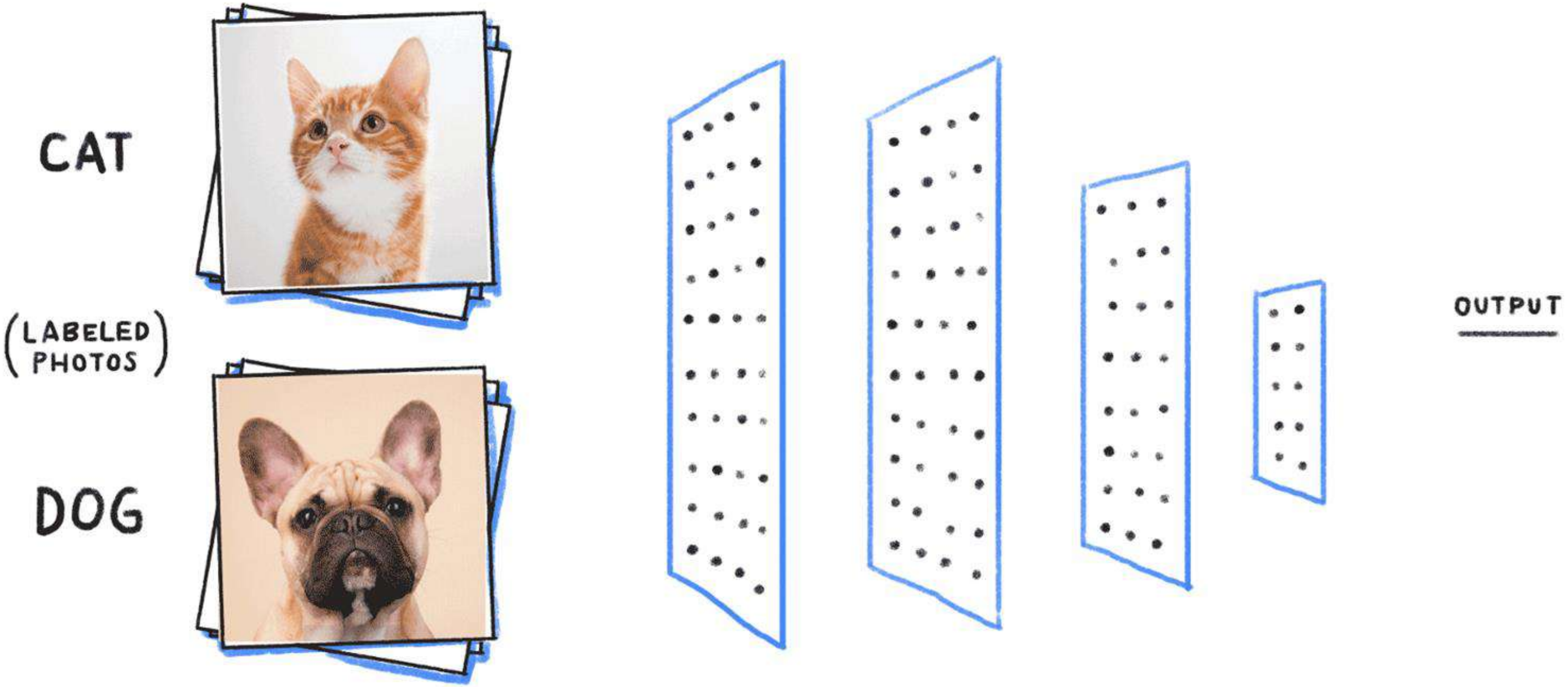
Department of Medical Physics and Biomedical Engineering
Centre for Medical Image Computing (CMIC)
Wellcome / EPSRC Centre for Interventional and Surgical Sciences (WEISS)

Deep Learning

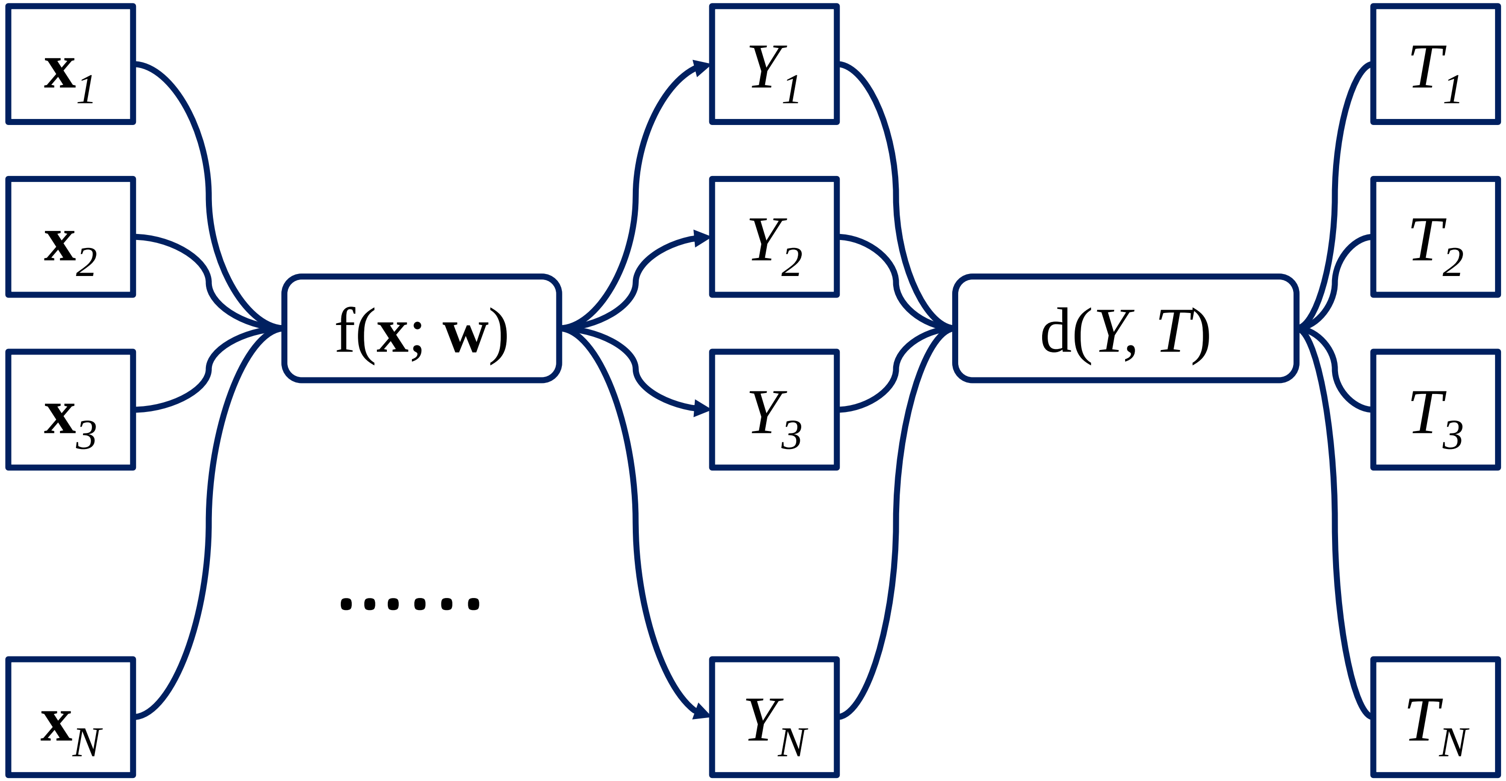
MPHY0041 Machine Learning in Medical Imaging

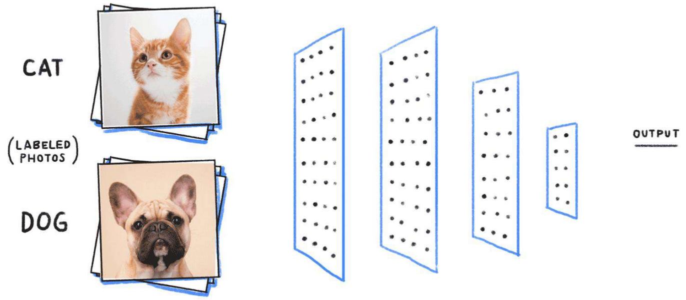
Yipeng Hu
yipeng.hu@ucl.ac.uk

Neural Networks | Supervised Learning


CAT
(LABELED PHOTOS)
DOG
OUTPUT

Training

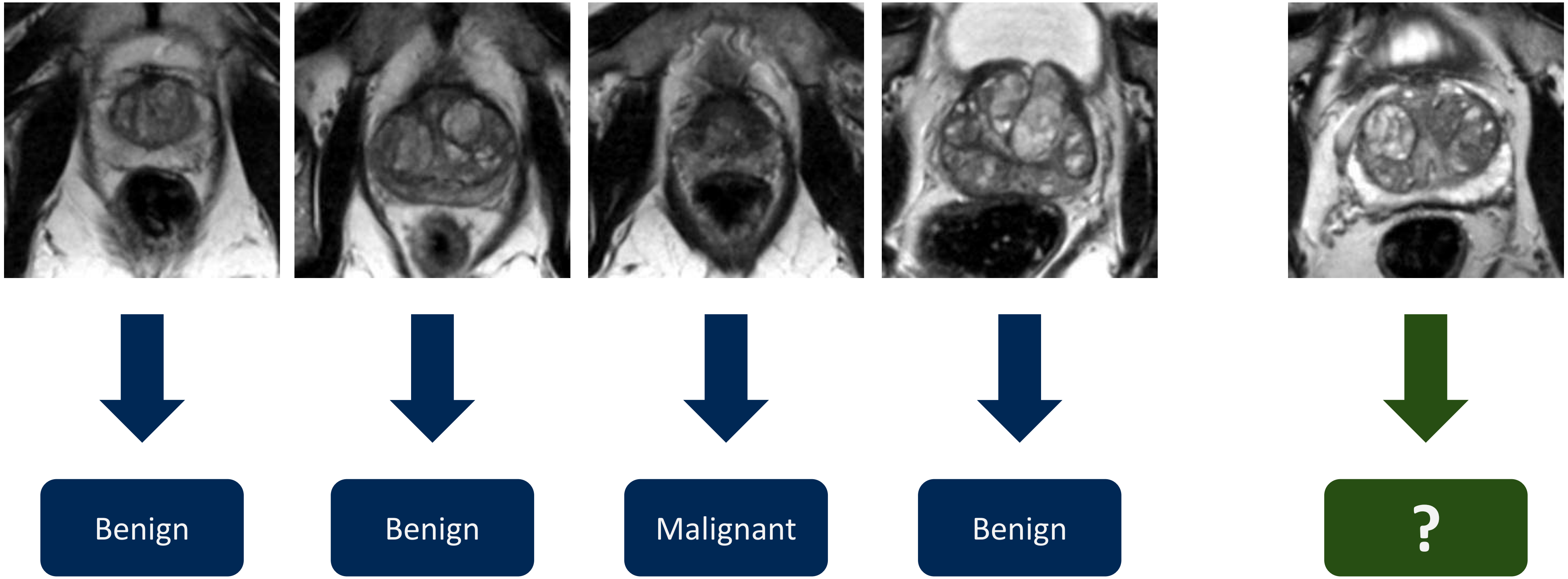




Prediction (Testing, Inference)

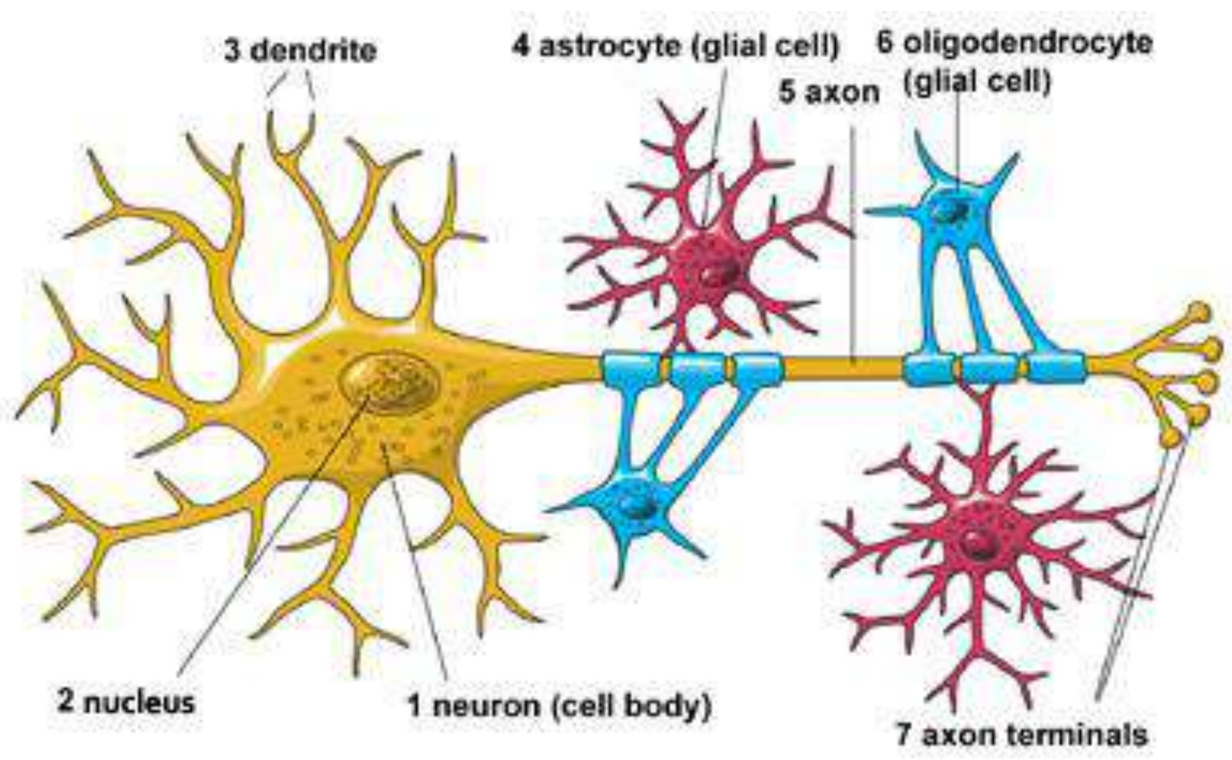
$$\mathbf{x}_{new} \longrightarrow f(\mathbf{x};\, \mathbf{w}_{opt}) \longrightarrow Y_{new}$$

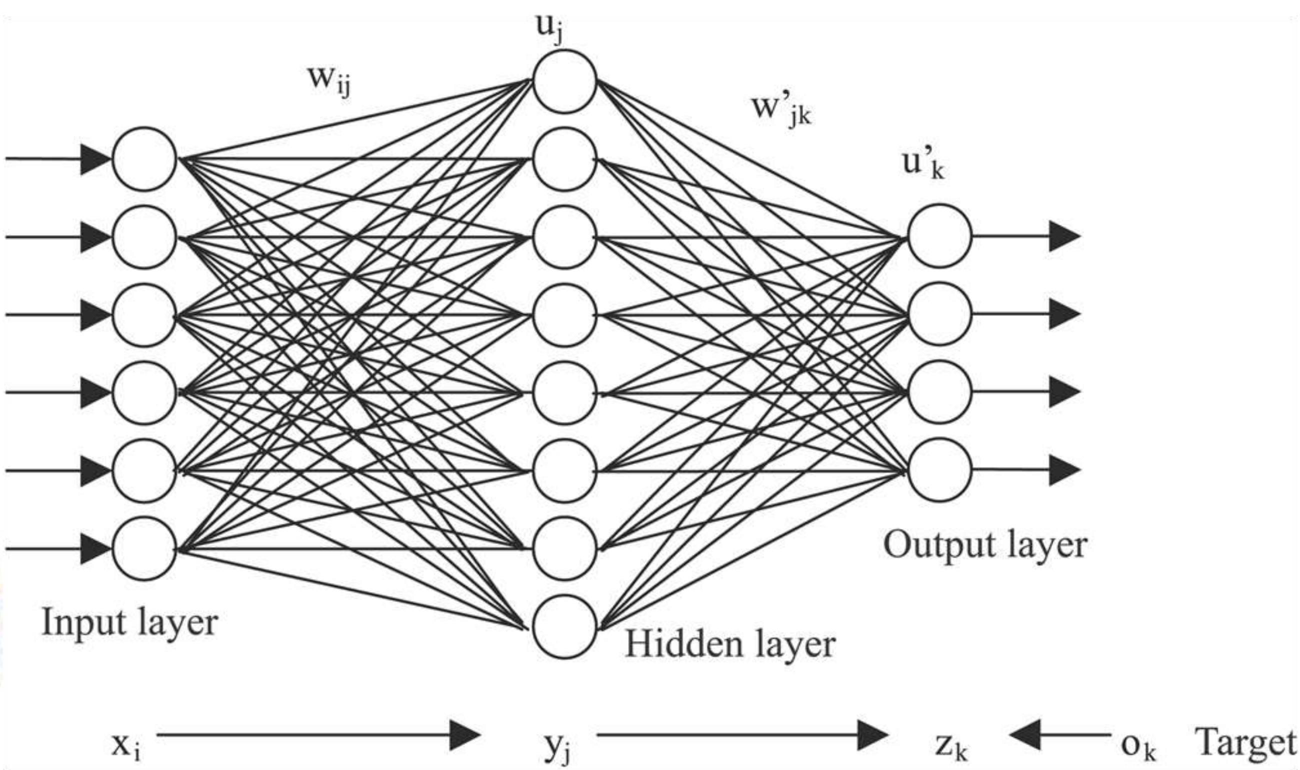
Computer-assisted diagnosis (CAD) for prostate cancer



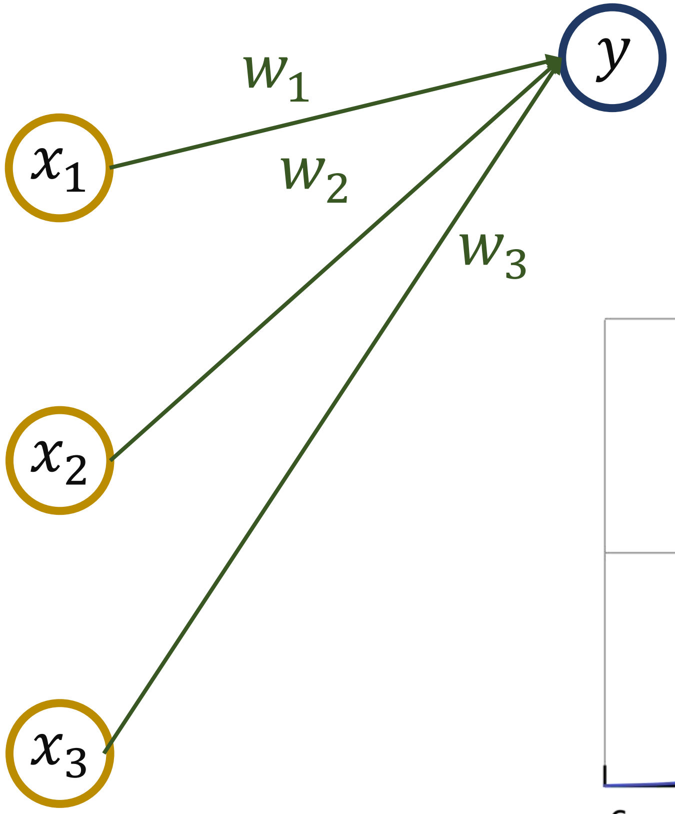
Neural Networks | Multilayer Perceptron

Artificial neural networks





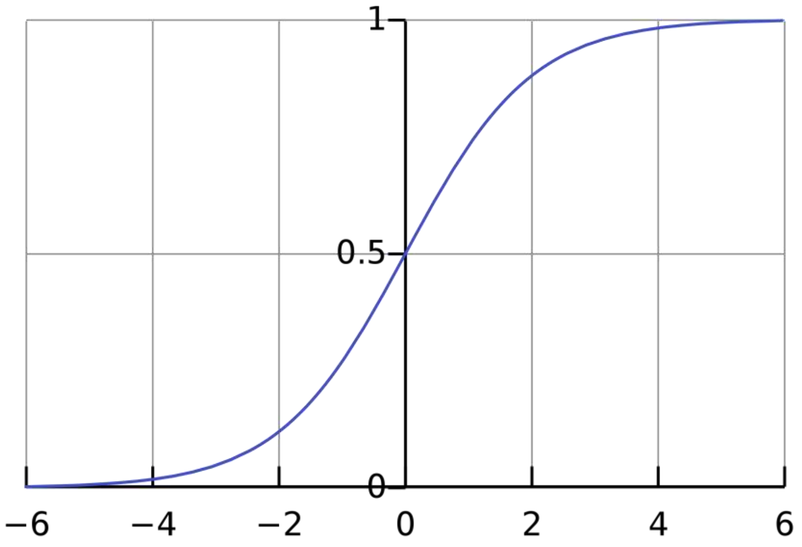
Logistic regression



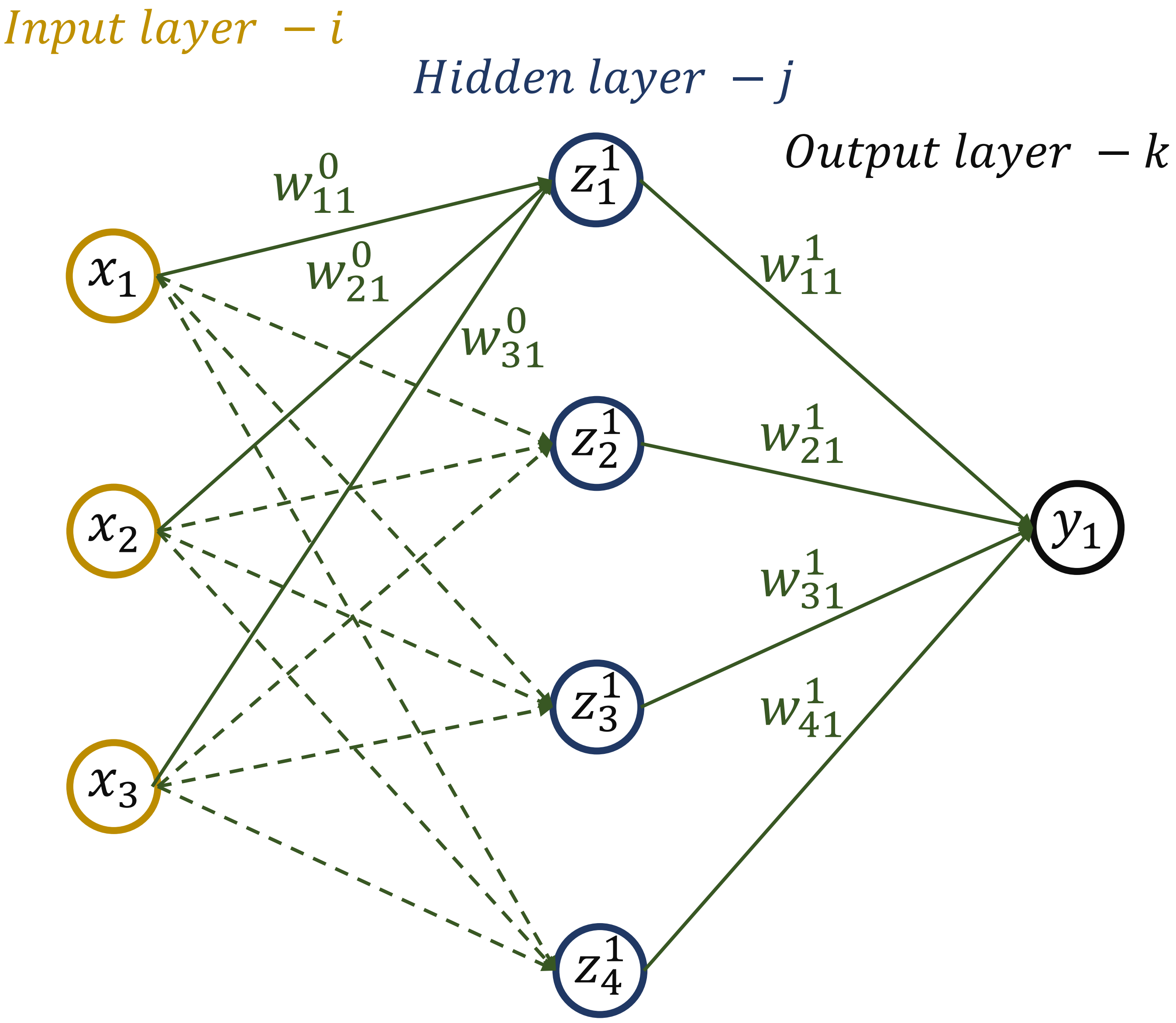
Input → y

$$y = \sigma\left[\sum_{i=1}^{i=3} x_i \mathrm{w}_i \;\; + \boldsymbol{b}\right] = \sigma[\mathbf{w}^{\mathrm{T}}\mathbf{x} + \boldsymbol{b}]$$

Activation function: $\sigma(z) = \frac{1}{1+\exp(-z)}$



Q: roles of activation function?



Activation function: $\sigma \rightarrow g$

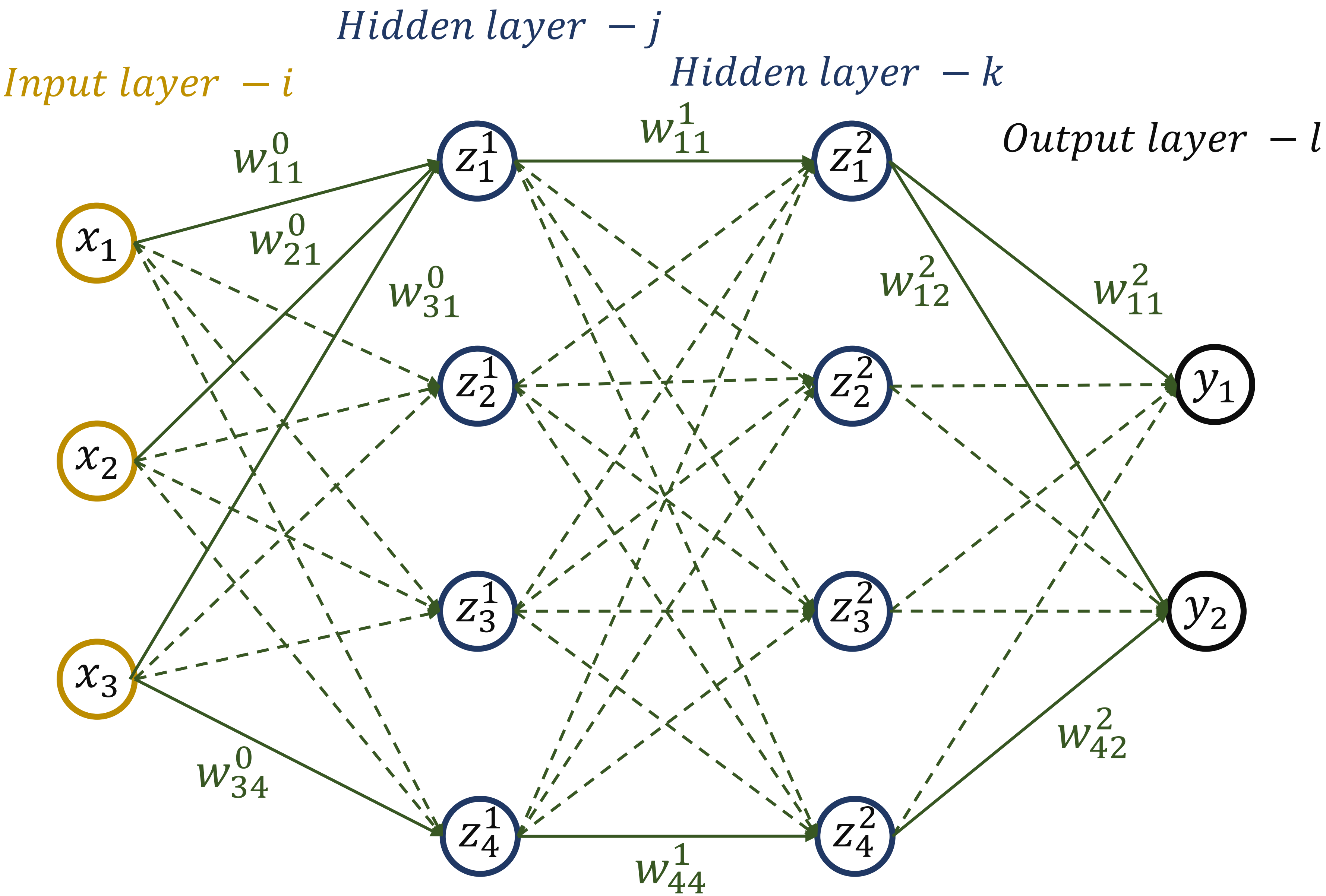
Input → H1: i → j

$$\mathrm{z}_j^1 = g_j^1 \left[\sum_{i=1}^{i=3} x_i \mathrm{w}_{ij}^0 + b_j^1 \right] = g_j^1 \left[\left(\mathbf{w}_j^0\right)^{\mathrm{T}} \mathbf{x} + b_j^1 \right]$$

H1 → Output: j → k

$$f(\mathbf{x}, \boldsymbol{W}) = y_k = g_k^1 \left[\left(\mathbf{w}_k^1\right)^{\mathrm{T}} \mathbf{z}^1 + b_k^1 \right]$$

where $\mathbf{z}^1 = [\mathrm{z}_1^1, \mathrm{z}_2^1, \mathrm{z}_3^1, \mathrm{z}_4^1]^{\mathrm{T}}, \boldsymbol{k} = \mathbf{1}$



Output layer $-l$

Input → H1: i → j

$$z^1_j = g^1_j\left[\left(\mathbf{w}^0_j\right)^{\mathrm{T}}\mathbf{x} + b^1_j\right]$$

H1 → H2: j → k

$$z^2_k = g^1_k\left[\left(\mathbf{w}^1_k\right)^{\mathrm{T}}\mathbf{z}^1 + b^1_k\right]$$

H2 → Output: k → l

$$y_k = g^1_l\left[\left(\mathbf{w}^2_l\right)^{\mathrm{T}}\mathbf{z}^2 + b^2_l\right]$$

Neural Networks | Activation Functions

$$h = g\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + \boldsymbol{b}\right)$$

h: hidden layer (output feature vector)
x: Input feature vector
w: Weights – network parameters
b: Bias
g(): Activation function

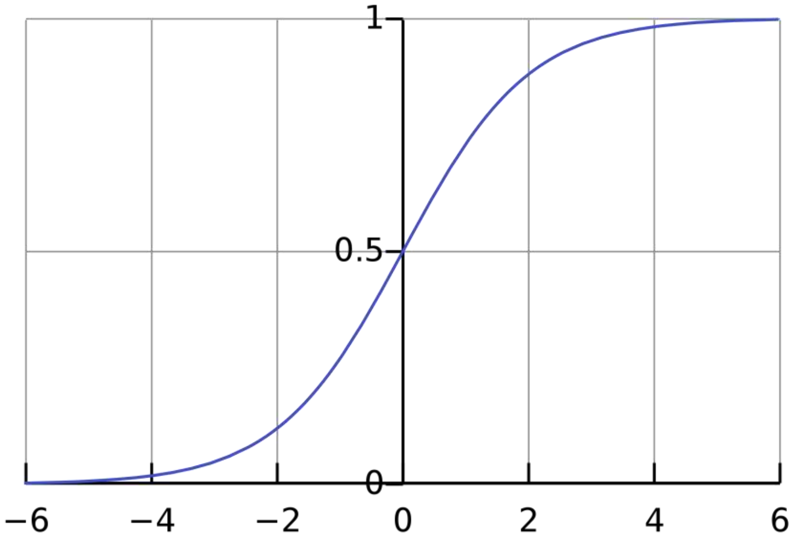
Activation functions

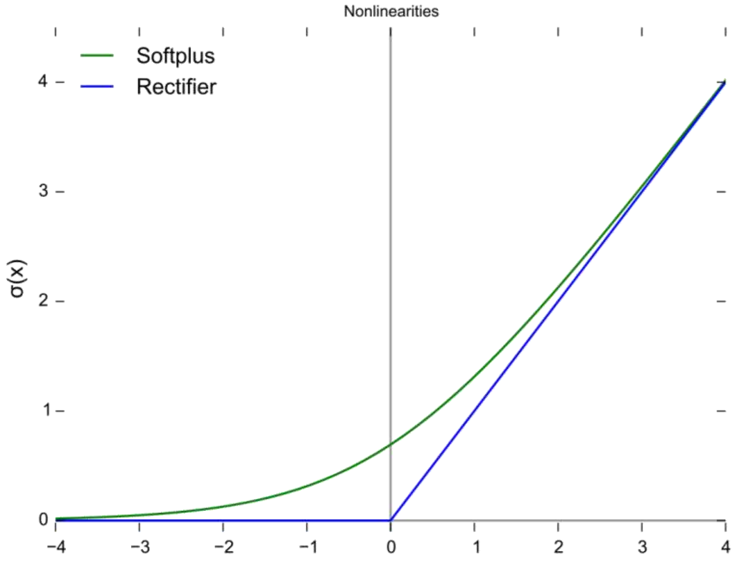
Q: why do we need activation functions?

$$
\begin{aligned}
y &= w_2^{\mathrm{T}}\left(w_1^{\mathrm{T}}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2 \\
&= w_2^{\mathrm{T}} w_1^{\mathrm{T}}\mathbf{x} + \left(w_2^{\mathrm{T}}\mathbf{b}_1 + \mathbf{b}_2\right) \\
&= w_3^{\mathrm{T}}\mathbf{x} + \mathbf{b}_3
\end{aligned}
$$

Activation functions (hidden units)

- Logistic sigmoid $\quad g(z) = \sigma(z) = \frac{1}{1+\exp(-z)}$
- Hyperbolic tangent $\quad g(z) = \tanh(z) = 2\sigma(2z) - 1$
- Rectified linear unit (ReLU) $\quad g(z) = \max\{0, z\}$





Activation functions (output units)

- Bernoulli output $$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)} \qquad 0 \leqslant y(\mathbf{x}, \mathbf{w}) \leqslant 1.$$
- Multinoulli output (softmax) $$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \qquad 0 \leqslant y_k \leqslant 1 \text{ and } \sum_k y_k = 1.$$
- Gaussian output

Q: what is the activation function for Gaussian output?

Other activation functions

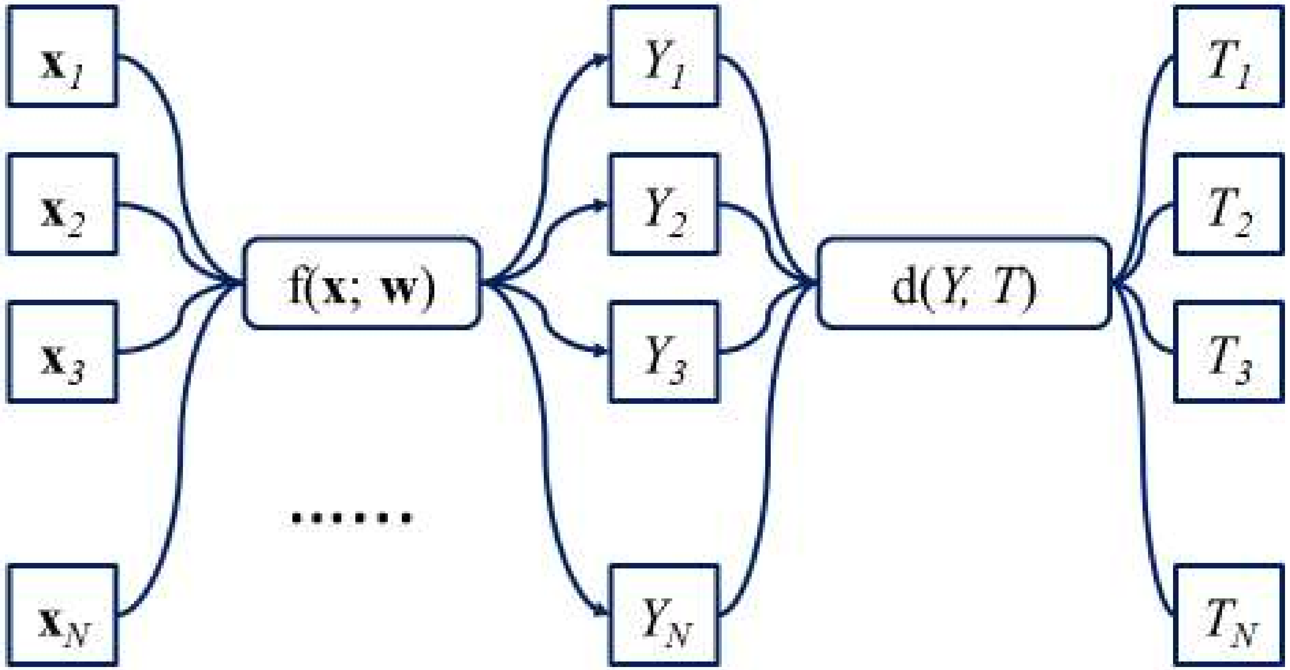
- Permutation-invariant, e.g. max, mean, min...
- Application specific, range and distribution, e.g. displacement/velocity, function parameters

Neural Networks | Loss Functions

Neural Networks | Loss Functions

Neural Networks | Supervised Learning

Training



Prediction (Testing, Inference)

Loss functions

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Given training data {$\mathbf{x}_n$, t_n}

- Regression loss

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

- Classification loss

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$$

- Multi-class classification loss

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}).$$

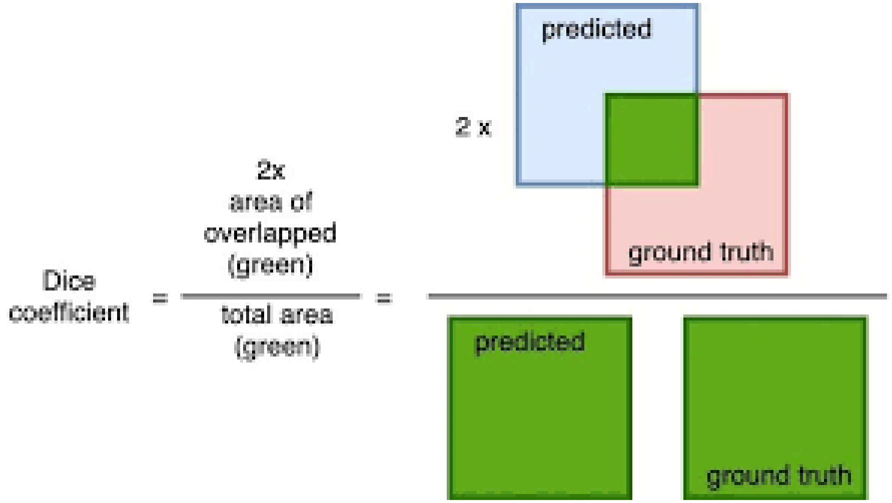
$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))} \qquad 0 \leqslant y_k \leqslant 1 \text{ and } \sum_k y_k = 1.$$

Loss functions

Given training data {$\mathbf{x}_n$, t_n}

- Cost-sensitive loss
- Dice loss
- Application-specific loss

	Predicted +ve	Predicted -ve
Positive		£?
Negative	£?	



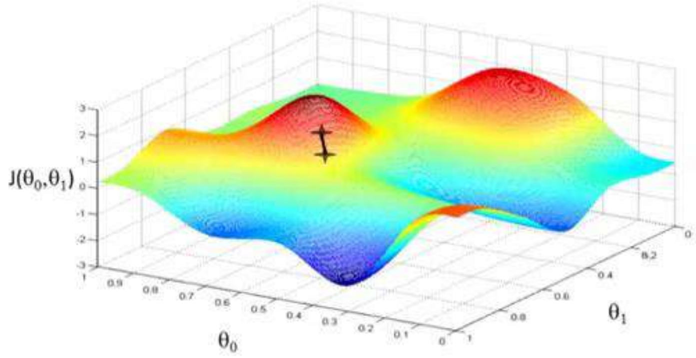
$$DiceLoss(\mathbf{x}, \boldsymbol{t}) = -\frac{2\sum_{i=1}^{I} x_i \cdot t_i}{\sum_{i=1}^{I} x_i + \sum_{i=1}^{I} t_i}$$

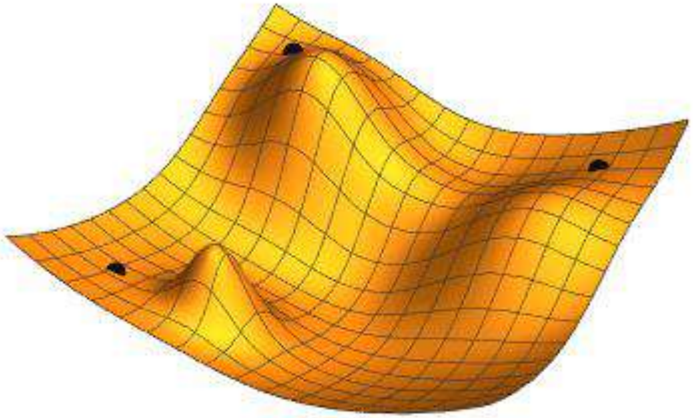
Neural Networks | Backpropagation

$$x^* = \arg\min f(x)$$

Gradient-descent

$$\frac{\partial}{\partial x_i} f(x)$$

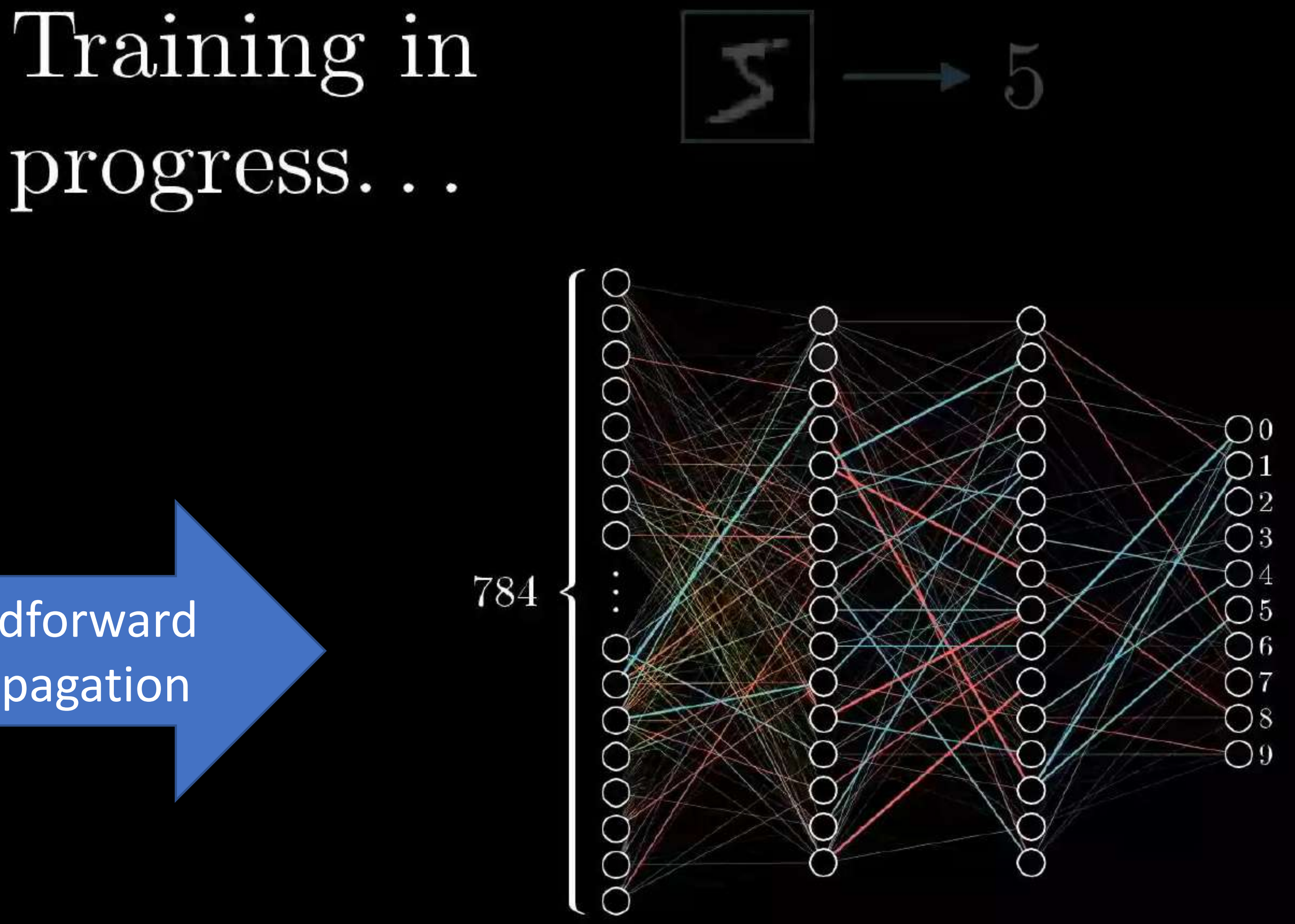




Andrew Ng

$$y_k = \sum_i w_{ki} x_i \qquad E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$$

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$



Feedforward propagation

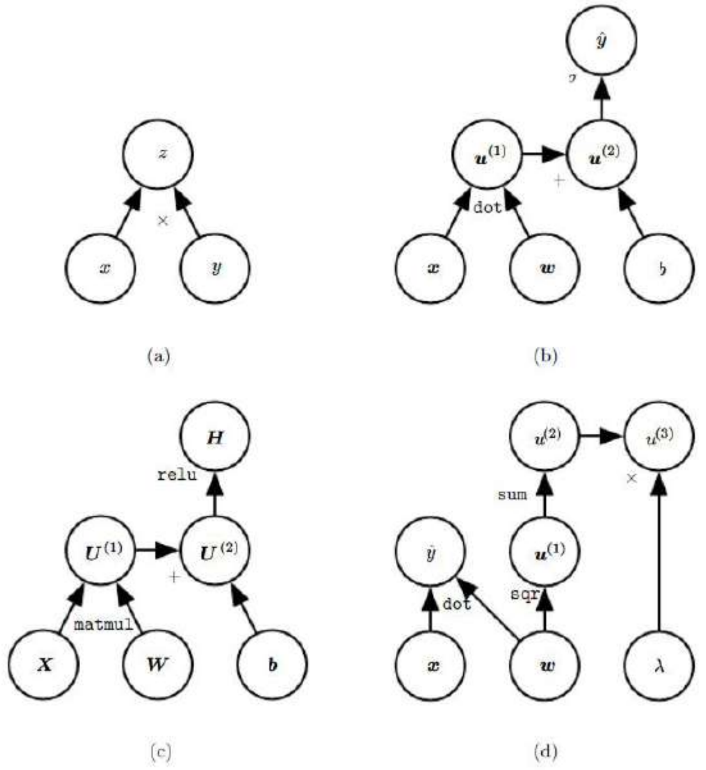
Back-propagation

An efficient algorithm

A general procedure for subset of variables, vector output and arbitrary function

$$y = g(x) \text{ and } z = f(g(x)) = f(y).$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}.$$



Computational graph

The Chain Rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

Automatic differentiation

Implementation considerations

- Store the intermediate gradient calculation or re-compute on the fly, at each node
- Linear complexity the number of node edges
- Memory consumption, e.g. Jacobian in larger tensors
- Parallel computing, PyTorch and TensorFlow
- Other gradient-based optimisation?

Stochastic gradient descent

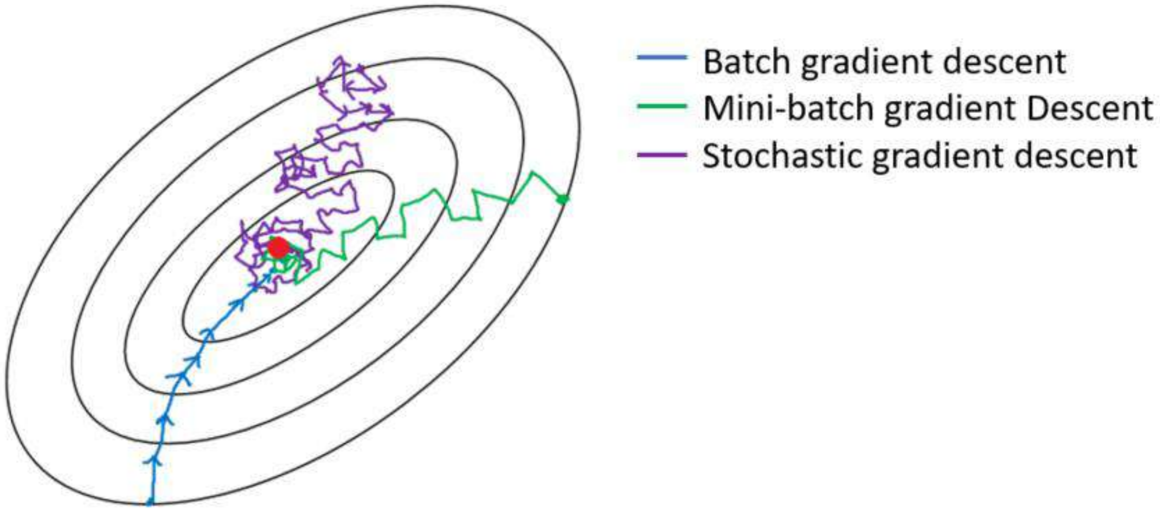
$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}.$$

- (Batch) gradient descent $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$
- Stochastic gradient descent

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}).$$



- Minibatch gradient descent

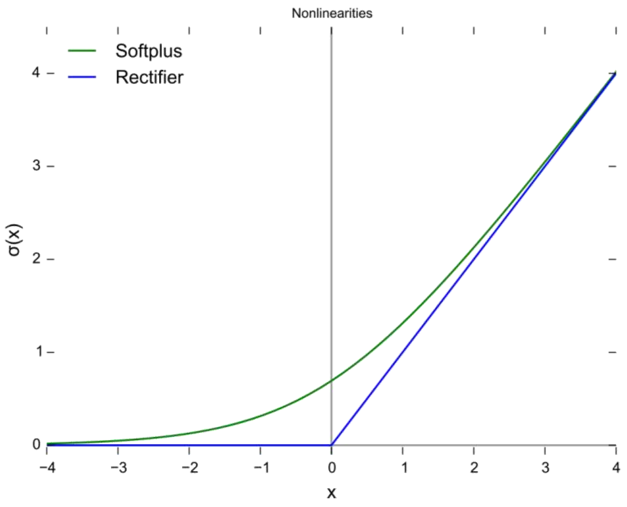
Q: how do you prove MBG converges to local minima?

When a function is not differentiable everywhere...

- Approximation

Q: How to approximate the max function using a differentiable function?

- Numerical solution



Definition of deep learning

- A class of machine learning algorithms
- Using deep neural networks
- Hierarchical feature extraction and representation
- Optimised by gradient-based backprop