

# Advanced Machine Learning from Theory to Practice

## Lecture 08

### Introduction to Neural Networks

E. Le Pennec - École polytechnique

Fall 2016

## 1 Introduction

## 2 Supervised Learning

## 3 (Deep) Neural Networks

## 4 Perceptron

## 5 Multilayer Perceptron

## 6 Backprop Algorithm

## Spam detection (Text classification)



- Data: email collection
- Input: email
- Output : Spam or No Spam

## Face Detection

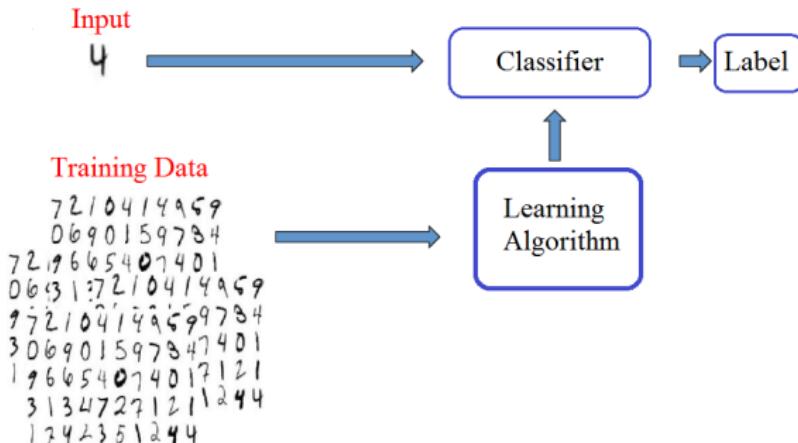


- Data: Annotated database of images
- Input : Sub window in the image
- Output : Presence or no of a face...

## Number Recognition

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 |
| 7 | 7 | 7 | 7 | 7 |
| 8 | 8 | 8 | 8 | 8 |
| 9 | 9 | 9 | 9 | 9 |

- Data: Annotated database of images (each image is represented by a vector of  $28 \times 28 = 784$  pixel intensities)
- Input: Image
- Output: Corresponding number



A definition by Tom Mitchell  
(<http://www.cs.cmu.edu/~tom/>)

A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

- 1 Introduction
- 2 Supervised Learning
- 3 (Deep) Neural Networks
- 4 Perceptron
- 5 Multilayer Perceptron
- 6 Backprop Algorithm

1 Introduction

2 Supervised Learning

3 (Deep) Neural Networks

4 Perceptron

5 Multilayer Perceptron

6 Backprop Algorithm

## Supervised Learning Framework

- Input measurement  $\mathbf{X} = (X^{(1)}, X^{(2)}, \dots, X^{(d)}) \in \mathcal{X}$
- Output measurement  $Y \in \mathcal{Y}$ .
- $(\mathbf{X}, Y) \sim \mathbf{P}$  with  $\mathbf{P}$  unknown.
- **Training data** :  $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)\}$  (i.i.d.  $\sim \mathbf{P}$ )
- Often
  - $\mathbf{X} \in \mathbb{R}^d$  and  $Y \in \{-1, 1\}$  (classification)
  - or  $\mathbf{X} \in \mathbb{R}^d$  and  $Y \in \mathbb{R}$  (regression).
- A **classifier** is a function in  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \mathcal{Y} \text{ measurable}\}$

## Goal

- Construct a **good** classifier  $\hat{f}$  from the training data.
- Need to specify the meaning of **good**.
- Formally, classification and regression are the same problem!

## Loss function

- **Loss function** :  $\ell(y, f(x))$  measure how well  $f(x)$  “predicts”  $y$ .
- Examples:
  - Prediction loss:  $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$
  - Quadratic loss:  $\ell(Y, \mathbf{X}) = |Y - f(\mathbf{X})|^2$

## Risk of a generic classifier

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E} [\ell(Y, f(\mathbf{X}))] = \mathbb{E}_{\mathbf{X}} \left[ \mathbb{E}_{Y|\mathbf{X}} [\ell(Y, f(\mathbf{X}))] \right]$$

- Examples:
  - Prediction loss:  $\mathbb{E} [\ell(Y, f(\mathbf{X}))] = \mathbb{P} \{ Y \neq f(\mathbf{X}) \}$
  - Quadratic loss:  $\mathbb{E} [\ell(Y, f(\mathbf{X}))] = \mathbb{E} [|Y - f(\mathbf{X})|^2]$

- **Beware:** As  $\hat{f}$  depends on  $\mathcal{D}_n$ ,  $\mathcal{R}(\hat{f})$  is a random variable!

## Experience, Task and Performance measure

- **Training data** :  $\mathcal{D} = \{(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)\}$  (i.i.d.  $\sim \mathbf{P}$ )
- **Predictor**:  $f : \mathcal{X} \rightarrow \mathcal{Y}$  measurable
- **Cost/Loss function** :  $\ell(Y, f(\mathbf{X}))$  measure how well  $f(\mathbf{X})$  “predicts”  $Y$
- **Risk**:

$$\mathcal{R}(f) = \mathbb{E} [\ell(Y, f(\mathbf{X}))] = \mathbb{E}_{\mathbf{X}} \left[ \mathbb{E}_{Y|\mathbf{X}} [\ell(Y, f(\mathbf{X}))] \right]$$

- Often  $\ell(Y, f(\mathbf{X})) = |f(\mathbf{X}) - Y|^2$  or  $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$

## Goal

- Learn a rule to construct a **classifier**  $\hat{f} \in \mathcal{F}$  from the training data  $\mathcal{D}_n$  s.t. **the risk**  $\mathcal{R}(\hat{f})$  is **small on average** or with high probability with respect to  $\mathcal{D}_n$ .

- The best solution  $f^*$  (which is independent of  $\mathcal{D}_n$ ) is

$$f^* = \arg \min_{f \in \mathcal{F}} R(f) = \arg \min_{f \in \mathcal{F}} \mathbb{E} [\ell(Y, f(\mathbf{X}))] = \arg \min_{f \in \mathcal{F}} \mathbb{E}_{\mathbf{X}} \left[ \mathbb{E}_{Y|\mathbf{X}} [\ell(Y, f(\mathbf{x}))] \right]$$

## Bayes Classifier (explicit solution)

- In binary classification with 0 – 1 loss:

$$f^*(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}\{Y = +1 | \mathbf{X}\} \geq \mathbb{P}\{Y = -1 | \mathbf{X}\} \\ & \Leftrightarrow \mathbb{P}\{Y = +1 | \mathbf{X}\} \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$

- In regression with the quadratic loss

$$f^*(\mathbf{X}) = \mathbb{E}[Y | \mathbf{X}]$$

**Issue:** Explicit solution requires to know  $\mathbb{E}[Y | \mathbf{X}]$  for all values of  $\mathbf{X}$ !

## Machine Learning

- Learn a rule to construct a **classifier**  $\hat{f} \in \mathcal{F}$  from the training data  $\mathcal{D}_n$  s.t. **the risk**  $\mathcal{R}(\hat{f})$  is **small on average** or with high probability with respect to  $\mathcal{D}_n$ .

## Canonical example: Empirical Risk Minimizer

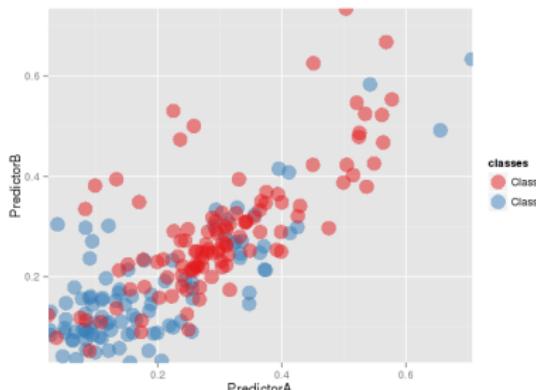
- One restricts  $f$  to a subset of functions  $\mathcal{S} = \{f_\theta, \theta \in \Theta\}$
- One replaces the minimization of the average loss by the minimization of the empirical loss

$$\hat{f} = \hat{f}_\theta = \operatorname{argmin}_{f_\theta, \theta \in \Theta} \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f_\theta(\mathbf{X}_i))$$

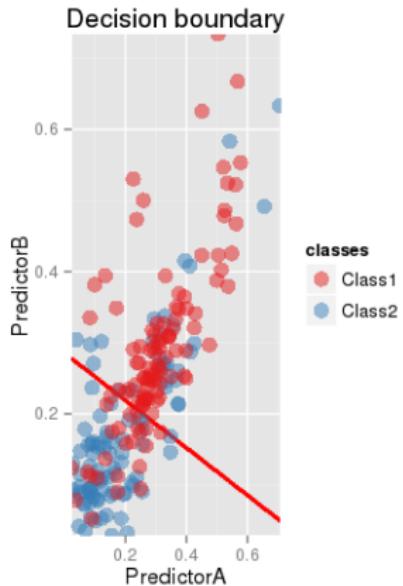
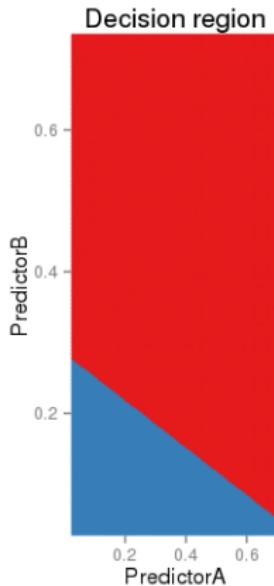
- Examples:
    - Linear regression
    - Linear discrimination with
- $$\mathcal{S} = \{\mathbf{x} \mapsto \operatorname{sign}\{\beta^T \mathbf{x} + \beta_0\} / \beta \in \mathbb{R}^d, \beta_0 \in \mathbb{R}\}$$

## Synthetic Dataset

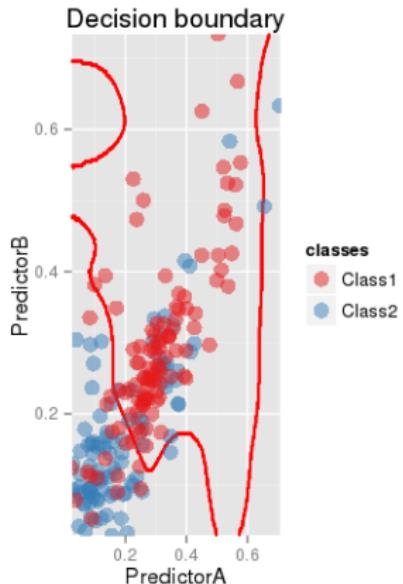
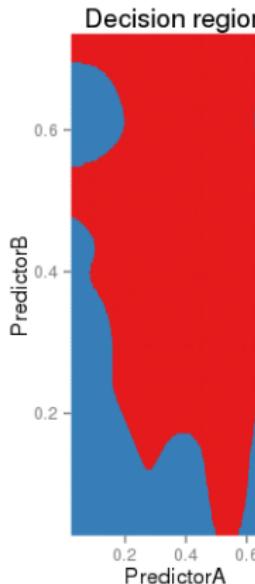
- Two features/covariates.
- Two classes.
- Dataset from *Applied Predictive Modeling*, M. Kuhn and K. Johnson, Springer
- Numerical experiments with **R** and the **caret** package.



# Example: Linear Discrimination

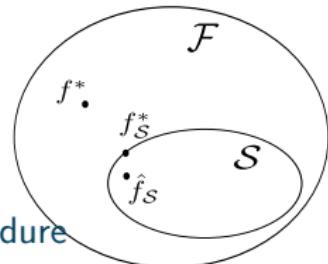


# Example: More complex model



- General setting:

- $\mathcal{F} = \{\text{measurable functions } \mathcal{X} \rightarrow \mathcal{Y}\}$
- Best solution:  $f^* = \operatorname{argmin}_{f \in \mathcal{F}} \mathcal{R}(f)$
- Class  $\mathcal{S} \subset \mathcal{F}$  of functions
- Ideal target in  $\mathcal{S}$ :  $f_S^* = \operatorname{argmin}_{f \in \mathcal{S}} \mathcal{R}(f)$
- Estimate in  $\mathcal{S}$ :  $\hat{f}_S$  obtained with some procedure



## Approximation error and estimation error (Bias/Variance)

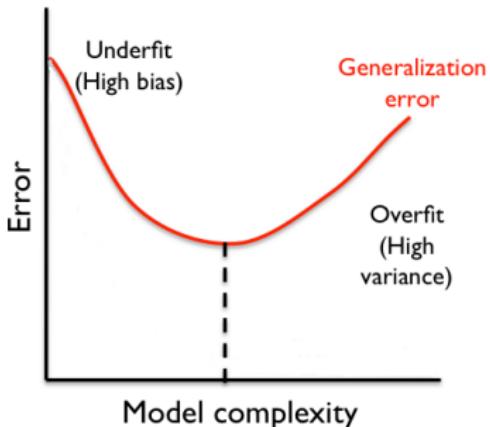
$$\mathcal{R}(\hat{f}_S) - \mathcal{R}(f^*) = \underbrace{\mathcal{R}(f_S^*) - \mathcal{R}(f^*)}_{\text{Approximation error}} + \underbrace{\mathcal{R}(\hat{f}_S) - \mathcal{R}(f_S^*)}_{\text{Estimation error}}$$

- Approx. error can be large if the model  $\mathcal{S}$  is not suitable.
- Estimation error can be large if the model is complex.

## Agnostic approach

- No assumption (so far) on the law of  $(\mathbf{X}, Y)$ .

# Under-fitting / Over-fitting Issue



- Different behavior for different model complexity
- Low complexity model are easily learned but the approximation error ("bias") may be large (**Under-fit**).
- High complexity model may contains a good ideal target but the estimation error ("variance") can be large (**Over-fit**)

Bias-variance trade-off  $\iff$  avoid overfitting and underfitting

# Statistical and Optimization Point of View Framework

Supervised Learning



How to find a good function  $f$  with a *small* risk

$$R(f) = \mathbb{E} [\ell(Y, f(X))] \quad ?$$

Canonical approach:  $\hat{f}_S = \operatorname{argmin}_{f \in S} \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f(\mathbf{X}_i))$

## Problems

- How to choose  $S$ ?
- How to compute the minimization?

## A Statistical Point of View

**Solution:** For  $\mathbf{X}$ , estimate  $Y|\mathbf{X}$  plug this estimate in the Bayes classifier: (Generalized) Linear Models, Kernel methods,  $k$ -nn, Naive Bayes, Tree, Bagging...

## An Optimization Point of View

**Solution:** If necessary replace the loss  $\ell$  by an upper bound  $\ell'$  and minimize the empirical loss: SVR, SVM, Neural Network, Tree, Boosting

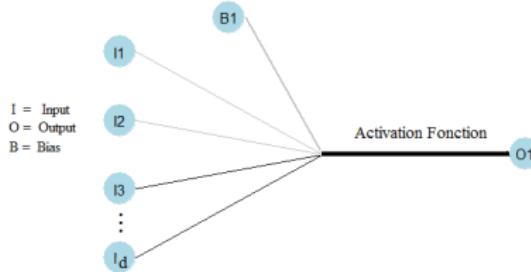
- 1 Introduction
- 2 Supervised Learning
- 3 (Deep) Neural Networks
- 4 Perceptron
- 5 Multilayer Perceptron
- 6 Backprop Algorithm

# Artificial Neuron and Logistic Regression

(Deep) Neural Networks



Activation Neuron Configuration

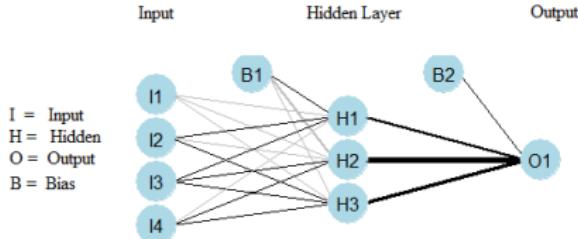


## Artificial neuron

- Structure:
  - Mix inputs with a weighted sum,
  - Apply a (non linear) activation function to this sum,
  - Eventually threshold the result to make a decision.
- Weights learned by minimizing a loss function.

## Logistic unit

- Structure:
  - Mix inputs with a weighted sum,
  - Apply the logistic function  $\sigma(t) = e^t / (1 + e^t)$ ,
  - Threshold at 1/2 to make a decision!
- Logistic weights learned by minimizing the -log-likelihood.



## Neural network structure

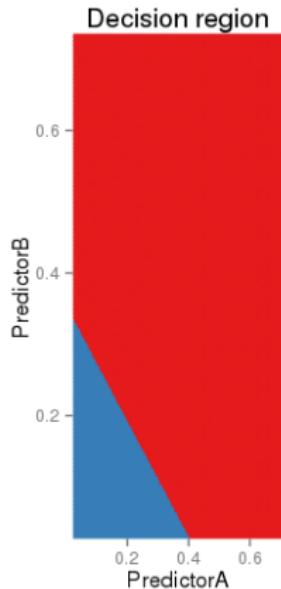
- Cascade of artificial neurons organized in layers
- Thresholding decision only at the output layer
- Most classical case use logistic neurons and the -log-likelihood as the criterion to minimize.
- Classical (stochastic) gradient descent algorithm (Back propagation)
- Non convex and thus may be trapped in local minima.

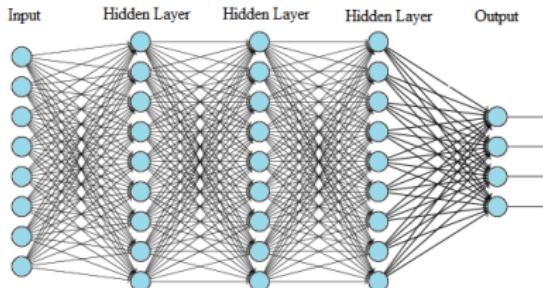
# Neural network

(Deep) Neural Networks



Neural Network





## Deep Neural Network structure

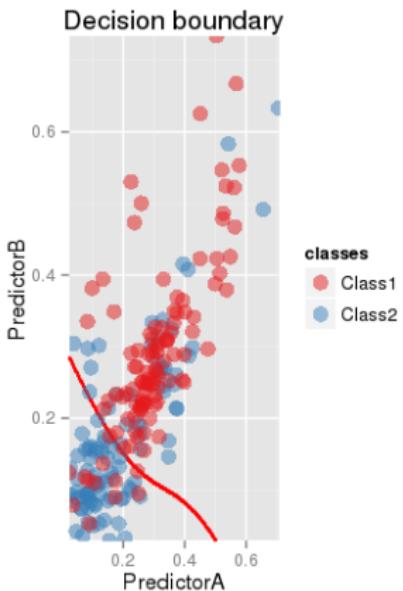
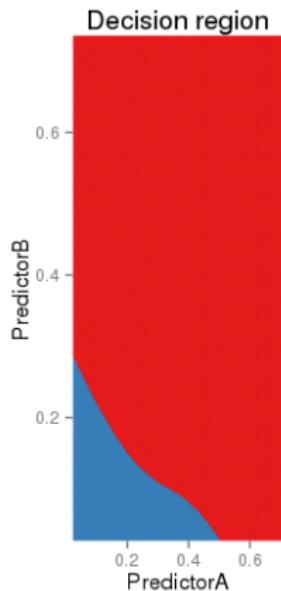
- Deep cascade of layers!
- No conceptual novelty optimization becomes a crucial issue.
- Clever initialization, clever stochastic optimization, clever use of GPU...
- Very impressive results!

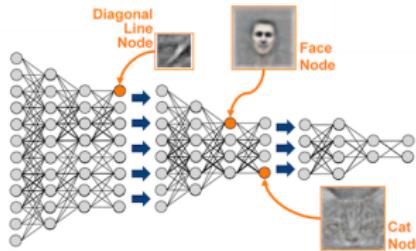
# Deep Neural Network

(Deep) Neural Networks



H2O NN





Family of Machine Learning algorithm combining:

- a (deep) multilayered structure,
  - a clever (often unsupervised) initialization,
  - a clever stochastic fine tuning optimization.
- 
- Examples: Deep Neural Network, Deep (Restricted) Boltzman Machine, Stacked Encoder...
  - Appears to be very efficient but lack of theoretical fundation!

# Outline

Perceptron



1 Introduction

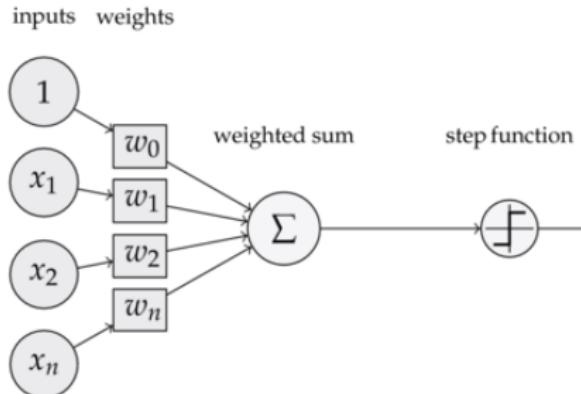
2 Supervised Learning

3 (Deep) Neural Networks

4 Perceptron

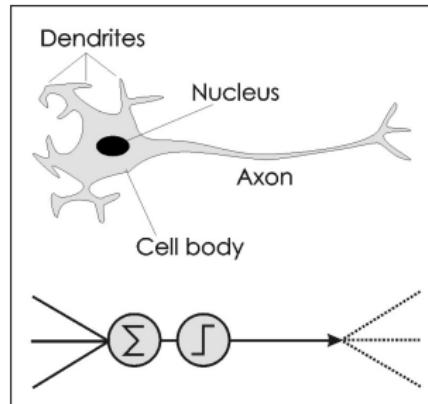
5 Multilayer Perceptron

6 Backprop Algorithm



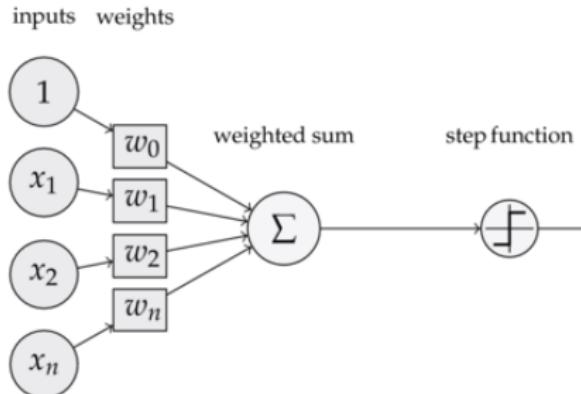
## Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.



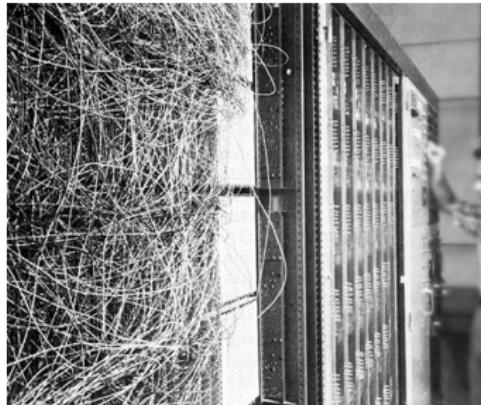
## Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.



## Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.

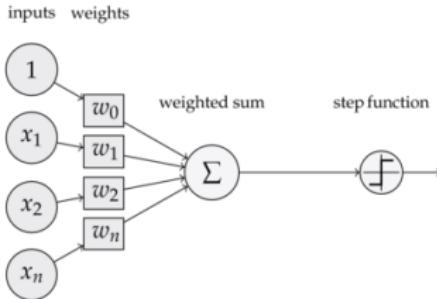


## Perceptron (Rosenblatt 1957)

- Inspired from biology.
- Very simple (linear) model!
- Physical implementation and proof of concept.

# The Perceptron Algorithm

Perceptron



- Key contribution: first (iterative) learning algorithm!

## Algorithm

- Start with  $w = 0$ .
- Repeat over all samples:
  - if  $\langle w, \mathbf{X}_i \rangle + b < 0$  modify  $w$  into  $w + Y_i \mathbf{X}_i$ ,
  - otherwise do not modify  $w$ .
- Convergence to a solution in a finite number of steps if the examples are linearly separable! (1962)

**Linear separability:** It exists  $w^*$  such that  $Y_i \langle w^*, X_i \rangle > 0$ .

**Proof:** Let  $C = \max_{X_i} \|C_i\|$  and  $\rho = \min Y_i \langle w^*, X_i \rangle > 0$ .

Let  $w_t$  be the weight at step  $t$ . If  $\min Y_i \langle w_t, X_i \rangle > 0$  then we are done. Otherwise, let  $(X_i, Y_i)$  be the first example such that  $Y_i(\langle w_t, X_i \rangle) \leq 0$  and let  $w_{t+1} = w_t + \alpha Y_i X_i$ . By construction,

$$\begin{aligned}\langle w^*, w_{t+1} \rangle &= \langle w^*, w_t \rangle + Y_i \langle w^*, X_i \rangle \\ &\geq \langle w^*, w_t \rangle + Y_i \langle w^*, W_i \rangle \\ &\geq \langle w^*, w_t \rangle + \rho \geq \langle w^*, w_0 \rangle + t\rho\end{aligned}$$

while

$$\begin{aligned}\|w_{t+1}\|^2 &= \|w_t\|^2 + \|X_i\|^2 + 2\langle w_t, Y_i X_i \rangle \\ &\leq \|w_t\|^2 + \|X_i\|^2 \leq \|w_t\|^2 + C^2 \leq \|w_0\|^2 + tC^2.\end{aligned}$$

Now  $\langle w^*, w_{t+1} \rangle \leq \|w^*\| \|w_{t+1}\|$  so that we have

$$\langle w^*, w_0 \rangle + t\rho \leq \|w^*\| (\|w_0\|^2 + tC^2)$$

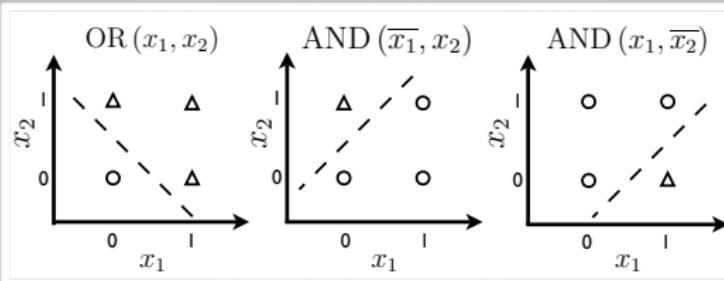
which implies that such a  $t$  is upperbounded and hence the algorithm converges.

- Slides *shamelessly* stolen from Hugo Larochelle (Sherbrooke)  
<http://www.dmi.usherb.ca/~larocheh/>
- Tremendous website!

## ARTIFICIAL NEURON

**Topics:** capacity of single neuron

- Can solve linearly separable problems



9

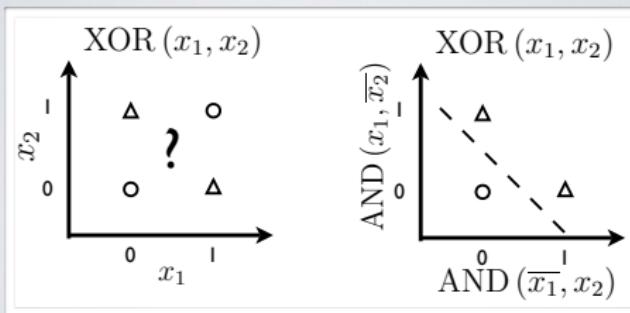
## Limitations

- Can't solve non linearly separable problems... (by design)
- Algorithm not robust to non linear separability!

## ARTIFICIAL NEURON

**Topics:** capacity of single neuron

- Can't solve non linearly separable problems...



- ... unless the input is transformed in a better representation

10

## Limitations

- Can't solve non linearly separable problems... (by design)
- Algorithm not robust to non linear separability!

- Prediction by  $f(\mathbf{X}) = \varphi(\langle w, \mathbf{X} \rangle)$  (or its sign).

## Activation function $\varphi$

- Hard threshold (non smooth)
- Sigmoid (logistic), tanh (smooth)
- Rectified Linear Unit (almost smooth, sparsification effect)

# ARTIFICIAL NEURON

**Topics:** connection weights, bias, activation function

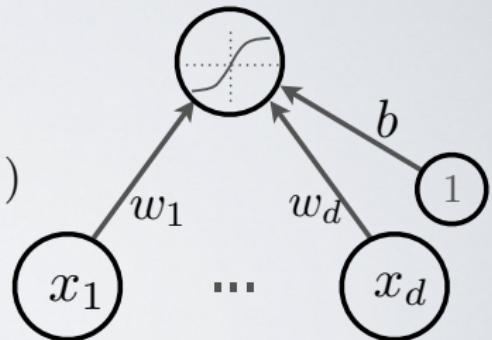
- Neuron input activation:

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron (output) activation

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

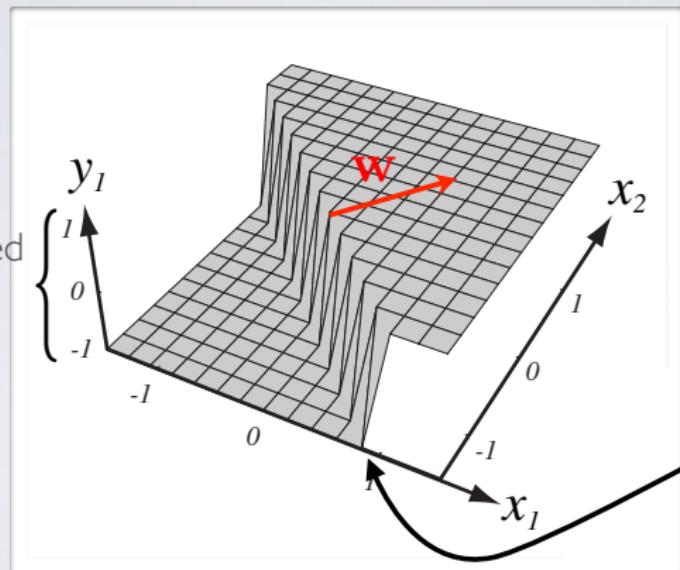
- $\mathbf{w}$  are the connection weights
- $b$  is the neuron bias
- $g(\cdot)$  is called the activation function



# ARTIFICIAL NEURON

**Topics:** connection weights, bias, activation function

range determined  
by  $g(\cdot)$



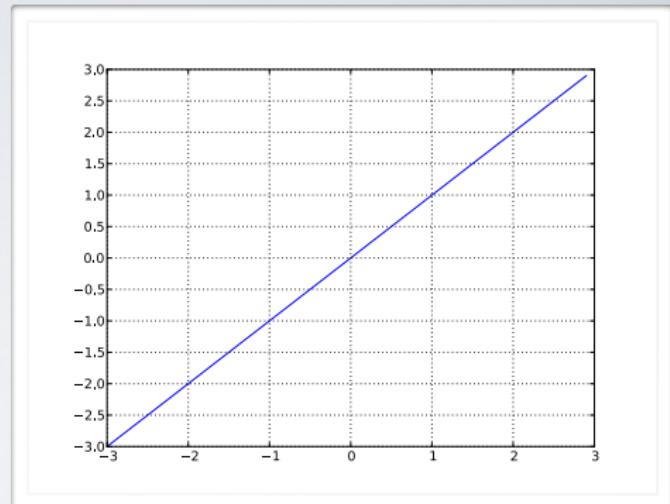
bias  $b$  only  
changes the  
position of  
the ripples

(from Pascal Vincent's slides)

# ARTIFICIAL NEURON

**Topics:** linear activation function

- Performs no input squashing
- Not very interesting...

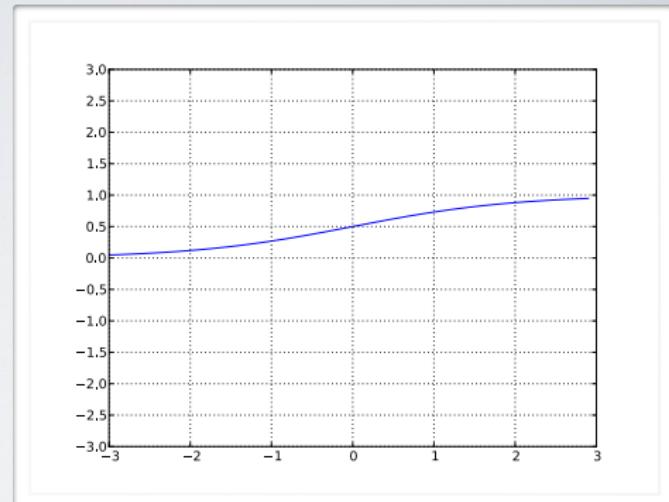


$$g(a) = a$$

# ARTIFICIAL NEURON

**Topics:** sigmoid activation function

- Squashes the neuron's input between 0 and 1
- Always positive
- Bounded
- Strictly increasing

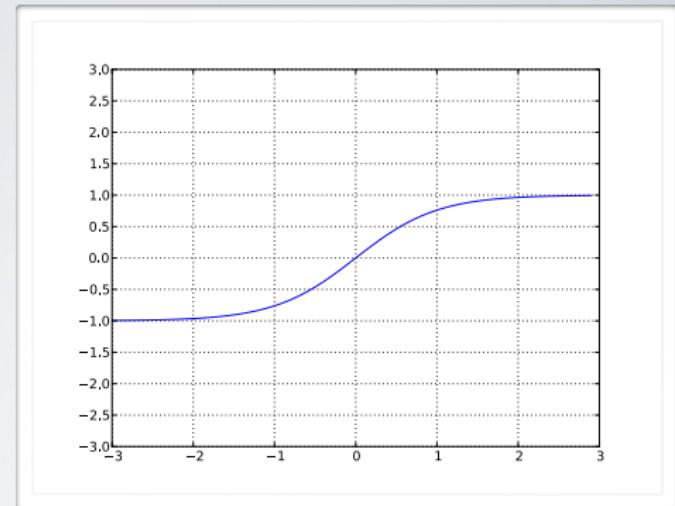


$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

# ARTIFICIAL NEURON

**Topics:** hyperbolic tangent (“tanh”) activation function

- Squashes the neuron’s input between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

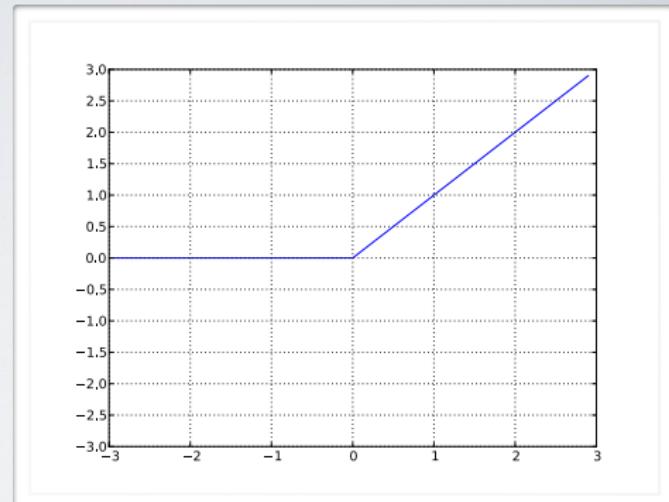


$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

# ARTIFICIAL NEURON

**Topics:** rectified linear activation function

- Bounded below by 0  
(always positive)
- Not upper bounded
- Strictly increasing
- Tends to give neurons  
with sparse activities



$$g(a) = \text{relin}(a) = \max(0, a)$$

- Prediction by  $f(\mathbf{X}) = \varphi(\langle w, \mathbf{X} \rangle)$  (or its sign).

## Activation function $\varphi$

- Hard threshold (non smooth)
- Sigmoid (logistic), tanh (smooth)
- Rectified Linear Unit (almost smooth, sparsification effect)
- Prediction error between  $Y$  and  $f(\mathbf{X})$  measure with a loss...

## Loss $\ell$

- Error count (non smooth)
- Probabilistic approach: log-likelihood
- Optimization approach: convex relaxation, squared loss...

# Gradient Descent Algorithm

Perceptron



- Empirical risk minimization:

$$\operatorname{argmin}_w \frac{1}{n} \sum_{i=1}^n \ell(Y_i, \varphi(\langle w, \mathbf{X}_i \rangle))$$

- If  $\ell$  and  $\varphi$  are almost differentiable (or convex):

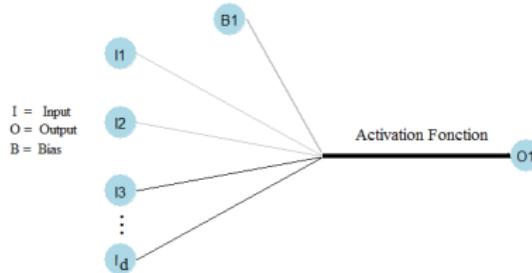
$$\nabla_i(w) = \frac{\partial \ell(Y_i, \varphi(\langle w, \mathbf{X}_i \rangle))}{\partial w} = \mathbf{X}_i \varphi'(\langle w, \mathbf{X}_i \rangle) \frac{\partial \ell}{\partial f}(Y_i, \varphi(\langle w, \mathbf{X}_i \rangle))$$

(Stochastic) Gradient descent rule:

$$w_{t+1} = w_t - h_y \frac{1}{|I_t|} \sum_{i \in I_t} \nabla_i(w_t)$$

- No convergence result except in the convex case...

Activation Neuron Configuration



## Artificial neuron

- Structure:
  - Mix inputs with a weighted sum,
  - Apply a (non linear) activation function to this sum,
  - Eventually threshold the result to make a decision.
- Weights learned by minimizing a loss function.

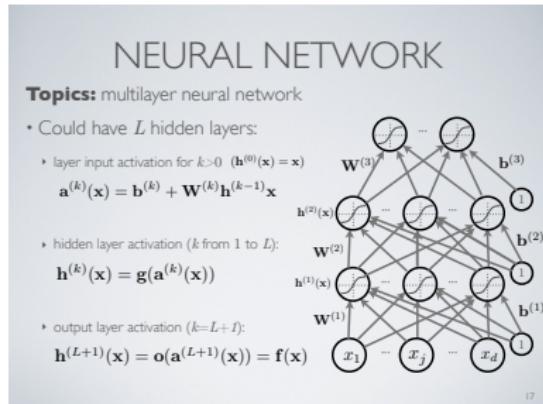
## Logistic unit

- Structure:
  - Mix inputs with a weighted sum,
  - Apply the logistic function  $\sigma(t) = e^t / (1 + e^t)$ ,
  - Threshold at 1/2 to make a decision!
- Logistic weights learned by minimizing the -log-likelihood.

- 1 Introduction
- 2 Supervised Learning
- 3 (Deep) Neural Networks
- 4 Perceptron
- 5 Multilayer Perceptron
- 6 Backprop Algorithm

# Multilayer Perceptron

Multilayer Perceptron



## MLP (Rumelhart, McClelland, Hinton - 1986)

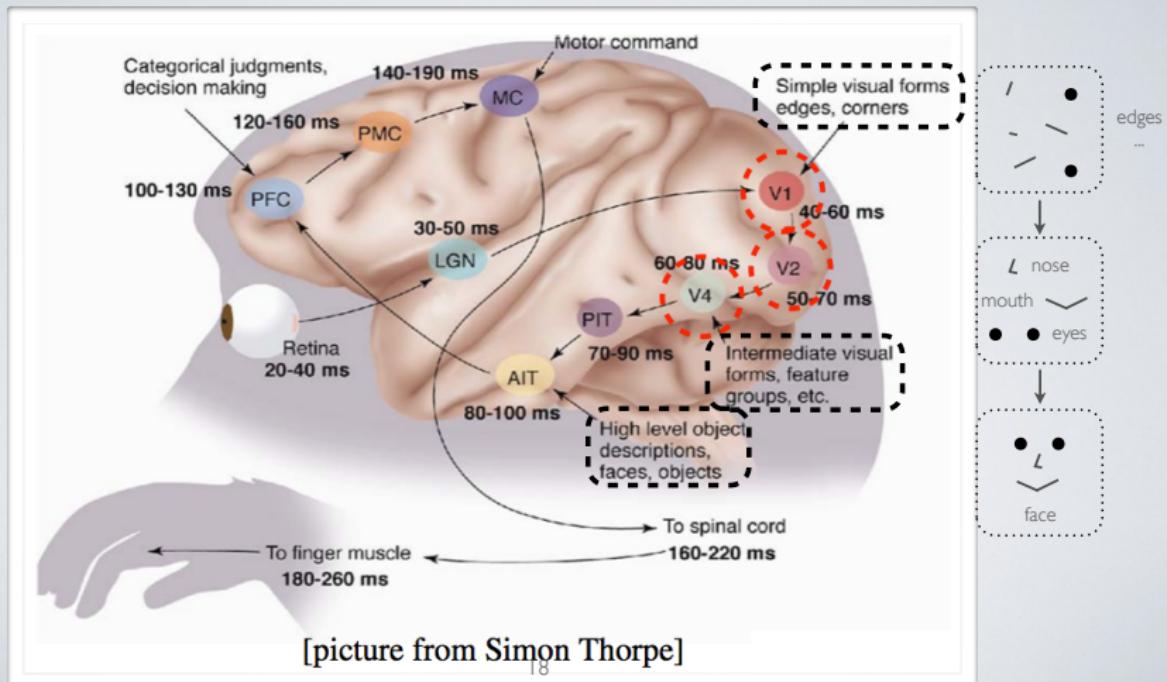
- Multilayer Perceptron: cascade of layers of artificial neuron units.
- Backprop algorithm to optimize it!
- Construction of a function by composing simple units.
- MLP corresponds to a specific direct acyclic graph structure.

## Universal Approximation Theorem (Hornik, 1991)

- A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units
- Valid for most activation functions.
- A single hidden layer is sufficient but more may be more efficient.
- No bounds on the number of required units... (Asymptotic flavour)

# NEURAL NETWORK

**Topics:** parallel with the visual cortex



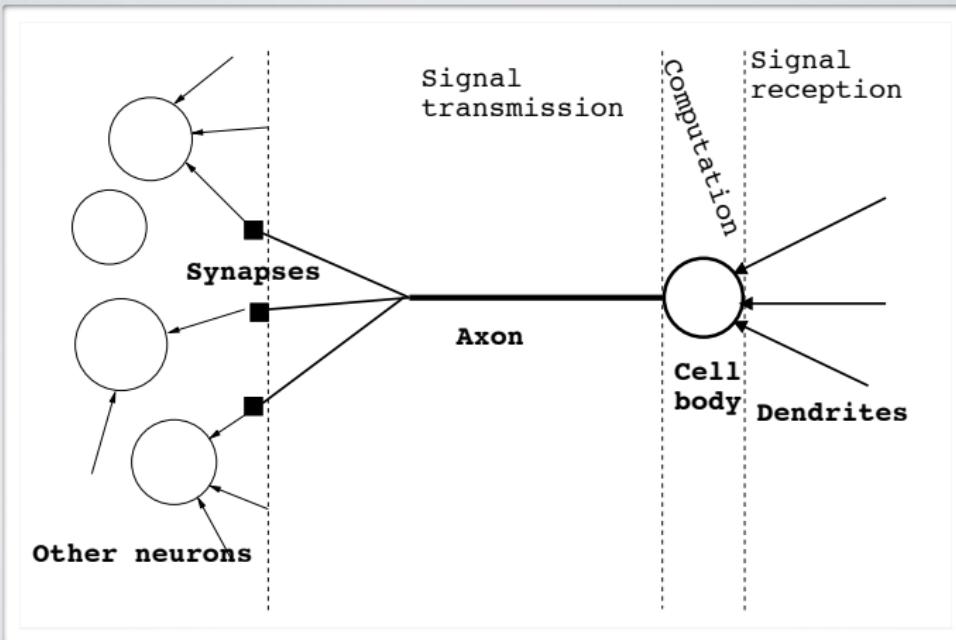
# BIOLOGICAL NEURONS

**Topics:** synapse, axon, dendrite

- We estimate around  $10^{10}$  and  $10^{11}$  the number of neurons in the human brain:
  - ▶ they receive information from other neurons through their dendrites
  - ▶ they “process” the information in their cell body (soma)
  - ▶ they send information through a “cable” called an axon
  - ▶ the point of connection between the axon branches and other neurons’ dendrites are called synapses

# BIOLOGICAL NEURONS

**Topics:** synapse, axon, dendrite



(from Hyvärinen, Hurri and Hoyer's book)

# BIOLOGICAL NEURONS

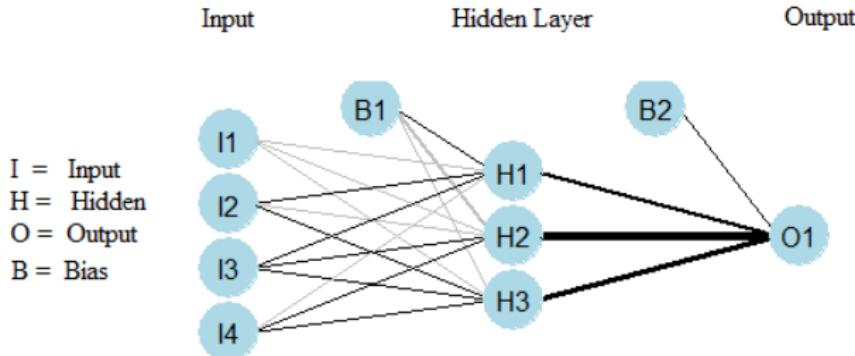
**Topics:** action potential, firing rate

- An action potential is an electrical impulse that travels through the axon:
  - ▶ this is how neurons communicate
  - ▶ it generates a “spike” in the electric potential (voltage) of the axon
  - ▶ an action potential is generated at neuron only if it receives enough (over some threshold) of the “right” pattern of spikes from other neurons
- Neurons can generate several such spikes every seconds:
  - ▶ the frequency of the spikes, called firing rate, is what characterizes the activity of a neuron
    - neurons are always firing a little bit, (spontaneous firing rate), but they will fire more, given the right stimulus

# BIOLOGICAL NEURONS

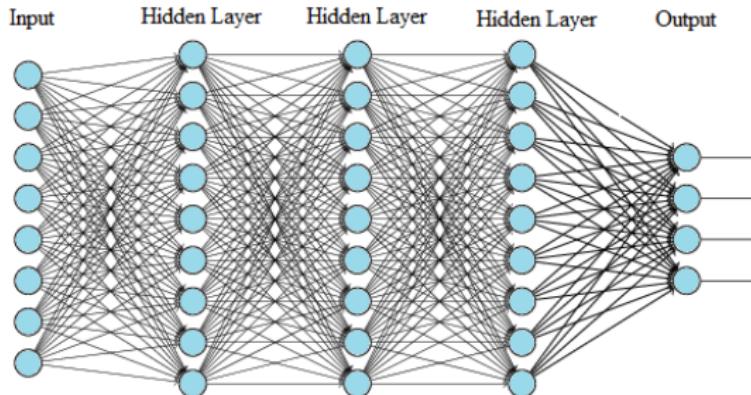
**Topics:** action potential, firing rate

- Firing rates of different input neurons combine to influence the firing rate of other neurons:
  - ▶ depending on the dendrite and axon, a neuron can either work to increase (excite) or decrease (inhibit) the firing rate of another neuron
- This is what artificial neurons approximate:
  - ▶ the activation corresponds to a “sort of” firing rate
  - ▶ the weights between neurons model whether neurons excite or inhibit each other
  - ▶ the activation function and bias model the thresholded behavior of action potentials



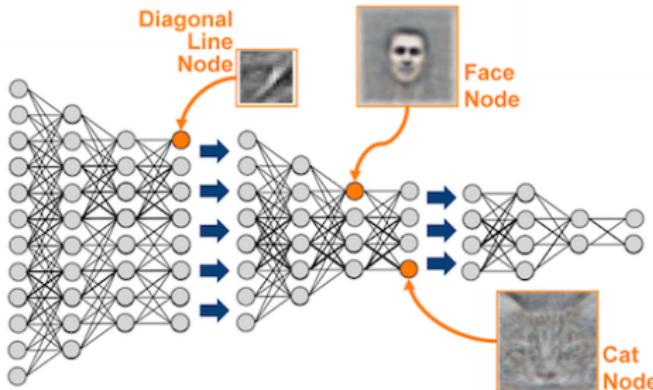
## Design

- Number of layers
- Connectivity
- Tricks: convolution (weight sharing), polling...



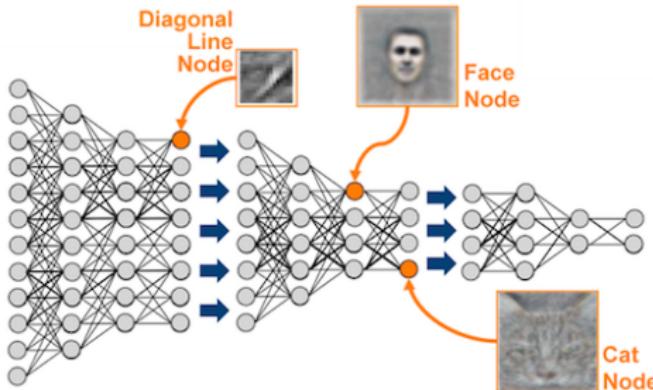
## Design

- Number of layers
- Connectivity
- Tricks: convolution (weight sharing), polling...



## Design

- Number of layers
- Connectivity
- Tricks: convolution (weight sharing), polling...



## Design

- Number of layers
- Connectivity
- Tricks: convolution (weight sharing), polling...

- Generalization of the logistic output for multiclass.

## A probabilistic approach

- At the outer nodes, computes an *activation*  $a_i$  for all classes.
- Use a *joint softmax* activation function:

$$o_i = \frac{e^{a_i}}{\sum_{i'=1}^K e^{a_{i'}}}$$

- Amounts to model the probability of each class by a conditional multinomial logistic model on the output of the last hidden layer.

- Perfectly fit for ML approach.
- Rk:** Slight overparameterization...

# MACHINE LEARNING

**Topics:** empirical risk minimization, regularization

- Empirical risk minimization
  - ▶ framework to design learning algorithms

$$\arg \min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

- $l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$  is a loss function
- $\Omega(\boldsymbol{\theta})$  is a regularizer (penalizes certain values of  $\boldsymbol{\theta}$ )
- Learning is cast as optimization
  - ▶ ideally, we'd optimize classification error, but it's not smooth
  - ▶ loss function is a surrogate for what we truly should optimize (e.g. upper bound)

# REGULARIZATION

**Topics:** L2 regularization

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left( W_{i,j}^{(k)} \right)^2 = \sum_k \|\mathbf{W}^{(k)}\|_F^2$$

- Gradient:  $\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = 2\mathbf{W}^{(k)}$
- Only applied on weights, not on biases (weight decay)
- Can be interpreted as having a Gaussian prior over the weights

# REGULARIZATION

**Topics:** L1 regularization

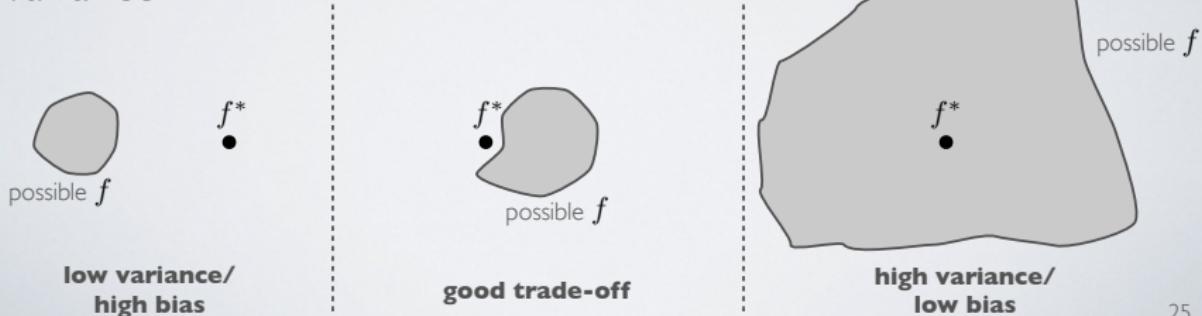
$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

- Gradient:  $\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = \text{sign}(\mathbf{W}^{(k)})$ 
  - where  $\text{sign}(\mathbf{W}^{(k)})_{i,j} = 1_{\mathbf{W}_{i,j}^{(k)} > 0} - 1_{\mathbf{W}_{i,j}^{(k)} < 0}$
- Also only applied on weights
- Unlike L2, L1 will push certain weights to be exactly 0
- Can be interpreted as having a Laplacian prior over the weights

# MACHINE LEARNING

**Topics:** bias-variance trade-off

- Variance of trained model: does it vary a lot if the training set changes
- Bias of trained model: is the average model close to the true solution
- Generalization error can be seen as the sum of bias and the variance



- 1 Introduction
- 2 Supervised Learning
- 3 (Deep) Neural Networks
- 4 Perceptron
- 5 Multilayer Perceptron
- 6 Backprop Algorithm

## A Clever Gradient Descent Implementation

- Popularized by Rumelhart, McClelland, Hinton in 1986.
  - Can be traced back to Werbos in 1974.
  - Nothing but the use of chain rule derivation with a touch of dynamic programming.
- 
- Key ingredient to make the Neural Networks work!
  - Still at the core of Deep Learning algorithm.

# MACHINE LEARNING

**Topics:** stochastic gradient descent (SGD)

- Algorithm that performs updates after each example

- initialize  $\boldsymbol{\theta}$  ( $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$ )

- for N iterations

- for each training example  $(\mathbf{x}^{(t)}, y^{(t)})$ 
      - ✓  $\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$
      - ✓  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$

} training epoch  
= iteration over **all** examples

- To apply this algorithm to neural network training, we need

- the loss function  $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

- a procedure to compute the parameter gradients  $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

- the regularizer  $\Omega(\boldsymbol{\theta})$  (and the gradient  $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$ )

- initialization method

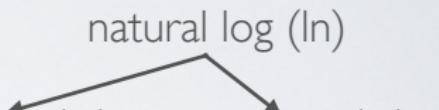
# LOSS FUNCTION

**Topics:** loss function for classification

- Neural network estimates  $f(\mathbf{x})_c = p(y = c|\mathbf{x})$ 
  - ▶ we could maximize the probabilities of  $y^{(t)}$  given  $\mathbf{x}^{(t)}$  in the training set
- To frame as minimization, we minimize the negative log-likelihood

$$l(\mathbf{f}(\mathbf{x}), y) = - \sum_c 1_{(y=c)} \log f(\mathbf{x})_c = - \log f(\mathbf{x})_y$$

natural log ( $\ln$ )



- ▶ we take the log to simplify for numerical stability and math simplicity
- ▶ sometimes referred to as cross-entropy

# LOSS FUNCTION

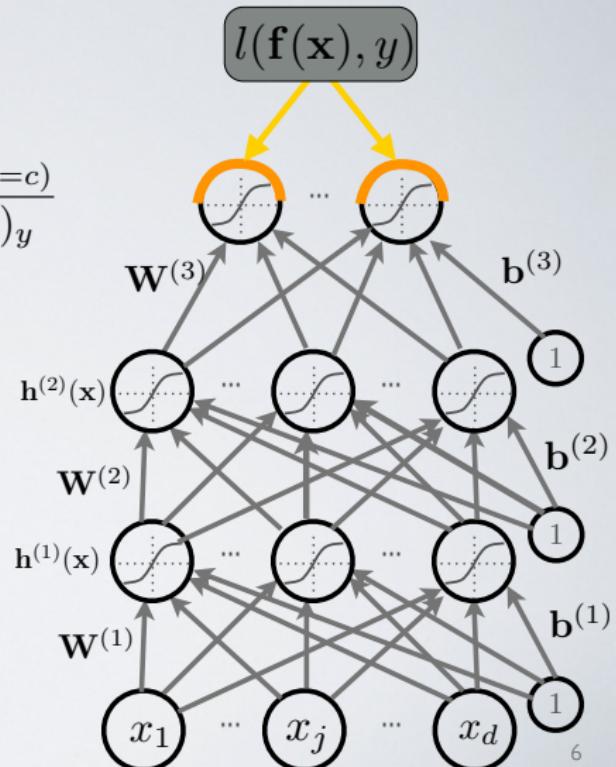
**Topics:** loss gradient at output

- Partial derivative:

$$\frac{\partial}{\partial f(\mathbf{x})_c} - \log f(\mathbf{x})_y = \frac{-1_{(y=c)}}{f(\mathbf{x})_y}$$

- Gradient:

$$\begin{aligned}\nabla_{\mathbf{f}(\mathbf{x})} - \log f(\mathbf{x})_y \\ &= \frac{-1}{f(\mathbf{x})_y} \begin{bmatrix} 1_{(y=0)} \\ \vdots \\ 1_{(y=C-1)} \end{bmatrix} \\ &= \frac{-\mathbf{e}(y)}{f(\mathbf{x})_y}\end{aligned}$$



# LOSS FUNCTION

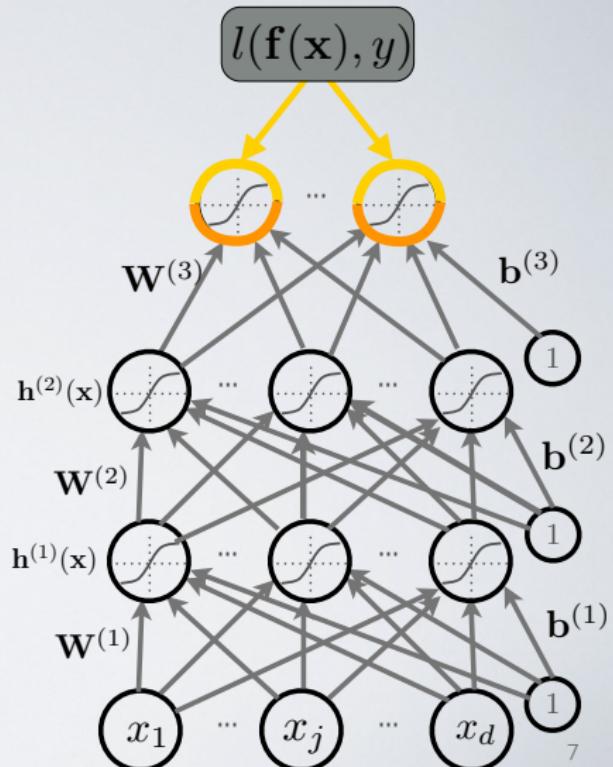
**Topics:** loss gradient at output  
(before activation)

- Partial derivative:

$$\begin{aligned} & \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\ = & - (1_{(y=c)} - f(\mathbf{x})_c) \end{aligned}$$

- Gradient:

$$\begin{aligned} & \nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \\ = & - (\mathbf{e}(y) - \mathbf{f}(\mathbf{x})) \end{aligned}$$



$$\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y$$

$$\begin{aligned} & \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\ = & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y
\end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})}
\end{aligned}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})}
\end{aligned}$$

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left( \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left( \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right)
\end{aligned}$$

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left( \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left( \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right)
\end{aligned}$$

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left( \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left( \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( 1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right)
\end{aligned}$$

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left( \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left( \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( 1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right) \\
= & \frac{-1}{f(\mathbf{x})_y} (1_{(y=c)} f(\mathbf{x})_y - f(\mathbf{x})_y f(\mathbf{x})_c)
\end{aligned}$$

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}$$

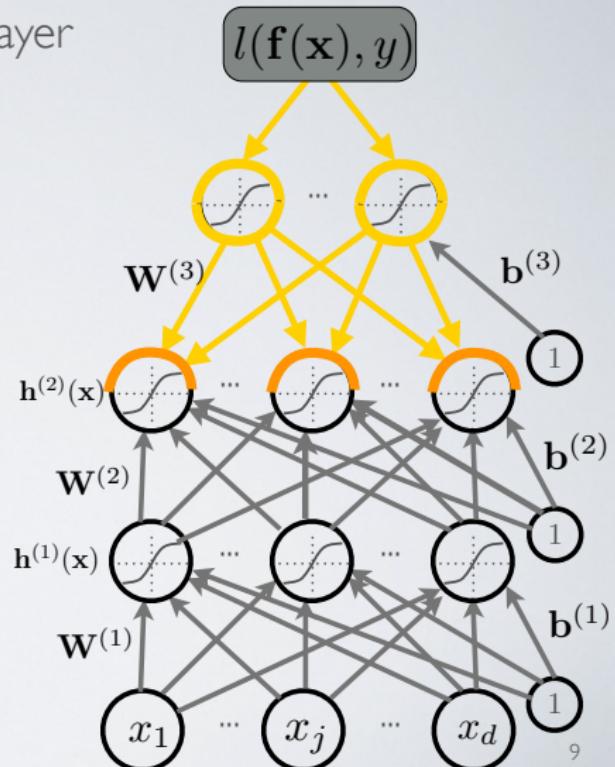
$$\boxed{\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x} \frac{1}{h(x)} - \frac{g(x)}{h(x)^2} \frac{\partial h(x)}{\partial x}}$$

$$\begin{aligned}
& \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
= & \frac{-1}{f(\mathbf{x})_y} \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y) \left( \frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)}{\left( \sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'}) \right)^2} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( \frac{1_{(y=c)} \exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'} \exp(a^{(L+1)}(\mathbf{x})_{c'})} \right) \\
= & \frac{-1}{f(\mathbf{x})_y} \left( 1_{(y=c)} \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \text{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c \right) \\
= & \frac{-1}{f(\mathbf{x})_y} (1_{(y=c)} f(\mathbf{x})_y - f(\mathbf{x})_y f(\mathbf{x})_c) \\
= & - (1_{(y=c)} - f(\mathbf{x})_c)
\end{aligned}$$

# LOSS FUNCTION

**Topics:** loss gradient at hidden layer

- ... this is getting complicated!!

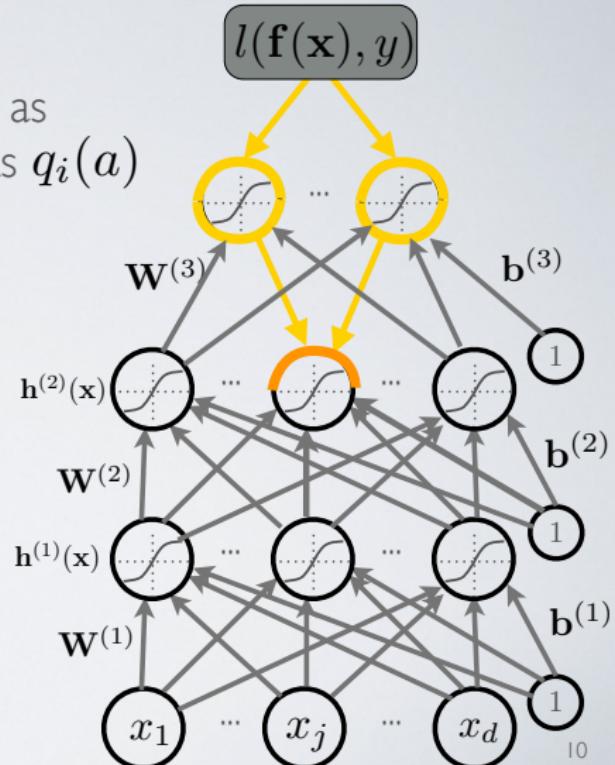


# LOSS FUNCTION

**Topics:** chain rule

- If a function  $p(a)$  can be written as a function of intermediate results  $q_i(a)$  then we have:

$$\frac{\partial p(a)}{\partial a} = \sum_i \frac{\partial p(a)}{\partial q_i(a)} \frac{\partial q_i(a)}{\partial a}$$



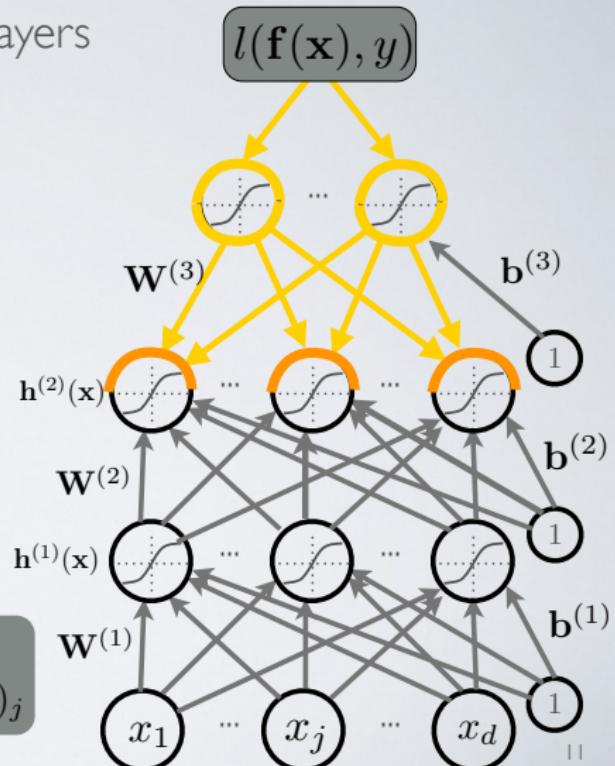
- We can invoke it by setting
  - $a$  to a unit in layer
  - $q_i(a)$  to input of units in the layer above
  - $p(a)$  is the loss function

# LOSS FUNCTION

**Topics:** loss gradient at hidden layers

- Partial derivative:

$$\begin{aligned} & \frac{\partial}{\partial h^{(k)}(\mathbf{x})_j} - \log f(\mathbf{x})_y \\ = & \sum_i \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k+1)}(\mathbf{x})_i} \frac{\partial a^{(k+1)}(\mathbf{x})_i}{\partial h^{(k)}(\mathbf{x})_j} \\ = & \sum_i \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k+1)}(\mathbf{x})_i} W_{i,j}^{(k+1)} \end{aligned}$$



REMINDER

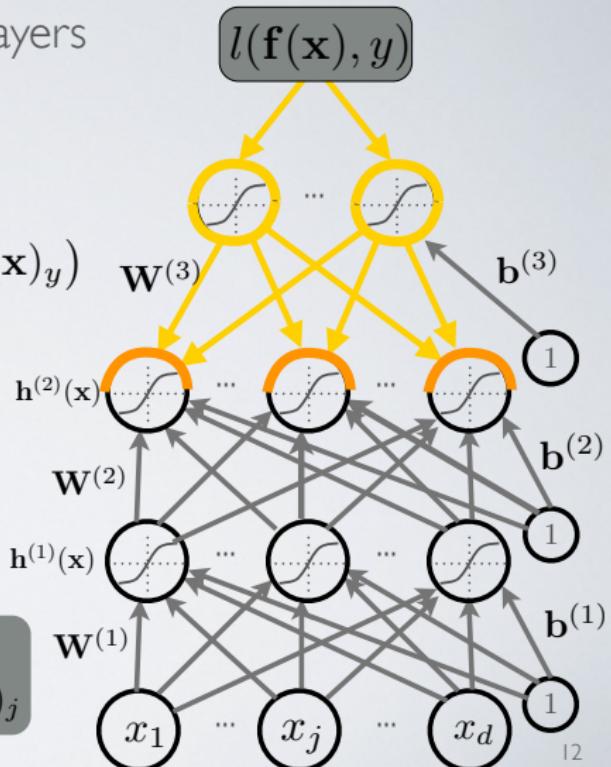
$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

# LOSS FUNCTION

**Topics:** loss gradient at hidden layers

- Gradient:

$$\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y \\ = \mathbf{W}^{(k+1)^\top} (\nabla_{\mathbf{a}^{(k+1)}(\mathbf{x})} - \log f(\mathbf{x})_y)$$



REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

# LOSS FUNCTION

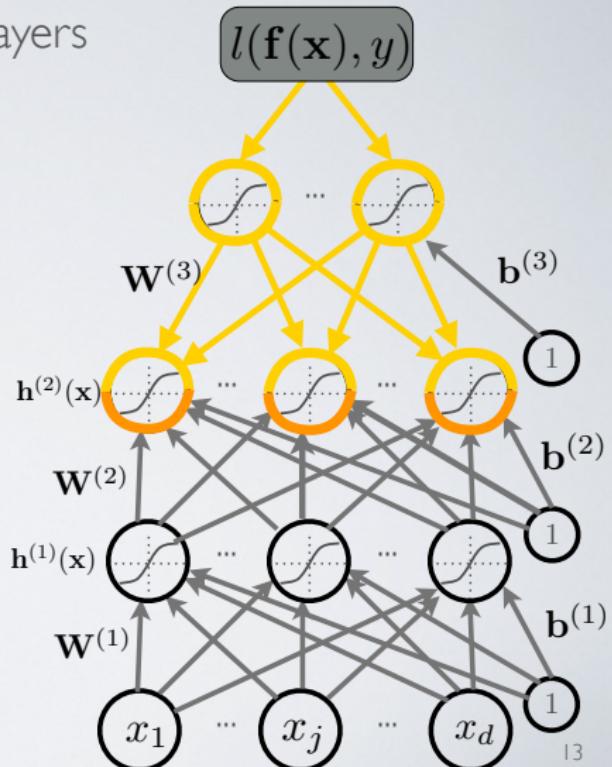
**Topics:** loss gradient at hidden layers  
(before activation)

- Partial derivative:

$$\begin{aligned} & \frac{\partial}{\partial a^{(k)}(\mathbf{x})_j} - \log f(\mathbf{x})_y \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial h^{(k)}(\mathbf{x})_j} \frac{\partial h^{(k)}(\mathbf{x})_j}{\partial a^{(k)}(\mathbf{x})_j} \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial h^{(k)}(\mathbf{x})_j} g'(a^{(k)}(\mathbf{x})_j) \end{aligned}$$

REMINDER

$$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$$



# LOSS FUNCTION

**Topics:** loss gradient at hidden layers  
(before activation)

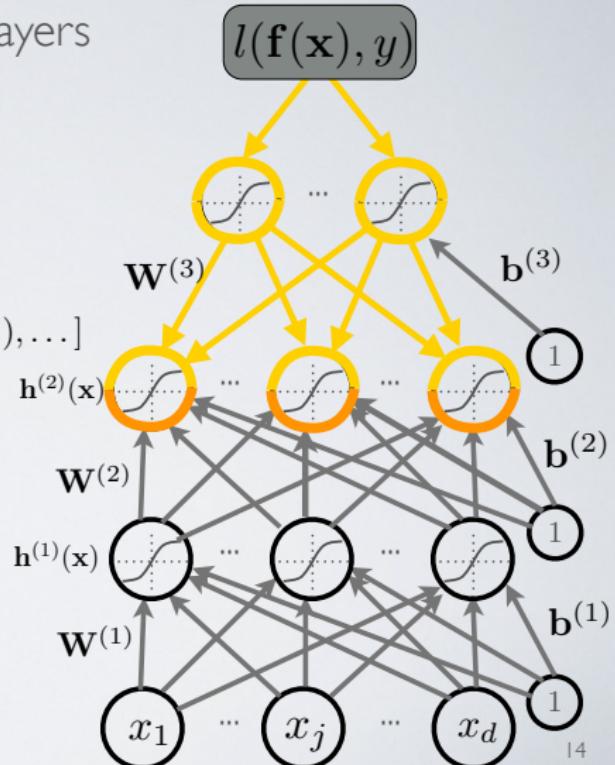
- Gradient:

$$\begin{aligned} & \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y \\ = & (\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y)^\top \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} \mathbf{h}^{(k)}(\mathbf{x}) \\ = & (\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \odot [\dots, g'(a^{(k)}(\mathbf{x})_j), \dots] \end{aligned}$$

↑  
element-wise  
product

REMINDER

$$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$$



# LOSS FUNCTION

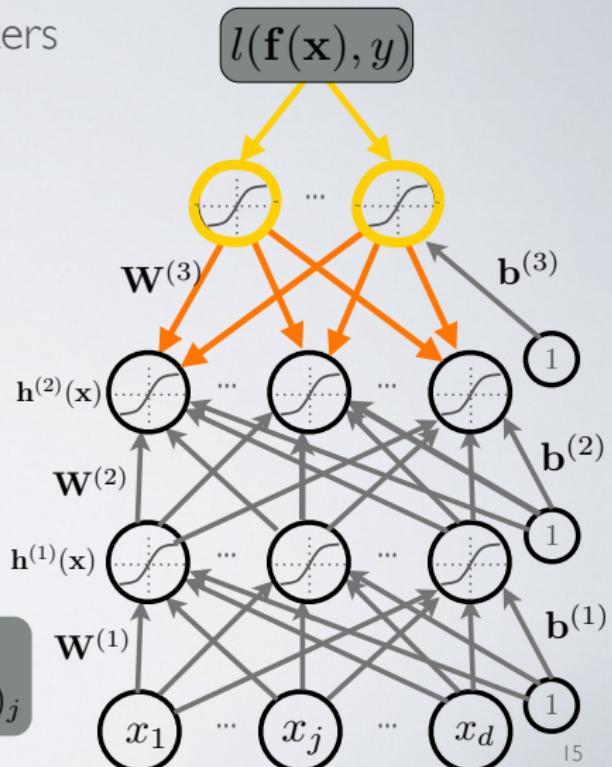
**Topics:** loss gradient of parameters

- Partial derivative (weights):

$$\begin{aligned} & \frac{\partial}{\partial W_{i,j}^{(k)}} - \log f(\mathbf{x})_y \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \frac{\partial a^{(k)}(\mathbf{x})_i}{\partial W_{i,j}^{(k)}} \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} h_j^{(k-1)}(\mathbf{x}) \end{aligned}$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

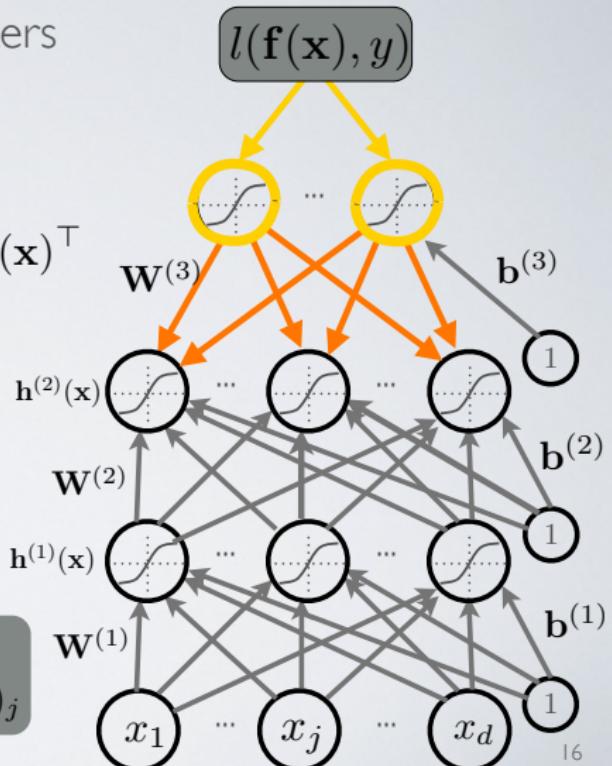


# LOSS FUNCTION

**Topics:** loss gradient of parameters

- Gradient (weights):

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \\ = (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \quad \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$



REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

# LOSS FUNCTION

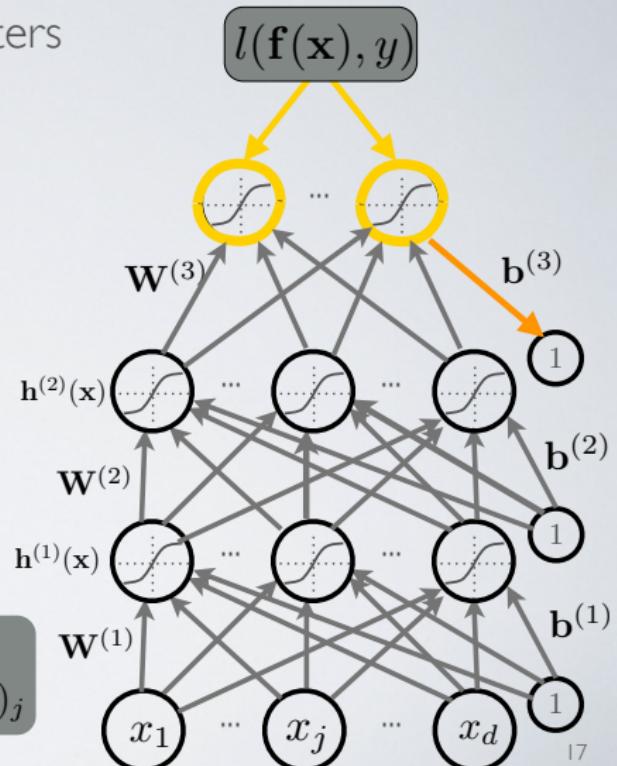
**Topics:** loss gradient of parameters

- Partial derivative (biases):

$$\begin{aligned} & \frac{\partial}{\partial b_i^{(k)}} - \log f(\mathbf{x})_y \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \frac{\partial a^{(k)}(\mathbf{x})_i}{\partial b_i^{(k)}} \\ = & \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \end{aligned}$$

REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

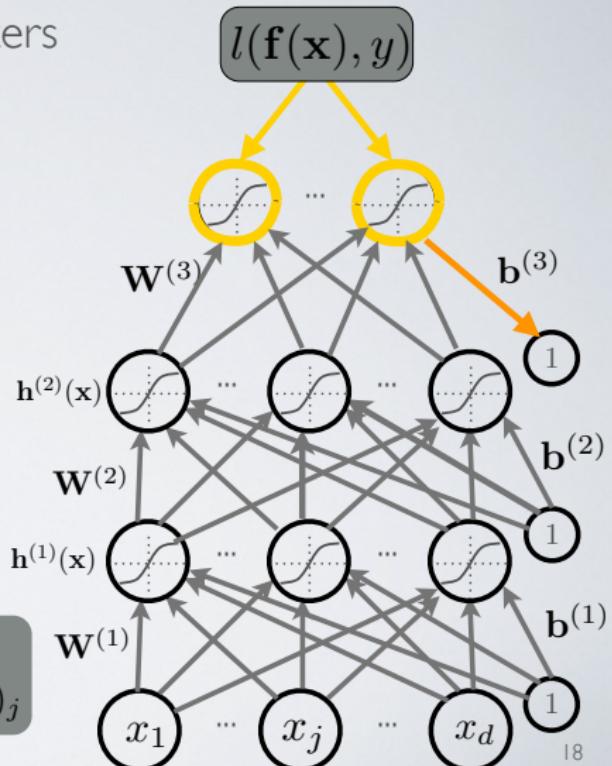


# LOSS FUNCTION

**Topics:** loss gradient of parameters

- Gradient (biases):

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \\ = \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$



REMINDER

$$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$$

# BACKPROPAGATION

**Topics:** backpropagation algorithm

- ▶ compute output gradient (before activation)

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

- ▶ for  $k$  from  $L+1$  to 1

- compute gradients of hidden layer parameter

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \iff (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y) \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \iff \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

- compute gradient of hidden layer below

$$\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff \mathbf{W}^{(k)^\top} (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y)$$

- compute gradient of hidden layer below (before activation)

$$\nabla_{\mathbf{a}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \iff (\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y) \odot [\dots, g'(a^{(k-1)}(\mathbf{x})_j), \dots]$$

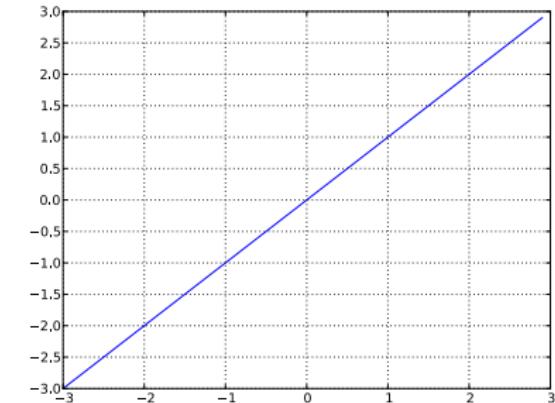
- This assumes a forward propagation has been made before

# ACTIVATION FUNCTIONS

**Topics:** linear activation function gradient

- Partial derivative:

$$g'(a) = 1$$



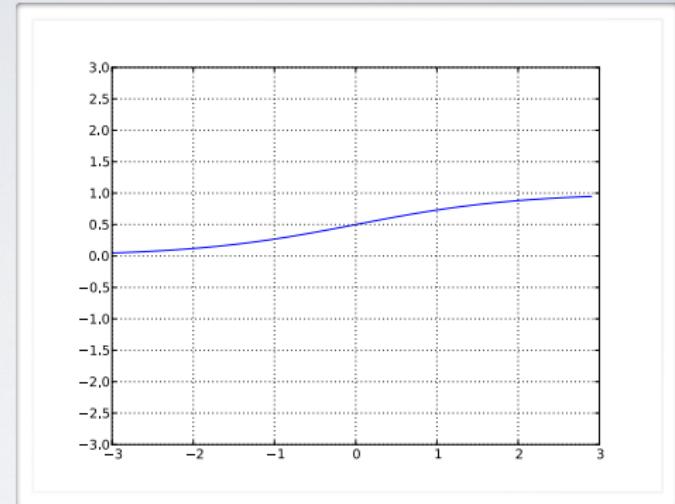
$$g(a) = a$$

# ACTIVATION FUNCTIONS

**Topics:** sigmoid activation function gradient

- Partial derivative:

$$g'(a) = g(a)(1 - g(a))$$



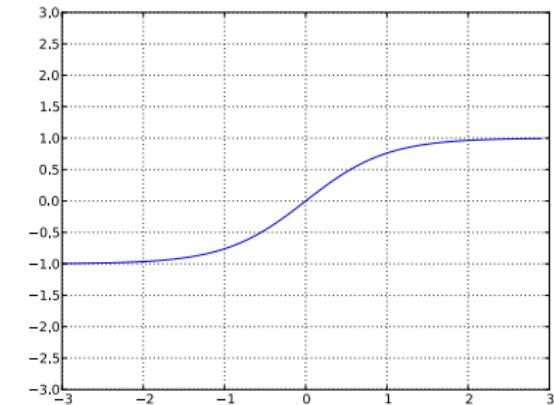
$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

# ACTIVATION FUNCTIONS

**Topics:** tanh activation function gradient

- Partial derivative:

$$g'(a) = 1 - g(a)^2$$



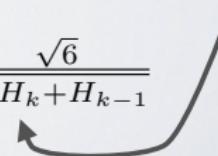
$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

# INITIALIZATION

## Topics: initialization

- For biases
  - ▶ initialize all to 0
- For weights
  - ▶ Can't initialize weights to 0 with tanh activation
    - we can show that all gradients would then be 0 (saddle point)
  - ▶ Can't initialize all weights to the same value
    - we can show that all hidden units in a layer will always behave the same
    - need to break symmetry
  - ▶ Recipe: sample  $\mathbf{W}_{i,j}^{(k)}$  from  $U[-b, b]$  where  $b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$ 
    - the idea is to sample around 0 but break symmetry
    - other values of  $b$  could work well (not an exact science) ( see Glorot & Bengio, 2010)

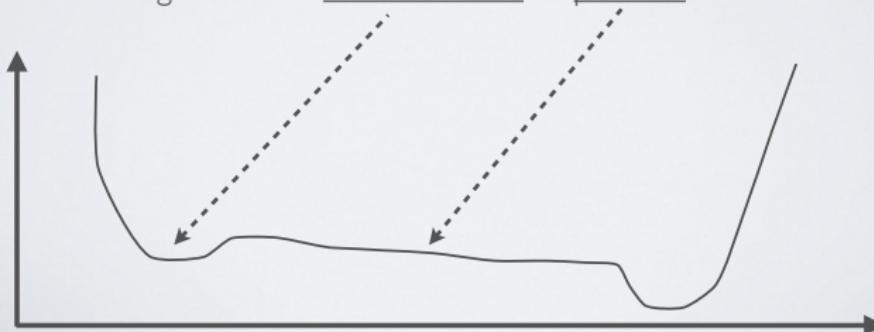
size of  $\mathbf{h}^{(k)}(\mathbf{x})$



# OPTIMIZATION

**Topics:** local optimum, global optimum, plateau

- Notes on the optimization problem
  - ▶ There isn't a single global optimum (non-convex optimization)
    - we can permute the hidden units (with their connections) and get the same function
    - we say that the hidden unit parameters are not identifiable
  - ▶ Optimization can get stuck in local minimum or plateaus



# OPTIMIZATION

**Topics:** local optimum, global optimum, plateau

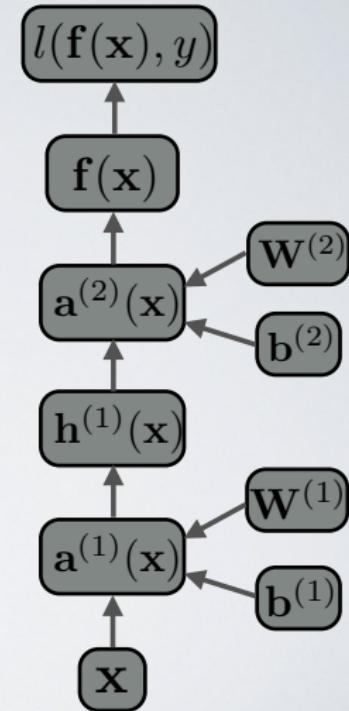
## **Neural network training demo**

<http://cs.stanford.edu/~karpathy/svmjs/demo/demonn.html>

# FLOW GRAPH

## Topics: flow graph

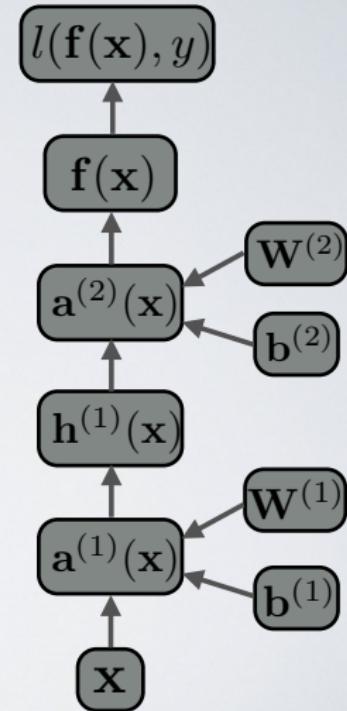
- Forward propagation can be represented as an acyclic flow graph
- It's a nice way of implementing forward propagation in a modular way
  - each box is an object with an fprop method, that outputs the value of the box given its children
  - calling the fprop method of each box in the right order yield forward propagation



# FLOW GRAPH

**Topics:** automatic differentiation

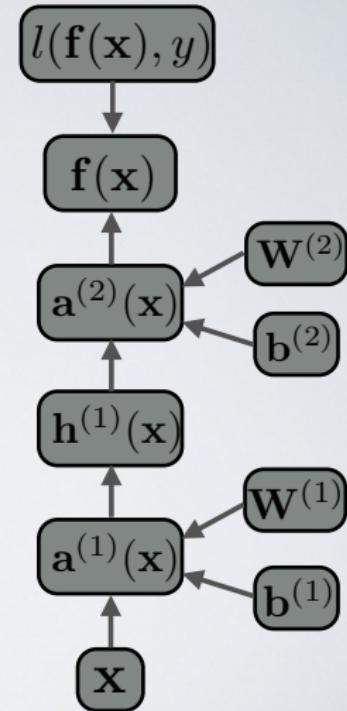
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

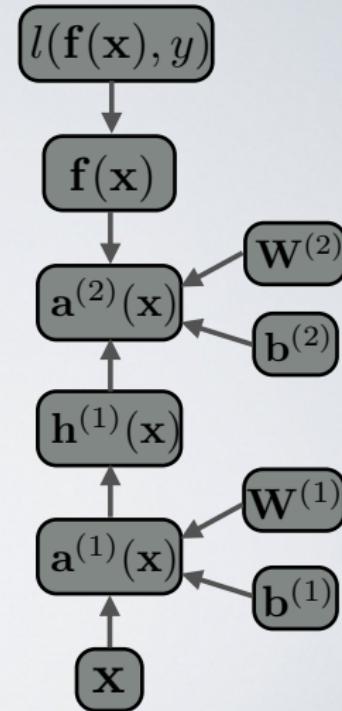
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

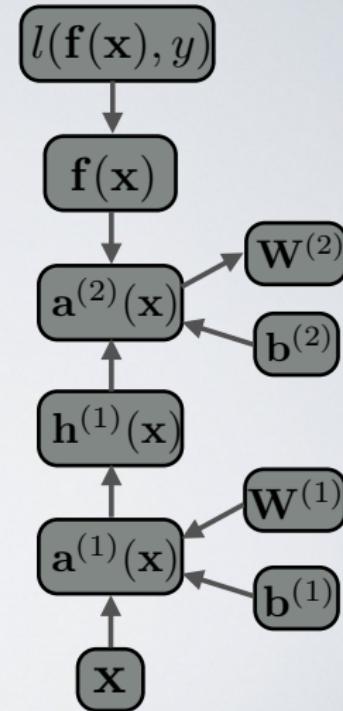
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

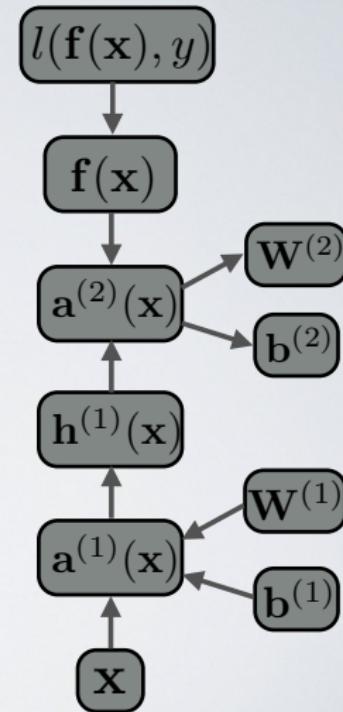
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

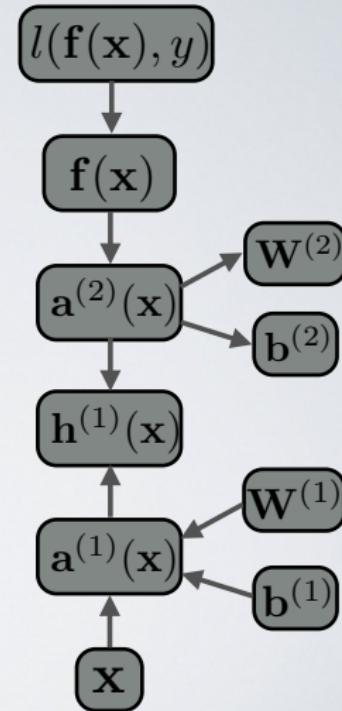
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

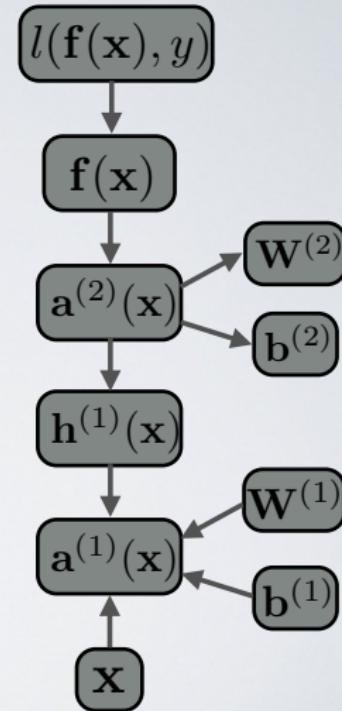
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

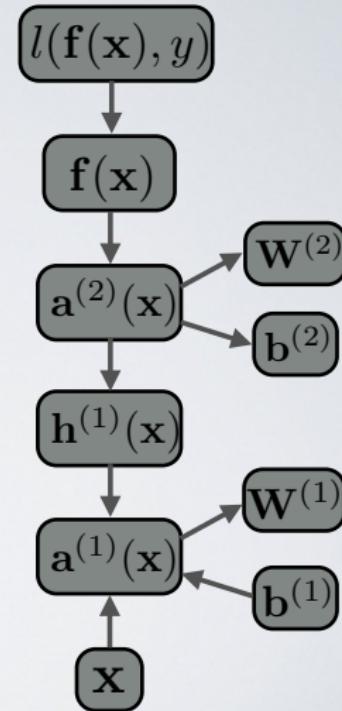
- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters



# FLOW GRAPH

**Topics:** automatic differentiation

- Each object also has a bprop method
  - it outputs the gradient of the loss given each children
  - fprop depends on the fprop of a box's children, while bprop depends the bprop of a box's parents
- By calling bprop in the reverse order, we get backpropagation
  - only need to reach the parameters

