Deep Learning in Computer Vision

June 6th, 2019

Janus Nørtoft Jensen, jnje@dtu.dk

Who are we?

ASM Shihavuddin (Shihav)
Assistant Professor at DTU Compute
Section for Image Analysis and Computer Graphics
shihav@dtu.dk

Morten Hannemose
PhD student at DTU Compute
Section for Image Analysis and Computer Graphics
mohan@dtu.dk

Janus Nørtoft Jensen
PhD student at DTU Compute
Section for Image Analysis and Computer Graphics
jnje@dtu.dk

Programme

- The course consist of lectures + exercises + work on assignments
- The lectures will cover the basics but are limited to give you as much time for hands-on experience
- Since this is a brand new course some things might work well on some things might not
  - We will try to adjust the content accordingly

Mandatory hand-ins

- 3 mandatory pair/group (3 persons max) hand-ins during course
- Be concise! Limit the number of pages (max 5 pages)
  - Classification/Detection
    - Deadline 14/6 23.59
  - Segmentation
    - Deadline 20/6 23.59
  - Generative Adversarial Networks
    - Deadline 26/6 23.59

Exam

- Oral exam will be on the 27/6 and 28/6
- You will draw one of the 3 assignments
  - You will start by presenting your solutions and results
  - After that, we will start asking questions

Deep Learning in *Computer Vision*

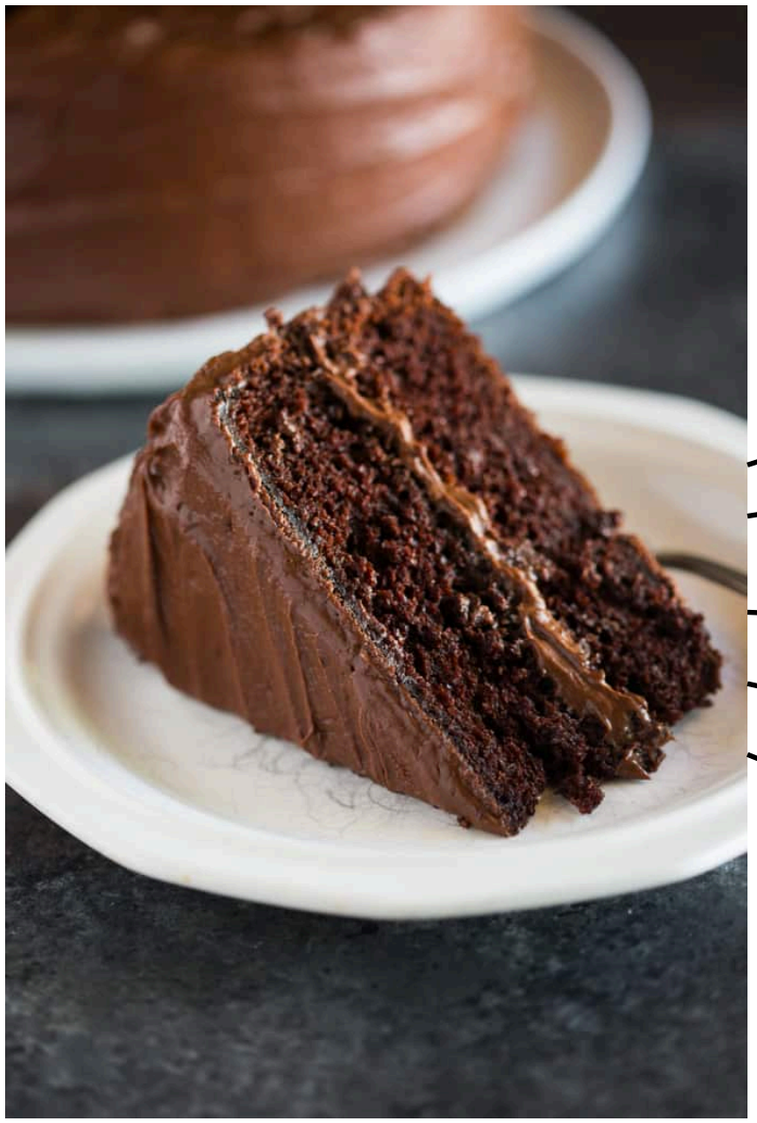
What is Computer Vision

- The goal in computer vision is to extract useful information from images
- This includes, but is not limited to:
  - Reconstructing properties such as shape, illumination and color distributions
  - Detecting objects - e.g. faces in images
  - Estimating motion in image sequences
  - Detecting and matching points of special interest between images - e.g. for creating panorama images
  - …

What is Computer Vision

- The goal in computer vision is to extract useful information from images
- This includes, but is not limited to:
  - **Reconstructing properties such as shape, illumination and color distributions**
- **Detecting objects - e.g. faces in images**
- Estimating motion in image sequences
- Detecting and matching points/features of special interest between images - e.g. for creating panorama images
- …

Image classification



Dog

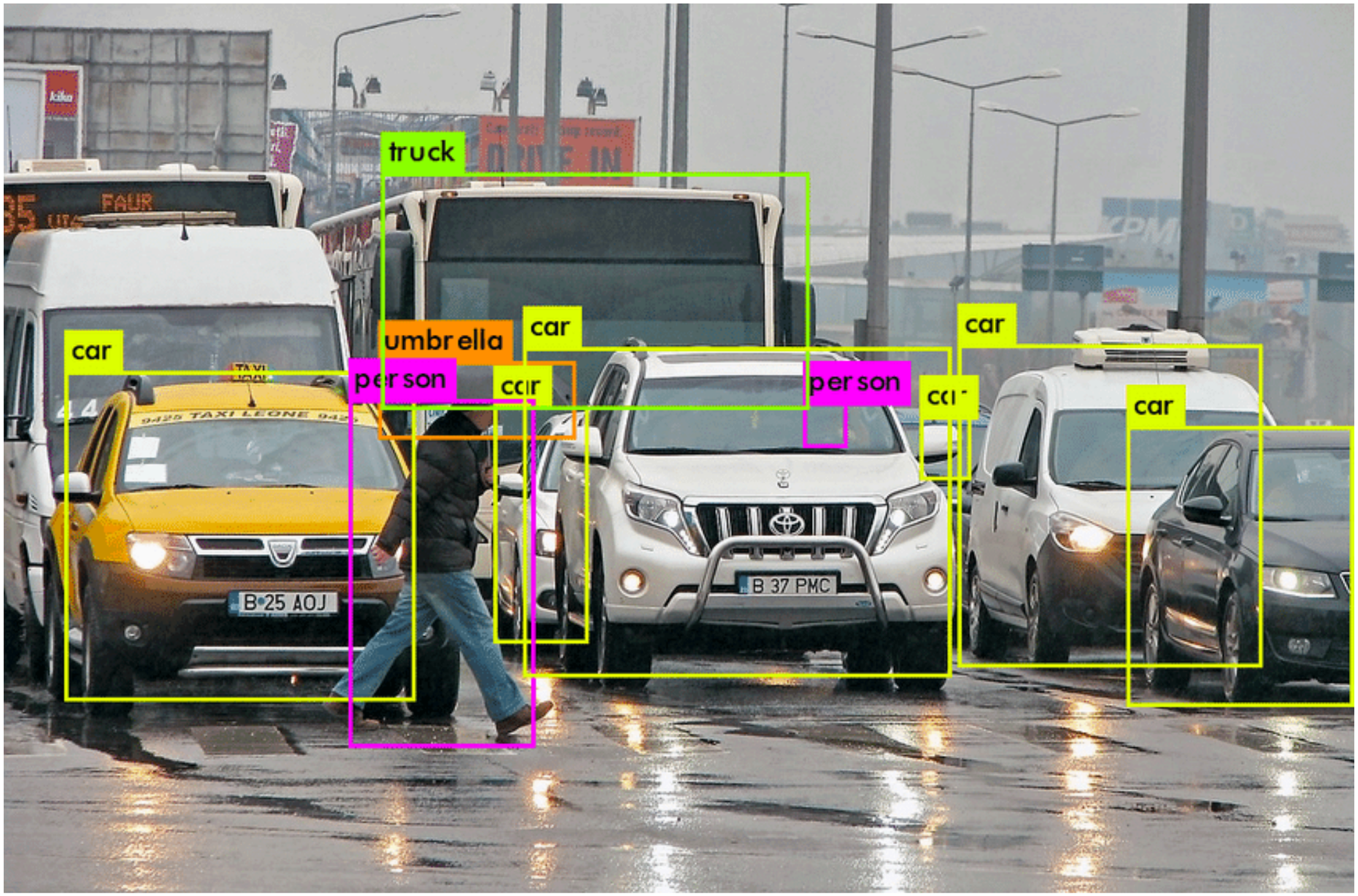
Piano

Cake

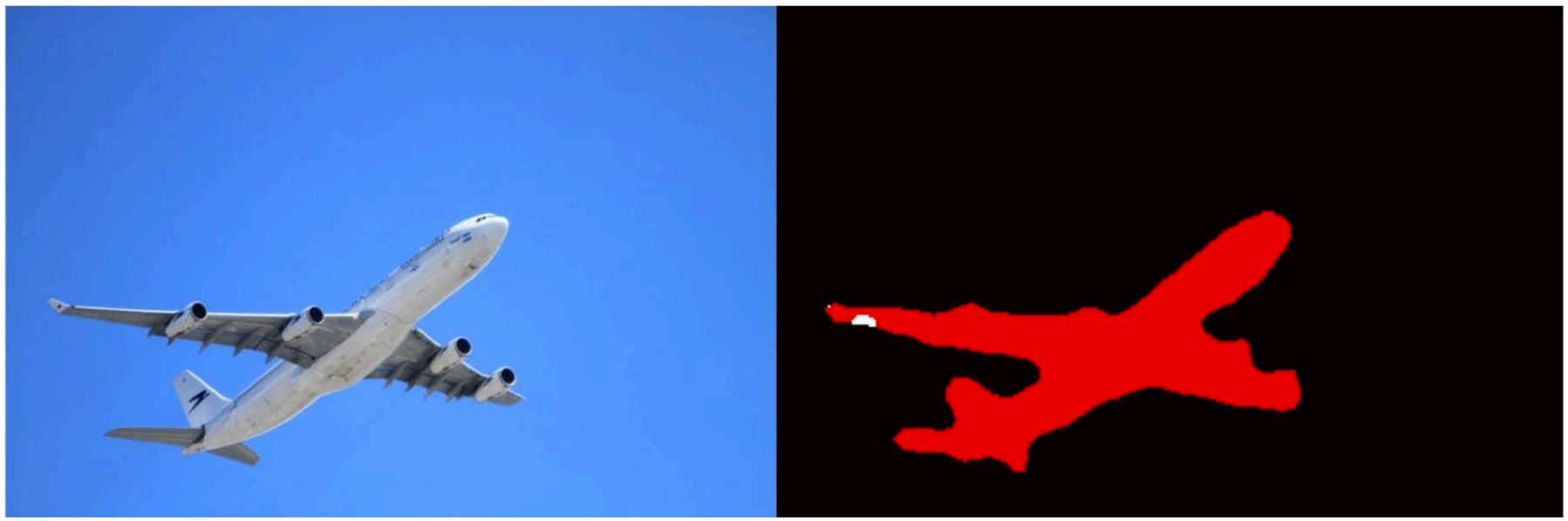
Computer

Unknown

Object Detection



Segmentation



Generative Models

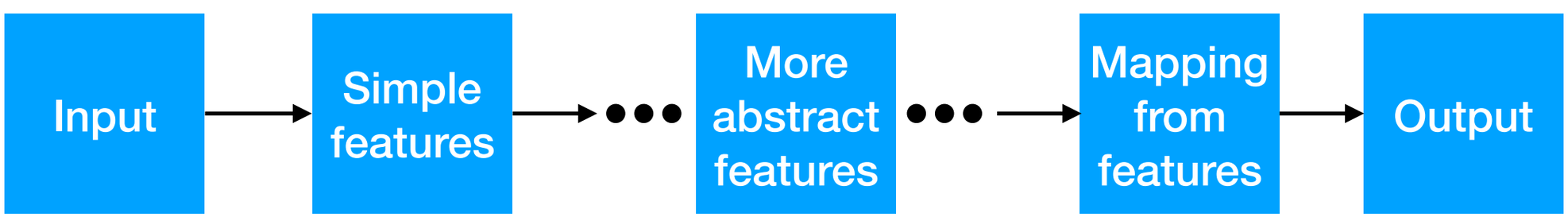
Deep Learning in Computer Vision

Learning

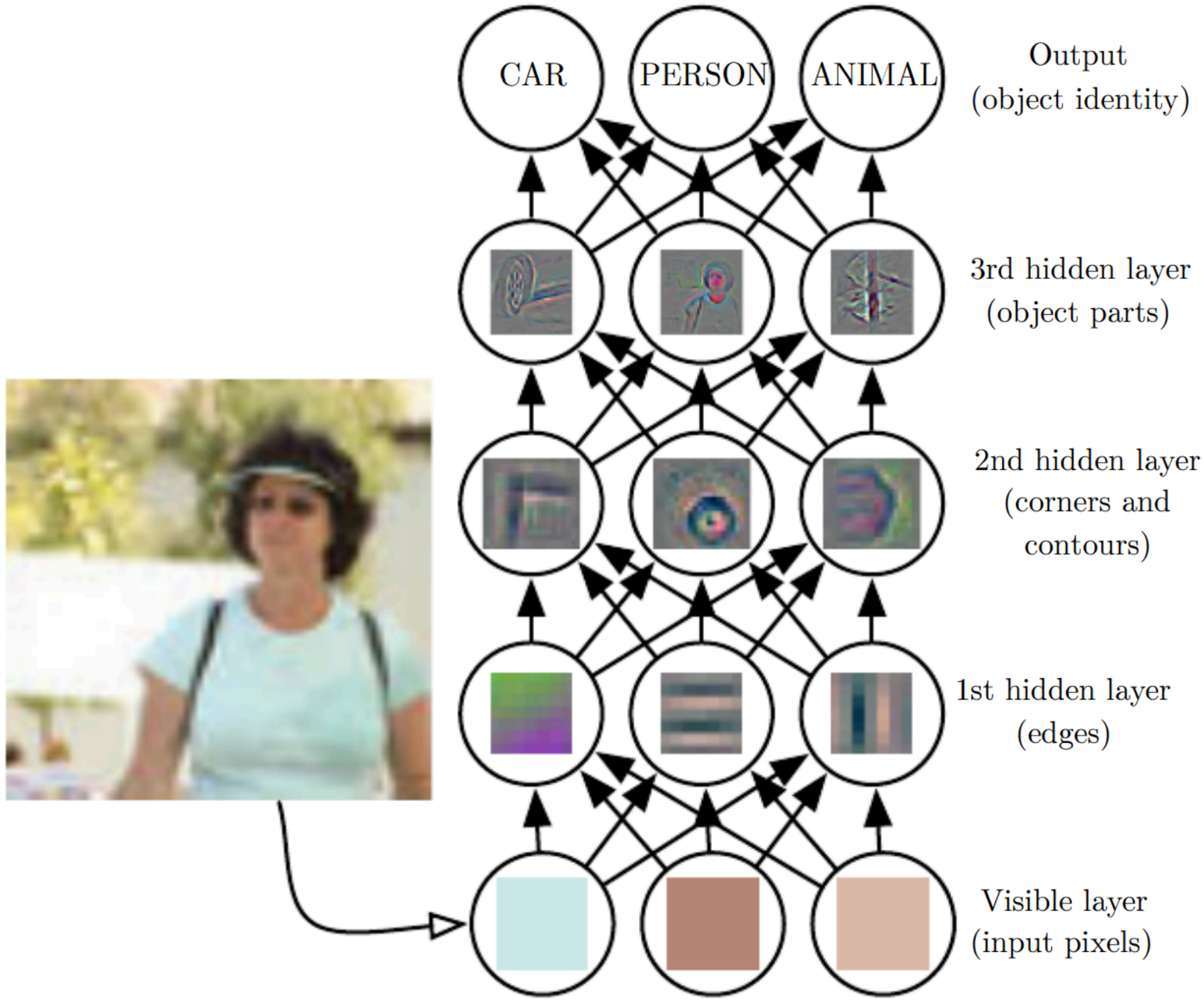
- Supervised learning
  - Learn mapping from input to output given training examples with known output
- Unsupervised learning
  - Learn “something” about the distribution of some input examples without having known output
- Reinforcement learning
  - Learn actions based on rewards

Deep Learning

- In Deep Learning we seek to map a set of input values to output values
- Going directly from input to output is in most cases not possible
- Instead we learn representations of the inputs from which it is easier to predict the output
- In Deep Learning we learn increasingly complex representations/features that are expressed in terms of simpler representations/features



Deep Learning



Deep Learning, Goodfellow et al.

Neural networks

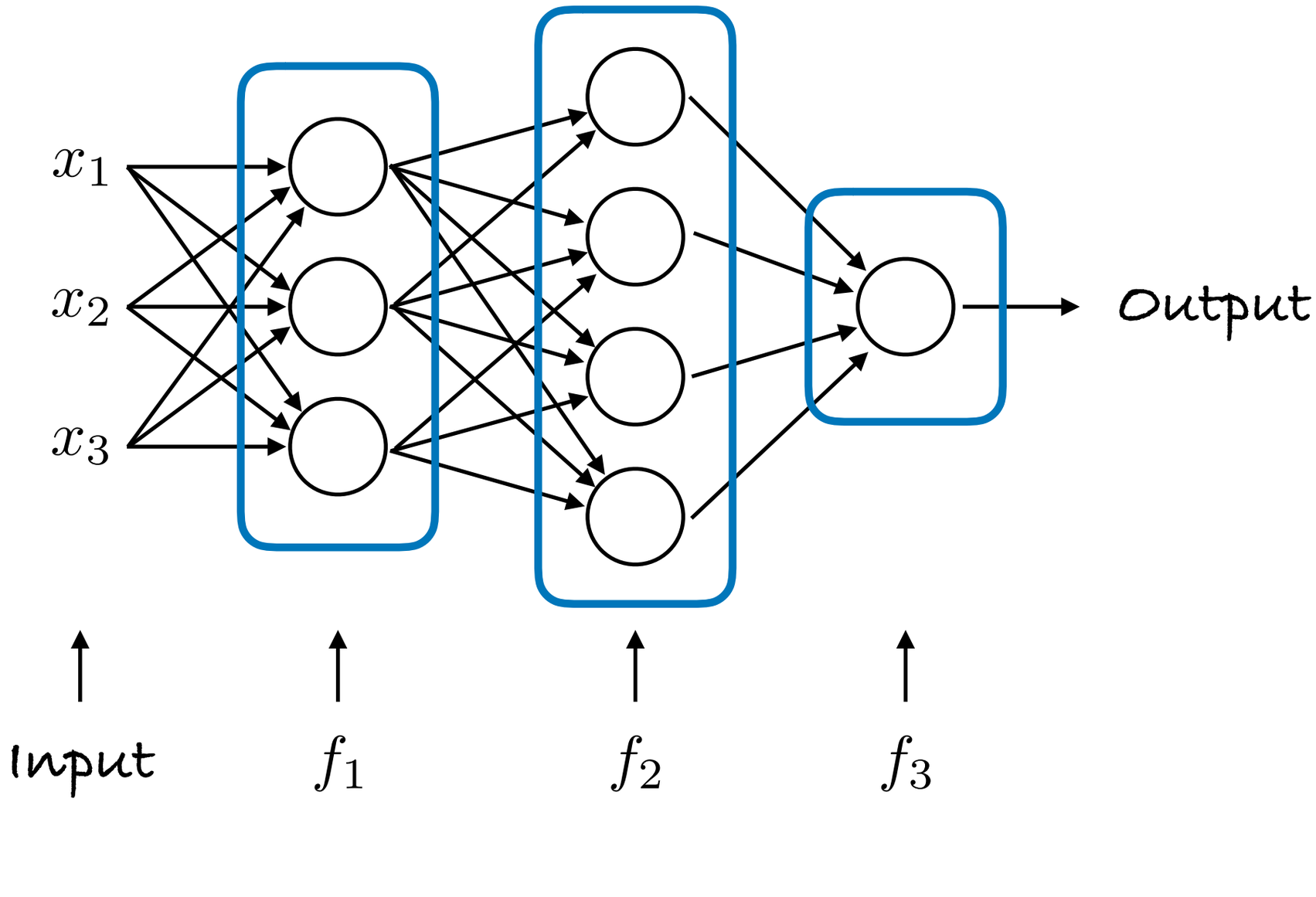
- The previous example can be written as:

$$f(x) = f_4(f_3(f_2(f_1(x))))$$

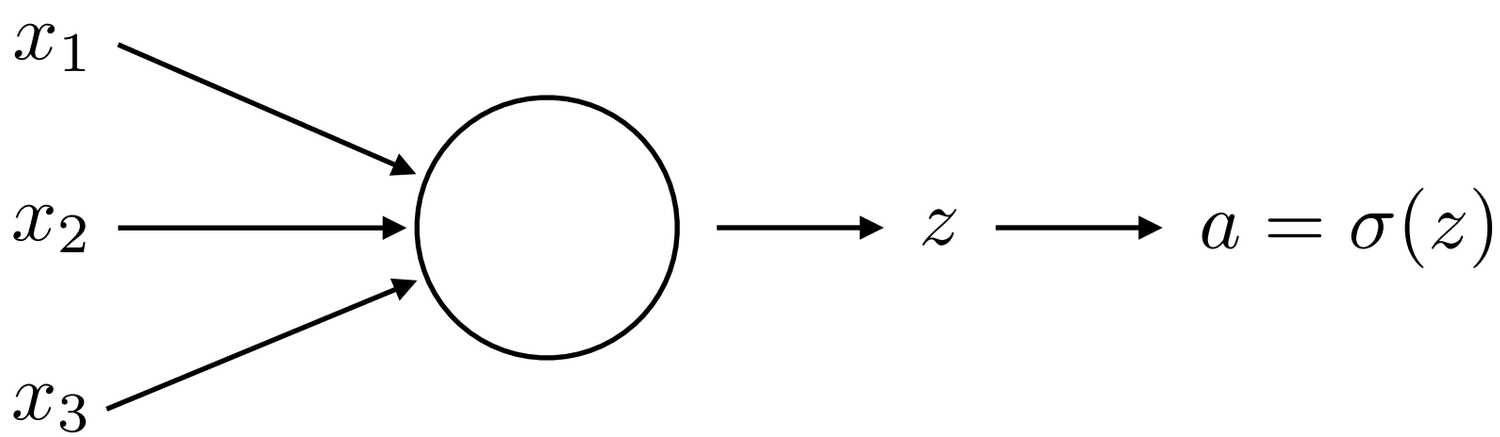
Output ↑ — ↑ Input

- How do we represent the functions f_1, f_2, f_3, f_4 ?

Feed-Forward Neural networks



Neurons - the building block

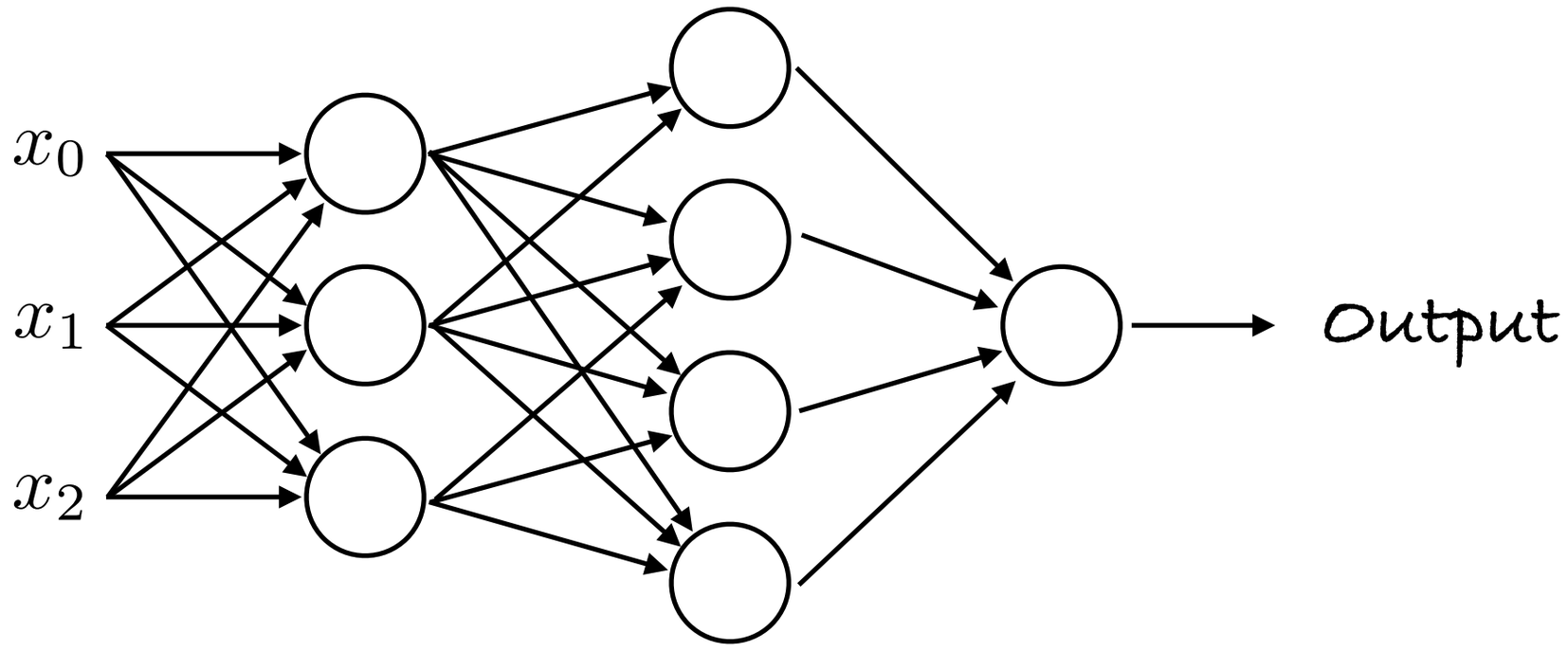


$$z = w_1 x_1 + w_2 x_2 + w_3 x_3 + b = \mathbf{w}\mathbf{x} + b$$

$a = \sigma(z)$ is the output of the neuron

$\sigma(\cdot)$ is a non-linear activation function

Feed-forward computation



$$\mathbf{a}^0 = \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

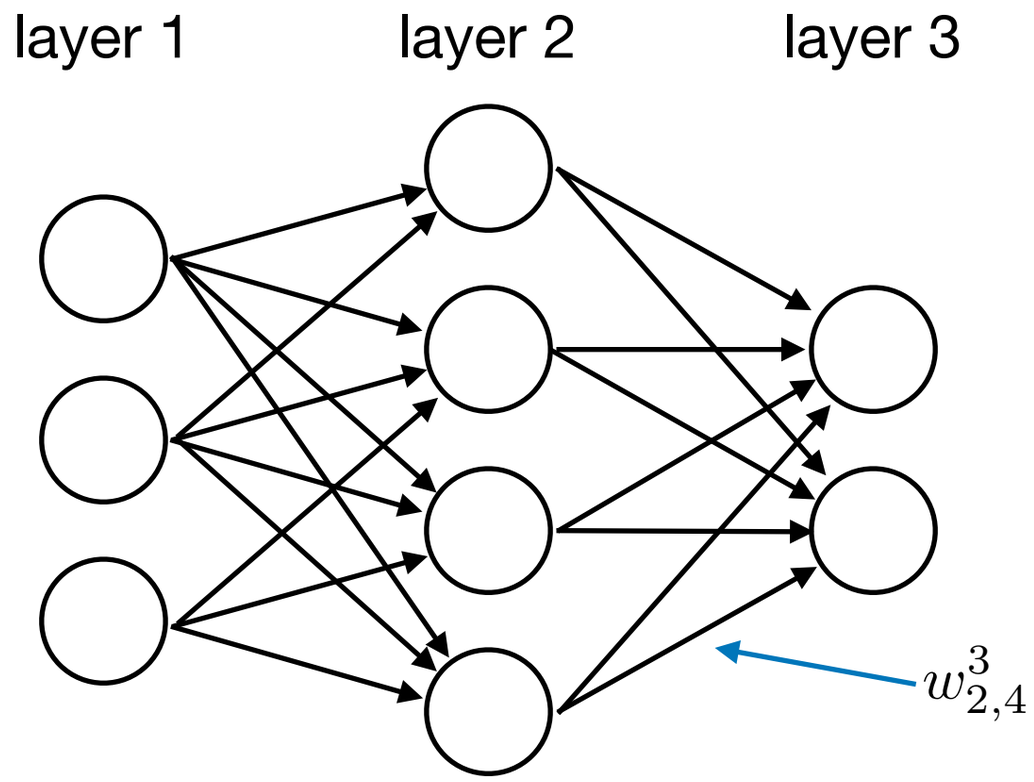
$$\mathbf{a}^1 = \sigma(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1)$$

$$= \left(\begin{bmatrix} w^1_{0,0} & w^1_{0,1} & w^1_{0,2} \\ w^1_{1,0} & w^1_{1,1} & w^1_{1,2} \\ w^1_{2,0} & w^1_{2,1} & w^1_{2,2} \end{bmatrix} \begin{bmatrix} a^0_0 \\ a^0_2 \\ a^0_2 \end{bmatrix} + \begin{bmatrix} b^1_0 \\ b^1_1 \\ b^1_2 \end{bmatrix} \right)$$

$$\mathbf{a}^2 = \sigma(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2)$$

$$\dots$$

Feed-forward computation



$$\mathbf{a}^\ell = \sigma(\mathbf{W}^\ell \mathbf{a}^{\ell-1} + \mathbf{b}^\ell)$$

$$= \sigma\left(\sum_k w^\ell_{j,k} a^{\ell-1}_k + b^\ell_j\right)$$

$w^\ell_{j,k}$ is the weight from the *k*th neuron in the *(l-1)th* layer to the *j*th neuron in the *l*th layer

b^ℓ_j is the bias of the *j*th neuron in the *l*th layer

a^ℓ_j is the activation of the *j*th neuron in the *l*th layer

How do we learn the parameters and biases?

- We need a loss function that measures how well our network is doing with a given set of parameters *W* and *b?*
- How can we use this to learn the parameters?

How do we learn the parameters and biases?

- Random search?
  - Try many different random weights and biases and choose the best one
- Random local search?
  - Generate random weights and biases close to our current and choose the best one
- Gradient Descent?
  - Use the gradient of our loss function, which gives the direction of maximum descent at any point x

How do we learn the parameters and biases?

- Random search?
  - Try many different random weights and biases and choose the best one
- Random local search?
  - Generate random weights and biases close to our current and choose the best one
- **Gradient Descent**
  - **Use the gradient of our loss function, which gives the direction of maximum descent at any point x**

Gradient descent

- If *L* is the loss function we wish to minimise with respect to parameters *p*
  - The gradient $\nabla_p \mathcal{L}$ gives us the direction of maximum ascent of the loss function
  - $-\nabla_p \mathcal{L}$ is the direction of maximum descent
- Gradient descent algorithm:
  - Compute the gradient
  - Take a step in the negative gradient direction

$$p \to p' = p - \alpha \nabla_p \mathcal{L}$$

  - Here α is called the learning rate and determines the step size
  - Repeat until convergence

Backpropagation

- Which parameters do we want to find the partial derivates of $\mathcal{L}$ with respect to?
  - The weights ***W*** and biases ***b***
- How?
  - We propagate the error back through the network

The equations of backpropagation

- Our goal is to find the partial derivatives of the loss function $\mathcal{L}$ with respect to any weight $w^{\ell}_{j,k}$ or bias b^{ℓ}_{j}
- By the chain rule we have that

$$\frac{\partial \mathcal{L}}{\partial w^{\ell}_{j,k}} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}} \frac{\partial z^{\ell}_{j}}{\partial w^{\ell}_{j,k}} = a^{l-1}_{k} \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}}$$

$$\frac{\partial \mathcal{L}}{\partial b^{\ell}_{j}} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}} \frac{\partial z^{\ell}_{j}}{\partial b^{\ell}_{j}} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}}$$

- We thus need a way to calculate $\frac{\partial \mathcal{L}}{\partial z^{\ell}_{j}}$

The equations of back propagation

- For the last layer in our network this is easily done

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L)$$

- The first term after the last equal depends on both the choice of loss function and choice of activation function

The equations of back propagation

- For the rest of the layers in the network we have that ($\odot$ is elementwise product)

$$\frac{\partial \mathcal{L}}{\partial z_j^\ell} = \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{\ell+1}} \frac{\partial z_k^{\ell+1}}{\partial z_j^\ell}$$

$$= \sum_k w_{k,j}^{\ell+1} \sigma'(z_j^\ell) \frac{\partial \mathcal{L}}{\partial z_k^{\ell+1}}$$

$$\frac{\partial \mathcal{L}}{\partial z^\ell} = ((\mathbf{W}^{\ell+1})^T \frac{\partial \mathcal{L}}{\partial z^{\ell+1}}) \odot \sigma'(z^\ell)$$

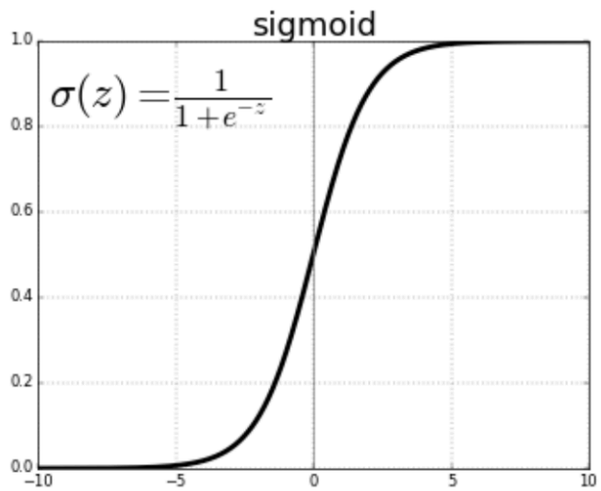
- Since

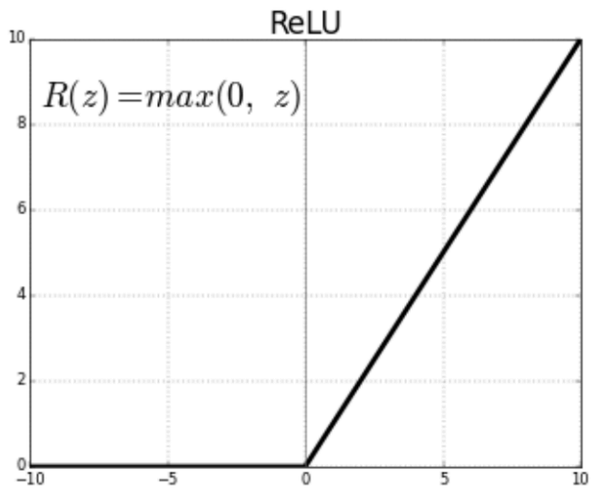
$$z_k^{\ell+1} = \sum_j w_{k,j}^{\ell+1} \sigma(z_j^\ell) + b_k^{\ell+1}$$

Activation functions

- The two most commonly used activation functions are:
- Sigmoid $$\sigma(x) = \frac{1}{1 + e^{-x}}$$
- Rectified Linear Unit:

$$\sigma(x) = \mathrm{ReLU}(x) = \max(0, x)$$





Derivatives of activation functions

- Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

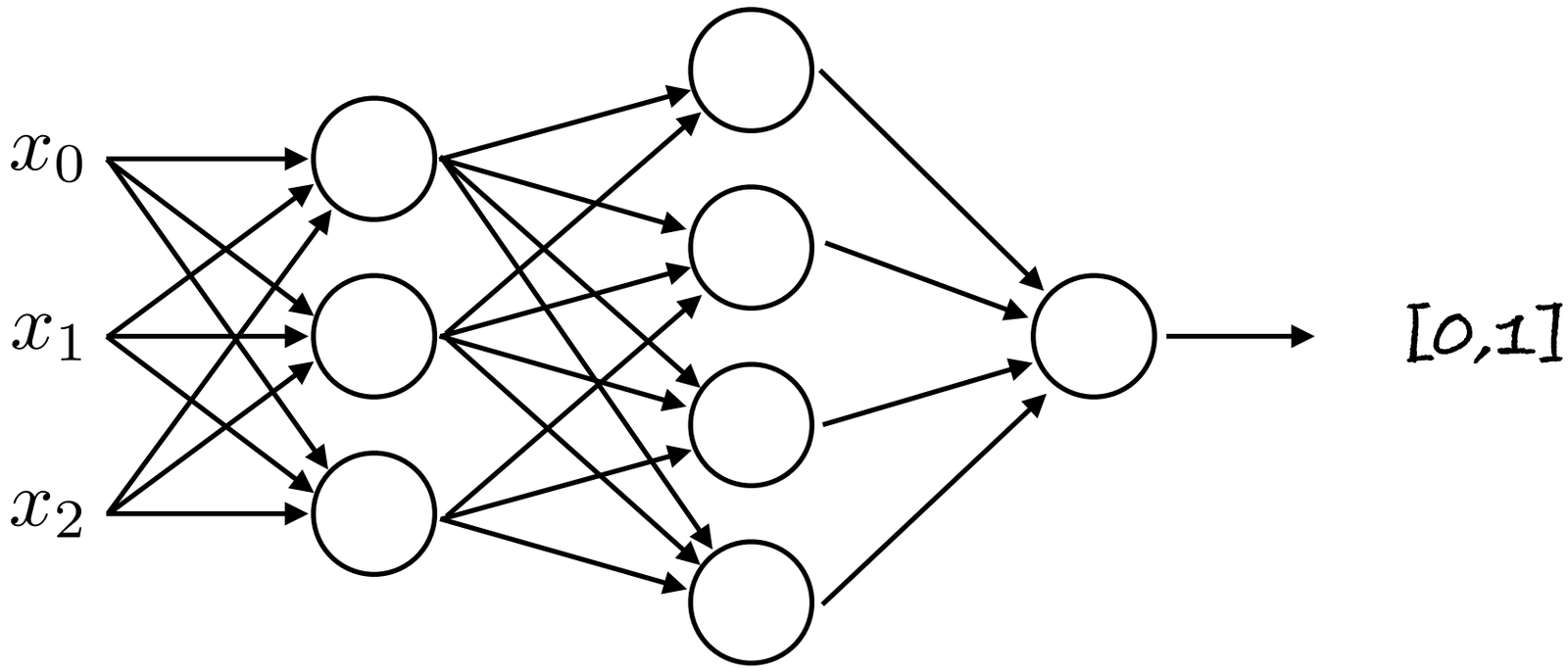
- Rectified Linear Unit:

$$\sigma(x) = \mathrm{ReLU}(x) = \max(0, x) \qquad \sigma'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Loss functions

- The choice of loss functions depends on the task
  - For regression: MSE or L1
  - For two-class classification: Binary cross-entropy
  - For multi-class classification: Cross-entropy

Two-class classification



- The output from the neuron in the last layer is mapped to the range [0,1] by using a sigmoid activation function on last neuronthe neuron

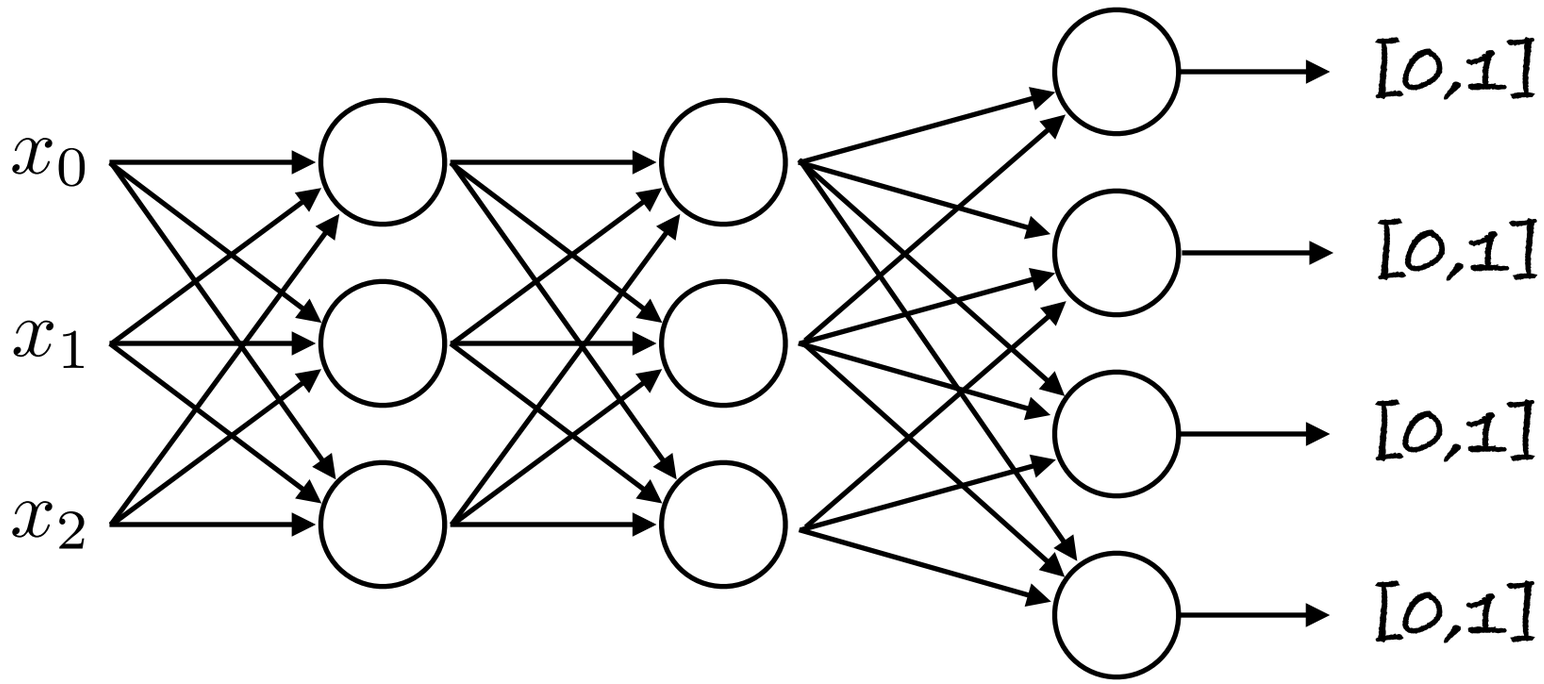
Binary cross-entropy

- If $\hat{y} \in [0, 1]$ is the output from a neural network and $y \in \{0, 1\}$ is the true value for an input *x* the binary cross-entropy is given by

$$\mathcal{L}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

$$\frac{\partial \mathcal{L}}{\partial a^L} = \frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

Multi-class classification



- The output from the neurons in the last layer is mapped to the range [0,1] such that they sum to 1 by using a softmax activation function in the last layer

Softmax activation function

- The softmax activation function maps the outputs from all neurons in layer to the range [0,1] such that they sum to 1

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

$$\sum_i x_i = 1$$

Cross-entropy

- The cross-entropy is given by

$$\mathcal{L}(\hat{y}, y) = -\sum_i y_i \log \hat{y}_i = -\log \hat{y}_I, \quad I = \arg_i(y_i = 1)$$

- Remember that to start the backpropagation we need to compute

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L)$$

- For softmax+cross-entropy we can calculate this term directly instead of computing the two terms after the last equal sign

Softmax + Cross-entropy back propagation

$$L(\hat{y}, y) = -\log a_I^L$$

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial(-\log a_I^L)}{\partial z_j^L} = \frac{\partial(-\log z_I^L + \sum_k e^{z_k^L})}{\partial z_j^L}$$

$$= -y_j + a_j^l$$

$$a_j^L = \hat{y}_j = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

Updating the parameters

- The parameters can now be updated

$$w_{j,k}^{\ell} \rightarrow w_{j,k}^{\bar{\ell}} = w_{j,k}^{\ell} - \alpha \frac{\partial L}{\partial w_{j,k}^{\ell}}$$

$$b_{j}^{\ell} \rightarrow b_{j}^{\bar{\ell}} = b_{j}^{\ell} - \alpha \frac{\partial L}{\partial b_{j}^{\ell}}$$

Stochastic gradient descent

- So far we only discussed how to train a network based on a single example
  - This is called *stochastic gradient descent*
- We can train over multiple training examples by simply averaging the loss and gradients over the examples
- If we use all our training examples we call it *batch gradient descent*
- If we use random subsets of our examples it is called *mini-batch gradient descent*
  - *Note: this is often mistakingly called stochastic gradient descent*
- In practice we always use mini-batch gradient descent and the size of the mini batches is a hyperparameter

Suggested reading

- Michael Nielsen -
  www.neuralnetworksanddeeplearning.com

- Goodfellow et al., Deep Learning
  www.deeplearningbook.org