Applied Deep Learning

Dr. Philippe Blaettchen
Bayes Business School (formerly Cass)

www.bayes.city.ac.uk

Learning objectives of today

Goals: Creating neural networks – from logistic regression to feed-forward networks

- Use what we have learned about linear algebra and calculus to create a logistic regression algorithm from scratch
- Understand how what we learned generalizes to neural networks in general

How will we do this?

- We start with a quick recap on our discussion about logistic regression so far
- We then implement a logistic regression algorithm with numpy only
- Next, we visualize more general neural networks with the TensorFlow playground, before defining the concepts relevant for running our own networks

Taking a step back – logistic regression

What do we actually do when training a logistic regression model?

$$X = \begin{array}{c|c} & 1 \; \cdots \; m \\ 1 & \\ \vdots & \left(\begin{array}{c} x^{(1)} \\ \vdots \\ x^{(n)} \end{array} \right) \\ n & \end{array}$$

$$Y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{pmatrix}$$

- We are given values $(\boldsymbol{x}^{(i)}, y^{(i)})$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \{0,1\}$
- Our prediction $\hat{y}^{(i)}$ should reflect the probability that $y^{(i)} = 1$: $\hat{y}^{(i)} = P\left(y^{(i)} = 1 \middle| \boldsymbol{x}^{(i)}\right)$

- We model this probability, using the sigmoid function:

What do we actually do when training a logistic regression model?

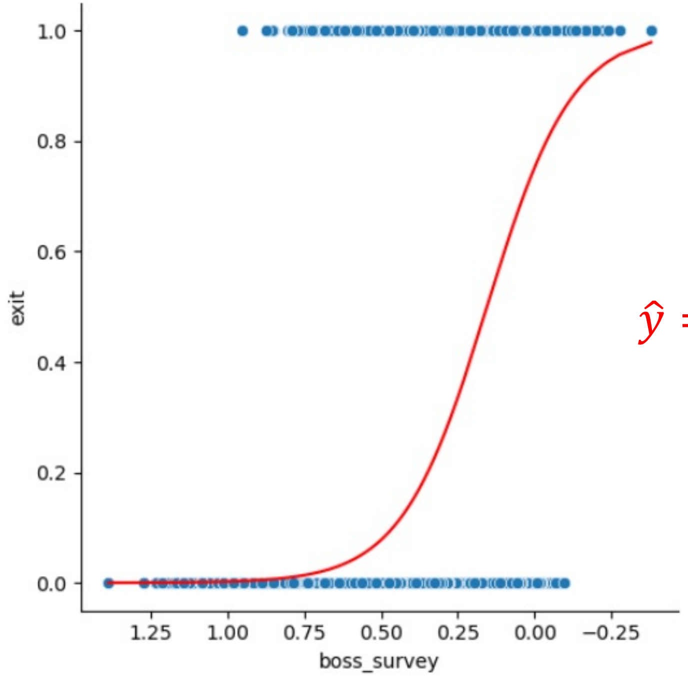
$a_0 + a_1 x_1^{(i)} + a_2 x_2^{(i)} \dots + a_m x_m^{(i)}$

$b + w_1 x_1 + w_2 x_2 + \dots + w_m x_m$

- We are given values $(\boldsymbol{x}^{(i)}, y^{(i)})$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^m$ and $y^{(i)} \in \{0,1\}$
- Our prediction $\hat{y}^{(i)}$ should reflect the probability that $y^{(i)} = 1$: $\hat{y}^{(i)} = P\left(y^{(i)} = 1 \middle| \boldsymbol{x}^{(i)}\right)$

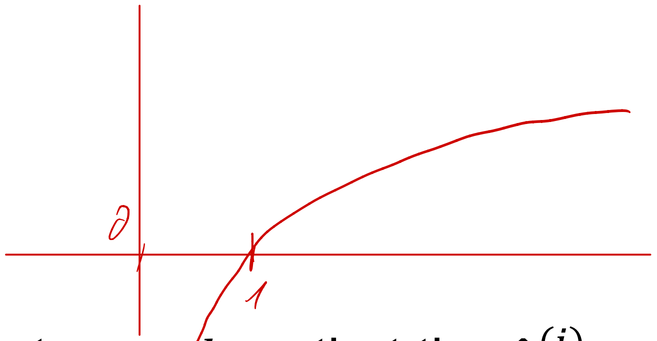
- We model this probability, using the sigmoid function:

$$\underset{(1,m)}{x}\ \underset{(m,1)}{w} + \underset{(1,1)}{b}$$



$$\hat{y} = \frac{1}{1 + e^{-(\boldsymbol{x}\cdot\boldsymbol{w}+\boldsymbol{b})}} = \frac{e^{(\dots)}}{1 + e^{(\dots)}}$$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON



- Remember that $\boldsymbol{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$
- To get to the "right" model, we optimize our parameters $\boldsymbol{w}, b$ so that the $\hat{y}^{(i)}$s are "as close as possible" to the y^is
- What we do is to minimize the "cost-function" $J(\boldsymbol{w}, b)$, where $\hat{y}^{(i)} = \frac{1}{1+e^{-\left(\boldsymbol{x}^{(i)}\boldsymbol{w}+\boldsymbol{b}\right)}}$:

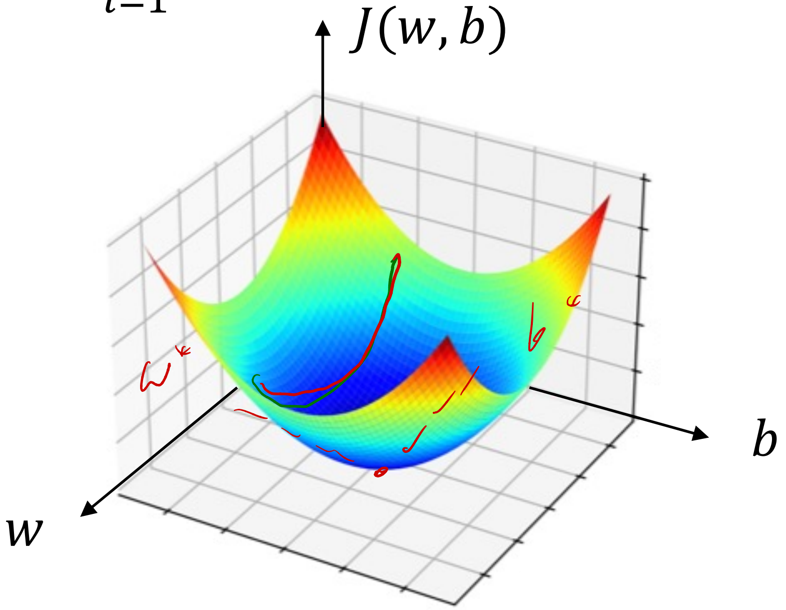
$$J(\boldsymbol{w}, b) = -\frac{1}{n}\sum_{i=1}^{n}\underbrace{\left[y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right)\ln\left(1 - \hat{y}^{(i)}\right)\right]}_{L^{(i)}}$$

$y^{(i)} = 1 : -[\ln \hat{y}^{(i)} + 0]$ $\to 0 : \hat{y}^{(i)} \to 1$; $\to \infty : \hat{y}^{(i)} \to 0$

$y^{(i)} = 0 : -[0 + \ln(1 - \hat{y}^{(i)})]$ $\to 0 : \hat{y}^{(i)} \to 0$; $\to \infty : \hat{y}^{(i)} \to 1$

- Remember that $\boldsymbol{w} \in \mathbb{R}^m$ and $b \in \mathbb{R}$
- To get to the "right" model, we optimize our parameters $\boldsymbol{w}, b$ so that the $\hat{y}^{(i)}$s are "as close as possible" to the y^is
- What we do is to minimize the "cost-function" $J(\boldsymbol{w}, b)$, where $\hat{y}^{(i)} = \frac{1}{1+e^{-\left(\boldsymbol{x}^{(i)}\boldsymbol{w}+\boldsymbol{b}\right)}}$:

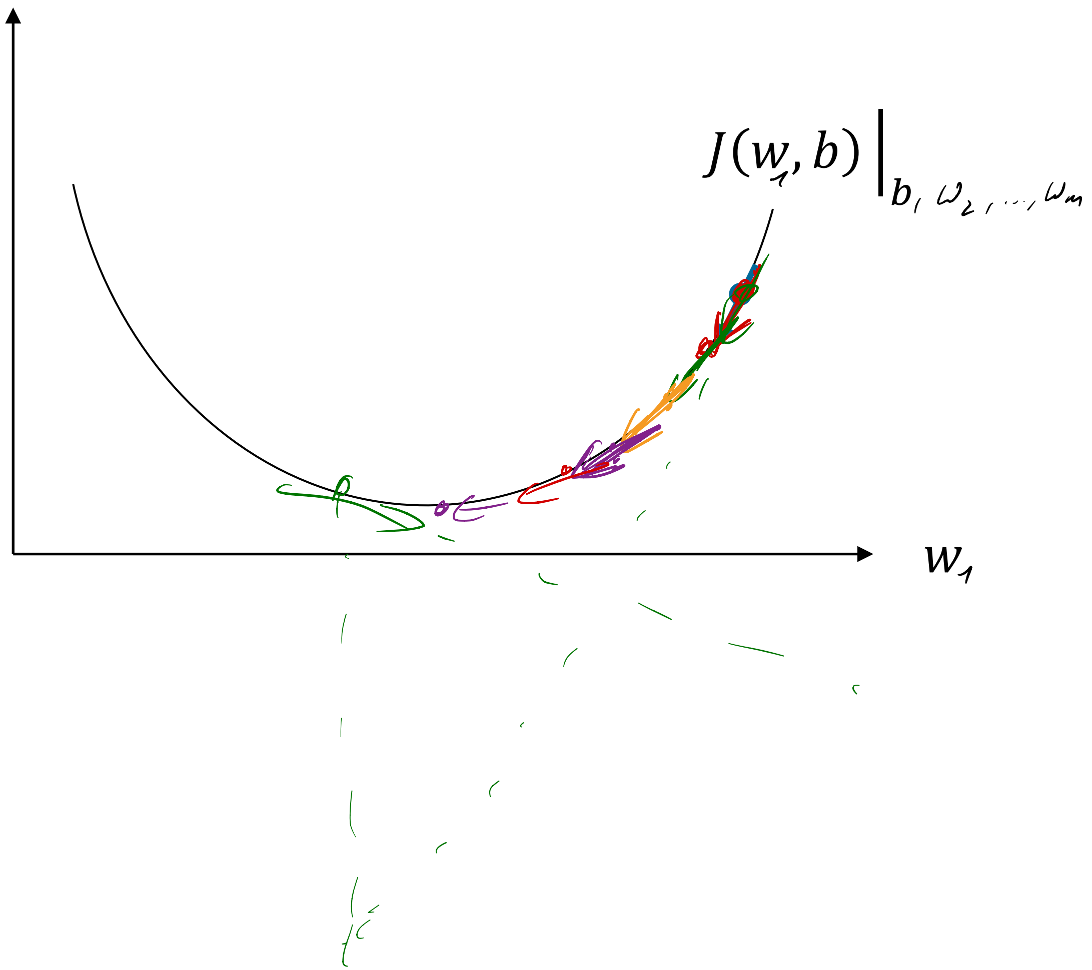
$$J(\boldsymbol{w}, b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \ln\left(1 - \hat{y}^{(i)}\right)\right]$$



$$\underline{y^{(i)}} = \frac{1}{1 + e^{-w^* x^{(i)} + b^*}}$$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Solving the optimization problem through gradient descent



$w_1 \rightsquigarrow \frac{\partial J}{\partial w_1}$

$w_1 := w_1 - \alpha \frac{\partial J}{\partial w_1}$

$w_1 := w_1 - \alpha \frac{\partial J}{\partial w_1}$ #

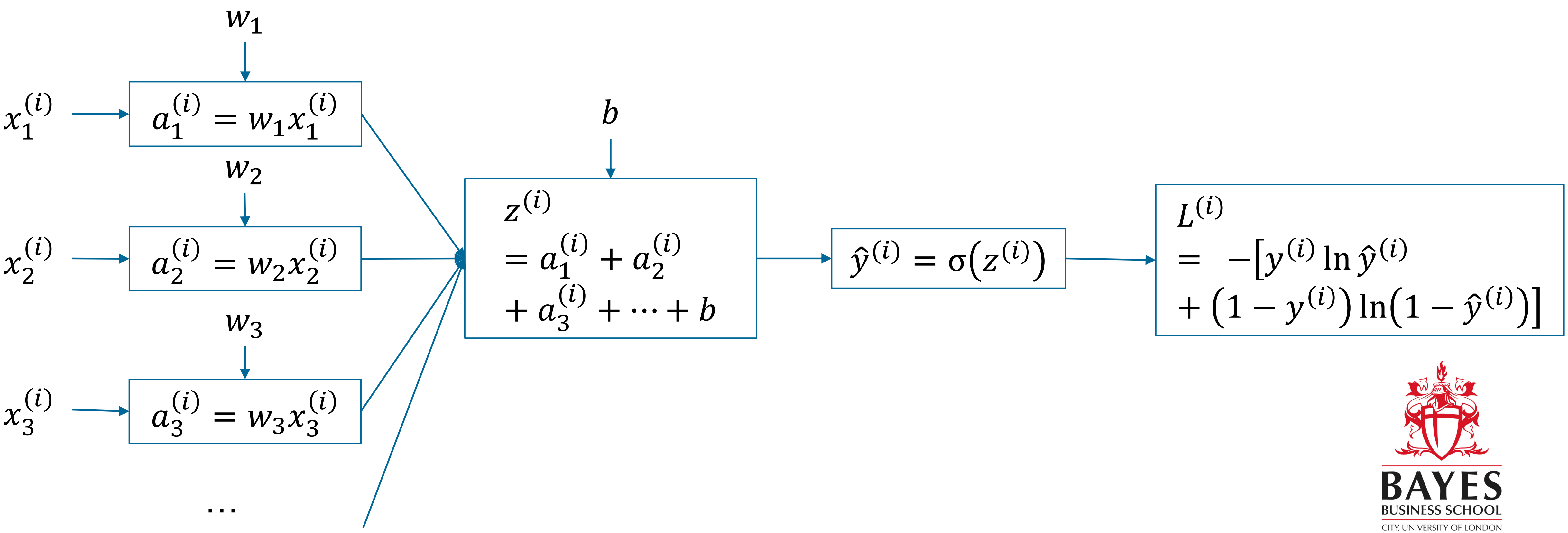
Our first optimization algorithm

1. Decide a "learning rate" α
2. Start with some $\boldsymbol{w}$ and b and compute $J(\boldsymbol{w}, b)$
3. Until J "doesn't change" anymore:
   - Let $w_1 := w_1 - \alpha \frac{\partial J(\boldsymbol{w},b)}{\partial w_1}$
   - Let $w_2 := w_2 - \alpha \frac{\partial J(\boldsymbol{w},b)}{\partial w_2}$
   - …
   - Let $w_m := w_m - \alpha \frac{\partial J(\boldsymbol{w},b)}{\partial w_m}$
   - Let $b := b - \alpha \frac{\partial J(\boldsymbol{w},b)}{\partial b}$
   - Recompute $J(\boldsymbol{w}, b)$
4. Enjoy the fruits of your labor: you have fit a logistic regression model manually!

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Wait a second, how do we find all those derivatives?

- We can use again the computation graph!
- Recall that $\hat{y}^{(i)} = \frac{1}{1+e^{-\left(\boldsymbol{x}^{(i)}\boldsymbol{w}+\boldsymbol{b}\right)}} = \sigma\left(\boldsymbol{x}^{(i)}\boldsymbol{w} + b\right)$



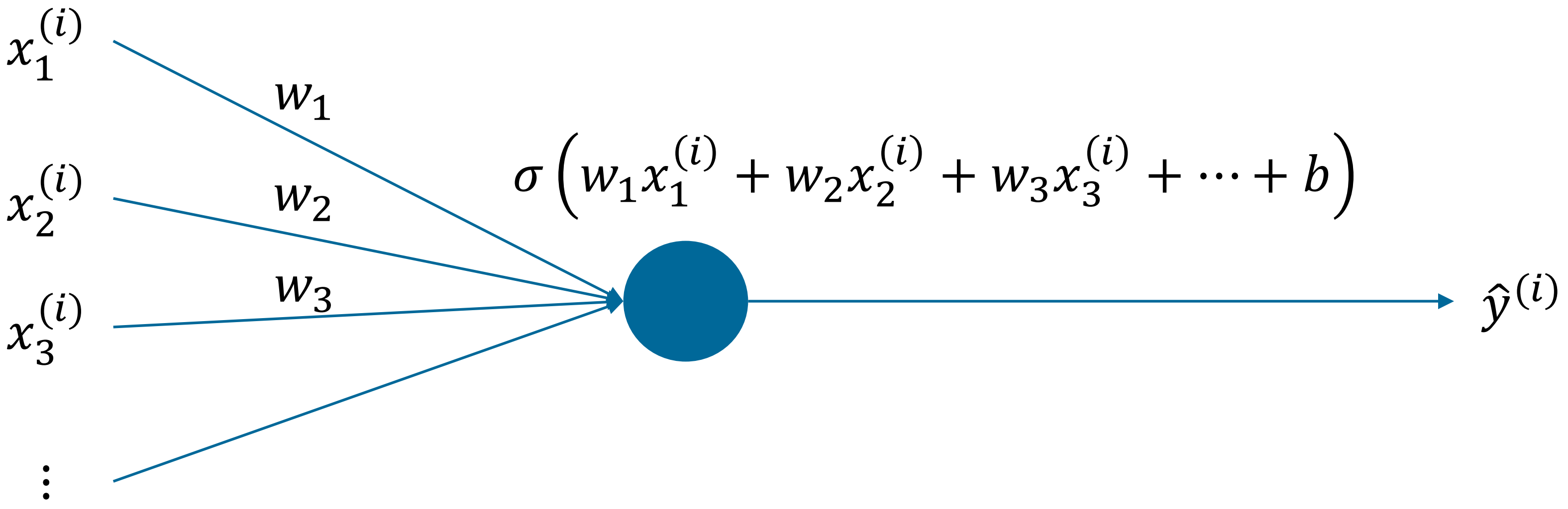
As the same parameters influence all examples, we have to consider one final step

- Recall that $J(\boldsymbol{w}, b) = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \ln\left(1 - \hat{y}^{(i)}\right)\right] = \frac{1}{n}\sum_{i=1}^{n} L^{(i)}$
- We have that $\frac{\partial J(\boldsymbol{w},b)}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial L^{(i)}}{\partial w_j}$

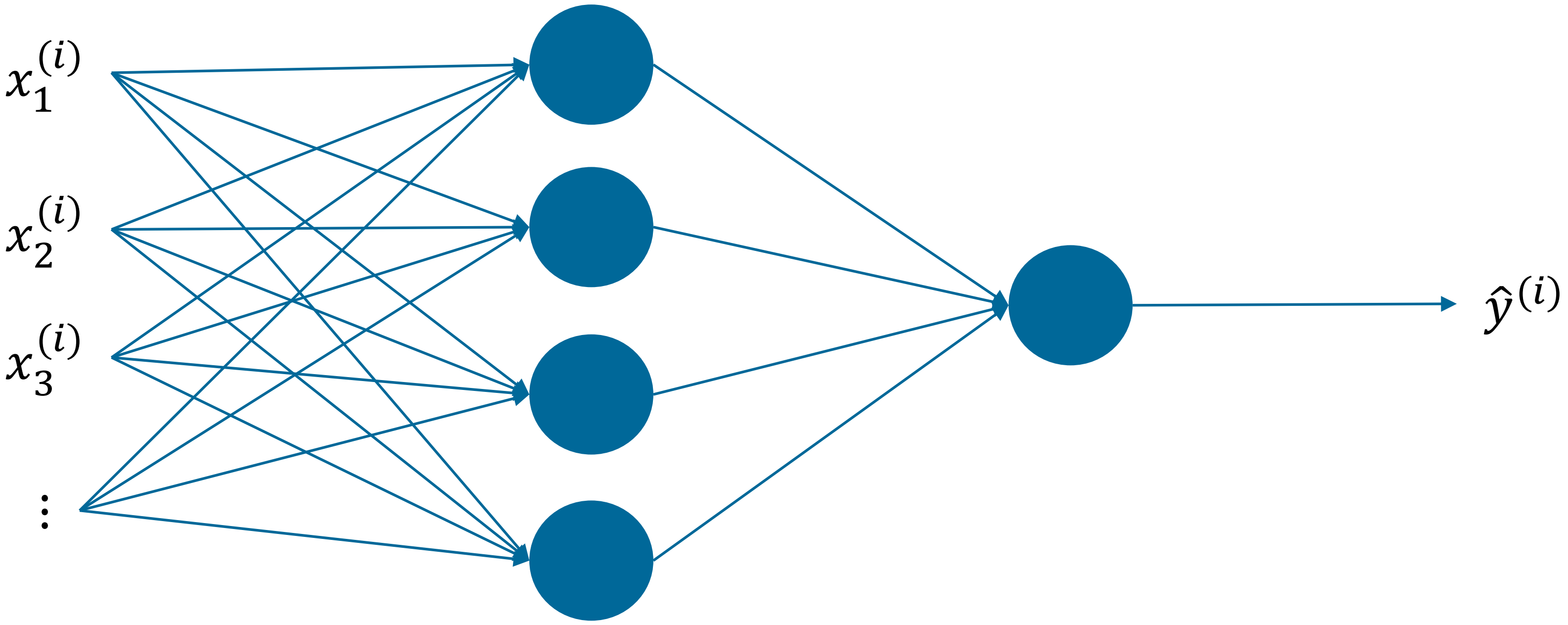
We can now implement a logistic regression

Visualizing a logistic regression – our first neural network

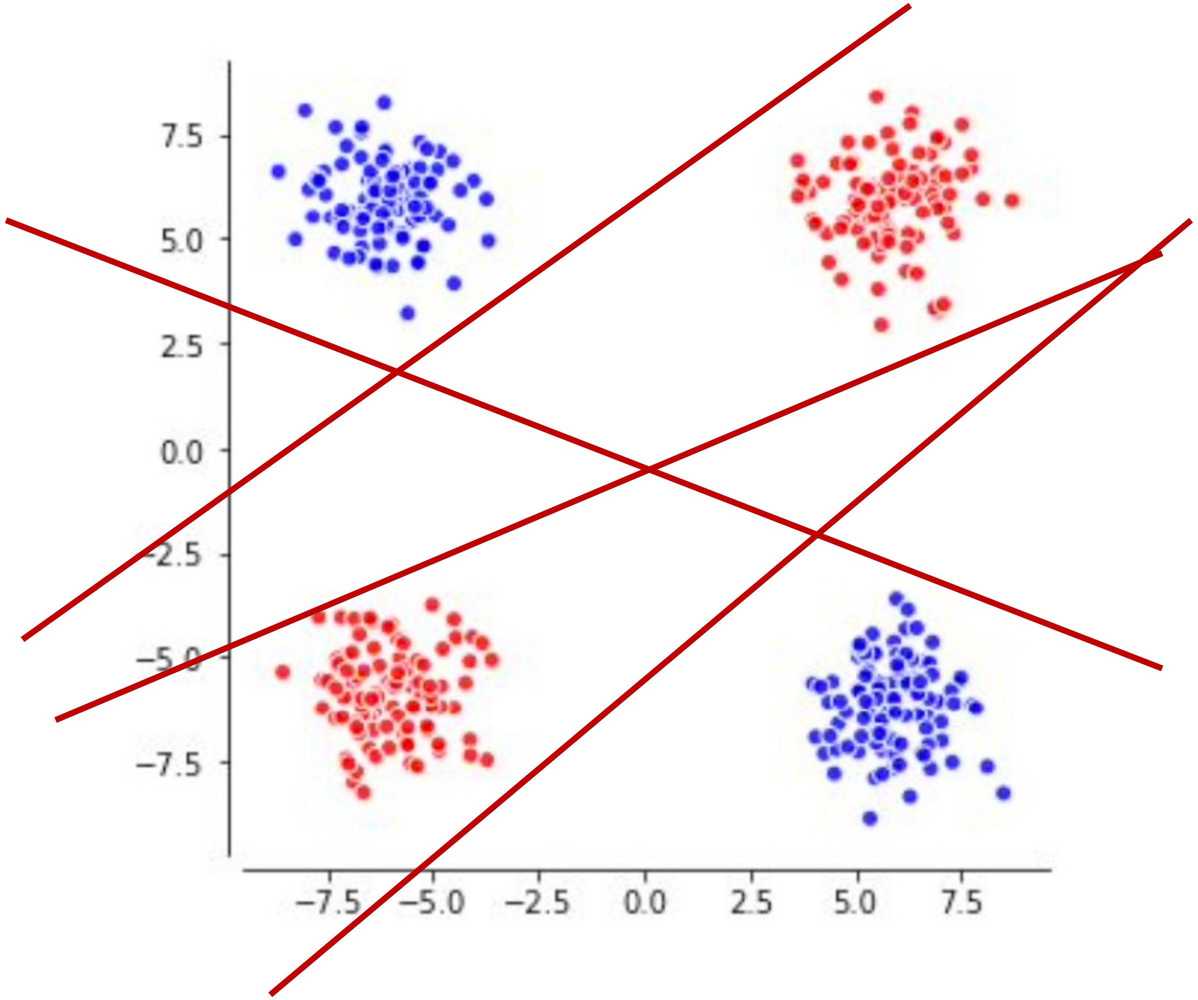
Schema of a logistic regression



Putting multiple neurons together



What neurons learn



Source: Czarnecki

Training neural networks visually

Open https://playground.tensorflow.org/

1. A simple case of binary classification:
   - Change to the pattern on the lower left
   - Set the level of “Noise” to 50
   - Set “Ratio of training to test data” to 50%
   - Set up the neural network: 1 hidden layer, 1 neuron, then press play
   - Answer the following questions:
     - Did the training eventually find a model that seems to capture the pattern in the data?
     - How would you describe the pattern the model captured?
     - Record the “Training loss” and “Test loss”
   - How do your answers change when you select the pattern at the top right? What about setting the noise to 0?

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

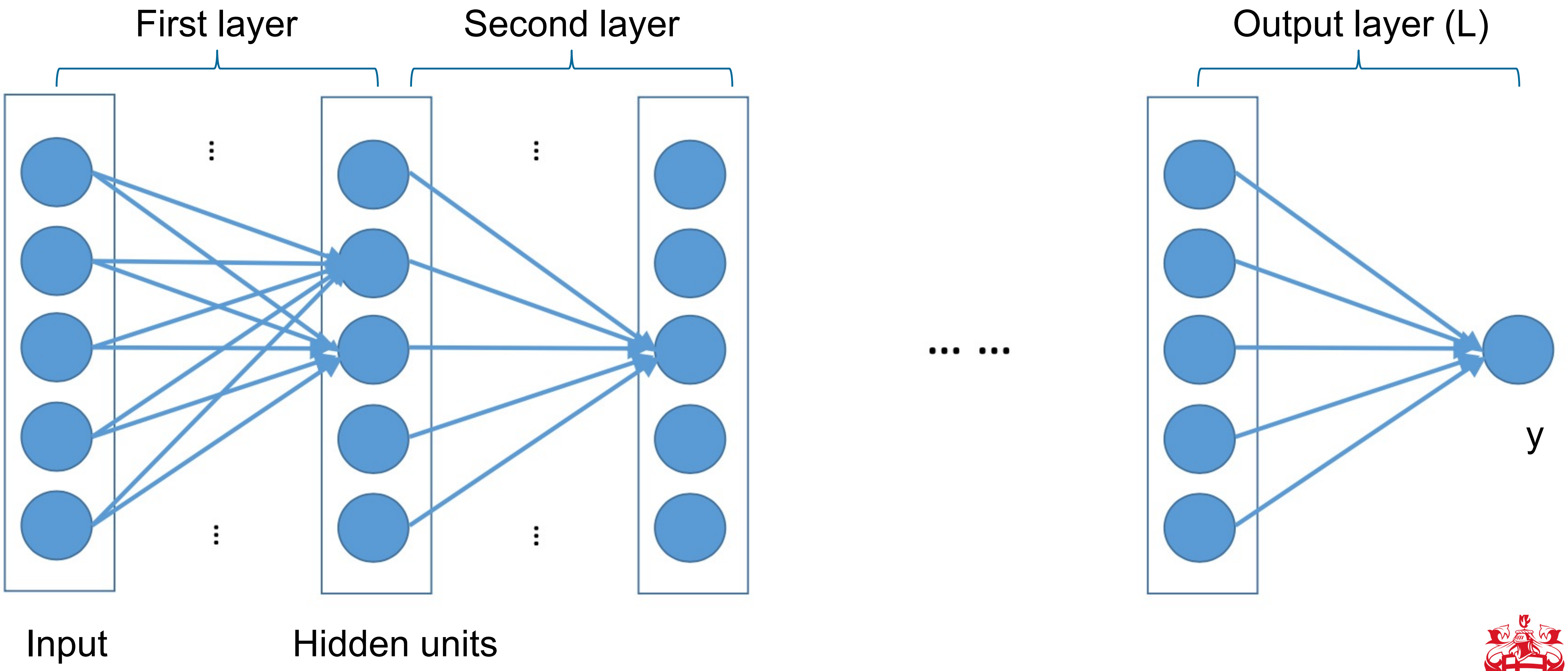
Training neural networks visually

2. A shallow neural network:
- Stick with the pattern at the top right, a noise of 0 and a ratio of 50%
- Now use 3 neurons for your hidden layer
- Answer the three questions from before:
  - Did the training eventually find a model that seems to capture the pattern in the data?
  - How would you describe the pattern the model captured?
  - Record the “Training loss” and “Test loss”
- How do your answers change when you use 6 neurons instead?

3. A deep neural network:
- Use a second hidden layer, with 3 neurons each (and the other setups from 2.)
- How do your answers change now?

Key components of a neural network

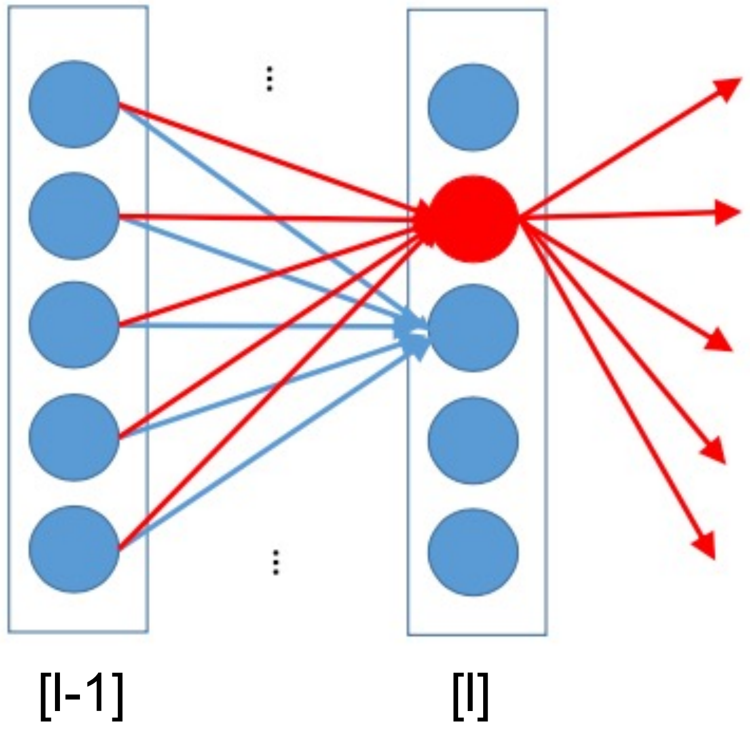
Components



Source: Liang

Hidden layers

$$\text{“}\boldsymbol{x}\text{”} = \boldsymbol{a}^{[l-1]} \qquad z = \boldsymbol{a}^{[\boldsymbol{l}-\mathbf{1}]}\boldsymbol{w} + \boldsymbol{b} \qquad f(\mathrm{z})$$



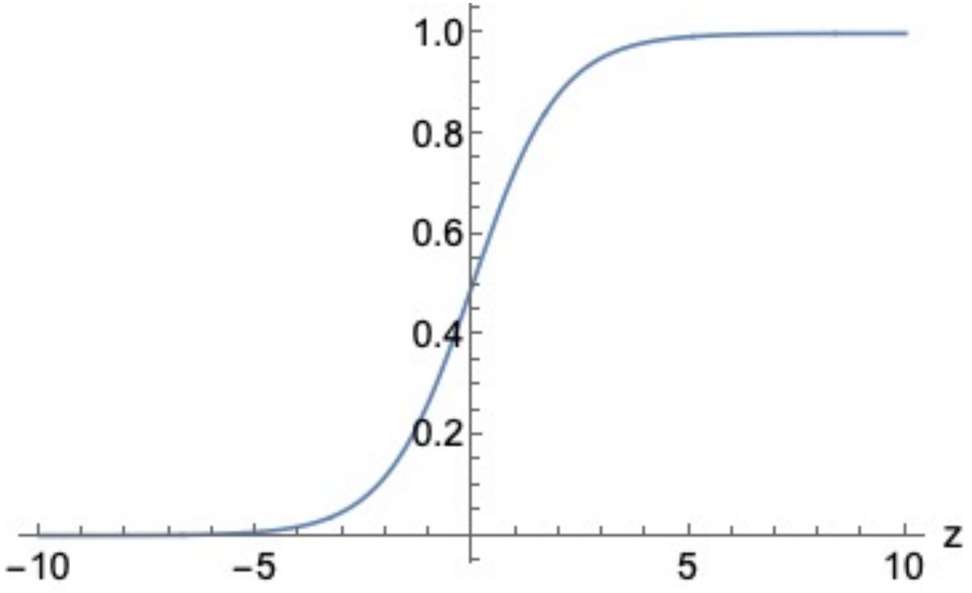
- f is what we call an “activation function”
- There are many activation functions, and new ones are invented all the time
- Many of these functions do just fine, or slightly better than existing ones

Source: Liang

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

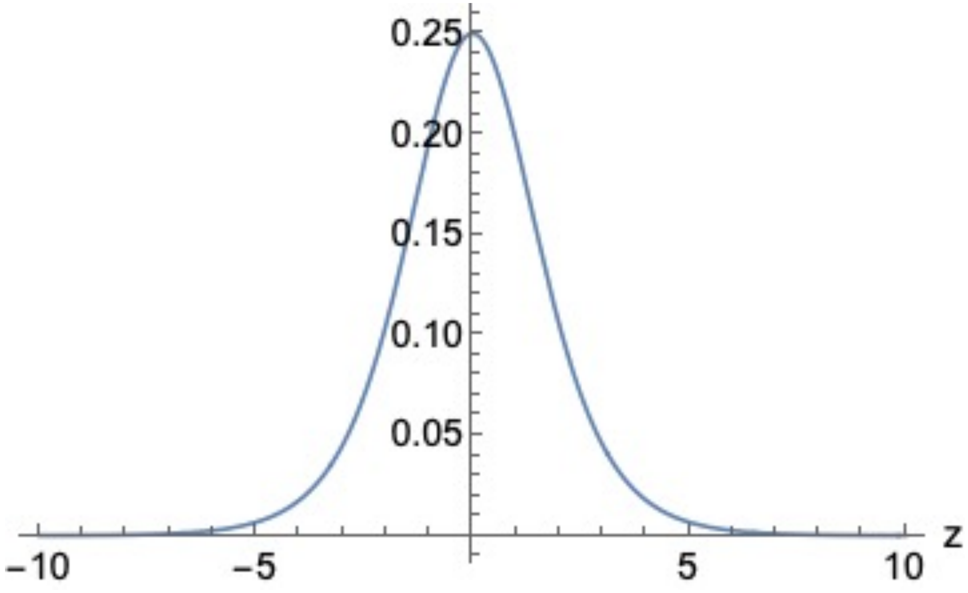
Typical activation functions: logistic (sigmoid) function

Logistic (sigmoid) function



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

Derivative

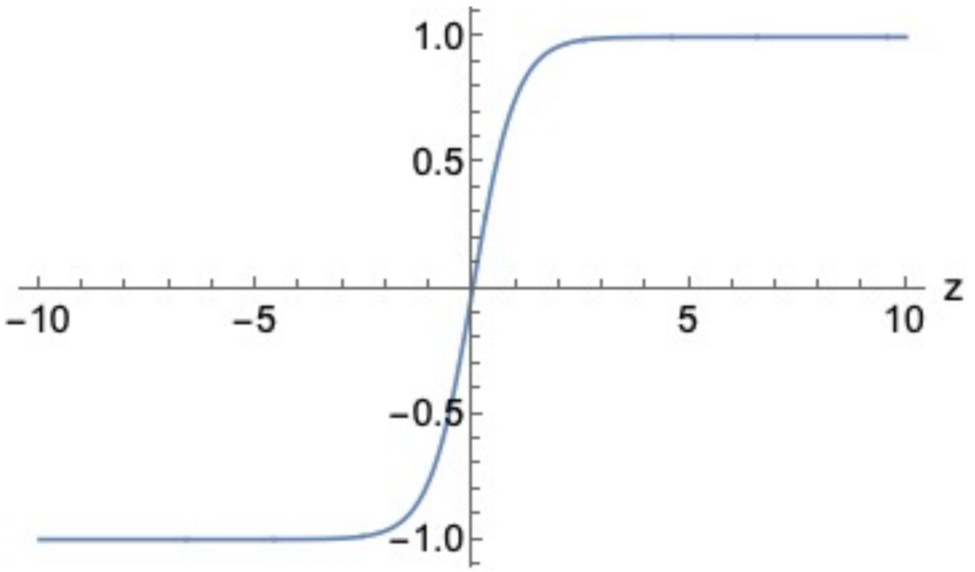


$$f'(z) = \sigma(z)\big(1 - \sigma(z)\big)$$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

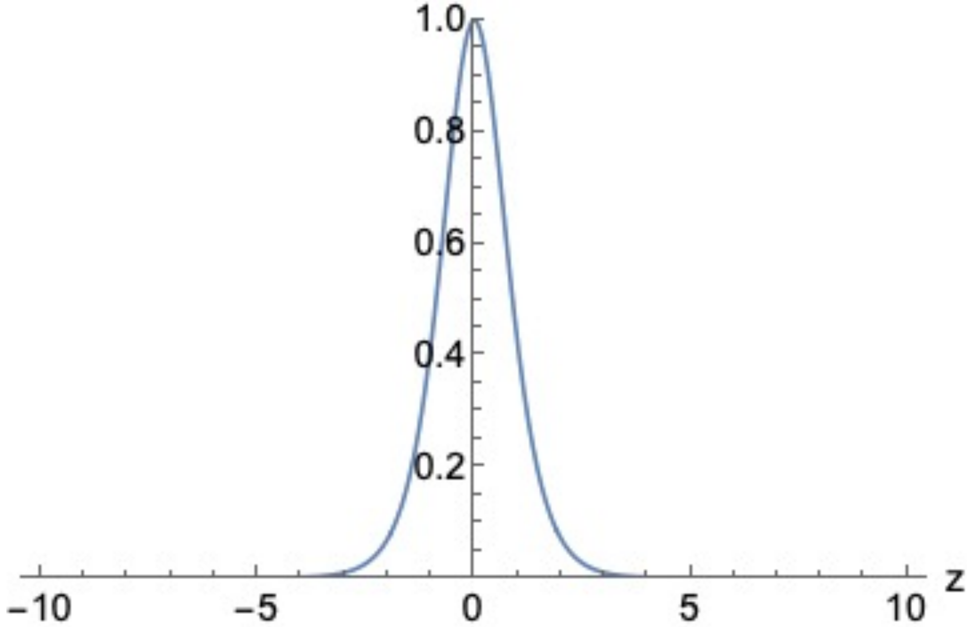
Typical activation functions: hyperbolic tangent

Hyperbolic tangent



$$f(z) = \tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

Derivative

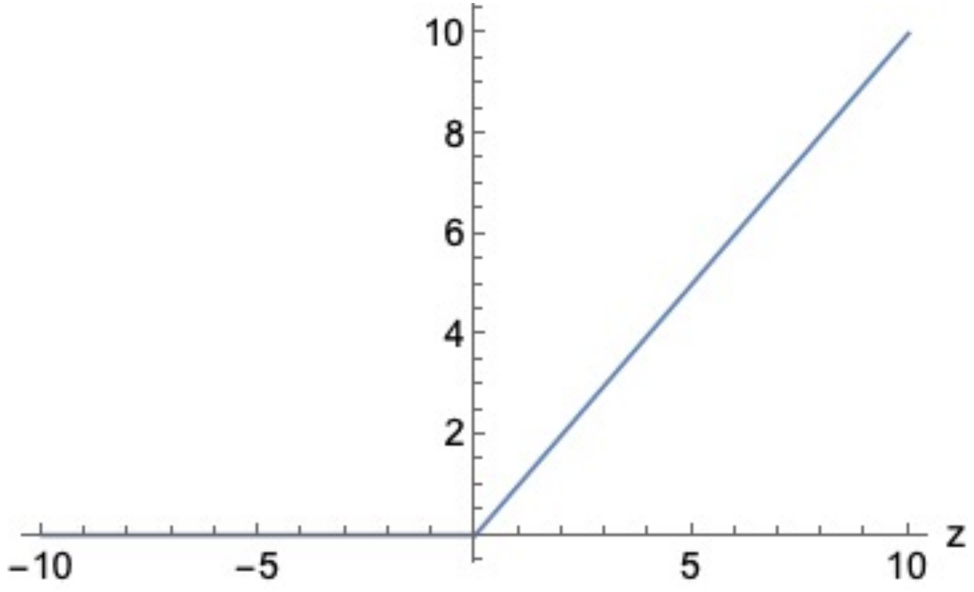


$$f'(z) = \operatorname{sech}(z)^2$$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

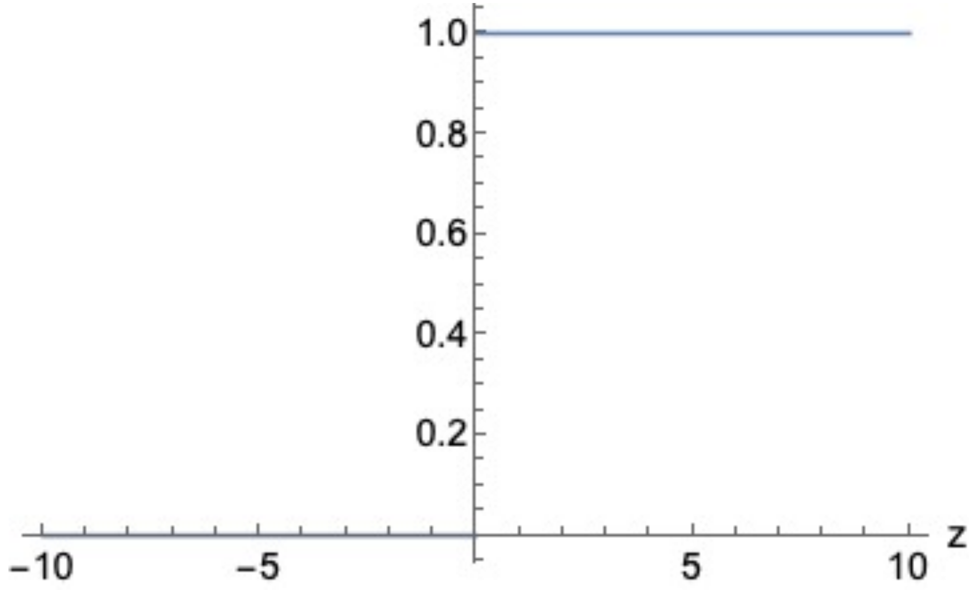
Typical activation functions: Rectified Linear Unit

Rectified Linear Unit (ReLU)



$$f(z) = \max\{0, z\}$$

“Derivative”

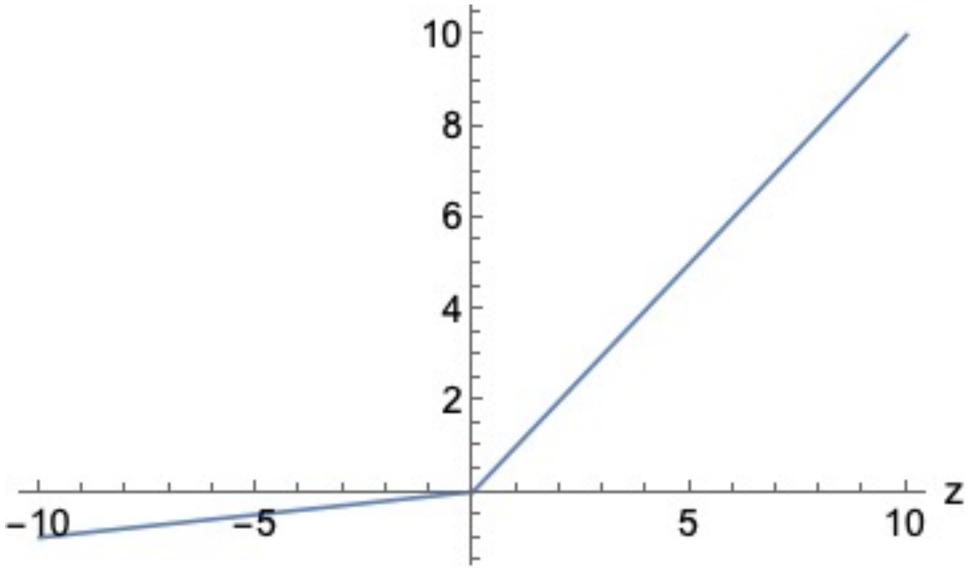


$$f'(z) = \begin{cases} 0, if\ z < 0 \\ 1, if\ z > 0 \end{cases}$$

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

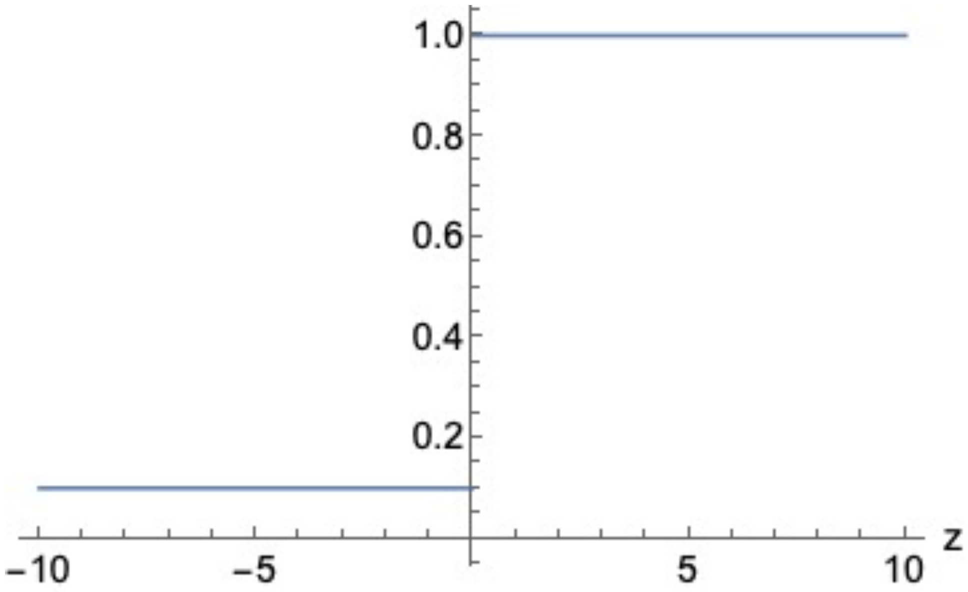
Typical activation functions: Leaky ReLU

Leaky ReLU



$$f(z) = \max\{0.1, z\}$$

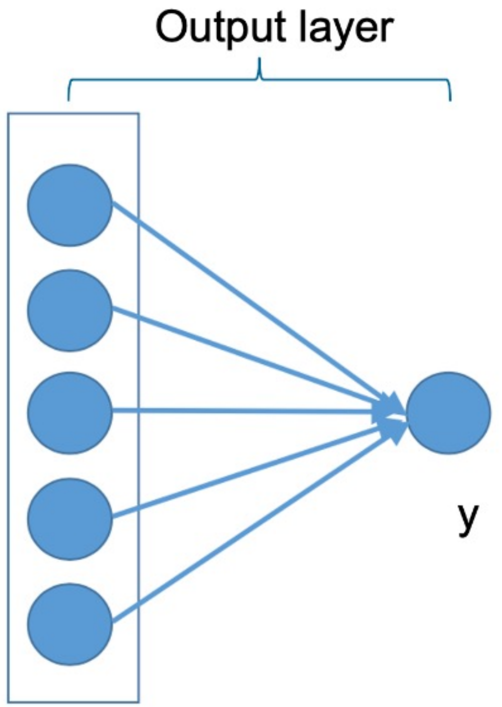
"Derivative"



$$f'(z) = \begin{cases} 0.1, if\ z < 0 \\ 1, if\ z > 0 \end{cases}$$

Binary classification

- Input: $\boldsymbol{a}^{[L-1](i)}$
- As usual, we make a linear transformation: $z^{[L](i)} = \boldsymbol{a}^{[L-1](i)}\boldsymbol{w}^{[L]} + b^{[L]}$
- We then use the logistic sigmoid function $\hat{y}^{(i)} = f\left(z^{[L](i)}\right) = \sigma(z^{[L](i)}) = \frac{1}{1+e^{-z^{[L](i)}}}$
- We can interpret the output as the probability of $y^{(i)} = 1$



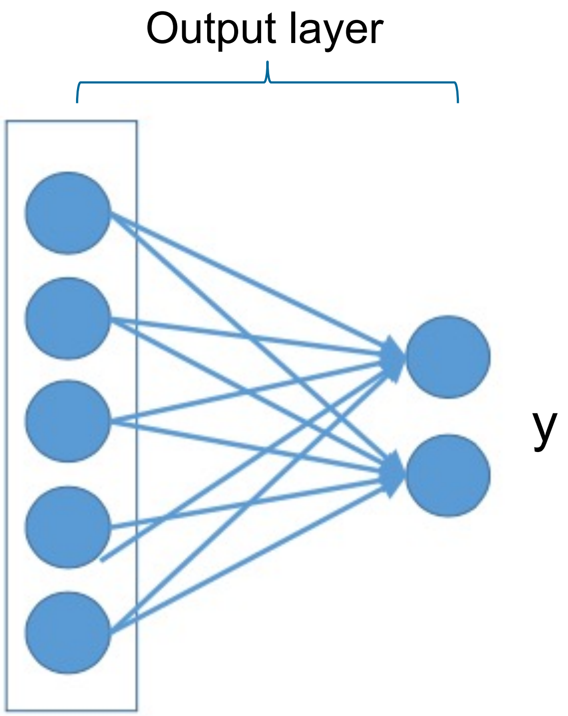
Source: Liang

Multi-class classification

- We again make a linear transformation with a matrix of weights: $\boldsymbol{z}^{[L](i)} = \boldsymbol{a}^{[L-1](i)}\boldsymbol{W}^{[L]} + \boldsymbol{b}^{[L]}$
- Note that $\boldsymbol{z}^{[L](i)} = \begin{pmatrix} z_1^{[L](i)} & z_2^{[L](i)} & \cdots & z_K^{[L](i)} \end{pmatrix}$
- We then use the softmax function on each of the outputs:

$$\hat{y}_k^{(i)} = f\left(\boldsymbol{z}^{[L](i)}\right) = \frac{e^{-z_k^{[L](i)}}}{\sum_{k=1}^{K} e^{-z_k^{[L](i)}}}$$

- This implies that $\hat{y}_k^{(i)} \in (0,1)$ and $\sum_{k=1}^{K} \hat{y}_k^{(i)} = 1$
- Hence, we can interpret $\hat{y}_k^{(i)}$ as the probability that $y^{(i)} = k$ ("belongs to class k")



Source: Liang

Cost functions

- Recall from logistic regression:

$$J(\boldsymbol{w}, b) = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} \ln \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \ln\left(1 - \hat{y}^{(i)}\right)\right]$$

- Generally, to learn parameters $\boldsymbol{\theta}$, we define the cross-entropy

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} = -\frac{1}{n}\sum_{i=1}^{n} \ln p_{\boldsymbol{\theta}}\left(y^{(i)} \middle| \boldsymbol{x}^{(i)}\right)$$

- Also known as "maximum likelihood estimator"

- Mean square error tends to perform poorly, especially when we have activation functions with e^z

Generalizing our optimization algorithm

1. Decide a "learning rate" α
2. Start with some parameters $\boldsymbol{\theta}$ and compute $J(\boldsymbol{\theta})$ (forward propagation)
3. Until J "doesn't change" anymore:
   - Let $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (back-propagation)
   - Recompute $J(\boldsymbol{\theta})$ (forward propagation)

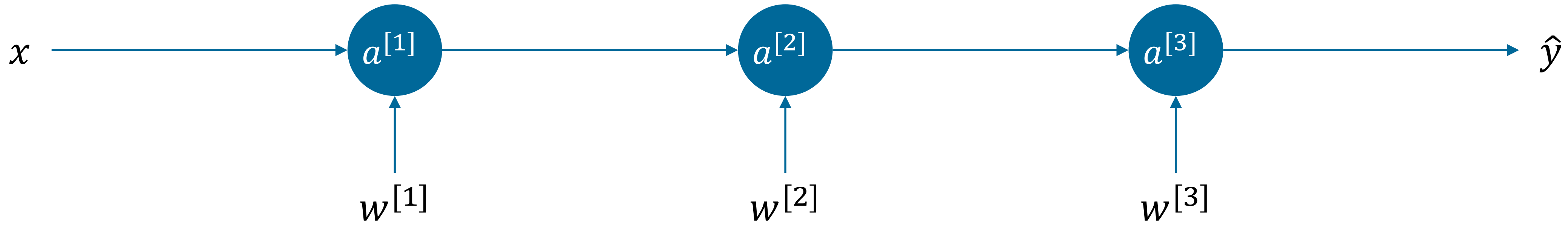
Initializing parameters

- Initialize weights to small random values (e.g., *np.random.randn(**shape of W**) * 0.01*)

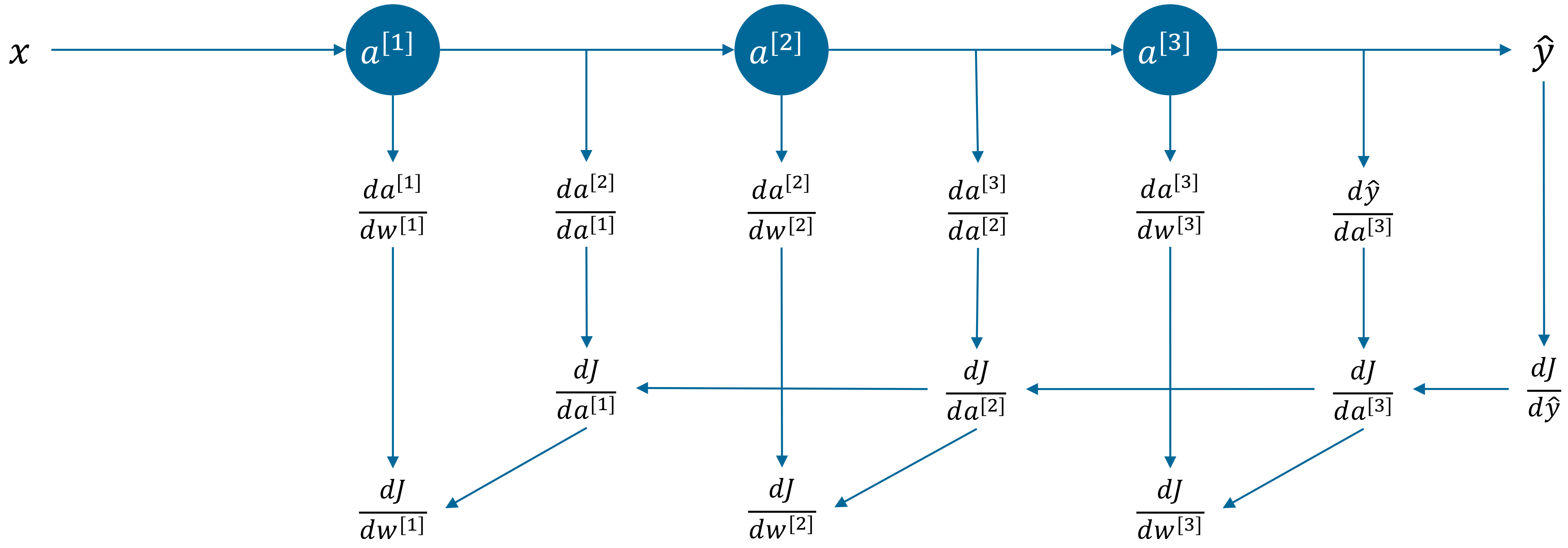
- Bias terms can be initialized randomly, but can also just be initialized to zero (e.g., *np.zeros(**shape of b**)*)

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Step 1: Forward propagation through the computational graph



Step 2: Back-propagation through the computational graph



See you next week!