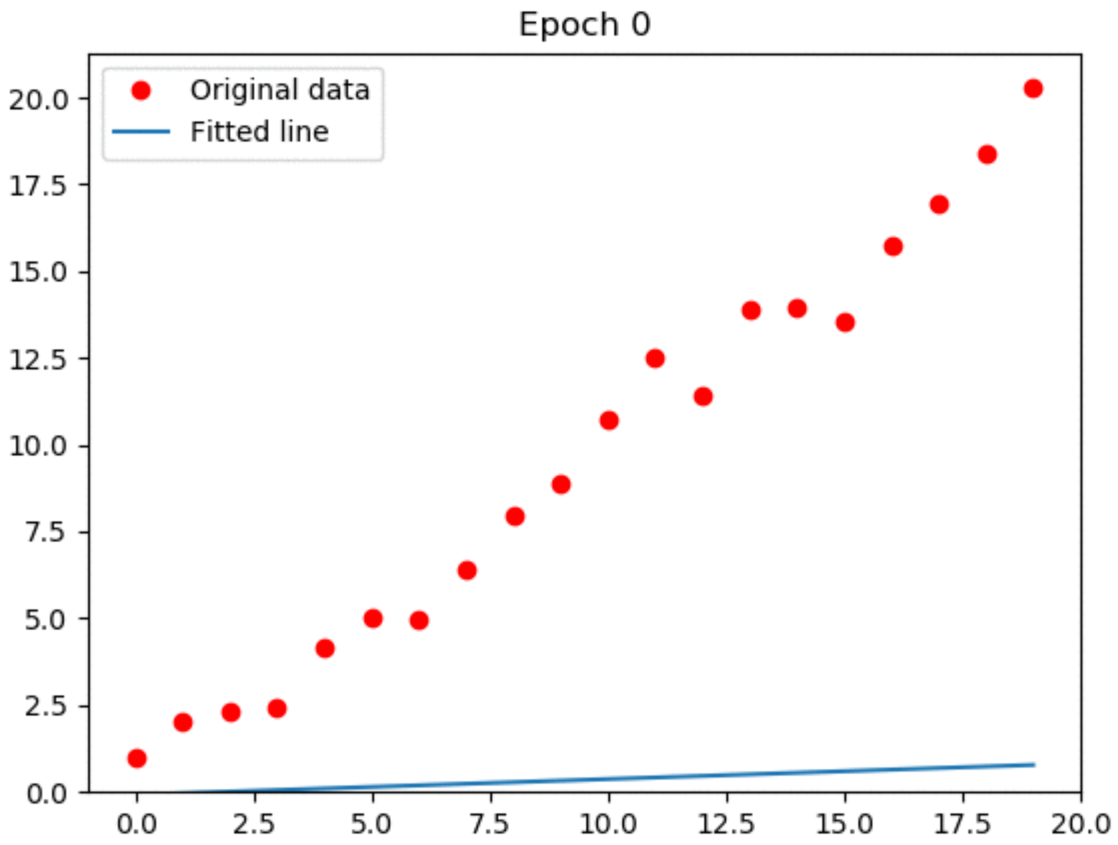
Deep Learning – I

1. Linear regression revisit
2. Multilayer perceptron
3. Convolutional neural network
4. Data preprocessing

Linear regression revisit

Linear regression revisit



Math behind:

1: Data $\left(x_1, y_1\right), \ldots, \left(x_n, y_n\right)$

2: Model $y = \boldsymbol{w}x + \boldsymbol{b}$

3: Loss function: mean square error

$$L(x, \theta) = \sum_{i=1}^{n} |(\boldsymbol{w}x_i + \boldsymbol{b}) - y_i|^2$$

$\theta = \{\boldsymbol{w}, \boldsymbol{b}\}$

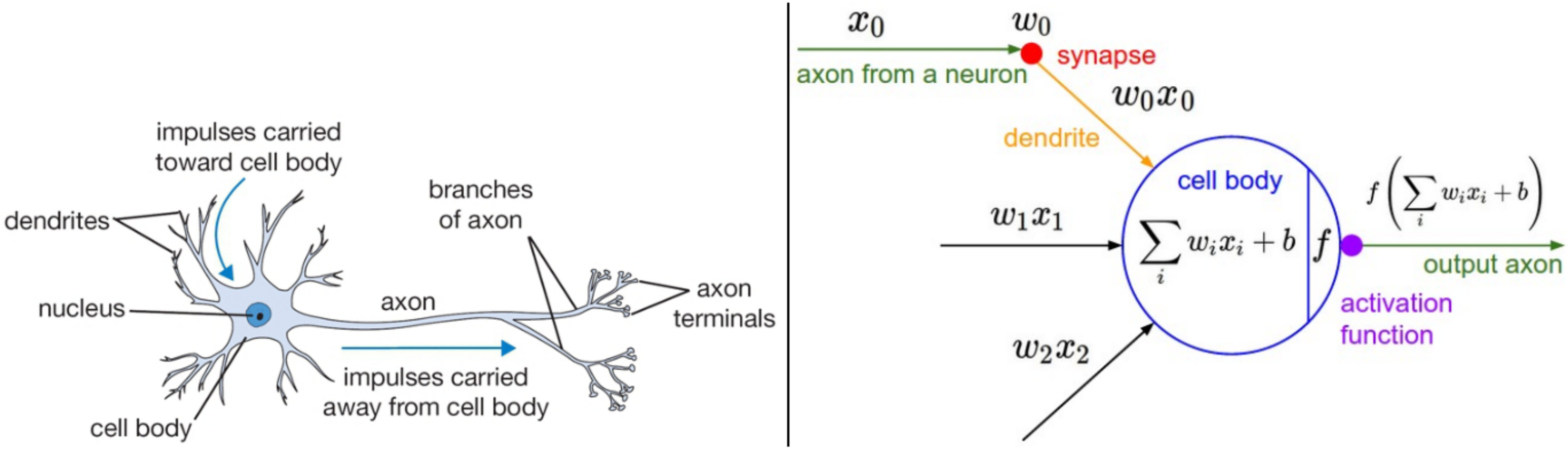
4: Optimization (training): gradient descent

$$\theta^{j+1} = \theta^j - \nabla_\theta L(x, \theta^j)$$

5: Inference (deployment)

Multilayer perceptron

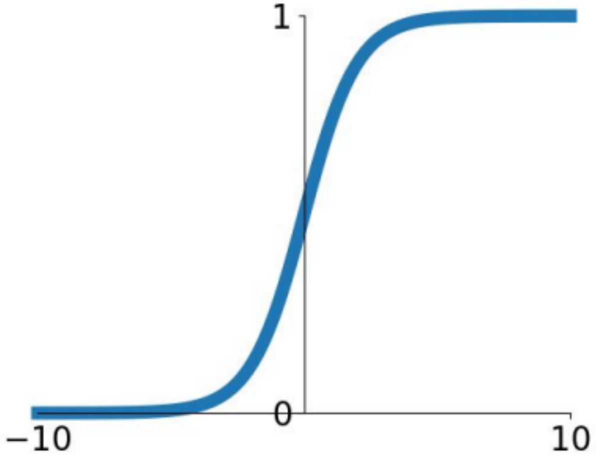
Multilayer perceptron



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

f is nonlinearity: Tanh, ReLu, leaky ReL
sigmoid, etc.

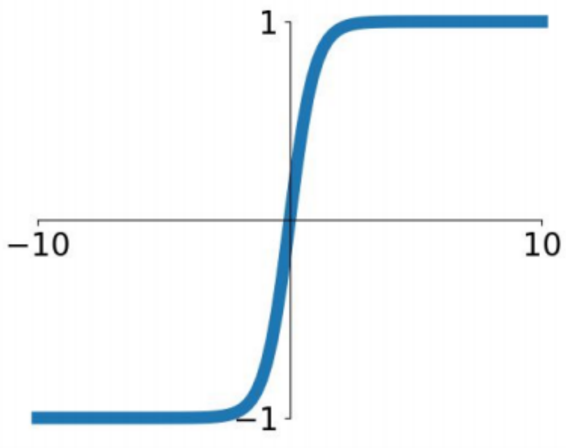
Activation Functions – Sigmoid



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

1. Normalise numbers to [0, 1]
2. Saturated neurons kill gradients
3. Exponential function is more expensive

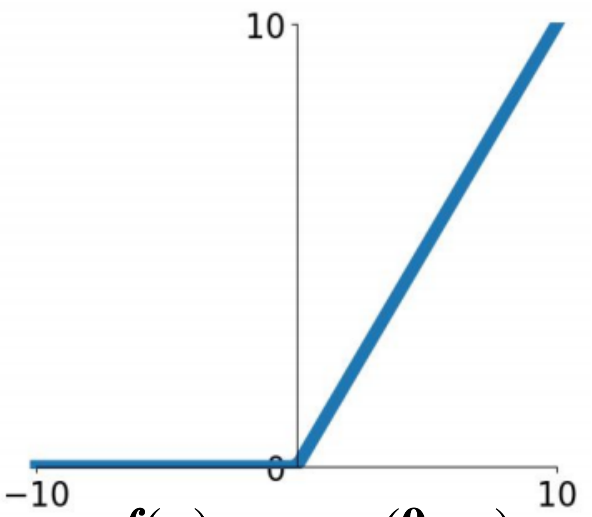
Activation Functions – tanh



$$f(x) = tanh(x)$$

1. Normalise numbers to [-1 1]
2. Saturated neurons kill gradients

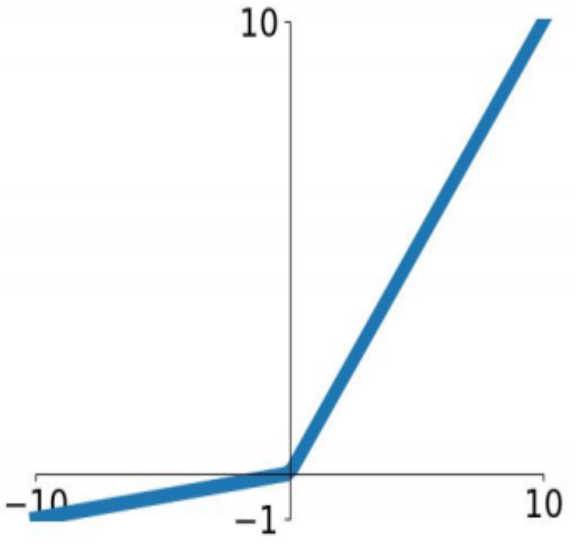
Activation Functions – RuLU (Rectified Linear Unit)



$$f(x) = max(0,\ x)$$

1. Does not saturate in positive region
2. Converges much faster than sigmoid/tanh (eg. 6x)
3. Dead ReLU will have no gradients

Activation Functions – Leaky ReLU



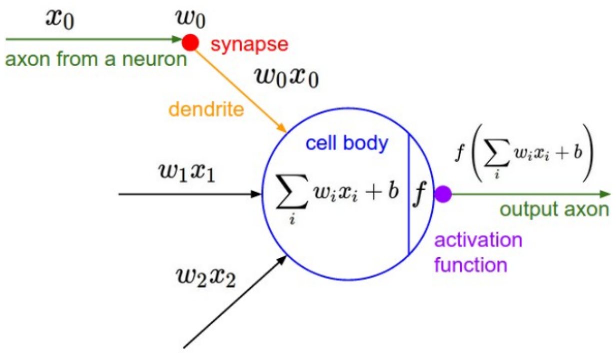
$$f(x) = max(0.01x,\ x)$$

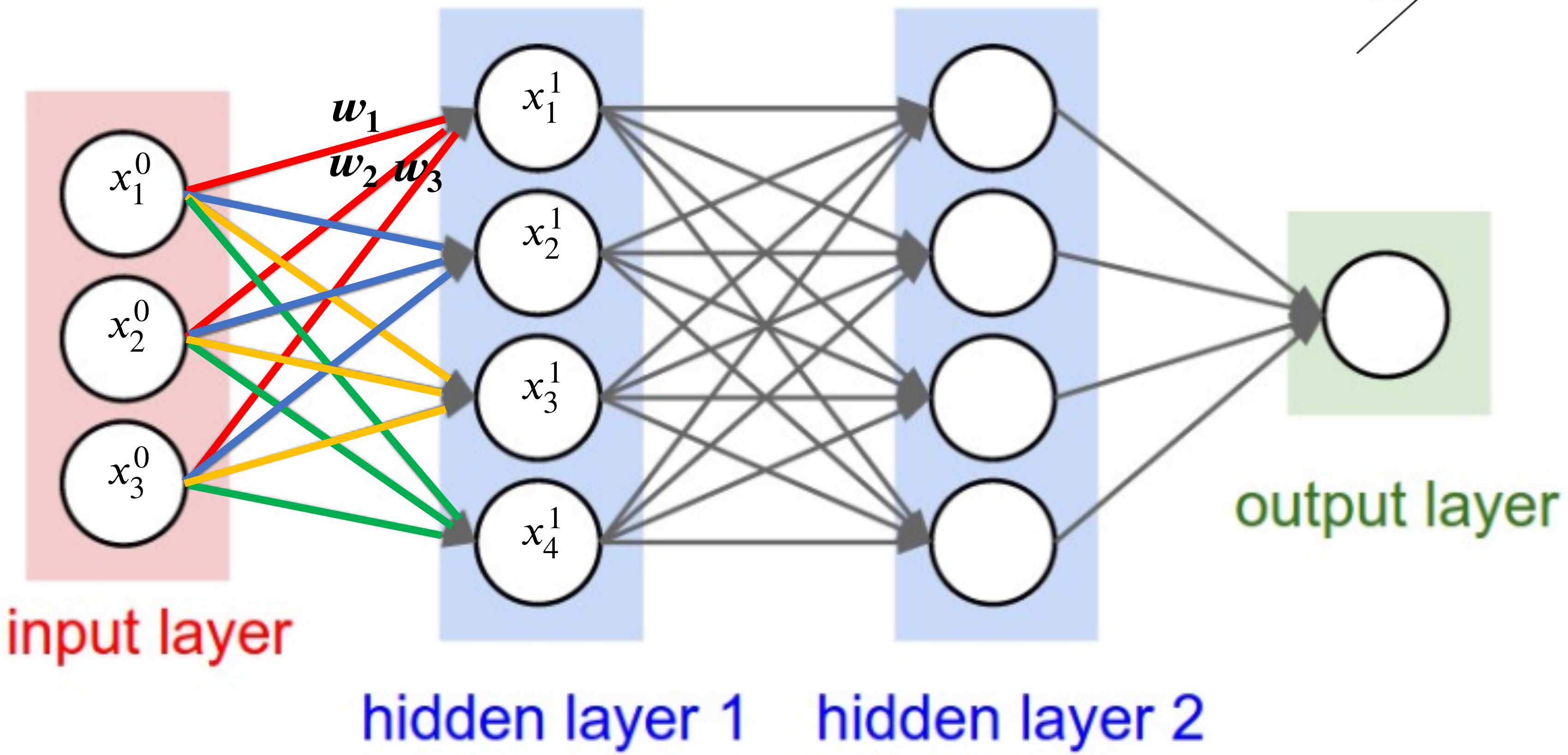
1. Does not saturate in both negative and positive regions
2. Converges much faster than sigmoid/tanh (eg. 6x)
3. Will not die

Tips in practice

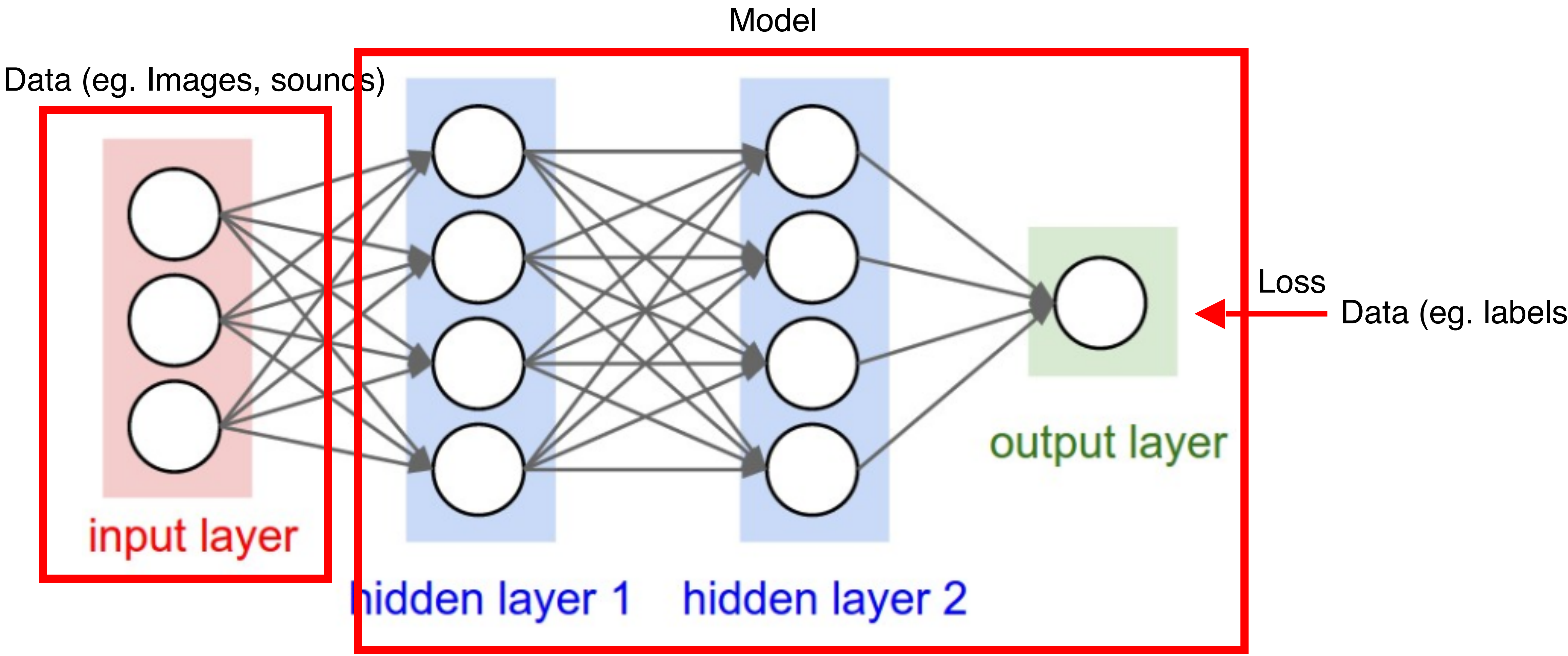
- ReLU is the most popular choice
- Try out Leaky ReLU sometimes
- Try out tanh but do not expect much (normally used at last layer)
- Use Sigmoid only at the last layer

Multilayer perceptron

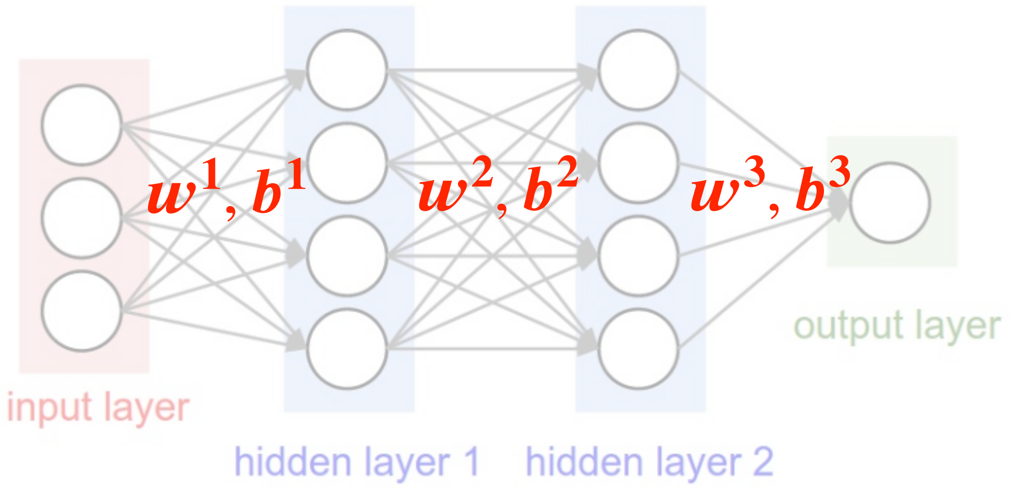




Multilayer perceptron



Multilayer perceptron



For classification

See Lab 7&8 for details

$$x^{(1)} = f\left(W^{(1)}x + b^{(1)}\right)$$
$$x^{(2)} = f\left(W^{(2)}x^{(1)} + b^{(2)}\right)$$
$$\cdots\cdots$$
$$x^{(n)} = f\left(W^{(n)}x^{(n-1)} + b^{(n)}\right)$$
$$y_i = \frac{\exp(x_i^{(n)})}{\sum_{j=1}^{L}\exp(x_j^{(n)})} \quad \text{softmax}$$

Optimisation

- Loss function

$$\min_{\theta} L(x,\theta) = -\sum_{i=1}^{L} z_i \log(y_i(x,\theta))$$

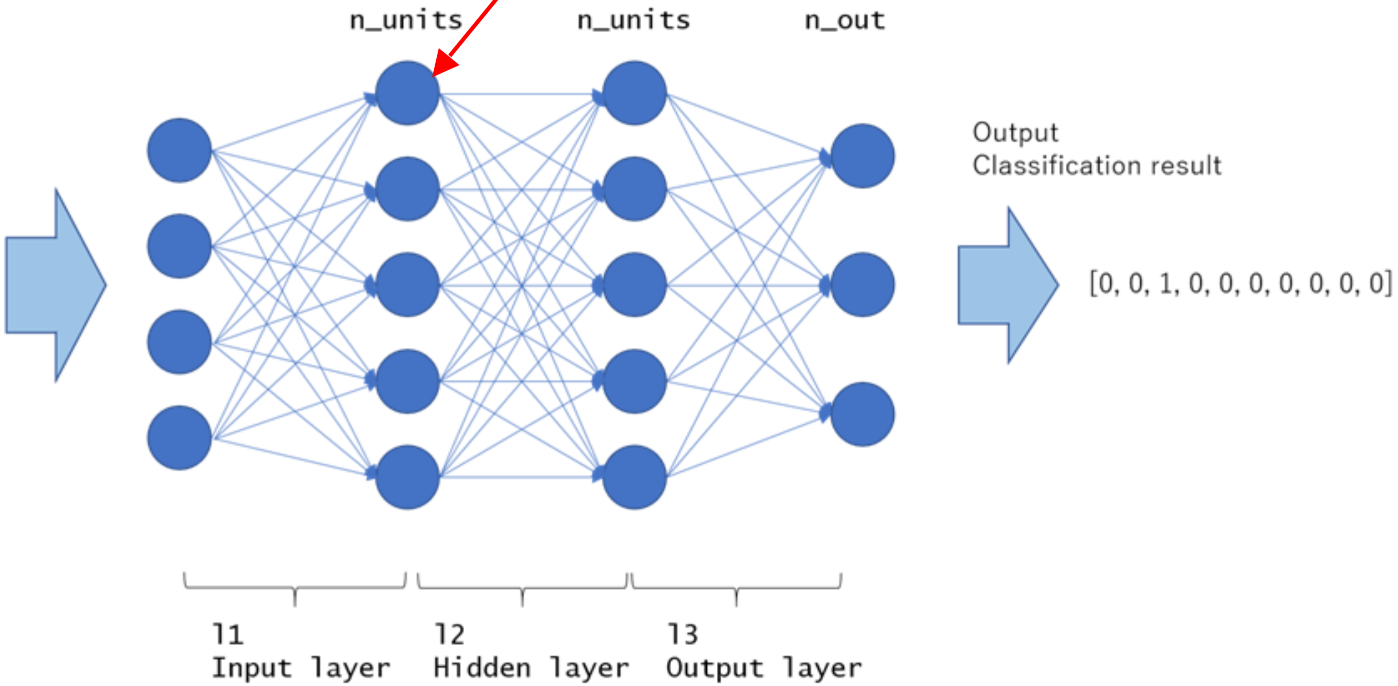
$\theta = \{W^{(i)}, b^{(i)}\}$ labels predicted label map

- Stochastic gradient descent (SGD)

$$\theta^{(n)} = \theta^{(n-1)} - \nabla_\theta L(x, \theta^{(n-1)})$$

500x500 pixels

We need 250k number of weights/parameters to map the input to a single neuron. Too expensive!



Convolutional Neural Network

Dot product and convolution

The dot product of two vectors **a** = $[a_1, a_2, \ldots, a_n]$ and **b** = $[b_1, b_2, \ldots, b_n]$ is defined as:

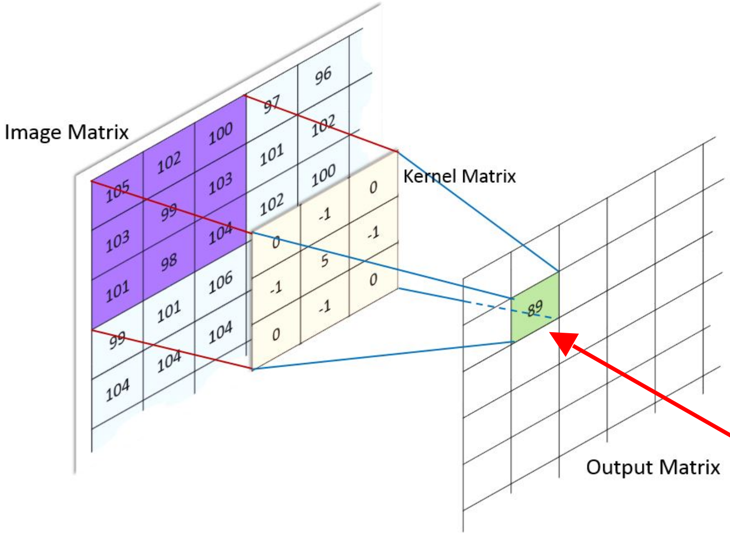
$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

The convolution between an image x and a kernel $\boldsymbol{w}$ is given as

$$G = \boldsymbol{w} * x \qquad G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \boldsymbol{w}[u,v]\, x[i-u, j-v]$$

Where u, v are indices in the kernel grid and i, j are indices in the image grid. k denotes the radius of the kernel.

Convolutional Neural Network

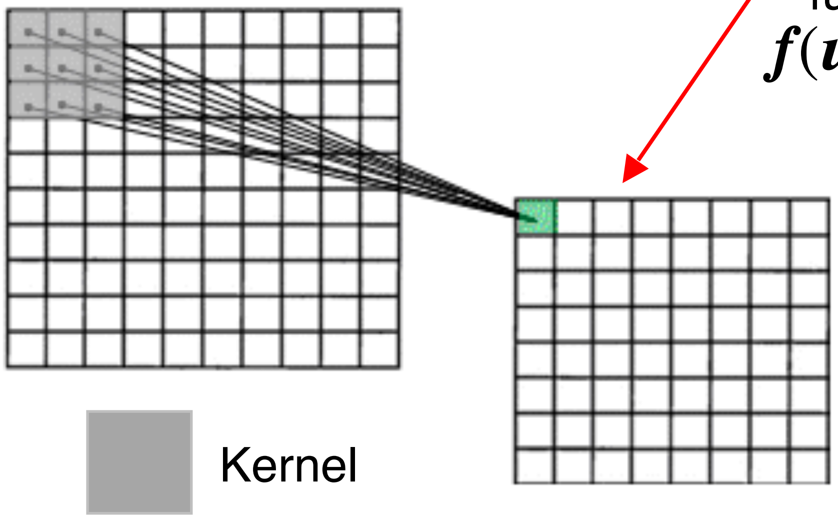


dot product

followed by activation function f

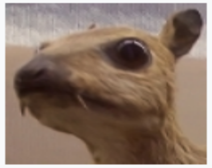
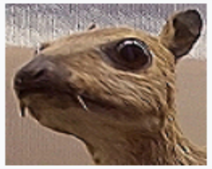
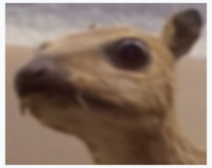
$$f(\boldsymbol{w} * x + b)$$

Image matrix

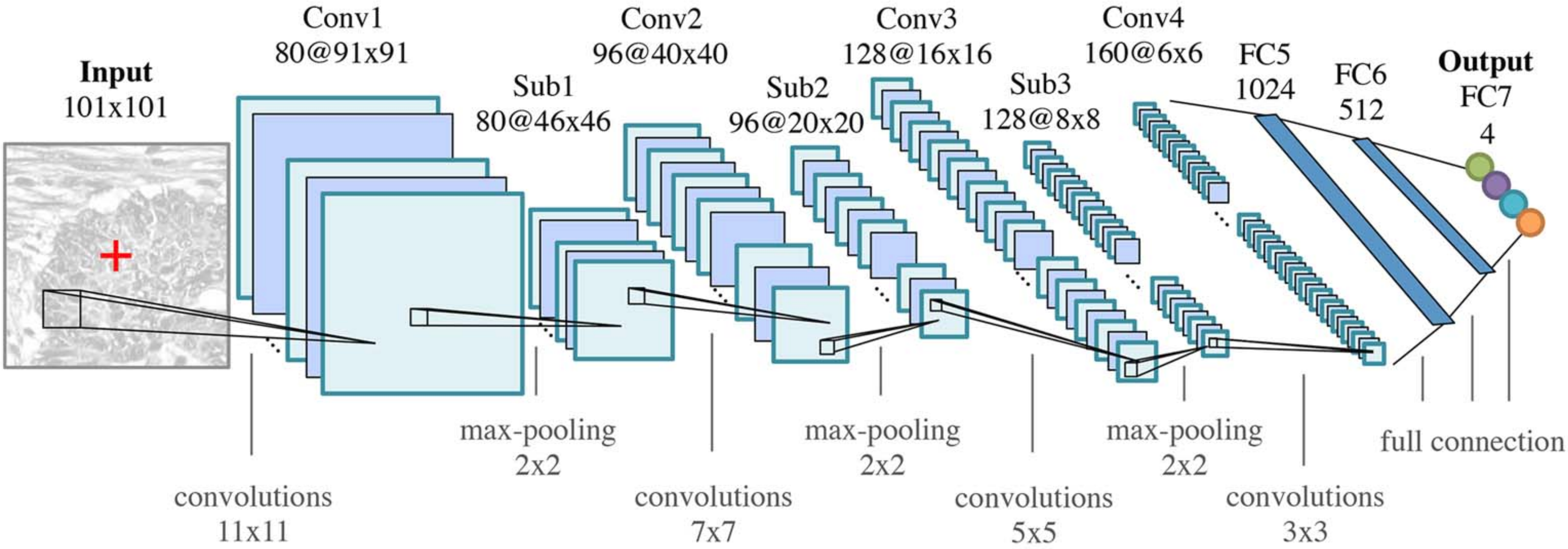


convolution

Depending on values, a kernel can cause a wide range of effects

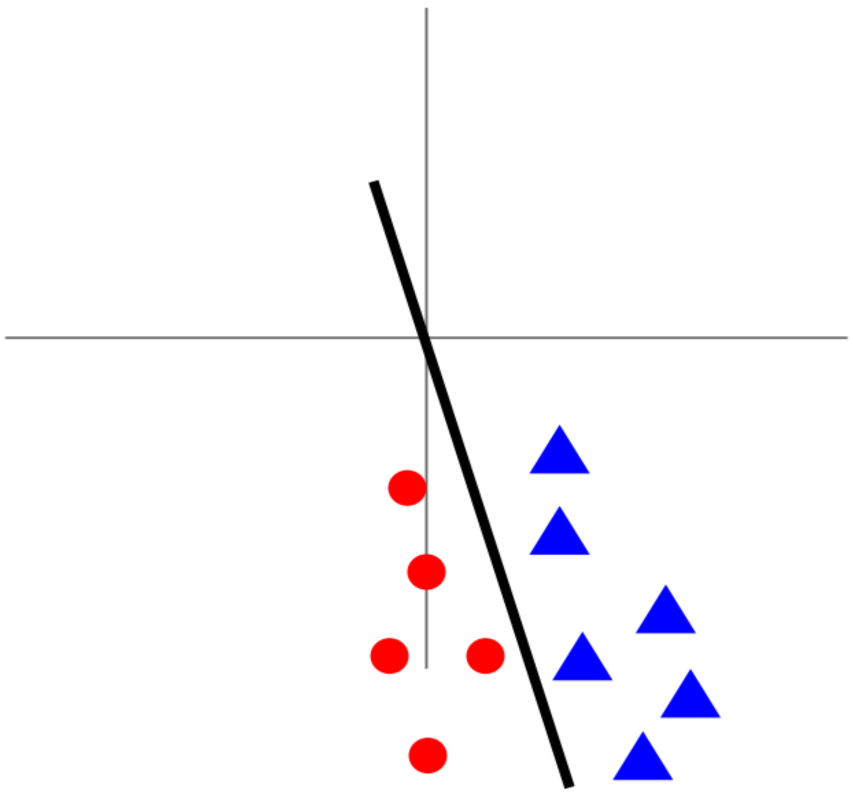
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolutional Neural Network

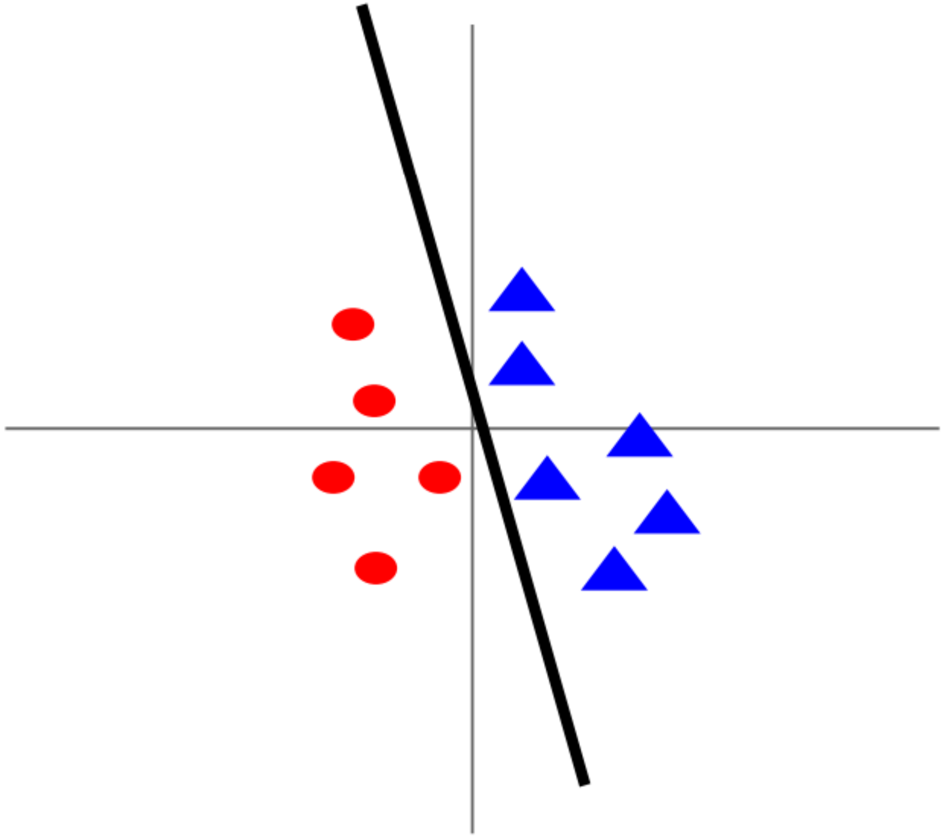


Data/Model/Loss function/Optimisation/Inference

Data preprocessing



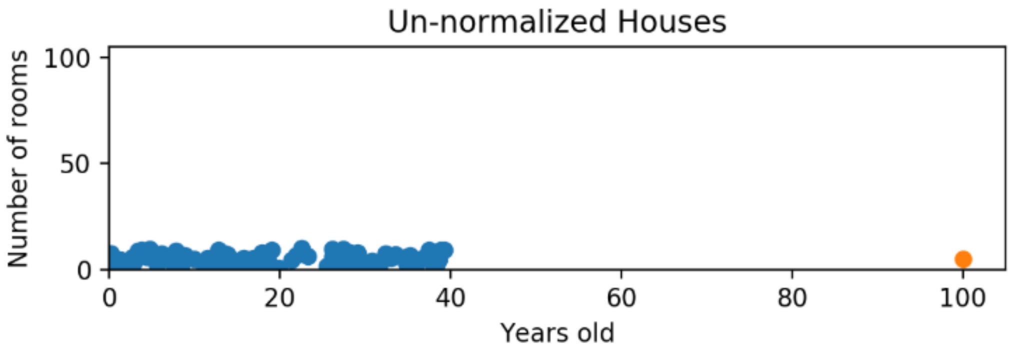
Before normalization, loss is sensitive to changes in model parameters, hard to optimize

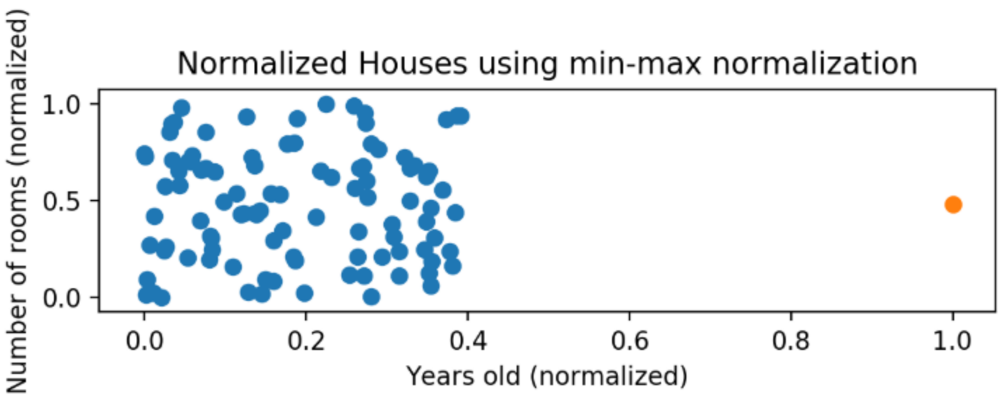


After normalization, loss is less sensitive to changes in model parameters, easier to optimize

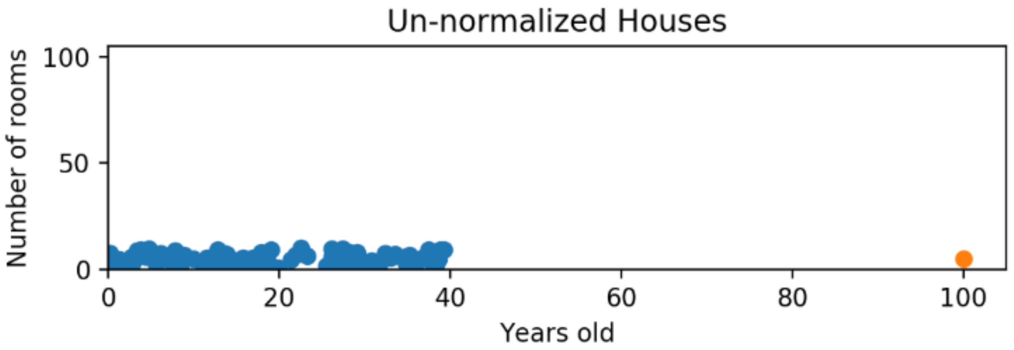
$$\text{Min-Max} = \frac{value - min}{max - min}$$

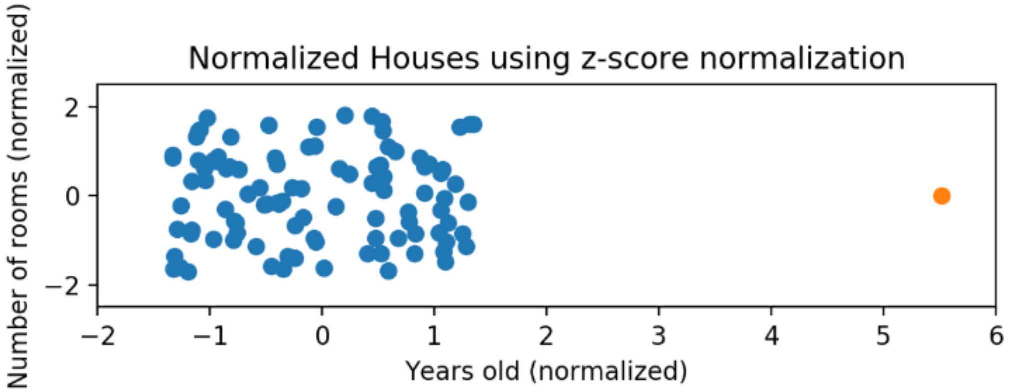
$$\text{Z-score} = \frac{value - mean}{std}$$



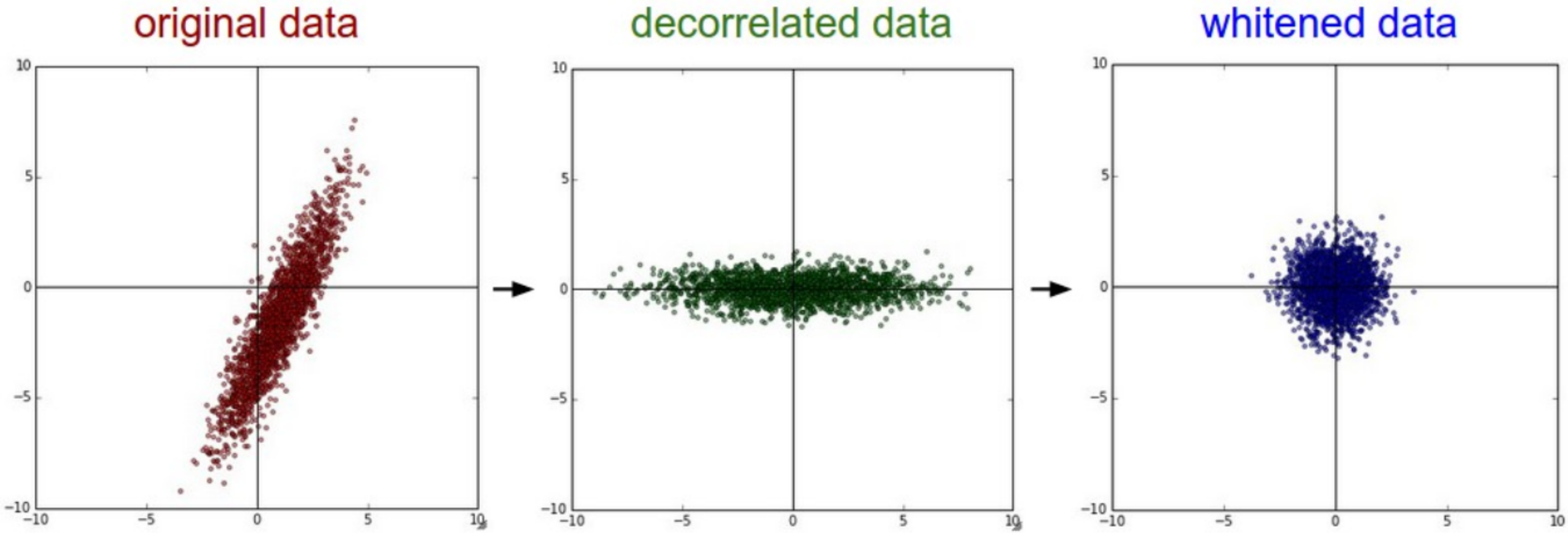


Min-max normalization: Guarantees all features will have the exact same scale but does not handle outliers well





Z-score normalization: Handles outliers, but does not produce normalized data with the *exact* same scale.



PCA / Whitening. **Left**: Original toy, 2-dimensional input data. **Middle**: After performing PCA. The data is centered at zero and then rotated into the eigenbasis of the data covariance matrix. This decorrelates the data (the covariance matrix becomes diagonal). **Right**: Each dimension is additionally scaled by the eigenvalues, transforming the data covariance matrix into the identity matrix. Geometrically, this corresponds to stretching and squeezing the data into an isotropic gaussian blob.

Less used in convolutional neural network

1. Linear regression (5 steps)
2. Multilayer perceptron
3. Convolutional neural network
4. Data preprocessing