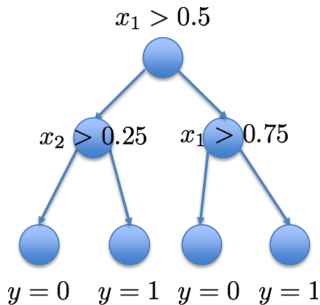
Classification Trees: Why do we need a universal approximator?



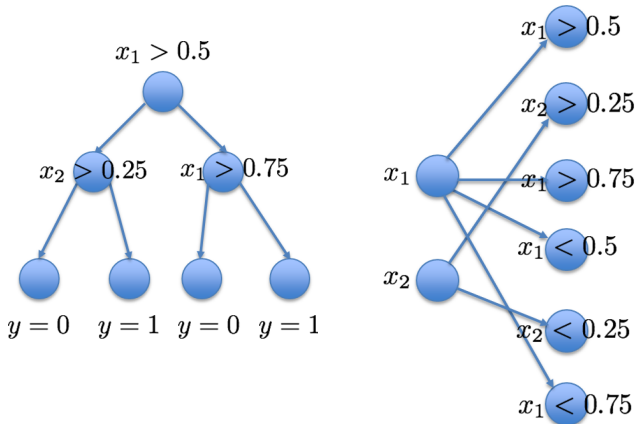
Classification Trees: Why do we need a universal approximator?



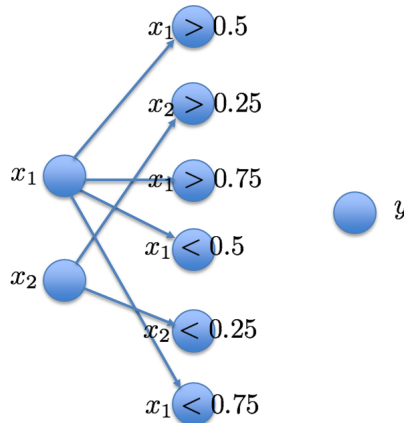
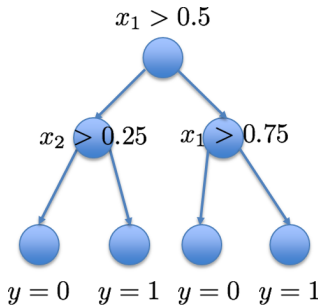
Classification Trees: Why do we need a universal approximator?



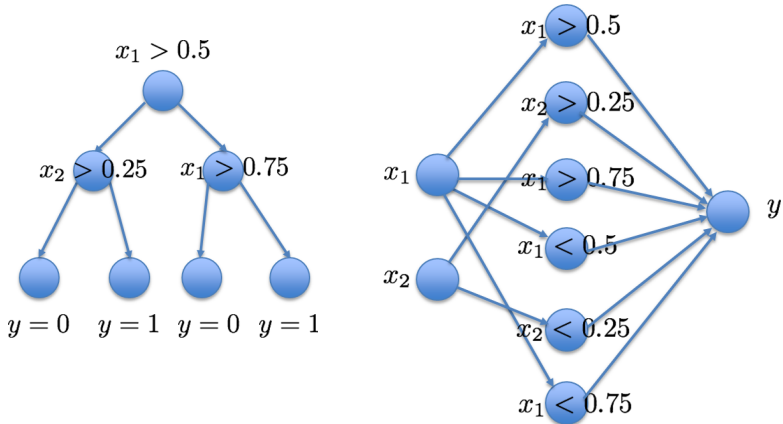
Classification Trees: Why do we need a universal approximator?



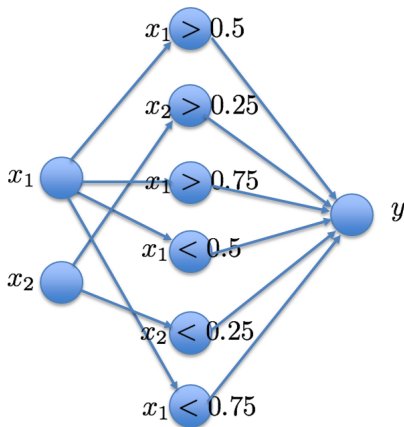
Classification Trees: Why do we need a universal approximator?



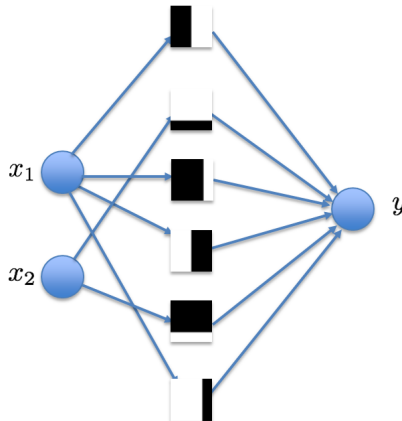
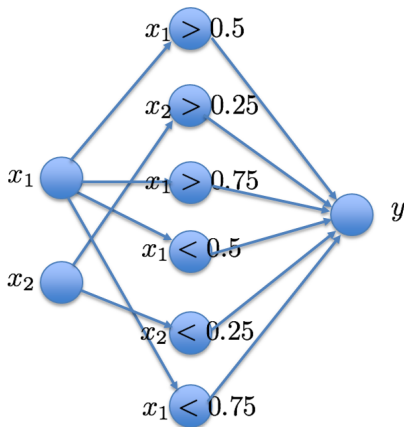
Classification Trees: Why do we need a universal approximator?



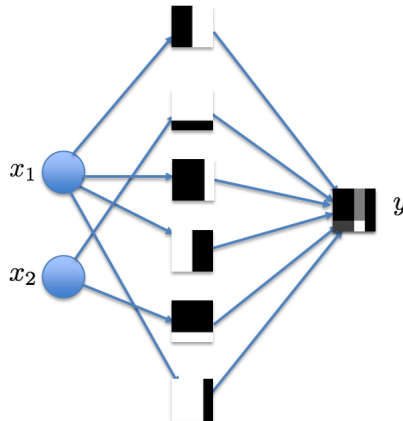
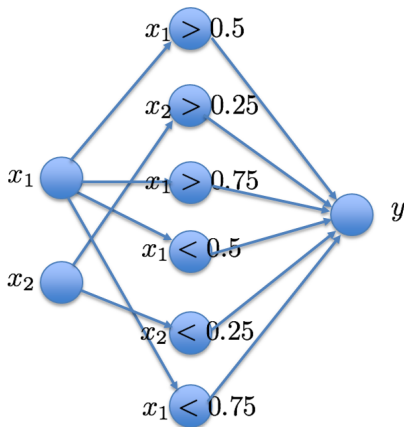
Classification Trees: Why do we need a universal approximator?



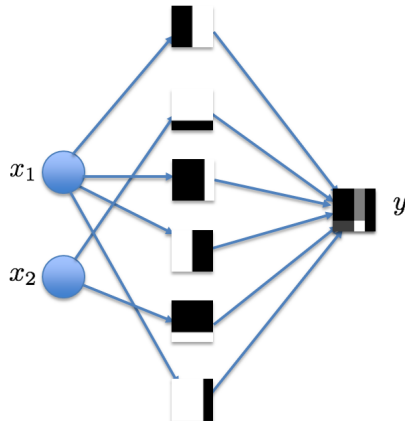
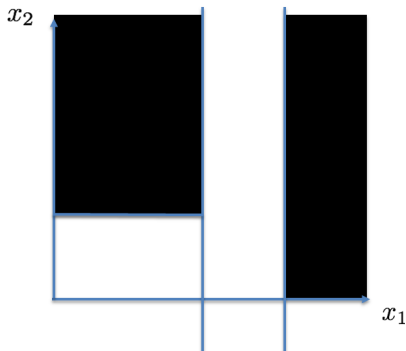
Classification Trees: Why do we need a universal approximator?



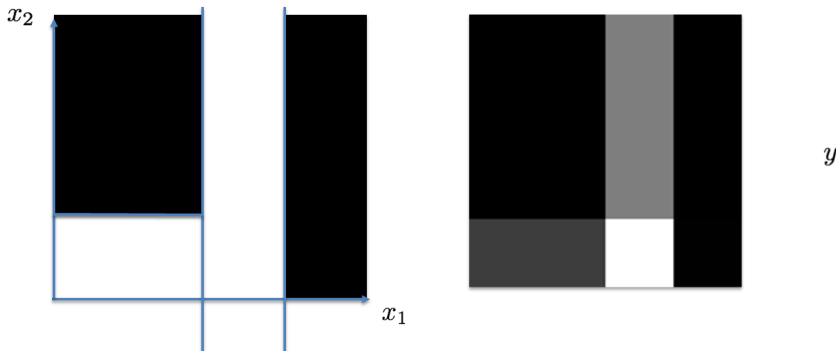
Classification Trees: Why do we need a universal approximator?



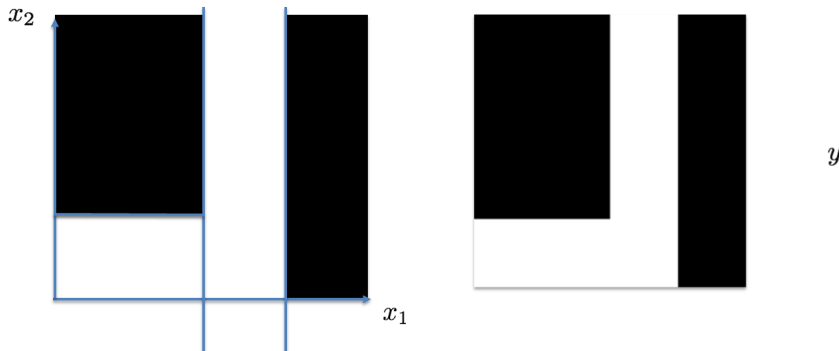
Classification Trees: Why do we need a universal approximator?



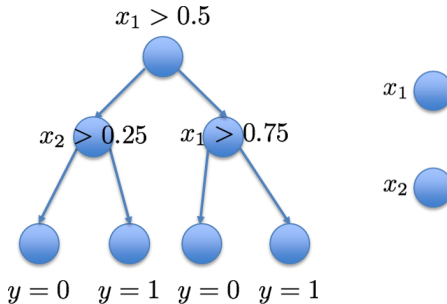
Classification Trees: Why do we need a universal approximator?



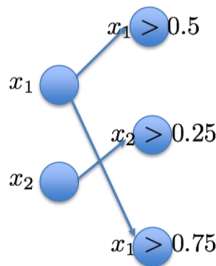
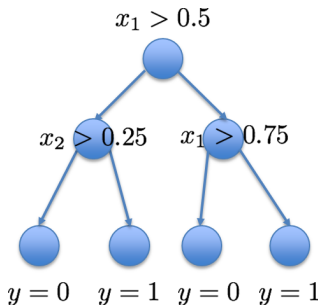
Classification Trees: Why do we need a universal approximator?



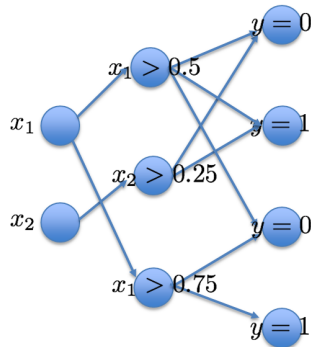
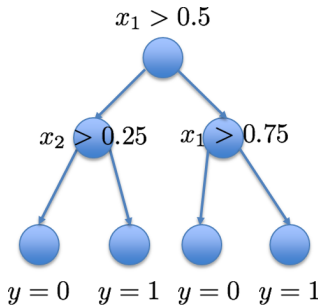
Classification Trees: Why do we need a universal approximator?



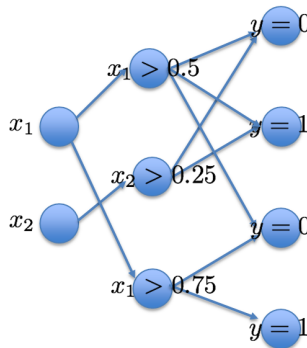
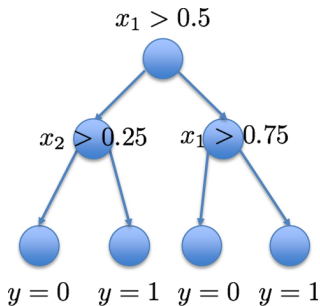
Classification Trees: Why do we need a universal approximator?



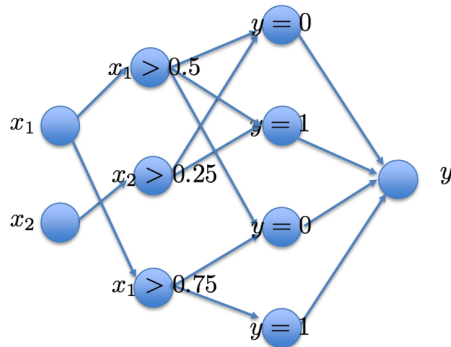
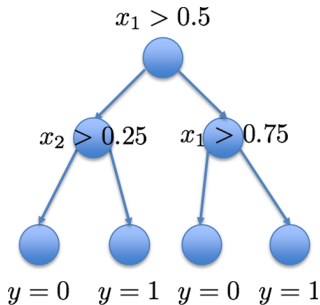
Classification Trees: Why do we need a universal approximator?



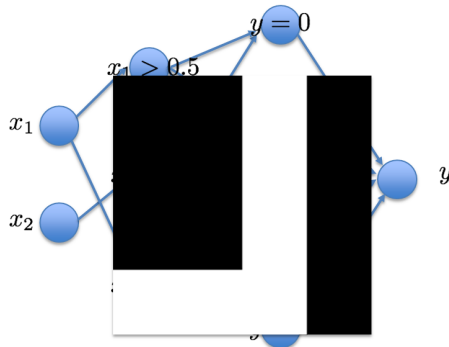
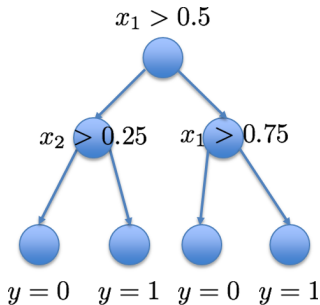
Classification Trees: Why do we need a universal approximator?



Classification Trees: Why do we need a universal approximator?



Classification Trees: Why do we need a universal approximator?



Universal Approximation Theorem

- Let $\varphi(\cdot)$ be a non-constant, bounded and monotonically increasing function.
- For any $\varepsilon > 0$ and any continuous function f defined on a compact subset of \mathbb{R}^m , there exist an integer N , real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ where $i = 1, \dots, N$, such that we can define:

$$F(\mathbf{x}) = \sum_{i=1}^N v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i) \quad \text{with}$$
$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

- We can approximate *any function with just one hidden layer* with a sensible activation function.
- **We have no idea *how*: how many nodes, how to train, ...**



FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

From Activations to Classifications: Softmax Function



Terminology

- So far: ground truth/estimated label is described by $y/\hat{y} \in \{-1, 1\}$.
- Instead, we can use a vector $\mathbf{y} = (y_1, \dots, y_K)^T$ where $K = \# \text{classes}$.
- For exclusive classes, \mathbf{y} looks as follows:

$$y_k = \begin{cases} 1 & \text{if } k \text{ is the index of the true class,} \\ 0 & \text{otherwise} \end{cases}$$

- Called **one-hot encoding**: Only one element is $\neq 0$.
- Classifier output $\hat{\mathbf{y}}$ can represent class probabilities.
- Better descriptor, especially for multi-class problems!

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- $\hat{\mathbf{y}}$ has two properties:

1. $\sum_{k=1}^K \hat{y}_k = 1$
2. $\hat{y}_k \geq 0 \quad \forall \hat{y}_k \in \hat{\mathbf{y}}$

- These are two of Kolmogorov's axioms for a probability distribution.
- This allows to treat the output as normalized probabilities.
- The softmax function is also known as the normalized exponential function.

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- Example: three-class problem



Label	x_k	$\exp(x_k)$	\hat{y}_k
Tiger			
Airplane			
Boat			

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- Example: threefour-class problem



Label	x_k	$\exp(x_k)$	\hat{y}_k
Tiger			
Airplane			
Boat			
Heavy Metal			

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- Example: threefour-class problem



Label	x_k	$\exp(x_k)$	\hat{y}_k
Tiger	-3.44		
Airplane	1.16		
Boat	-0.81		
Heavy Metal	3.91		

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- Example: threefour-class problem



Label	x_k	$\exp(x_k)$	\hat{y}_k
Tiger	-3.44	0.03	
Airplane	1.16	3.19	
Boat	-0.81	0.44	
Heavy Metal	3.91	49.90	

Softmax activation function

- The softmax function rescales a vector \mathbf{x} using:

$$\hat{y}_k = \frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)}$$

- Example: threefour-class problem



Label	x_k	$\exp(x_k)$	\hat{y}_k
Tiger	-3.44	0.03	0.0006
Airplane	1.16	3.19	0.0596
Boat	-0.81	0.44	0.0083
Heavy Metal	3.91	49.90	0.9315

Loss functions

- The cross entropy H of probability distributions \mathbf{p} and \mathbf{q}

$$H(\mathbf{p}, \mathbf{q}) = - \sum_{k=1}^K p_k \log(q_k)$$

- Based on H , we formulate a loss function L :

$$L(\mathbf{y}, \hat{\mathbf{y}}) = - \log(\hat{y}_k) |_{y_k=1}$$

- We will talk more about this during the next session!

"Softmax loss"

- Cross-entropy and the Softmax function exclusively appear together

$$L(\mathbf{y}, \mathbf{x}) = -\log \left(\frac{\exp(x_k)}{\sum_{j=1}^K \exp(x_j)} \right) |_{y_k=1}$$

- One-hot encoding very convenient → represents a histogram
- Naturally handles multiple class problems



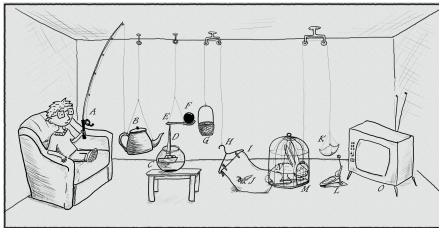
FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Optimization

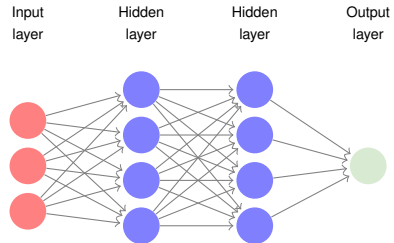


Credit Assignment Problem

- What do those two images have in common?

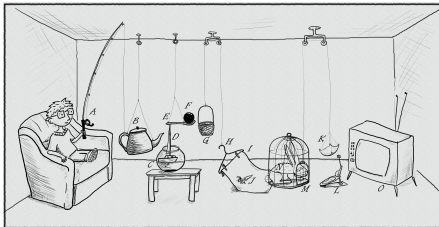


Source: <https://krypt3ia.files.wordpress.com/2011/11/rube.jpg>

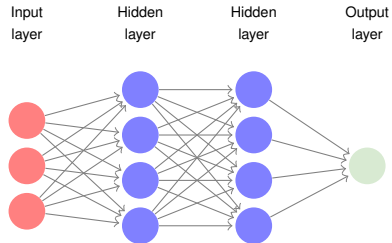


Credit Assignment Problem

- What do those two images have in common?



Source: <https://krypt3ia.files.wordpress.com/2011/11/rube.jpg>



- If it doesn't work it's hard to know which parts to adjust.

Formalization as Optimization Problem

Goal: Find optimal weights \mathbf{w} for all layers:

- Abstract the whole network as a function:

$$L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

- Consider all M training samples:

$$\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}(\mathbf{x}, \mathbf{y})} [L(\mathbf{w}, \mathbf{x}, \mathbf{y})] = \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

- We now know what to do:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \{L(\mathbf{w}, \mathbf{x}, \mathbf{y})\}$$

Gradient Descent

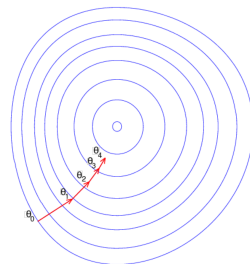
$$\operatorname{argmin}_{\mathbf{w}} \left\{ \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y}) \right\}$$

- Method of choice: Gradient Descent

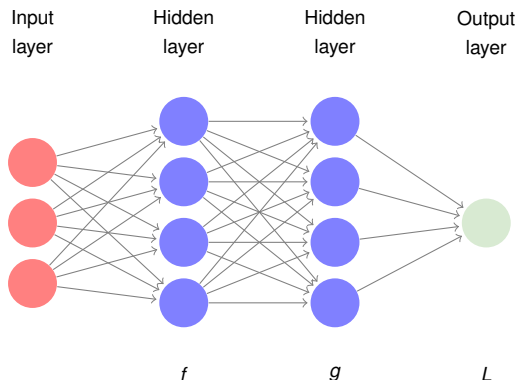
1. Initialize \mathbf{w}
2. Iterate until convergence:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} \frac{1}{M} \sum_{m=1}^M L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$

3. η is commonly referred to as the **learning rate**



What is this L we are trying to optimize?



- Complex network can be seen as composed functions:

$$L(\mathbf{w}, \mathbf{x}, \mathbf{y}) = L(g(f(\mathbf{x}, \mathbf{w}_f), \mathbf{w}_g), \mathbf{y})$$

How to Calculate Derivatives in Complex Neural Networks?

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Two algorithms:

- Finite differences
- Analytic derivative

Finite Differences

Definition of derivative:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Due to finite precision the symmetric definition is preferred:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+\frac{1}{2}h) - f(x-\frac{1}{2}h)}{h}$$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h}$$

Let's calculate it:

- Set h to $2 \cdot 10^{-2}$

$$\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \frac{\left((2(1 + 10^{-2}) + 9)^2 + 3 \right) - \left((2(1 - 10^{-2}) + 9)^2 + 3 \right)}{2 \cdot 10^{-2}}$$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h}$$

Let's calculate it:

- Set h to $2 \cdot 10^{-2}$

$$\begin{aligned} \frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} &= \frac{\left((2(1 + 10^{-2}) + 9)^2 + 3 \right) - \left((2(1 - 10^{-2}) + 9)^2 + 3 \right)}{2 \cdot 10^{-2}} \\ &= \frac{(124.4404 - 123.5604)}{2 \cdot 10^{-2}} \end{aligned}$$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + \frac{1}{2}h) - f(x - \frac{1}{2}h)}{h}$$

Let's calculate it:

- Set h to $2 \cdot 10^{-2}$

$$\begin{aligned} \frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} &= \frac{\left((2(1 + 10^{-2}) + 9)^2 + 3 \right) - \left((2(1 - 10^{-2}) + 9)^2 + 3 \right)}{2 \cdot 10^{-2}} \\ &= \frac{(124.4404 - 123.5604)}{2 \cdot 10^{-2}} \\ &= 43.9999 \end{aligned}$$

Finite Differences Summed up

- For practical use it often suffices to use $h = 1 \cdot 10^{-5}$
- For a more accurate derivative [7] use: $h = \epsilon_f^{\frac{1}{3}} \cdot x_c$
 - Where $\epsilon_f \approx 10^{-7}$
 - The characteristic scale is approximated as $x_c = x$
 - Prevent division by zero at $x = 0$

Conclusion

- **Easy** to use
- We only need to be able to **evaluate** functions
- Computationally **inefficient**
- **Frequently used** to check implementations

Analytic gradient

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Four analytic rules:

1. $\frac{d}{dx} \text{const} = 0$
2. Linearity: $\frac{d}{dx}$ is a linear operator
3. Monomials: $\frac{d}{dx} x^n = n \cdot x^{n-1}$
4. Chain rule: $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Let's calculate it:

1. $\frac{d}{dx} \text{const} = 0$
2. $\frac{d}{dx}$ is linear
3. $\frac{d}{dx} x^n = n \cdot x^{n-1}$
4. $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Let's calculate it:

$$\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \frac{\partial}{\partial x_1} (2x_1 + 9)^2$$

1. $\frac{d}{dx} \text{const} = 0$
2. $\frac{d}{dx}$ is linear
3. $\frac{d}{dx} x^n = n \cdot x^{n-1}$
4. $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Rules 1 and 2

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \left(\begin{pmatrix} 1 \\ 3 \end{pmatrix} \right)$

Let's calculate it:

$$\begin{aligned} \frac{\partial}{\partial x_1} f \left(\begin{pmatrix} 1 \\ 3 \end{pmatrix} \right) &= \frac{\partial}{\partial x_1} (2x_1 + 9)^2 \\ &= \frac{\partial}{\partial z} (z)^2 \frac{\partial}{\partial x_1} (2x_1 + 9) \end{aligned}$$

1. $\frac{d}{dx} \text{const} = 0$
2. $\frac{d}{dx}$ is linear
3. $\frac{d}{dx} x^n = n \cdot x^{n-1}$
4. $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Rules 1 and 2

Rule 4 and $2x_1 + 9 = z$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Let's calculate it:

$$\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \frac{\partial}{\partial x_1} (2x_1 + 9)^2$$

$$= \frac{\partial}{\partial z} (z)^2 \frac{\partial}{\partial x_1} (2x_1 + 9)$$

$$= 2(2x_1 + 9) \frac{\partial}{\partial x_1} (2x_1 + 9)$$

1. $\frac{d}{dx} \text{const} = 0$

2. $\frac{d}{dx}$ is linear

3. $\frac{d}{dx} x^n = n \cdot x^{n-1}$

4. $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Rules 1 and 2

Rule 4 and $2x_1 + 9 = z$

Rule 3

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

Let's calculate it:

$$\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix} = \frac{\partial}{\partial x_1} (2x_1 + 9)^2$$

$$= \frac{\partial}{\partial z} (z)^2 \frac{\partial}{\partial x_1} (2x_1 + 9)$$

$$= 2(2x_1 + 9) \frac{\partial}{\partial x_1} (2x_1 + 9)$$

$$= 2(2x_1 + 9) \cdot 2 = 44$$

1. $\frac{d}{dx} \text{const} = 0$

2. $\frac{d}{dx}$ is linear

3. $\frac{d}{dx} x^n = n \cdot x^{n-1}$

4. $\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \cdot \frac{d}{dx} g(x)$

Rules 1 and 2

Rule 4 and $2x_1 + 9 = z$

Rule 3

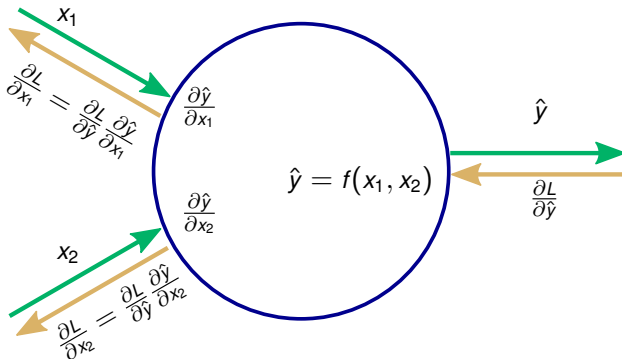
Rules 1 and 2 and $x_1 = 1$

Analytic Gradient Summed up

- **Chain rule** and **Linearity** enable to **decompose** complex functions
- Analytic formulas have to be calculated manually
- Computationally more **efficient** than finite differences

Can we compute analytic gradients automatically?

Backpropagation Algorithm



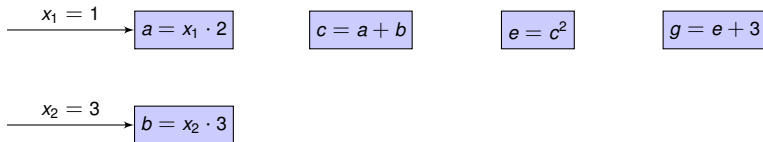
1. Forward pass: Compute activations
2. Backward pass: Recursively apply chain rule

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$

$$\frac{d}{dx}f(g(x)) = \frac{d}{dg}f(g) \frac{d}{dx}g(x)$$

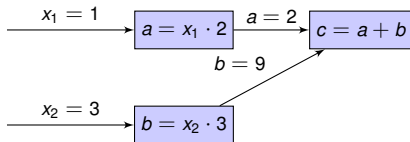
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$



Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$



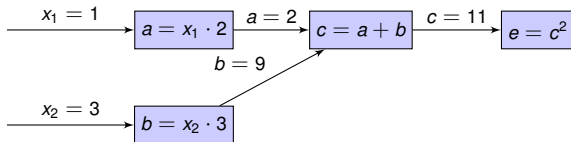
$$e = c^2$$

$$g = e + 3$$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$

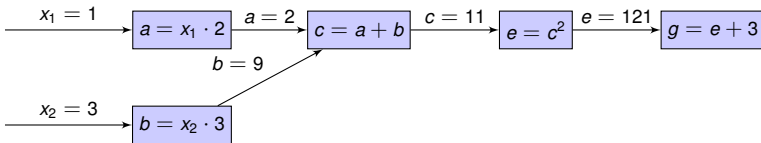


$$g = e + 3$$

Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$

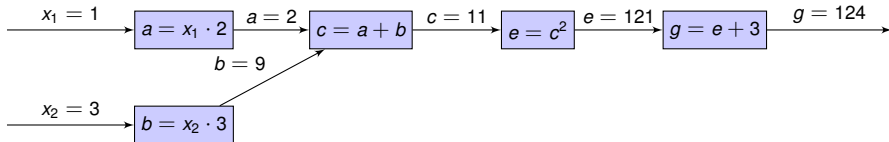


Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$

$$\frac{d}{dx}f(g(x)) = \frac{d}{dg}f(g) \frac{d}{dx}g(x)$$

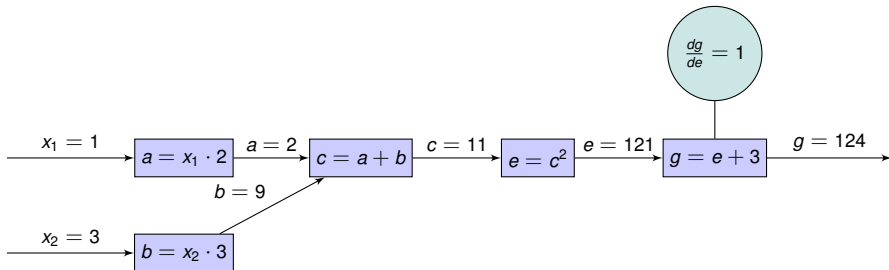
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$



Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

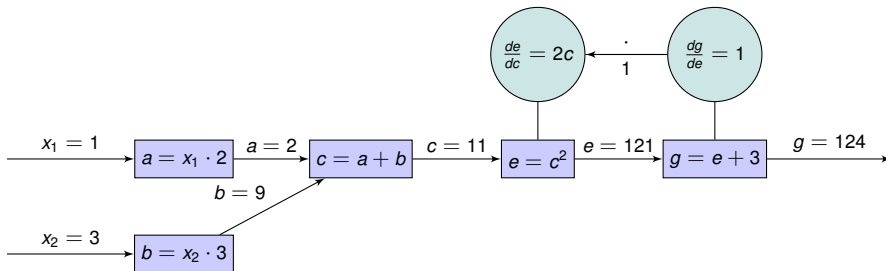
$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$



Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$

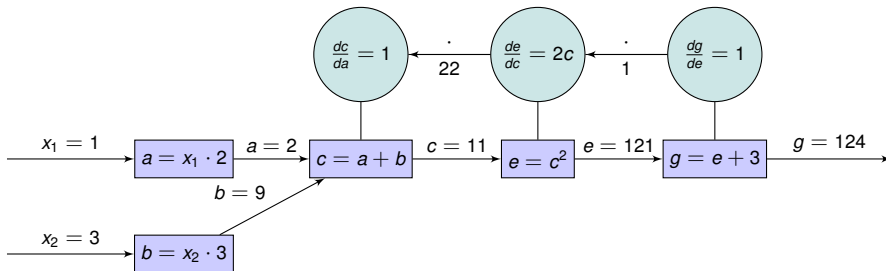


Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$

- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$

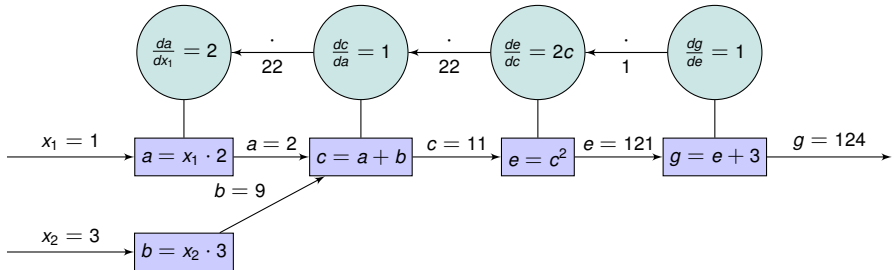


Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$

- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

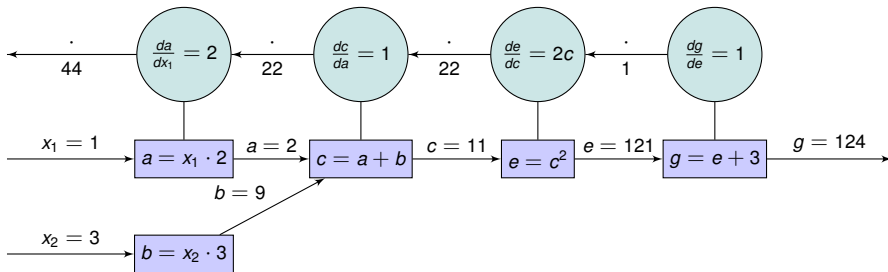


Example Problem

- Function: $\hat{y} = f(\mathbf{x}) = (2x_1 + 3x_2)^2 + 3$

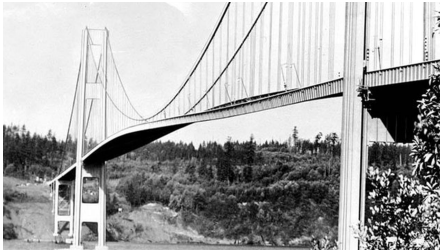
- Evaluate $\frac{\partial}{\partial x_1} f \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g) \frac{d}{dx} g(x)$$

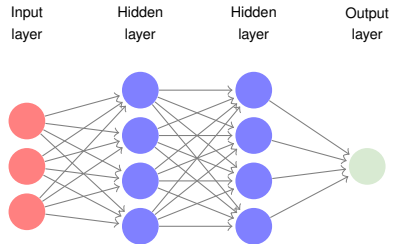


Stability Problem

- What do those two images have in common?

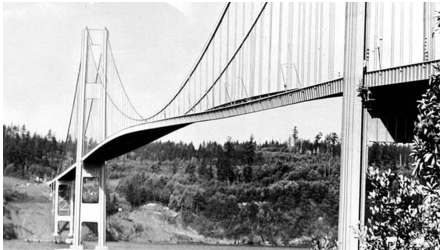


Click for video

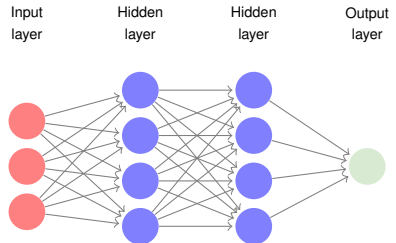


Stability Problem

- What do those two images have in common?

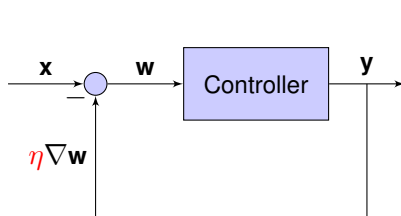


Click for video

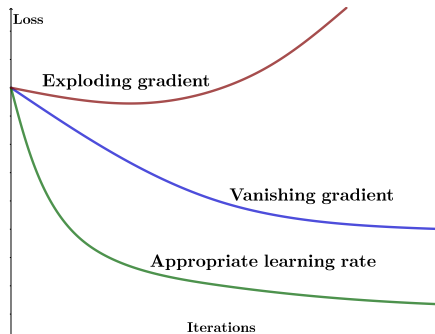


- Both suffer from positive feedback!
- This can cause disaster

Feedback loop



Analogy to control theory



- If η is too high \rightarrow **positive feedback** \rightarrow loss grows **without bounds**
- If η is too small \rightarrow **negative feedback** \rightarrow **gradient vanishes**
- Choice of η is **critical** for learning

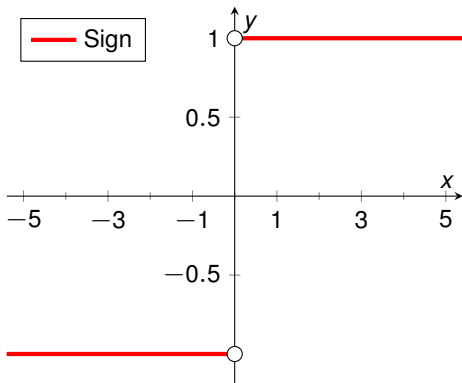
Backpropagation Summed up

- Built around the **chain rule**
- Uses a **forward-pass** through the function
- Computationally very **efficient** by using a **dynamic programming** approach
- Is **no training algorithm**, because it just computes a gradient

Consequences

- Product of partials \rightarrow numerical **errors multiply**
- Product of partials \rightarrow **vanishing** or **exploding** gradient

About the sign Activation Function



Sign

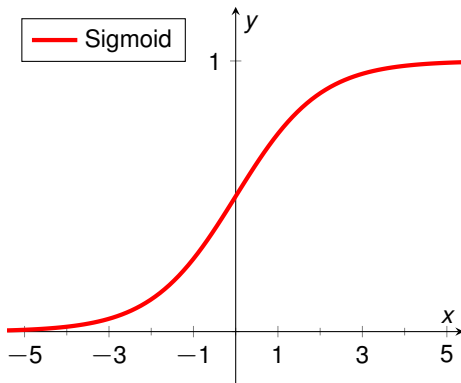
$$f(x) = \begin{cases} +1 & \text{for } x \geq 0 \\ -1 & \text{for } x < 0 \end{cases}$$

$$f'(x) = 2\delta(x)$$

+ Normalized output

— Gradient vanishes almost everywhere!

Smooth Activation Function



Sigmoid (logistic function)

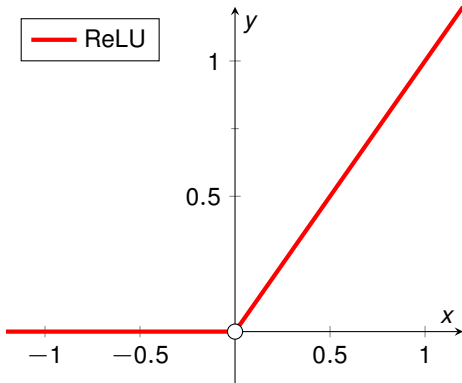
$$f(x) = \frac{1}{1 + \exp(-x)}$$

$$f'(x) = f(x)(1 - f(x))$$

+ Normalized output

— Gradient still eventually vanishes

Piecewise-linear Activation Function



Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

+ Less vanishing gradient



FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Layer Abstraction



From Graphs of Nodes to Graphs of Layers

- We introduced **layers** but computed everything on individual nodes
- It is convenient to add further abstraction
- But how can we express this?

Recall: Single neuron

- Add a bias unit to $\mathbf{x} \in \mathbb{R}^{N-1}$ by adding a dimension with $x_n = 1$
- This is a connection from every input element to the single output element:

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

Representing the connections

- Assume we have M neurons $\rightarrow M$ sets of weights: \mathbf{w}_m for $m \in \{1, \dots, M\}$

$$\hat{y}_m = \mathbf{w}_m^T \mathbf{x}$$

- We rewrite this operation as matrix-vector multiplication:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$$

- This is known as **fully connected layer**.
- It represents any arbitrary connection topology between layers.
- We can describe back-propagation in this more abstract view as well!

Fully Connected Layer

$$\begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \end{pmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

- The forward-pass is:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$$

- After the forward-pass through all layers, we can compute a loss that depends on our loss function L .
- We need two gradients for the backward-pass:
 - Gradient with respect to the weights: $\frac{\partial L}{\partial \mathbf{W}}$ for gradient descent
 - Gradient with respect to the inputs: $\frac{\partial L}{\partial \mathbf{x}}$ for backpropagation

Fully Connected Layer Summed up

- Can represent any connection topology
- Enables higher level view concentrating on layers instead of nodes
- Is a matrix multiplication:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}$$

- Its gradient with respect to the weights:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \mathbf{x}^T ;$$

- Its gradient with respect to the input:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}} = \mathbf{W}^T \frac{\partial L}{\partial \hat{\mathbf{y}}} ;$$

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = Wx$$

- Associated loss function:
- Gradient:

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = \mathbf{W}x$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|\mathbf{W}x - y\|_2^2$$

- Gradient:

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = \mathbf{W}x$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|\mathbf{W}x - y\|_2^2$$

- Gradient:

$$\frac{\partial L}{\partial \mathbf{W}} =$$

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = \mathbf{W}x$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|\mathbf{W}x - y\|_2^2$$

- Gradient:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{f}}$$

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = \mathbf{W}x$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|\mathbf{W}x - y\|_2^2$$

- Gradient:

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \hat{f}} \frac{\partial \hat{f}}{\partial \mathbf{W}}$$

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = \mathbf{W}x$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|\mathbf{W}x - y\|_2^2$$

- Gradient:

$$\frac{\partial L}{\partial \mathbf{W}} = \underbrace{\frac{\partial L}{\partial \hat{f}}}_{(\mathbf{W}x - y)} \frac{\partial \hat{f}}{\partial \mathbf{W}}$$

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = Wx$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|Wx - y\|_2^2$$

- Gradient:

$$\frac{\partial L}{\partial W} = \underbrace{\frac{\partial L}{\partial \hat{f}}}_{(Wx - y)} \underbrace{\frac{\partial \hat{f}}{\partial W}}_{(x^\top)}$$

Back-propagation in a nut-shell

- Matrix multiplication as network:

$$\hat{y} = \hat{f}(x) = Wx$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\hat{f}(x) - y\|_2^2 = \frac{1}{2} \|Wx - y\|_2^2$$

- Gradient:

$$\frac{\partial L}{\partial W} = \underbrace{\frac{\partial L}{\partial \hat{f}}}_{(Wx - y)} \underbrace{\frac{\partial \hat{f}}{\partial W}}_{\cdot (x^\top)} = (Wx - y)(x^\top)$$

Back-propagation in a nut-shell (2)

- Let's add some layers:

$$\hat{y} = \hat{f}_3(\hat{f}_2(\hat{f}_1(x))) = \tilde{W}_3 \tilde{W}_2 \tilde{W}_1 \hat{x}$$

- Associated loss function:
- Gradients?

Back-propagation in a nut-shell (2)

- Let's add some layers:

$$\hat{y} = \hat{f}_3(\hat{f}_2(\hat{f}_1(x))) = \tilde{W}_3 W_2 W_1 \hat{x}$$

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Gradients?

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Last layer gradient

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y}\|_2^2$$

- Last layer gradient

$$\frac{\partial L}{\partial \mathbf{W}_3} =$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Last layer gradient

$$\frac{\partial L}{\partial \mathbf{W}_3} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_3}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Last layer gradient

$$\frac{\partial L}{\partial \mathbf{W}_3} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_3}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Last layer gradient

$$\frac{\partial L}{\partial \mathbf{W}_3} = \underbrace{\frac{\partial L}{\partial \hat{\mathbf{f}}_3}}_{(\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y})} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_3}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Last layer gradient

$$\frac{\partial L}{\partial \mathbf{W}_3} = \underbrace{\frac{\partial L}{\partial \hat{\mathbf{f}}_3}}_{(\mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y})} \cdot \underbrace{\frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_3}}_{(\mathbf{W}_2 \mathbf{W}_1 \mathbf{x})^\top}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Last layer gradient

$$\frac{\partial L}{\partial W_3} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial W_3}}_{(W_2 W_1 x)^\top} = (W_3 W_2 W_1 x - y)(W_2 W_1 x)^\top$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Deeper gradients

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial \mathbf{W}_2} =$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_2} =$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \| \mathbf{W}_3 \mathbf{W}_2 \mathbf{W}_1 \mathbf{x} - \mathbf{y} \|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \hat{\mathbf{f}}_3} \frac{\partial \hat{\mathbf{f}}_3}{\partial \hat{\mathbf{f}}_2} \frac{\partial \hat{\mathbf{f}}_2}{\partial \mathbf{W}_2}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_2}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \frac{\partial \hat{f}_2}{\partial W_2}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \cdot \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{\cdot (W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{\cdot (W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top$$

$$\frac{\partial L}{\partial W_1} =$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} =$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} =$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{\cdot (W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \end{aligned}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial \hat{f}_1} \end{aligned}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial \hat{f}_1} \frac{\partial \hat{f}_1}{\partial W_1} \end{aligned}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{\cdot (W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial \hat{f}_1} \frac{\partial \hat{f}_1}{\partial W_1} \end{aligned}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|^2$$

- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \frac{\partial \hat{f}_2}{\partial \hat{f}_1} \frac{\partial \hat{f}_1}{\partial W_1} \end{aligned}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{(W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial \hat{f}_1}}_{(W_2)^\top} \frac{\partial \hat{f}_1}{\partial W_1} \end{aligned}$$

Back-propagation in a nut-shell (3)

- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|_2^2$$

- Deeper gradients

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{\cdot (W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial \hat{f}_1}}_{(W_2)^\top} \underbrace{\frac{\partial \hat{f}_1}{\partial W_1}}_{\cdot (x)^\top}$$

Back-propagation in a nut-shell (3)

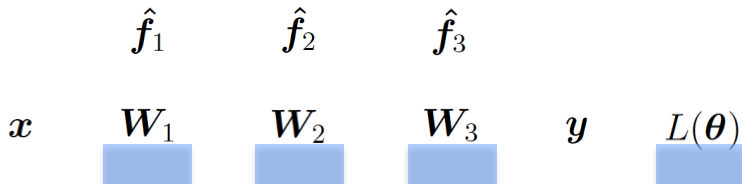
- Associated loss function:

$$L(\theta) = \frac{1}{2} \|W_3 W_2 W_1 x - y\|^2$$

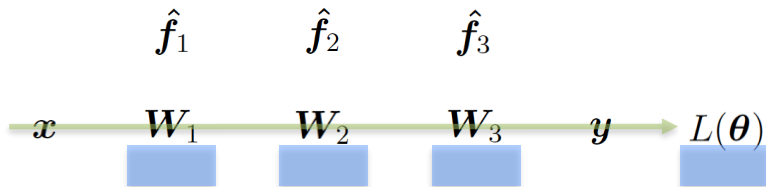
- Deeper gradients

$$\begin{aligned} \frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_2} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial W_2}}_{\cdot (W_1 x)^\top} = W_3^\top (W_3 W_2 W_1 x - y) (W_1 x)^\top \\ \frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial W_1} = \frac{\partial L}{\partial \hat{f}_3} \frac{\partial \hat{f}_3}{\partial \hat{f}_2} \frac{\partial \hat{f}_2}{\partial W_1} = \underbrace{\frac{\partial L}{\partial \hat{f}_3}}_{(W_3 W_2 W_1 x - y)} \underbrace{\frac{\partial \hat{f}_3}{\partial \hat{f}_2}}_{(W_3)^\top} \underbrace{\frac{\partial \hat{f}_2}{\partial \hat{f}_1}}_{(W_2)^\top} \underbrace{\frac{\partial \hat{f}_1}{\partial W_1}}_{\cdot (x)^\top} \\ &= W_2^\top W_3^\top (W_3 W_2 W_1 x - y) (x)^\top \end{aligned}$$

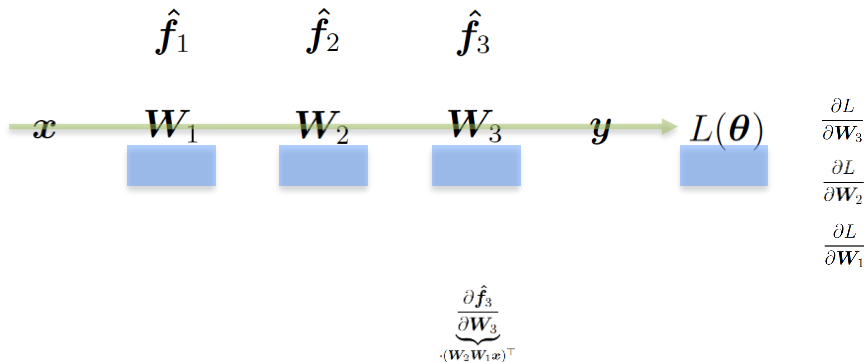
Back-propagation in a nut-shell (4)



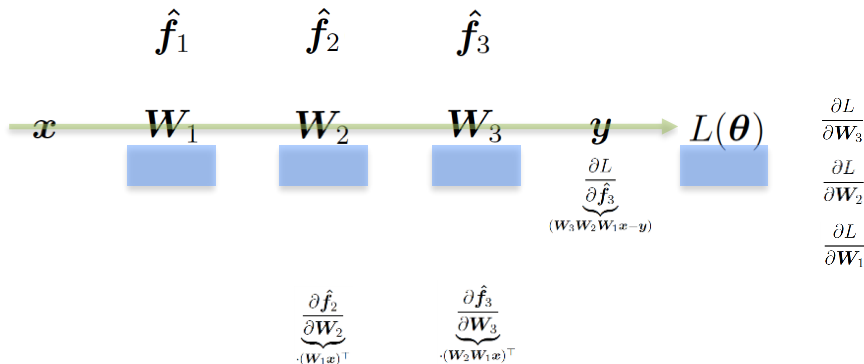
Back-propagation in a nut-shell (4)



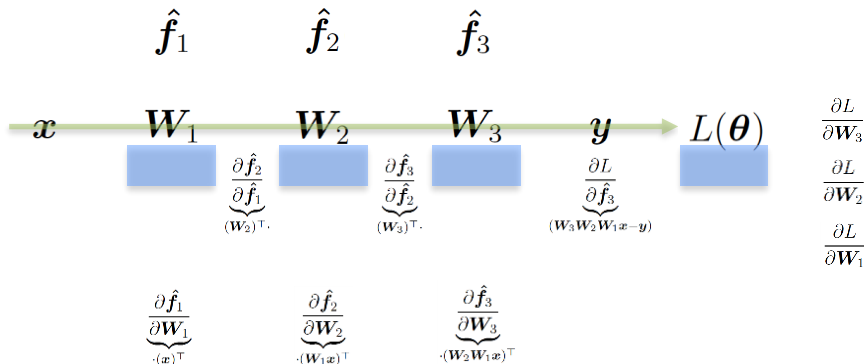
Back-propagation in a nut-shell (4)



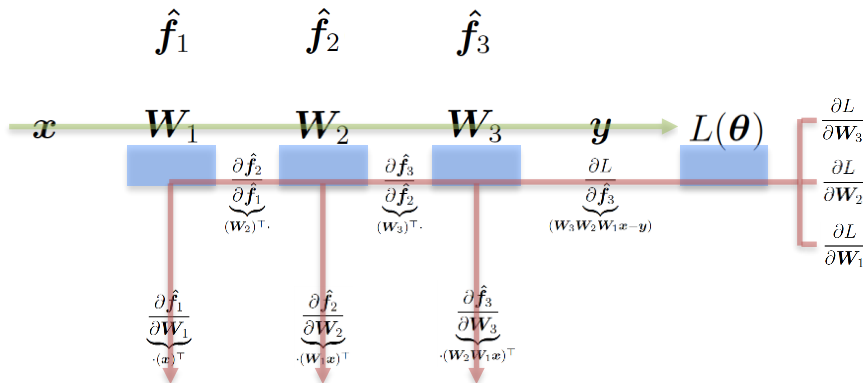
Back-propagation in a nut-shell (4)



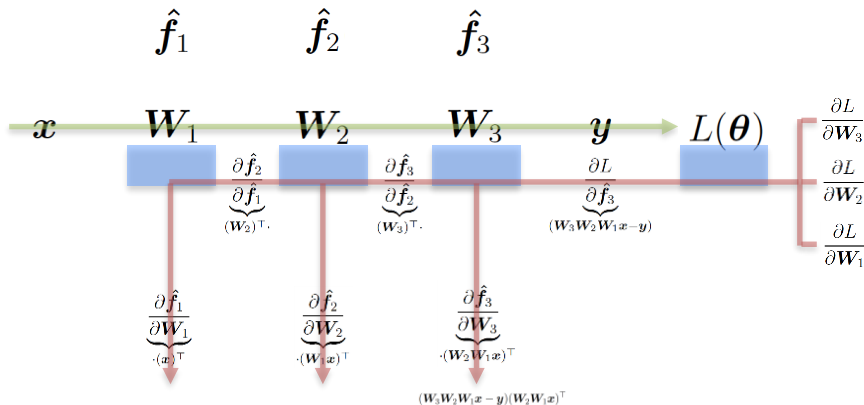
Back-propagation in a nut-shell (4)



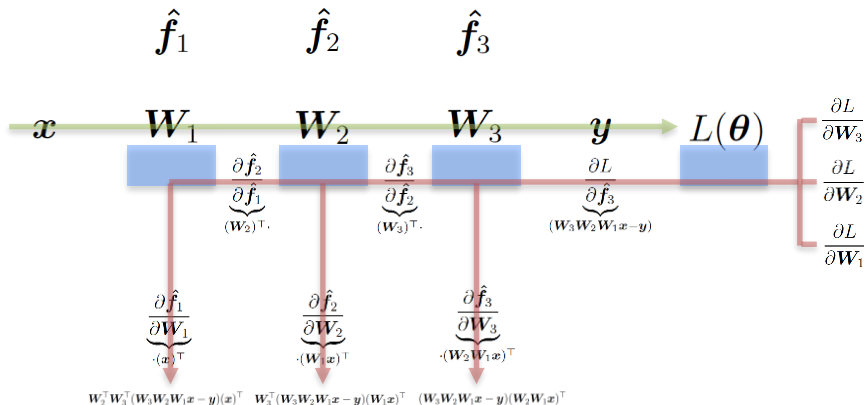
Back-propagation in a nut-shell (4)



Back-propagation in a nut-shell (4)



Back-propagation in a nut-shell (4)



Summary

- **Softmax activation** function with **cross entropy loss** mostly go together as "Softmax Loss".
- **Gradient descent** is our default training algorithm in deep learning.
- We can compute gradients using **finite differences** to check our implementation.
- We use the **backpropagation** algorithm to compute gradients efficiently.
- The **fully connected** layer is the most general connectivity between layers in a feed-forward neural network.

NEXT TIME

ON DEEP LEARNING

- Problem adapted loss functions
- Sophisticated optimization routines
- Optimization adapted to the needs of every single parameter
- An argument why neural networks shouldn't perform well
- Some very recent insights why they do perform well

Comprehensive Questions

- Name a loss function for multi-class classification in deep learning.
- Explain how this loss function works.
- How can you check if the derivative implementation of a loss function is correct?
- What does backpropagation do?
- How does backpropagation work?
- Explain the exploding and vanishing gradient problems.
- Why is the signum function not used in deep learning?

Further Reading

- [Link](#) - The original paper introducing ReLUs
- [Link](#) - The original paper introducing backpropagation
- [Link](#) - Bishop - Mathematical compendium for machine learning
- [Link](#) - Blog article putting backpropagation in a very general context



FRIEDRICH-ALEXANDER-
UNIVERSITÄT
ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

References



References I

- [1] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. John Wiley and Sons, inc., 2000.
- [2] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
- [3] F. Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.”. In: Psychological Review 65.6 (1958), pp. 386–408.
- [4] WS. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity.”. In: Bulletin of mathematical biophysics 5 (1943), pp. 99–115.

References II

- [5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors.”. In: Nature 323 (1986), pp. 533–536.
- [6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence Vol. 15. 2011, pp. 315–323.
- [7] William H. Press, Saul A. Teukolsky, William T. Vetterling, et al. Numerical Recipes 3rd Edition: The Art of Scientific Computing. 3rd ed. New York, NY, USA: Cambridge University Press, 2007.