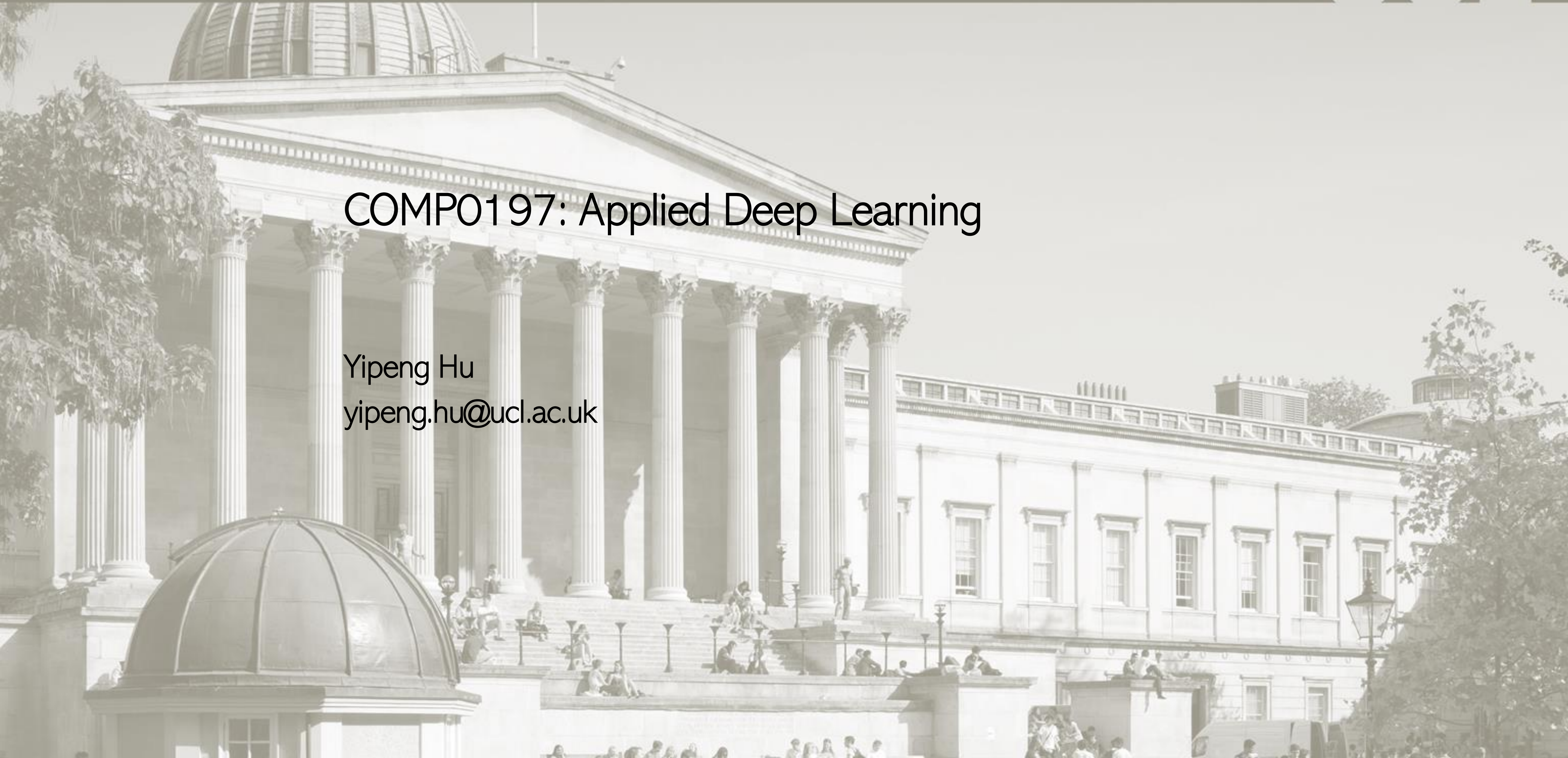
Department of Computer Science

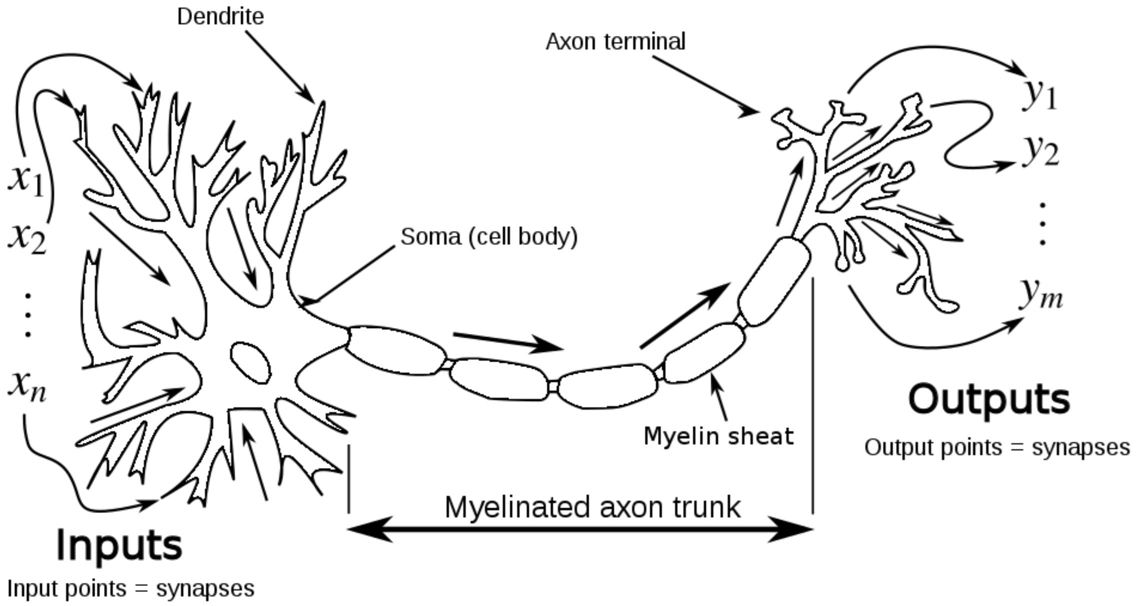


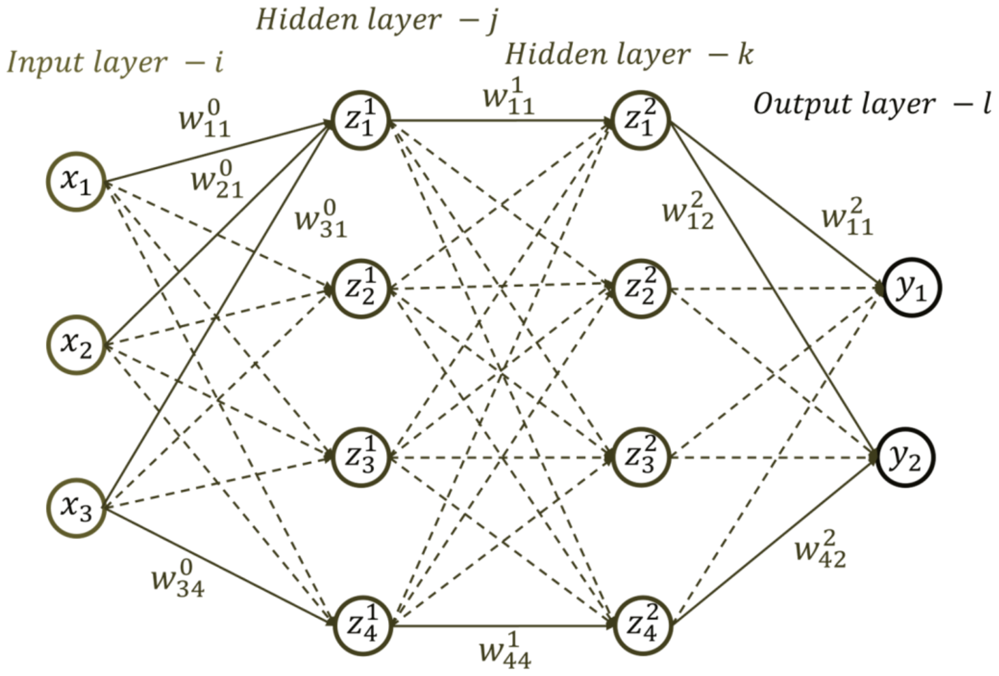
COMP0197: Applied Deep Learning

Yipeng Hu
yipeng.hu@ucl.ac.uk

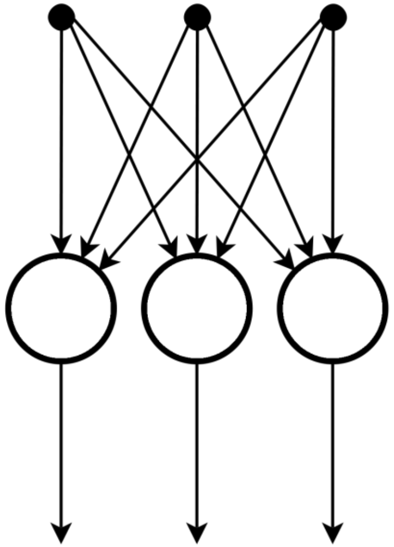
Deep Feedforward Neural Networks

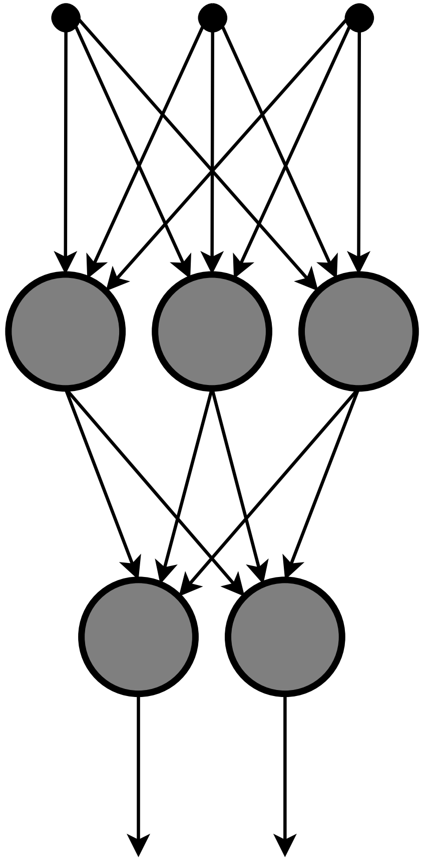
Biological and artificial neurons





Feedforward models



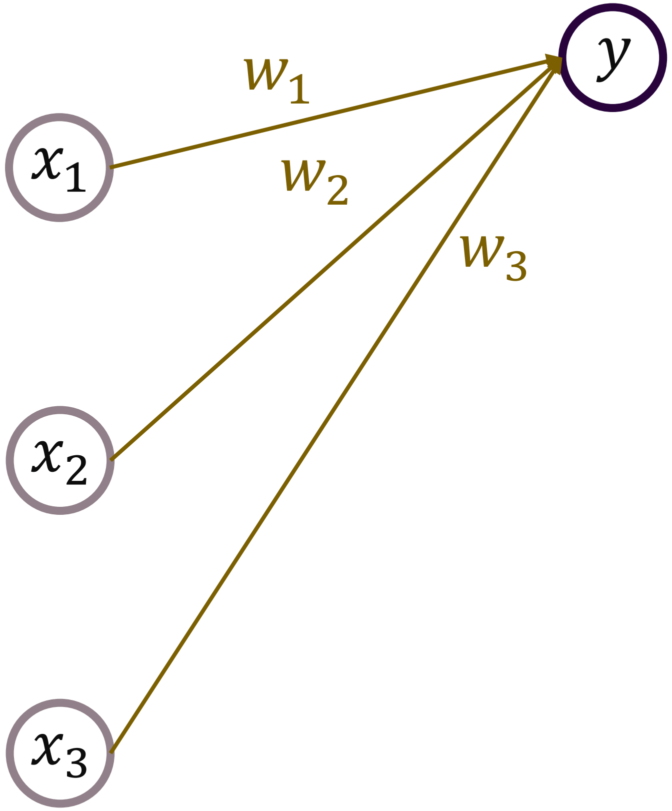


- Multilayer Perceptron (MLP)
- Anatomy of A Layer
- Activation Functions
- Architecture

  Width and Depth | Branching and Joining | Skipping | Sampling | “Ignoring”
- Architecture Search

Deep Feedforward Neural Networks | MLP

Logistic regression



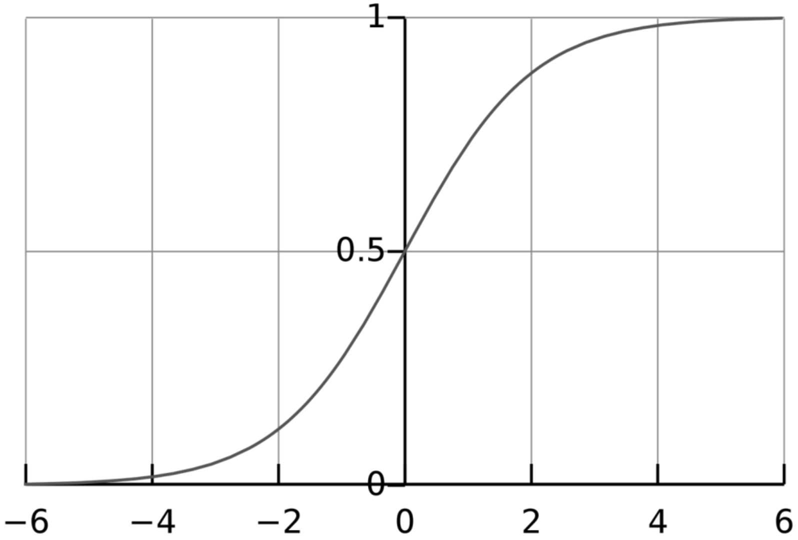
x: Input

y: Output

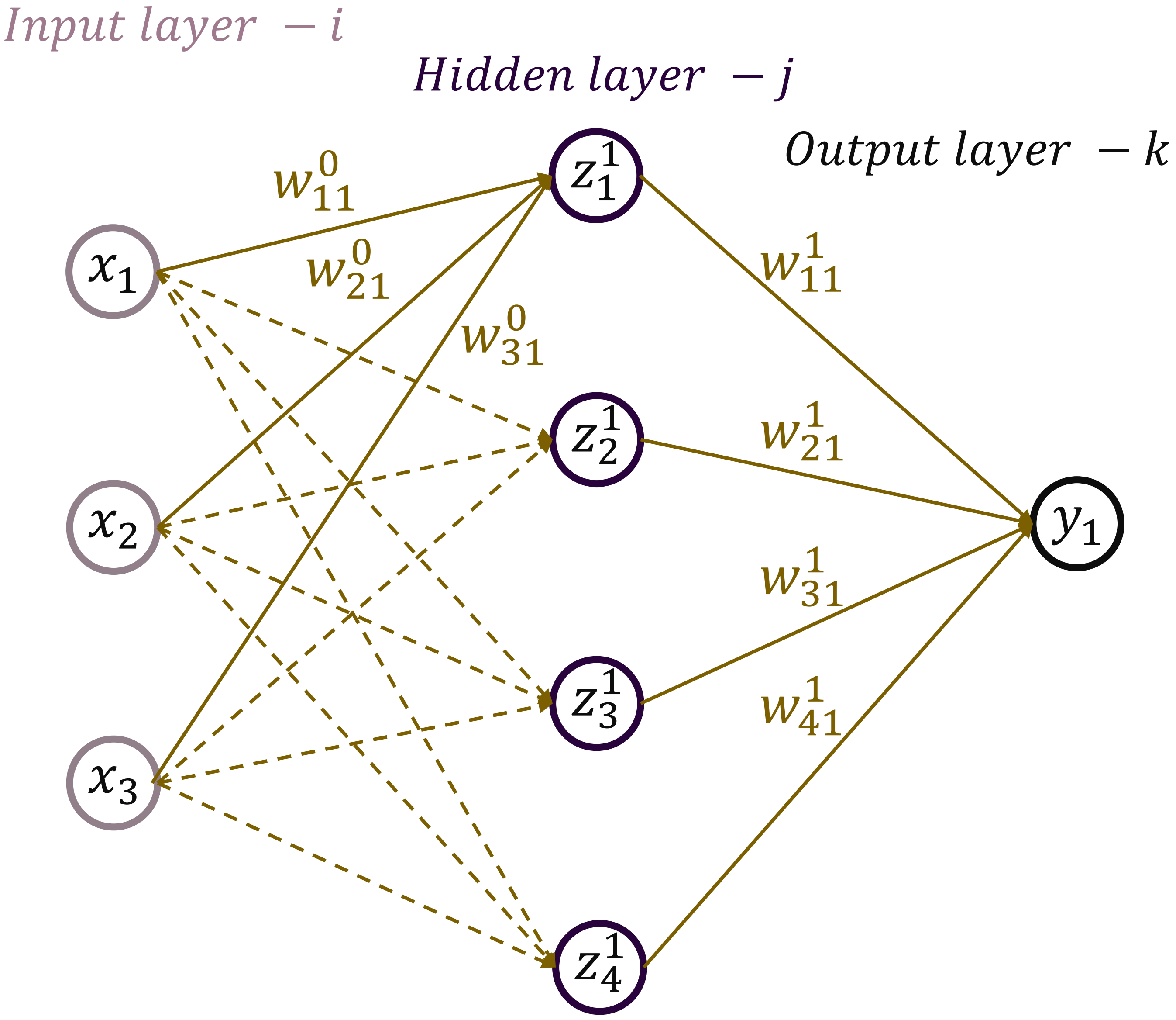
$\mathbf{w}$: Parameters/weights

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

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



Deep Feedforward Neural Networks | MLP



Activation function: $\sigma \rightarrow g$

Input ⟶ H1: i ⟶ j

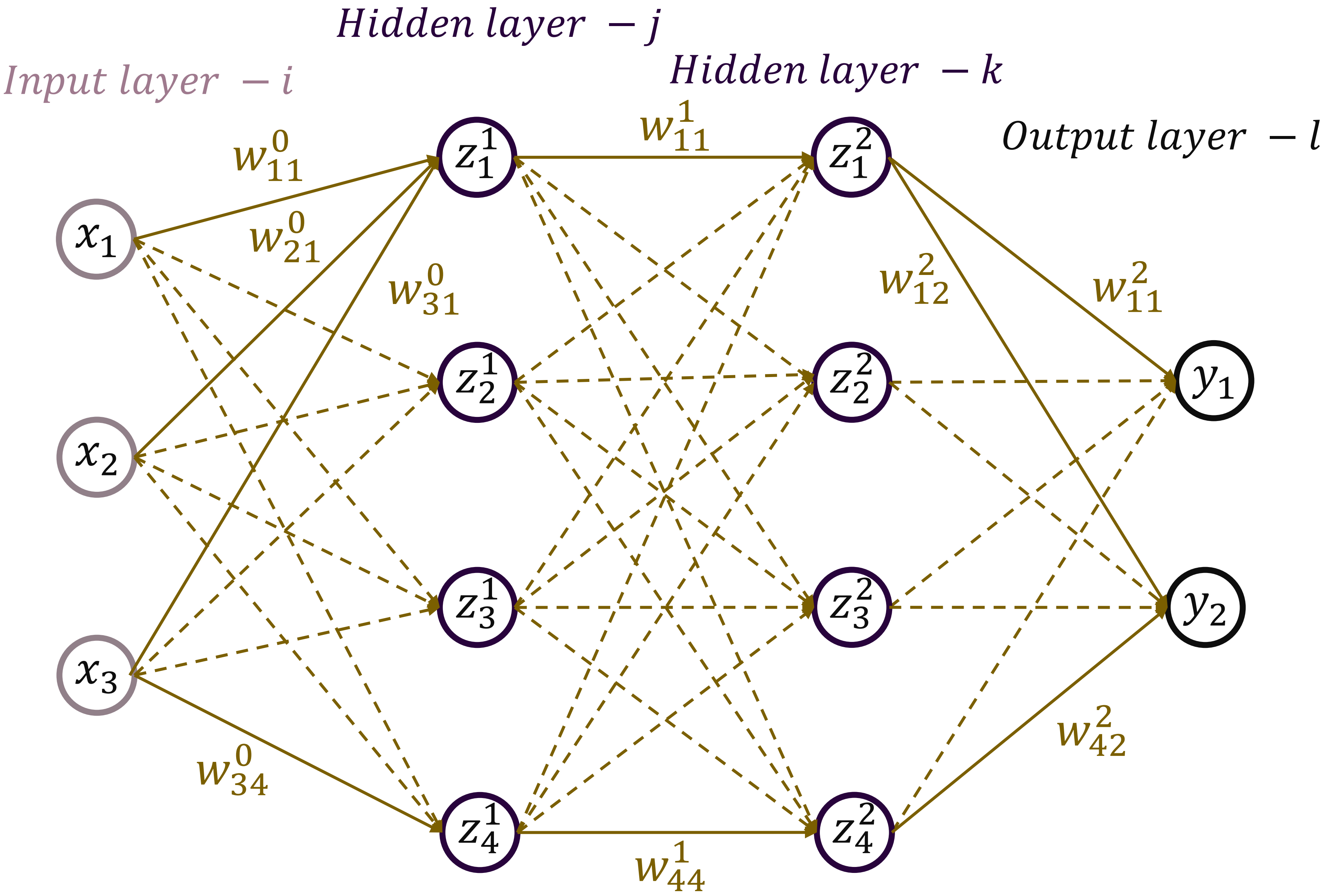
$$z_j^1 = g_j^1\left[\sum_{i=1}^{i=3} x_i w_{ij}^0 + b_j^1\right] = g_j^1\left[\left(\mathbf{w}_j^0\right)^T \boldsymbol{x} + b_j^1\right]$$

H1 ⟶ Output: j ⟶ k

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

where $\mathbf{z}^1 = [z_1^1, z_2^1, z_3^1, z_4^1]^T, k = 1$

Deep Feedforward Neural Networks | MLP



Input layer $-\ i$

Hidden layer $-\ j$

Hidden layer $-\ k$

Output layer $-\ l$

Input → H1: i → j

$$z_j^1 = g_j^1\left[\left(\mathbf{w}_j^0\right)^{\mathrm{T}}\mathbf{x} + b_j^1\right]$$

H1 → H2: j → k

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

H2 → Output: k → l

$$y_l = g_l\left[\left(\mathbf{w}_l^2\right)^{\mathrm{T}}\mathbf{z}^2 + b_l^2\right]$$

Deep Feedforward Neural Networks | Anatomy of A Layer

Deep Feedforward Neural Networks | Anatomy of A Layer

A unit: $h = g(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$

h: Hidden layer & output feature (vectors)

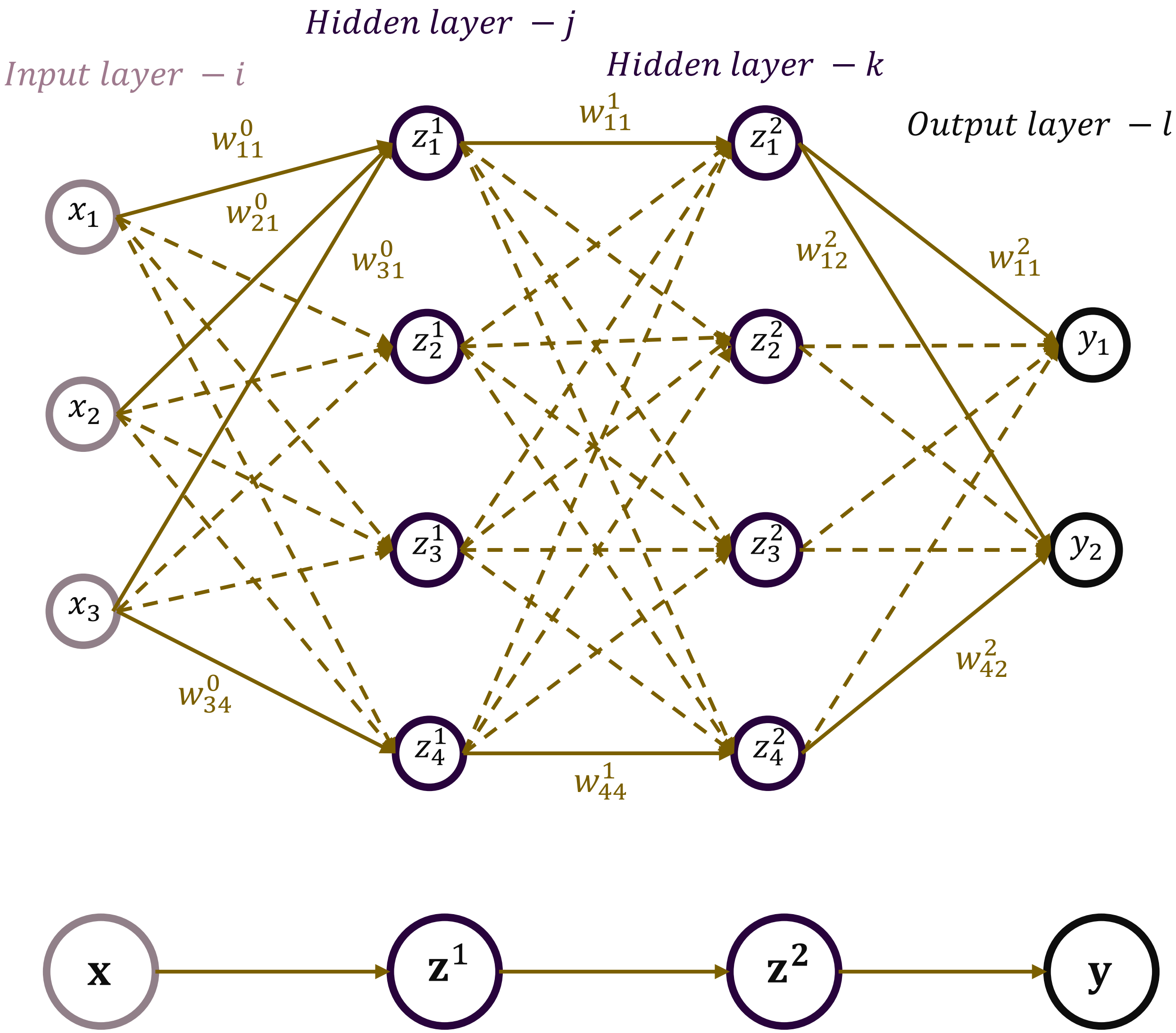
$\mathbf{x}$: Input feature vector

$\mathbf{w}$: Weights – network parameters

b: Bias

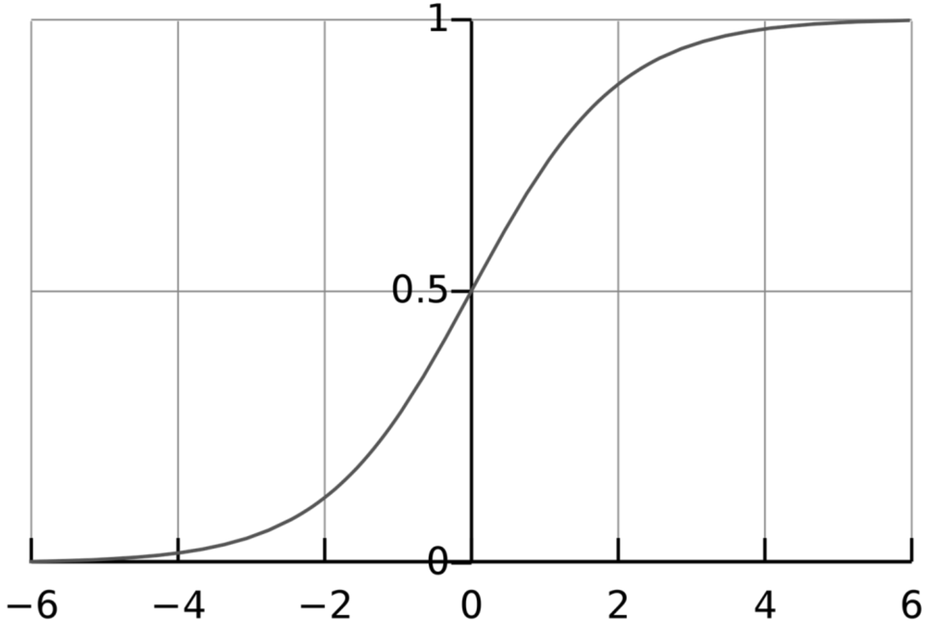
$g(\cdot)$: Activation function

A layer is a collection of units from the same input



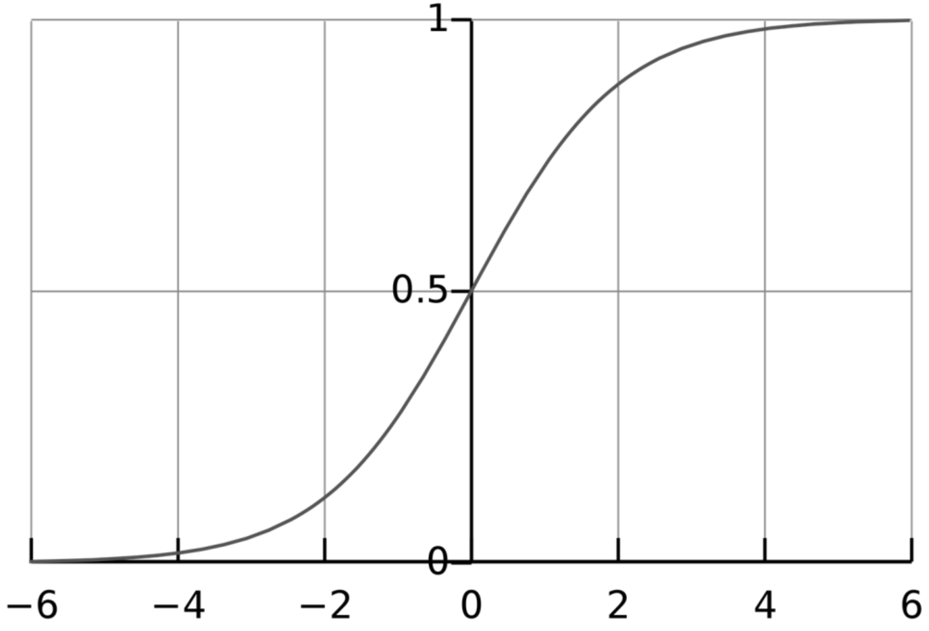
Deep Feedforward Neural Networks | Activation Functions

Nonlinearity?



Nonlinearity

$$
\begin{aligned}
y &= \mathbf{w}_j^{\mathrm{T}}\left(\mathbf{w}_i^{\mathrm{T}}\mathbf{x} + \mathbf{b}_i\right) + \mathbf{b}_j \\
&= \mathbf{w}_j^{\mathrm{T}}\mathbf{w}_i^{\mathrm{T}}\mathbf{x} + \left(\mathbf{w}_j^{\mathrm{T}}\mathbf{b}_i + \mathbf{b}_j\right) \\
&= \widetilde{\mathbf{w}}^{\mathrm{T}}\mathbf{x} + \tilde{\mathbf{b}}
\end{aligned}
$$

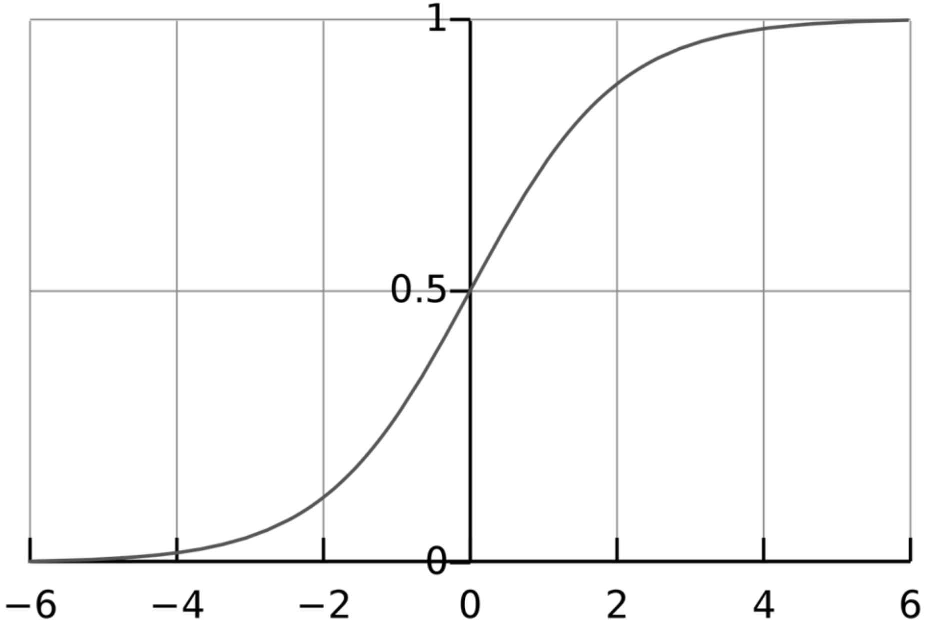


Nonlinearity

Range $[\mathbf{0}, \mathbf{1}]$

The squashing effect

A “unit” usually has one activation value



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

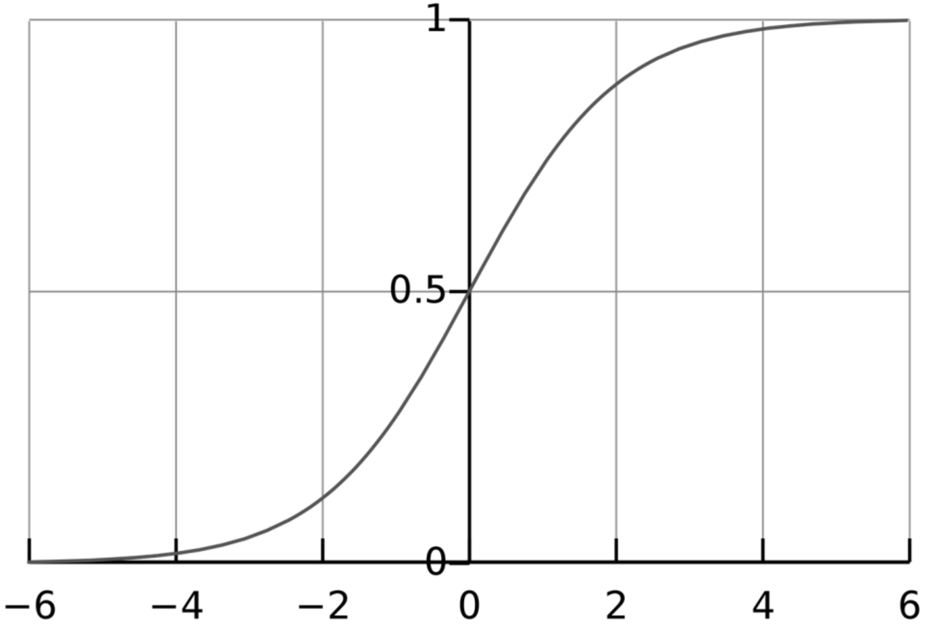
The sigmoid function

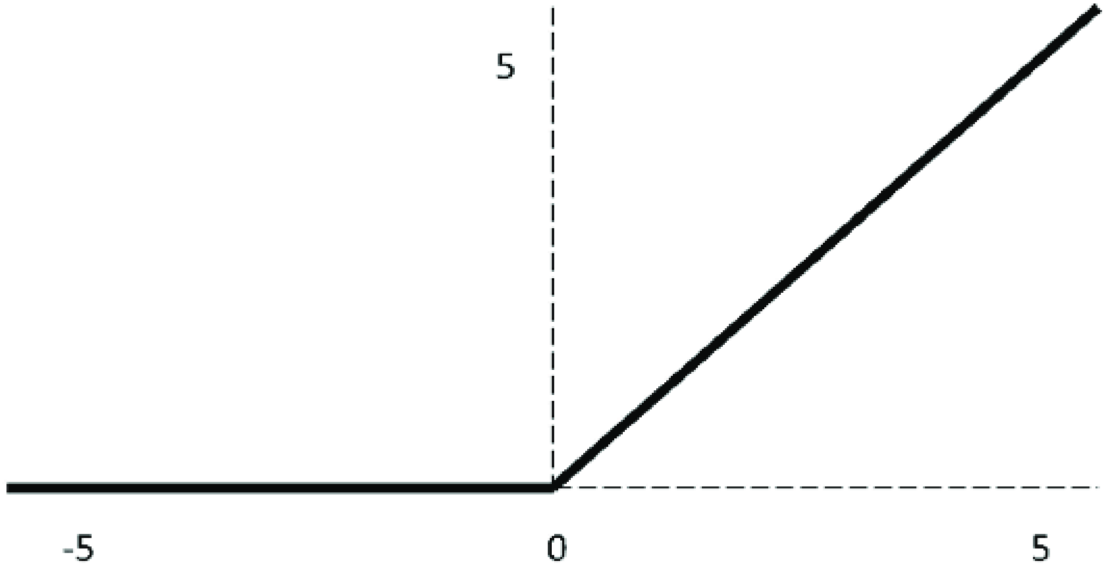
For hidden units

Logistic sigmoid $\mathrm{g}(z) = \sigma(z) = \frac{1}{1+e^{-z}}$

Hyperbolic tangent $\mathrm{g}(z) = \tanh(z) = 2\sigma(2z) - 1$

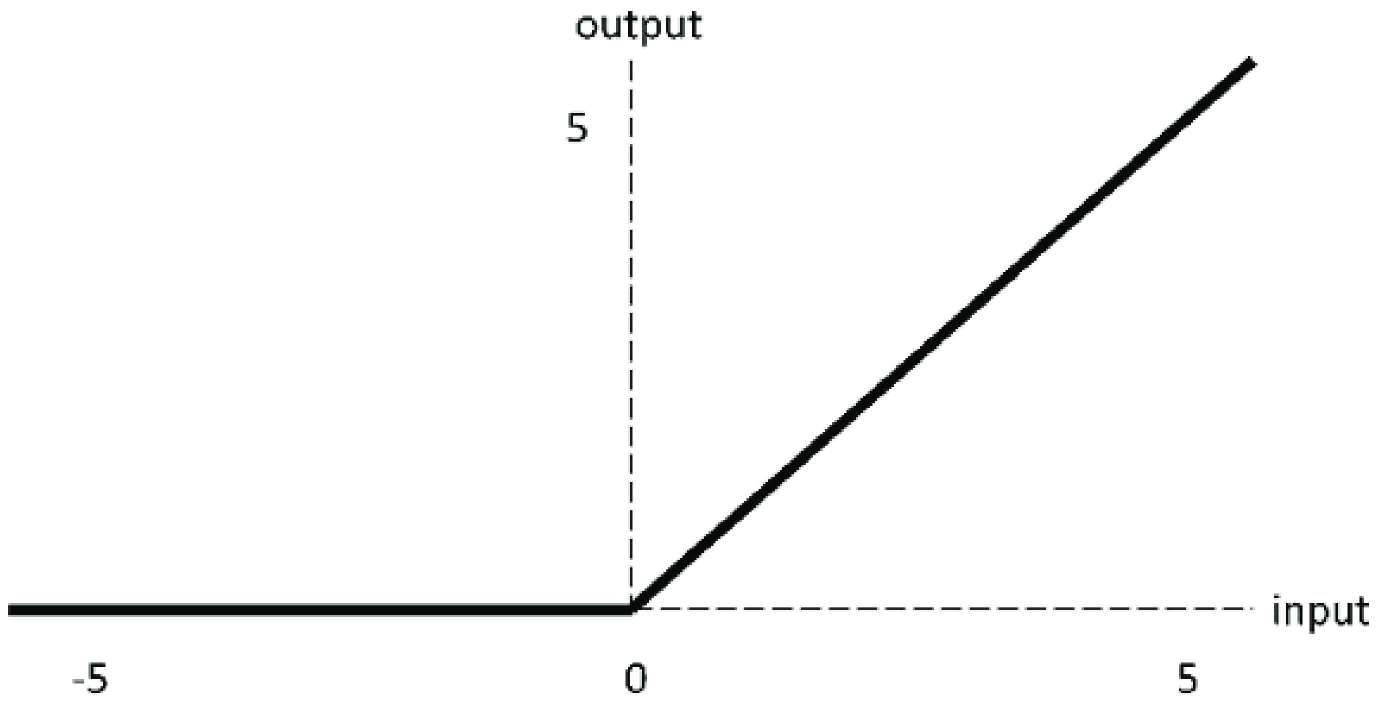
Rectified linear unit (ReLU) $\mathrm{g}(z) = \max\{0, z\}$





For hidden units

Rectified linear unit (ReLU) $g(z) = \max\{0, z\}$

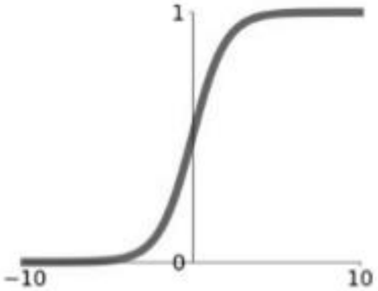


Deep Feedforward Neural Networks | Activation Functions

For hidden units

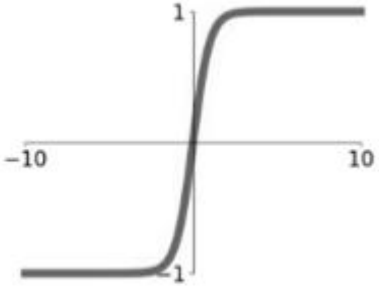
Sigmoid

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



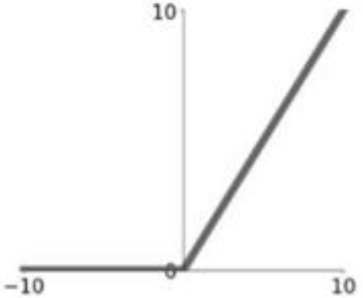
tanh

$\tanh(x)$



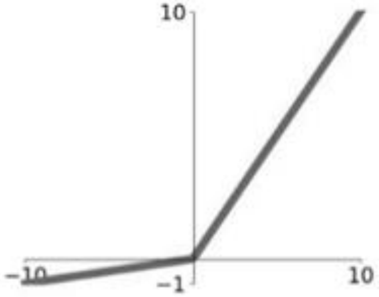
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

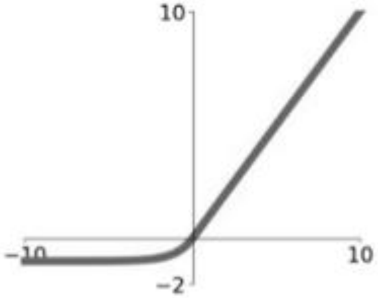


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

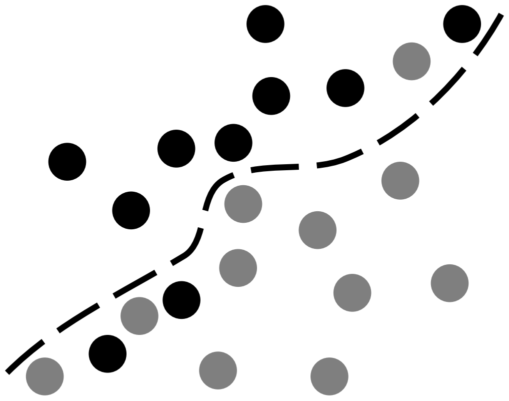
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



For output units

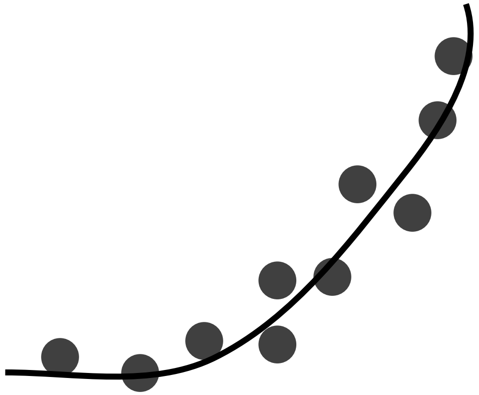
Bernoulli output (logistic sigmoid): $y = g(z) = \frac{1}{1+e^{-z}}$



Multinoulli output (softmax): $y_c = g(z_c) = \frac{\exp(z_c)}{\sum_{j=1}^{C} \exp(z_j)}$

$$\rightarrow \begin{cases} \sum_{j=1}^{C} y_c = 1 \\ 0 \leq y_c \leq 1 \end{cases}$$

Gaussian output



Other activation functions

Permutation-invariant, e.g. max, mean, min…

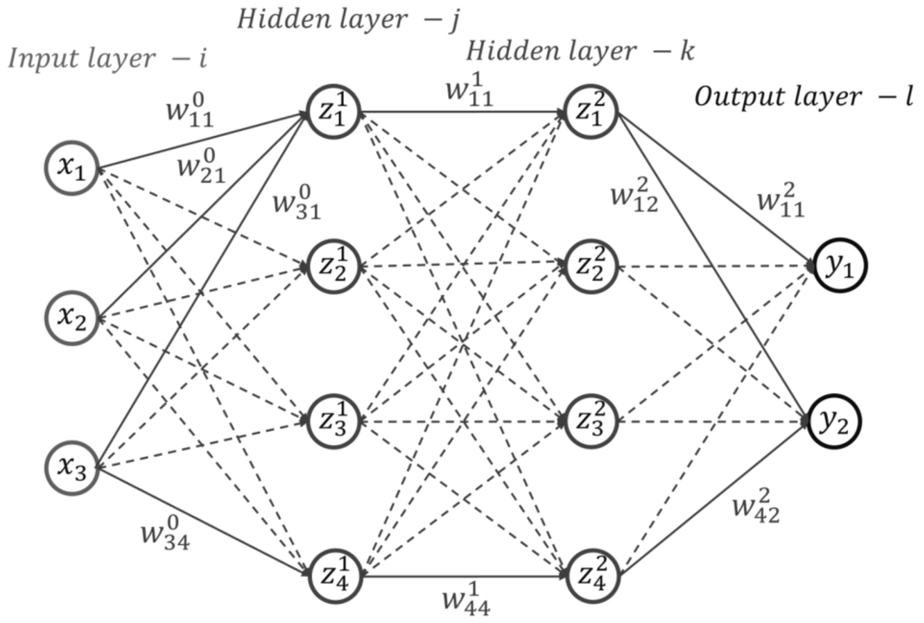
Known specific range and distribution

e.g. displacement/velocity (optical flow, registration), function parameters (camera position, pose)

Deep Feedforward Neural Networks | Architecture

Deep Feedforward Neural Networks | Architecture

Width and depth



How many layers?

Universal approximation theorem

A sufficiently-wide network approximates any “practically useful” functions

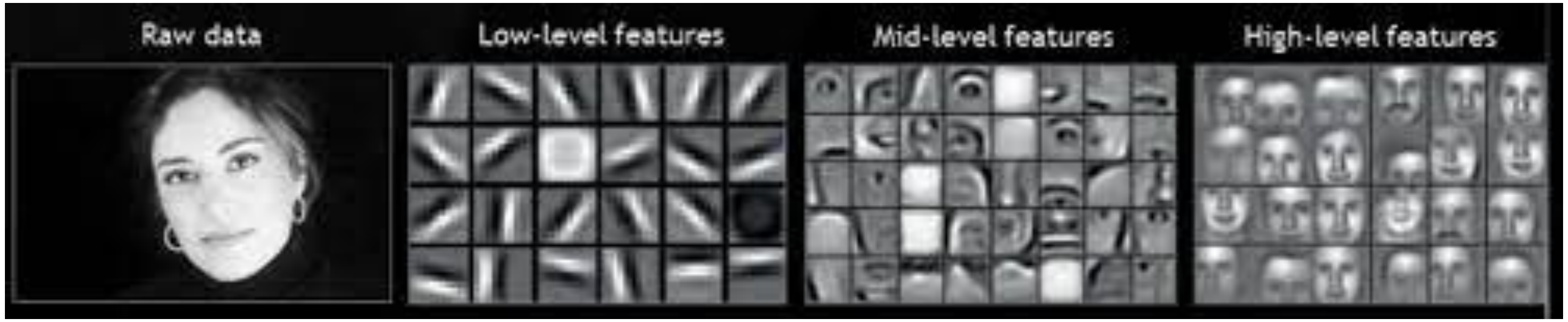
The wider the better?

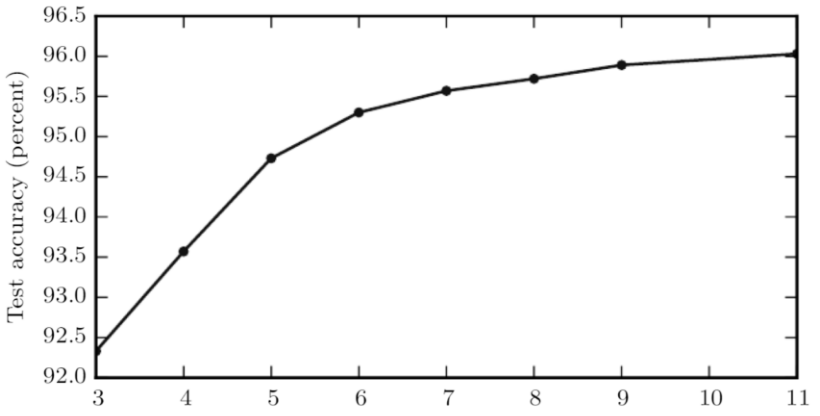
The deeper the better?

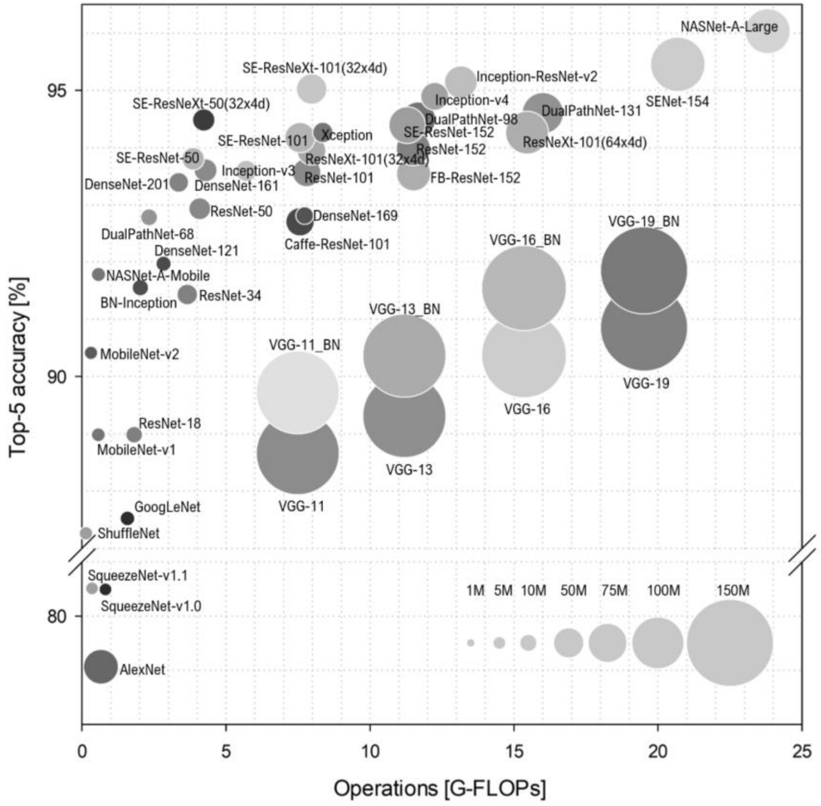
Width and depth

Why deep?

Hierarchical representation learning, Empirical results



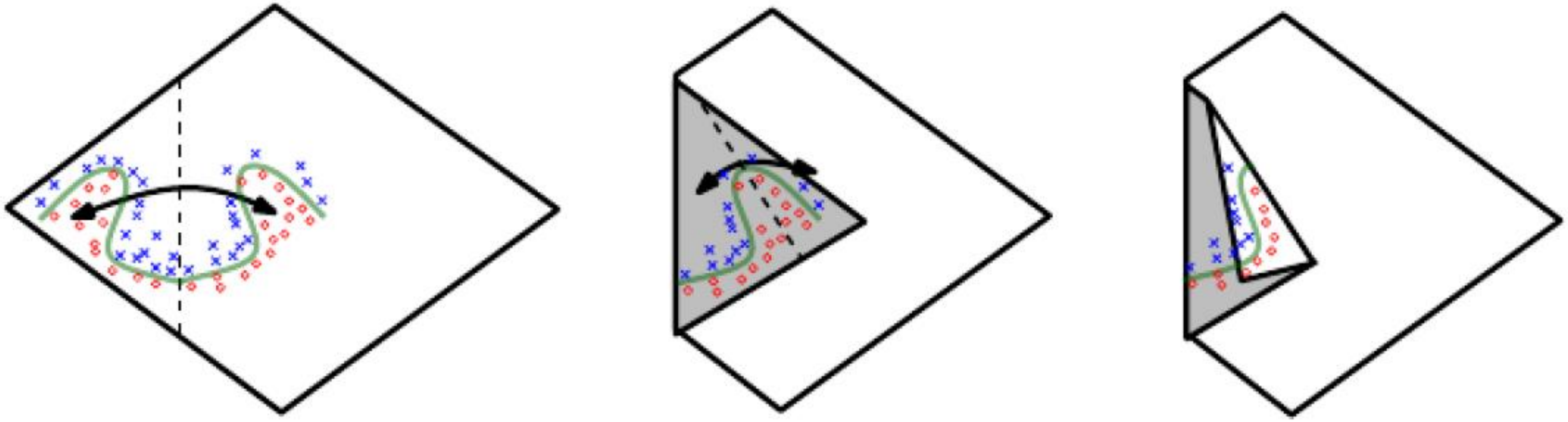




Width and depth

Why deep?

Hierarchical representation learning, Empirical results, Efficiency



Deep Feedforward Neural Networks | Architecture

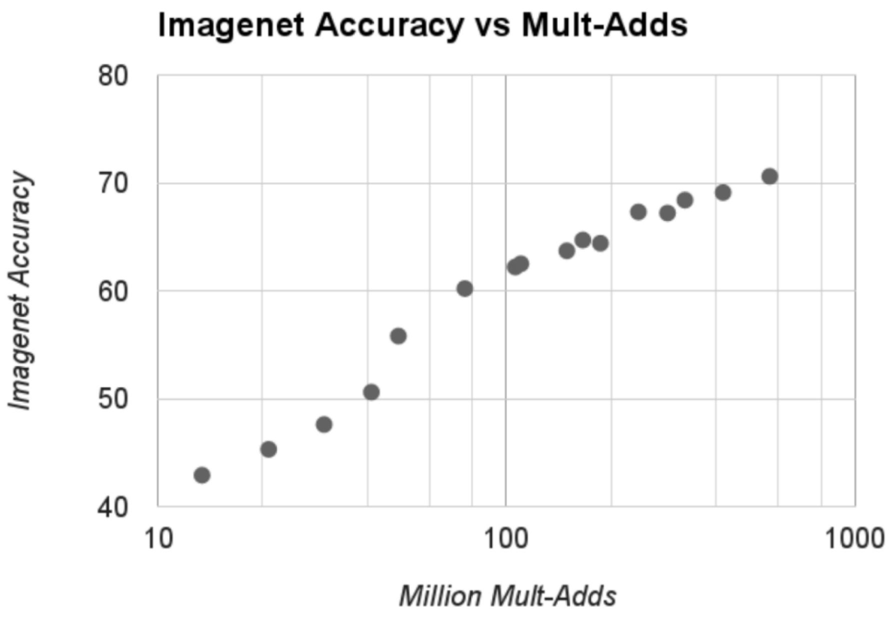
Size

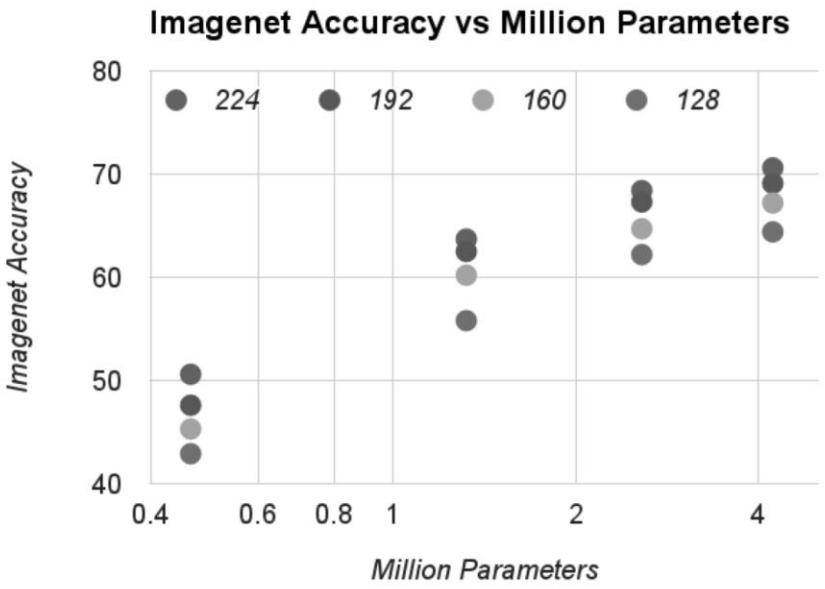
Number of parameters vs storage size ?

Memory consumption ?

Trade-off between accuracy and size

- "MobileNet" & "EfficientNet"*





Branching and joining

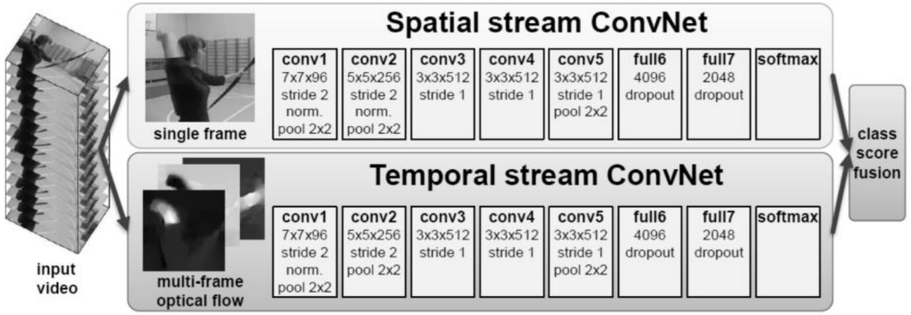
- Multi-stream

- Multi-task*

- “Inception”

- Different types of input, output/loss, and/or processing

- The same activation or not, i.e. a sparse unit or a layer



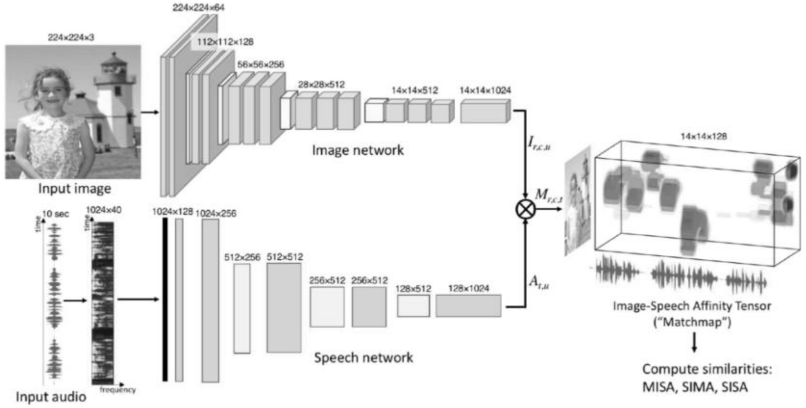
Spatial stream ConvNet

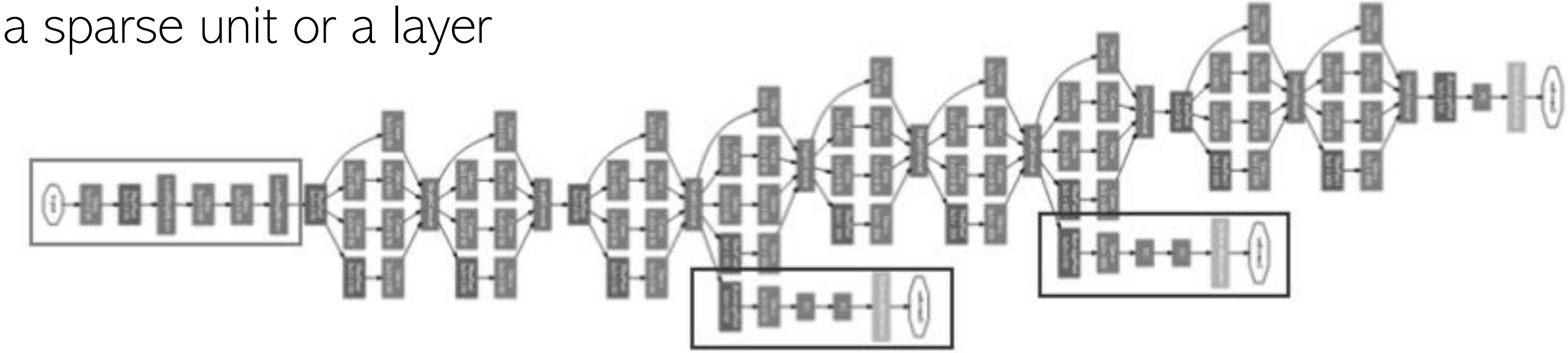
conv1	conv2	conv3	conv4	conv5	full6	full7	softmax
7x7x96 stride 2 norm. pool 2x2	5x5x256 stride 2 norm. pool 2x2	3x3x512 stride 1	3x3x512 stride 1	3x3x512 stride 1 pool 2x2	4096 dropout	2048 dropout	

Temporal stream ConvNet

conv1	conv2	conv3	conv4	conv5	full6	full7	softmax
7x7x96 stride 2 norm. pool 2x2	5x5x256 stride 2 pool 2x2	3x3x512 stride 1	3x3x512 stride 1	3x3x512 stride 1 pool 2x2	4096 dropout	2048 dropout	

Figure 1: **Two-stream architecture for video classification.**





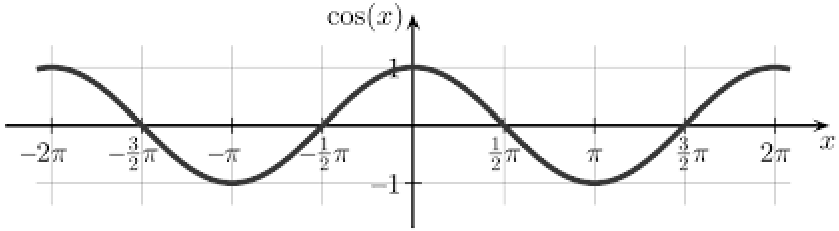
Branching and joining

- Implementing branching with two sets of (non-)shared weights:

$$h_1 = g(\mathbf{w}_1^\mathrm{T}\mathbf{x} + b_1) \text{ and } h_2 = g(\mathbf{w}_2^\mathrm{T}\mathbf{x} + b_2)$$

- Implementing joining:

Dot product: $\mathbf{h}_1 \cdot \mathbf{h}_2 = |\mathbf{h}_1||\mathbf{h}_2| \cos\theta$



Concatenation vs. summation

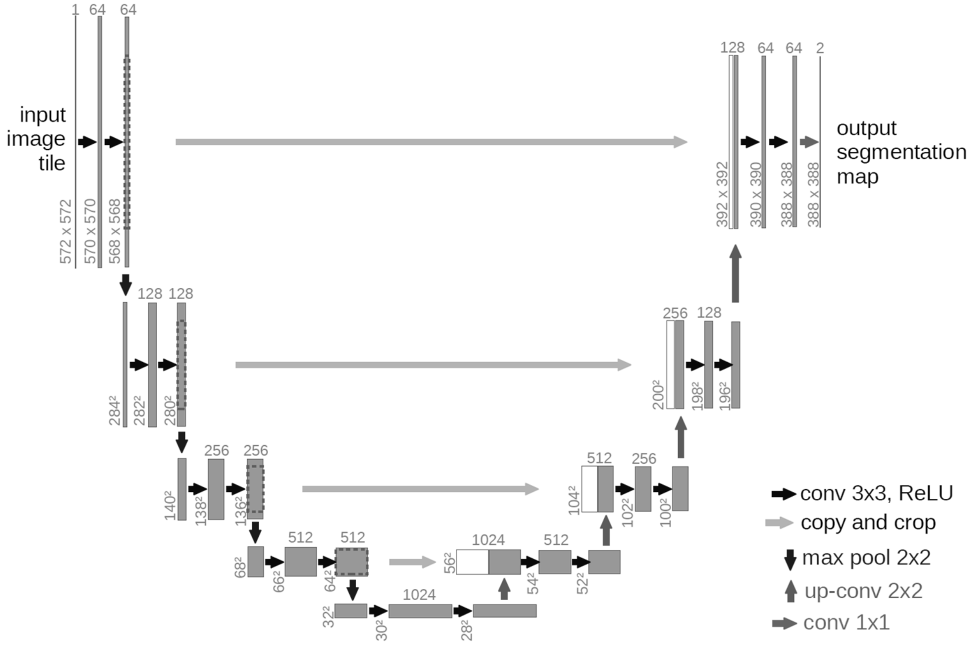
$$(\mathbf{h}_1, \mathbf{h}_2) \rightarrow [\mathbf{h}_1^T, \mathbf{h}_2^T]^T$$

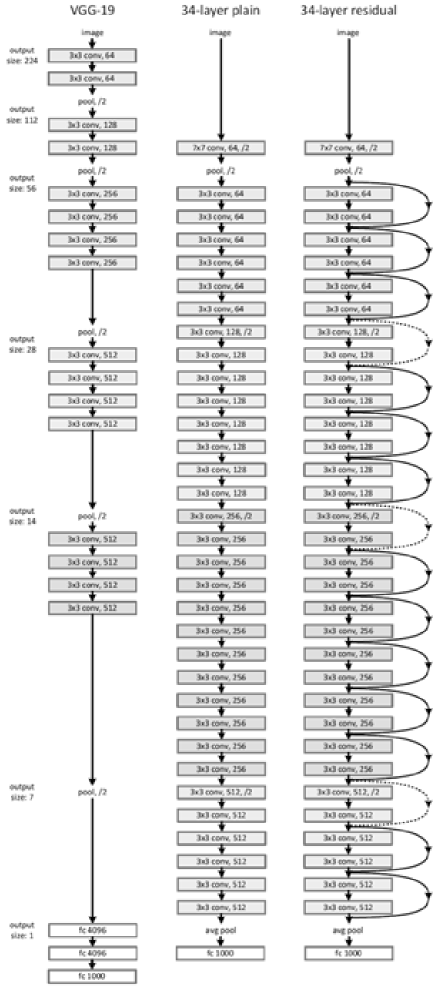
$$(\mathbf{h}_1, \mathbf{h}_2) \rightarrow \mathbf{h}_1 + \mathbf{h}_2$$

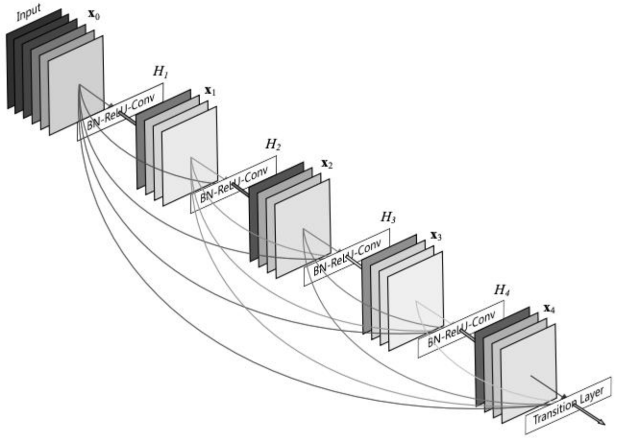
Skipping

- Shortcuts, skip layers, residual connections

- “U-Net”*





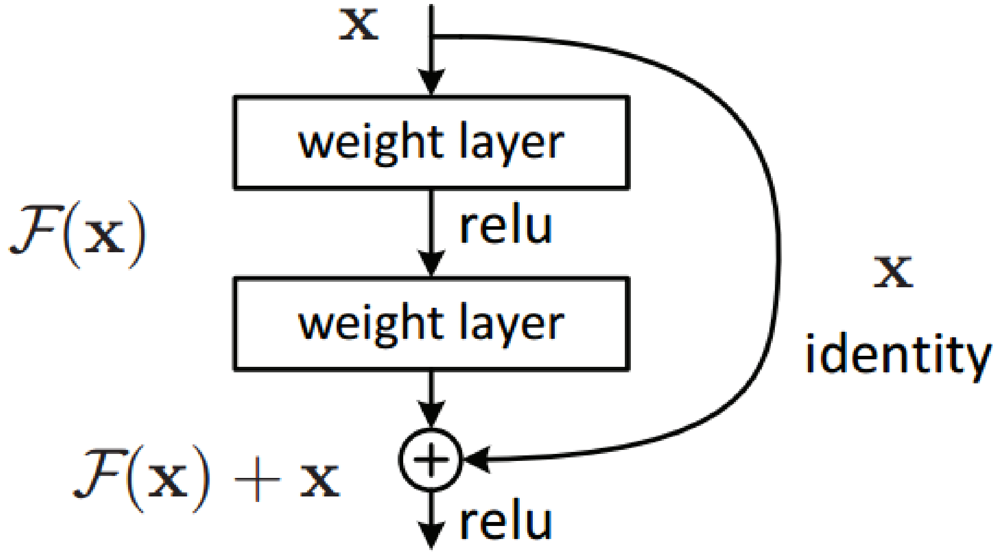


Skipping

- Shortcuts, skip layers, residual connections

- “U-Net”*

- “ResNet”*

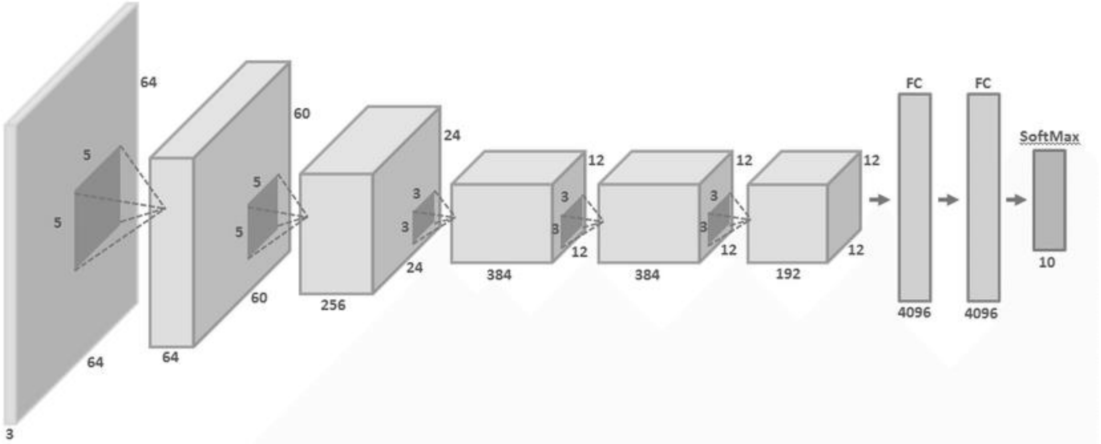


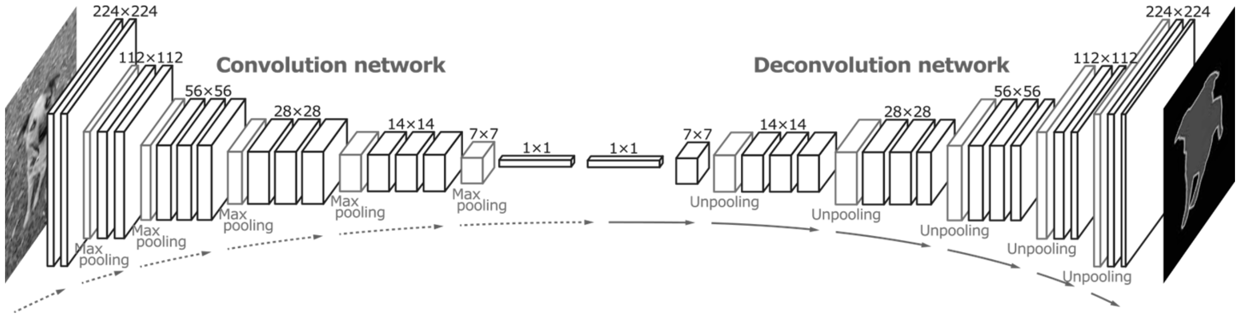
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

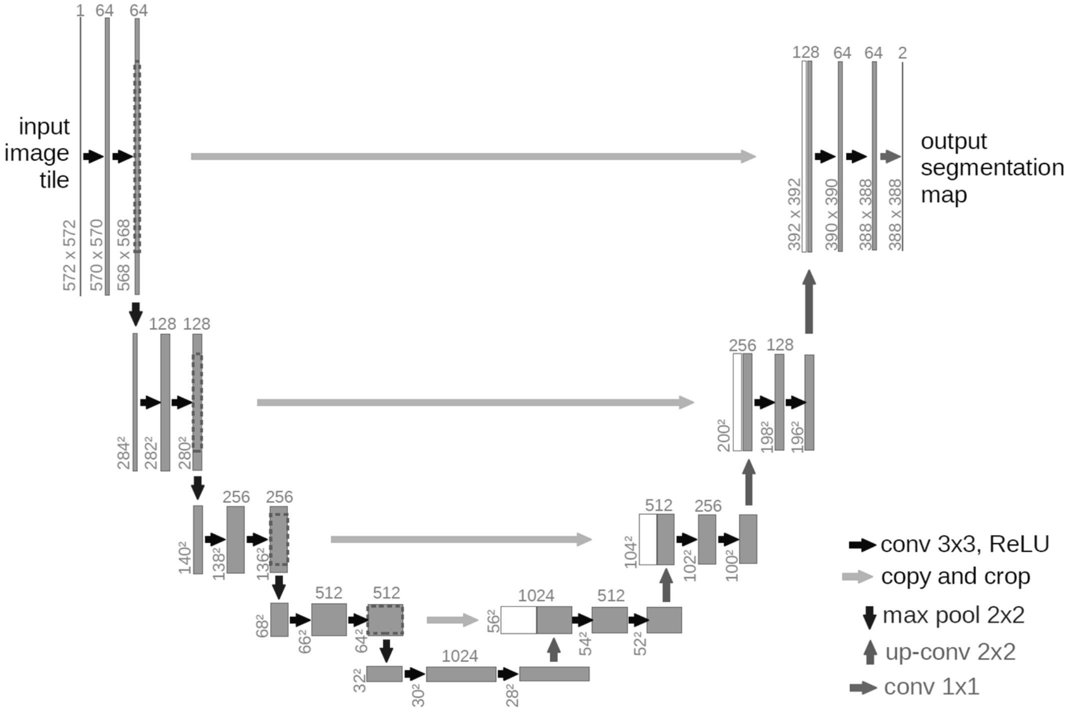
Deep Feedforward Neural Networks | Architecture

Sampling

- Down-sampling for encoding ?
- “AlexNet” & “VGG”
- Up-sampling for decoding ?
- “U-Net” & “Autoencoder”
- Sampling for convolution*







Sampling

- Implement vector down-sampling: averaging, matrix multiplication

- Implement vector up-sampling: matrix multiplication, interpolation ?

$$h_n = g\left(\mathbf{w}_n^{\mathrm{T}}\mathbf{x} + b_n\right)$$

$$\begin{bmatrix} h_1 \\ \vdots \\ h_N \end{bmatrix} = g\left(\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}}\mathbf{x} + b_1 \\ \vdots \\ \mathbf{w}_N^{\mathrm{T}}\mathbf{x} + b_N \end{bmatrix}\right) = g([\mathbf{w}_1 \quad \dots \quad \mathbf{w}_N]^T\mathbf{x} + [b_1 \quad \dots \quad b_N]^T)$$

$$\mathbf{h}^{(N\times1)} = \mathbf{W}^{(N\times M)}\mathbf{x}^{(M\times1)} + \mathbf{b}^{(N\times1)}$$

- Implement feature map sampling*

“Ignoring”

$$f(\{x_1, x_2, x_3, \ldots x_n\}) \approx g(\{h(x_1), h(x_2), h(x_3), \ldots h(x_n)\})$$

Deep Feedforward Neural Networks | Architecture

“Ignoring”

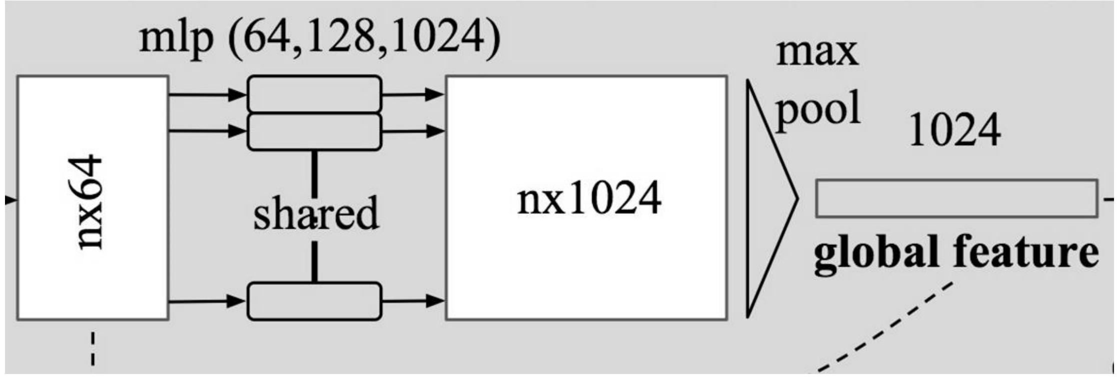
- Permutation-invariant

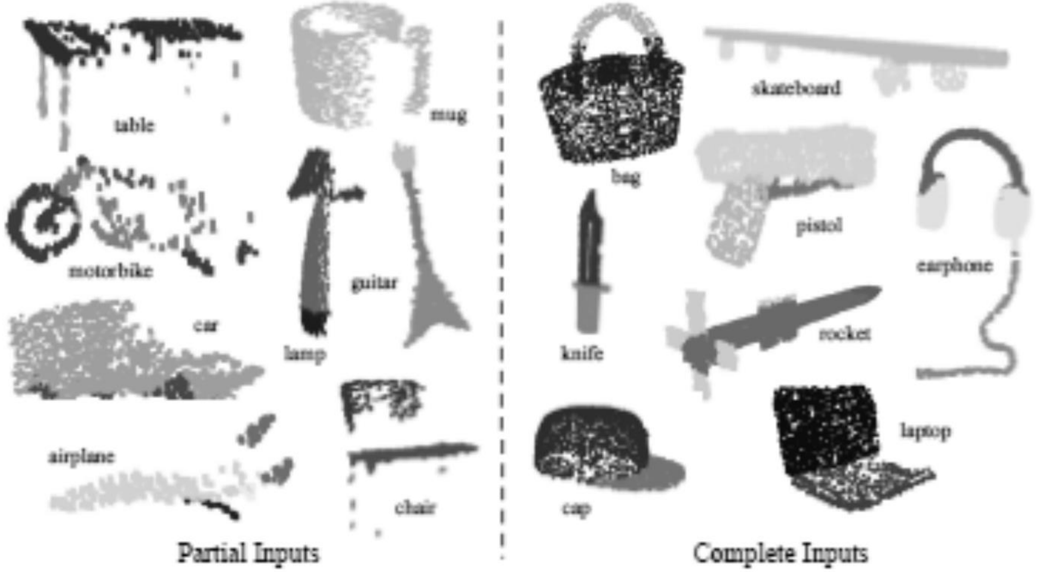
$$f(\{x_1, x_2, x_3, \dots x_n\}) \approx g(\{h(x_1), h(x_2), h(x_3), \dots h(x_n)\})$$

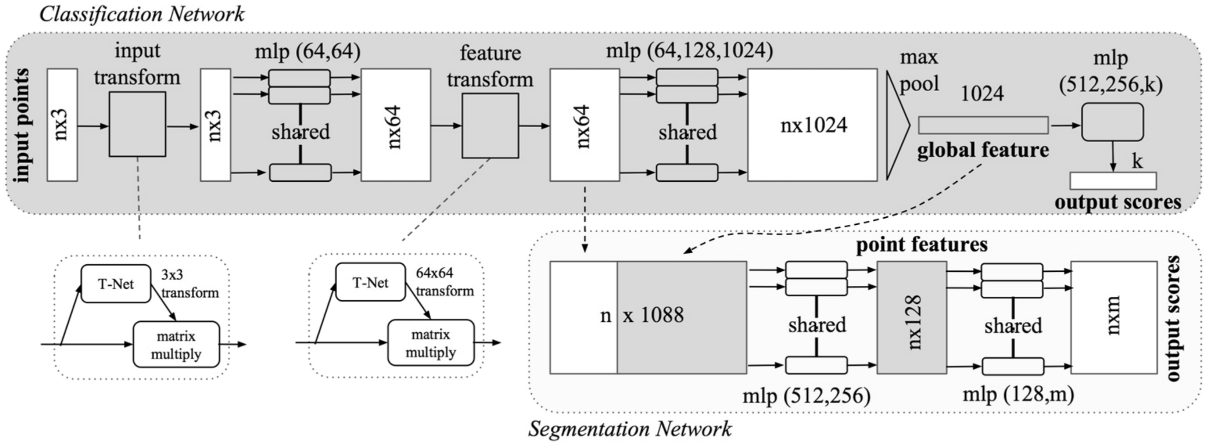
- “PointNet”

max, min, mean…

- Translation-invariant* (pooling, convolution)







Deep Feedforward Neural Networks | Architecture Search

Hyperparameters

- Predefined
- Might not be optimisable or should not be optimised?
- e.g. degree of polynomial, NN architecture choice (width, depth, branches…)

Hyperparameter tuning in AutoML

Hyperparameter search

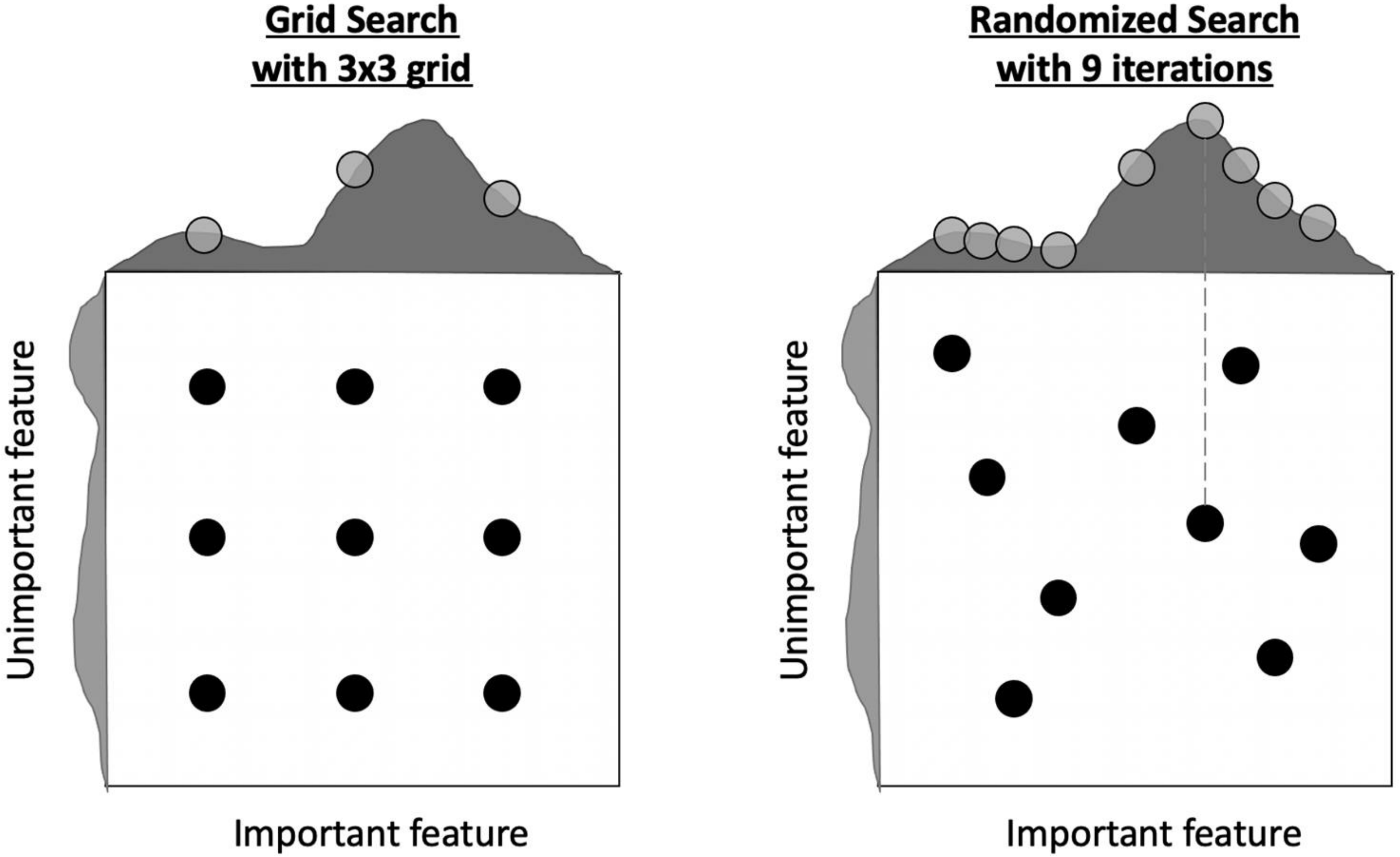
Training-validation-test 3-way split

Grid search vs. random search

As an optimisation,

e.g. evolution algorithms

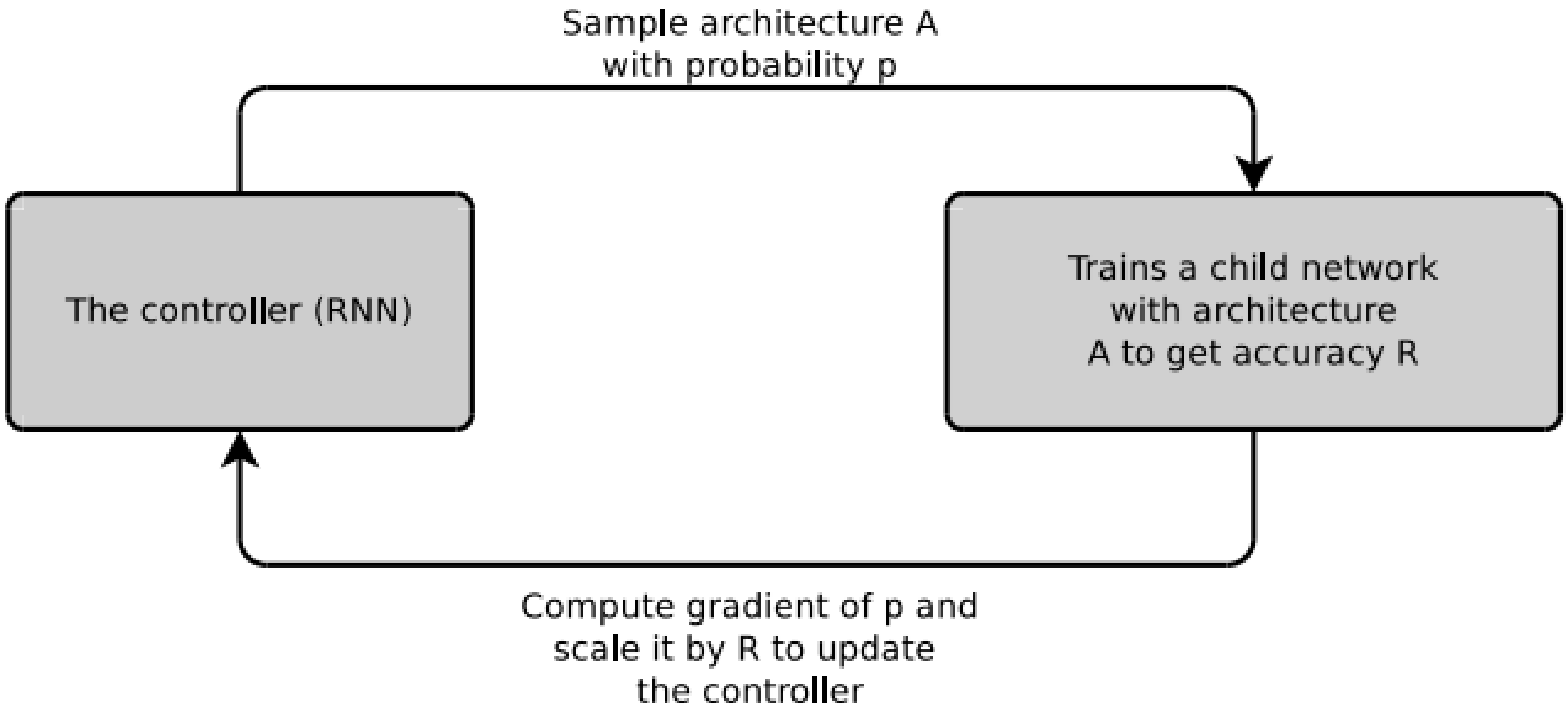
- "EfficientNet"*



Grid Search with 3x3 grid

Randomized Search with 9 iterations

Meta-learning | reinforcement learning



Sample architecture A
with probability p

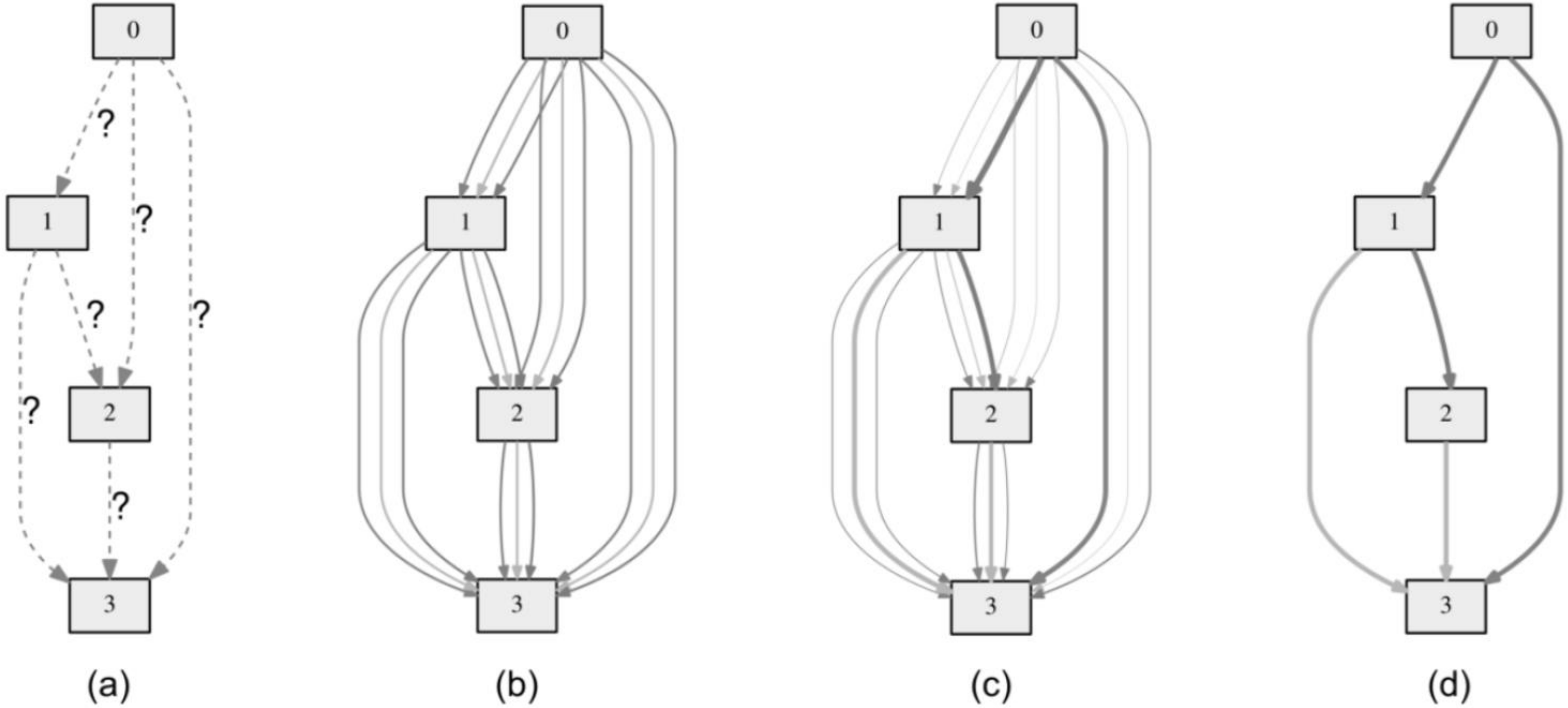
The controller (RNN)

Trains a child network
with architecture
A to get accuracy R

Compute gradient of p and
scale it by R to update
the controller

Meta-learning | differentiable architecture search

- “DARTS”



Deep Feedforward Neural Networks

- Multilayer Perceptron (MLP)
- Anatomy of A Layer
- Activation Functions
- Architecture

  Width and Depth | Branching and Joining | Skipping | Sampling | “Ignoring”

- Architecture Search

Using a fully-connected network for the “image classification” tutorial