[course site]

#DLUPC

Day 1 Lecture 2

Training a Multilayer Perceptron

Xavier Giro-i-Nieto
xavier.giro@upc.edu

Associate Professor
Universitat Politecnica de Catalunya
Technical University of Catalonia

Acknowledgements

Santiago Pascual

Kevin McGuinness
kevin.mcguinness@dcu.ie

Research Fellow
Insight Centre for Data Analytics
Dublin City University

Outline

1. **Supervised learning: regression/classification**
2. Single neuron models (perceptrons)
    a. Linear regression
    b. Logistic regression
    c. Multiple outputs and softmax regression
3. Multi-Layer Perceptrons (MLP)
4. Training a deep neural network

Types of machine learning

Yann Lecun’s Black Forest cake

- **“Pure” Reinforcement Learning** (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning** (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning** (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- (Yes, I know, this picture is slightly offensive to RL folks. But I’ll make it up)

Types of machine learning

We can categorize three types of learning procedures:

1. Supervised Learning:

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

2. Unsupervised Learning:

   $f(\mathbf{x})$

3. Reinforcement Learning:

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

   $\mathbf{z}$

Types of machine learning

We can categorize three types of learning procedures:

1. Supervised Learning:

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

2. Unsupervised Learning:

   $$f(\mathbf{x})$$

3. Reinforcement Learning:

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

   $$\mathbf{z}$$

Types of machine learning

We can categorize three types of learning procedures:

1. **Supervised Learning:**

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

We have a labeled dataset with pairs ($\mathbf{x}$, $\mathbf{y}$), e.g. classify a signal window as containing speech or not:

$\mathbf{x}_1$ = [x(1), x(2), ..., x(T)] $\mathbf{y}_1$ = "no"
$\mathbf{x}_2$ = [x(T+1), ..., x(2T)] $\mathbf{y}_2$ = "yes"
$\mathbf{x}_3$ = [x(2T+1), ..., x(3T)] $\mathbf{y}_3$ = "yes"
...

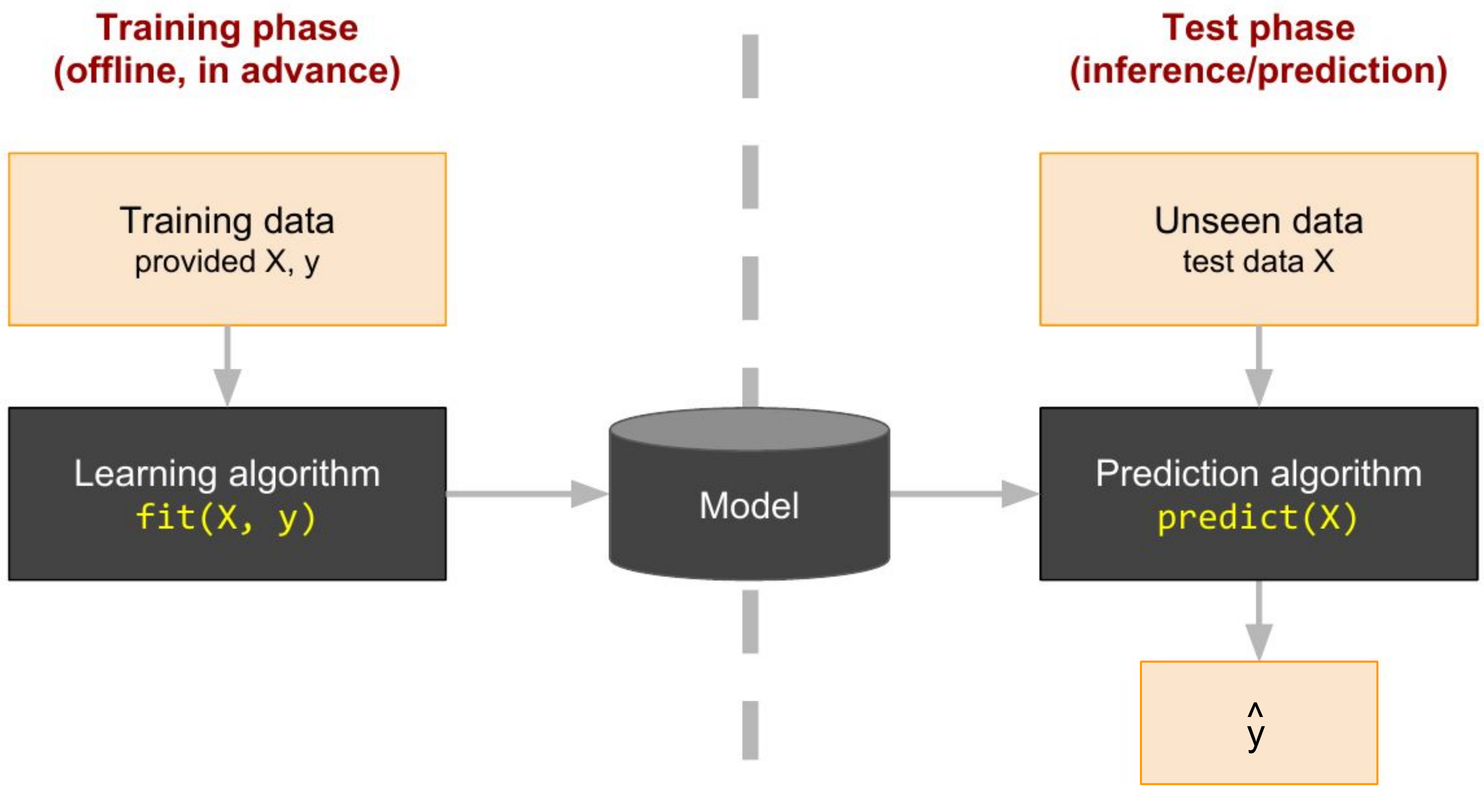
Supervised learning

Fit a function: $\mathbf{y} = f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^m$

Given paired training examples $\{(\mathbf{x}_i, \mathbf{y}_i)\}$

Key point: **generalize well to unseen examples**

Black box abstraction of supervised learning



Regression vs Classification

Depending on the type of target $\mathbf{y}$ we get:

- **Regression:** $\mathbf{y} \in \mathbb{R}^N$ is continuous (e.g. temperatures $\mathbf{y} = \{19^\circ, 23^\circ, 22^\circ\}$)

- **Classification:** $\mathbf{y}$ is discrete (e.g. $\mathbf{y} = \{1, 2, 5, 2, 2\}$).

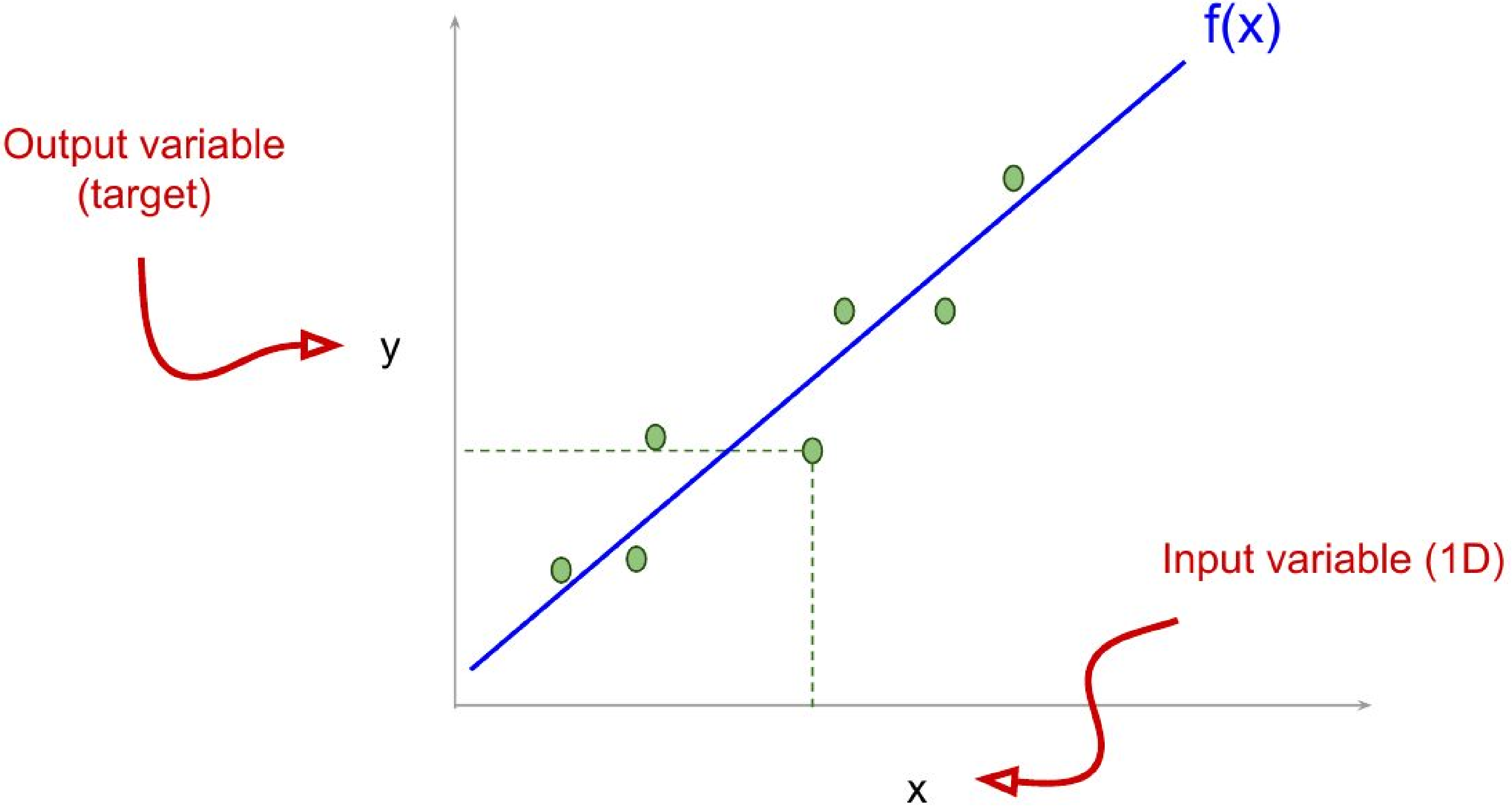
Regression vs Classification

Depending on the type of target $\mathbf{y}$ we get:

- **<u>Regression</u>:** $\mathbf{y} \in \mathbb{R}^N$ is continuous (e.g. temperatures $\mathbf{y}$ = {19º, 23º, 22º})

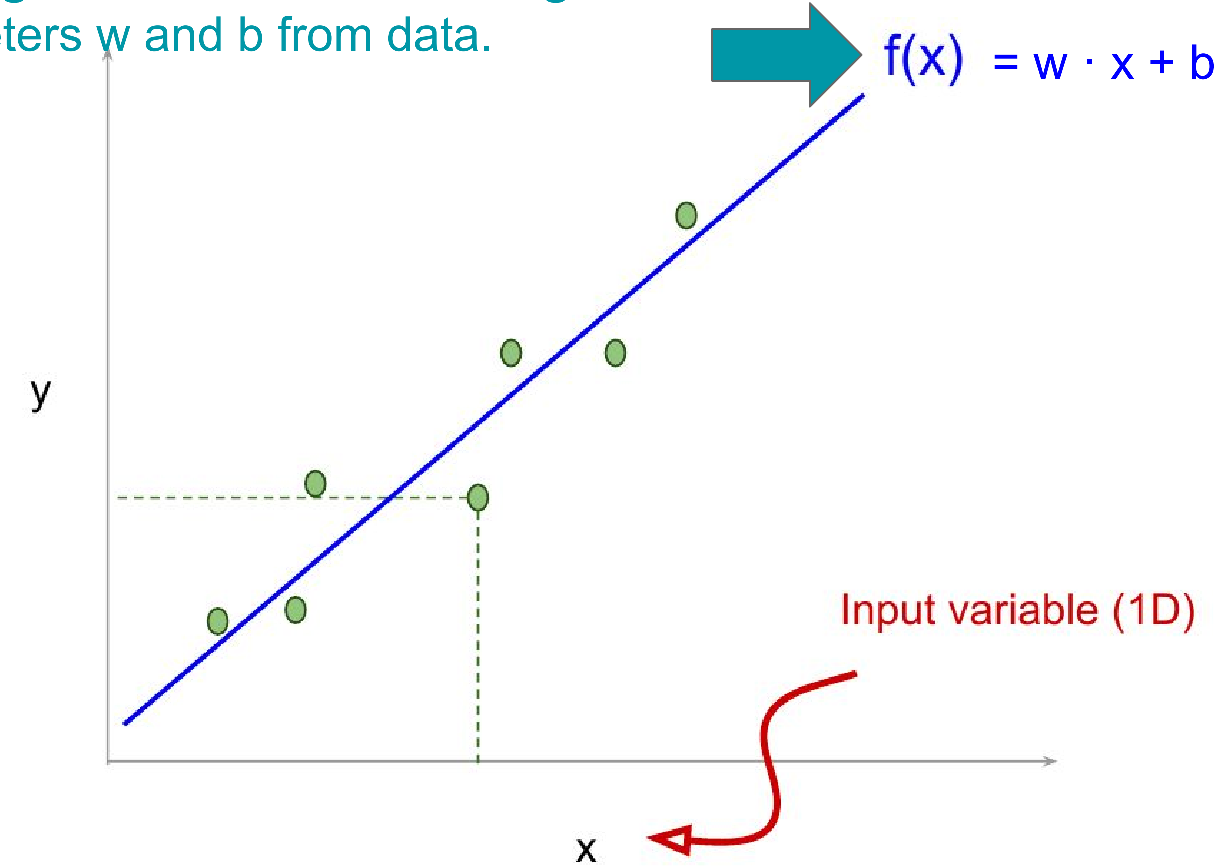
- **<u>Classification</u>:** $\mathbf{y}$ is discrete (e.g. $\mathbf{y}$ = {1, 2, 5, 2, 2}).

Linear Regression (eg. 1D input - 1D ouput)



Linear Regression (eg. 1D input - 1D ouput)

Training a model means **learning** parameters w and b from data.



Linear Regression (M-D input)

Input data can also be M-dimensional with vector **x**:

$$\boxed{y = \mathbf{w}^T \cdot \mathbf{x} + b} = \ w1 \cdot x1 + w2 \cdot x2 + w3 \cdot x3 + \ldots + wM \cdot xM + b$$

e.g. we want to predict the **price of a house (y)** based on:

x1 = square-meters (sqm)

x2,3 = location (lat, lon)

x4 = year of construction (yoc)

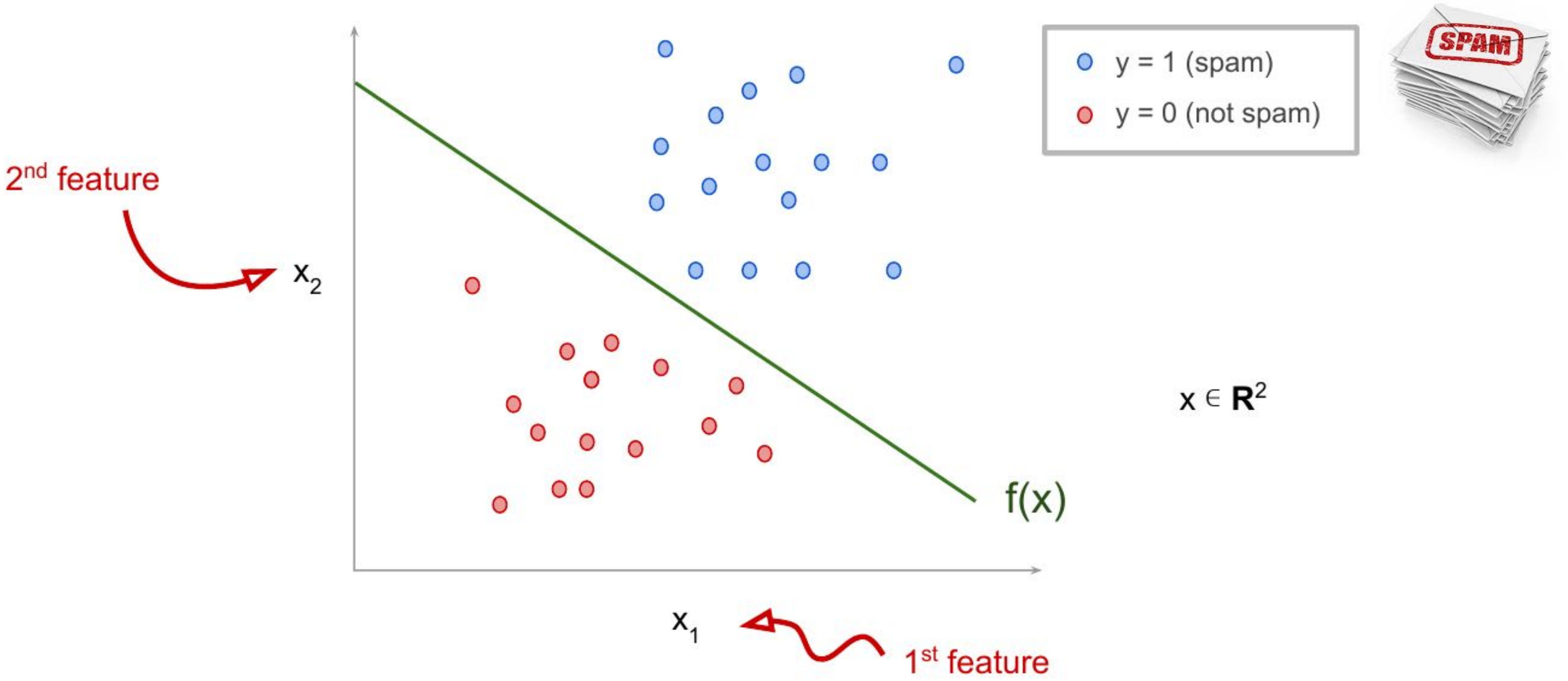
y = price = *w1*·(sqm) + *w2*·(lat) + *w3*·(lon) + *w4*·(yoc) + b

Regression vs Classification

Depending on the type of target $\mathbf{y}$ we get:

- **Regression:** $\mathbf{y} \in \mathbb{R}^N$ is continuous (e.g. temperatures $\mathbf{y}$ = {19º, 23º, 22º})
- **Classification:** $\mathbf{y}$ is discrete (e.g. $\mathbf{y}$ = {1, 2, 5, 2, 2}).

Binary Classification (eg. 2D input, 1D ouput)

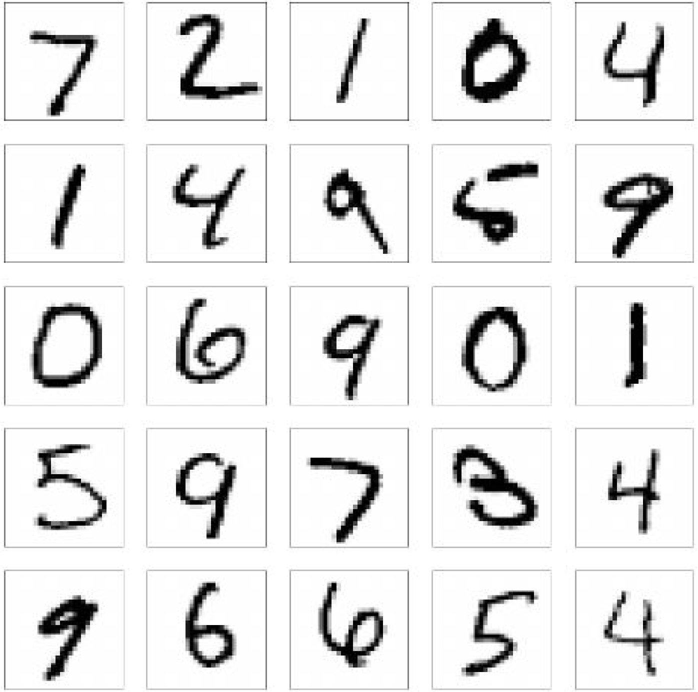


Multi-class Classification

Produce a classifier to map from pixels to the digit.

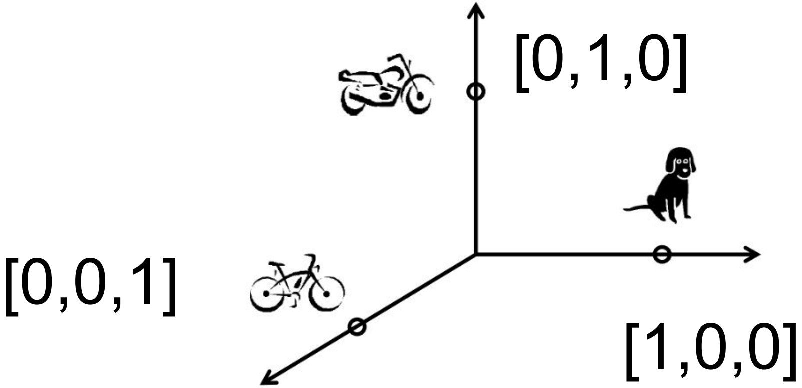
- If images are grayscale and 28×28 pixels in size, then $\mathbf{x}_i \in \mathbb{R}^{784}$
- $y_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Example of a **multi-class classification** task.



Multi-class Classification

- **Classification:** $\mathbf{y}$ is discrete (e.g. $\mathbf{y} = \{1, 2, 5, 2, 2\}$).
  - Beware! These are unordered categories, not numerically meaningful outputs: e.g. code[1] = "dog", code[2] = "cat", code[5] = "ostrich", ...
  - Classes are often coded as **one-hot vector** (each class corresponds to a different dimension of the output space)



One-hot representation

Perronin, F., CVPR Tutorial on LSVR @ CVPR'14, Output embedding for LSVR

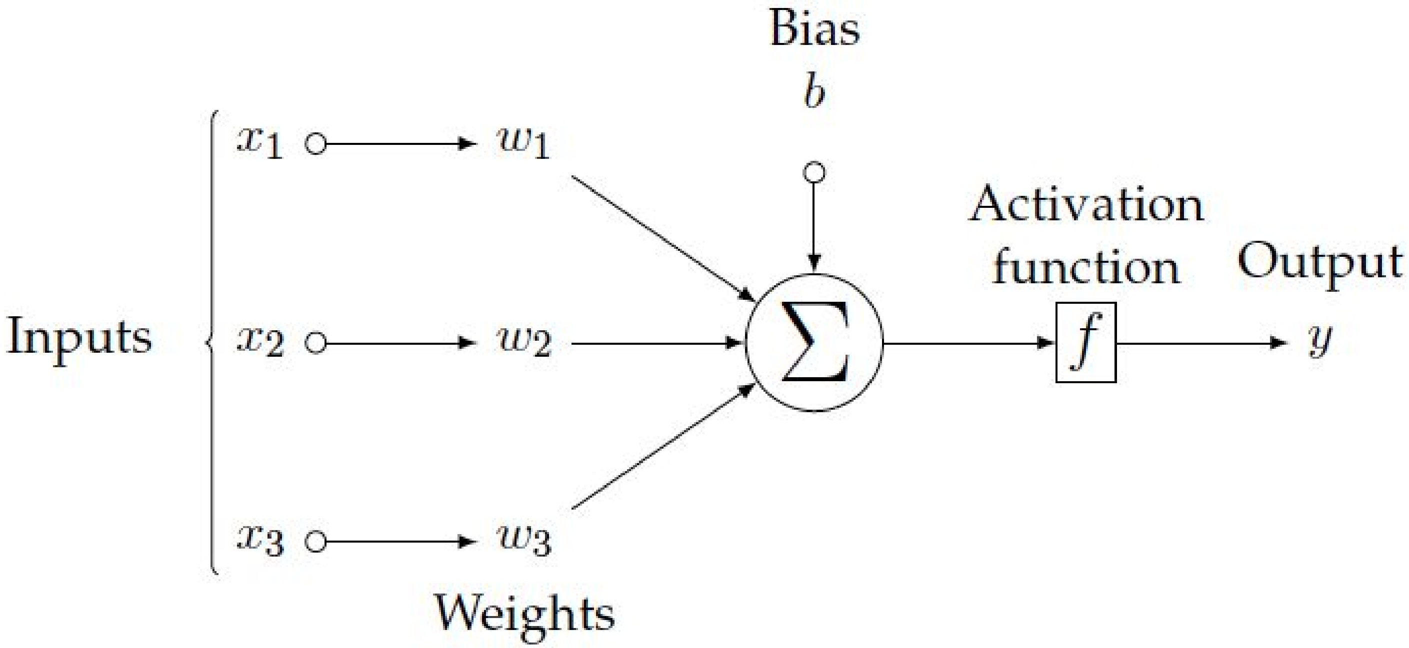
Outline

1. Supervised learning: regression/classification
2. **Single neuron models (perceptrons)**
    a. Linear regression
    b. Logistic regression
    c. Multiple outputs and softmax regression
3. Multi-Layer Perceptrons (MLP)
4. Training a deep neural network

Video lecture (DLSL 2017)

Single Neuron Model (Perceptron)

Both regression and classification problems can be addressed with the perceptron:

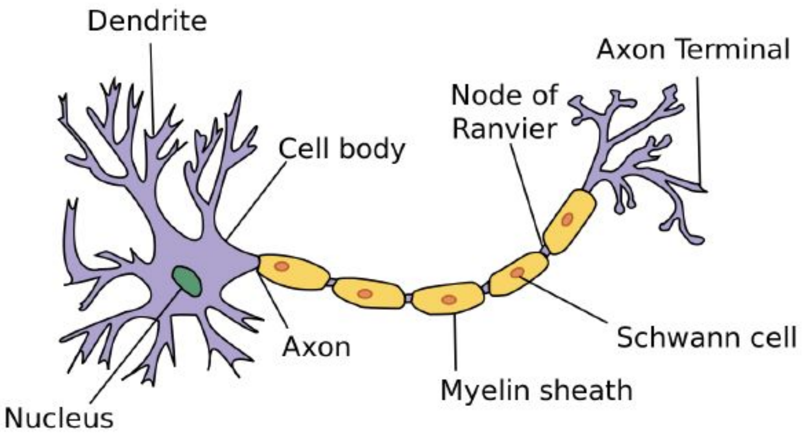


Single neuron model (perceptron)

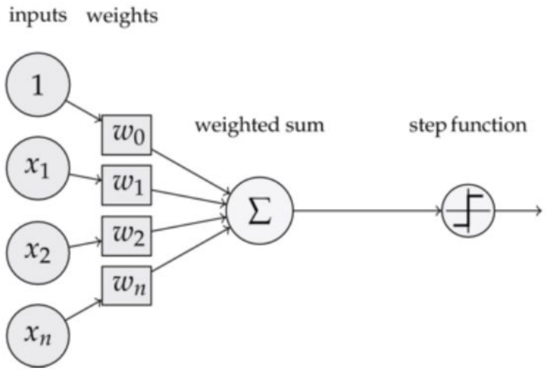
The Perceptron is seen as an **analogy** to a biological neuron.

Biological neurons fire an impulse once the sum of all inputs is over a threshold.

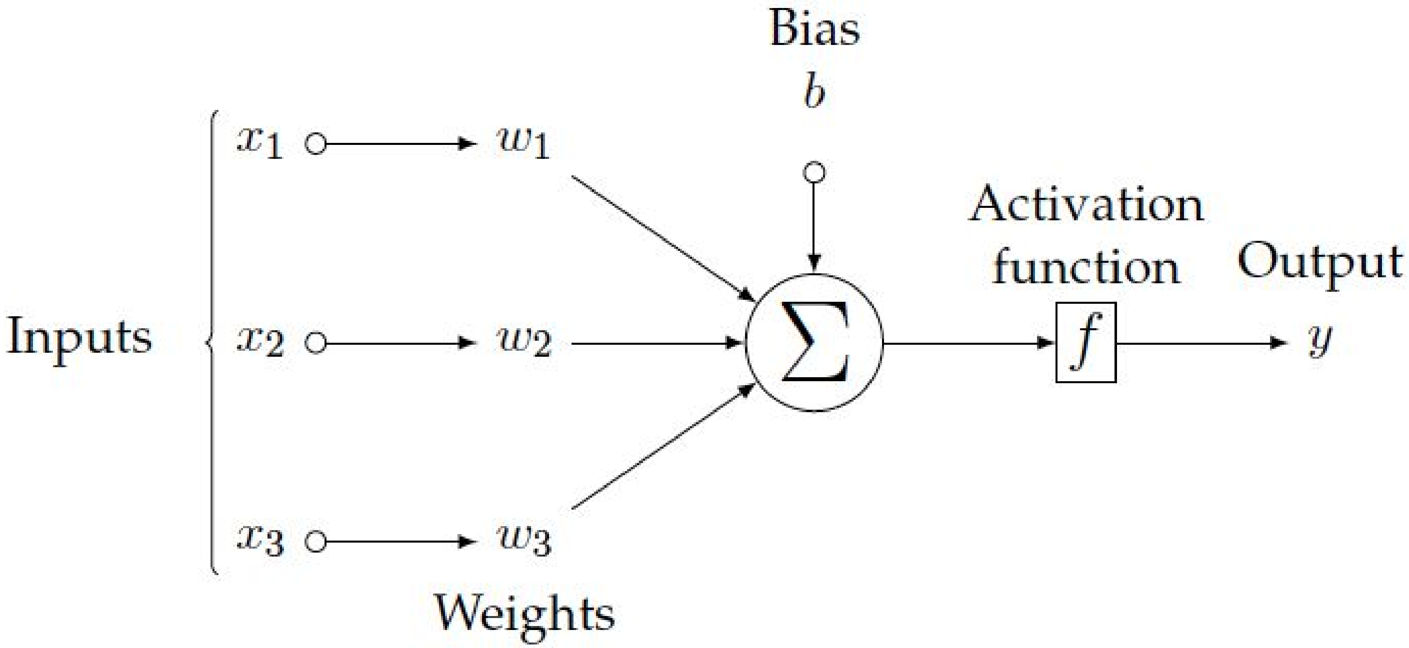
The perceptron acts like a switch (learn how in the next slides...).



Rosenblatt's Perceptron (1958)

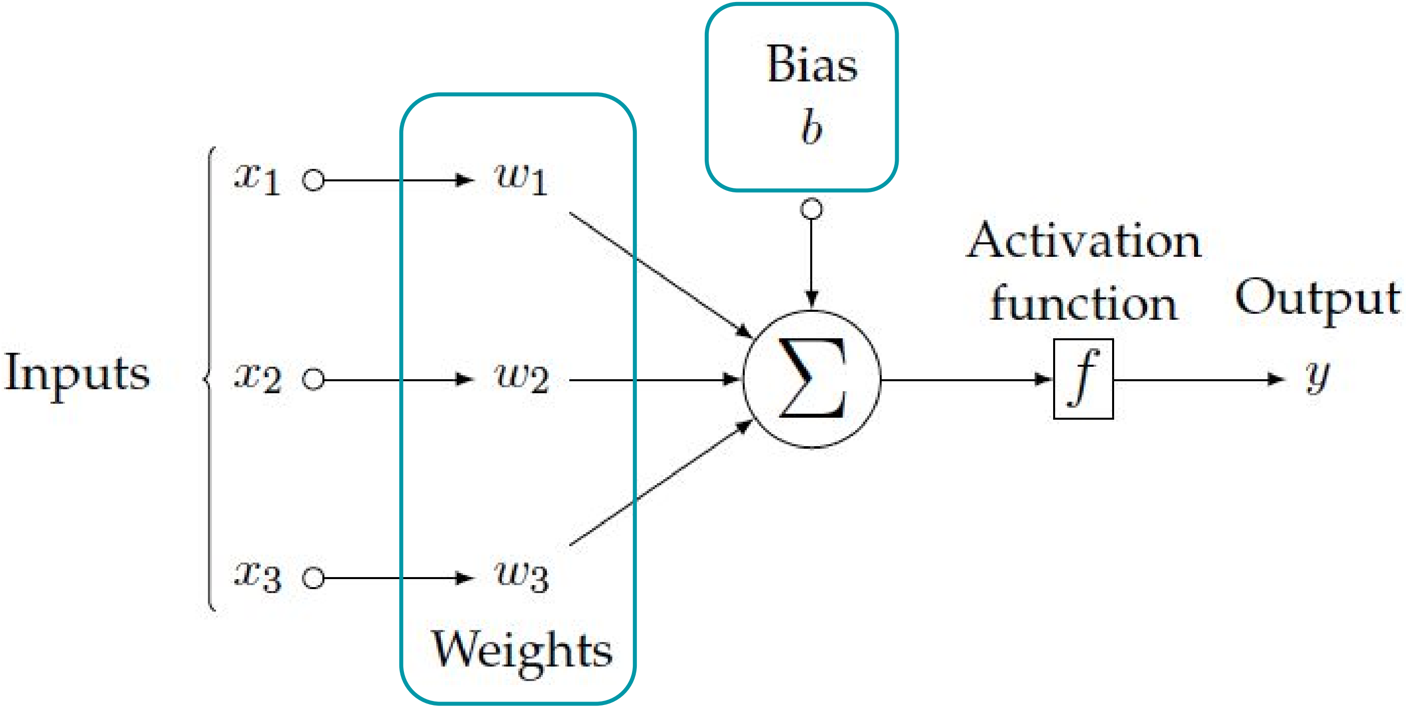


Single neuron model (perceptron)



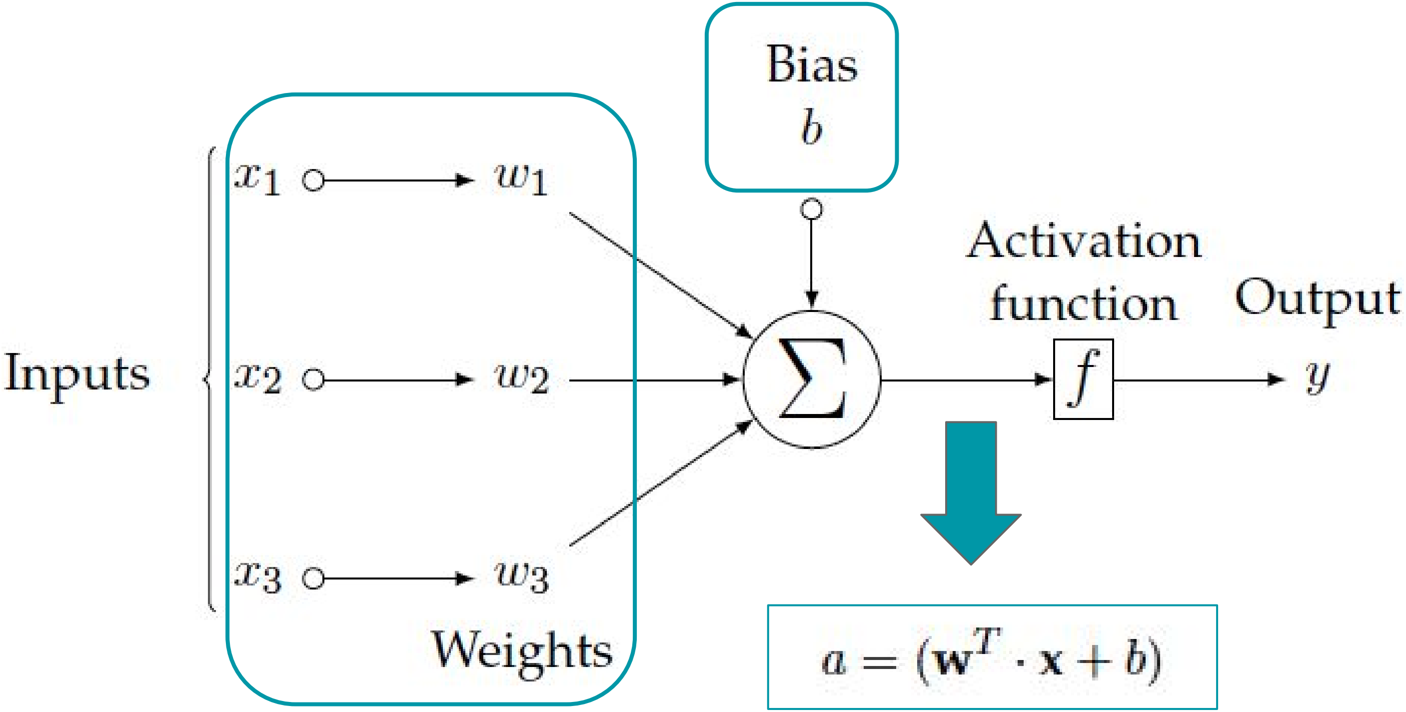
Single neuron model (perceptron)

Weights and bias are the parameters that define the behavior (must be learned).



Single neuron model (perceptron)

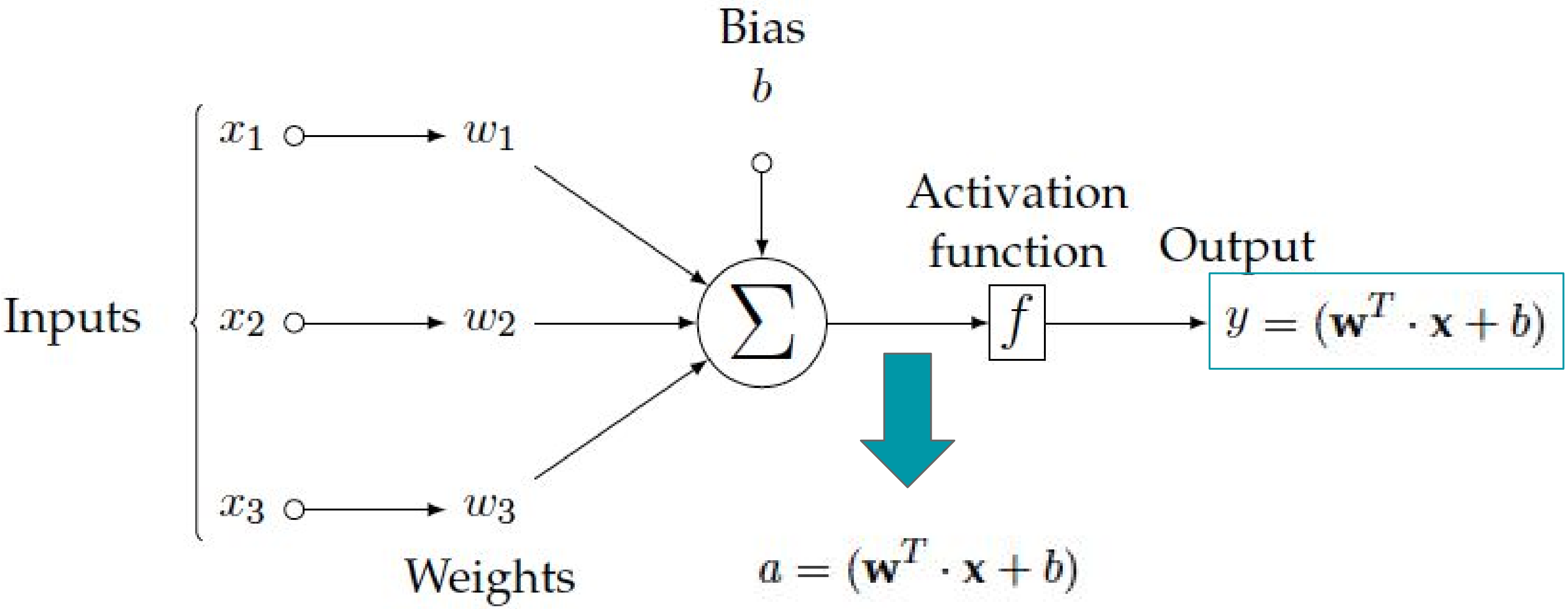
The output y is derived from a sum of the **weighted** inputs plus a **bias** term.



$$a = (\mathbf{w}^T \cdot \mathbf{x} + b)$$

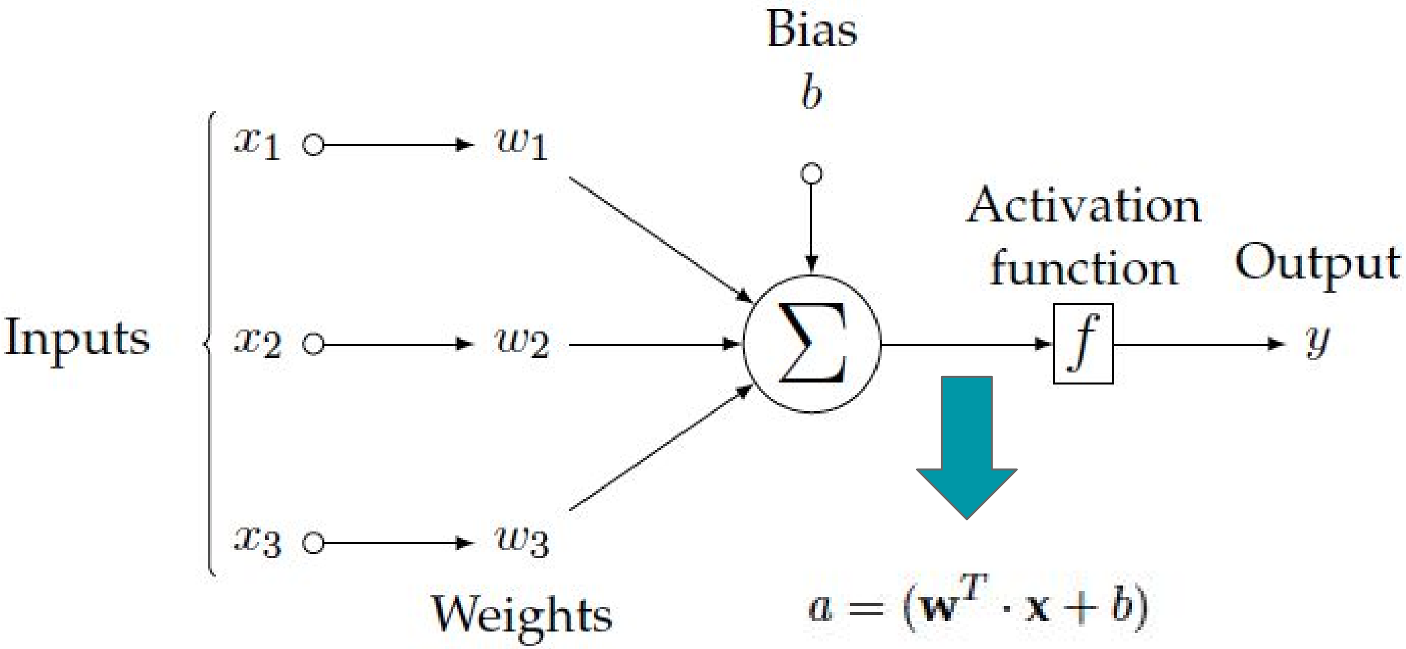
Single neuron model: Regression

The perceptron can solve regression problems when f(a)=a. [identity]



Single neuron model: Binary Classification

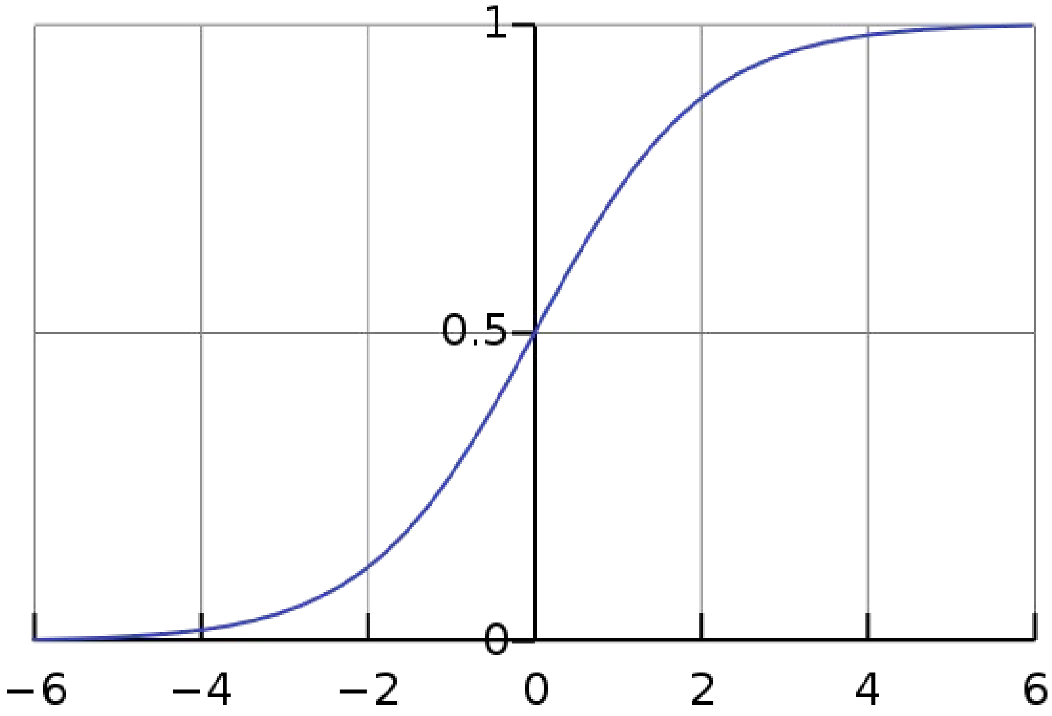
The perceptron can solve classification problems when f(a)=σ(a). [sigmoid]



Single neuron model: Binary Classification

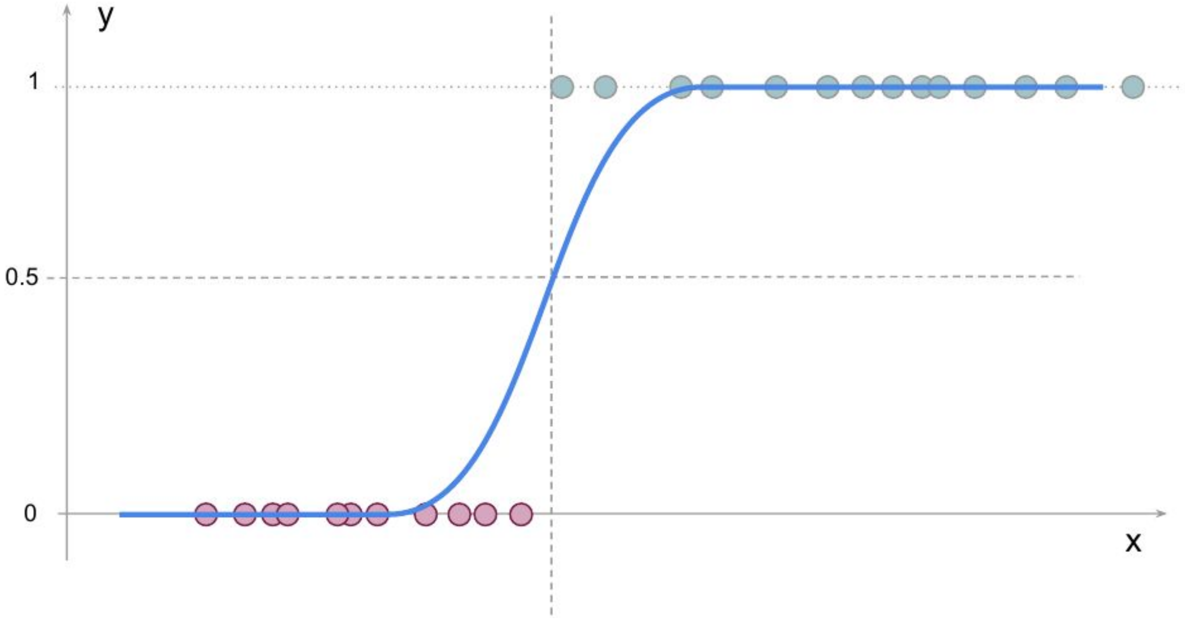
The **sigmoid function σ(x)** or **logistic curve** maps any input x between [0,1]:

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



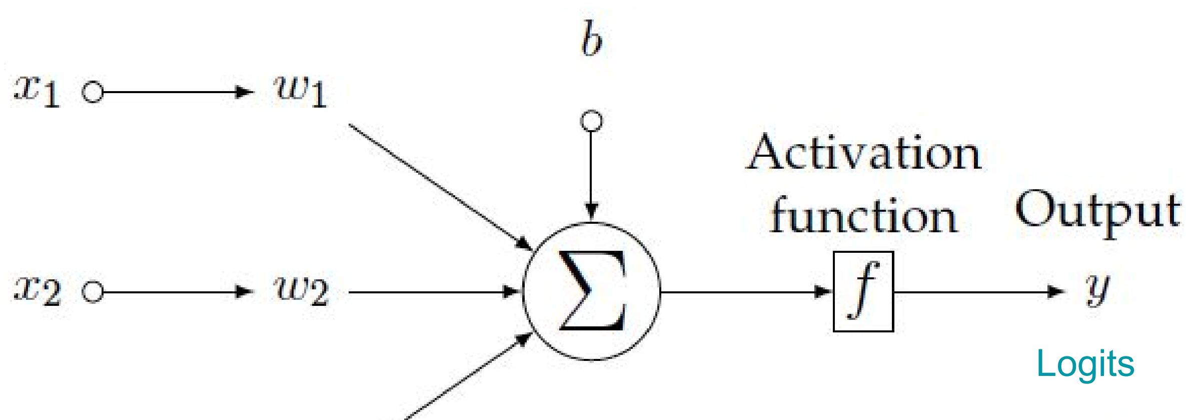
Single neuron model: Binary Classification

For classification, regressed values must be bounded between 0 and 1 to represent probabilities.



Single neuron model: Binary Classification

Setting a **threshold (thr)** at the output of the perceptron allows solving classification problems between two classes (binary) & estimate probabilities:



$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \cdot \mathbf{x} + b}}$$

y > thr → class 1 (eg. green)

y < thr → class 2 (eg. no green)

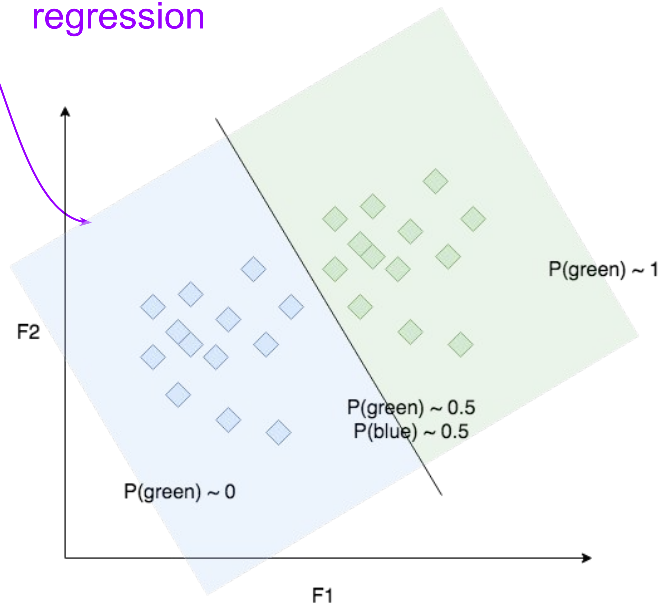
Single neuron model: Binary Classification

Setting a **threshold (thr)** at the output of the perceptron allows solving classification problems between two classes (binary) & estimate probabilities:

Logistic regression

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \cdot \mathbf{x} + b}}$$

Linear regression



Softmax classifier: Mulitclass

Ideally we would like to predict the *probability* that y takes a particular value given $\mathbf{x}$,

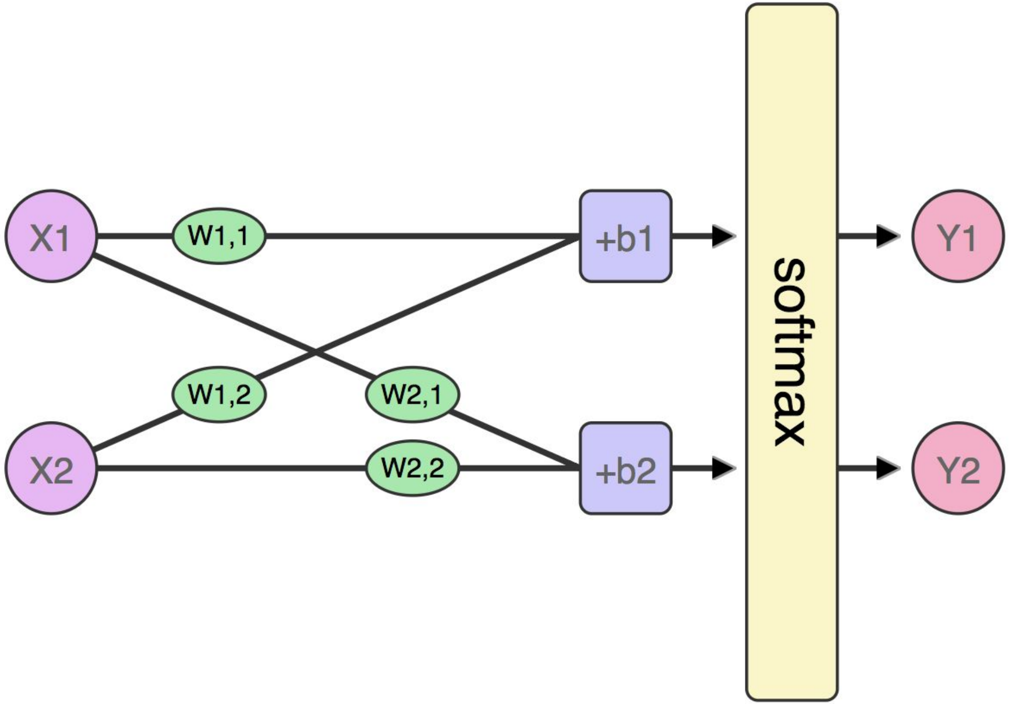
$$P(y = j|\mathbf{x}), \quad j \in \{1, \ldots, K\}$$

with:

$$\sum_{j=1}^{K} P(y = j|x) = 1$$

The logistic regression classifier does this for the binary case. The **softmax classifier** extends it to the multiclass case.

Softmax classifier: Multiclass



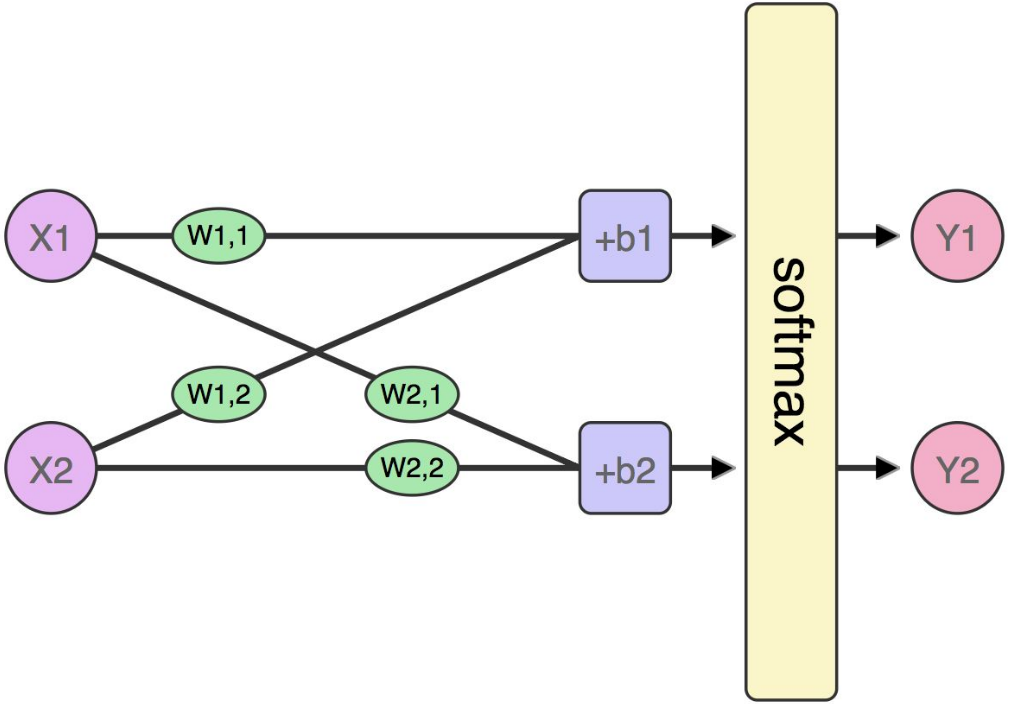
Probability estimations for each class can also be obtained by **softmax normalization** on the output of two neurons, one specialised for each class.

Softmax regression

$$P(y=k|\mathbf{x}) = \frac{\exp \mathbf{x}^T \mathbf{w}_k}{\sum_{n=1}^{N} \exp \mathbf{x}^T \mathbf{w}_n}$$

J. Alammar, "A visual and interactive guide to the Basics of Neural Networks" (2016)

Softmax classifier: Multiclass



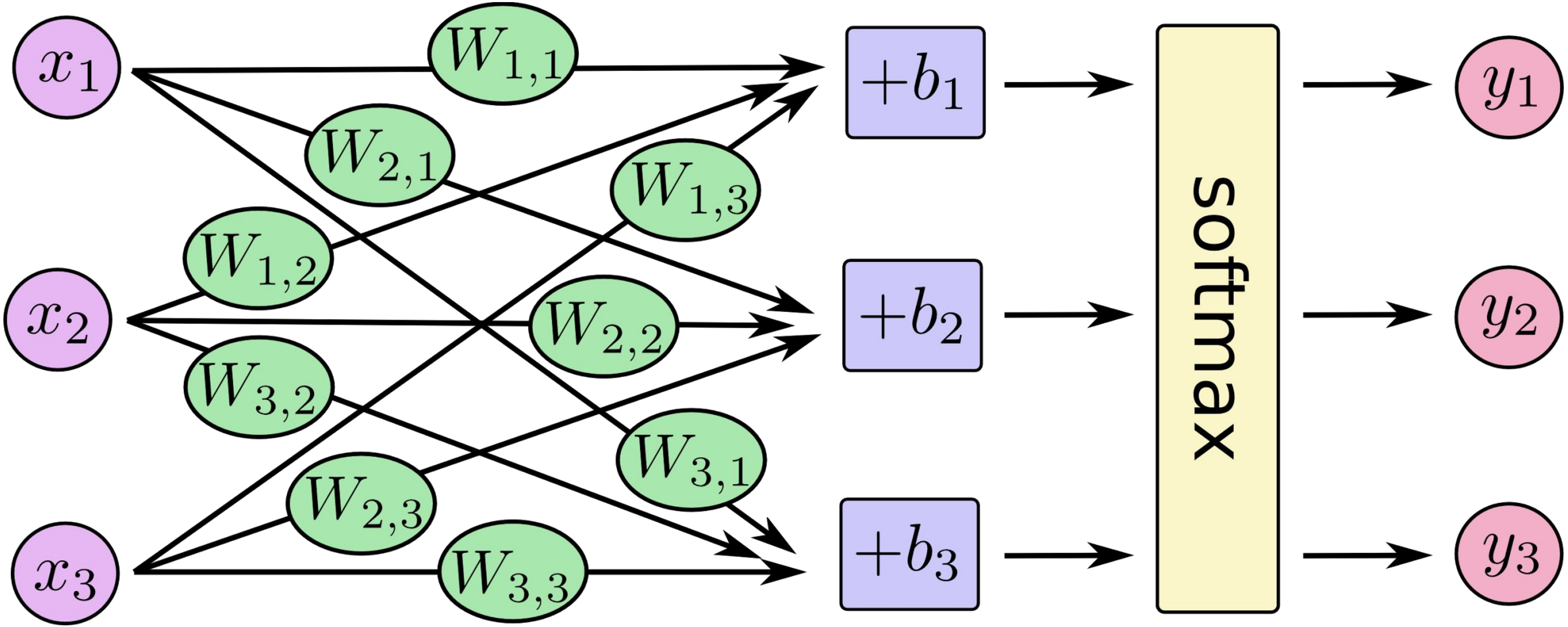
Softmax regression

$$P(y = k|\mathbf{x}) = \frac{\exp \mathbf{x}^T \mathbf{w}_k}{\boxed{\sum_{n=1}^{N} \exp \mathbf{x}^T \mathbf{w}_n}}$$

Normalization factor so that the sum of probabilities sum up to 1.

J. Alammar, [“A visual and interactive guide to the Basics of Neural Networks”](#) (2016)

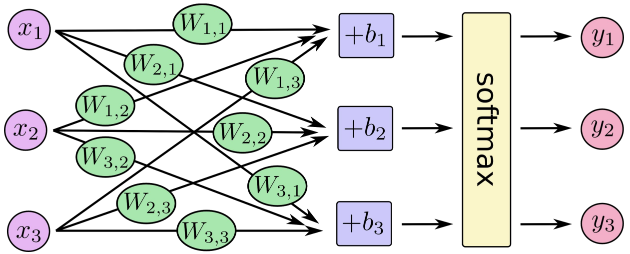
Softmax classifier: Multiclass (3 classes)



TensorFlow, “MNIST for ML beginners”

Softmax classifier: Multiclass (3 classes)

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

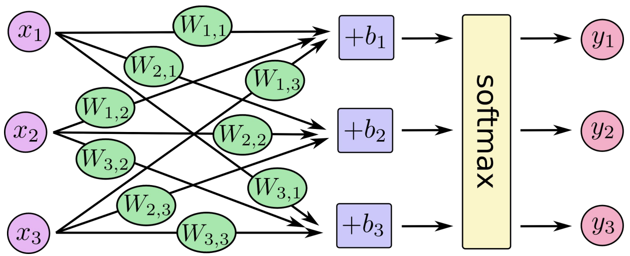


TensorFlow, “MNIST for ML beginners”

Softmax classifier: Multiclass (3 classes)

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$\boxed{y = \text{softmax}(Wx + b)}$$

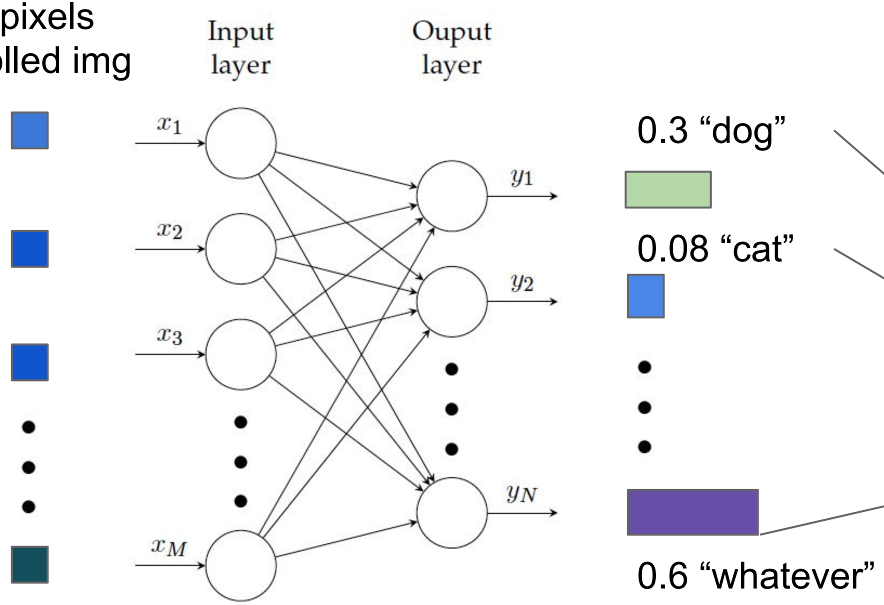


TensorFlow, [“MNIST for ML beginners”]

Softmax classifier: Multiclass (3 classes)

Multiple classes can be predicted by putting many neurons in parallel, each processing its binary output out of N possible classes.

raw pixels
unrolled img



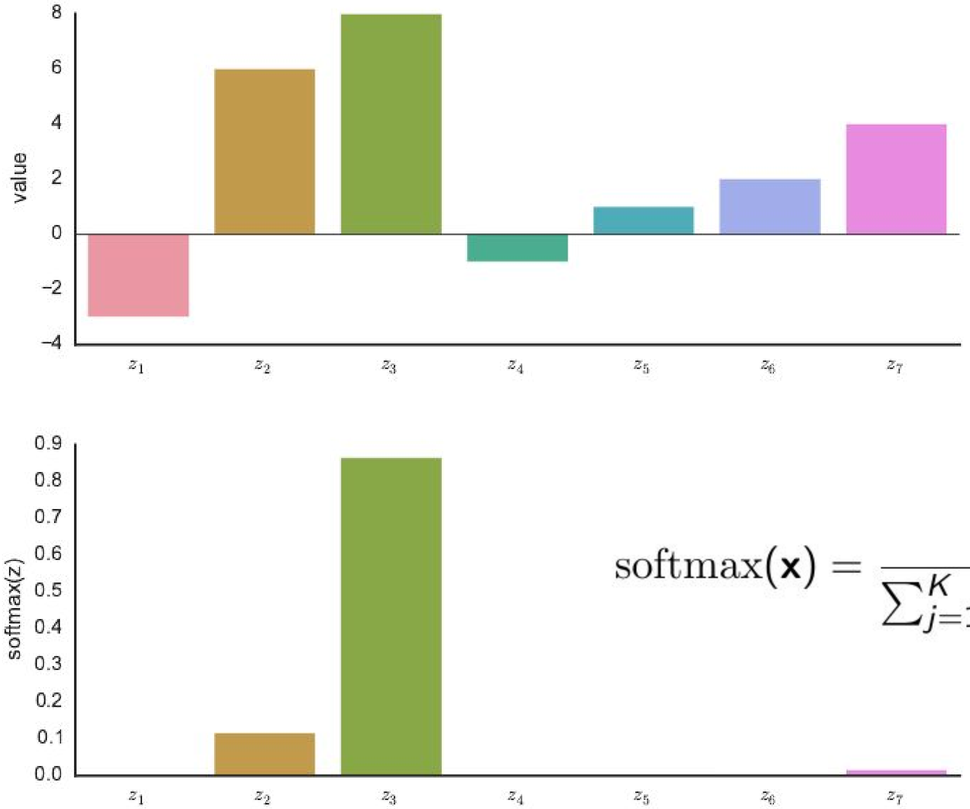
Softmax function

$$P(y = k|\mathbf{x}) = \frac{\exp \mathbf{x}^T \mathbf{w}_k}{\sum_{n=1}^{N} \exp \mathbf{x}^T \mathbf{w}_n}$$

Normalization factor, remember: we want a pdf at the output! → all output P’s sum up to 1.

38

Effect of the softmax



$$\mathrm{softmax}(\mathbf{x}) = \frac{1}{\sum_{j=1}^{K} \exp(x_j)} \begin{bmatrix} \exp(x_1) \\ \exp(x_2) \\ \vdots \\ \exp(x_K) \end{bmatrix}$$

Outline

1. Supervised learning: regression/classification
2. Single neuron models (perceptrons)
    a. Linear regression
    b. Logistic regression
    c. Multiple outputs and softmax regression
3. **Multi-Layer Perceptrons (MLP)**
4. Training a deep neural network

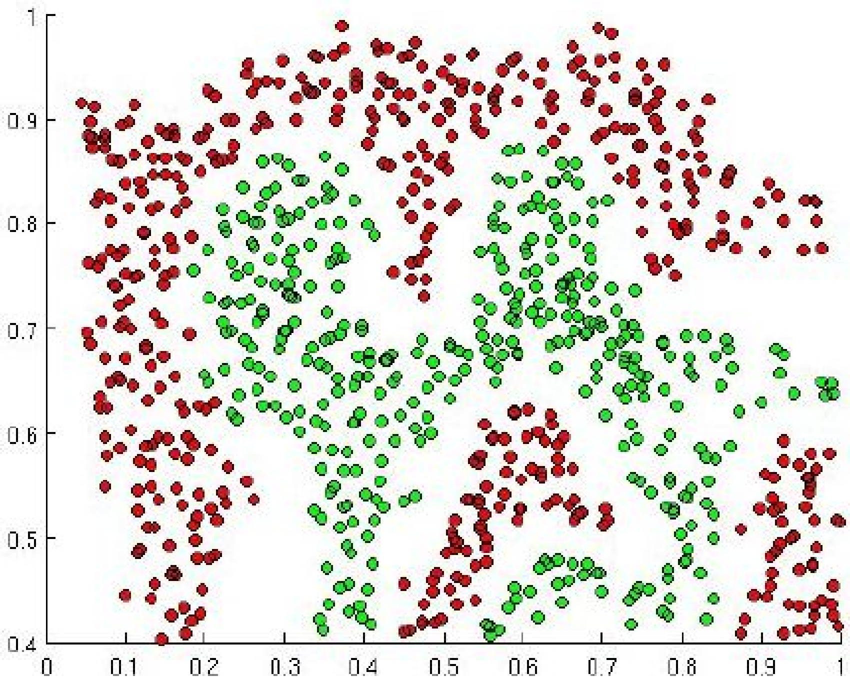
Limitation of the Perceptron

Perceptrons can only produce linear decision boundaries.

Many interesting problems are not linearly separable.

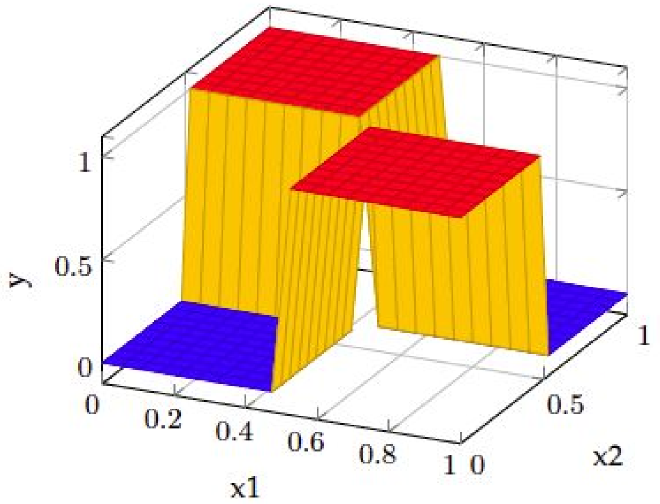
Real world problems often need non-linear boundaries

- Images
- Audio
- Text



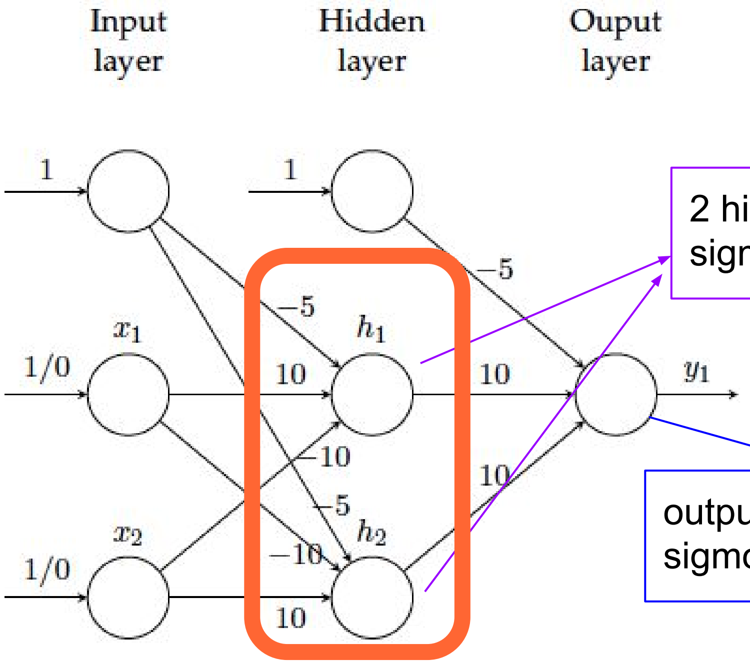
Limitation of the Perceptron

The XOR problem: sometimes a single neuron is not enough → Just a single decision split doesn't work



Multi-Layer Perceptron (MLP)

Solution: arrange many neurons in a first intermediate **non-linear** mapping (Hidden Layer), **connecting everything** from layer to layer in a **feed-forward** fashion.



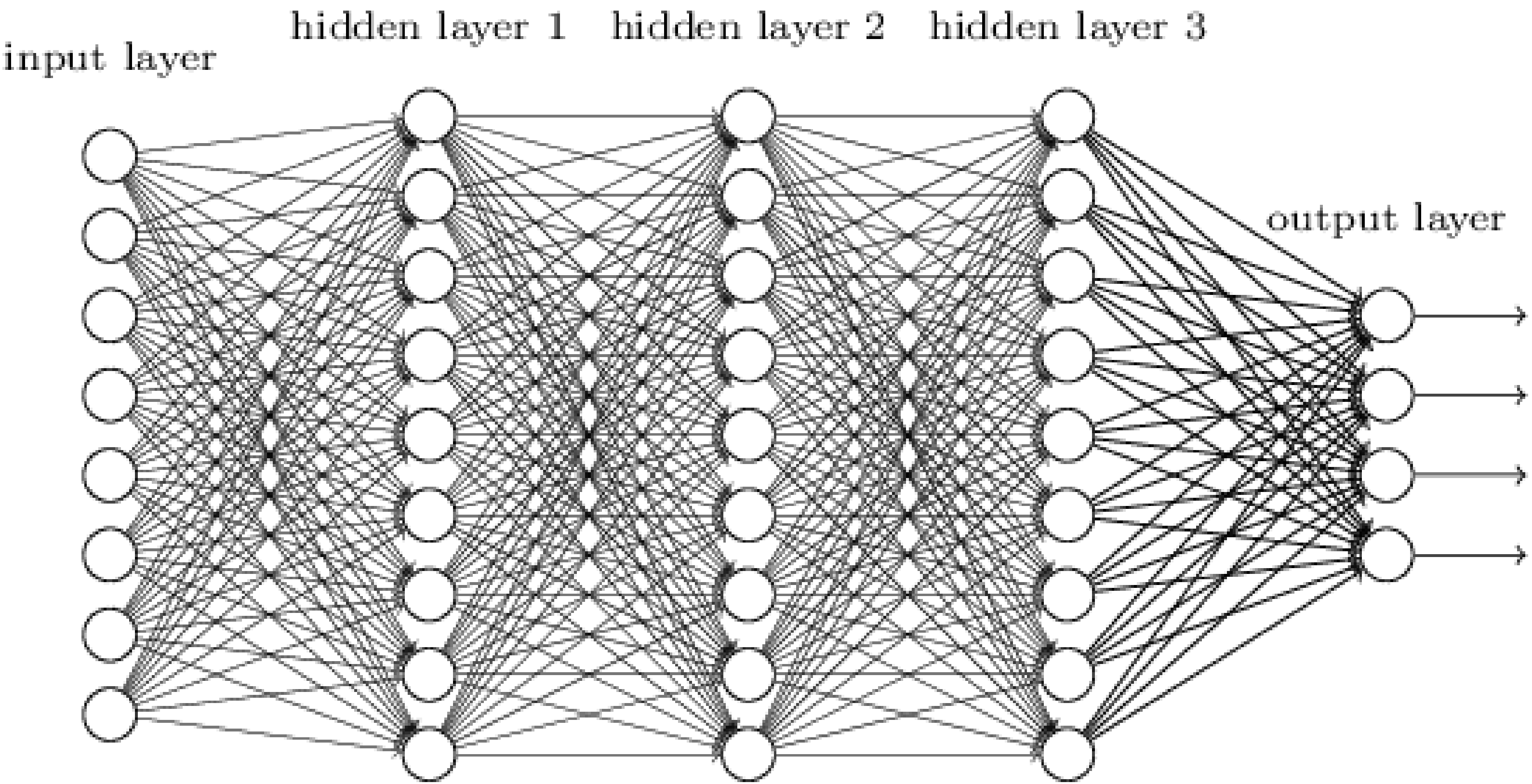
Warning! Inputs are not neurons, but they are usually depicted like neurons.

2 hidden units with sigmoid activations

Example XOR NN

output unit with sigmoid activation

Multi-Layer Perceptron (MLP)



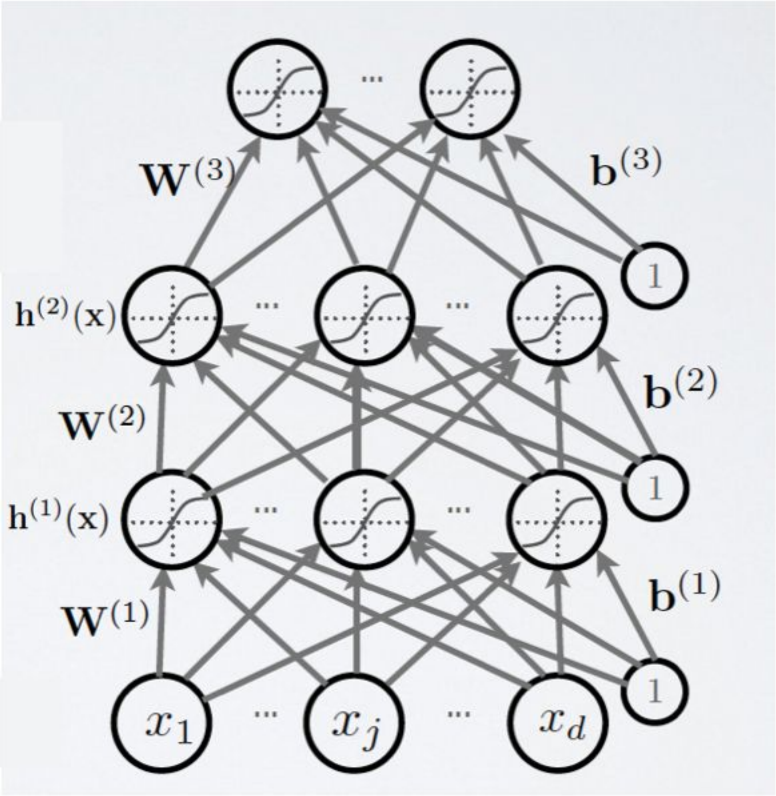
Multi-Layer Perceptron (MLP)

The i-th layer is defined by a matrix $\mathbf{W}_i$ and a vector $\mathbf{b}_i$, and the activation is simply a dot product plus $\mathbf{b}_i$:

$$\boldsymbol{h_i} = f(\boldsymbol{W_i} \cdot \boldsymbol{h_{i-1}} + \boldsymbol{b_i})$$

Num parameters to learn at i-th layer:

$$N^i_{params} = N^i_{inputs} \times N^i_{units} + N^i_{units}$$



Slide Credit: Hugo Laroche NN course

Multi-Layer Perceptron (MLP)

Important remarks:

- We can put as many hidden layers as we want whenever training can be effectively done and we have enough data
  - **The amount of parameters to estimate grows very quickly** with the num of layers and units! → There is **no formula to know the amount of units per layer nor the amount of layers**, pitty...
- **The power of NNets comes from non-linear mappings**: hidden units must be followed by a non-linear activation!
  - *sigmoid, tanh, relu, leaky-relu, prelu, exp, softplus, …*

Outline

1. Supervised learning: regression/classification
2. Single neuron models (perceptrons)
    a. Linear regression
    b. Logistic regression
    c. Multiple outputs and softmax regression
3. Multi-Layer Perceptrons (MLP)
4. **Training a deep neural network**

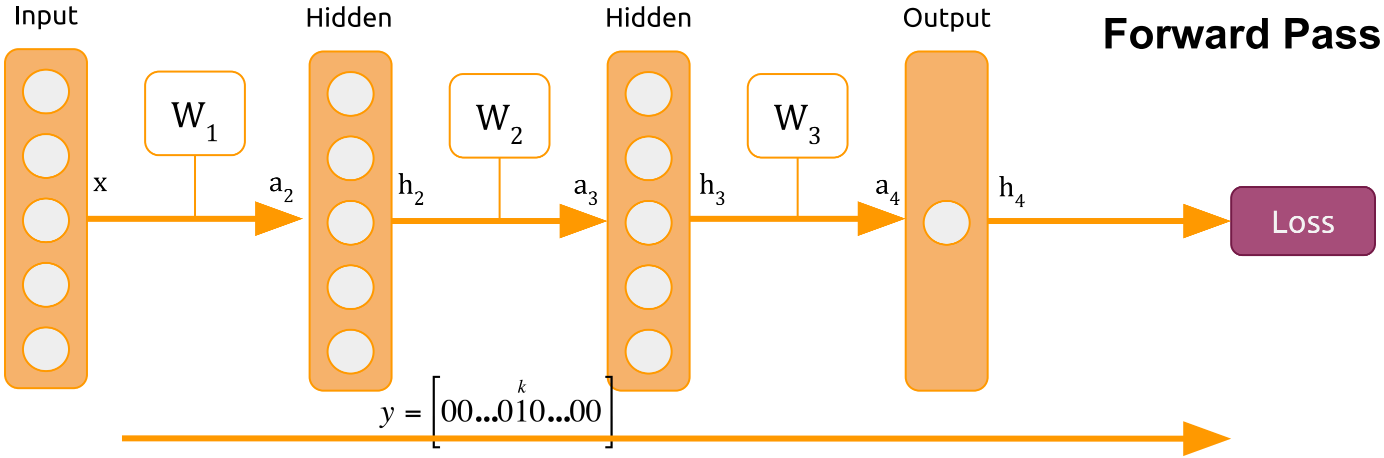
Backpropagation algorithm

The output of the classification network gives class **scores** that depens on the input and the parameters

$$f(\mathbf{x}) = \mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{o}(\mathbf{b}^{(L)} + \mathbf{W}^{(L)}\mathbf{h}^{(L)}(\mathbf{x}))$$

- Define a **loss function** that quantifies our unhappiness with the scores across the training data.

- Come up with a way of efficiently finding the parameters that minimize the loss function (**optimization**)



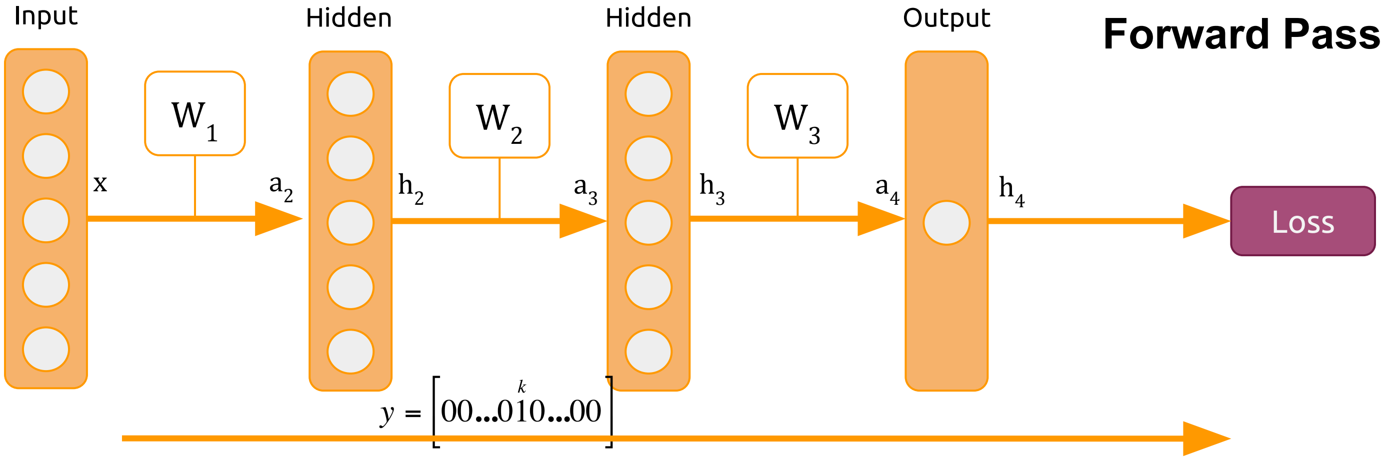
Forward Pass

Probability Class given an input (softmax)

$$p(c_k = 1 | \mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

Figure Credit: Kevin McGuiness

Forward Pass



Probability Class given an input (softmax)

$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

Loss function; e.g., negative log-likelihood (good for classification)

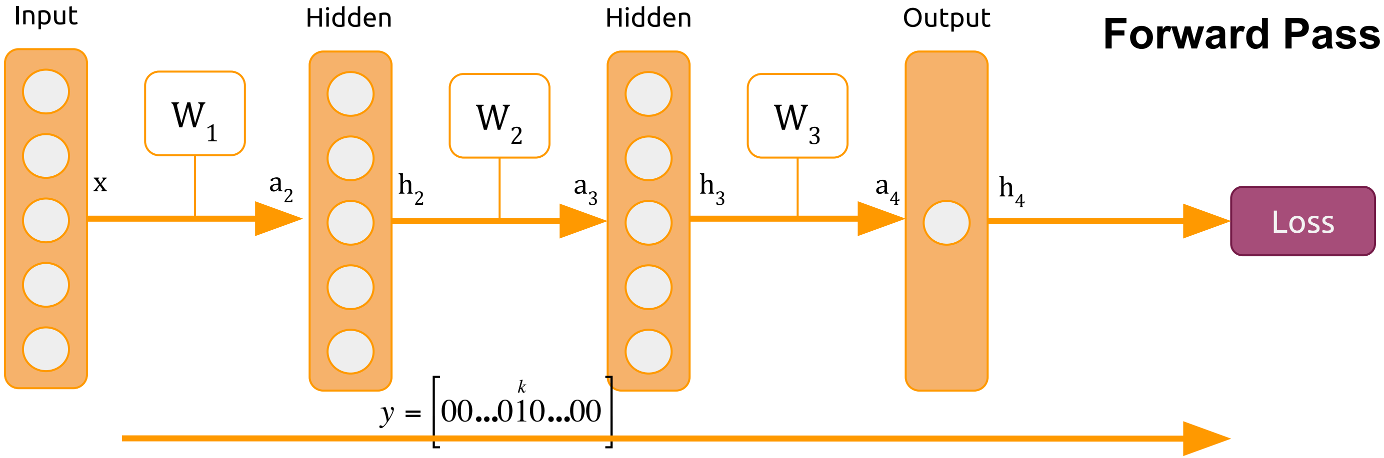
$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x}))$$

Regularization term (L2 Norm) aka as weight decay

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x})) + \frac{\lambda}{2}\|\mathbf{W}\|_2^2$$

Figure Credit: Kevin McGuiness

Forward Pass



Probability Class given an input (softmax)

$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

Loss function; e.g., negative log-likelihood (good for classification)

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x}))$$

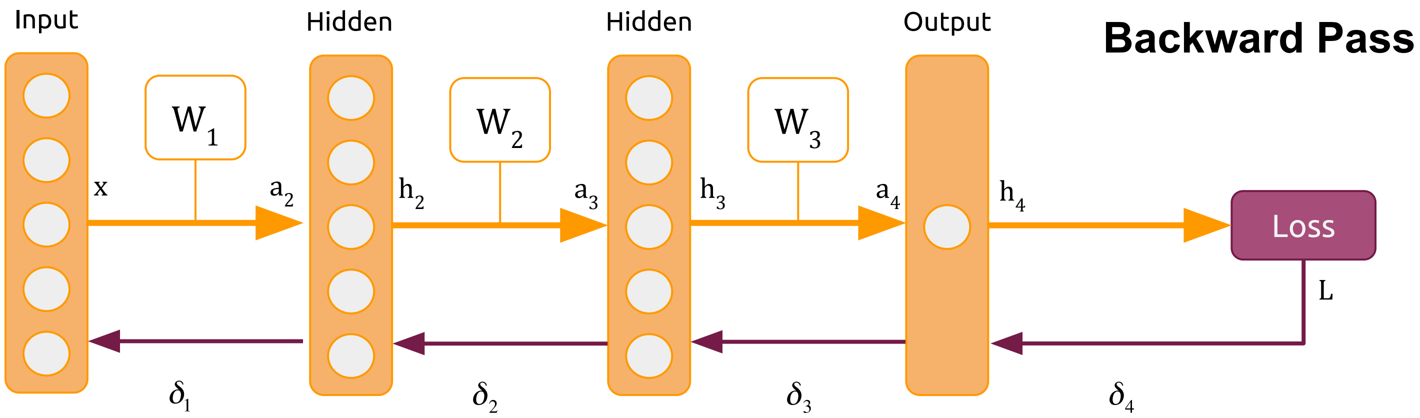
Regularization term (L2 Norm) aka as weight decay

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x})) + \frac{\lambda}{2}\|\mathbf{W}\|_2^2$$

Minimize the loss (plus some regularization term) w.r.t. Parameters over the whole training set.

$$\mathbf{W}^* = argmin_\theta \sum_j L(\mathbf{x}^n, y^n; \mathbf{W})$$

Figure Credit: Kevin McGuiness



1. Find the error in the top layer:

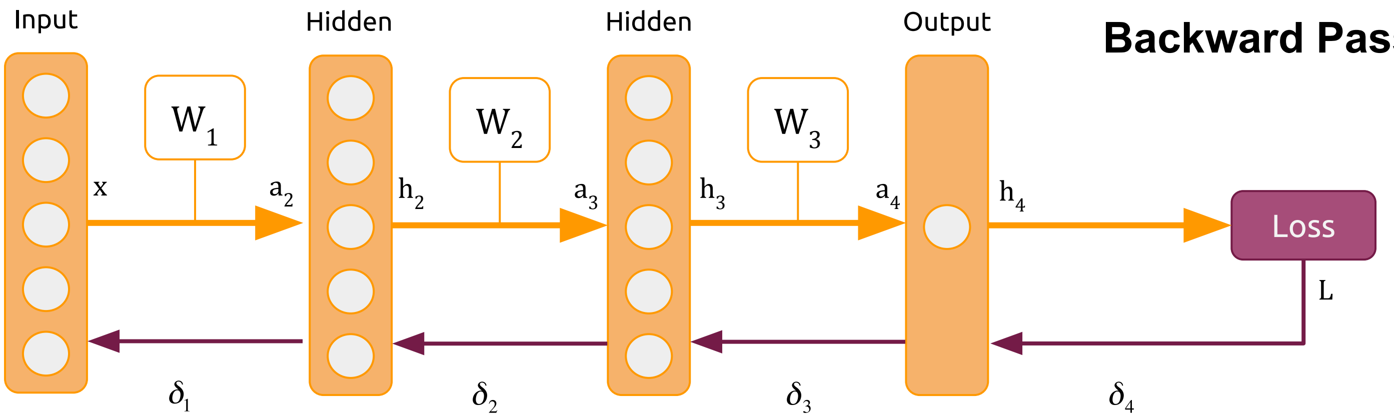
$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

Figure Credit: Kevin McGuiness

Backward Pass



1. Find the error in the top layer:

$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K}\frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

2. Compute weight updates

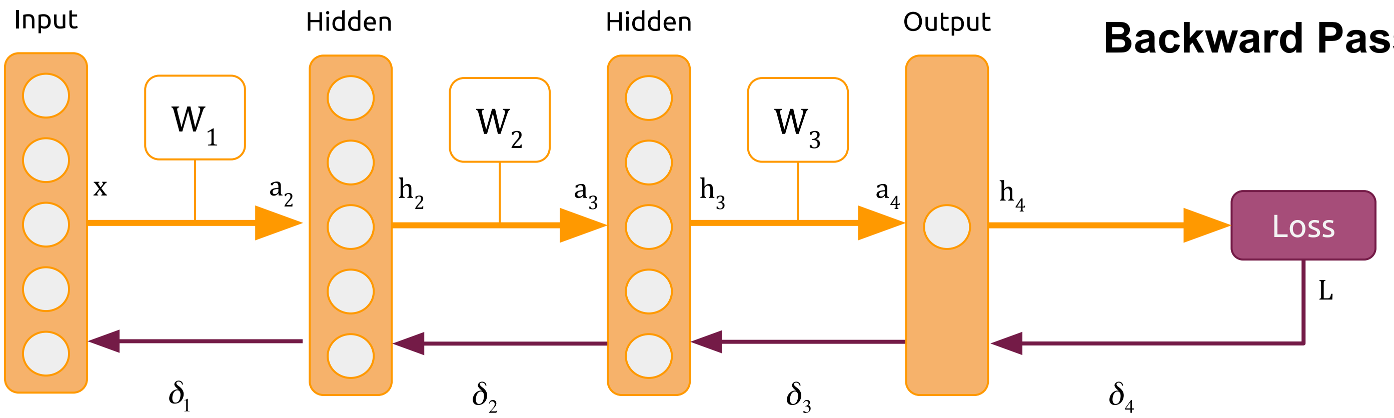
$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial W_k}$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}} \bullet h_k$$

$$\frac{\partial L}{\partial W_k} = \delta_{k+1} \bullet h_k$$

To simplify we don't consider the bias

Figure Credit: Kevin McGuiness



Backward Pass

1. Find the error in the top layer:

$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K}\frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

2. Compute weight updates

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial W_k}$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}} \bullet h_k$$

$$\frac{\partial L}{\partial W_k} = \delta_{k+1} \bullet h_k$$

To simplify we don't consider the bias

3. Backpropagate error to layer below

$$\delta_k = \frac{\partial L}{\partial a_k}$$

$$\delta_k = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial h_k}\frac{\partial h_k}{\partial a_k}$$

$$\delta_k = W_k^T \frac{\partial L}{\partial a_{k+1}} \bullet g'(a_k)$$

$$\delta_k = W_k^T \delta_{k+1} \bullet g'(a_k)$$

Figure Credit: Kevin McGuiness

Optimization

Stochastic Gradient Descent

$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \frac{\partial L}{\partial W}$$

η : learning rate

Stochastic Gradient Descent with momentum

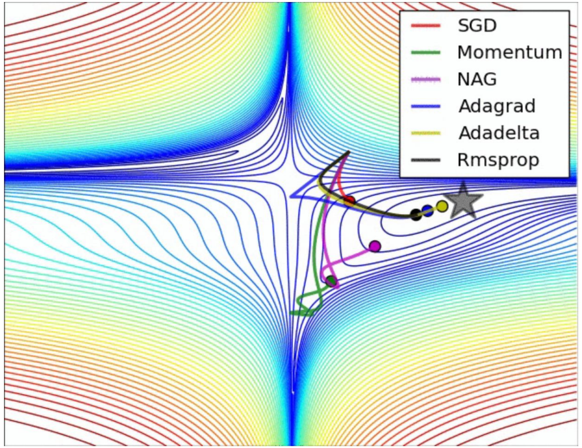
$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \Delta$$

$$\Delta \longleftarrow 0.9\Delta + \frac{\partial L}{\partial \mathbf{W}}$$

Stochastic Gradient Descent with L2 regularization

$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \frac{\partial L}{\partial W} - \lambda |\mathbf{W}|$$

λ : weight decay



Sebastian Ruder Blog

Recommended lectures:

[Backpropagation: http://cs231n.github.io/optimization-2/](http://cs231n.github.io/optimization-2/)
[Optimization: http://sebastianruder.com/optimizing-gradient-descent/](http://sebastianruder.com/optimizing-gradient-descent/)

Outline

1. Supervised learning: regression/classification

2. Single neuron models (perceptrons)

    a. Linear regression

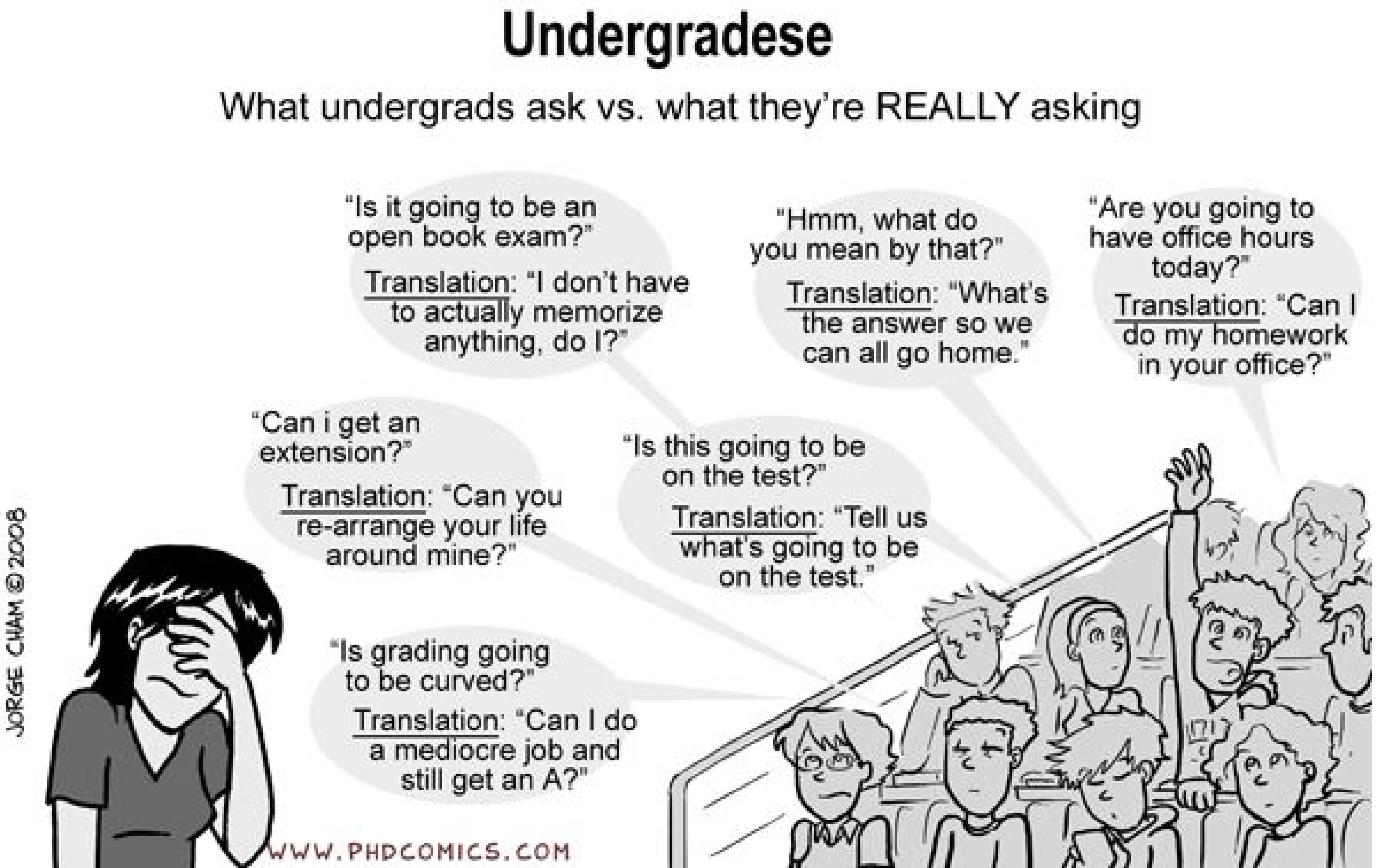
    b. Logistic regression

    c. Multiple outputs and softmax regression

3. Multi-Layer Perceptrons (MLP)

4. Training a deep neural network

Questions ?


Undergradese
What undergrads ask vs. what they're REALLY asking
"Is it going to be an open book exam?" Translation: "I don't have to actually memorize anything, do I?"
"Hmm, what do you mean by that?" Translation: "What's the answer so we can all go home."
"Are you going to have office hours today?" Translation: "Can I do my homework in your office?"
"Can i get an extension?" Translation: "Can you re-arrange your life around mine?"
"Is this going to be on the test?" Translation: "Tell us what's going to be on the test."
"Is grading going to be curved?" Translation: "Can I do a mediocre job and still get an A?"
JORGE CHAM © 2008
WWW.PHDCOMICS.COM