

# Deep Learning for Intelligent Signal Processing

Monday

Deep Learning Summer School 2019

Prof. mult. Dr. habil.

**Björn W. Schuller**

MD, CSO, EiC, FIEEE

Imperial College  
London



**UNIA** Universität  
Augsburg  
University



 audEERING  
intelligent Audio Engineering

- **Challenges**

- Unknown pre-processing
- Unknown signals
- Raw data hardly accessible
- Short life-time
- Little labelled data
- Efficient processing

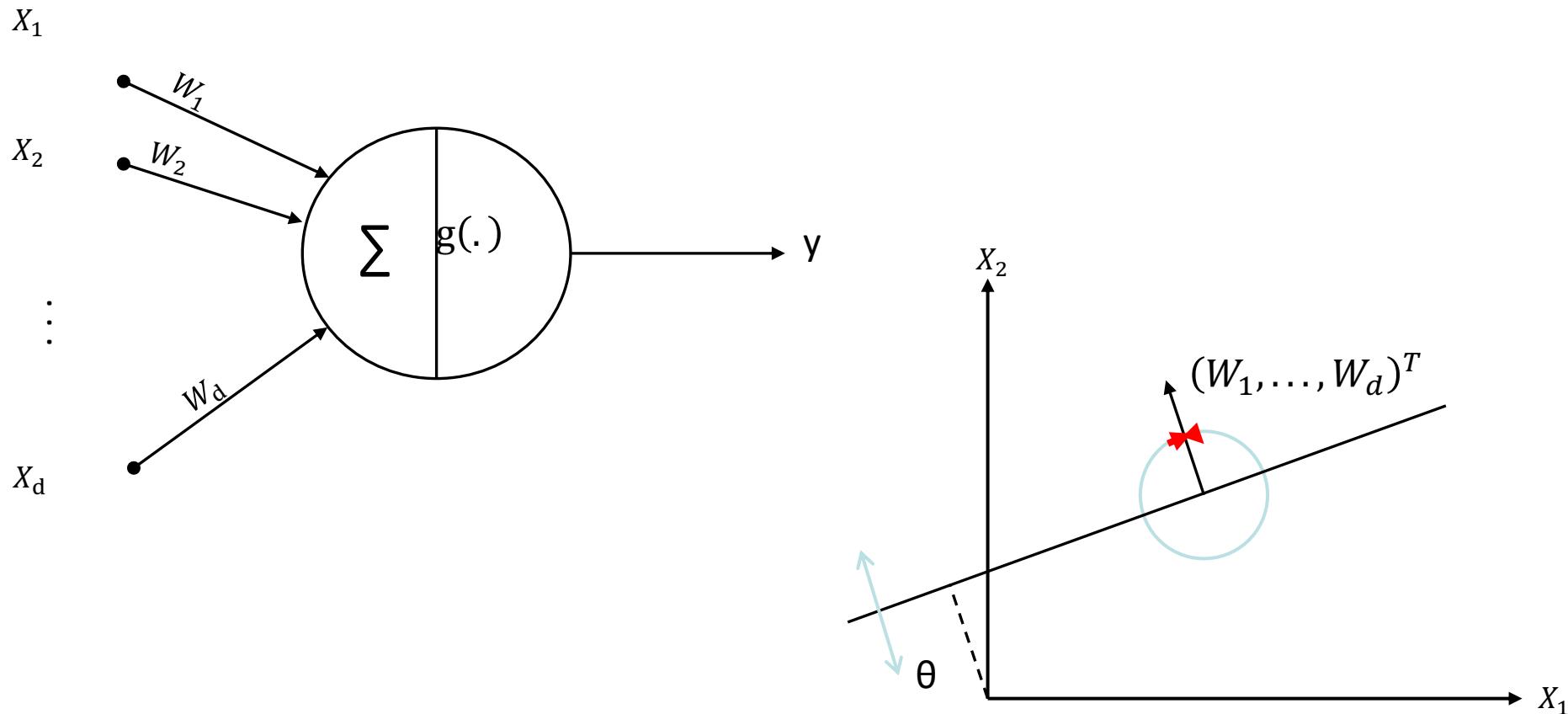
- **Content**
  - CNNs
  - RNNs w/ LSTM/GRU and CTC
  - Attention
  - GANs
  - Reinforcement Learning for Auto ML
  - Efficiency
  
- **Auto DL from raw data in efficient ways**

# Neural Networks.



- 1943 McCulloch+Pitts: 1st formal neuron model  
→ Computation of arbitrary arithmetic / logic functions
- 1949 Hebb: synaptic connection between 2 neurons is enhanced by frequent activation (Hebb rule)
- 1958 Rosenblatt: 1st Neurocomputer-Perceptron (mechanical)
- 1960 Widrow+Hoff: *Least mean square* Algorithm  
+ description of *adaptive linear neuron* (*Adaline*)
- 1965 Vakhnenko+Lapa: 1st „deep Net“
- 1969 Minsky+Papert: Limits of multilayer perceptrons → stagnation
- 1979 Fukushima: convolutional nets (CNNs)
- 1986 Rumelhart+Hinton+Williams: Backpropagation in n-layer models,  
Dechter: „Deep Learning“ (Hinton: uses it 2006)
- 1997 Hochreiter+Schmidhuber: LSTM Net
- 2010 Hinton / Ruslan: Stacking, hierarchical feature learning
- ~2012 GPUs, Big Data, Rectified Linear Units, Dropout, ...

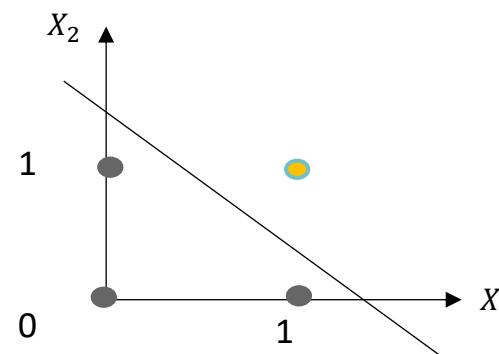
- Technical Approximation by Simplification



## Examples:

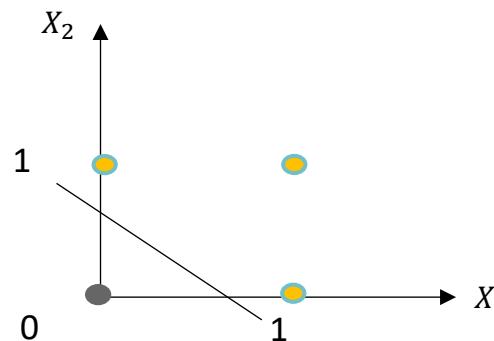
AND:

$X_1$	$X_2$	AND
0	0	0
0	1	0
1	0	0
1	1	1



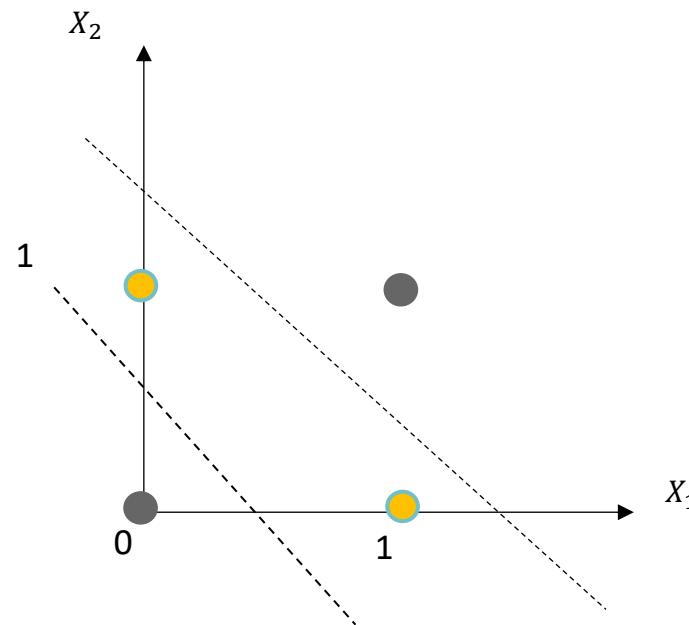
OR:

$X_1$	$X_2$	OR
0	0	0
0	1	1
1	0	1
1	1	1

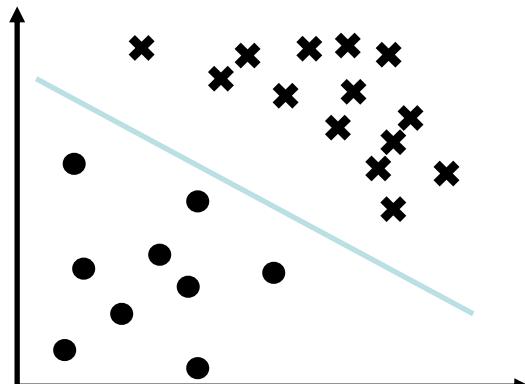


## XOR-Problem

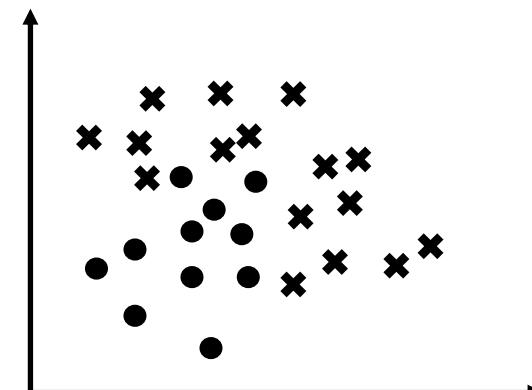
$X_1$	$X_2$	XOR
0	0	0
0	1	1
1	0	1
1	1	0



## Lineary seperable?

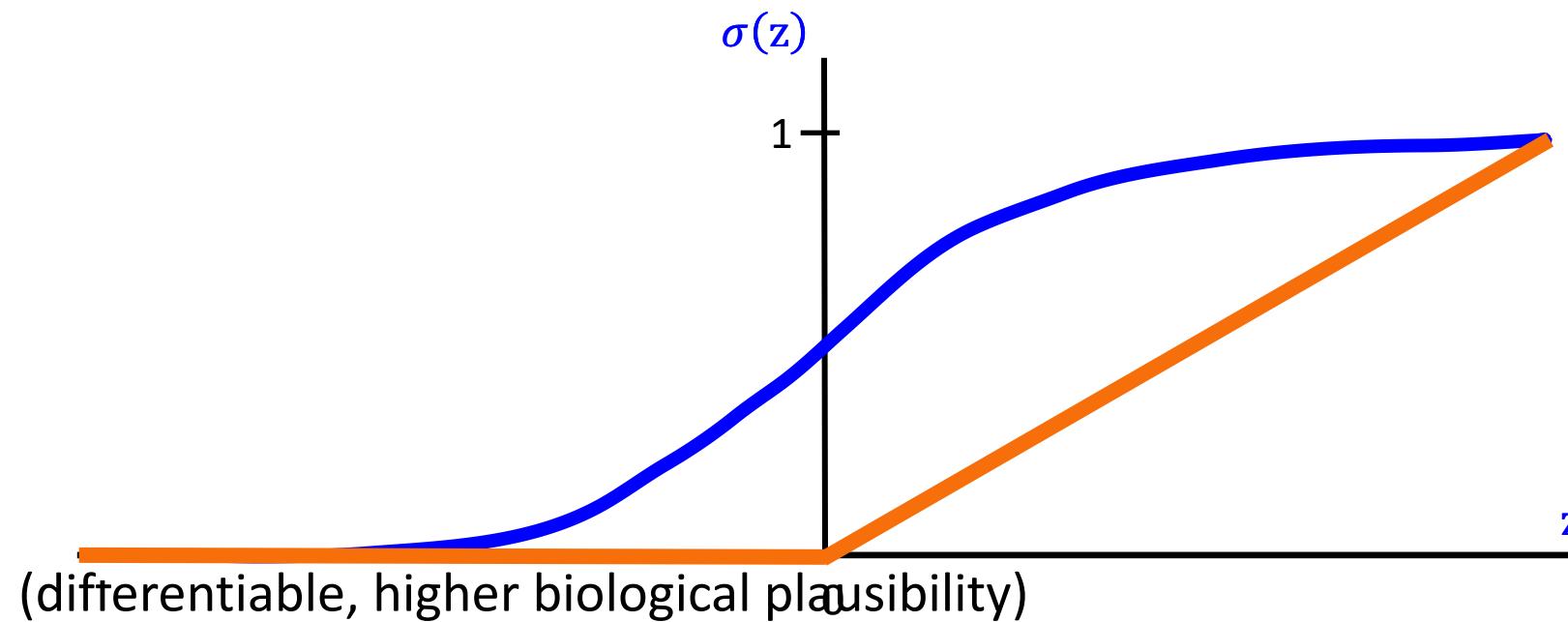


linearly separable



not linearly separable ...

Sigmoid function?



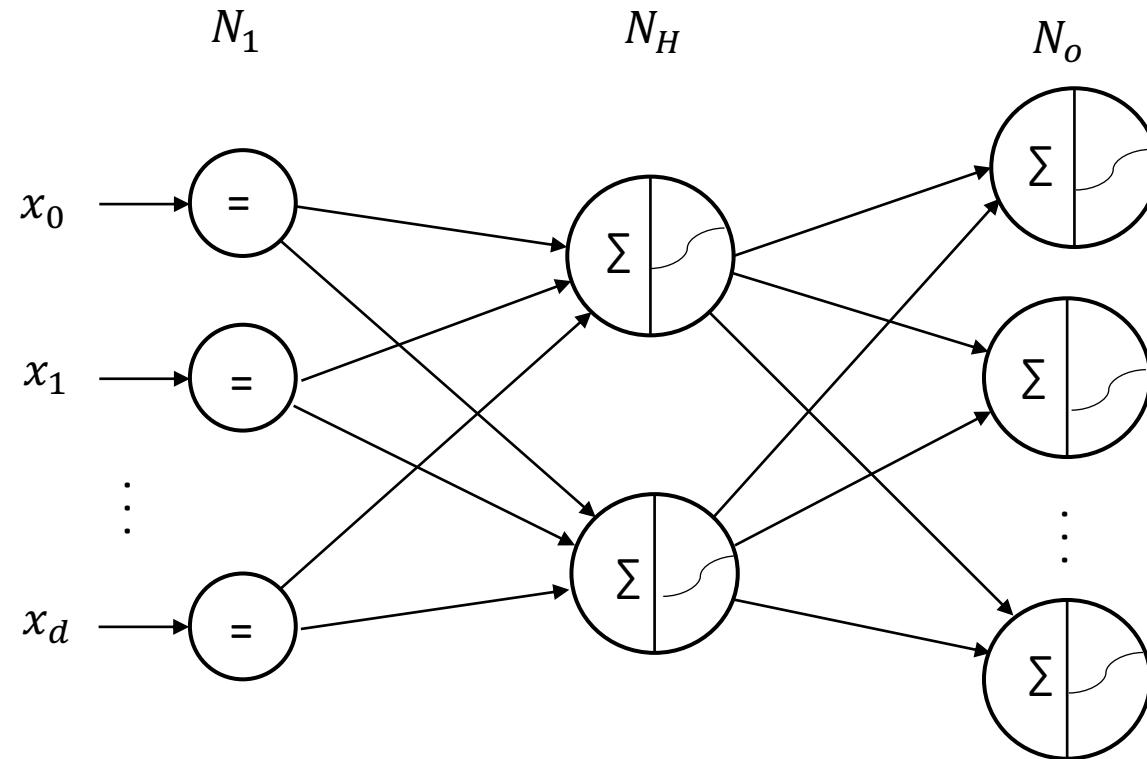
→ Deep Nets: Rectified Linear Unit

- Single Perceptron: only linear decision boundaries  
→ Combine multiple perceptrons...
- Nonlinear Neurons  
(Net of linear neurons: only linear functions)

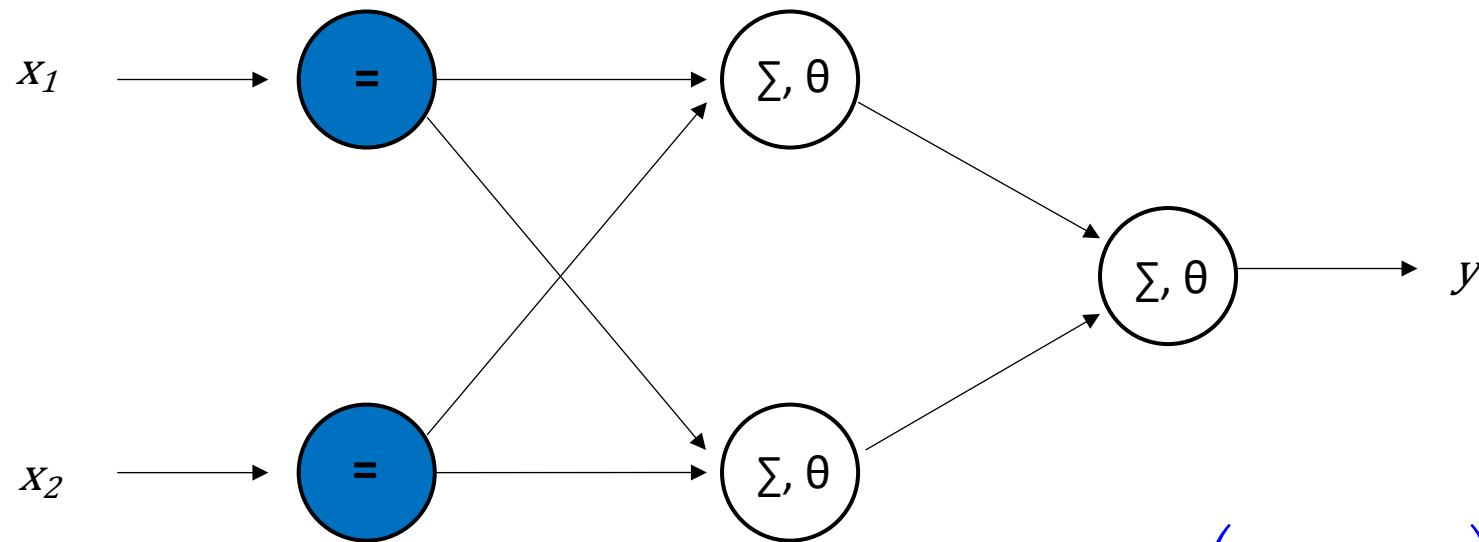
Multilayer Perceptron:

- Directed, weighted graph
- Neuron groups in (hierarchical) layers
- Connections between single layers
- Coding of information matters

## Multiple Layers



## Example XOR



$$x_1 \text{ xor } x_2 \Leftrightarrow (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

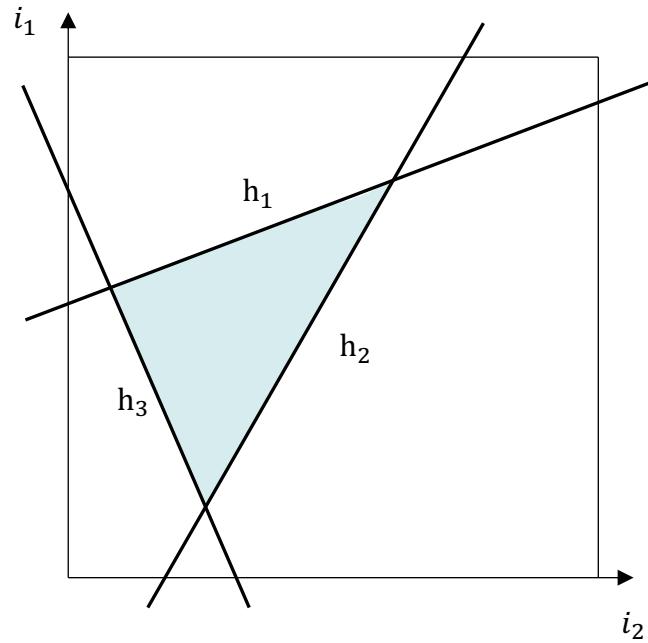
- Hidden-Neurons: each recognise one AND
- Output-Neuron: recognises OR

# Network Topology

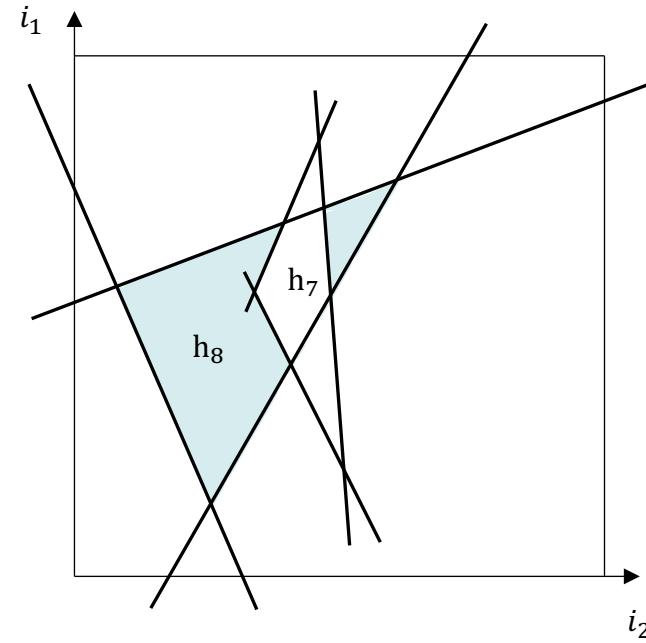
Björn W. Schuller

n	Classifiable
1	Hyper flat
2	Convex Polygon
3	Any boundary
>3	Also any boundary... (but... Deep Learning)

## Seperability by ANN:

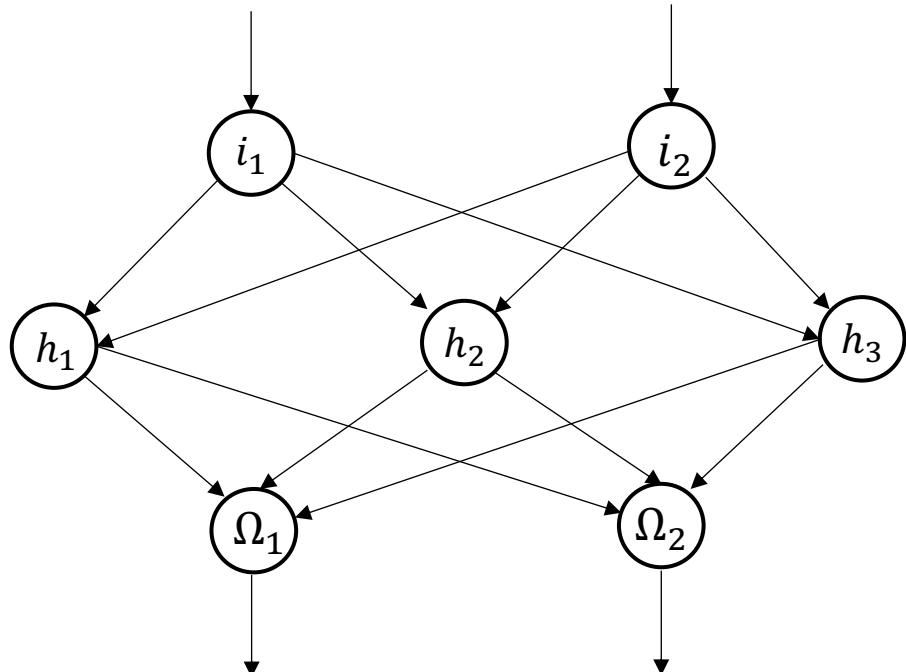


2-layer Net



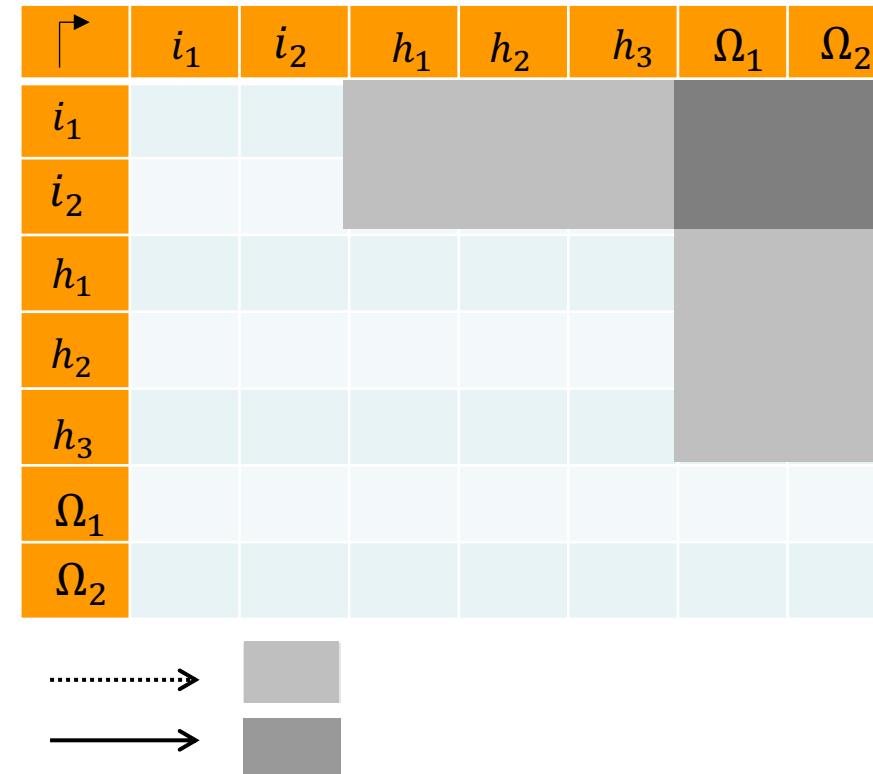
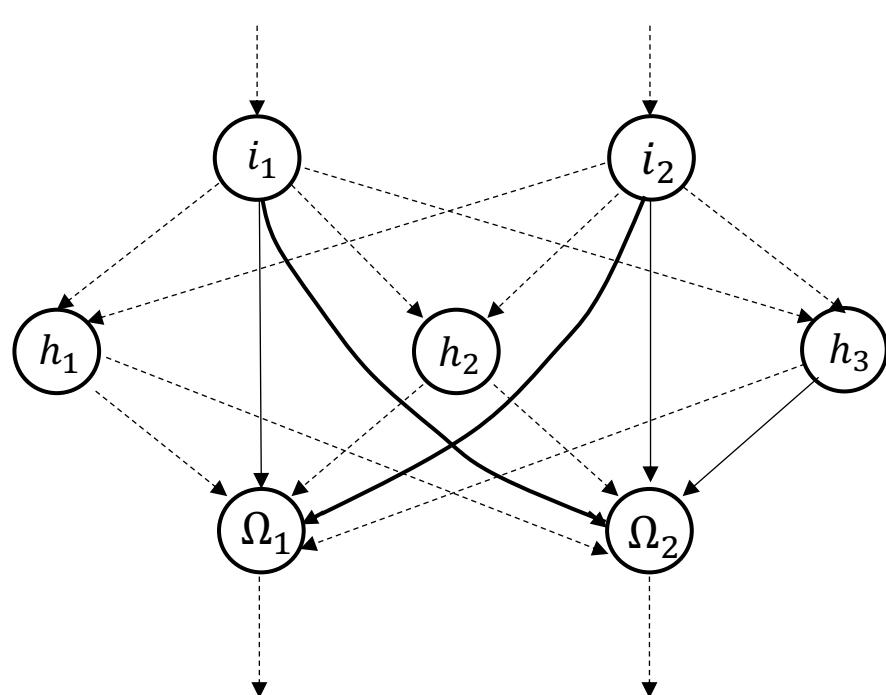
3-layer Net

Simple feed forward network:

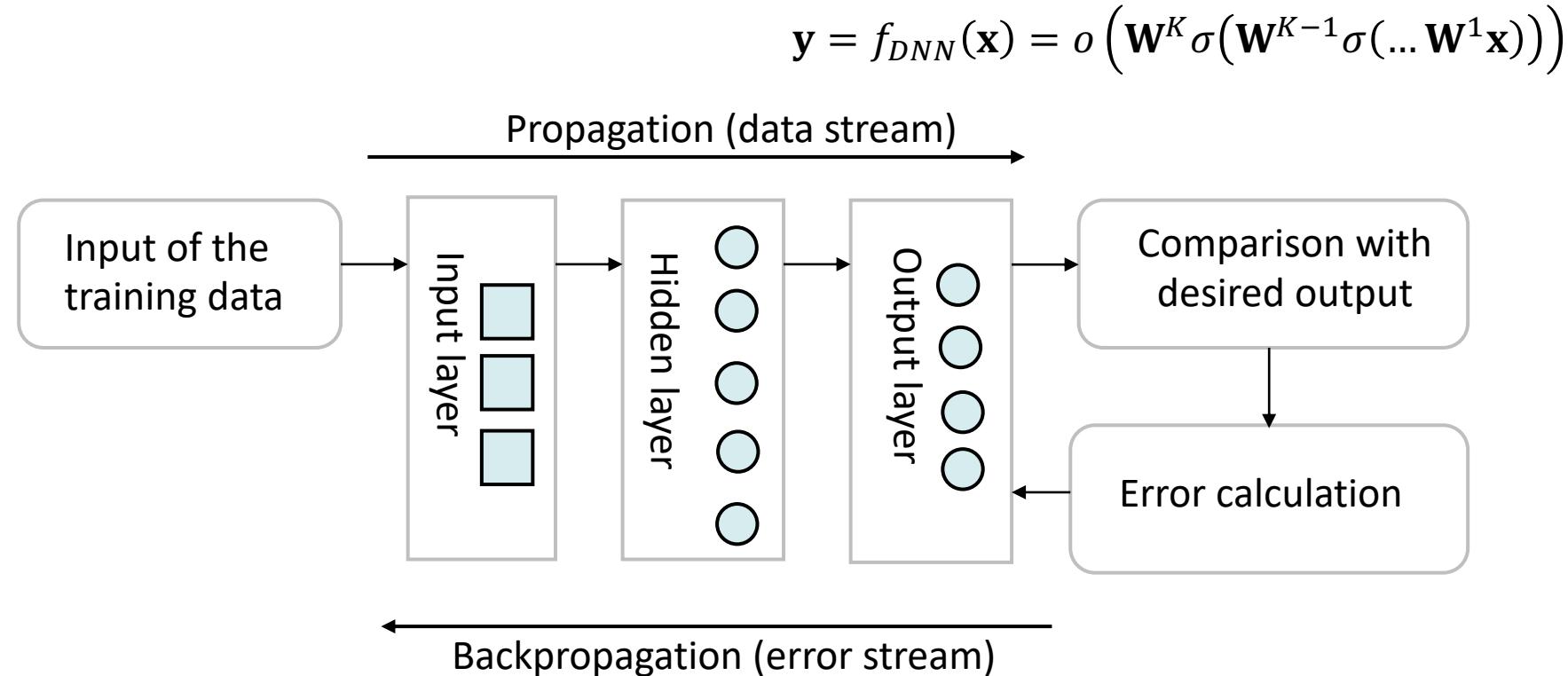


$\rightarrow$	$i_1$	$i_2$	$h_1$	$h_2$	$h_3$	$\Omega_1$	$\Omega_2$
$i_1$			■				
$i_2$							
$h_1$							
$h_2$							
$h_3$							
$\Omega_1$							
$\Omega_2$							

Feed forward network with shortcut connections:



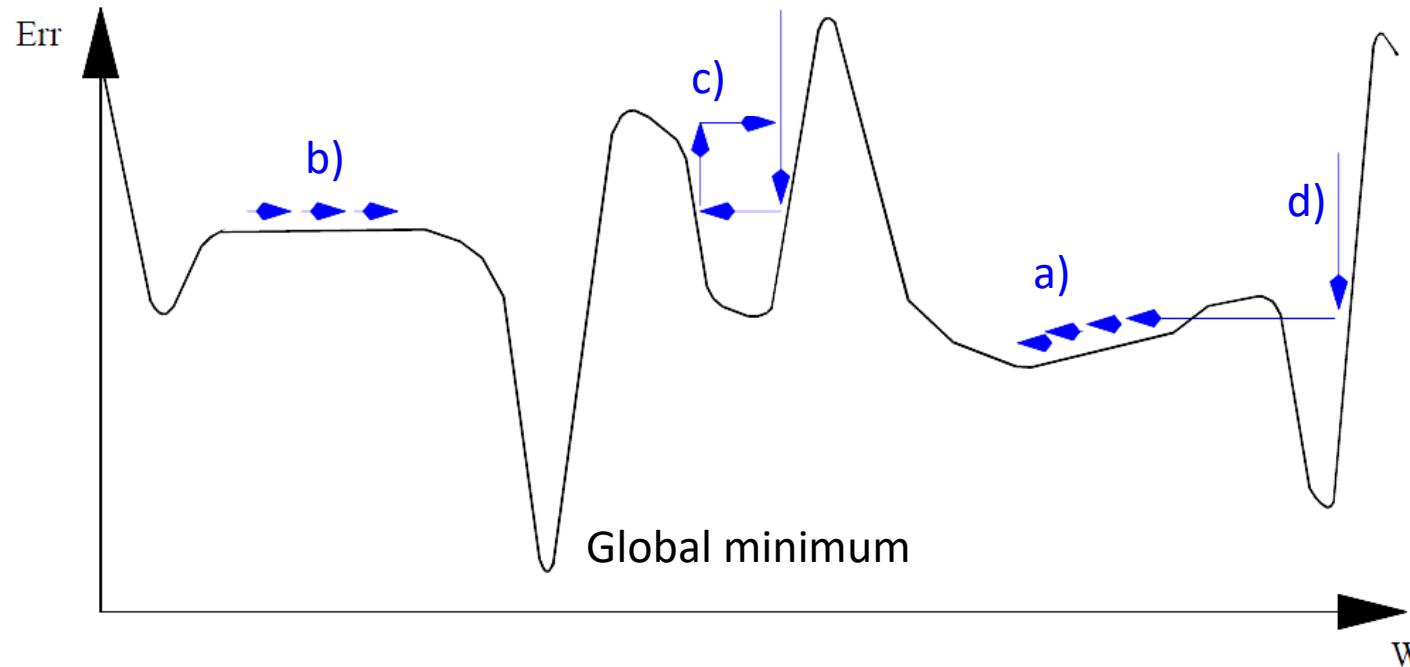
## Information flow



$$w^{(i+1)} = w^{(i)} - \eta \frac{\partial E_B}{\partial w}(w^{(i)}), B \subset Tr$$

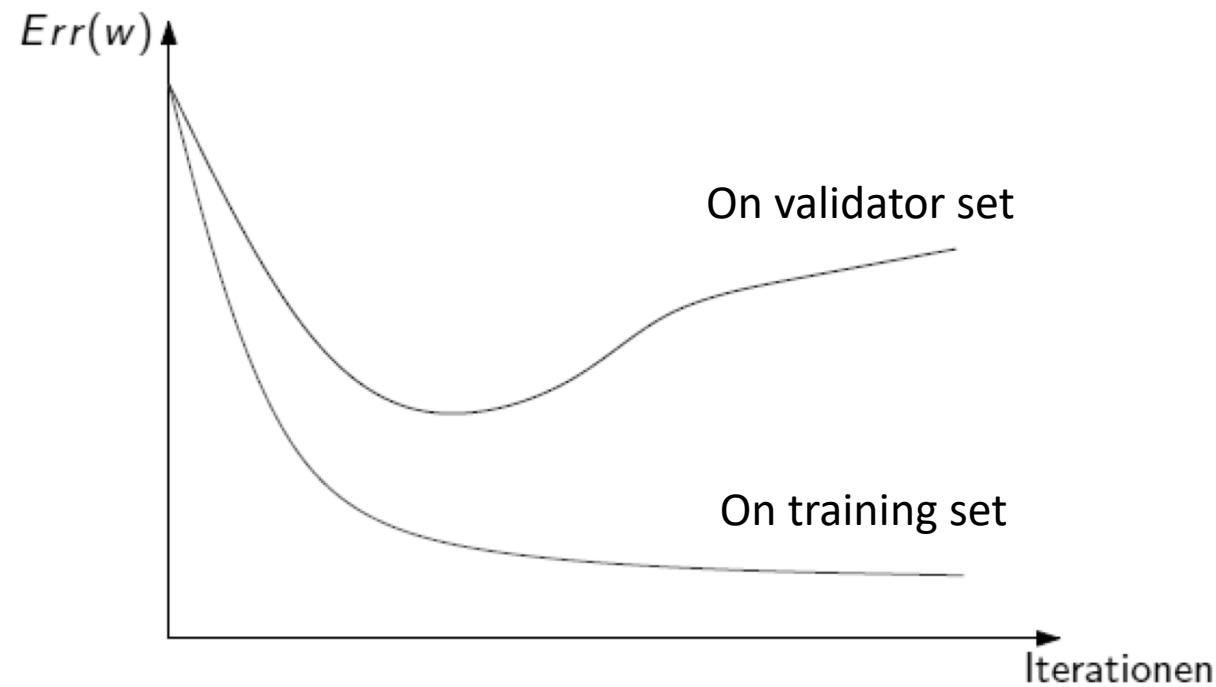
$$E_{Tr}(\mathbf{y}, \mathbf{y}^*) = \sum_{\mathbf{x} \in Tr} D(f_{DNN}(\mathbf{x}), \mathbf{y}^*)$$

## Potential problems during gradient descent



Potential problem during gradient descent: **a)** Finding local minima, **b)** Near halting due to small gradients, **c)** Oscillation in valleys, **d)** Leaving good minima

Overfitting: fitting the model too closely to the training data



# Learning Rules

Björn W. Schuller

	Hebb rule	Delta rule	Backprop	Competitive Learning
<b>Basic concept</b>	Simultaneous activation	Compare: desired vs. observed; Gradient descent	Backward-pass; Gradient descent	„The winner takes it all.“
<b>Type of learning rule</b>	Possible as supervised, unsupervised and reinforcement	Supervised learning	Supervised learning	Unsupervised learning
<b>Biological plausibility</b>	Partial	Mostly not	Mostly not	Partial
<b>Network types that make use of this learning rule (among others)</b>	Pattern Associator; Auto Associator	Pattern Associator; Auto Associator	Simple Recurrent Networks, Jordan Networks	Competitive Networks; conceptually Kohonen networks as well

# Learning Rules

Björn W. Schuller

	Hebb rule	Delta rule	Backprop	Competitive Learning
<b>Advantages</b>	Simplicity, biological plausibility	Simplicity, relative ease of implementation	Usable in networks with hidden units; more powerful than delta rule	Unsupervised learning; biological plausibility
<b>Disadvantages</b>	In the “classical” form: overflow of the weights and lacking power overall	Not usable in networks with hidden units; questionable biological plausibility; lacking power overall	Questionable biological plausibility; local minima	A single output unit can claim all input patterns → no classification possible anymore

Deep Learning.



## Artificial Intelligence:

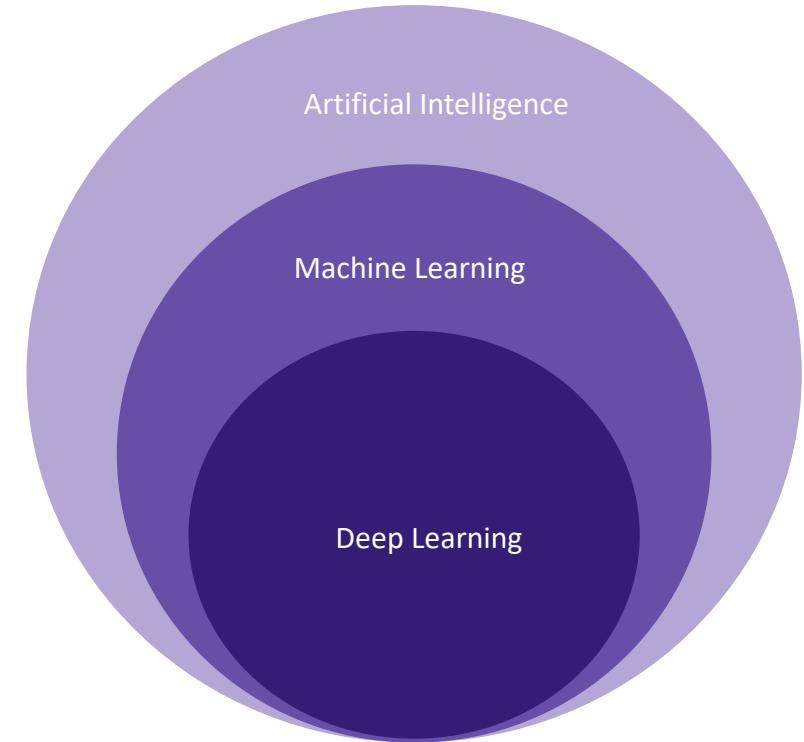
- A broad concept where machines think and act more like humans

## Machine Learning:

- An application of AI where machines use data to automatically improve at performing tasks

## Deep Learning:

- A machine learning technique that processes data through a multi-layered neural network much like the human brain



- **Machine Learning**

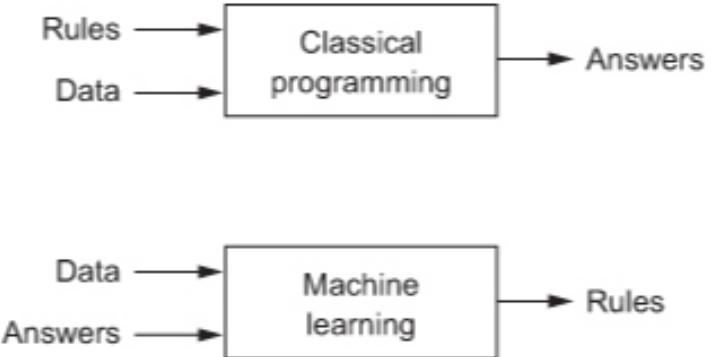
- Discovering rules to execute a data-processing task

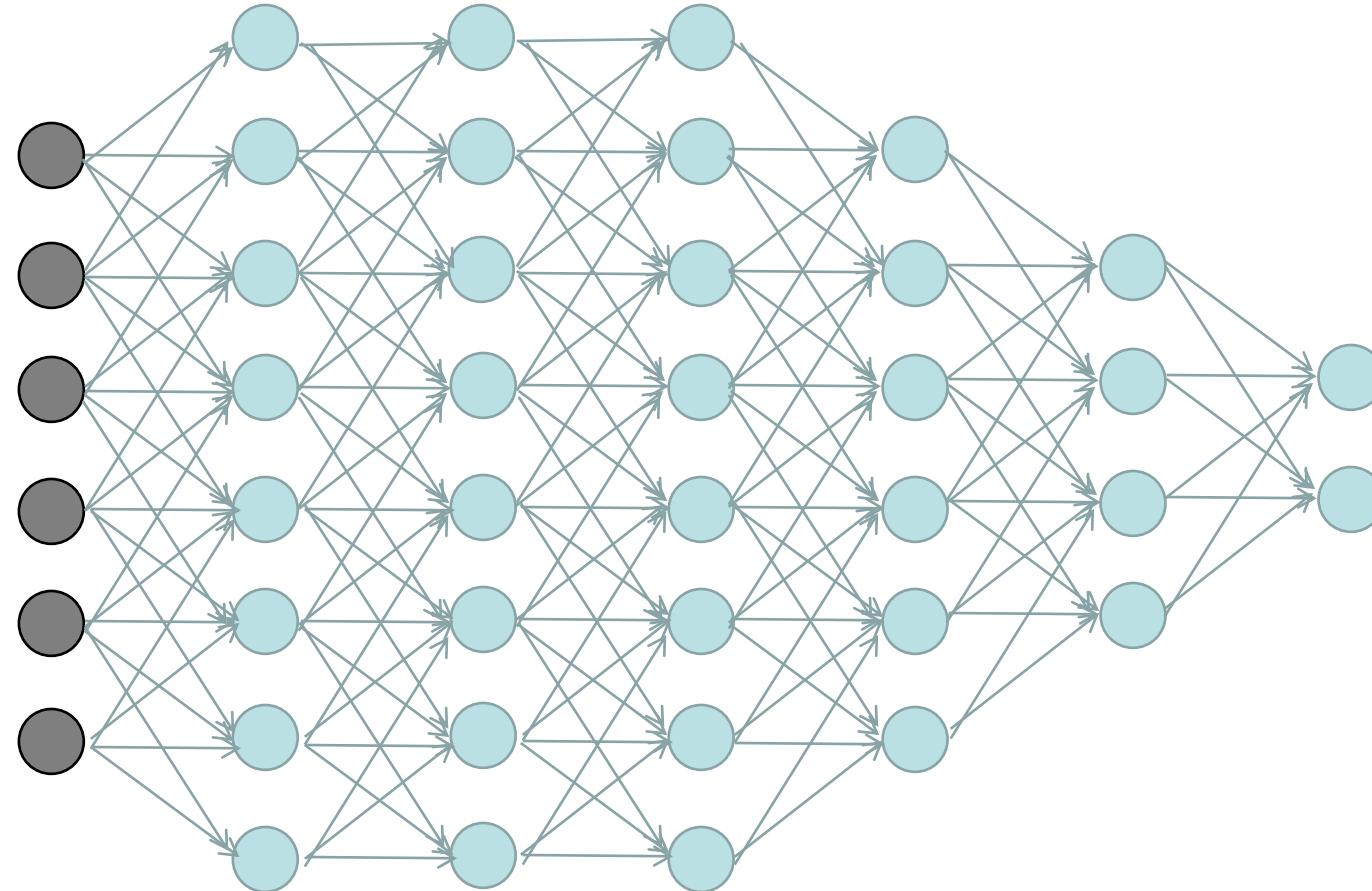
- **Classic programming:**

- Input rules (a program) and data
    - Process the data according to these rules
    - Output answers

- **Machine Learning:**

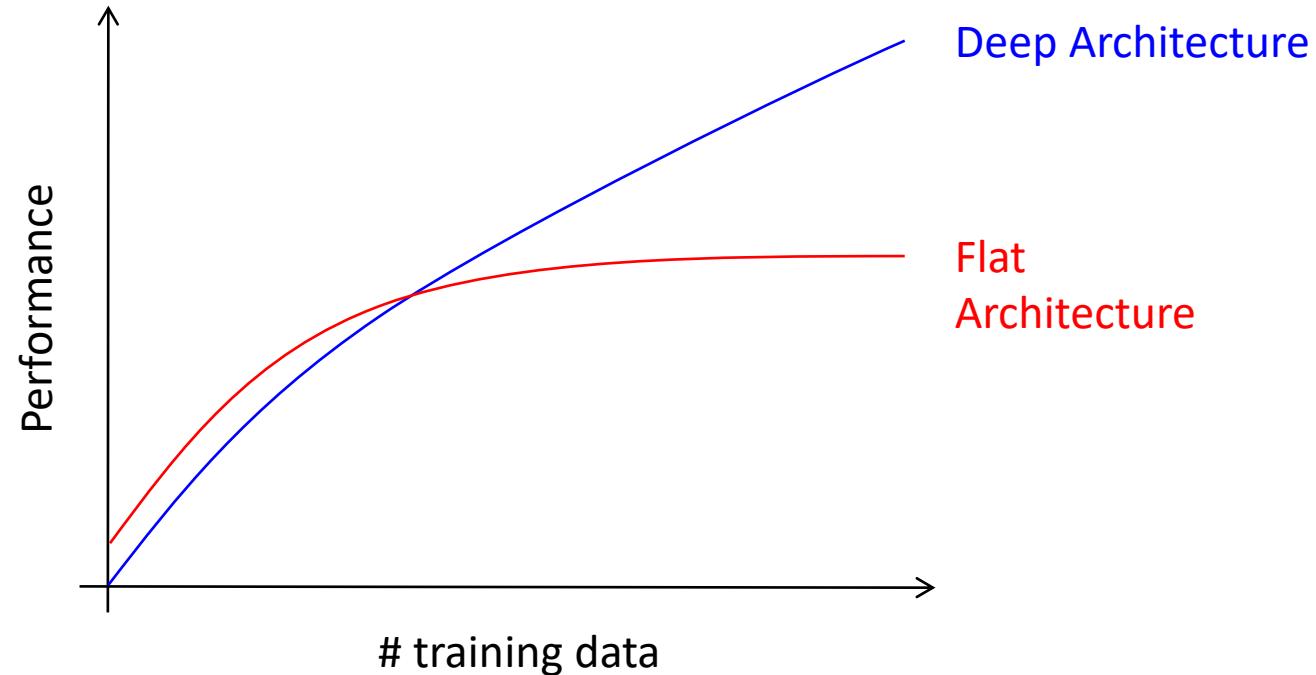
- Input data as well as the answers expected from the data
    - Learning the rules need to map between data and answers
    - Output the rules so they can then be applied to new data to produce original answers



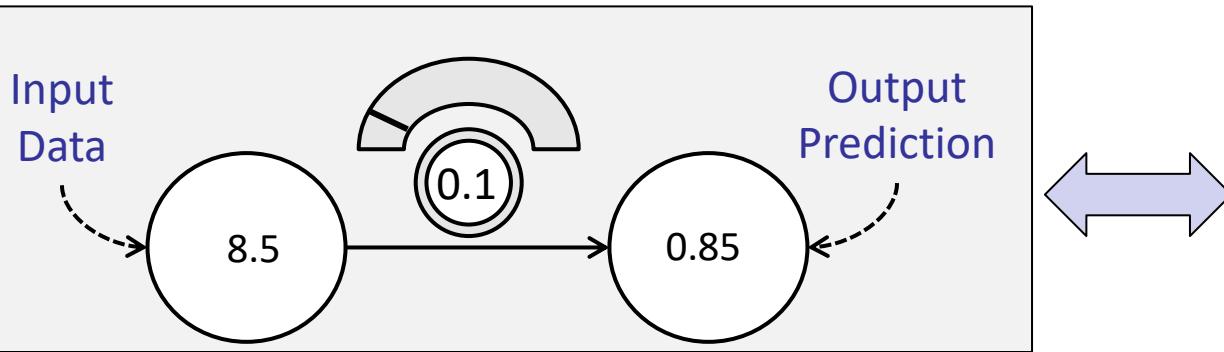


- Definition
  - Minimum ca. 4 layers
  - AlexNet (2012): 8 (learning) layers w/ 60 mio parametres
  - Inception-v3 (2015): 47 (learning) layers w/ 25 mio parametres
- Expectations
  - Learning of highly complex functions enabled
  - Application on hard tasks
  - Reuse of parts of the net for other tasks (→ Transfer-Learning)

"Flat" Architectures reach their limits (#parametres!):



- **Gradient Descent for neural learning**



```
pred = input * weight  
error = (pred - goal_pred) ** 2  
derivative = input * (pred - goal_pred)  
weight = weight - (alpha * derivative)
```

- **error** is a measure of how much the network missed by
  - We define **error** to be always positive
- **derivative** is the derivate of **weight** and **error**
  - Predicts both direction and amount to adjust the weights
- **Alpha** scales the **weight update**
  - Helps minimise divergence effects when the input is large

- **Gradient Descent**

- Learning is adjusting the weight to reduce the error to 0

- The function conforms to the patterns in the data

```
error = (pred - goal_pred) ** 2
```

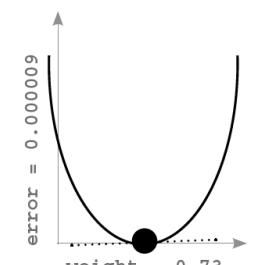
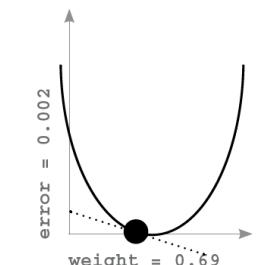
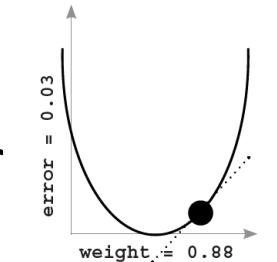
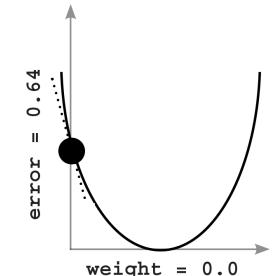
- Derivative of the error functions defines the amount that error changes when you change weight

```
derivative = input * (pred - goal_pred)
```

- How can we use the derivative to minimise the error?

- Adjust the weights in the opposite direction of the derivative

1. Calculate the derivative of weight with respect to error
2. Change weight in the opposite direction of that slope.



- **Data-Transformation**

- A machine-learning model transforms its input data into meaningful outputs

$$x \xrightarrow{f(\cdot)} y$$

- “learnt” from exposure to known examples of inputs and outputs
  - **Learning to meaningfully transfer data is the central problem in machine learning and deep learning**
    - Learn useful representations of the input data at hand
    - These representations should get us closer to the expected output

- **Role of Activation Functions**

- Weights in a single layer network learn the correlation between the input and output
- To learn when there is no correlation, just use more layers
  - Essence of deep learning: Each hidden layers attempts to learn a data representation that has greater correlation with the output
- However, stacking linear neural networks does not give a network any more power
- Activation functions can be thought of as means of inducing conditional correlation between layers

- **Role of Activation Functions**
  - **Good activation functions are nonlinear**
    - Allow for selective correlation: increase or decrease how correlated the neuron is to all the other incoming signals
  - **Other core properties**
    - The function must be continuous and infinite
    - The function should be monotonic
      - I.e., no two input values of have the same output value
    - The function and its derivative should be easily computable
      - Enable efficiency when training and deploying the network

- **Linear**

- $f(x) = ax$
  - Range  $-\infty$  to  $\infty$

- **Sigmoid**

- $\sigma(x) = \frac{1}{1+e^{-x}}$
  - Range: 0 to 1

- **Hyperbolic Tangent**

- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
  - Range:  $-1$  to  $1$

- **Rectified Linear Unit**

- $ReLU(x) = \max(0, x)$
  - Range 0 to  $\infty$

- **Leaky Rectified Linear Unit**

- $LReLU(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0.01x & \text{for } x < 0 \end{cases}$
  - Range  $-\infty$  to  $\infty$

- **Softmax**

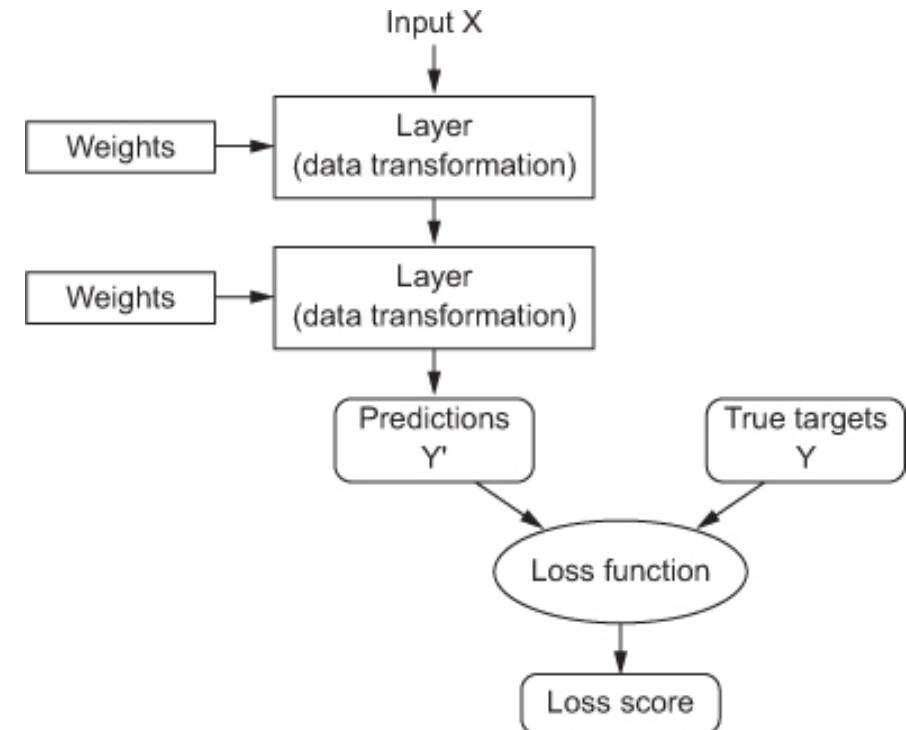
- $softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(j)}$
  - Range: 0 to 1

- **Learning in Deep Neural Networks**

- To control weight updates in neural networks we use a **loss function** to how far an output prediction is from what we expected
  - The loss function computes a single scalar value relating to network performance
  - Measures the difference between what we have predicted,  $\tilde{y}$ , with the what it should predicted  $y$ .

$$\mathcal{L}(\tilde{y}, y)$$

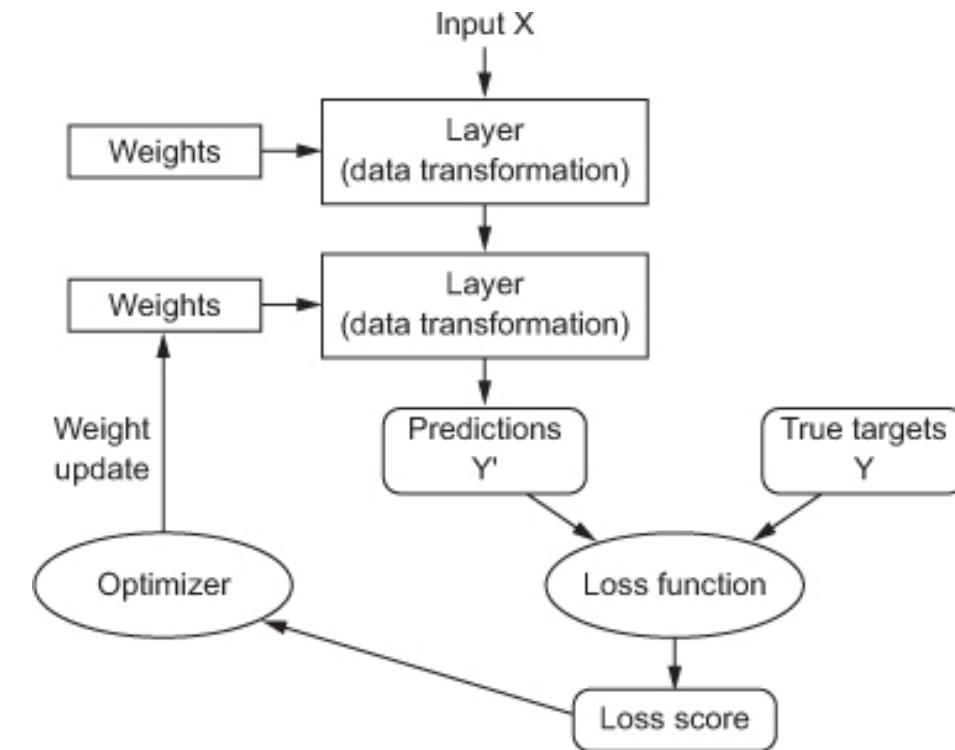
- We can then use this information to update the network



- **Regression**
  - Predicting a single numerical value
  - Final activation – **Linear**
  - Loss function – **Mean Squared Error**
- **Binary outcome**
  - Data is or isn't a class
  - Final Activation function – **Sigmoid**
  - Loss function – **Binary Cross Entropy**
- **Single label from multiple classes**
  - Multiple classes which are exclusive
  - Final Activation function – **Softmax**
  - Loss function – **Cross Entropy**
- **Multiple labels from multiple classes**
  - If there are multiple labels in your data
  - Final Activation function – **Sigmoid**
  - Loss function – **Binary Cross Entropy**

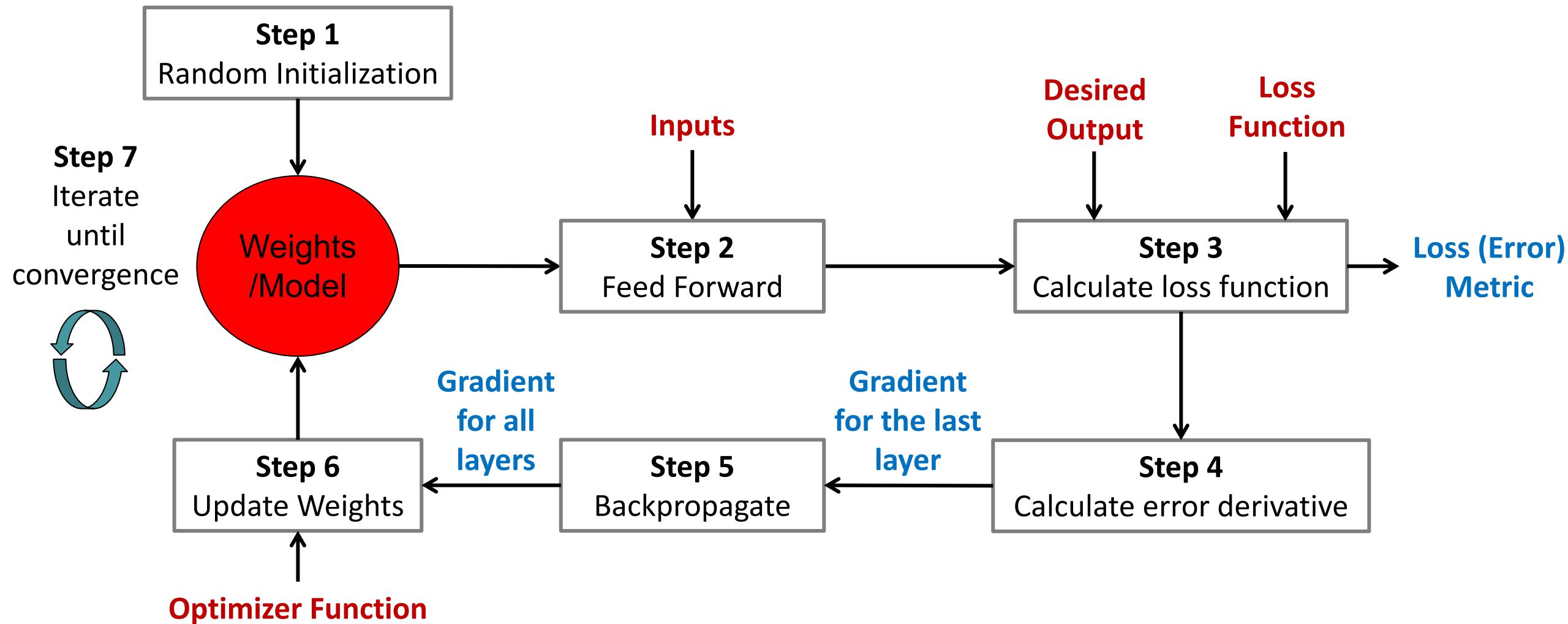
- **Learning in Deep Neural Networks**

- The loss function provides a feedback signal to adjust the weights by a small amount, in a particular direction that will lower the score
- This adjustment is performed by an *optimizer*, which is implementing the *Backpropagation* algorithm
  - Error attribution: figuring out how much each weight contributed to the final error by propagating the error back through the network



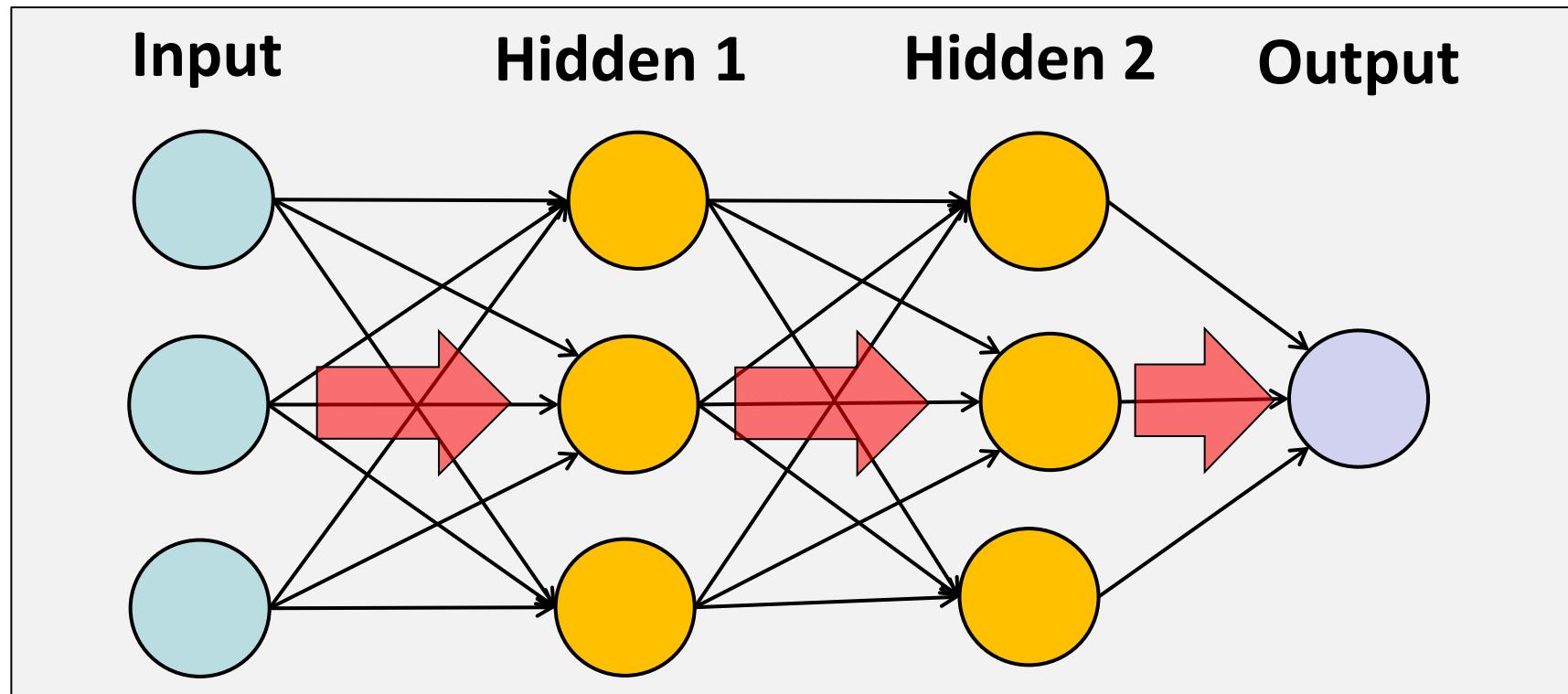
- **Optimisers**

- Gradient Descent
  - Slow to converge, need to heuristically set learning rate
- Momentum
  - Uses gradient of previous time step to accelerate convergence
- Nesterov Accelerated Gradient (NAG)
  - Calculates gradient with respect to the future step
- Adagrad Adaptive Gradient Algorithm
  - An adaptive learning rate method
- Adadelta and RMSProp
  - Adaptive approach which restricts the window size of accumulated past gradients
- Adam Adaptive Moment Estimation
  - Calculates adaptive learning rate from first and second moments of the gradients



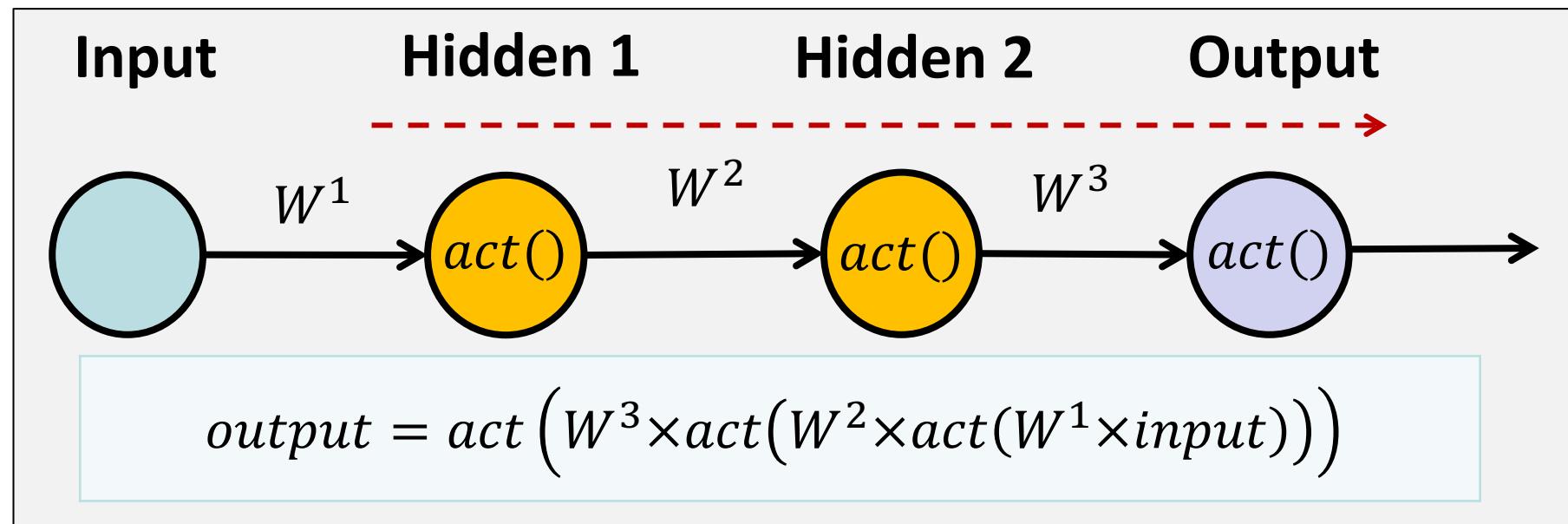
- **Forward Propagation**

- Information flows from input to output to make a predication



- **Forward Propagation**

- Each neuron is a function of the previous one connected to it
  - Output is a composite function of the weights, inputs, and activations
    - Change any one of these and ultimately the output will change



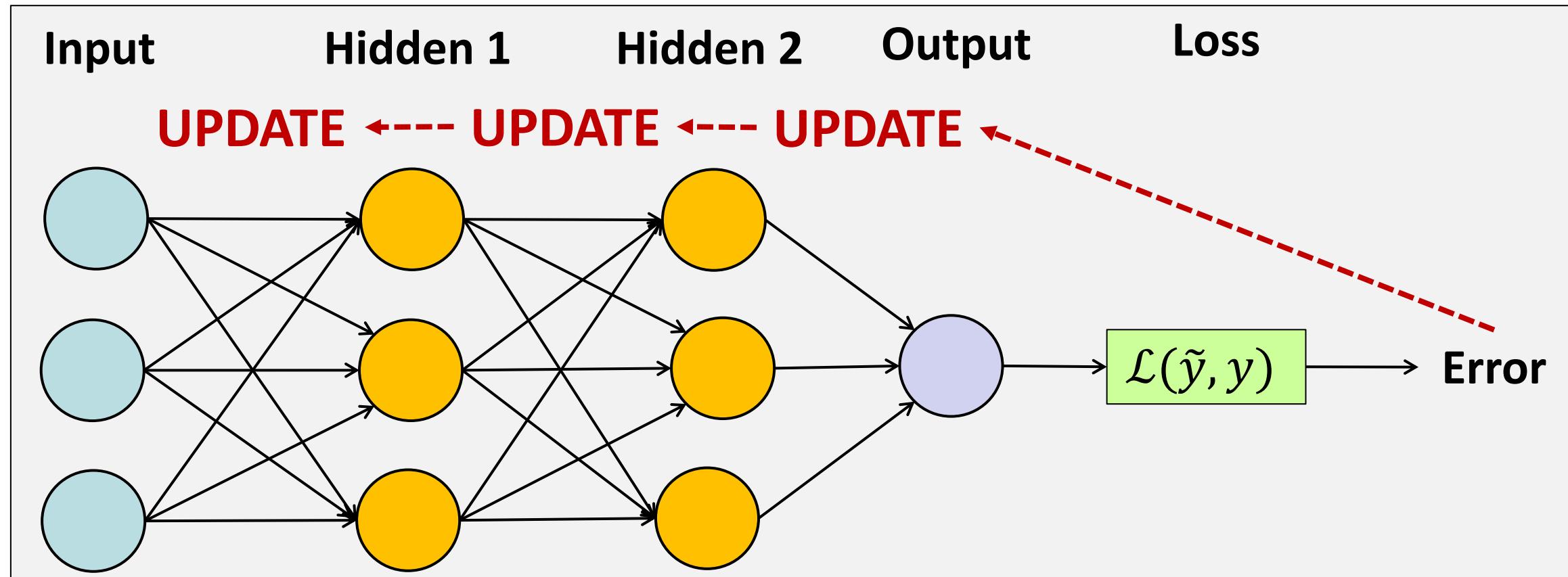
- **Weights Learnt via Gradient Descent**

- Update weights to minimise loss function
- This is achieved by taking the gradient of loss function with respect to the weights

$$W += W + \alpha \frac{\partial j}{\partial w}$$

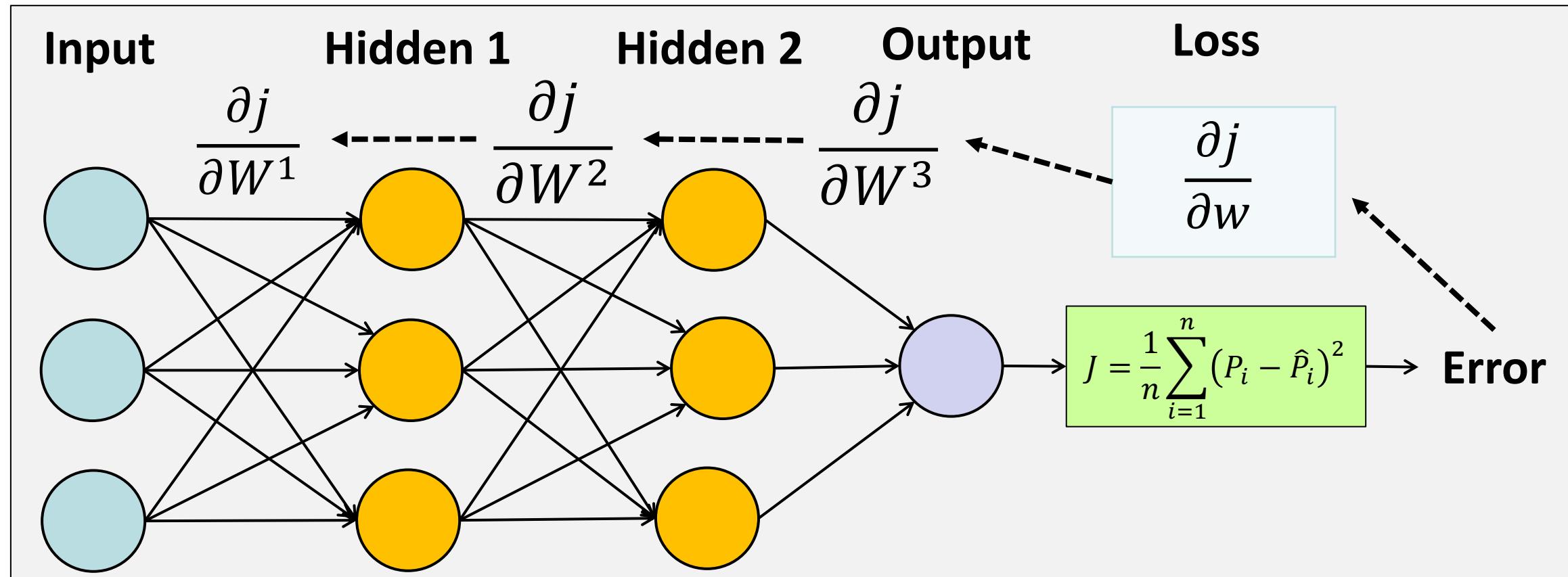
- Not a trivial process as neural networks are structured as a series of layers
  - The output is a composite function of the weights, inputs, and activation function(s)

- Perform Gradient descent
  - Output value effected by weights at all layers

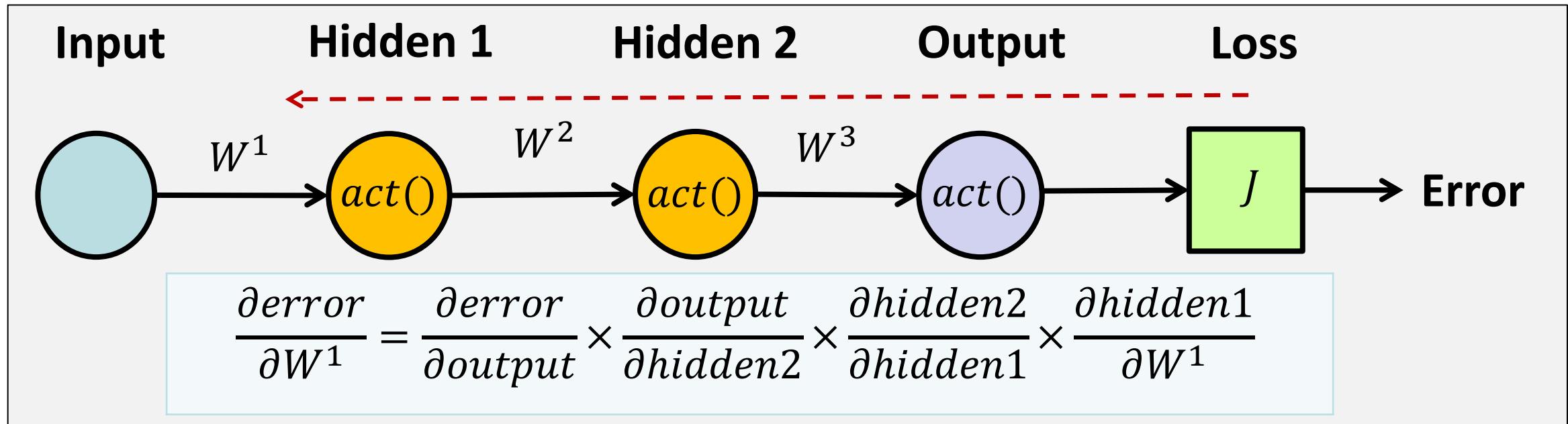


- **Backpropagation**

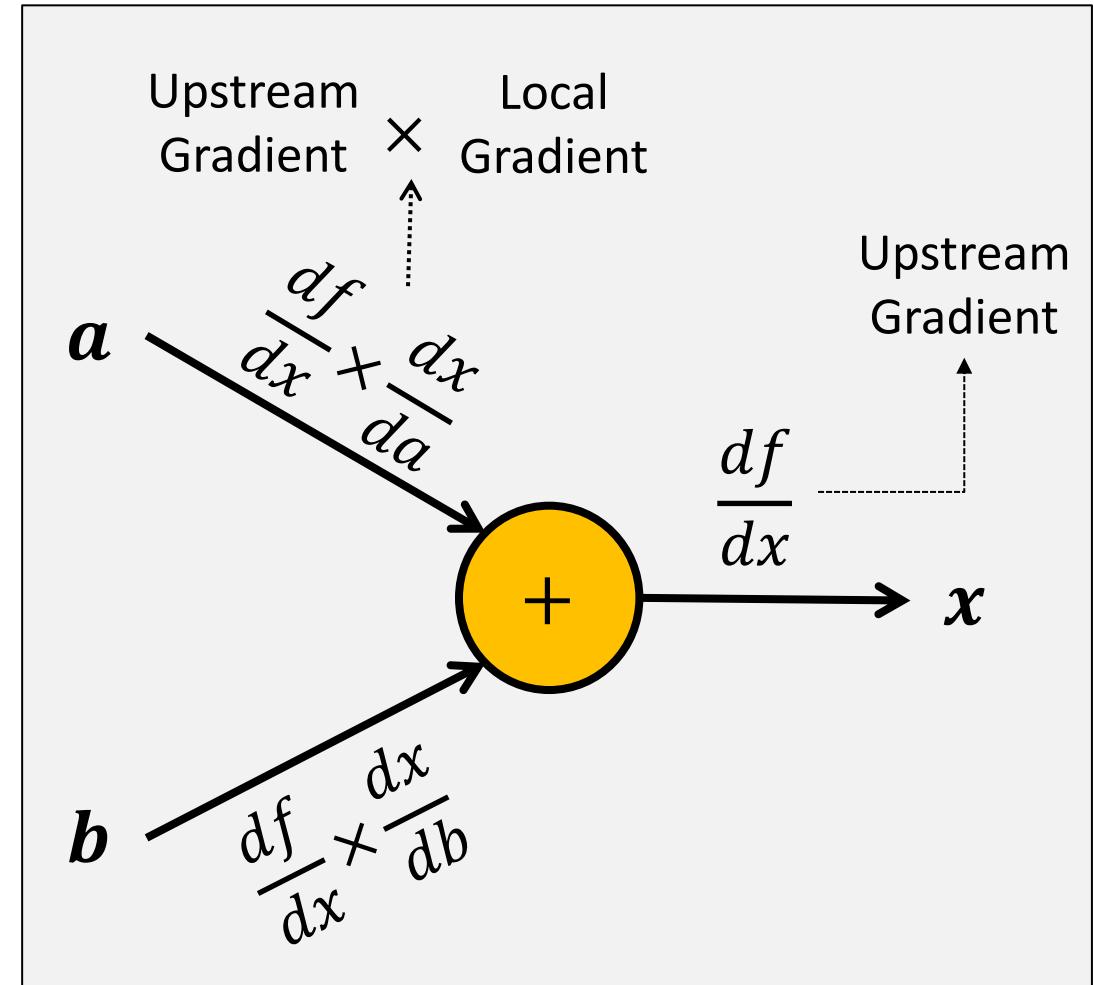
- Tool to calculate the gradient of the loss function for each weight



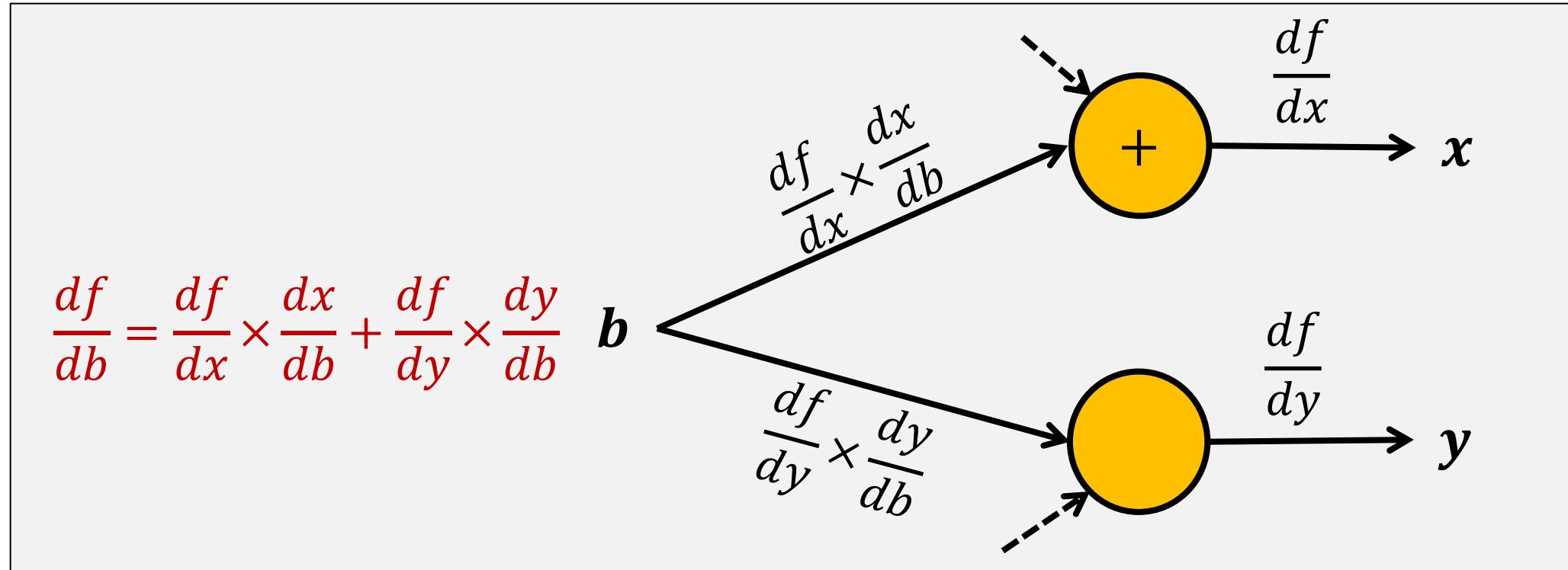
- **Calculating gradient for arbitrary weight**
  - Iteratively apply the chain rule
  - Note: Error is now a function of the output and hence a function of the input, weights, and activation functions



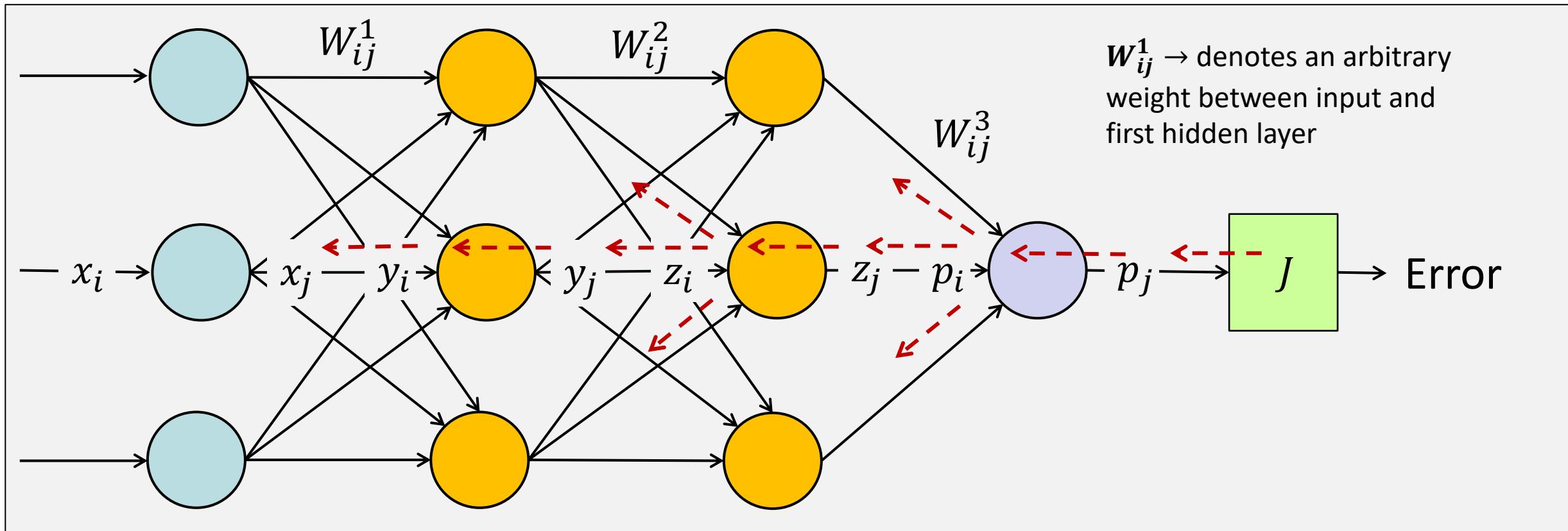
- **Calculating gradient via backpropagation**
  - Local gradients can be calculated before starting the backpropagation process
  - **Iteratively apply the chain rule**
    - The derivative of the output of the network with respect to a local variable is found by multiplying the local gradient with the upstream gradient



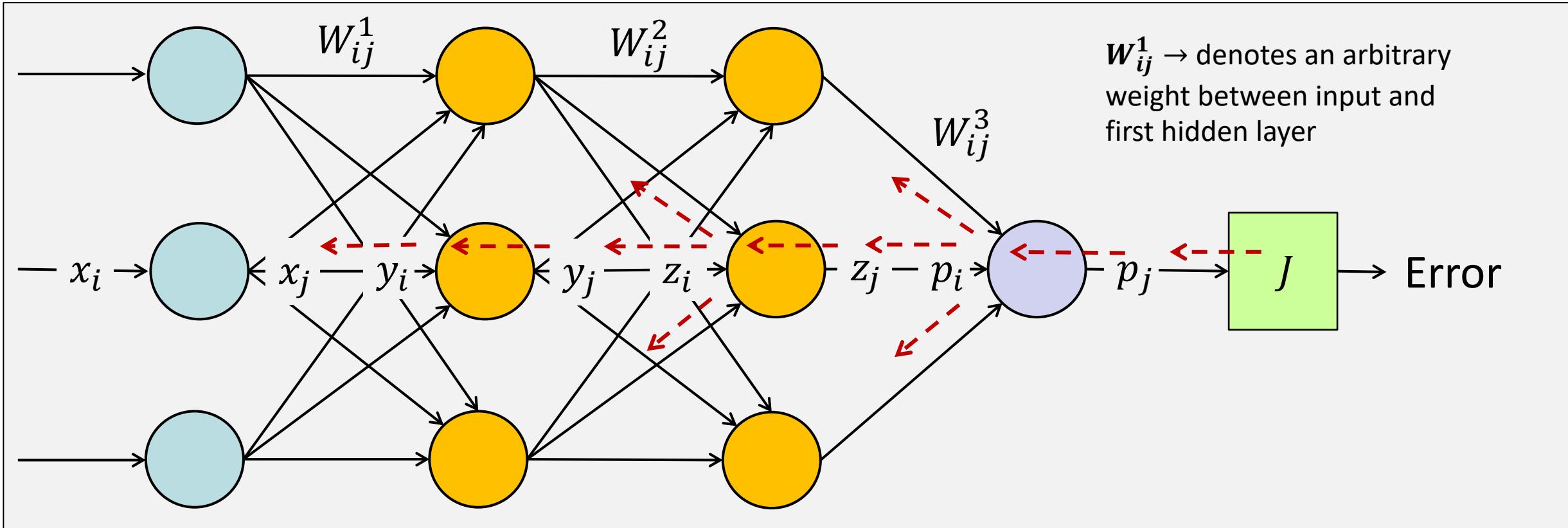
- Calculating gradient via backpropagation
  - Multivariate chain rule: Gradients add at branches



- Example: Calculating  $\frac{\partial J}{\partial W_{ij}^1}$



$$\frac{\partial J}{\partial W_{ij}^1} = (p_j - a)p_j(1 - p_j) \sum_j \left[ W_{ij}^3 z_j (1 - z_j) \sum_j \left[ W_{ij}^2 y_j (1 - y_j) \sum_j x_j \right] \right]$$

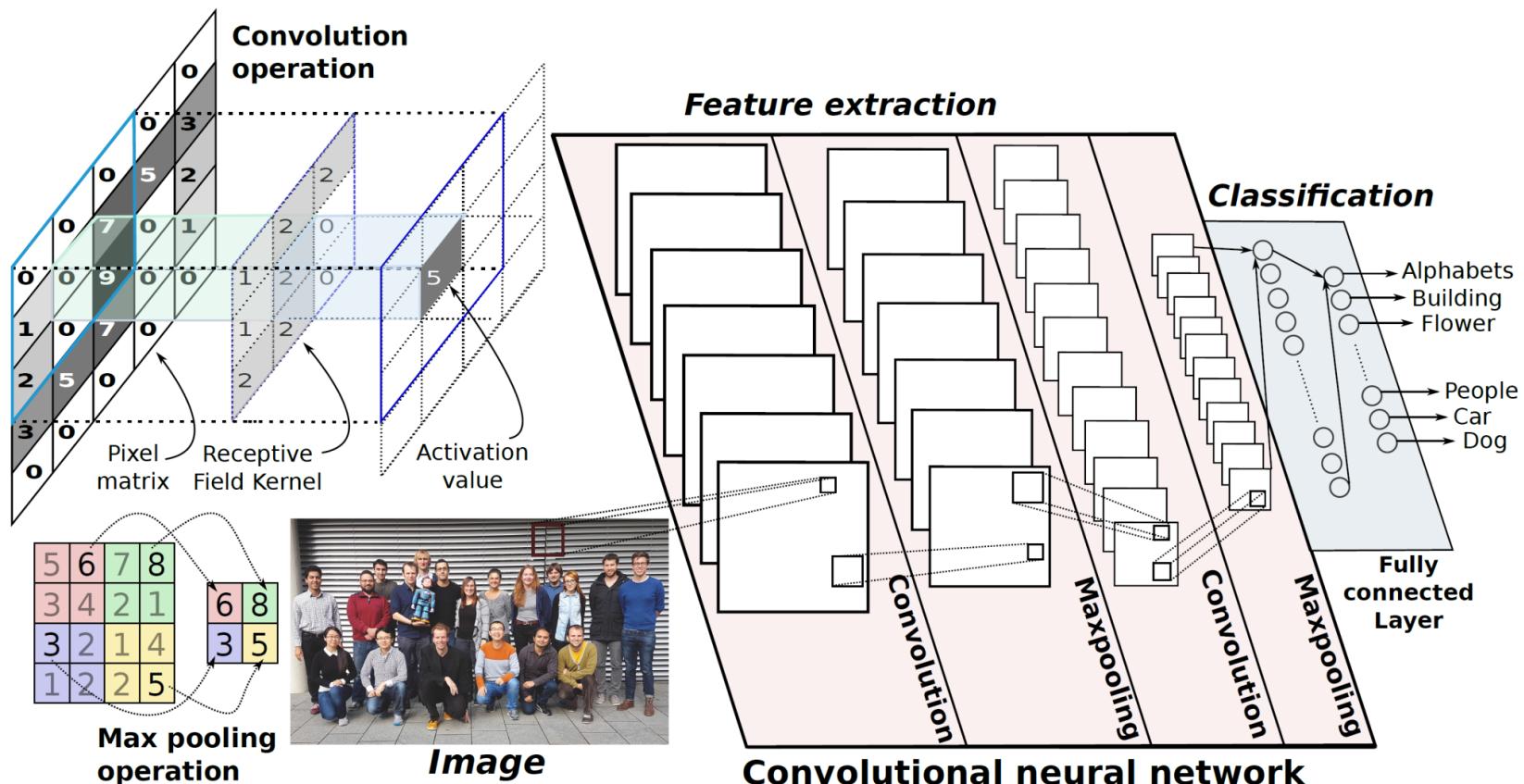


# Convolutional NNs .

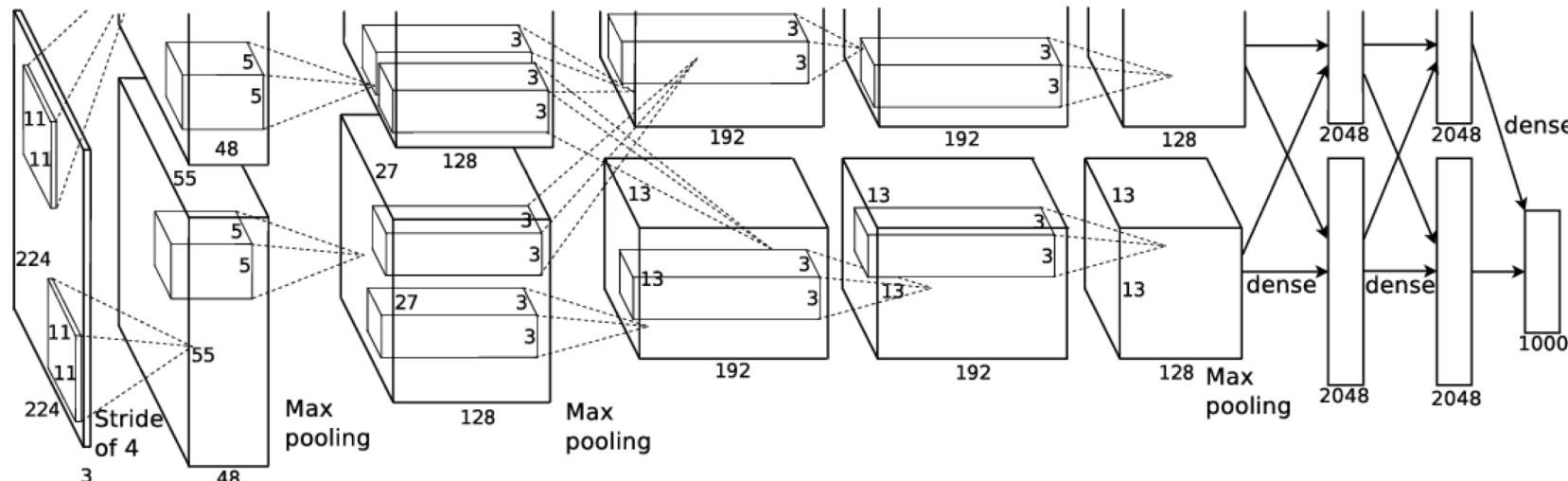


- **What is a Convolutional Neural Network?**
  - A CNN is a neural network with convolution operations instead of matrix multiplications in at least one of the layers
  - Special form of feed-forward network
    - Reuse the same neurons for repetitive convolution tasks
      - Convolution  $\approx$  Filtering
    - Convolutional kernels recognise patterns in a signal
    - Reuses weights to detect the same patterns in multiple places
    - Reduces overfitting and leads to much more accurate models

- **Convolutional Neural Network (CNN)**
  - Convolutional kernels perform feature extraction



- CNN originally proposed in image recognition in 1998
  - Gained wide spread notoriety in 2012 after a deep CNN easily won the ImageNet Large Scale Visual Recognition Challenge
    - AlexNet: Error rate: 17.0% on a 1000-class classification task



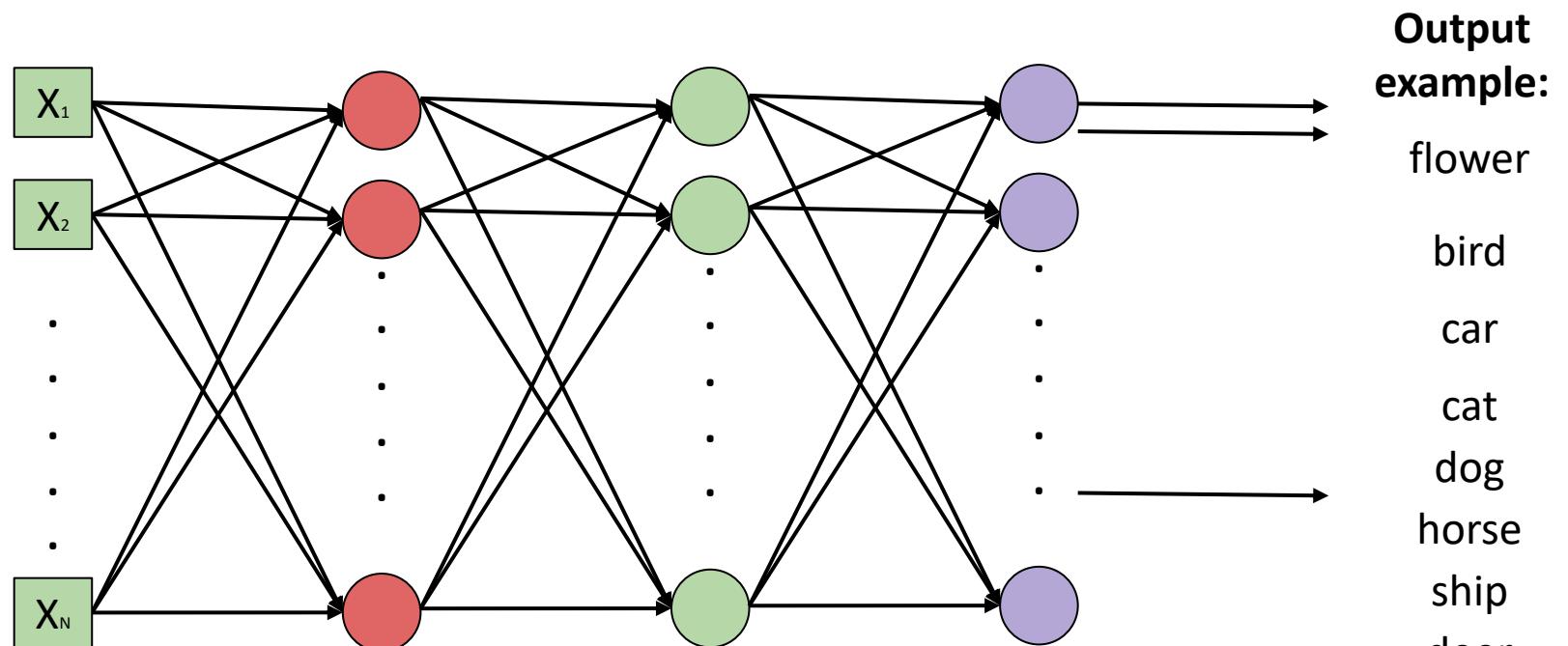
Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *NIPS*. 2012.

- When applying a fully connected feedforward neural network
  - Many thousands of weights and connections
  - However, network can be simplified by considering the properties of input signal



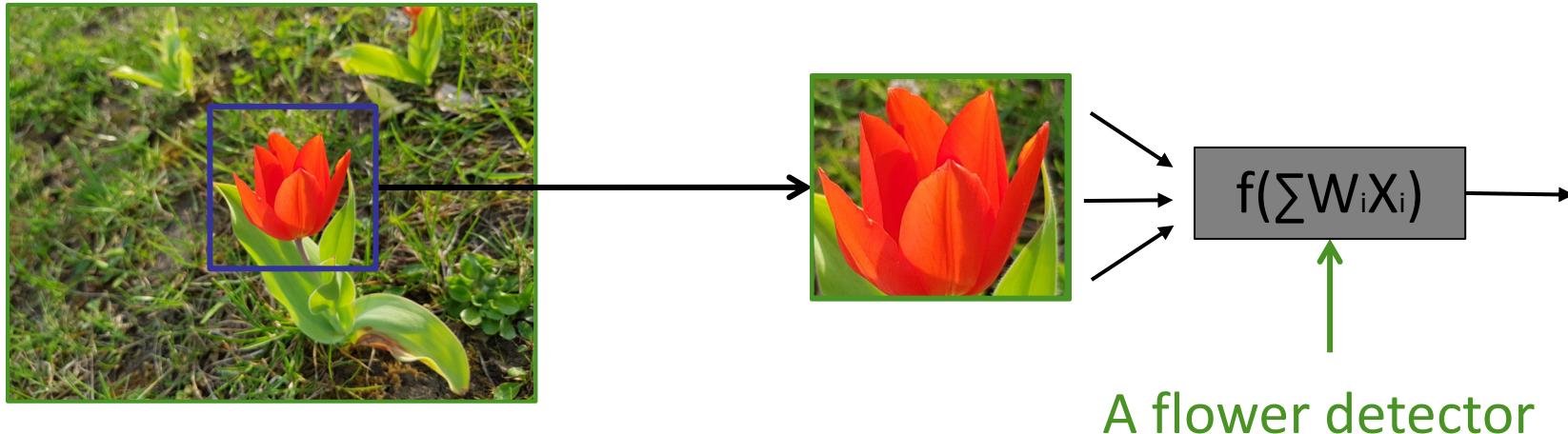
An RGB image can be represented as pixels  
 $(32 \times 32 \times 3)$

Input vector dimension  
 $N = 32 \times 32 \times 3 = 3072$

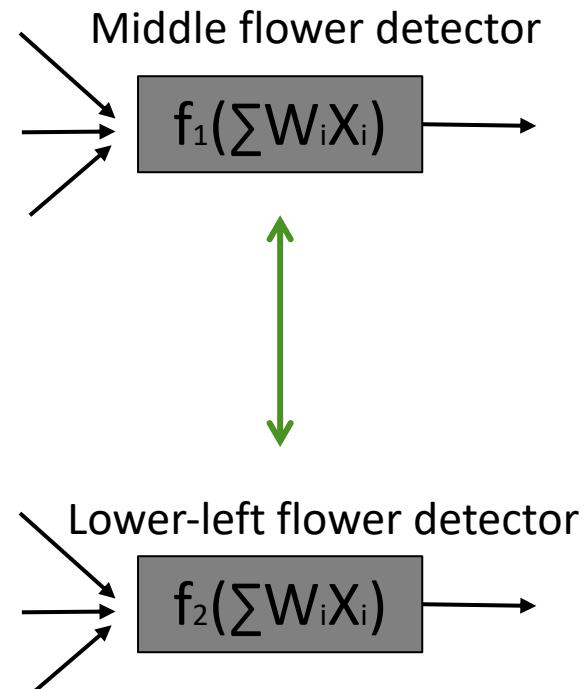
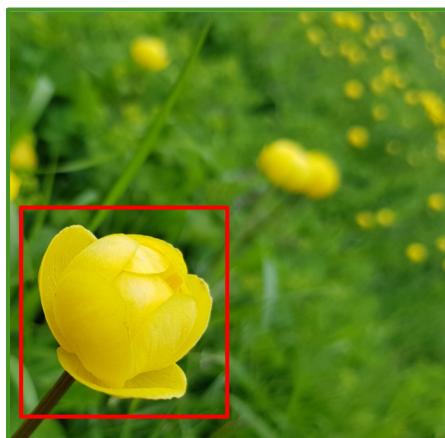


Output example:  
flower  
bird  
car  
cat  
dog  
horse  
ship  
deer

- Key patterns can be much smaller than the whole signal
  - Therefore a network does not have to see the whole signal to discover the pattern
  - Only need to be connected to small region ( $X_i$ )
  - Fewer parameters ( $W_i$ ) are required



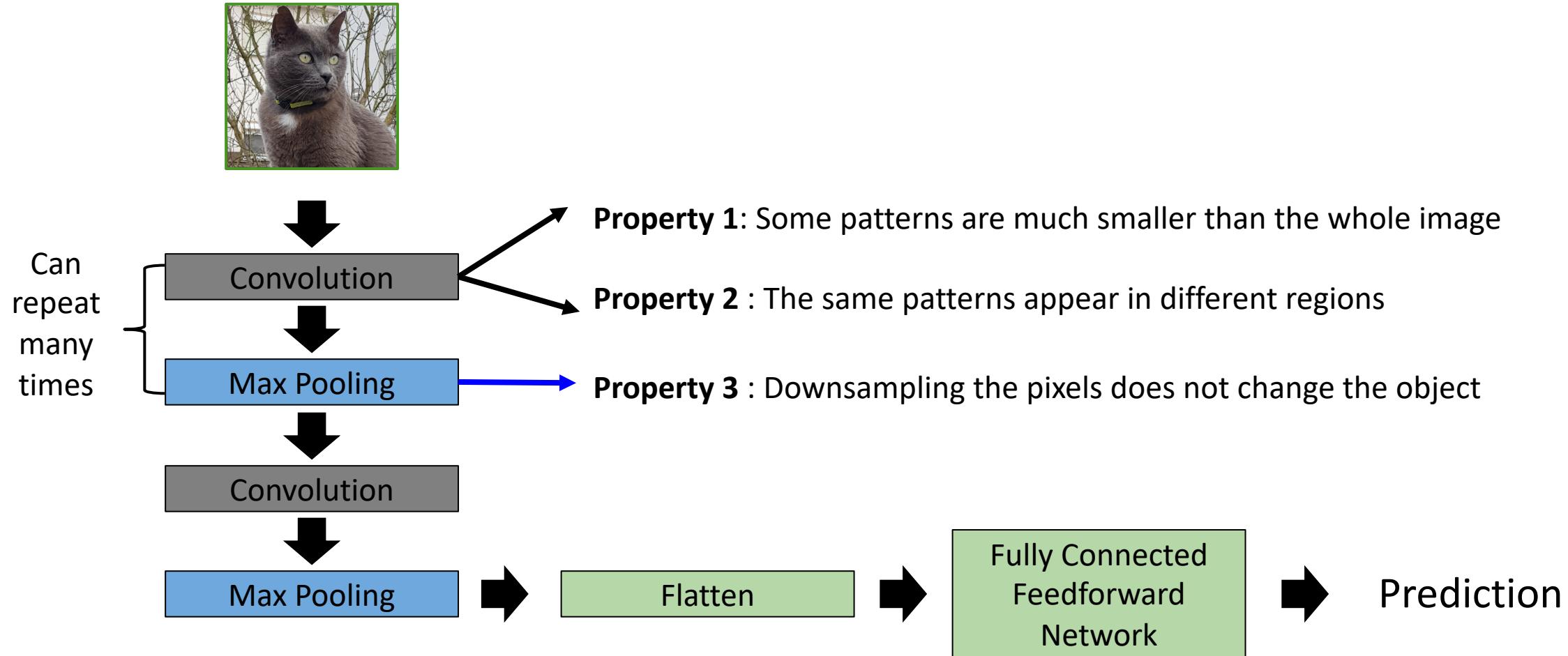
- Similar patterns appear in different regions



- Both networks are flower detectors
- They detect almost **the same** thing
- Therefore, they can use **the same** set of parameters

- **Downsampling (subsampling) does not change the key properties**
  - E.g., images are made **smaller** by downsampling
  - Less parameters for the network to process





## Convolution

- Convolution is the manipulation of two *signals* to form a third
- *Notation:* Convolution is denoted by the star  $*$  operator

$$y[n] = x[n] * h[n] = \sum_{i=-\infty}^{\infty} x[i]h[n-i]$$

$$y[n] = x[m, n] * h[m, n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x[i, j]h[m - i, n - j]$$

0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

6 x 6 image

Apply **small filters** to detect small patterns

Each filter has a size of **3 x 3**

-1	1	-1
-1	1	-1
-1	1	-1

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

⋮ ⋮ ⋮ ⋮ ⋮

- **Note:** Only the size of the filters is specified; the weights are initialised to arbitrary values before the start of training.
- The weights of the **filters are learnt** through the CNN training process

- **Key Parameters**

- **Filter size** – defines the height and width of the filter kernel
  - E.g., a filter kernel of size would have nine weights
- **Stride** – determines the number of steps to move in each spatial direction while performing convolution.
- **Padding** –appends zeroes to the boundary of an image to control the size of the output of convolution
  - When we convolve an image of a specific size by a filter, the resulting image is generally smaller than the original image

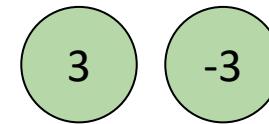
stride = 1

0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 1



Compute the **dot product** between the filter and a small 3 x 3 chunk of the image

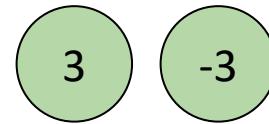
stride = 2

0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 1



We set stride = 1 below

Compute the **dot product** between the filter and a small 3 x 3 chunk of the image

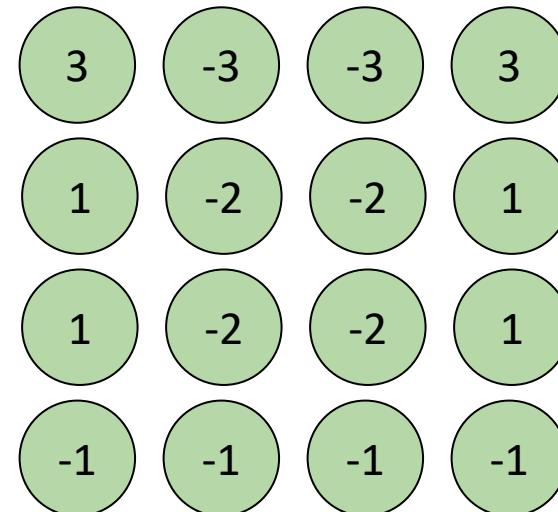
stride = 1

0	<b>1</b>	0	0	<b>1</b>	0
0	<b>1</b>	0	0	<b>1</b>	0
0	<b>1</b>	0	0	<b>1</b>	0
<b>1</b>	0	0	0	0	<b>1</b>
0	<b>1</b>	0	0	<b>1</b>	0
0	0	<b>1</b>	<b>1</b>	0	0

6 x 6 image

-1	<b>1</b>	-1
-1	<b>1</b>	-1
-1	<b>1</b>	-1

Filter 1

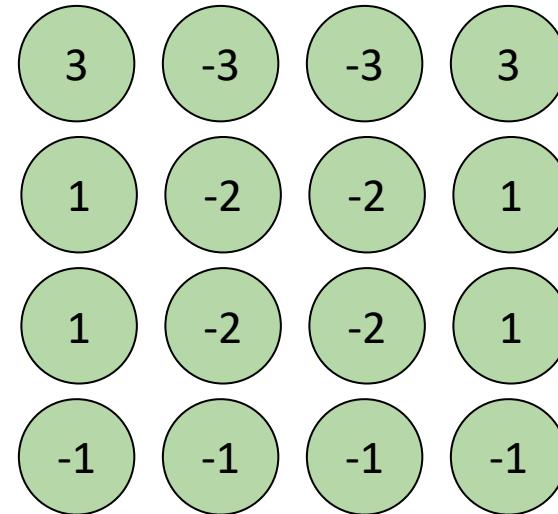


4 x 4 image

stride = 1, filter size = 3

0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

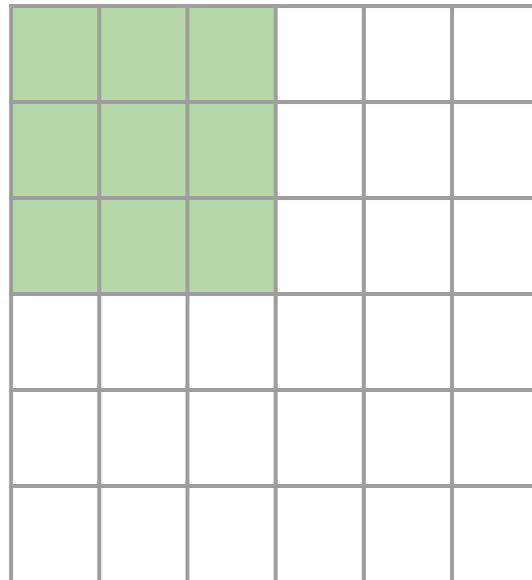
6 x 6 image



4 x 4 image

$$\text{output size: } (6 - 3) / 1 + 1 = 4$$

filter size = F



N x N image

output size:  $(N - F) / \text{stride} + 1$

for example:  $N = 6, F = 3$

$\text{stride} = 1 \rightarrow (6-3)/1 + 1 = 4$

$\text{stride} = 2 \rightarrow (6-3)/2 + 1 = 2.5 : \backslash$

$\text{stride} = 3 \rightarrow (6-3)/3 + 1 = 2$

## zero-padding to the border

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

- $N \times N$  image

For example:  $N = 6$ ,  $F = 3$ , stride = 1

*Without 0-padding:*

output size is  $(6-3)/1 + 1 = 4$

*With 0-padding with 1 pixel border:*

output size is  $(6-3+2\times 1)/1 + 1 = 6$

The output size is then the same as the input!

## zero-padding to the border

0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

$N \times N$  image

In general, stride=1, filters of size  $F \times F$ ,  
then zero-padding with  $(F-1)/2$ ,  
to preserve size spatially

e.g.  $F = 3 \rightarrow$  zero pad with 1 pixel to the border  
 $F = 5 \rightarrow$  zero pad with 2 pixels to the border  
 $F = 7 \rightarrow$  zero pad with 3 pixels to the border

stride = 1

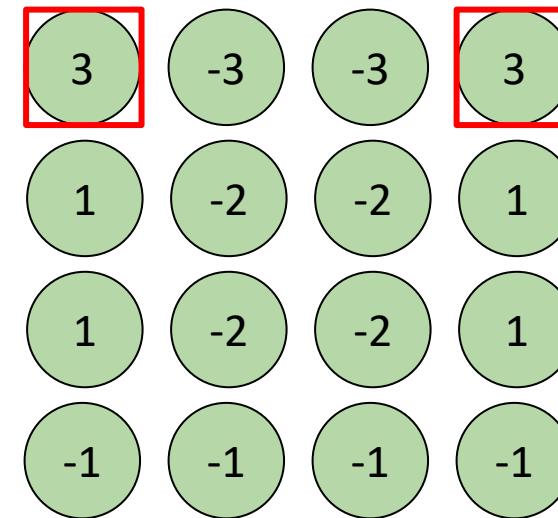
0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filter 1

detect a vertical line



The same pattern in **different locations** are detected with the same filter

# Convolutional Layers

Björn W. Schuller

stride = 1

0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0

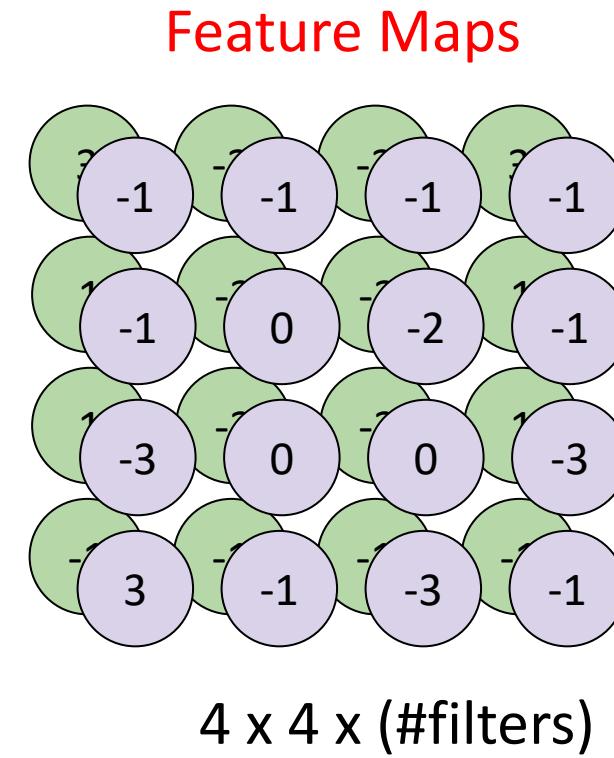
6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

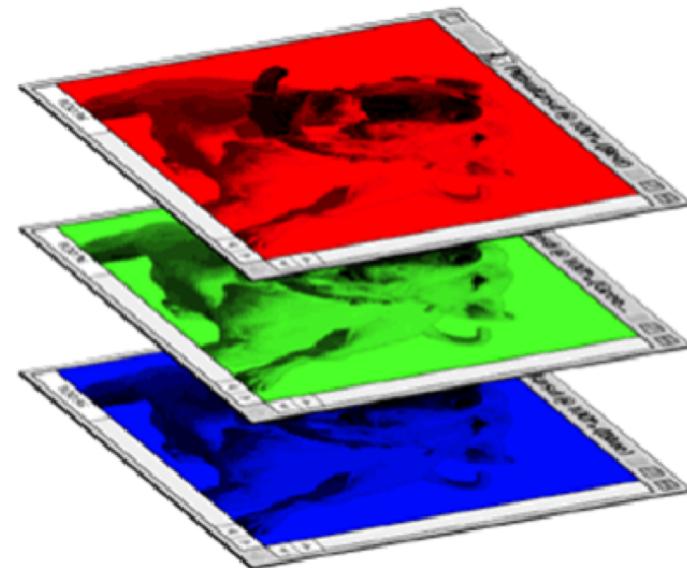
1	-1	-1
-1	1	-1
-1	-1	1

Filter 2



Do the same process for every filter

RGB images



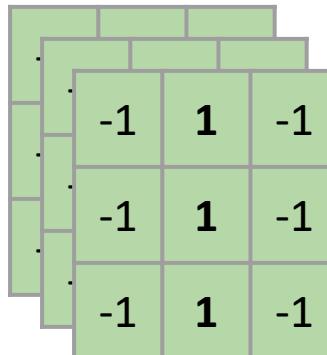
3 channels



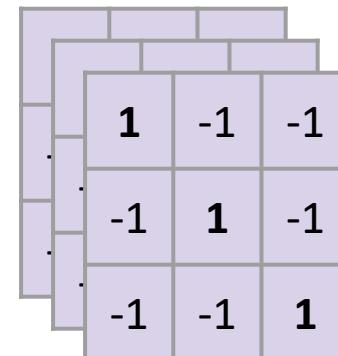
0	1	0	0	1	0	
0	0	1	0	0	1	0
0	0	<b>1</b>	0	0	<b>1</b>	0
1	0	<b>1</b>	0	0	<b>1</b>	0
1	<b>1</b>	0	0	0	0	<b>1</b>
0	<b>1</b>	0	0	<b>1</b>	0	
0	0	<b>1</b>	<b>1</b>	0	0	

$6 \times 6 \times 3$

Filters always extend the full  
depth of the input volume



Filter 1



Filter 2

$3 \times 3 \times 3$

- **Key Parameters:**

- Accepts an input of size  $W_1 \times H_1 \times D_1$
- Requires 4 hyperparameters:
  - Number of filters  $K$
  - Size of the filters  $F$
  - The stride  $S$
  - The amount of zero padding  $P$
- Produce an output of size  $W_2 \times H_2 \times D_2$ , where
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- With parameter sharing, it introduces  $F \times F \times D_1$  weights per filter, for a total of  $(F \times F \times D_1) \times K$  weights and  $K$  biases.

**Common settings:**

$K$ : powers of 2, such as 32, 64, 128, 512

$F = 3, S=1, P=2$

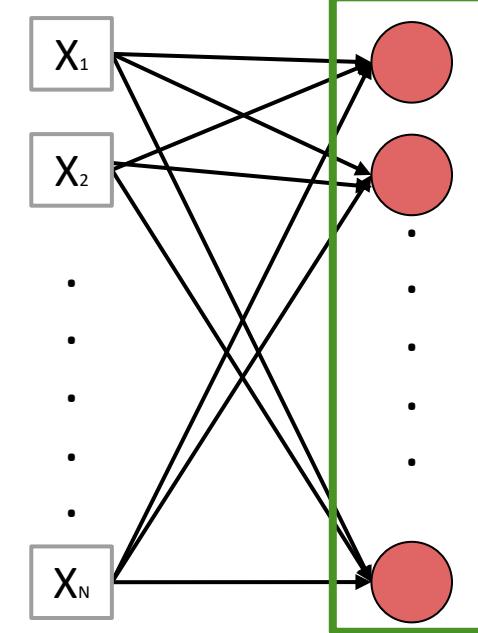
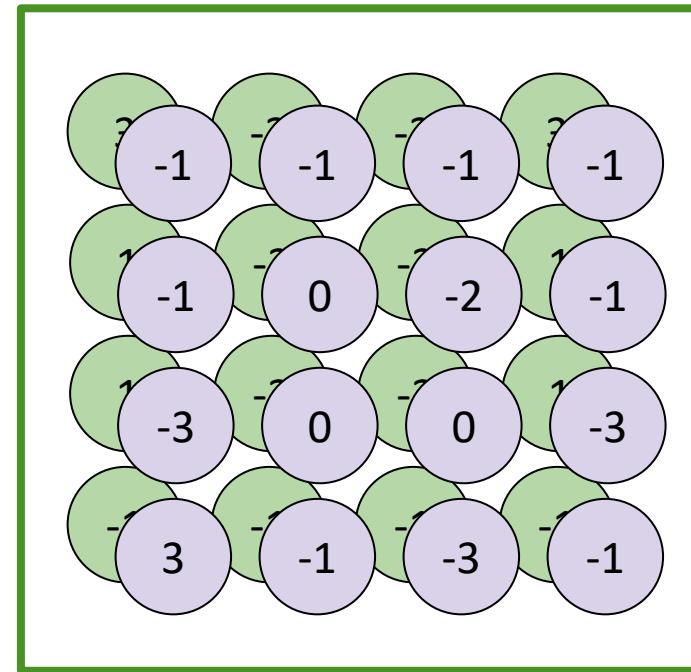
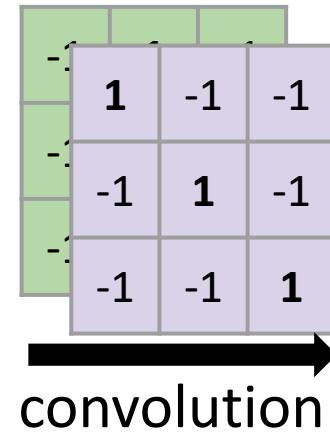
$F = 5, S=1, P=2$

... ...

# Convolution vs Fully Connected

Björn W. Schuller

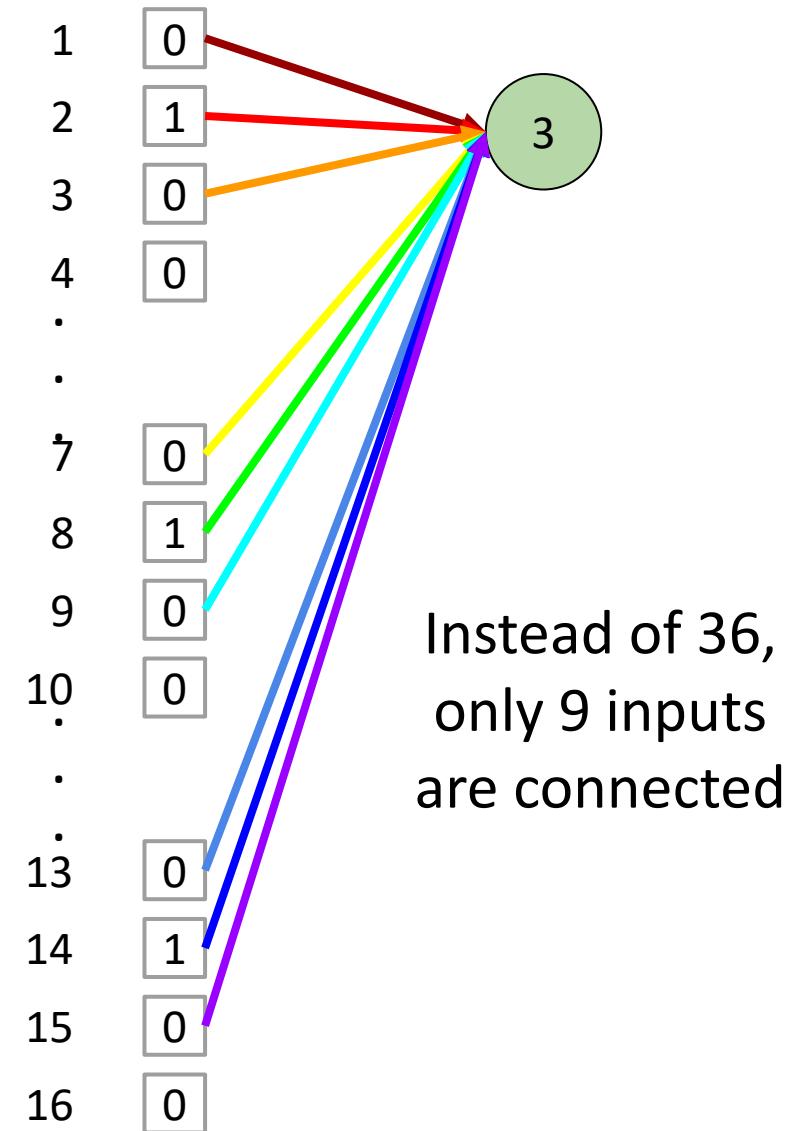
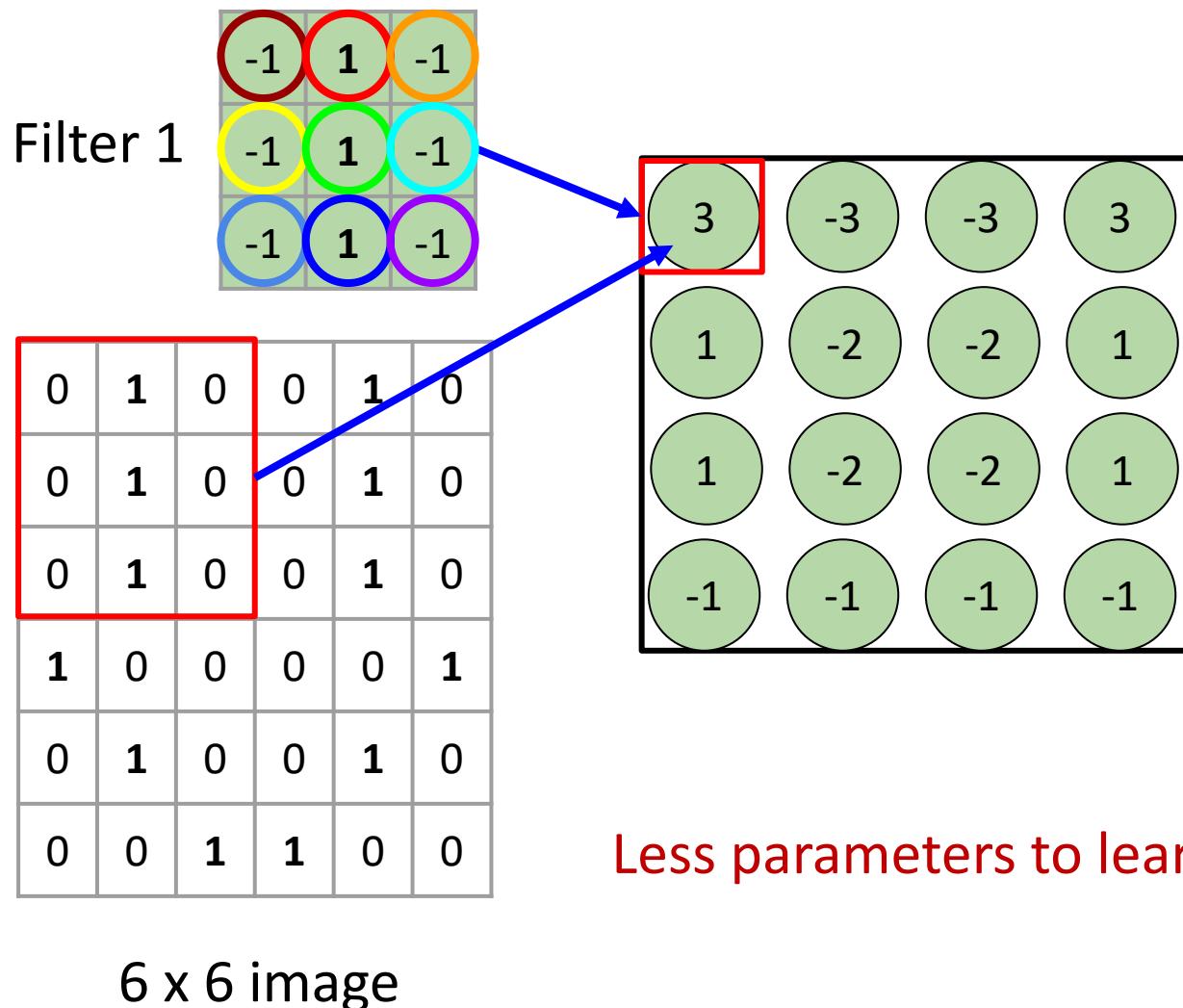
0	1	0	0	1	0
0	1	0	0	1	0
0	1	0	0	1	0
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0



fully connected

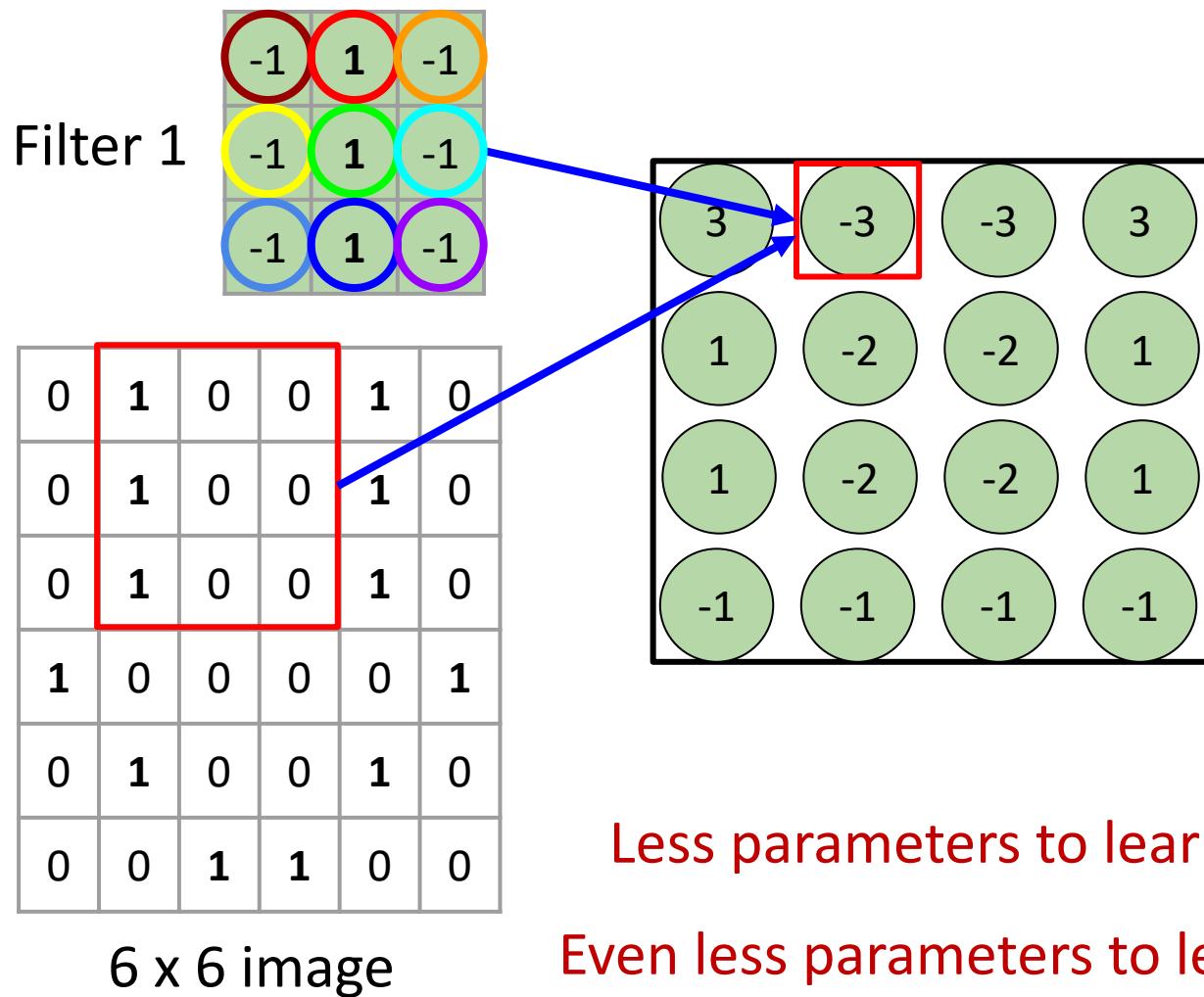
# Convolution vs Fully Connected

Björn W. Schuller



# Convolution vs Fully Connected

Björn W. Schuller



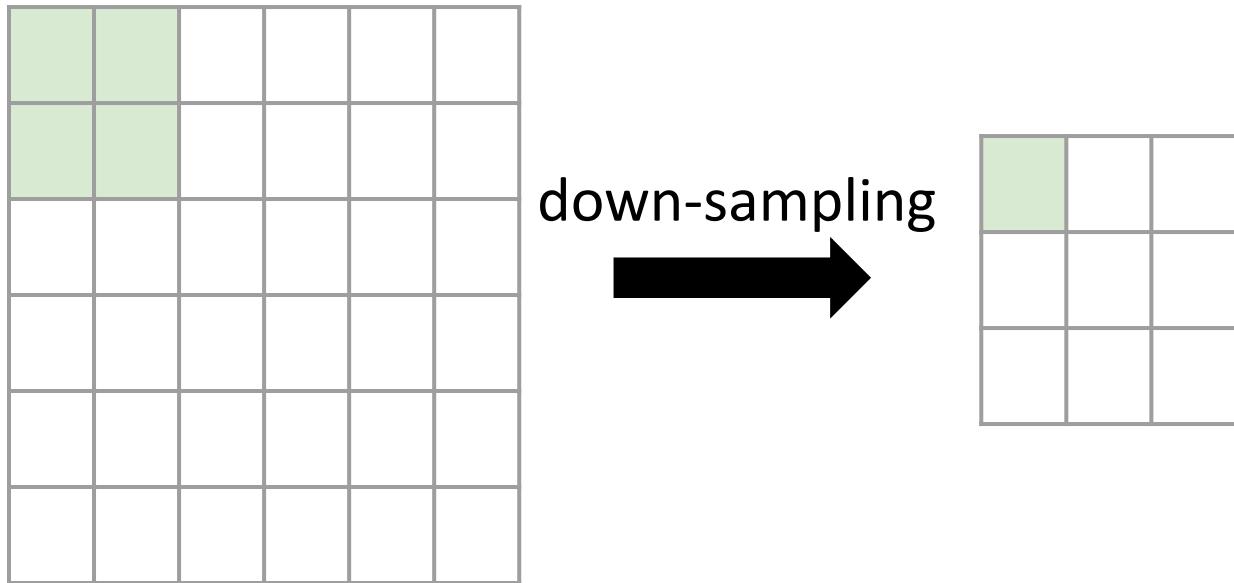
- **The core idea**

- Instead of having a large, dense linear layer with a connection from every input to every output, we have lots of small convolutional layers
  - Convolutional layers usually with fewer inputs and a single output
- The result is a smaller subset of kernel predictions, which are used as input to the next layer.
- Convolutional layers usually have many kernels

- **The core idea**

- Each kernel to learn a particular pattern and then search for the existence of that pattern somewhere in the image
- A single, small set of weights can train over a much larger set of training examples
- This changes the ratio of weights to datapoints on which those weights are being trained
- This has a powerful impact on the network, drastically reducing its ability to overfit to training data and increasing its ability to generalise

Pooling layers are usually present after a convolutional layer.  
They provide a **down-sampled** version of the convolution output.



In this example, a 2x2 region is used as input of the pooling.  
There are different types of pooling, the most used is **max pooling**.

# Max Pooling

Björn W. Schuller

Filter 1

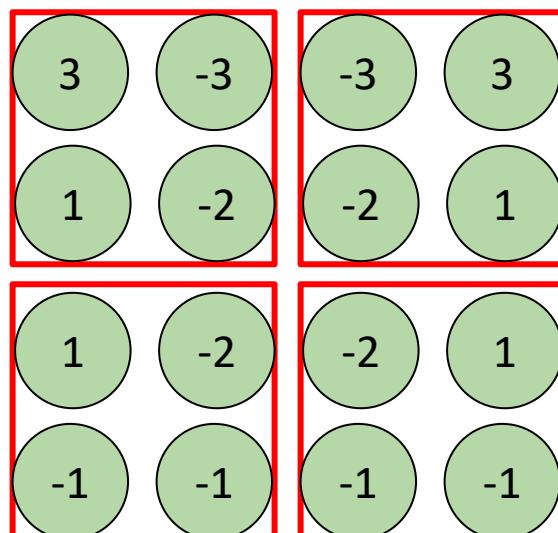
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

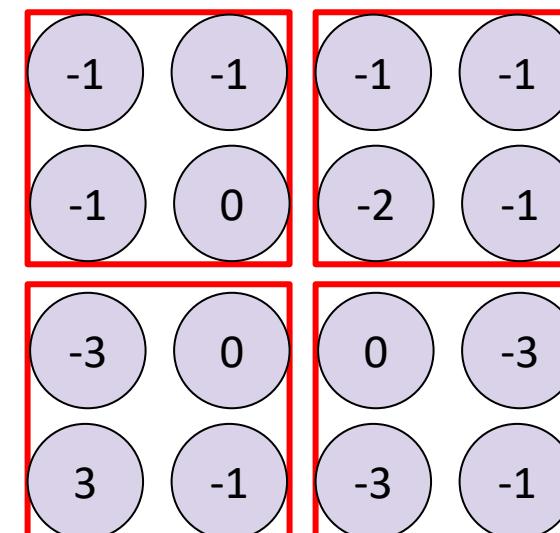
1	-1	-1
-1	1	-1
-1	-1	1

Pooling size =  $2 \times 2$

Stride = 2



Feature map 1



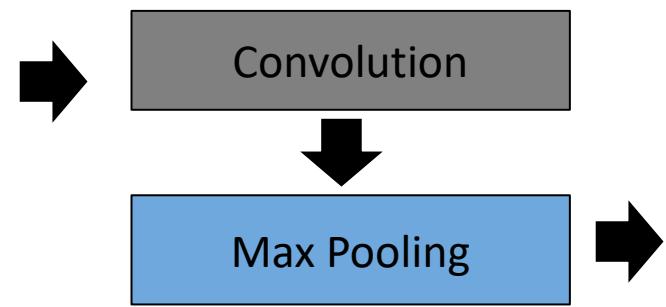
Feature map 2

Operates over each  
feature map  
independently

Invariant to small  
differences in the input

0	<b>1</b>	0	0	<b>1</b>	0
0	<b>1</b>	0	0	<b>1</b>	0
0	<b>1</b>	0	0	<b>1</b>	0
<b>1</b>	0	0	0	0	<b>1</b>
0	<b>1</b>	0	0	<b>1</b>	0
0	0	<b>1</b>	<b>1</b>	0	0

6 x 6 image



each filter is a channel

2 feature maps  
each of size **2 x 2**

Smaller and more manageable

## Key Parameters:

- Accepts an input of size  $W_1 \times H_1 \times D_1$
- Requires 2 hyperparameters:
  - Size of the filters  $F$
  - The stride  $S$
- Produce an output of size  $W_2 \times H_2 \times D_2$ , where
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- It introduces **zero** learnable parameters since it computes a fixed function of the input.

**Common settings:**

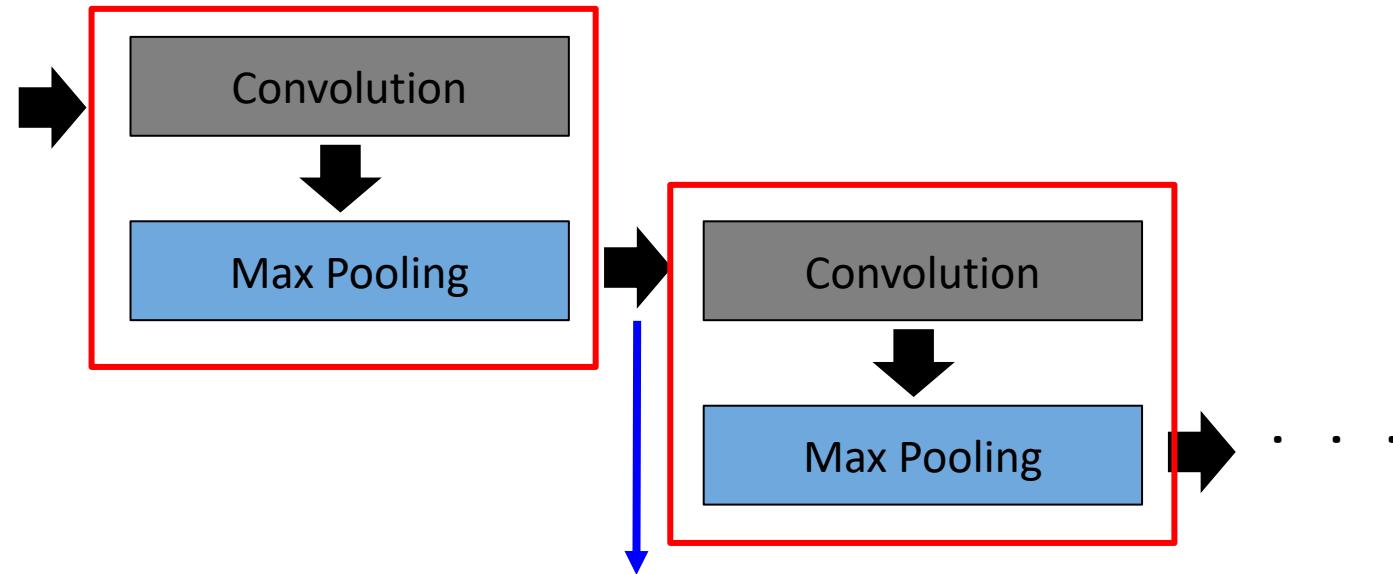
$F = 2, S=2$

$F = 3, S=2$

... ...



**Can be repeat many times**

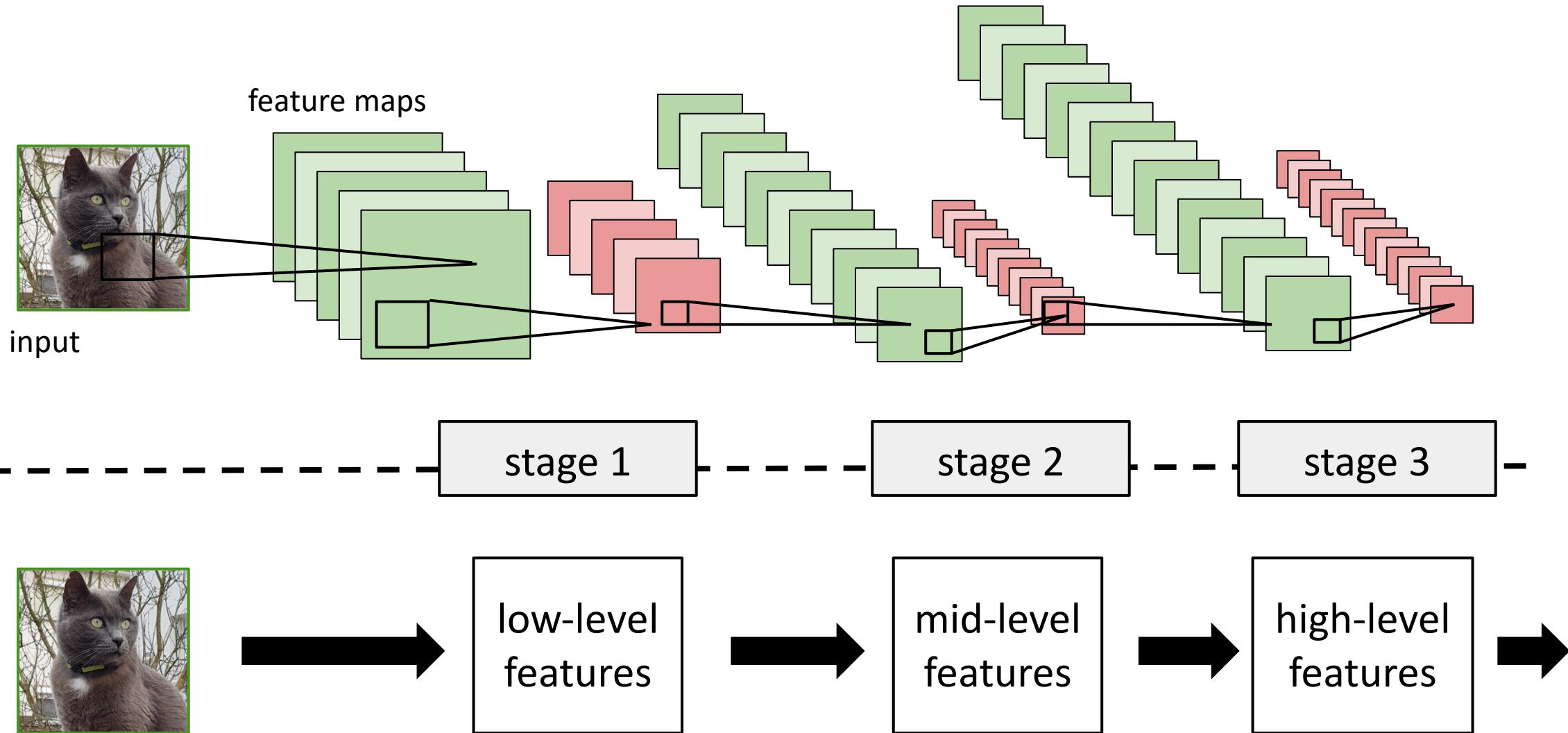


Output can be regarded as new images:

- Smaller than the original images
- The depth of new images is the number of filters

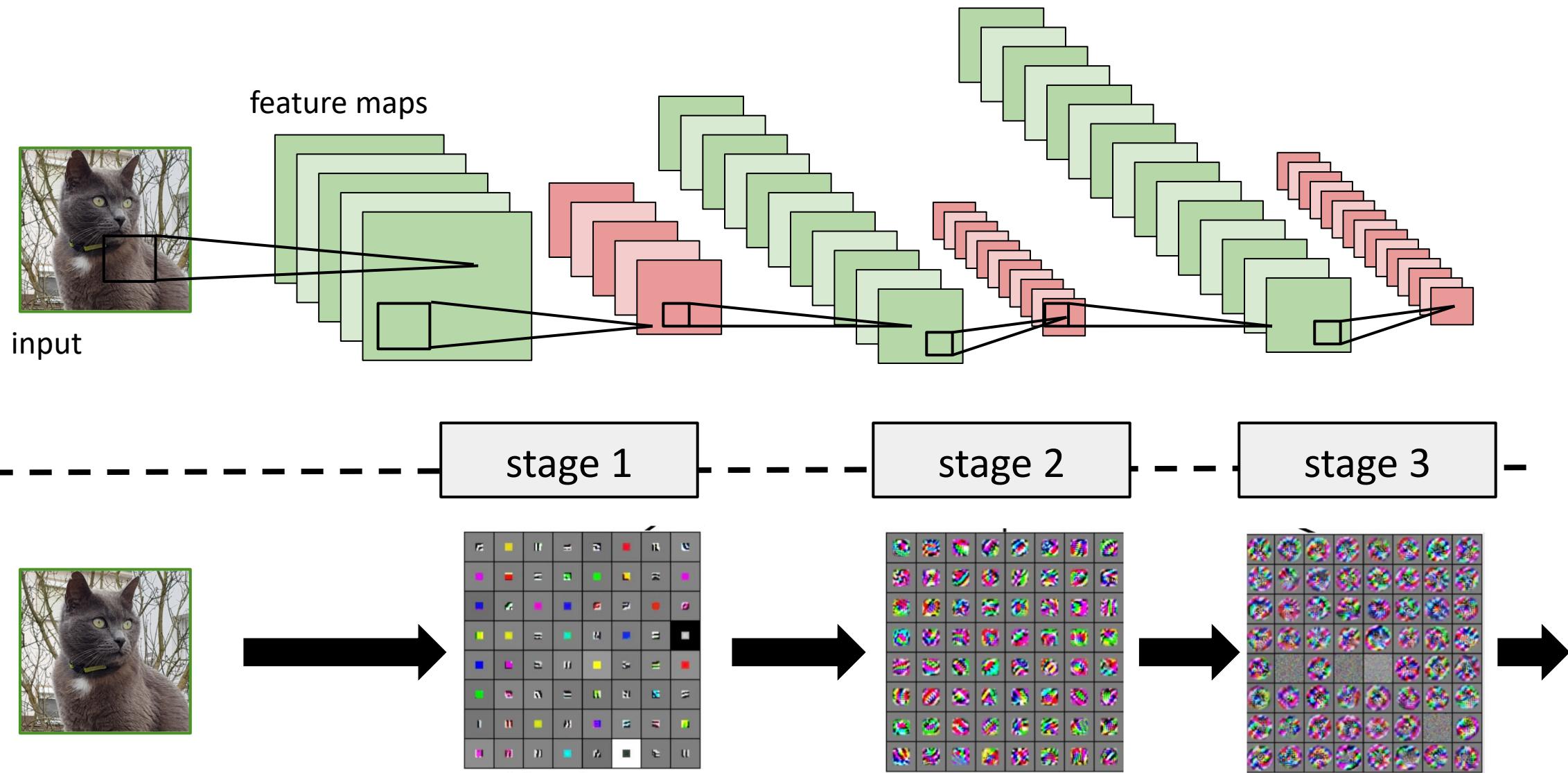
# Representation learning with CNN

Björn W. Schuller



# Representation learning with CNN

Björn W. Schuller



## Transfer Learning

- Features learned by CNNs on large dataset problem, can be helpful for other tasks. It is very common to pre-train a CNN on Imagenet and then use it as a fixed **feature extractor** or as **network initialisation**.

**Feature extractor:** remove the last layer and then use the remaining network to extract representations from hidden layers directly, which can then be utilised as features for other applications.

**Network initialisation:** use pre-trained network and continue its training on your own data and thus fine-tune the weights for your own specific problem, as the training becomes progressively specific to the details of the problem.

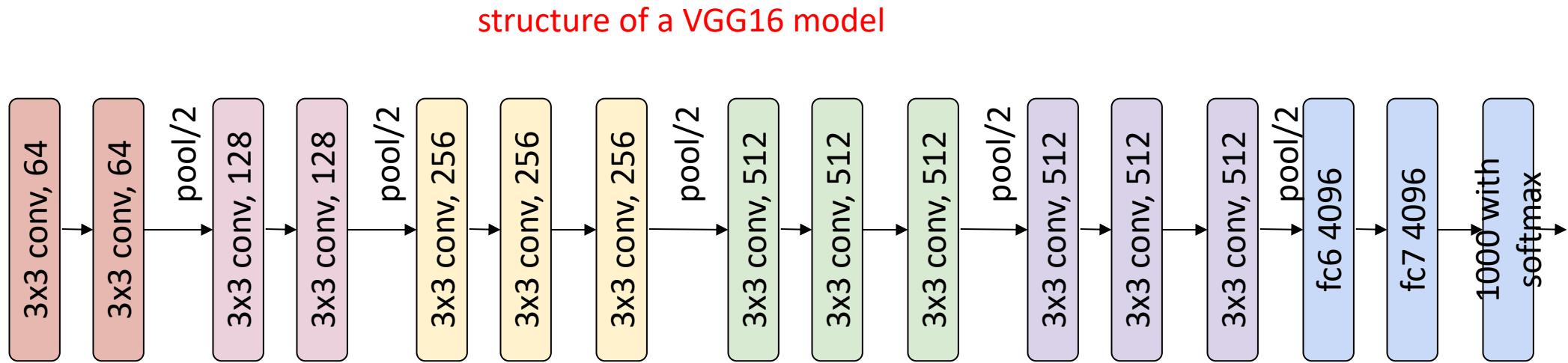
**Data augmentation** can help w/ overfitting and improve results.

For images input, modifications such as

*small rotations, flipping, scaling, adding noise, change brightness, etc.*

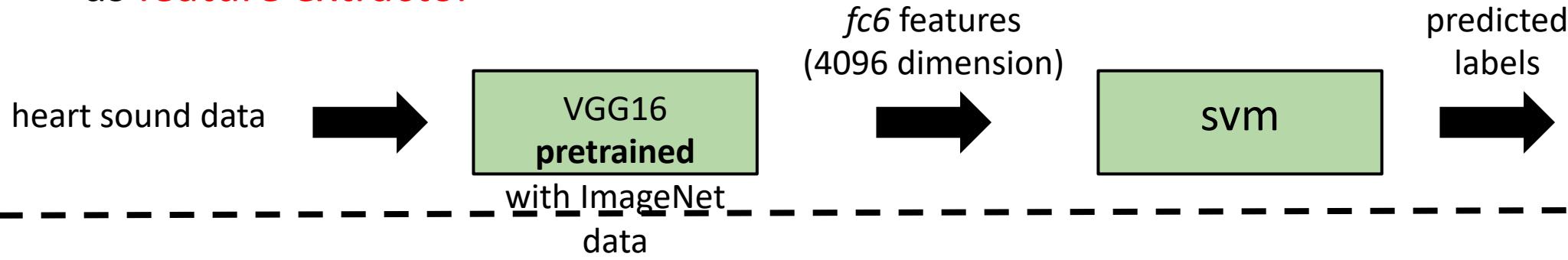


## Learning Image-based representations for heart sound classification

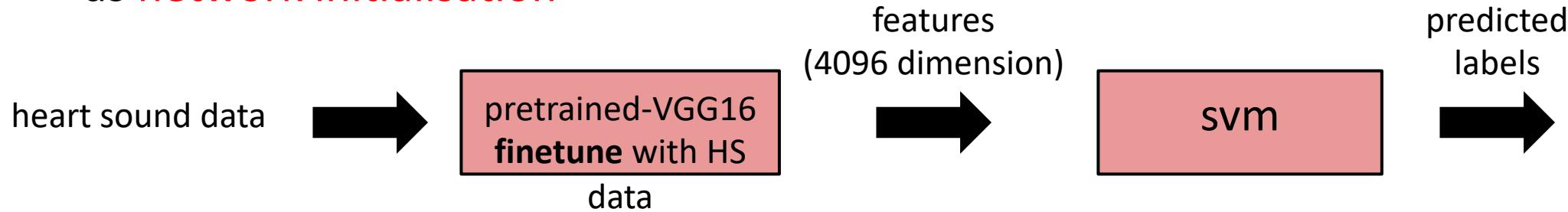


## Learning Image-based representations for Heart Sound Classification

as **feature extractor**



as **network initialisation**



## Learning Image-based representations for Heart Sound Classification

	<i>test set of the Heart Sound database</i>		
<i>performance (%)</i>	<i>sensitivity</i>	<i>specificity</i>	<i>mean accuracy</i>
<b><i>baseline: ComParE + SVM</i></b>	76.8	17.0	46.9
<b><i>pre-trained VGG + SVM</i></b>	24.6	87.1	55.9
<b><i>fine-tuned VGG + SVM</i></b>	24.6	87.8	<b>56.2</b>

*sensitivity:  $Se = TP / (TP + FN)$*

*specificity:  $Sp = TN / (TN + FP)$*

*mean accuracy:  $MAcc = (Se + Sp) / 2$*

- CNNs stack *convolutional layers*, *max pooling layers* and *fully connected feedforward layers* together.
- typical architectures look like

**[(Conv - ReLU)\*N - Pooling?] \*M - (FC - ReLU)\*K - Softmax**

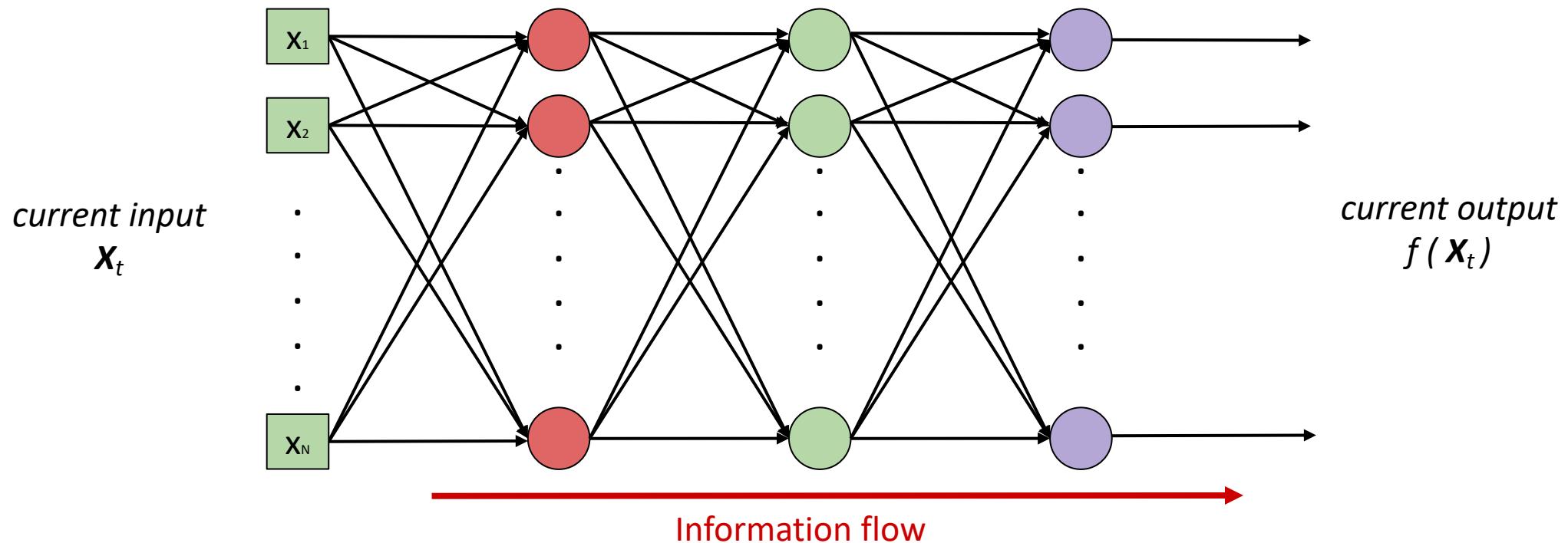
- trend towards smaller filters and deeper architectures
- trend towards getting rid of pooling and feedforward layers, i.e., just convolutional layers

# Recurrent NNs.

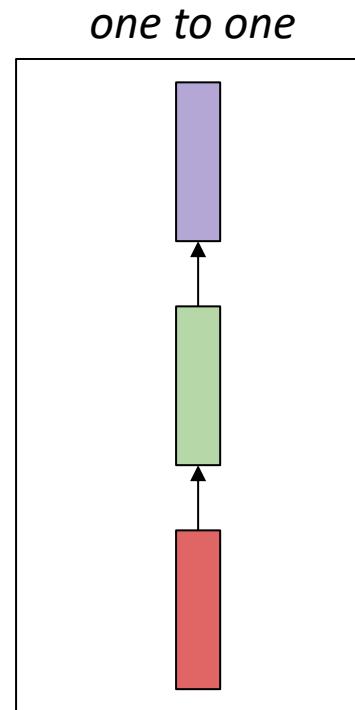


## “Vanilla” Neural Networks

- Information only propagates forward through the network
  - Previous, and future, results do not affect current prediction



## “Vanilla” Neural Networks



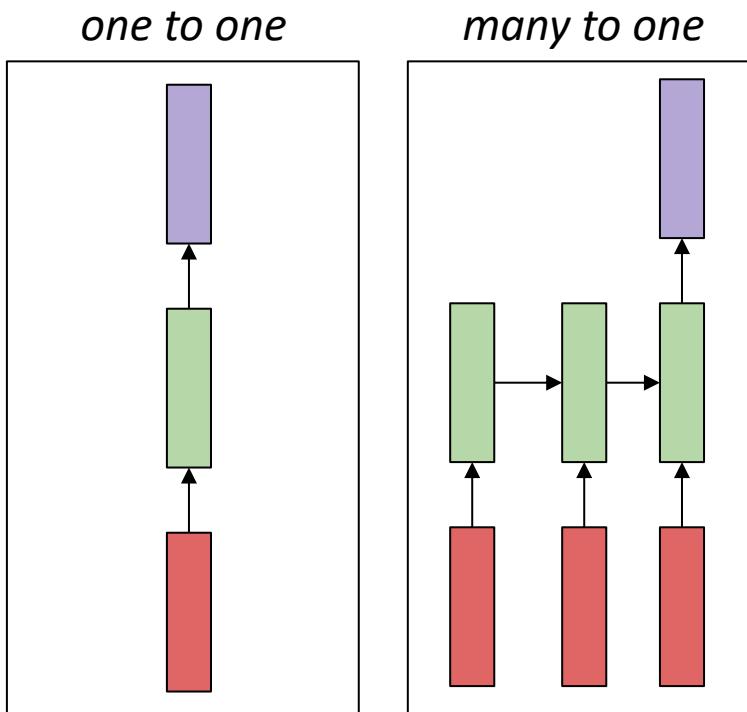
**Feedforward Neural Network:**

- *one input    one output*
- *classifier or regressor*
- ***no context information***

Neural Network needs **memory!**

Can the information/knowledge from **previous inputs** be included?

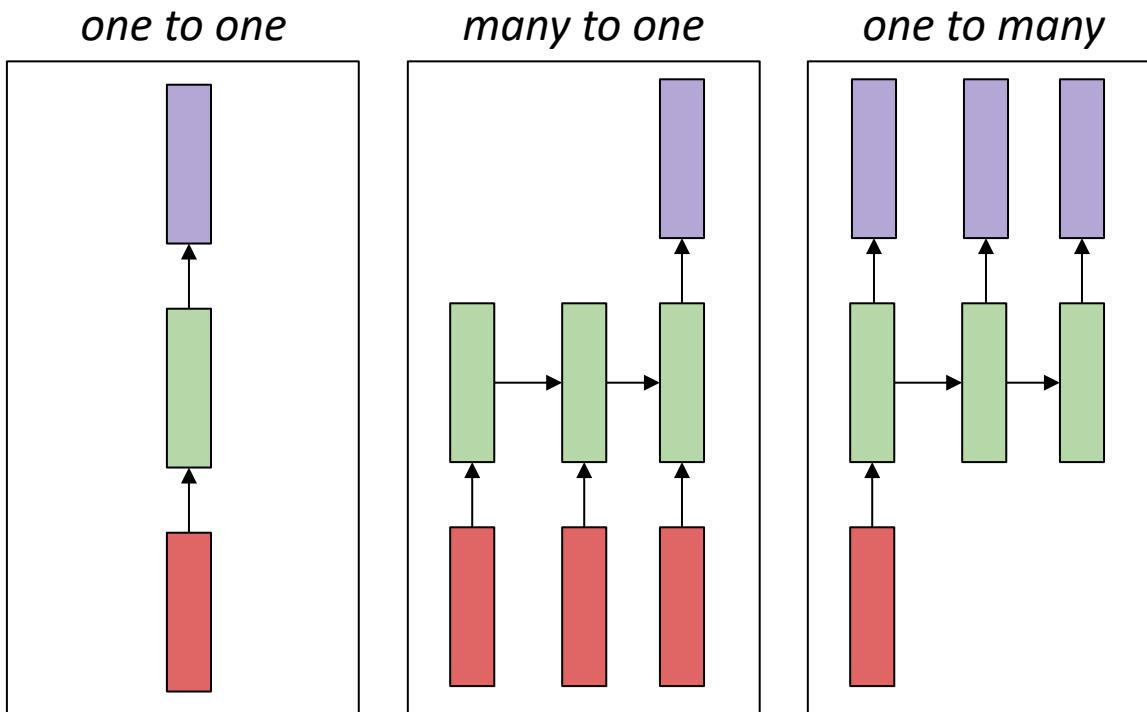
## Processing sequential inputs/outputs



Flexible to handle sequential inputs of varying length

- *speech recognition: speech waves    one word*
- *sentiment classification: words    positive/negative*
- *video classification: images    one category*

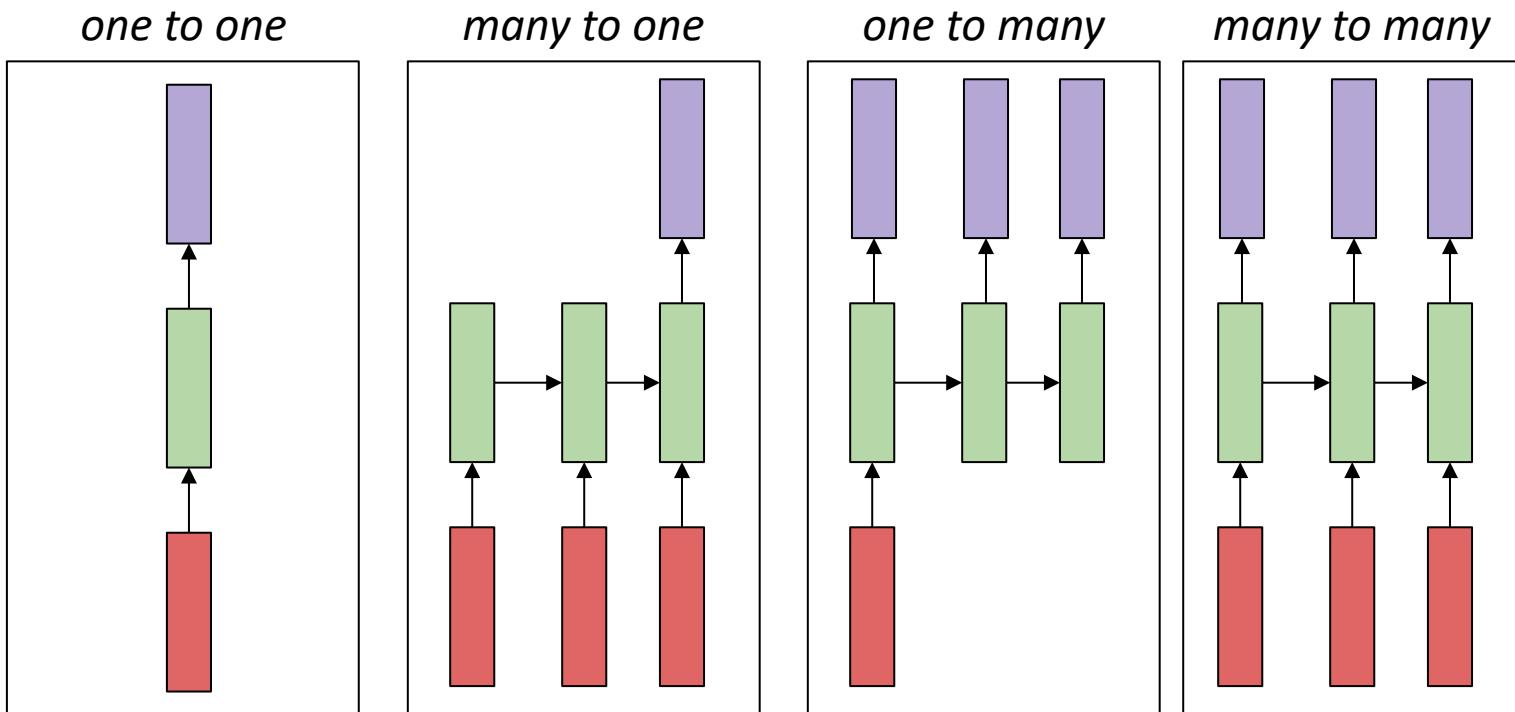
## Processing sequential inputs/outputs



To generate sequential outputs

- *music composition*
- *image captioning*
- *natural text generation*

## Processing sequential inputs/outputs

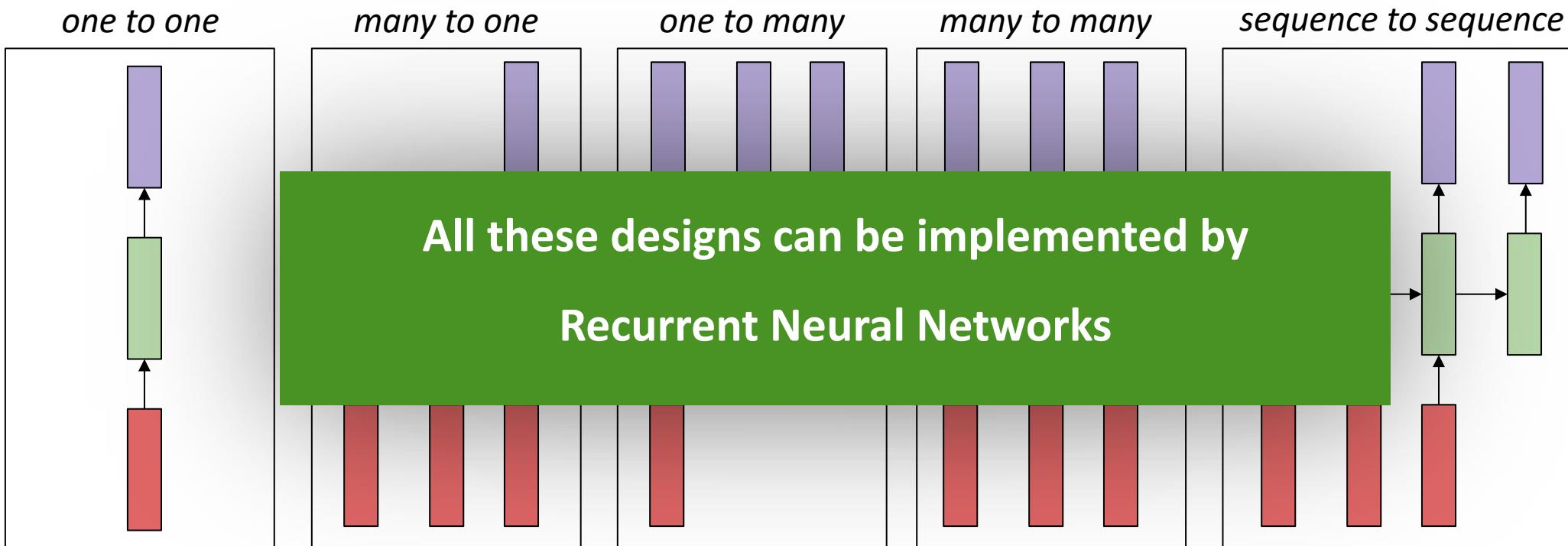


Inputs and outputs of same length

- *continuous emotion prediction*
- *speech enhancement*
- *video classification on frame level*

## Processing sequential inputs/outputs

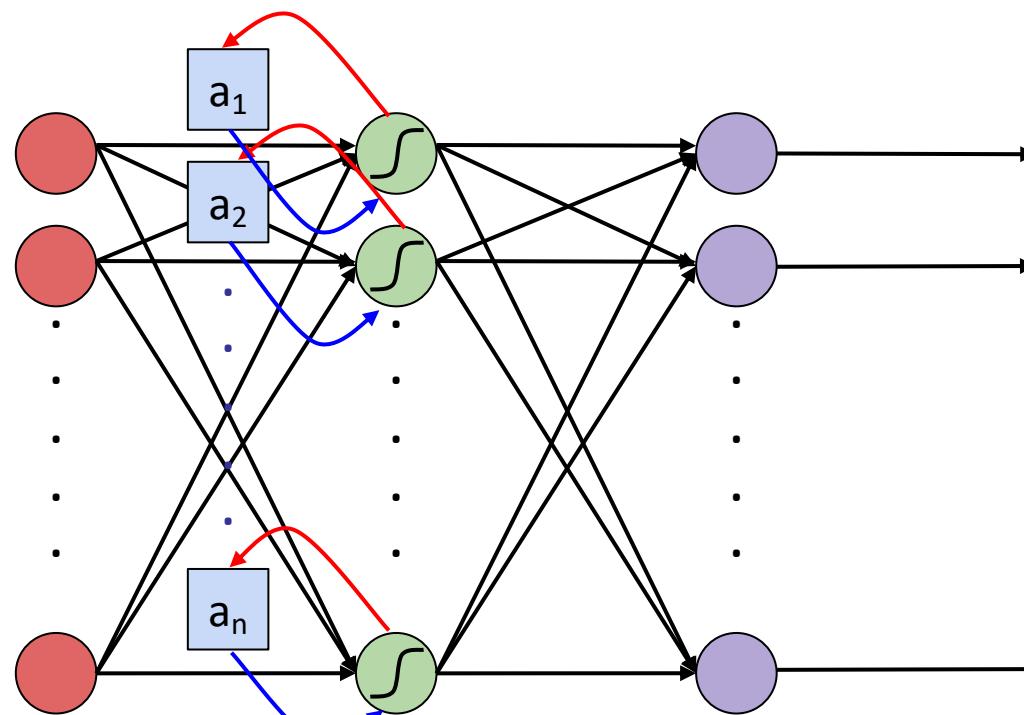
Different lengths:  
➤ *machine translation*  
➤ *question answering*



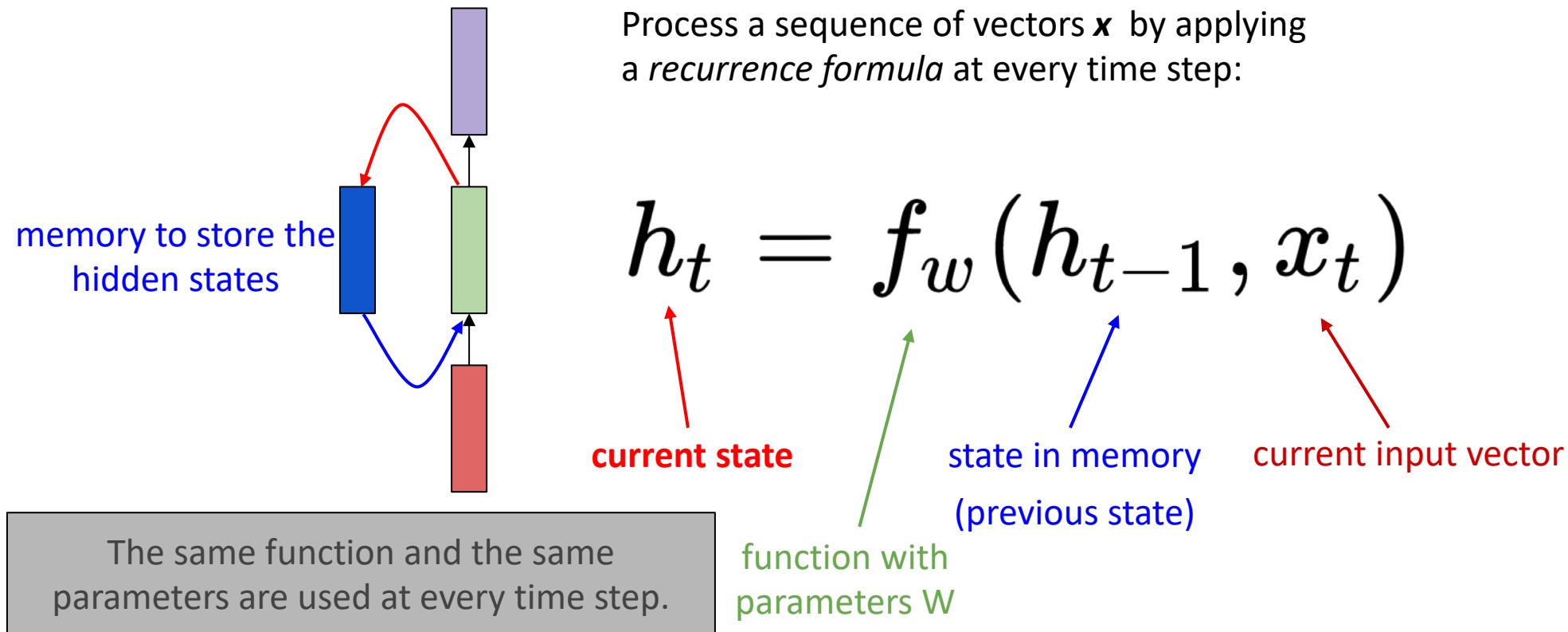
## Inclusion of feedback into the network structure

Output of hidden layer are stored in the memory

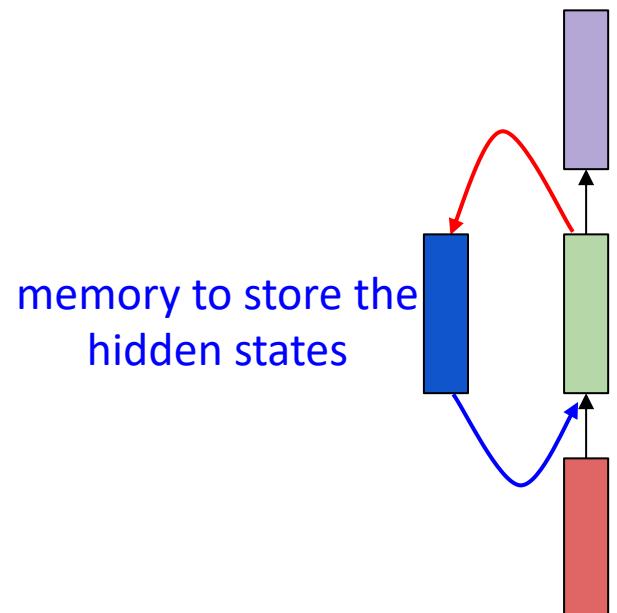
Values in the memory are considered as **additional input** in the next time step



## Inclusion of feedback into the network structure



- Inclusion of feedback into the network structure



Process a sequence of vectors  $x$  by applying a *recurrence formula* at every time step:

$$h_t = f_w(h_{t-1}, x_t)$$

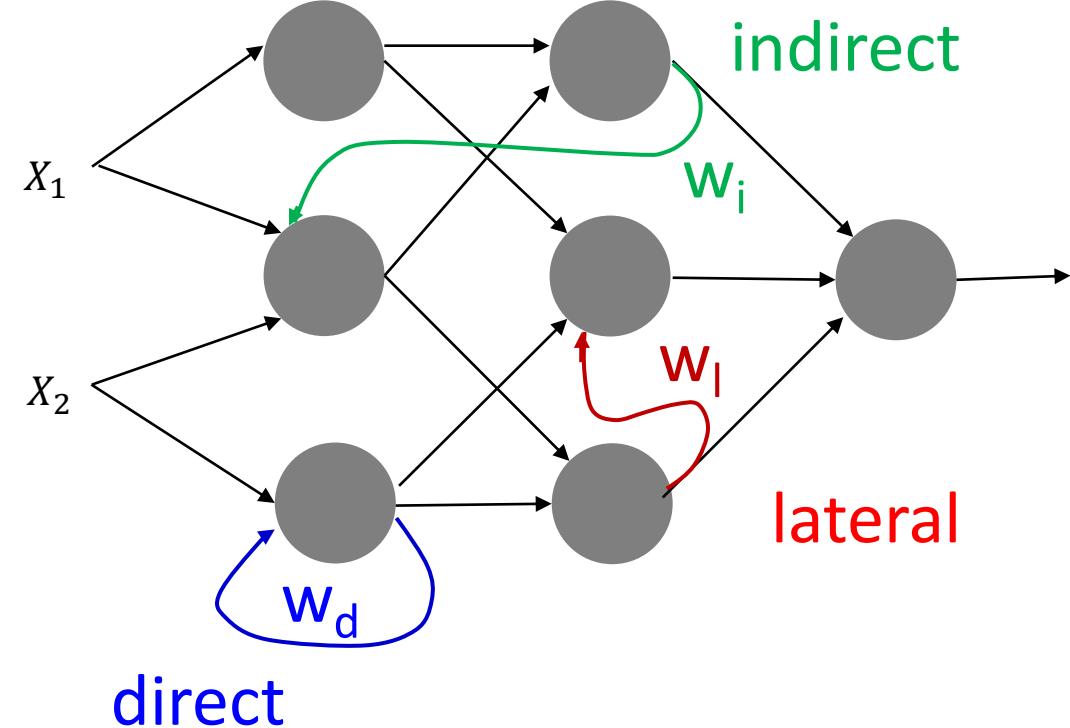


$$h_t = \text{tahn}(W_{hh}h_{t-1} + W_{xh}x_t)$$

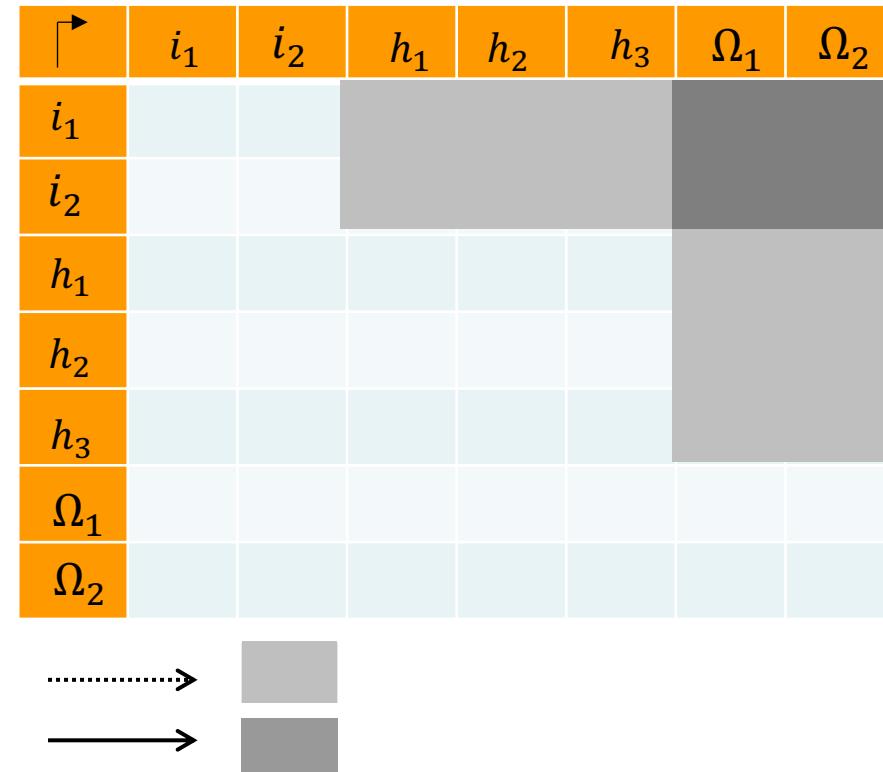
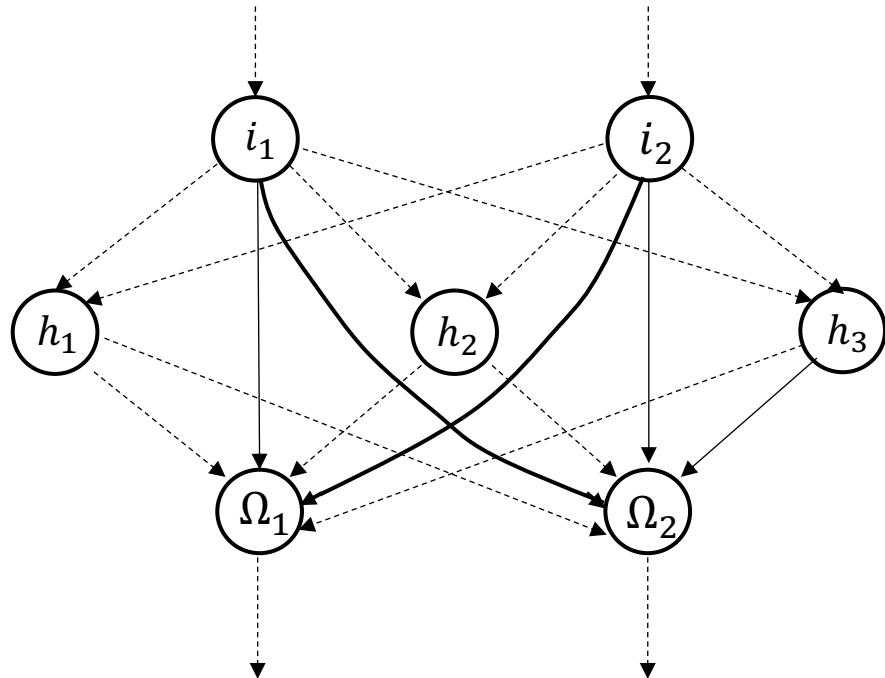
$$y_t = W_{hy}h_t$$

## Recurrent Connections

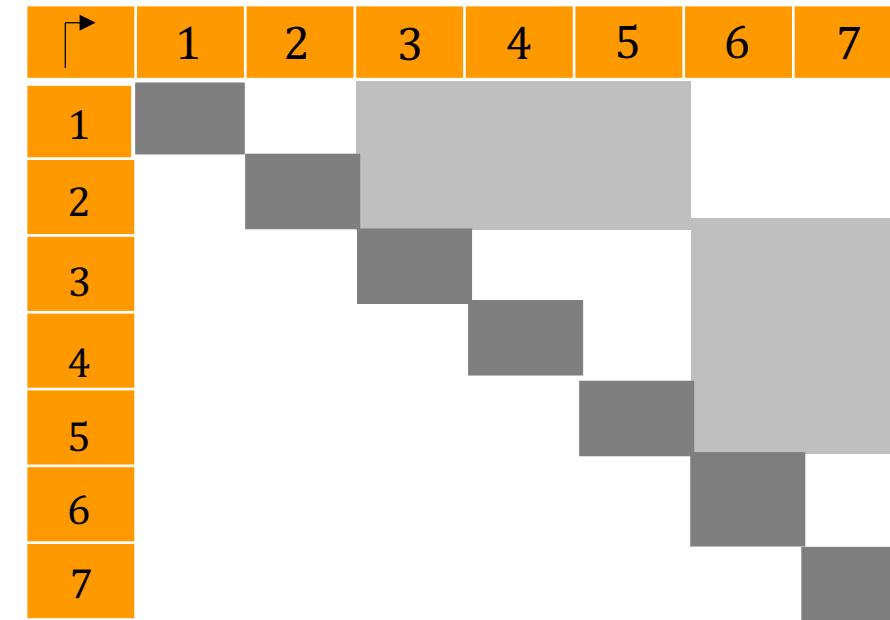
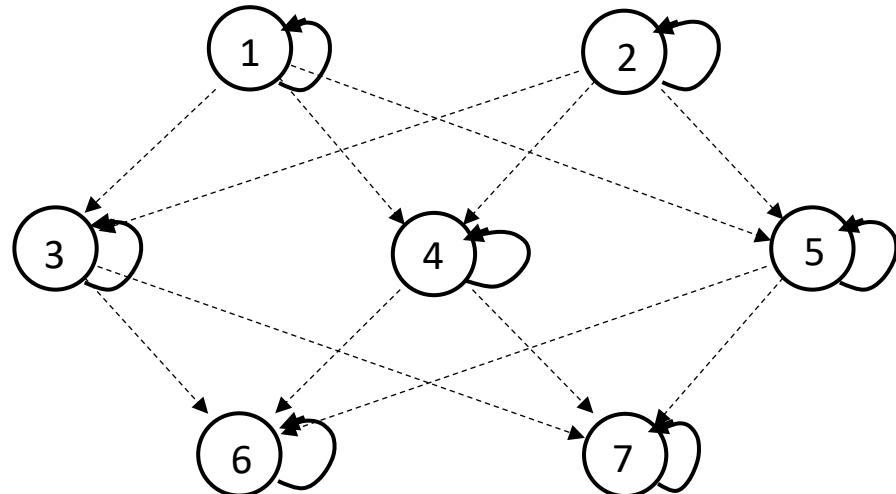
- Direct feedback
  - There exists a connection from a neuron  $j$  to itself with weight  $w_{j,j}$
- Indirect feedback
  - Activity is sent back to the previous layer in the neural network
- Lateral feedback:
  - Connect neurons within a layer



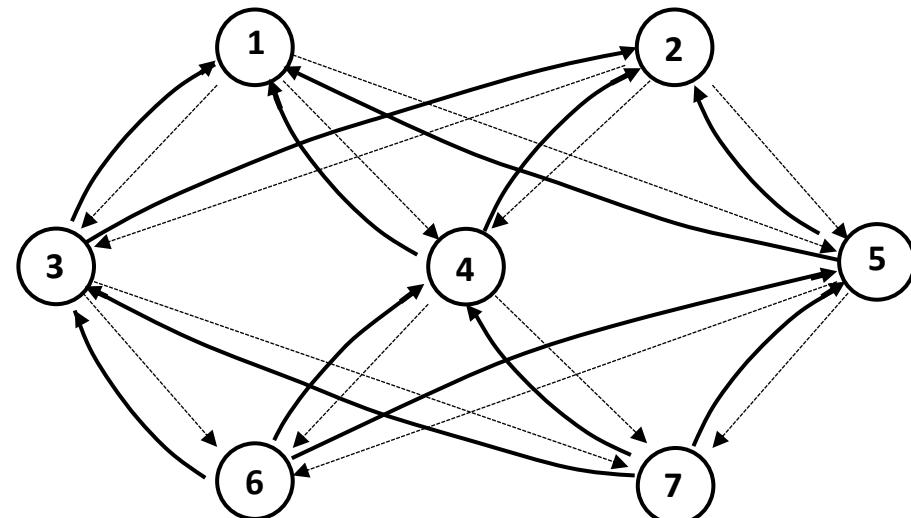
## Feed forward network with shortcut connections:



## Example of direct feedback

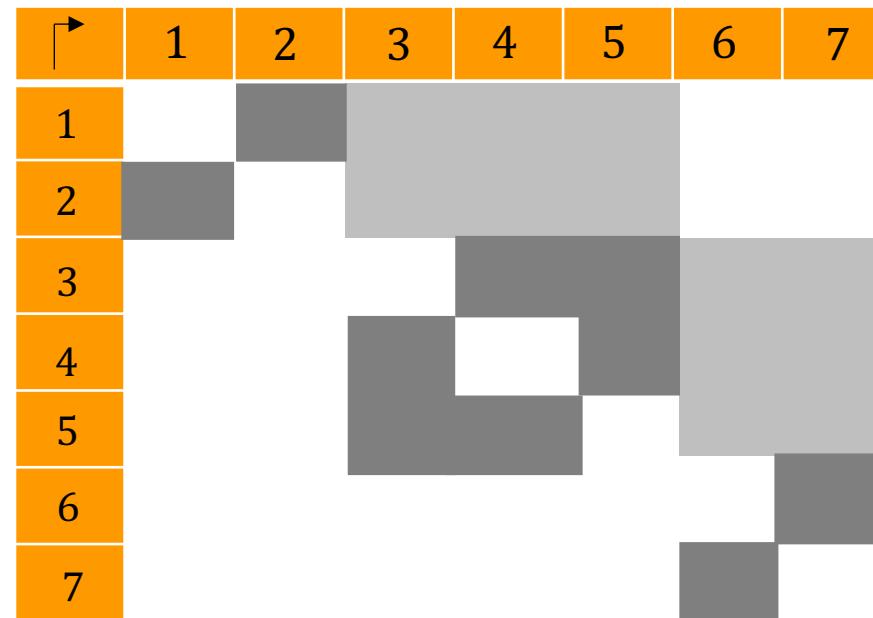
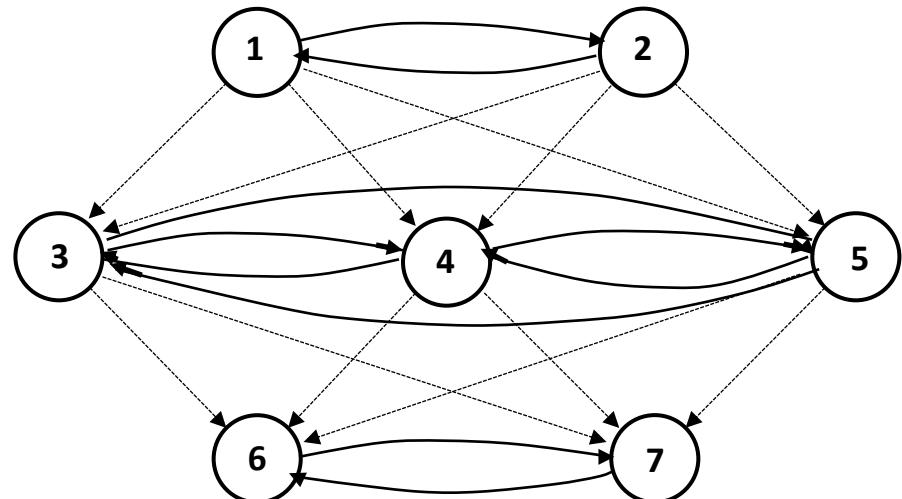


## Example of indirect feedback:

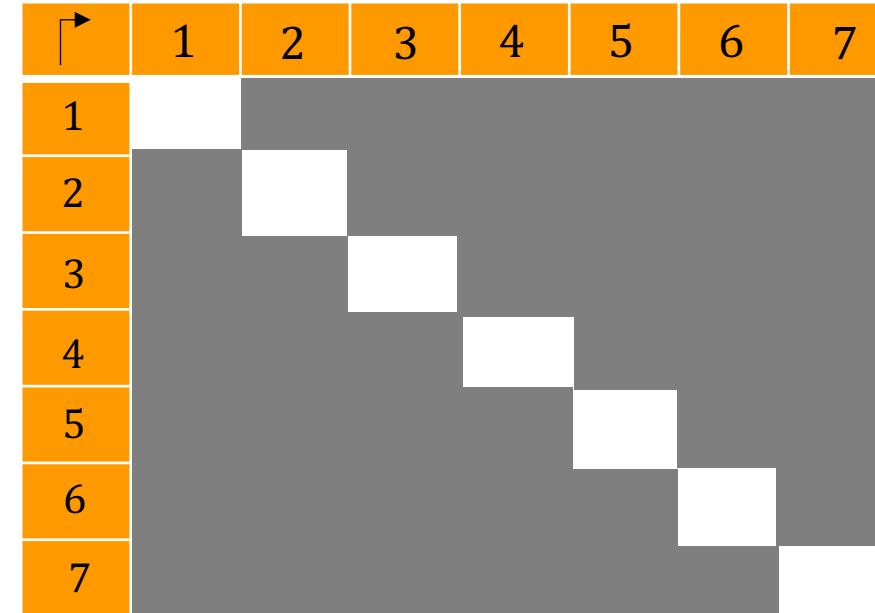
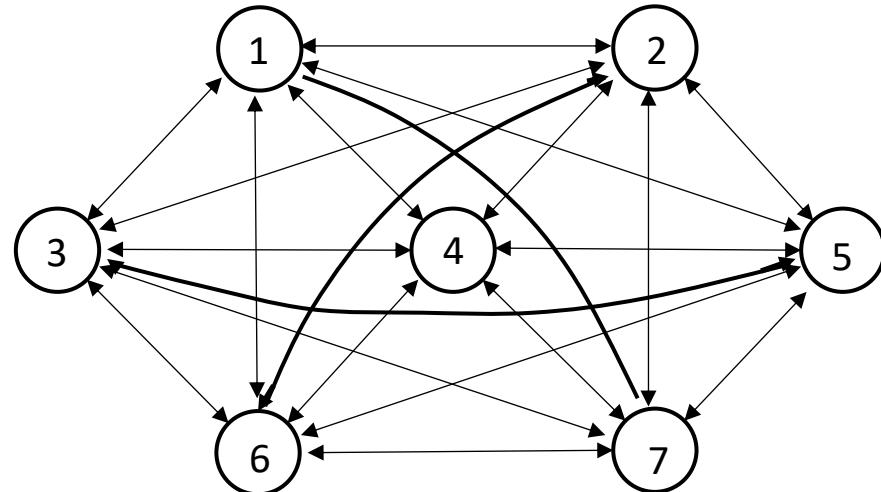


↑	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							

## Lateral feedback:

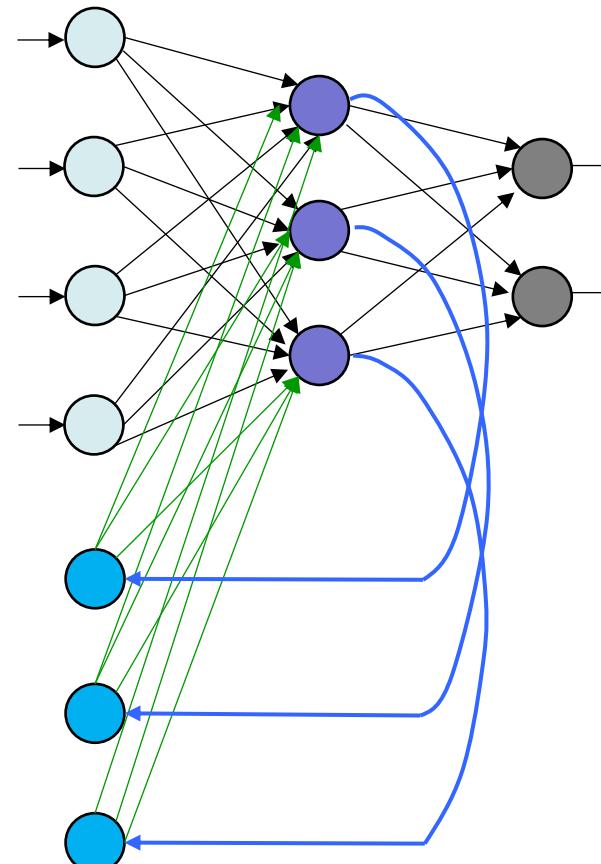


- **Example:**
  - Fully connected network with symmetrical connections

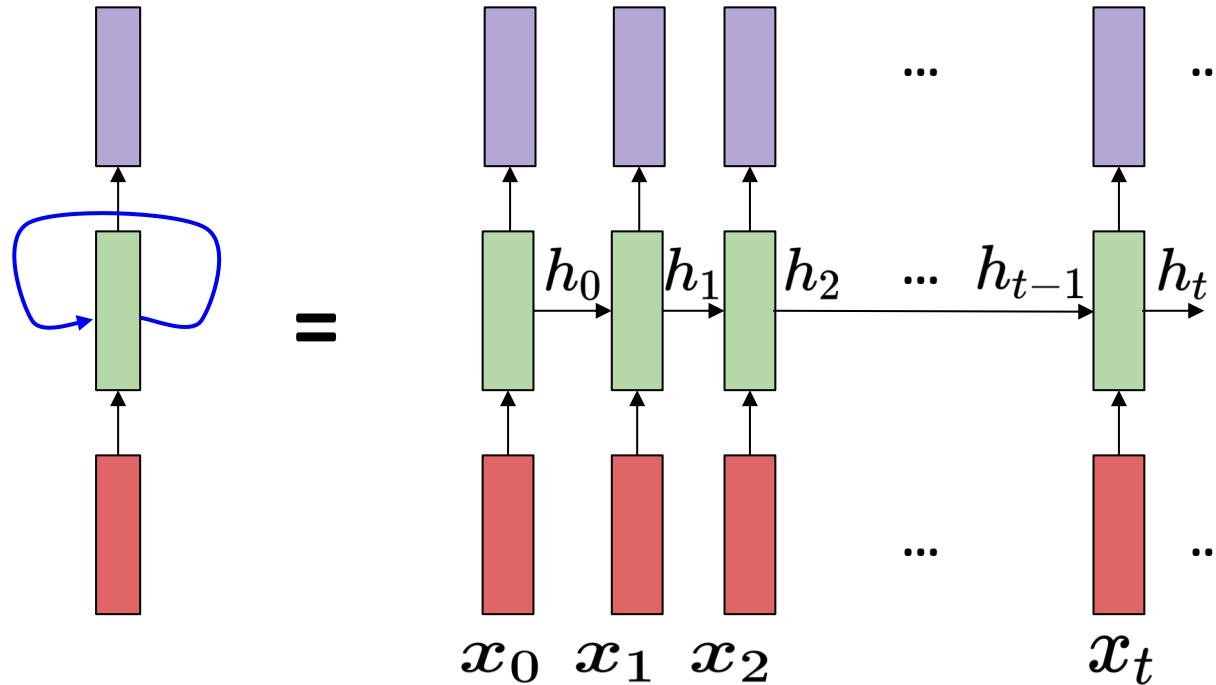


## Network structure

- Number of context units  
= Number of neurons in the corresponding **hidden layer**
- Every **context** unit gets information from exactly one hidden neuron
- **Output** of every context unit to all hidden neurons

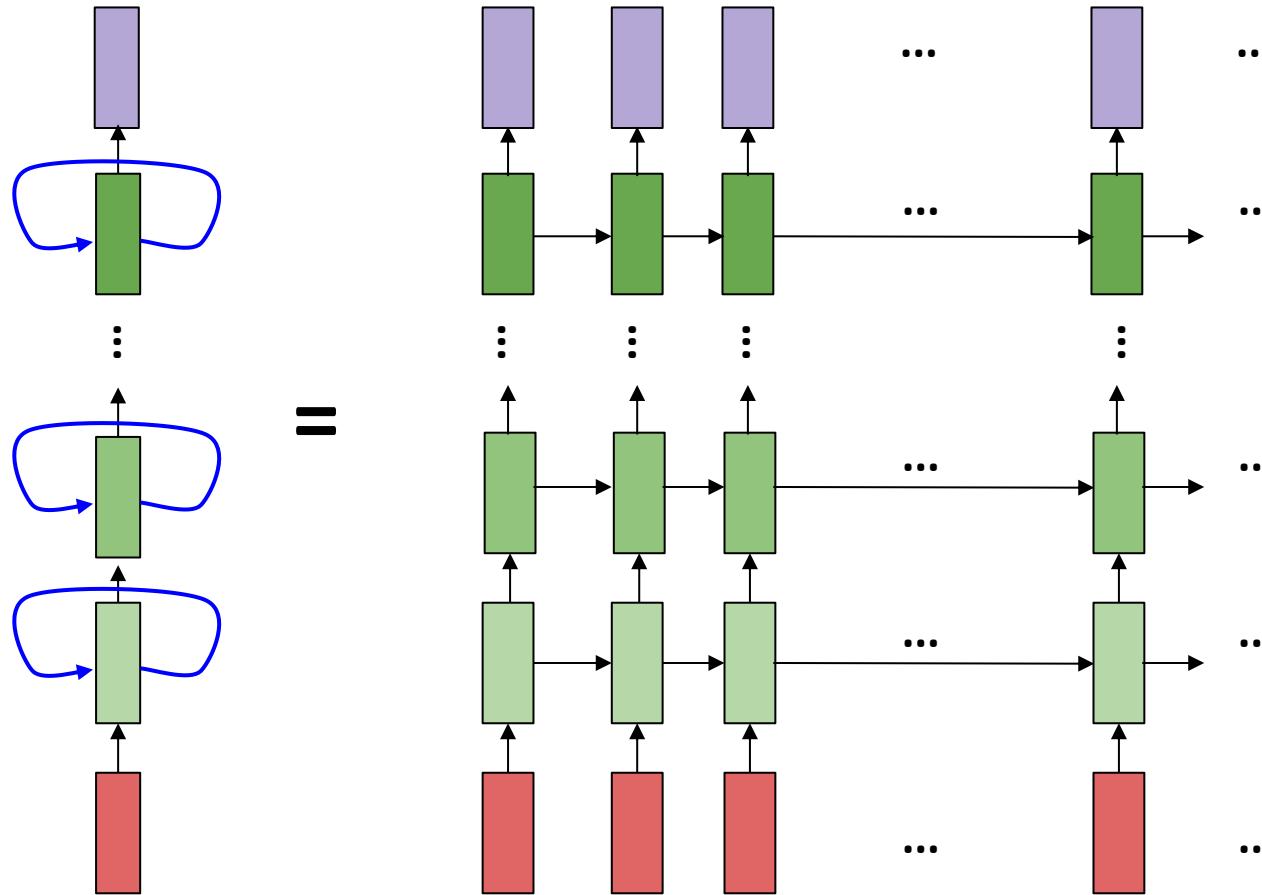


## Unrolled RNN



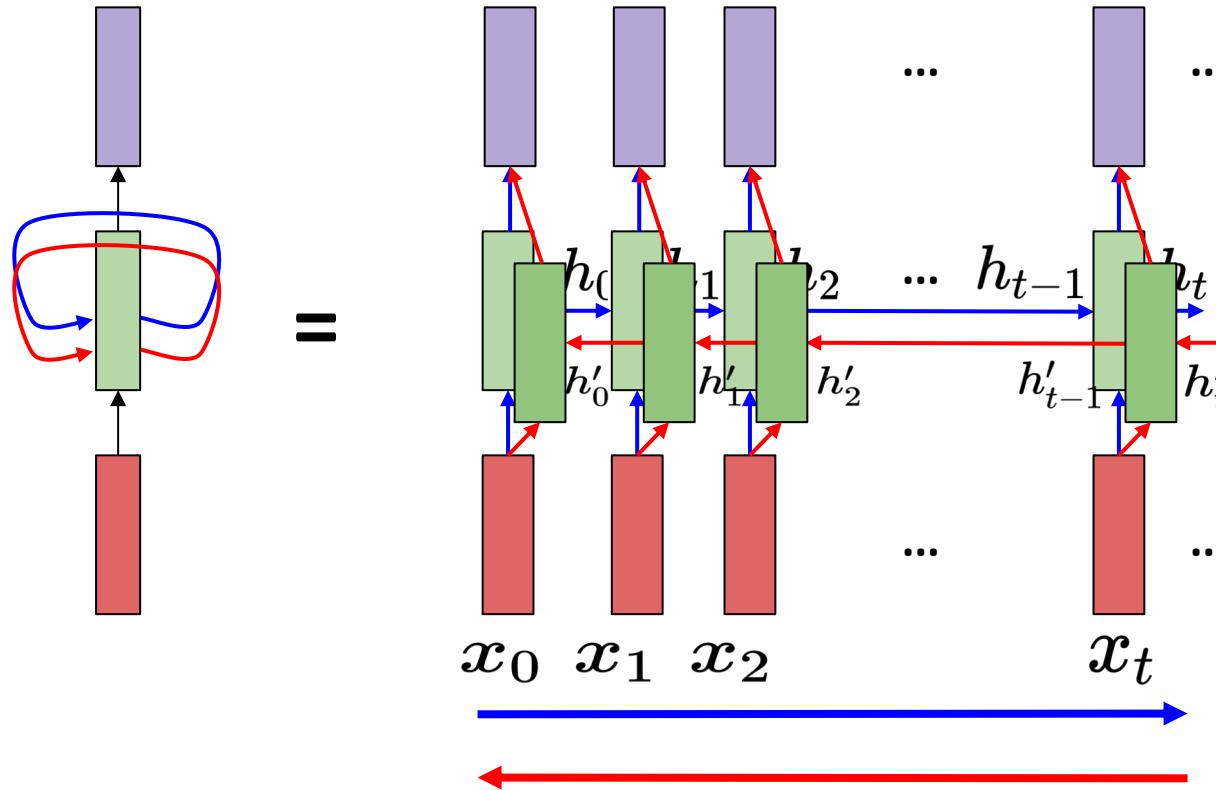
- Reuse the same weight matrix at every time step
- Makes the network easier to train

## Deep RNN



- Not easy to train
- Regularisation helps
  - *Batch normalisation*
  - *Dropout*

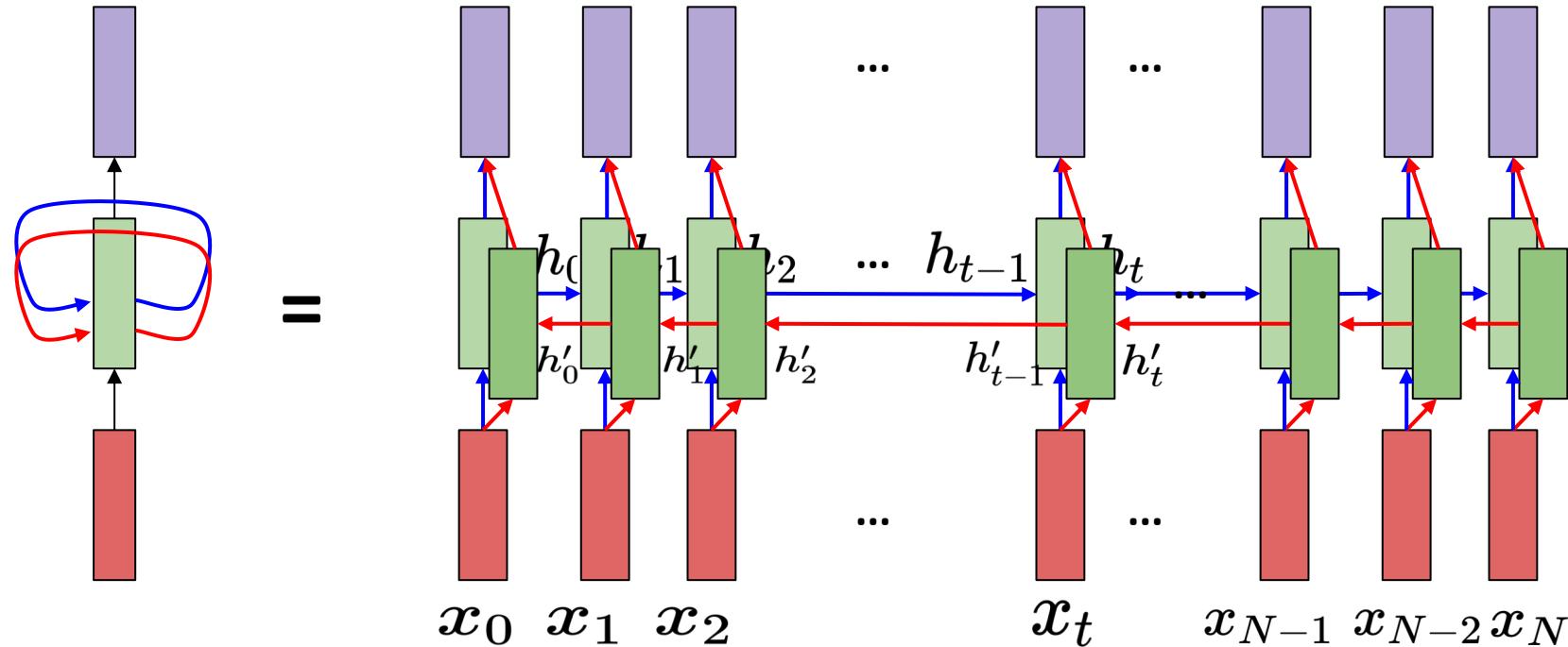
## Bidirectional RNN



Two hidden layers:

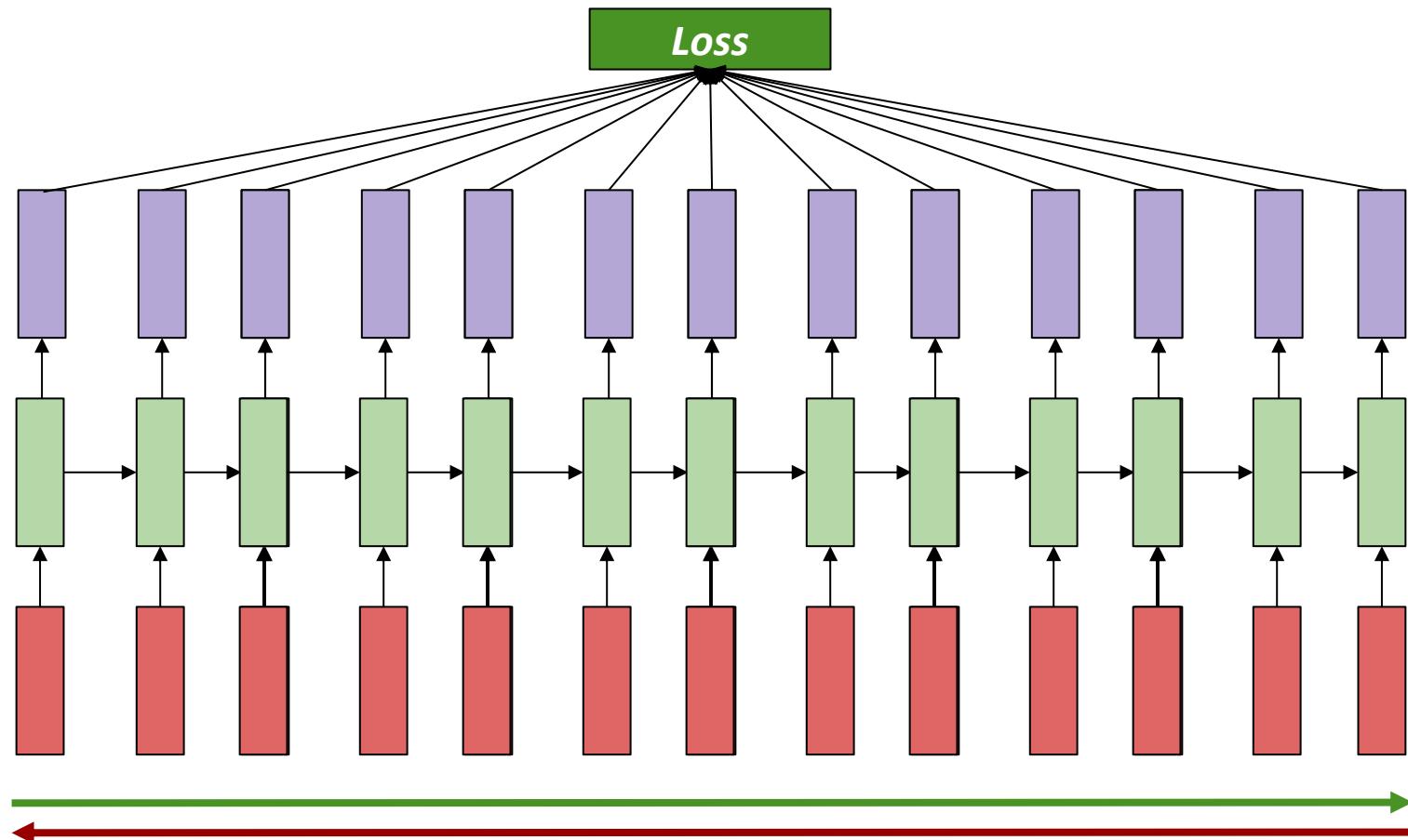
- Scan the input sequences separately
- One *forward* and one *backward*
- Connect to the same output layer
- Learn both *past* and *future* context

## Bidirectional RNN



*To determine  $y_t$  you have to  
see the whole sequence*

## Backpropagation through time (BPTT)



### Forward

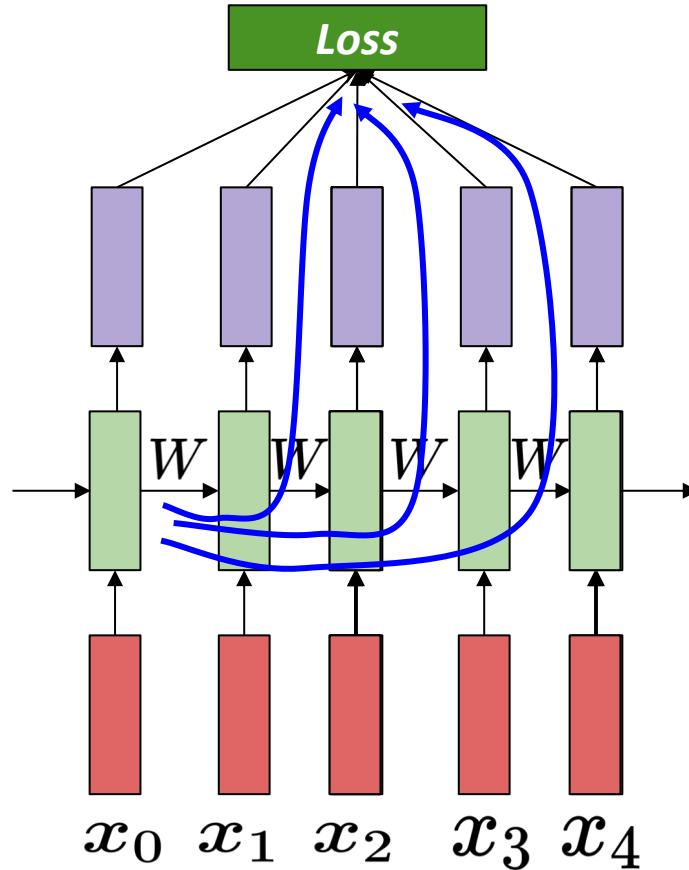
Run through entire sequence to compute the **loss**

### Backward

Run through entire sequence to compute the **gradient** and update the weight matrix:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

- Gradient flow in simple RNNs



$$w \leftarrow w - \eta \partial L / \partial w$$

Issue: **W** occurs each timestep

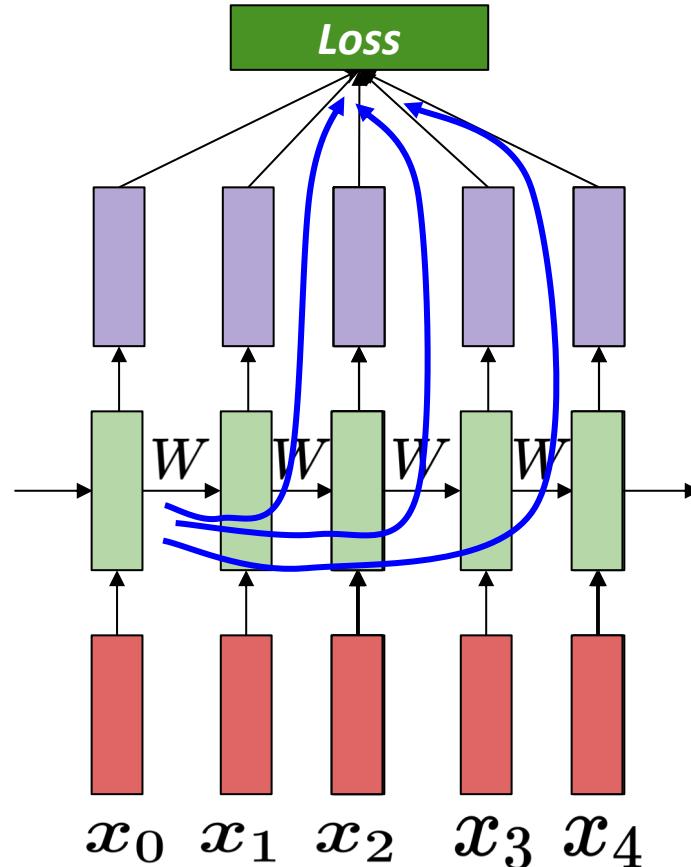
**Every** path from **W** to **Loss** is one dependency

All paths from **W** to **Loss** need to be involved

$$\frac{\partial L}{\partial w} = \sum_{j=0}^T \frac{\partial L_j}{\partial w}$$

$$\frac{\partial L}{\partial w} = \sum_{j=0}^T \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{t=k+1}^j \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_k}{\partial w}$$

## Gradient flow in simple RNNs



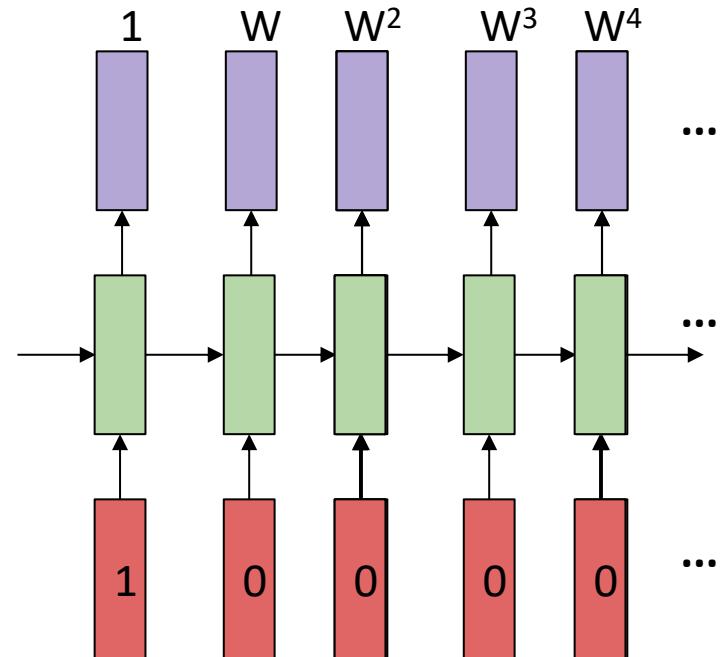
$$\frac{\partial L}{\partial w} = \sum_{j=0}^T \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{t=k+1}^j \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_k}{\partial w}$$

$$h_t = \text{tahn}(W_{hh}h_{t-1} + W_{xh}x_t)$$

computing gradient of  $h_{t-1}$  involves  $W_{hh}$

**Repeated** matrix multiplications leads to  
**vanishing** and **exploding** gradients.

- **Gradient flow in simple RNNs**



*Toy Example:*

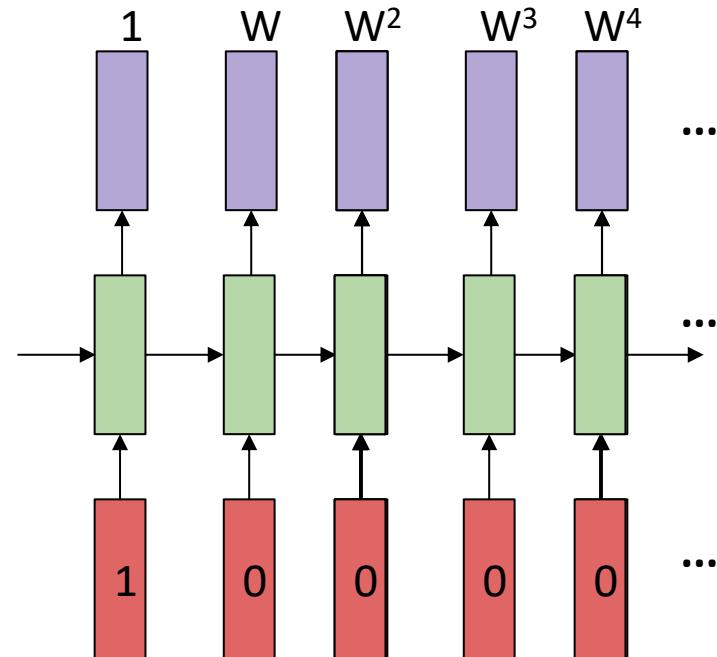
- 1-d input,  $x_0=1$ , and  $x_t=0$  ( $t>0$ )
- Sequence length is **1000**
- $W_{xh}=W_{hy}=1$
- Linear activation function

$$h_t = W_{hh} h_{t-1} + x_t$$

$$y_t = h_t$$

$$y_{1000} = W_{hh}^{999}$$

- Gradient flow in simple RNNs



$$y_{1000} = W_{hh}^{999}$$

$$w_{hh} = 1$$

$$w_{hh} = 1.01$$

$$y_{1000} = 1$$

$$y_{1000} \approx 20959$$

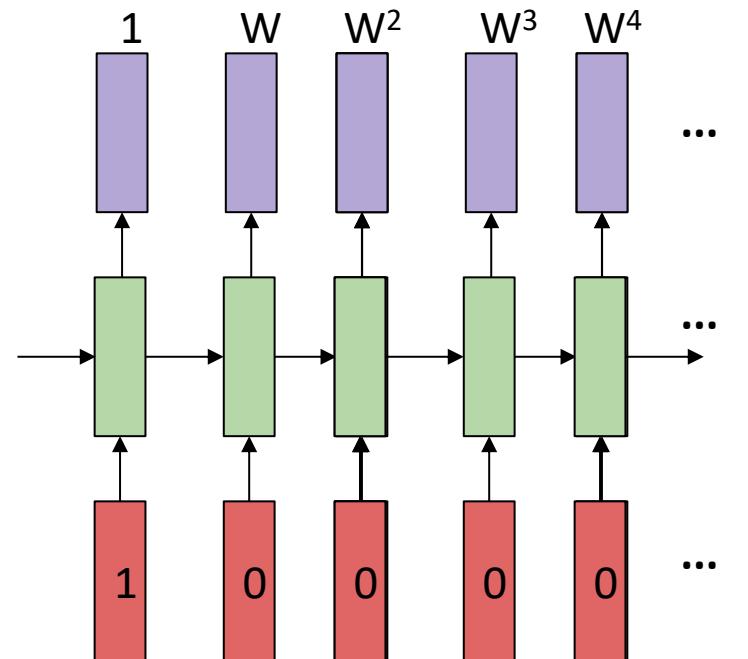
$\partial L / \partial w$  is too large

→ Exploding gradients

- Regulate via gradient clipping
  - Scale gradient if value gets to big

*if  $\|g\| > threshold$ ,  $g \leftarrow \frac{threshold \times g}{\|g\|}$*

- Gradient flow in simple RNNs



$\partial L / \partial w$  is too small

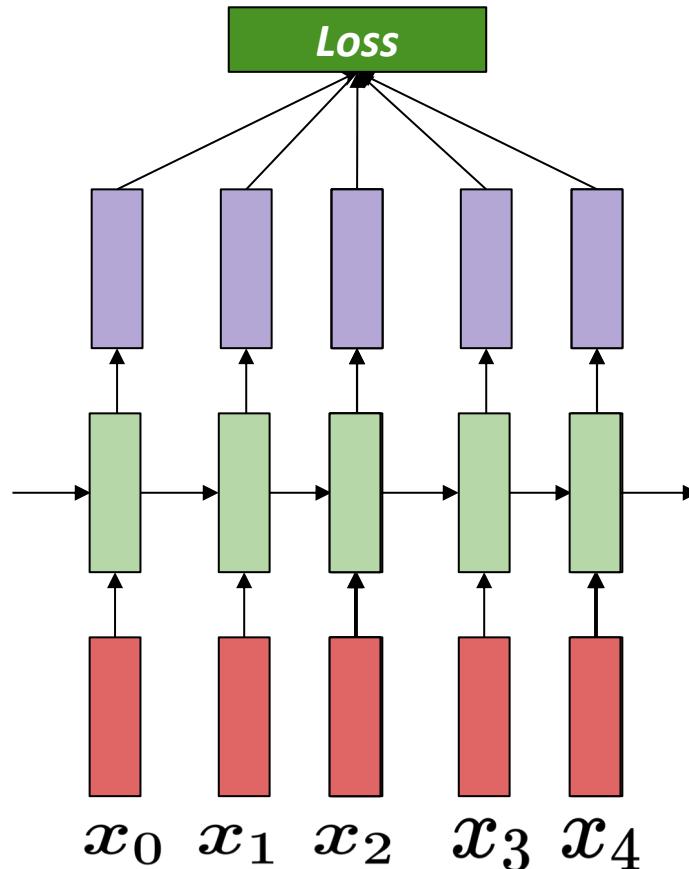
→ Vanishing gradients

- Need to change RNN architecture
- Gated RNNs

$$\begin{aligned}w_{hh} &= 0.99 \\w_{hh} &= 0.01\end{aligned}$$

$$\begin{aligned}y_{1000} &\approx 0 \\y_{1000} &\approx 0\end{aligned}$$

## Identity-RNN



- Network weights are initialised to the identity matrix
- Activation functions are all set to ReLU
- This encourages the network computations to stay close to the identity function
- **Gradient does not decay** as derivatives stay either 0 or 1
- Error is propagated all the way back

$$\frac{\partial L}{\partial w} = \sum_{j=0}^T \sum_{k=1}^j \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left( \prod_{t=k+1}^j \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_k}{\partial w}$$

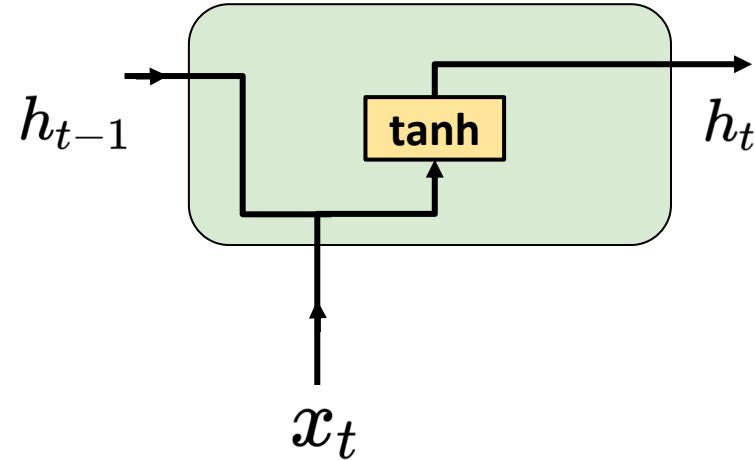
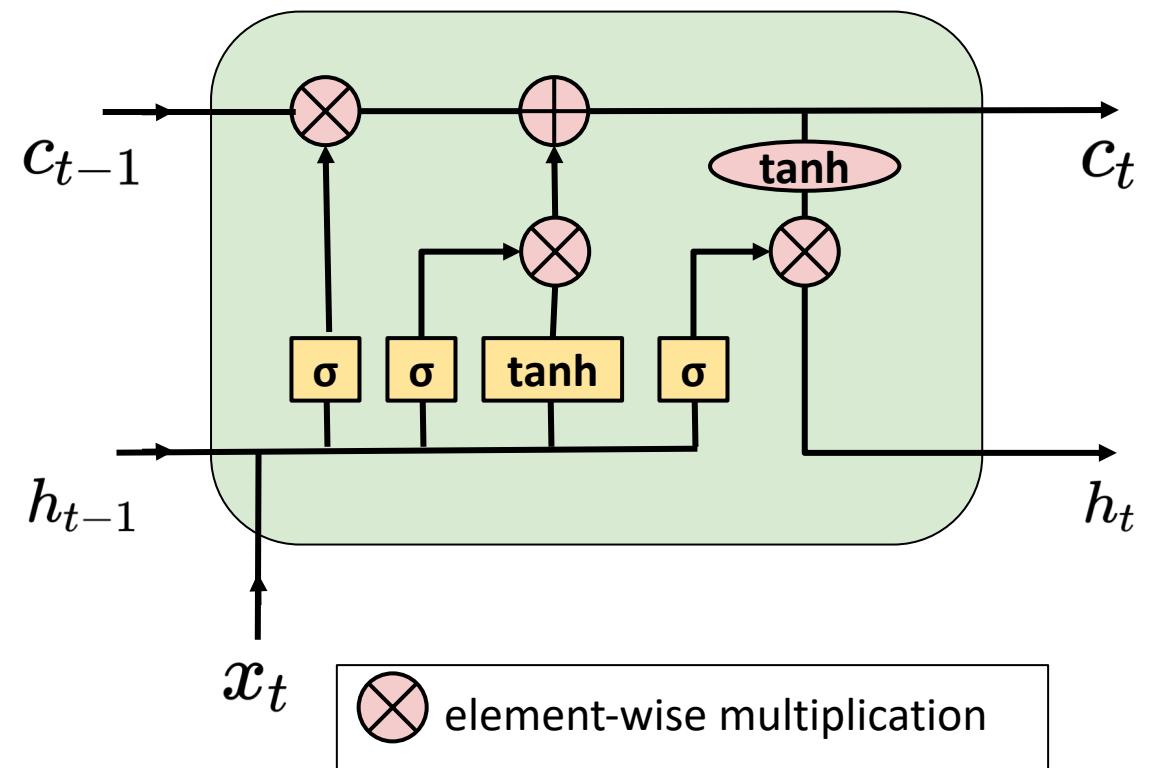
$$h_t = h_{t-1} + f(x_t) \rightarrow \left( \frac{\partial h_t}{\partial h_{t-1}} \right) = 1$$

## Highly effective sequence models

- Gated RNNs are based on the idea of creating paths that have derivatives that neither vanish nor explode
- Gated RNNs have connection weights that may change at each time step
- Gated RNNs also allow a network to forget an old state
- Instead of manually deciding when to clear the state, the network to learn to decide when to do it.

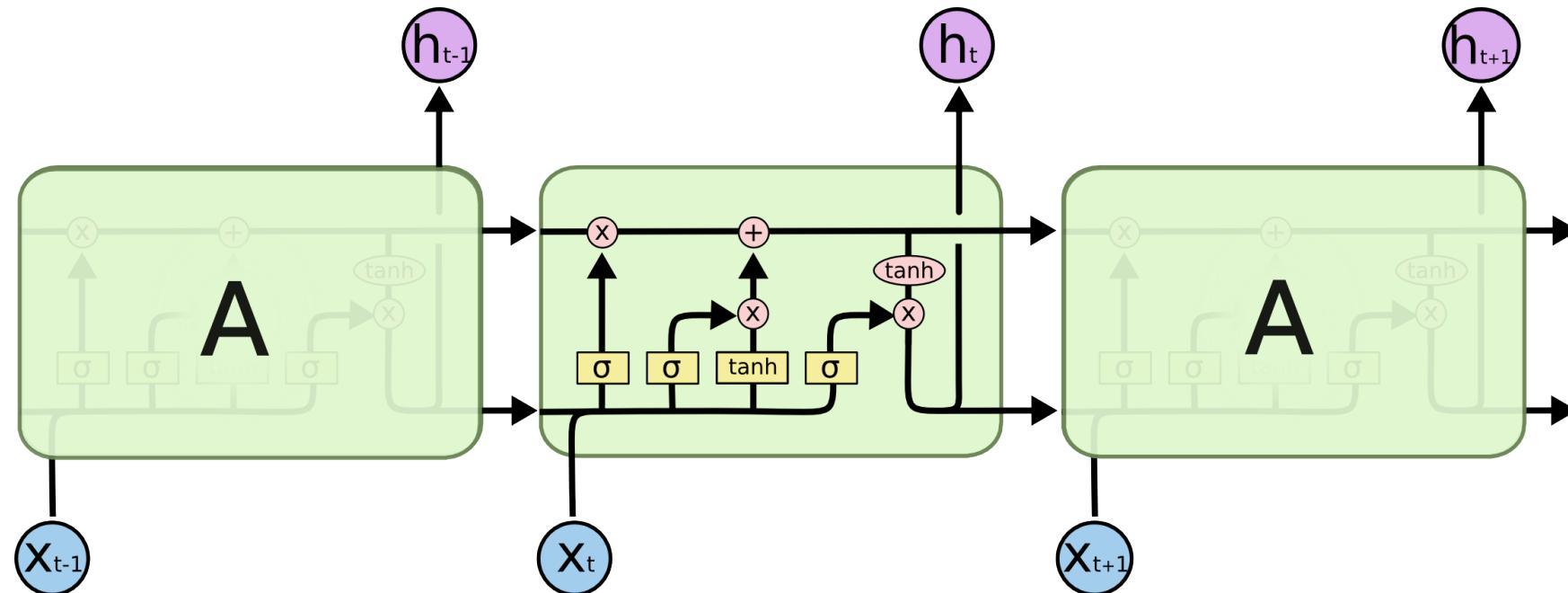
- **LSTM Recurrent Unit**

- Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks
- An input feature is computed with a regular artificial neuron
  - This value is accumulated into the state if the input gate allows it
- The state unit has a linear self-loop whose weight is controlled by the forget gate
- The output of the cell can be shut off by the output gate
- All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity

**Simple RNN unit****Gated LSTM unit**

- **Unrolled LSTM network**

- There are four interacting networks: **cell state**, **input gate**, **forget gate**, **output gate**



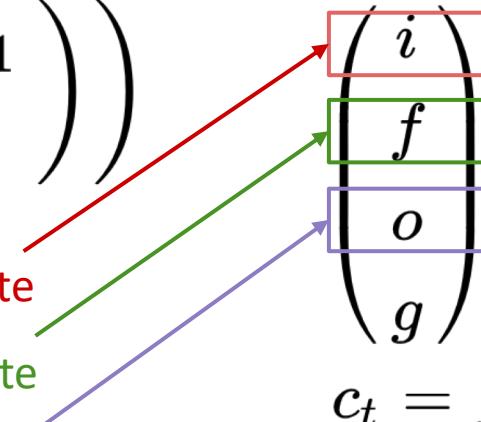
### Simple RNN unit

$$h_t = \tanh \left( (W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}) \right)$$

i : Control **input** gate

f: Control **forget** gate

o: Control **output** gate



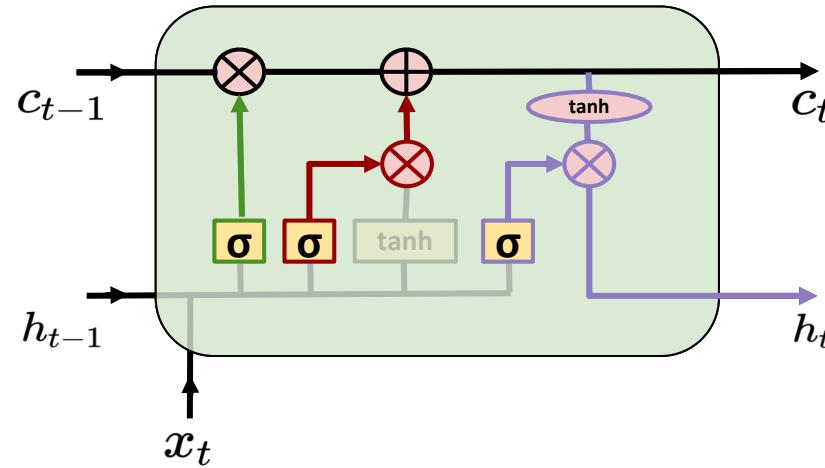
### LSTM unit

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g \quad <\text{ cell state}$$

$$h_t = o \odot \tanh(c_t) \quad <\text{ output of the unit}$$

Activation function is usually sigmoid for the three gates, returning values between 0 and 1, to mimic open and close of the gates.



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

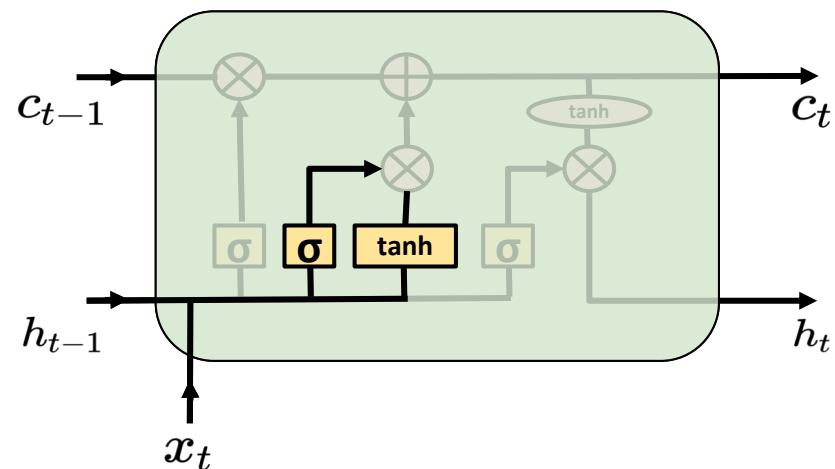
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

## Cell state $c$ :

- Core of LSTM unit
- Information is removed or added to the cell
- Controlled by three gates:
  - *Input gate*: how much to write to cell
  - *Forget gate*: how much to erase/keep in the cell
  - *Output gate*: how much to output from the cell

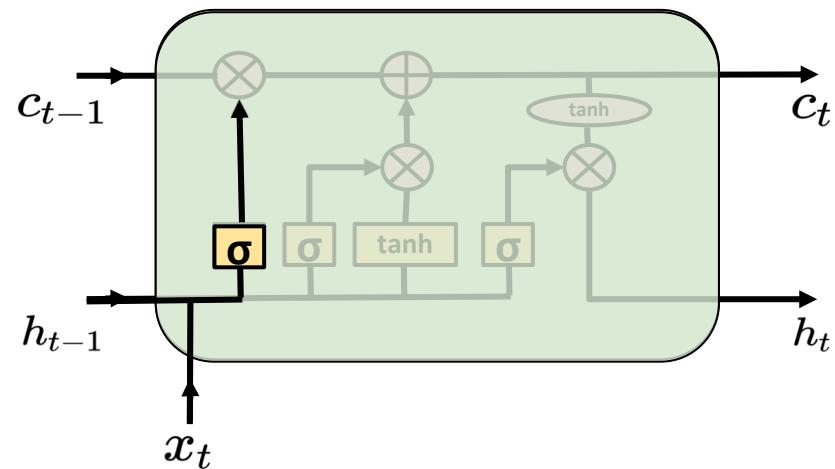
- **Input gate:** how much to write to cell



$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g)$$

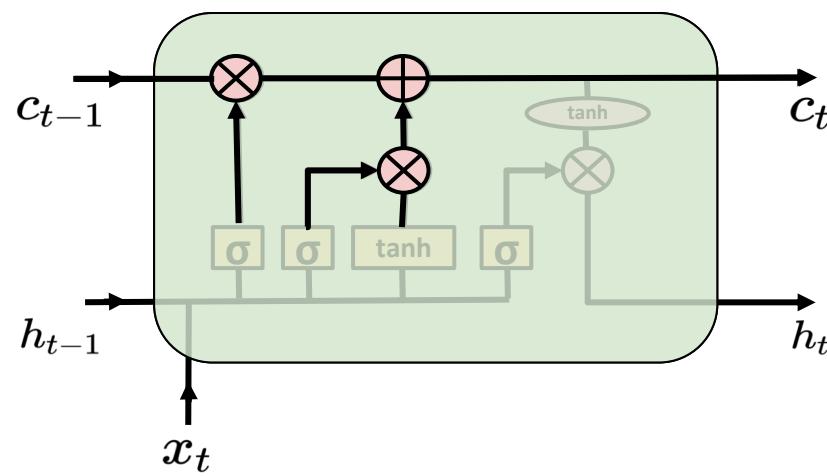
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

- **Forget gate:** how much to erase/keep in the cell



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

- Update the cell state from  $c_{t-1}$  to  $c_t$



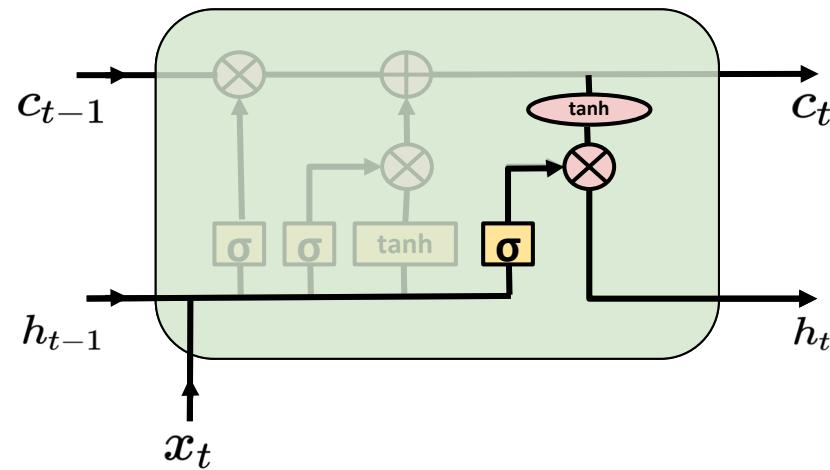
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

Influence of new information on cell

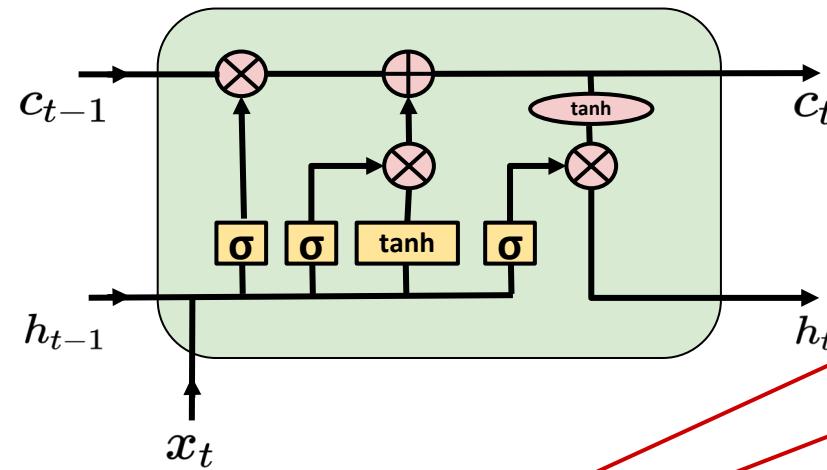
How much of previous cell state to forget

Note the previous cell state and input are added together which enables LSTM units to handle vanishing gradients

- **Output gate:** How much to output from the cell



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$



$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

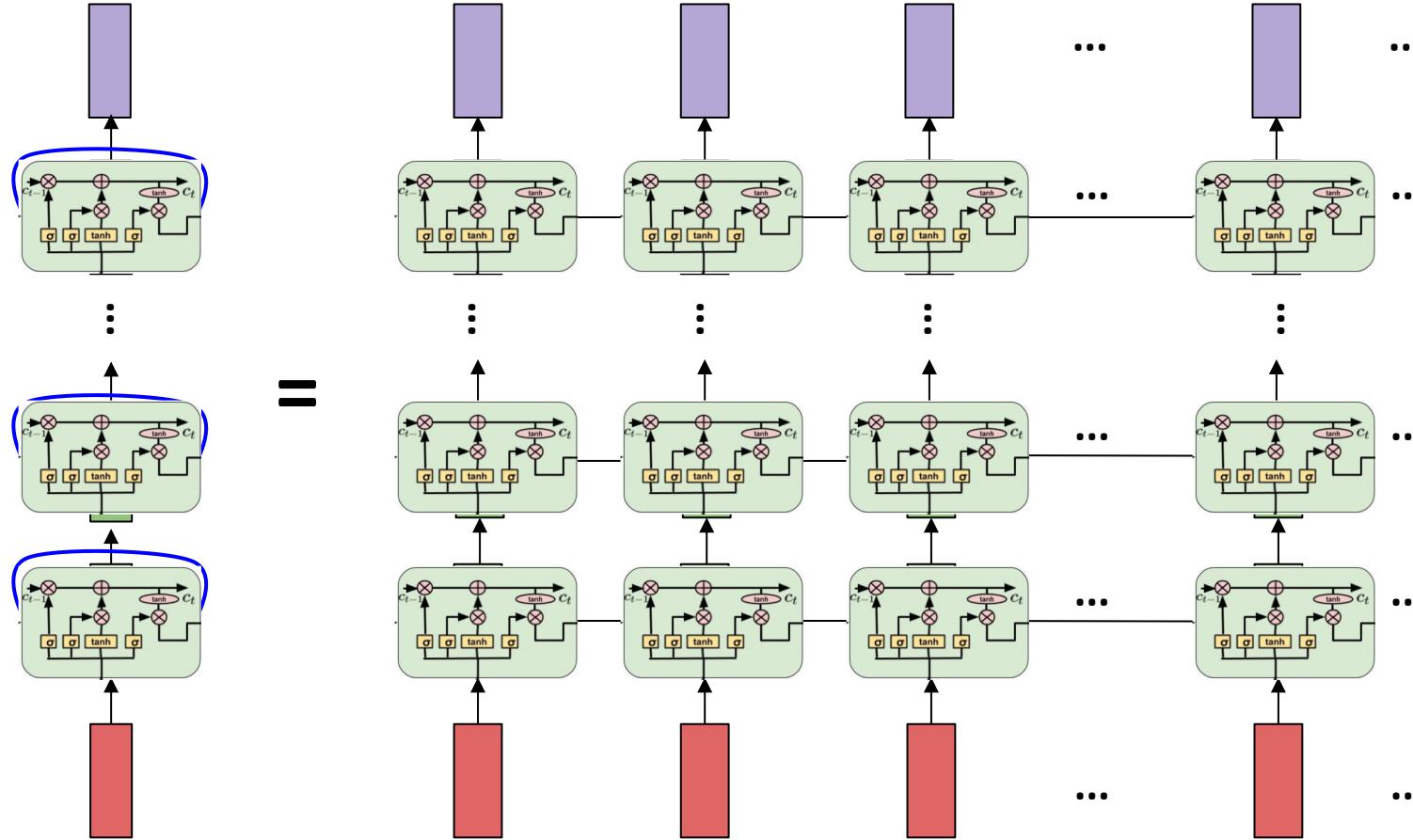
$$h_t = o_t \odot \tanh(c_t)$$

More weights to learn

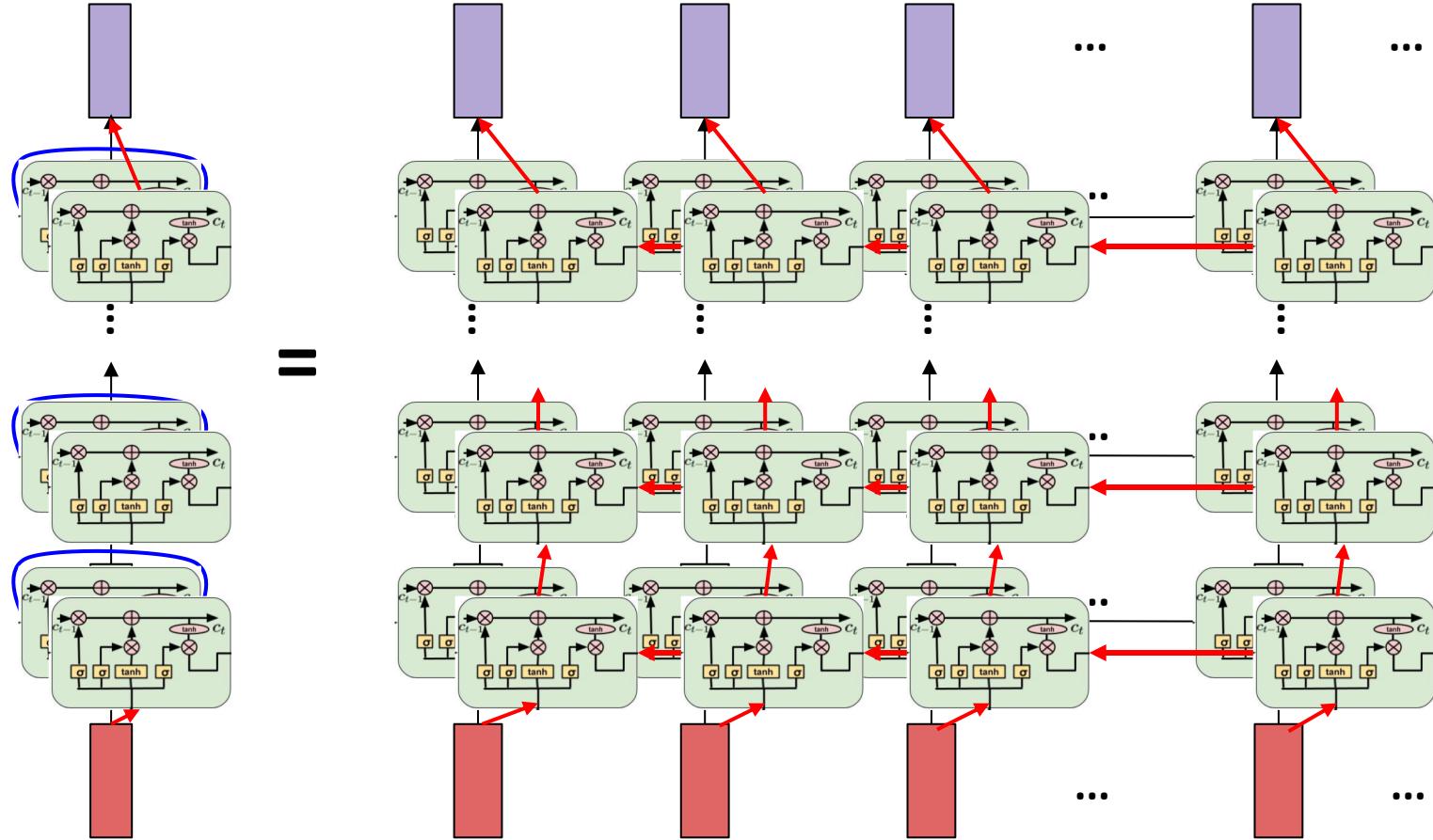
As in the *simple RNN*  
We reuse the same weight  
matrix at every time step

## Deep LSTM-RNN

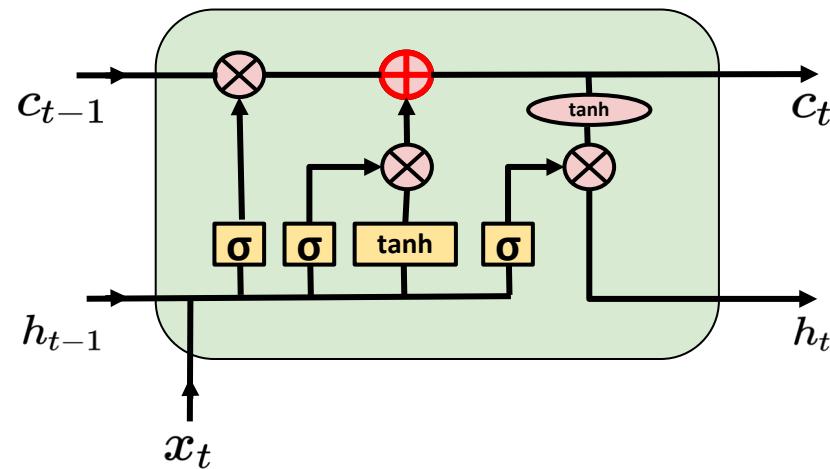
Replace simple RNN units with LSTM cells



## Deep BLSTM RNN

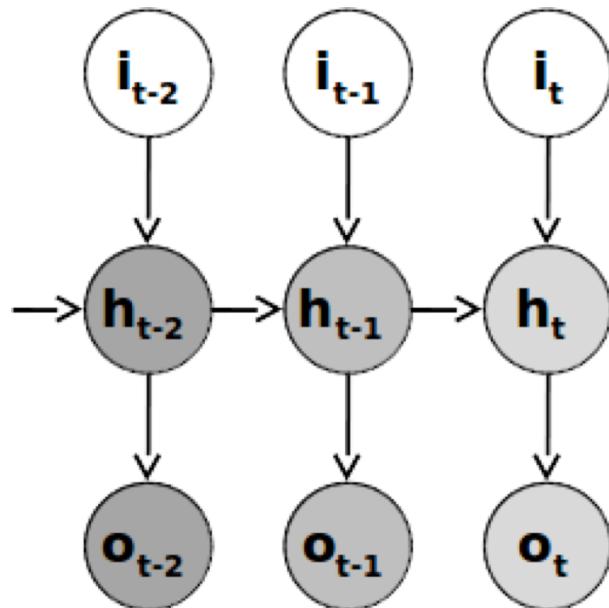


## LSTM can deal with the gradient vanishing:

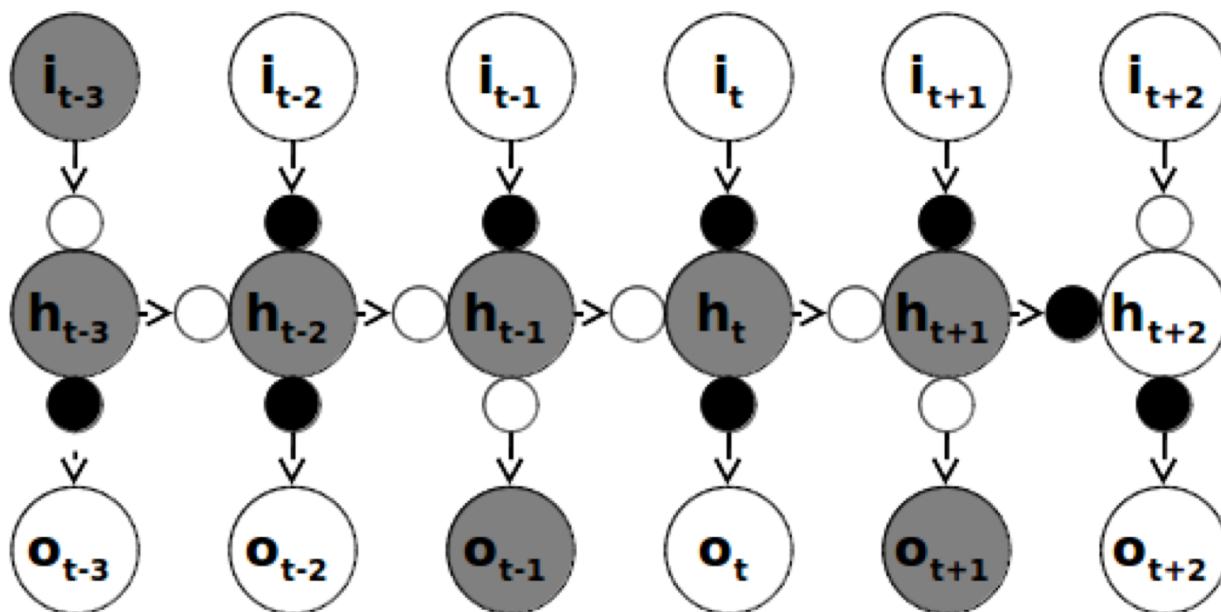


- Previous cell state and input are **added together**
- The influence never disappears unless the forget gate is completely closed
- Instead of computing new state as a matrix product with the old one, it rather computes the additive interactions
- Gradient flow is therefore improved
- Store/retrieve information over longer time periods

## Simple RNN

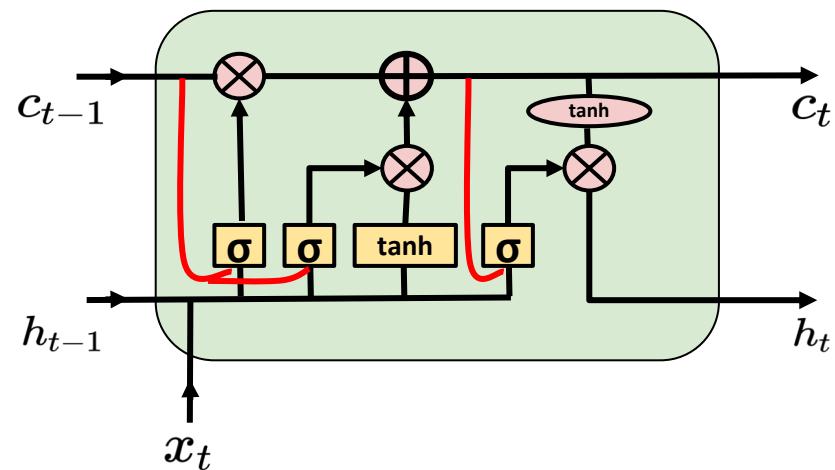


## LSTM RNN



## LSTM extension: “peephole connections”

- Allows the gates to inspect the current cell state
- Helps when learning precise timings of events.



Gates are also controlled by the *cell state c*

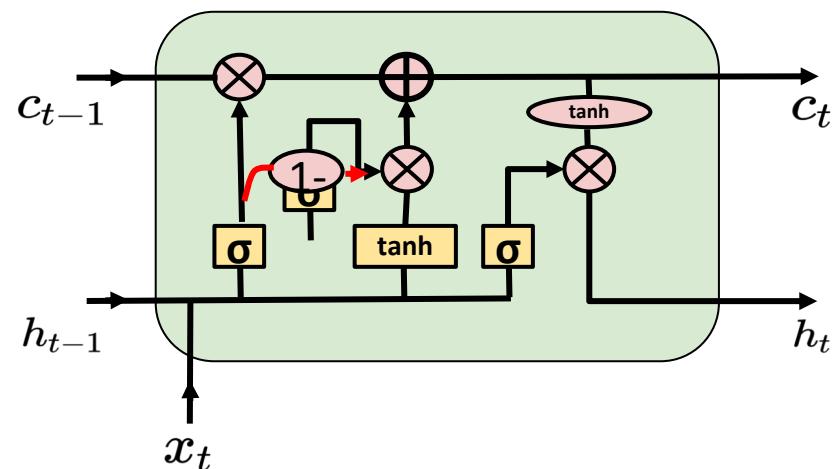
$$i_t = \sigma (W_i \cdot [c_{t-1}, h_{t-1}, x_t] + b_i)$$

$$f_t = \sigma (W_f \cdot [c_{t-1}, h_{t-1}, x_t] + b_f)$$

$$o_t = \sigma (W_o \cdot [c_t, h_{t-1}, x_t] + b_o)$$

## LSTM extension

- Combine forget and input gates



*Jointly decide* what to add and forget

$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g)$$

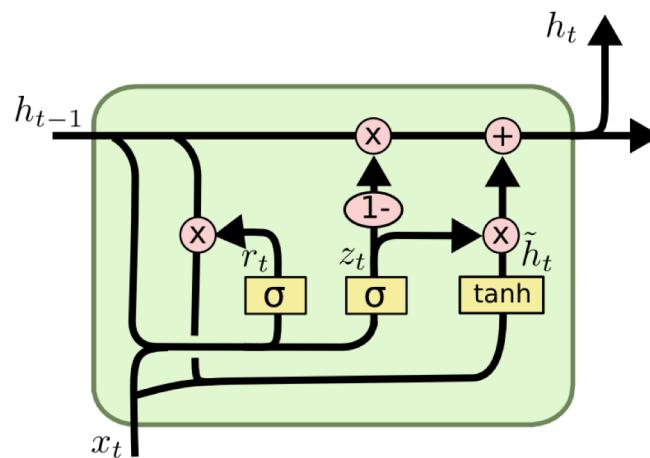
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot g_t$$

## Gated Recurrent Unit

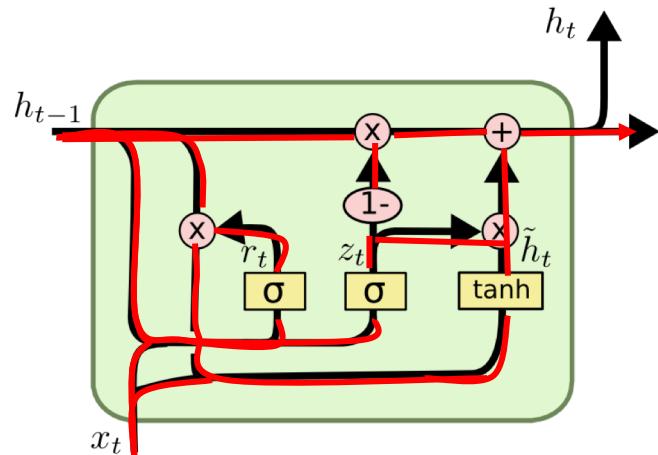
- Single gating unit simultaneously controls the forgetting factor and the decision to update the state unit



- Two gates only:
  - *Reset gate r*
  - *Update gate z*
- Eliminate internal cell state  $c$

## Gated Recurrent Unit

- Single gating unit simultaneously controls the forgetting factor and the decision to update the state unit



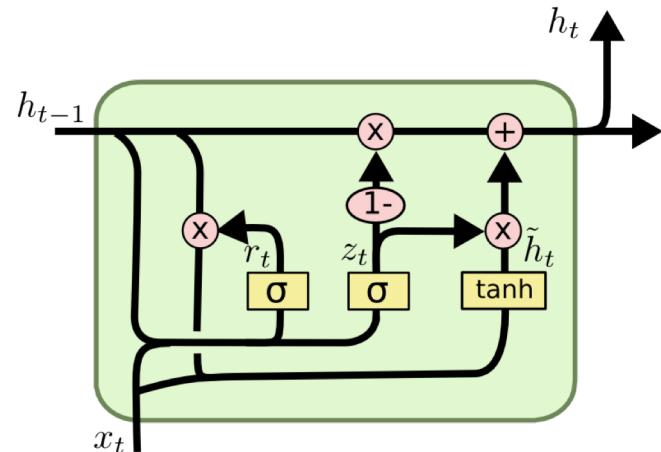
$$z_t = \sigma(\mathbf{W}_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(\mathbf{W}_r \cdot [h_{t-1}, x_t] + b_r)$$

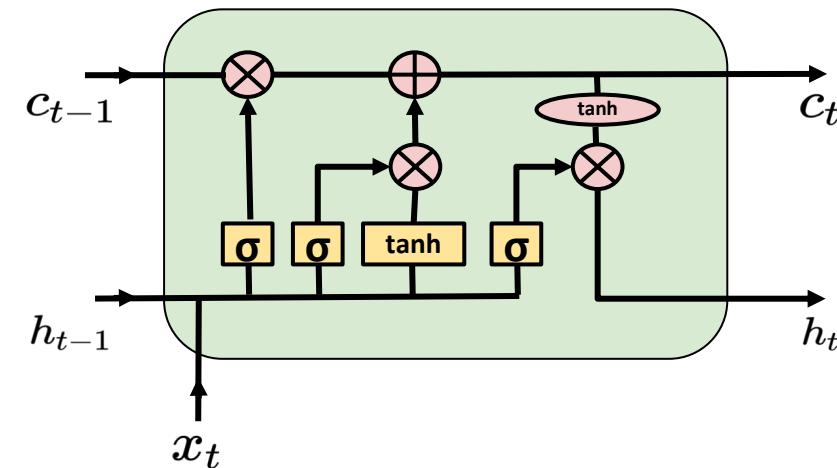
$$g_t = \tanh(\mathbf{W}_g \cdot [r_t \odot h_{t-1}, x_t] + b_g)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot g_t$$

## Comparing GRU and LSTM

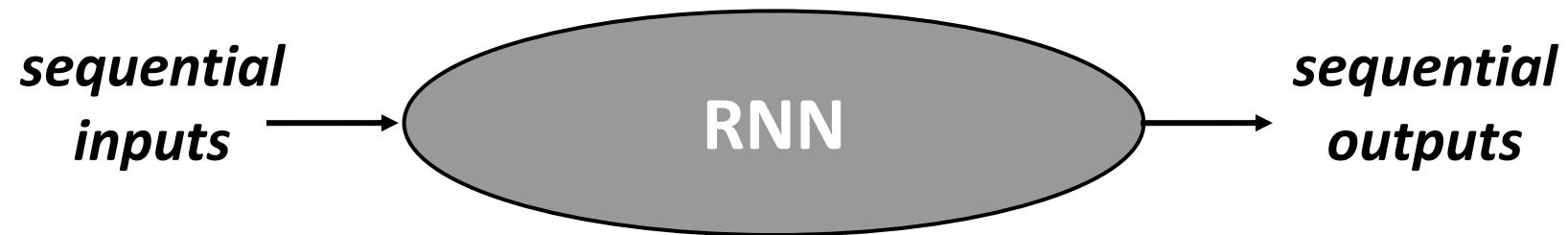


*which is better?  
do the differences  
matter?*



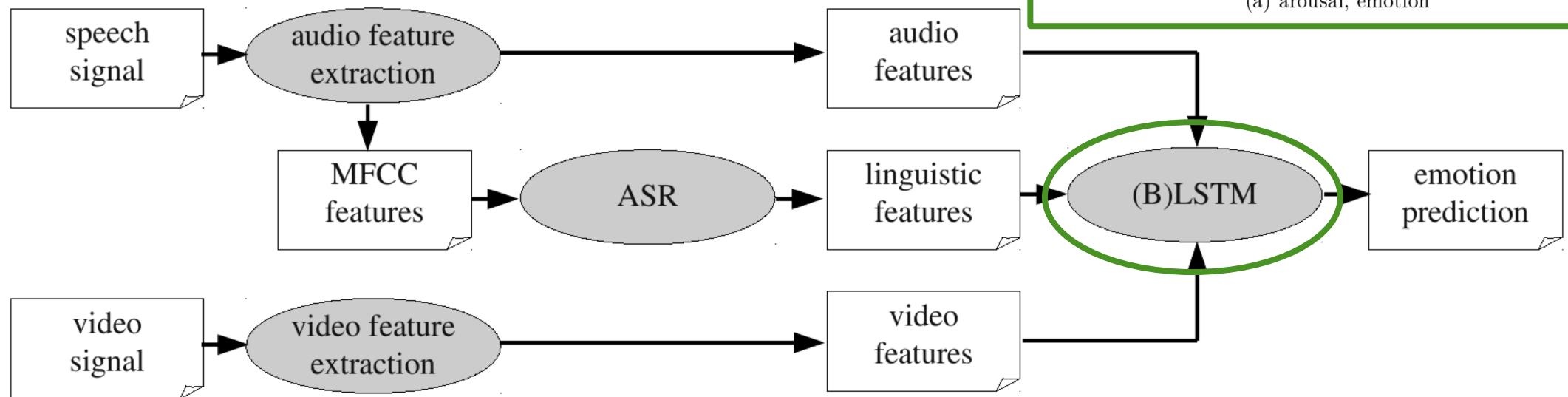
- Both GRU and LSTM are *better* than a simple RNN
- GRU is *faster* to train with fewer parameters
- GRU performs *comparably to* LSTM, on certain tasks even better than LSTM

## sequence labeling with RNNs



## sequence labeling with RNNs

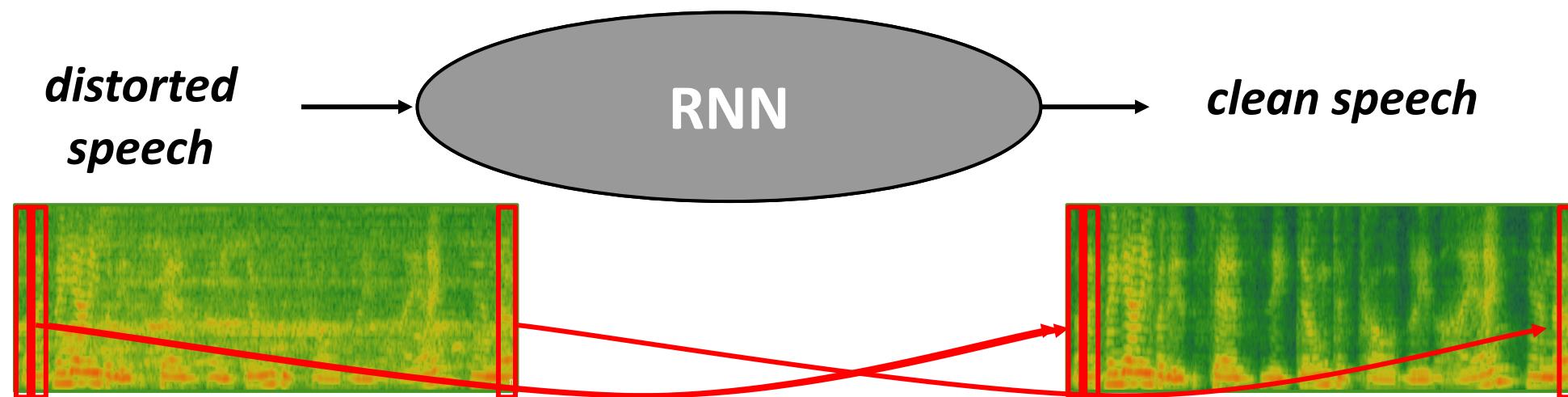
*Example: continuous audiovisual emotion recognition*



"LSTM-Modeling of continuous emotions in an audiovisual affect recognition framework". IMAVIS. 2013.

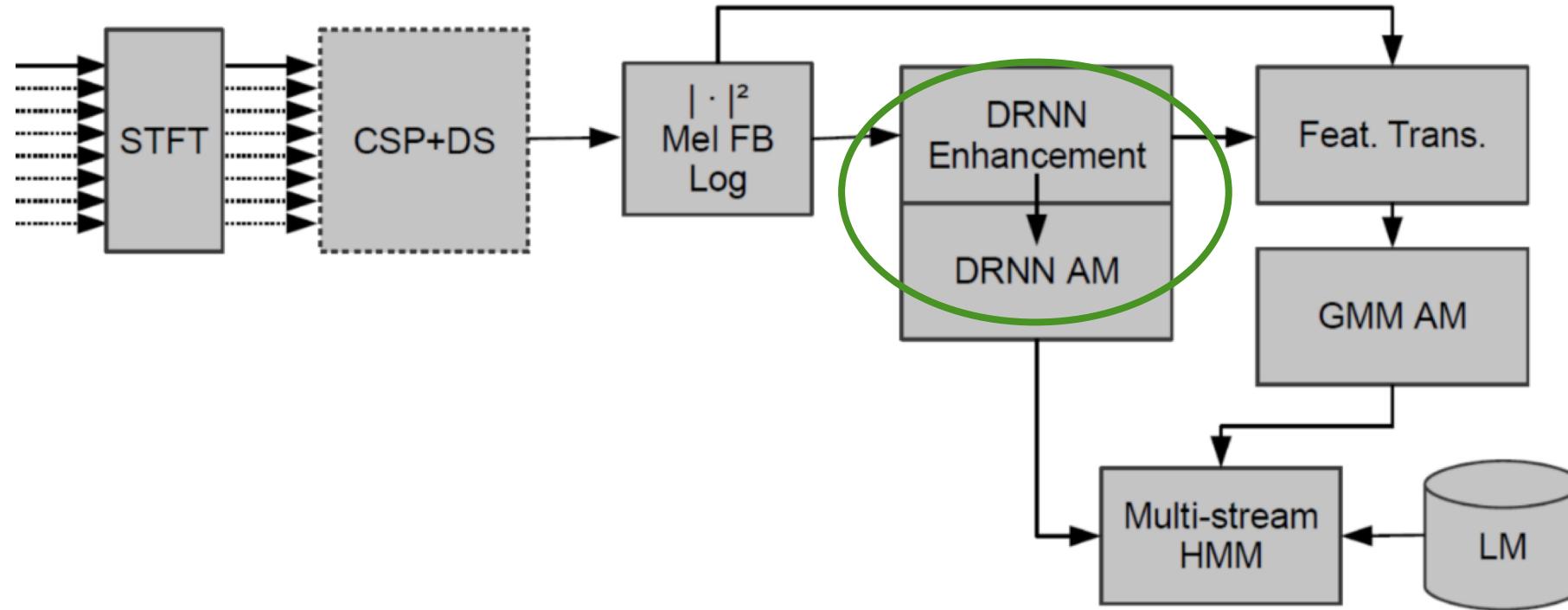
## feature enhancement with RNNs

*Example: speech denoising*



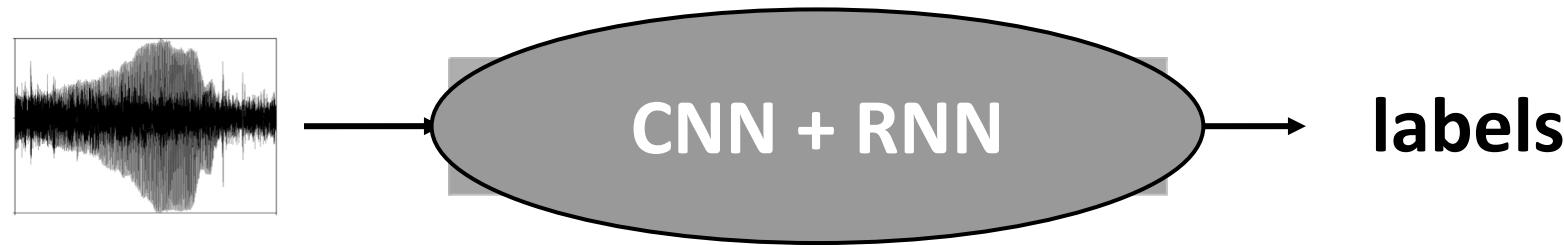
*"Facing realism in spontaneous emotion recognition from speech: Feature enhancement by autoencoder with LSTM neural networks",  
INTERSPEECH 2016.*

## speech recognition with RNNs



*The TUM+TUT+KUL Approach to the CHiME Challenge 2013: Multi-Stream ASR Exploiting BLSTM Networks and Sparse NMF”, CHiME, 2013.  
(Best Paper Award)*

combination of CNNs and RNNs  
*(End-to-End)*



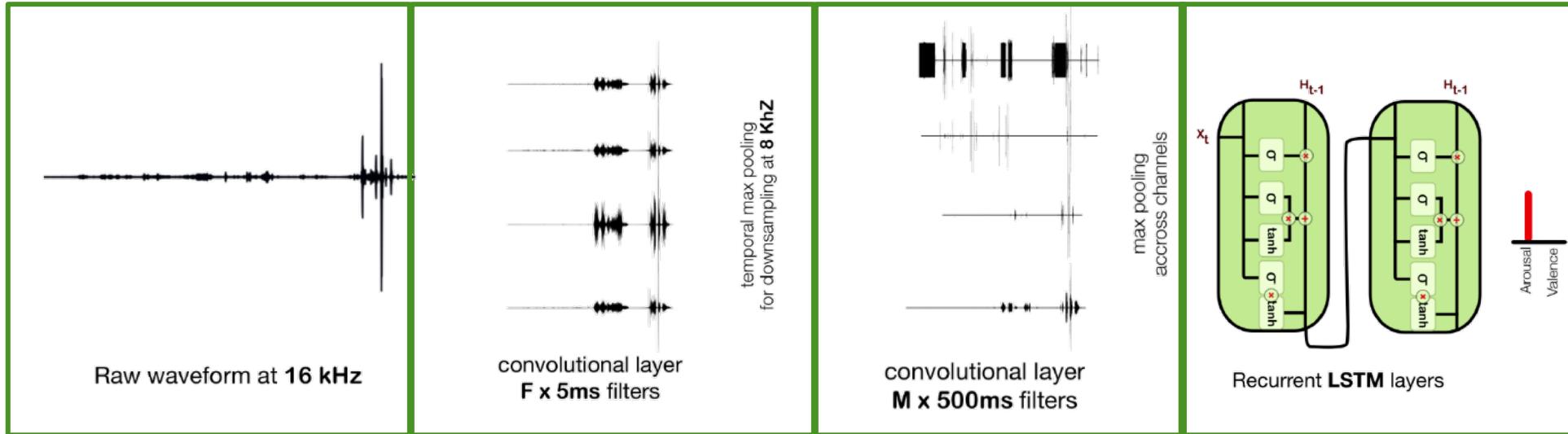
CNN and RNN are jointly trained

# Wide Application of RNNs

Björn W. Schuller

combination of CNNs and RNNs  
*(End-to-End)*

Arousal	CCC
RNN with ComParE features	.382
<b>CNN + RNN with raw signals</b>	<b>.686</b>

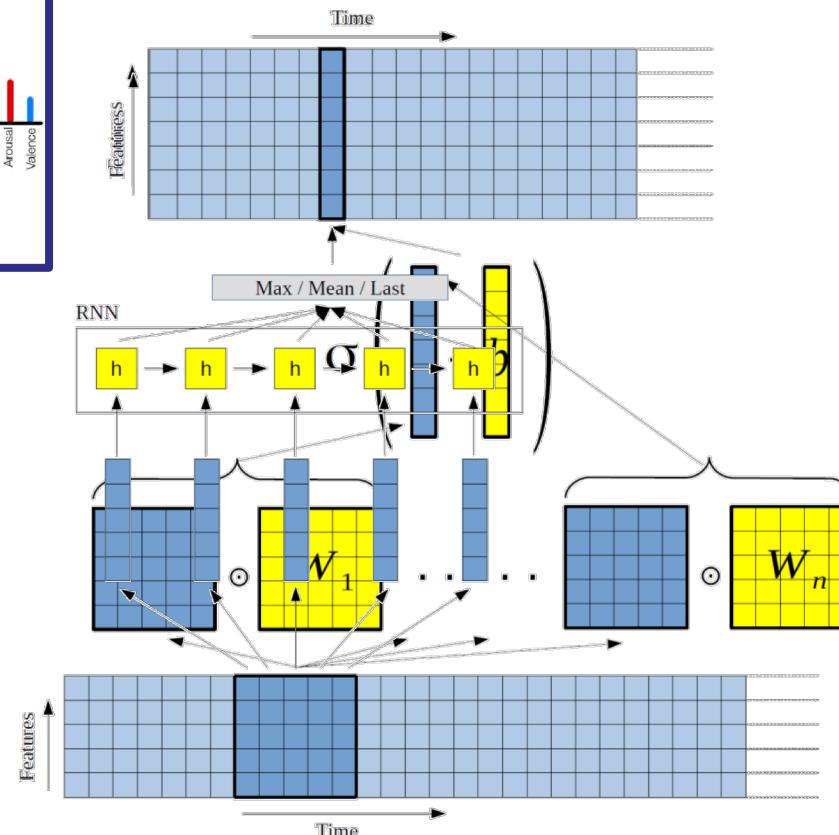
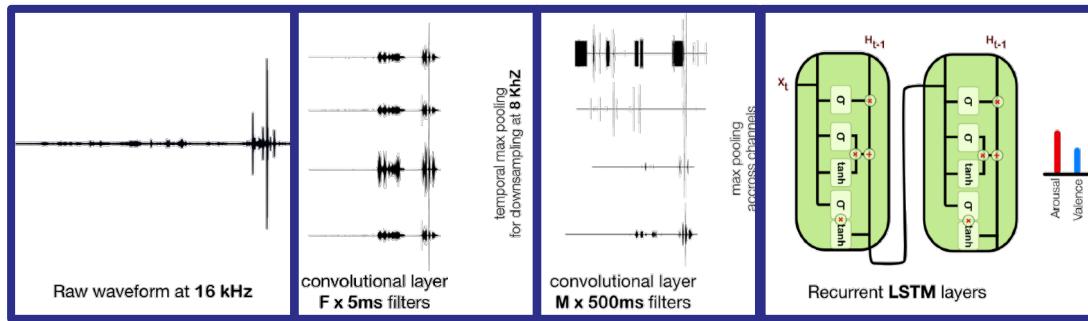


"Adieu Features? End-to-End Speech Emotion Recognition using a Deep Convolutional Recurrent Network". ICASSP. 2016.

# Wide Application of RNNs

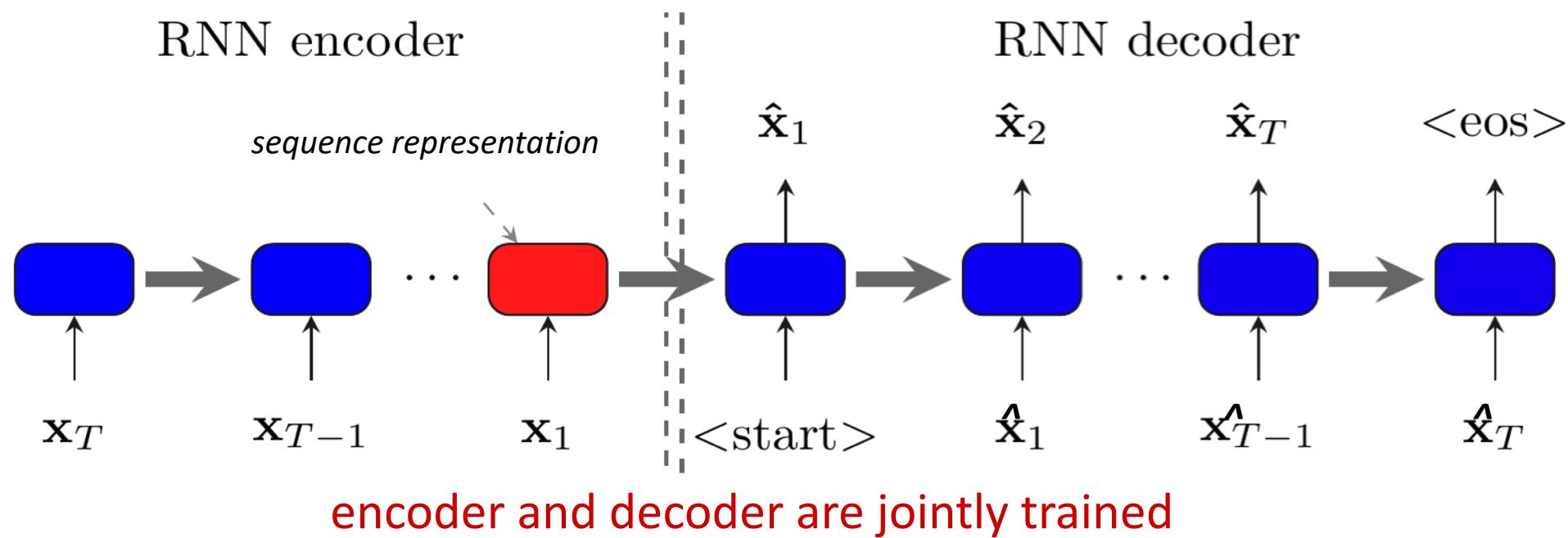
Björn W. Schuller

- CNN + LSTM → CLSTM ?

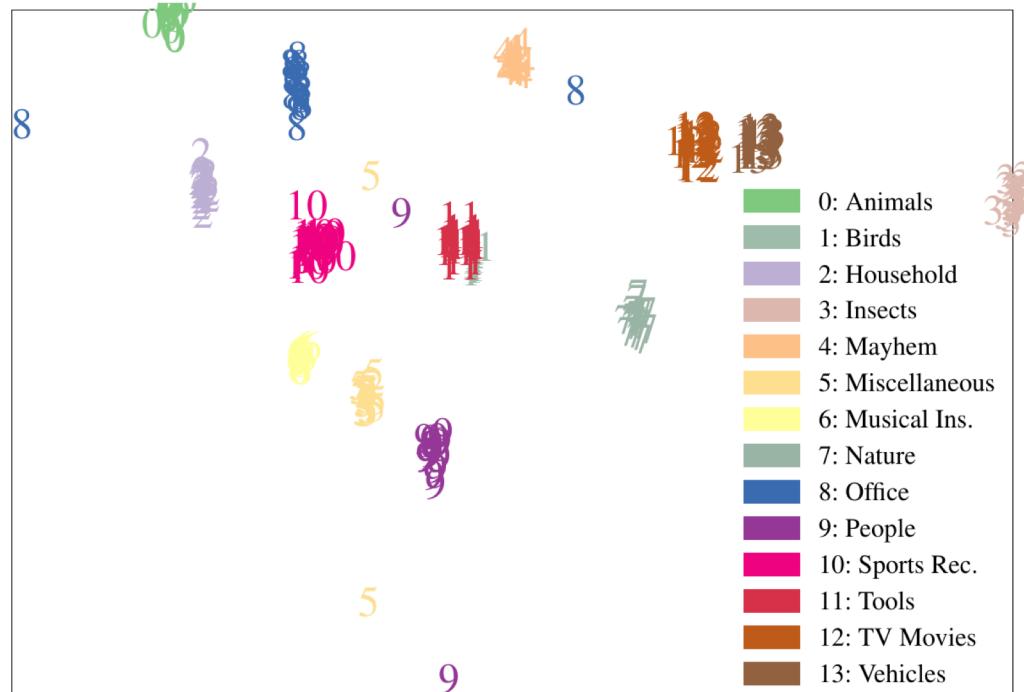


"Adieu Features? End-to-End Speech Emotion Recognition using a Deep Convolutional Recurrent Network", ICASSP, 2016.

## sequence representation learning with RNNs (*sequence-to-sequence*)



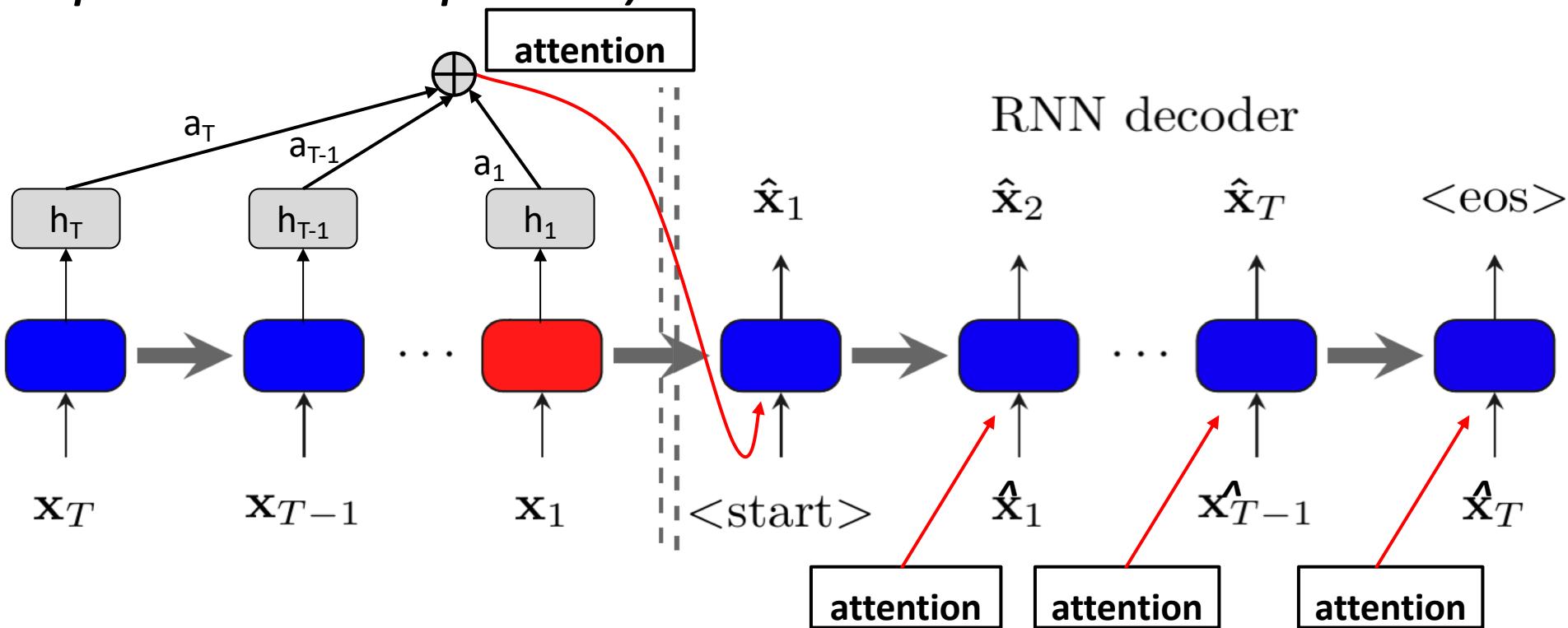
## sequence representation learning with RNNs (*sequence-to-sequence*)



"Learning Audio Sequence Representations for Acoustic Event Classification". arXiv 2017.

feature types	UA
ComParE features	.462
<b>features learnt by seq2seq</b>	<b>.876</b>

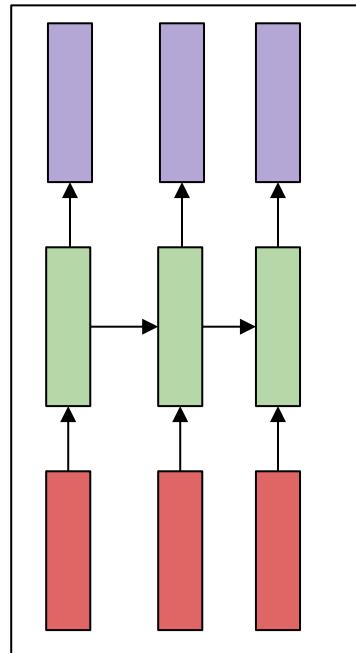
## RNNs *with Attention* (sequence-to-sequence)



# Connectionist Temporal Classification

Björn W. Schuller

Output: “deep learning”



Input: audio waveform of “deep learning”



**However, we do not want to pre-segment input data:**

- time-consuming
- too expensive
- not accurate in most cases

CTC is applied to address this **temporal classification problem** without the need for **frame-level alignments**, and normally **output sequence** is much **shorter than input sequence**.

# Connectionist Temporal Classification

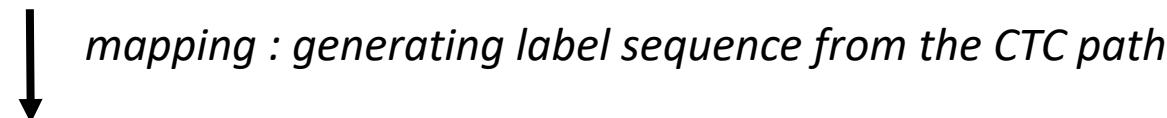
Björn W. Schuller

CTC is a sequence-to-sequence learning technique

E.g., *input X* =  $(x_1, \dots, x_T)$  - a sequence of frame-level observations



*CTC path P* =  $(y_1, \dots, y_T)$  - a sequence of frame-level predictions



*output Y* = ("d", "e", "e", "p") - a much shorter label sequence

$$L_{CTC} = \ln Pr(Y|X)$$

- a **CTC path P** is a sequence of labels on frame-level
- its likelihood can be decomposed into frames (*independent like HMMs*):

$$Pr(p|X) = \prod_{t=1}^T y_t$$

- it differs from labels as it
  - introduces a special symbol - blank “∅” - as an additional label, meaning no (actual labels) are assigned to the frame
  - allows **repetitions** of non-blank labels

e.g. D ∅ ∅ E ∅ E ∅∅ P∅ → DEEP

D D ∅ E ∅ E E ∅ P P → DEEP

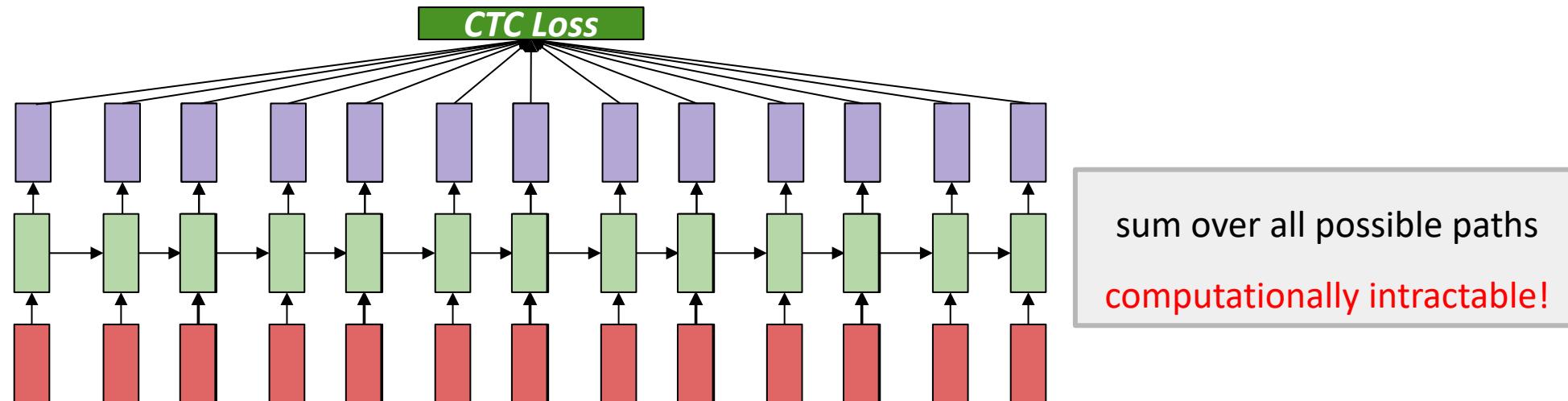
∅ D E E ∅ E E E P P → DEEP

∅ D ∅ E E E E ∅ P∅ → DEP

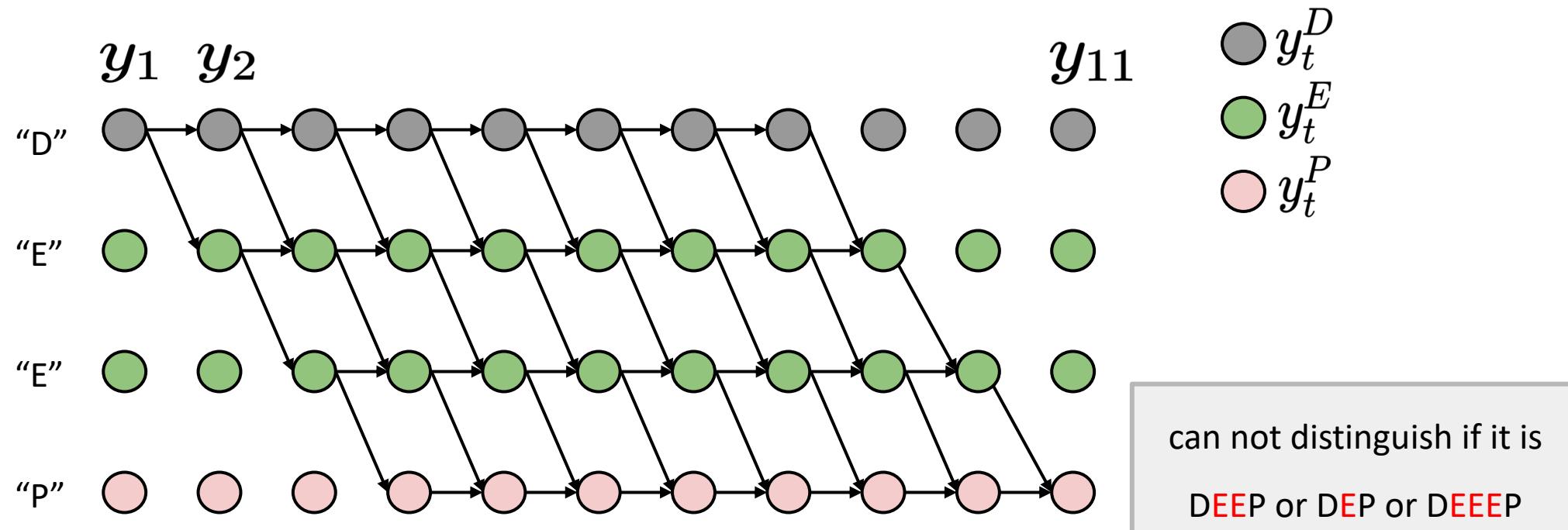
a lot of CTC paths mapping to  
a *final* label “DEEP”

during **training**: for a given label  $Y$ , CTC maximises the total probability of all frame-level paths  $p$  consistent with the output label  $Y$ , i.e.,  $p \in \phi(Y)$  to cover all possible alignments, with **CTC loss function**:

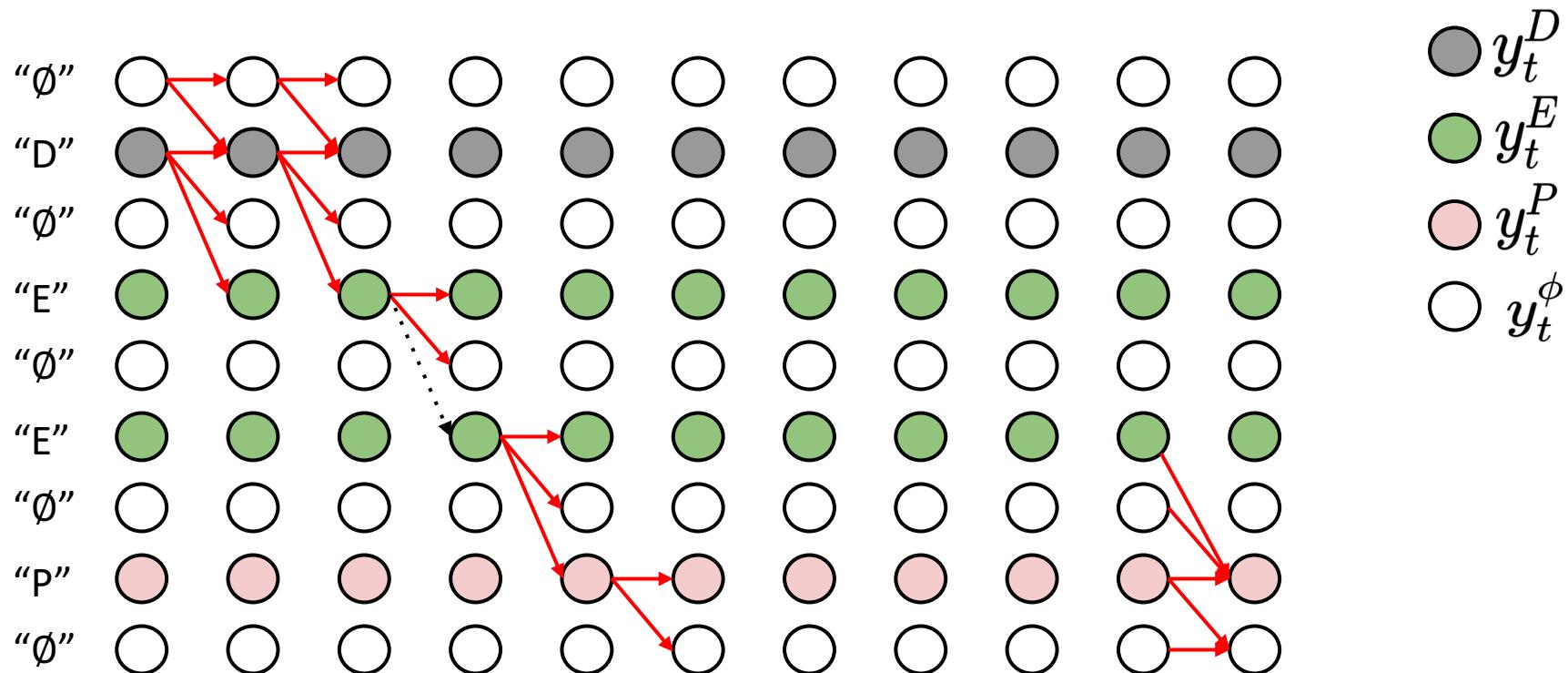
$$L_{CTC} = \ln Pr(Y|X) = \ln \sum_{p \in \phi(Y)} Pr(p|X)$$



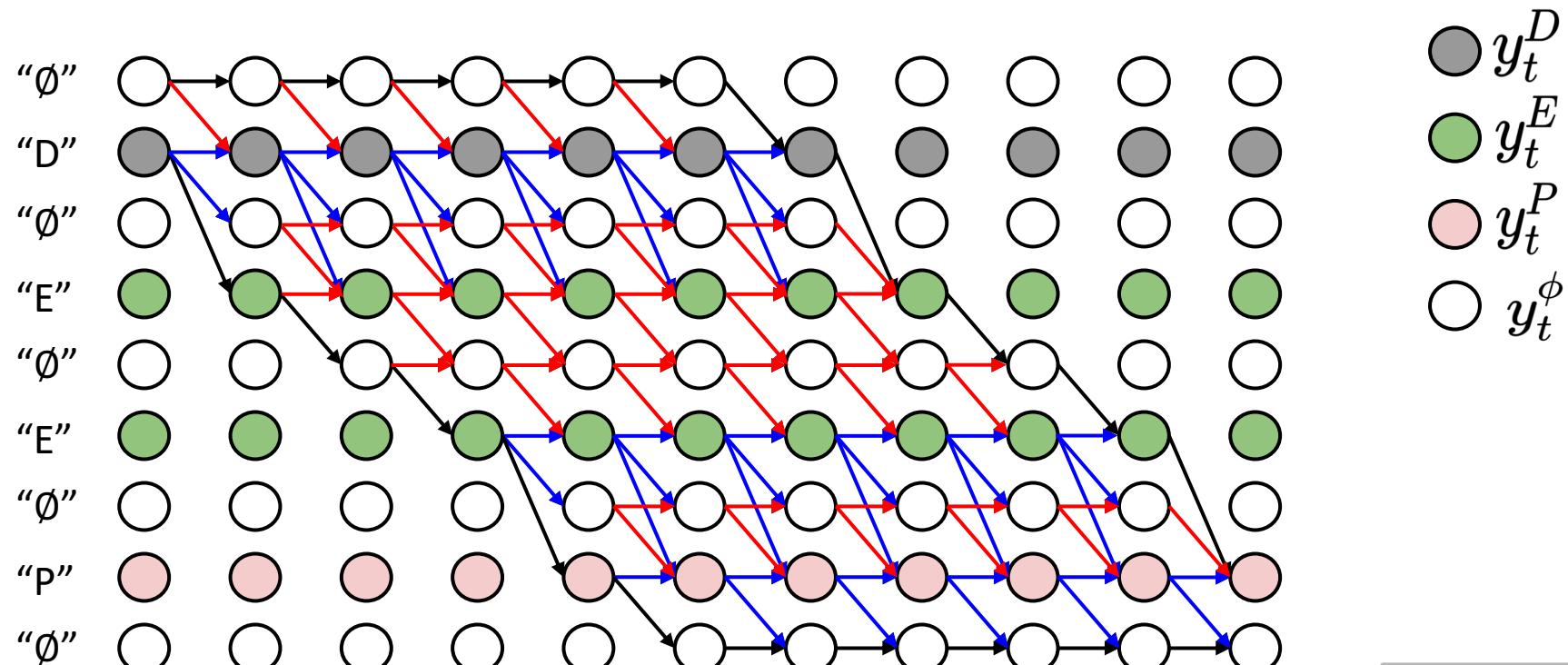
## Example of possible paths for output “DEEP”



Example of possible paths for output “DEEP” **with “Ø”**



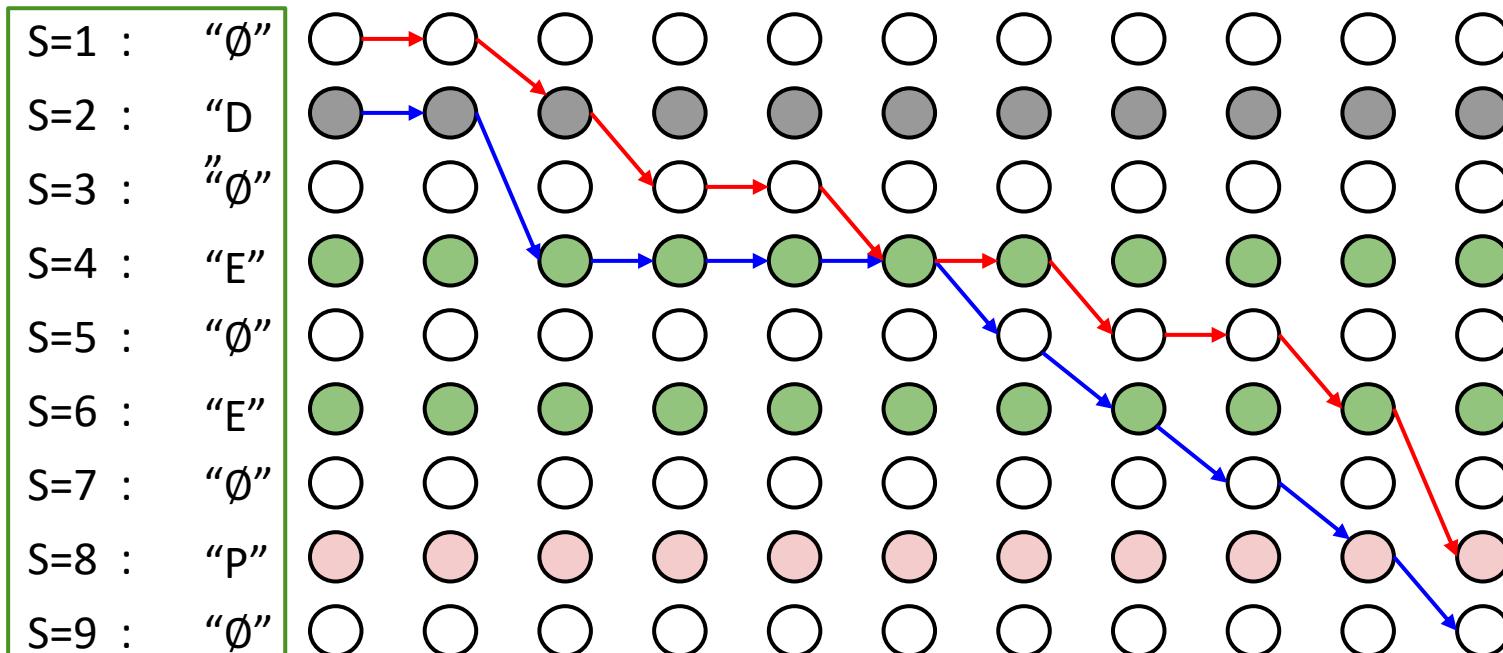
Example of possible paths for output “DEEP” **with “Ø”**



$$L_{CTC} = \ln \Pr(Y|X) = \ln \sum_{p \in \phi(Y)} \Pr(p|X)$$

compute with forward-backward algorithm

## forward-backward algorithm



summed probability  
of all the CTC paths  
**ending** at time  $t$  with  
state  $s$

$$\alpha_t(s) \cdot y_t^s \cdot \beta_t(s)$$

summed probability  
of all the CTC paths  
**starting** at time  $t$  with  
state  $s$

$$P_1 = y_1^\emptyset \cdot y_2^\emptyset \cdot y_3^D \cdot y_4^\emptyset \cdot y_5^\emptyset \cdot y_6^E \cdot y_7^E \cdot y_8^\emptyset \cdot y_9^\emptyset \cdot y_{10}^E \cdot y_{11}^P$$

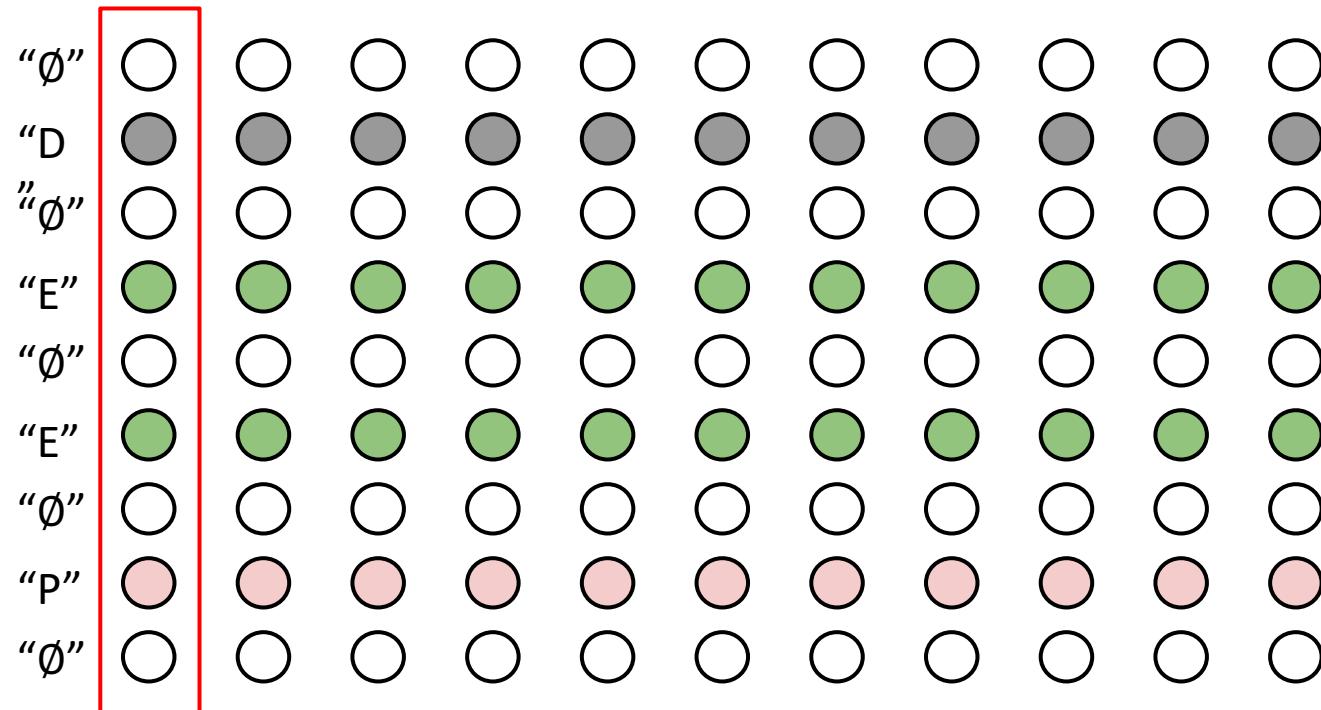
$$P_2 = y_1^D \cdot y_2^D \cdot y_3^E \cdot y_4^E \cdot y_5^E \cdot y_6^E \cdot y_7^\emptyset \cdot y_8^E \cdot y_9^\emptyset \cdot y_{10}^P \cdot y_{11}^\emptyset$$

$$P_3 = y_1^\emptyset \cdot y_2^\emptyset \cdot y_3^D \cdot y_4^\emptyset \cdot y_5^\emptyset \cdot y_6^E \cdot y_7^\emptyset \cdot y_8^E \cdot y_9^\emptyset \cdot y_{10}^P \cdot y_{11}^\emptyset$$

$$P_4 = y_1^D \cdot y_2^D \cdot y_3^E \cdot y_4^E \cdot y_5^E \cdot y_6^E \cdot y_7^E \cdot y_8^\emptyset \cdot y_9^\emptyset \cdot y_{10}^E \cdot y_{11}^P$$

more paths than these 4 paths  
that pass  $y_6^E$

forward computation  $\alpha_t(s)$



$$\alpha_1(\phi) = y_1^\phi$$

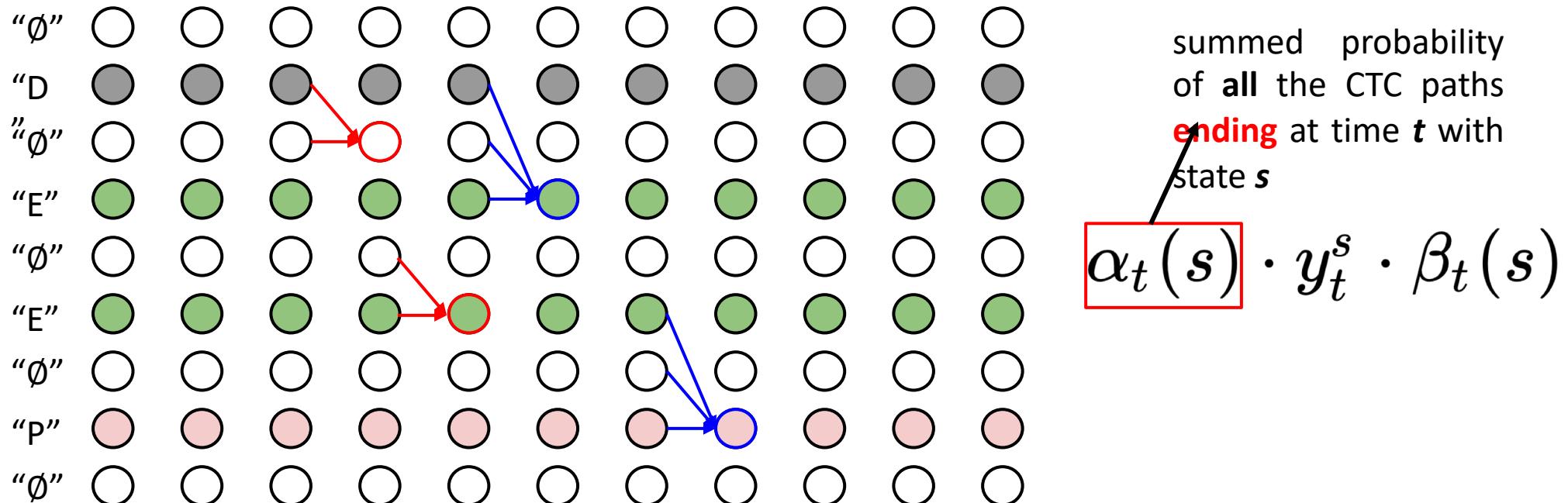
$$\alpha_1(D) = y_1^D$$

$$a_1(l_s) = 0, \text{ if } l_s \text{ is not } \phi \text{ or } D$$

summed probability  
of all the CTC paths  
**ending** at time  $t$  with  
state  $s$

$$\boxed{\alpha_t(s)} \cdot y_t^s \cdot \beta_t(s)$$

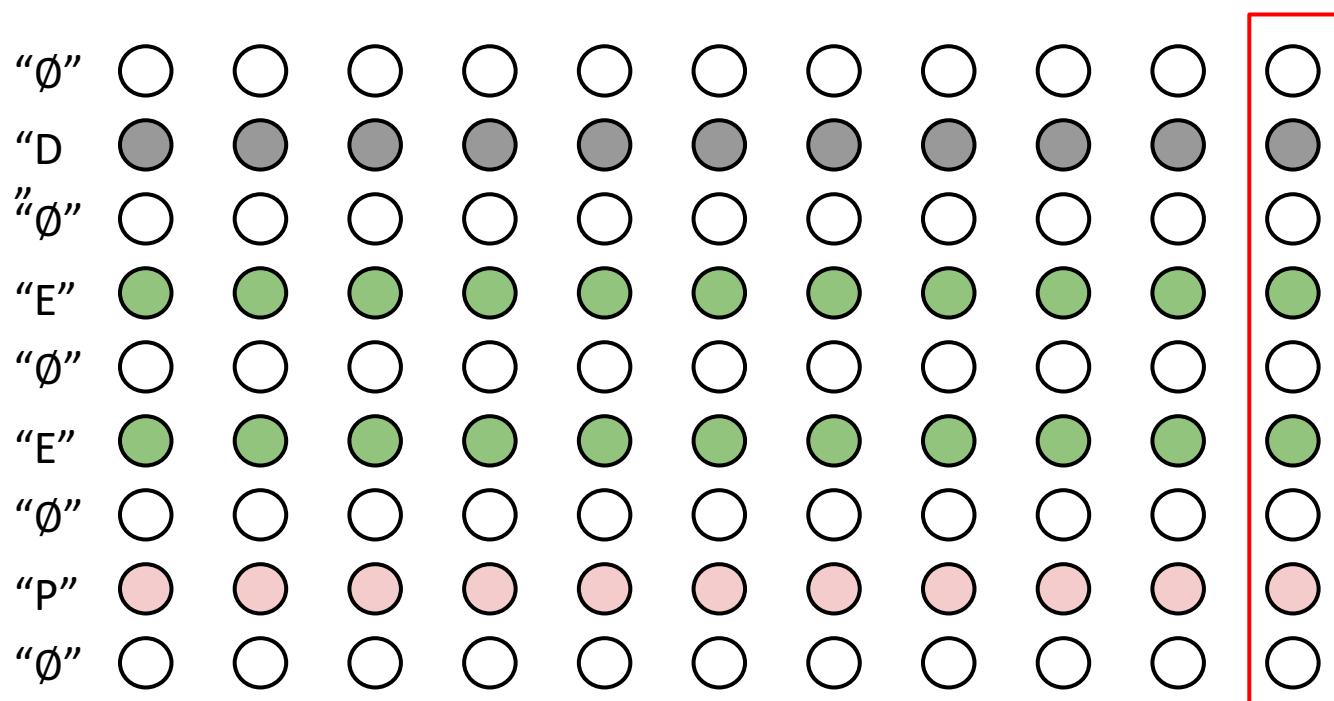
forward computation  $\alpha_t(s)$



$$a_t(s) = (a_{t-1}(s) + a_{t-1}(s-1)) y_t^{l_s} \quad \text{if } l_s = \phi \text{ or } l_s = l_{s-2}$$

$$a_t(s) = (a_{t-1}(s) + a_{t-1}(s-1) + a_{t-1}(s-2)) y_t^{l_s} \quad \text{otherwise}$$

backward computation  $\beta_t(s)$



$$\beta_T(\phi) = y_T^\phi$$

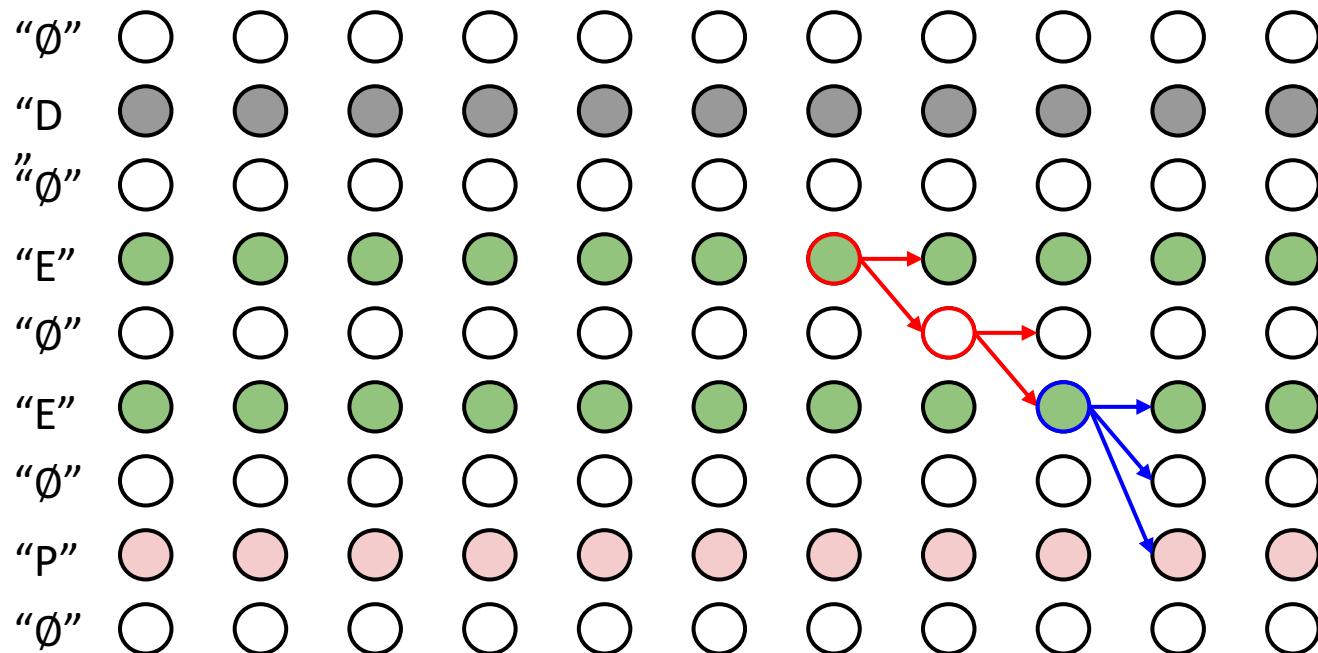
$$\beta_T(P) = y_T^P$$

$$\beta_T(l_s) = 0, \text{ if } l_s \text{ is not } \phi \text{ or } P$$

$$\alpha_t(s) \cdot y_t^s \cdot \boxed{\beta_t(s)}$$

summed probability  
of **all** the CTC paths  
**starting** at time  $t$  with

## backward computation $\beta_t(s)$



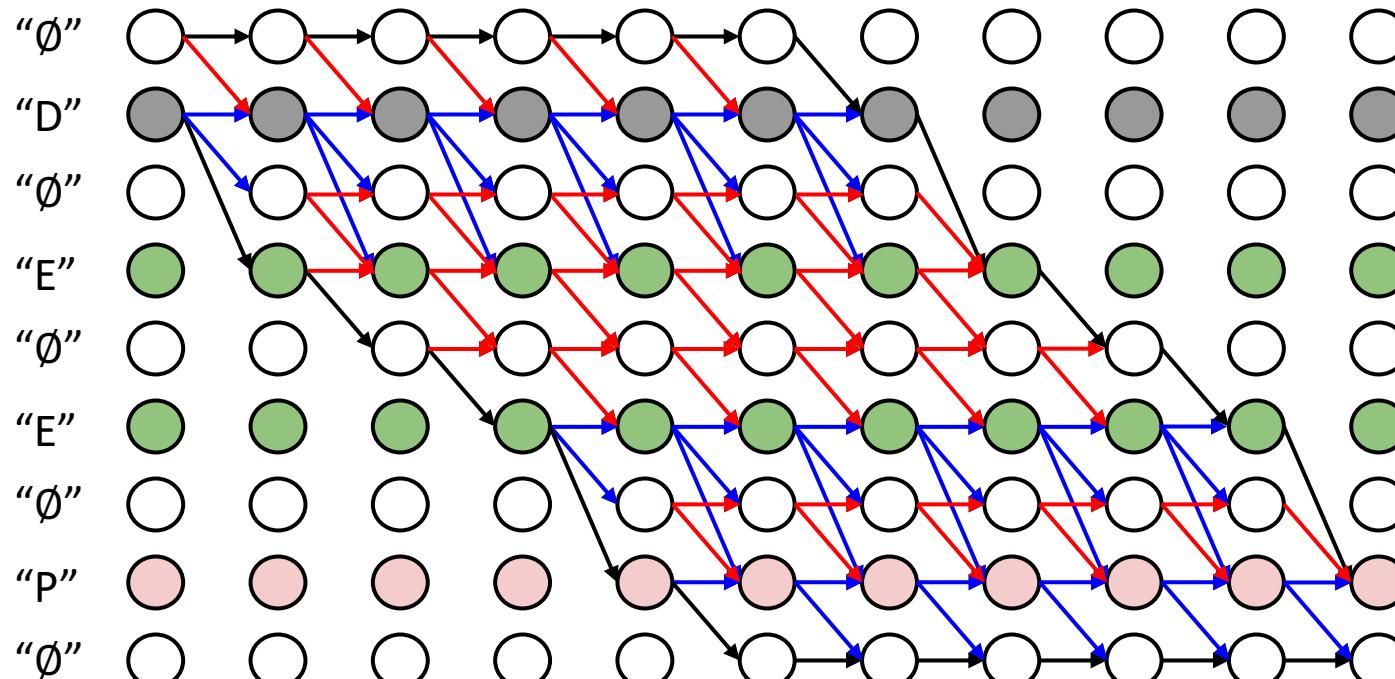
$$\alpha_t(s) \cdot y_t^s \cdot \boxed{\beta_t(s)}$$

summed probability  
of **all** the CTC paths  
**starting** at time  $t$  with

$$\beta_t(s) = y_t^{l_s} (\beta_{t+1}(s) + \beta_{t+1}(s+1)) \quad \text{if } l_s = \phi \text{ or } l_s = b_{s+2}^{\text{state}}$$

$$\beta_t(s) = y_t^{l_s} (\beta_{t+1}(s) + \beta_{t+1}(s+1) + \beta_{t+1}(s+2)) \quad \text{otherwise}$$

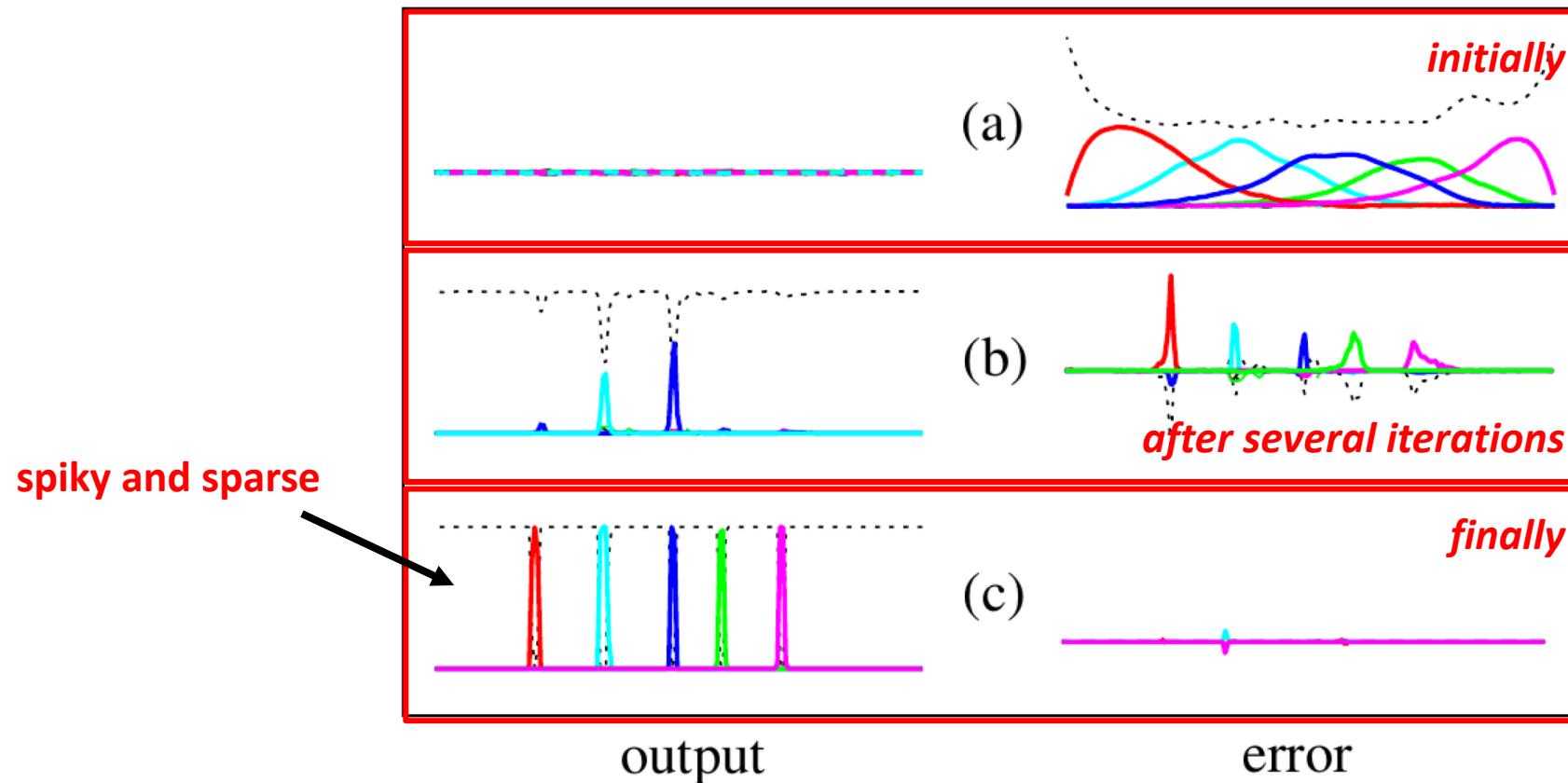
## CTC Loss



$$L_{CTC} = \ln Pr(Y|X) = \ln \sum_s \alpha_t(s) \beta_t(s)$$

compute gradients to  
update the weights

## Evolution of the CTC outputs and errors during training



## best path decoding

- compute the *best path* by taking the **most likely prediction per frame**

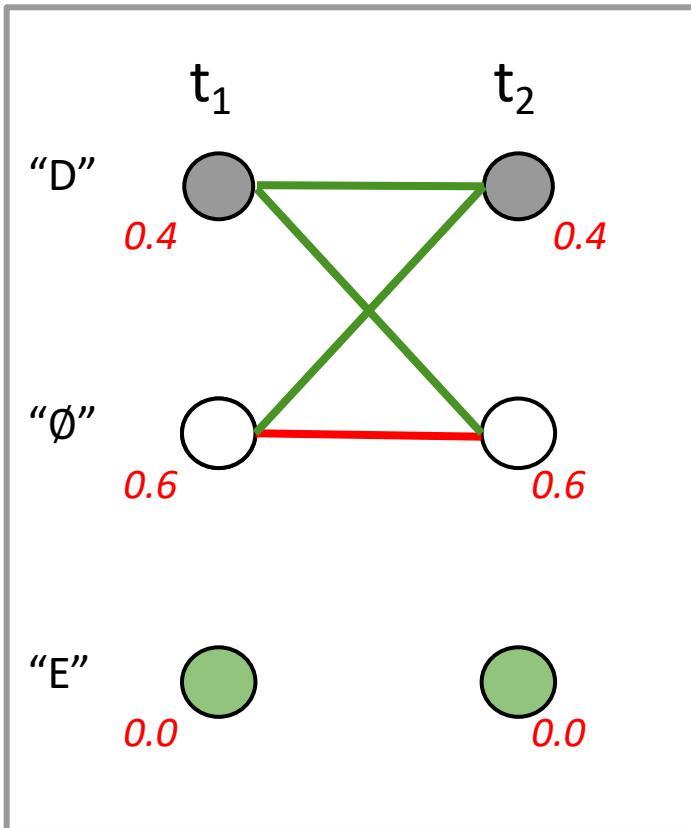
$$p^* = \operatorname{argmax} Pr(p|X)$$

- map the best path to a label sequence by **removing duplicated successive predictions** and by **removing all blanks** from the path

+ *simple and fast*

- *lead to errors in many practical situations*

*a toy example*



best path decoding outputs “blank”

$$\begin{aligned}\Pr(Y=\text{blank}) &= \Pr(p=\emptyset\emptyset) \\ &= 0.6 * 0.6 \\ &= 0.36\end{aligned}$$

$$\begin{aligned}\Pr(Y=D) &= \Pr(p=DD) + \Pr(p=D\emptyset) + \Pr(p=\emptyset D) \\ &= 0.4*0.4 + 0.4*0.6 + 0.6*0.4 \\ &= 0.64\end{aligned}$$

output labelling “D” is better in fact

decoding can be *improved* via:

- ❖ *beam search decoding*
  - iteratively create beam candidates and score them
- ❖ *prefix search decoding*
  - extend the label whose children have the largest cumulative probability
- ❖ *constrained decoding*
  - constrain the output labellings according to some predefined grammar

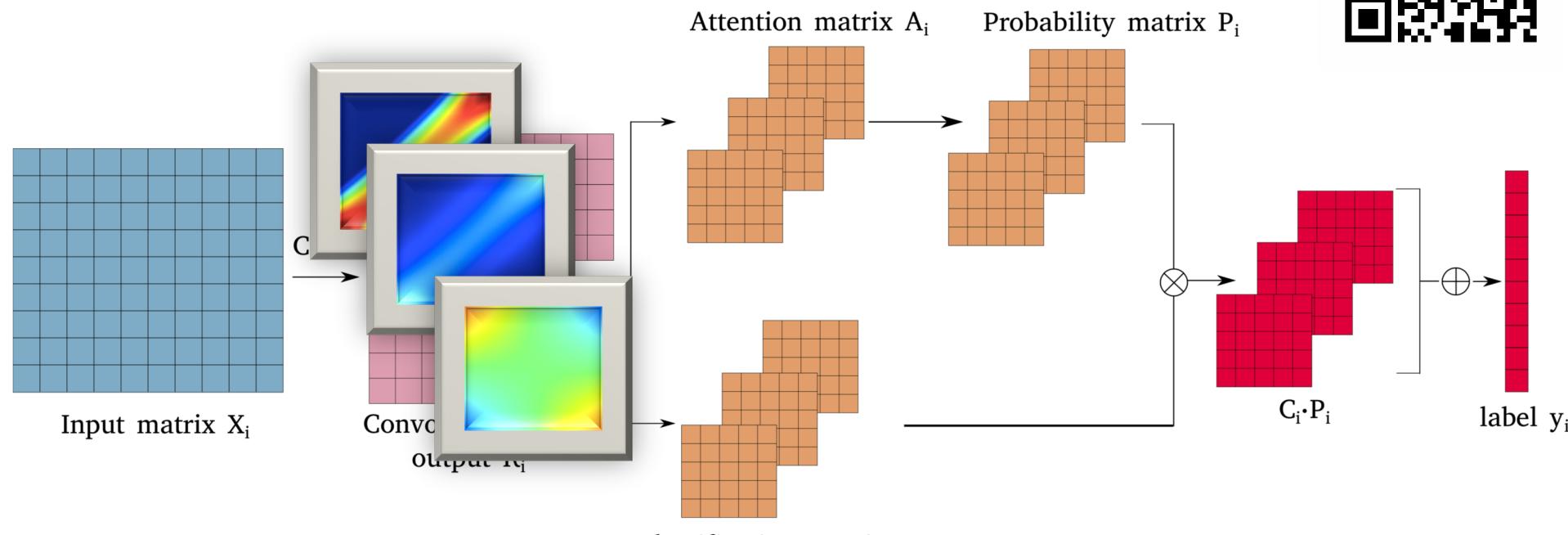
- CTC **does not require** aligned training data
- CTC defines forward-backward probability over all possible sequence
- training done with BPTT, evaluation done with decoding
- an **independence assumption** of outputs is made for efficiency
- cannot be used for models where outputs are dependent of previous outputs

Attention.



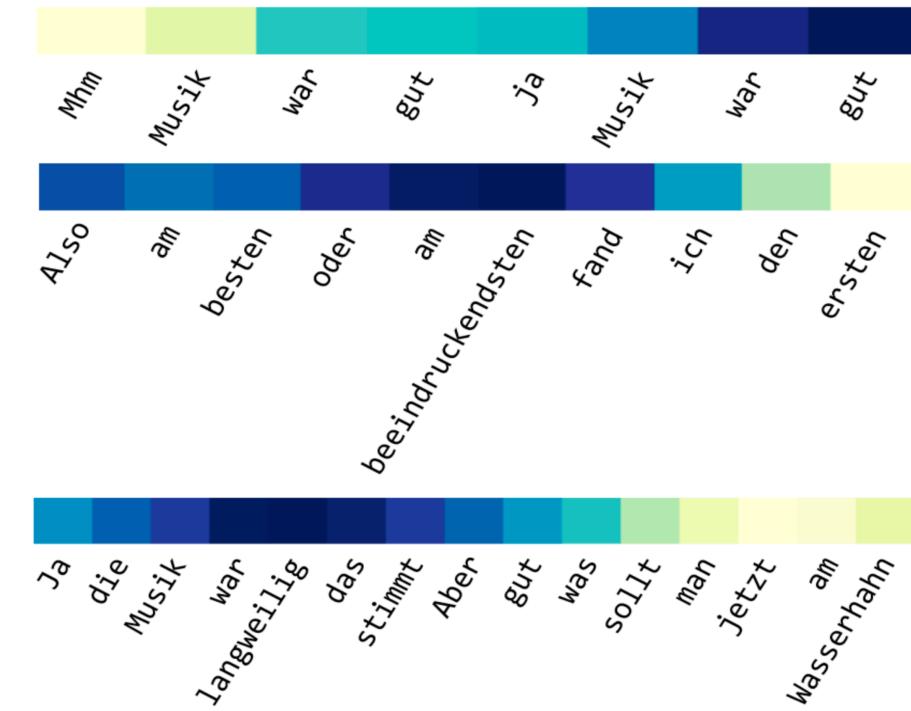
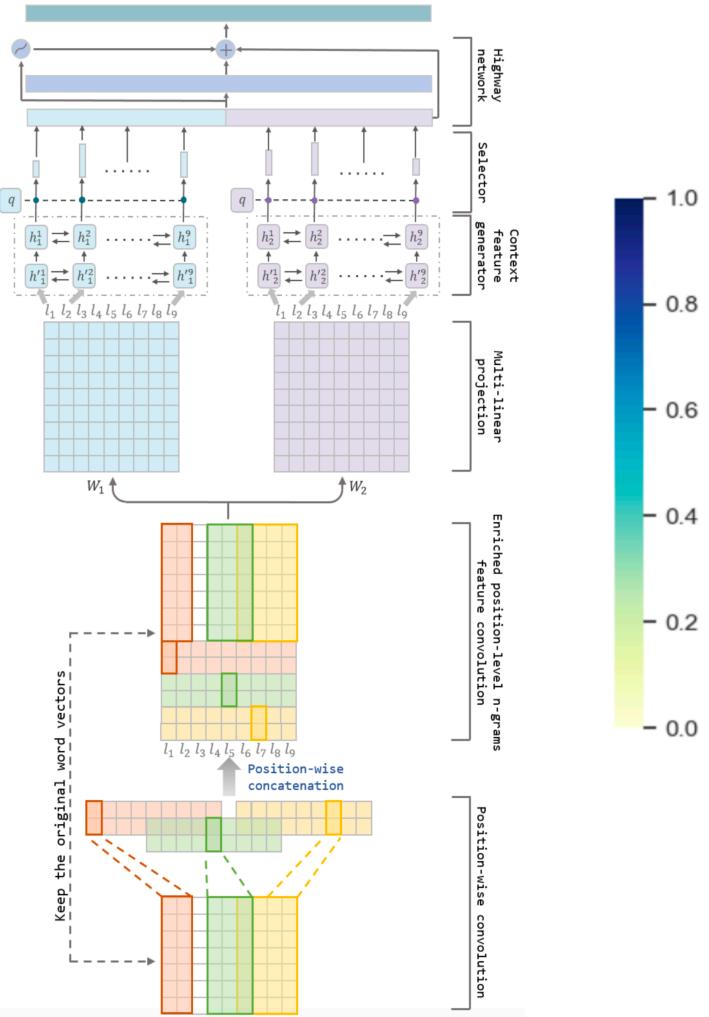
end2you >>||

- Self-Learning Representation: CNNs & Attention



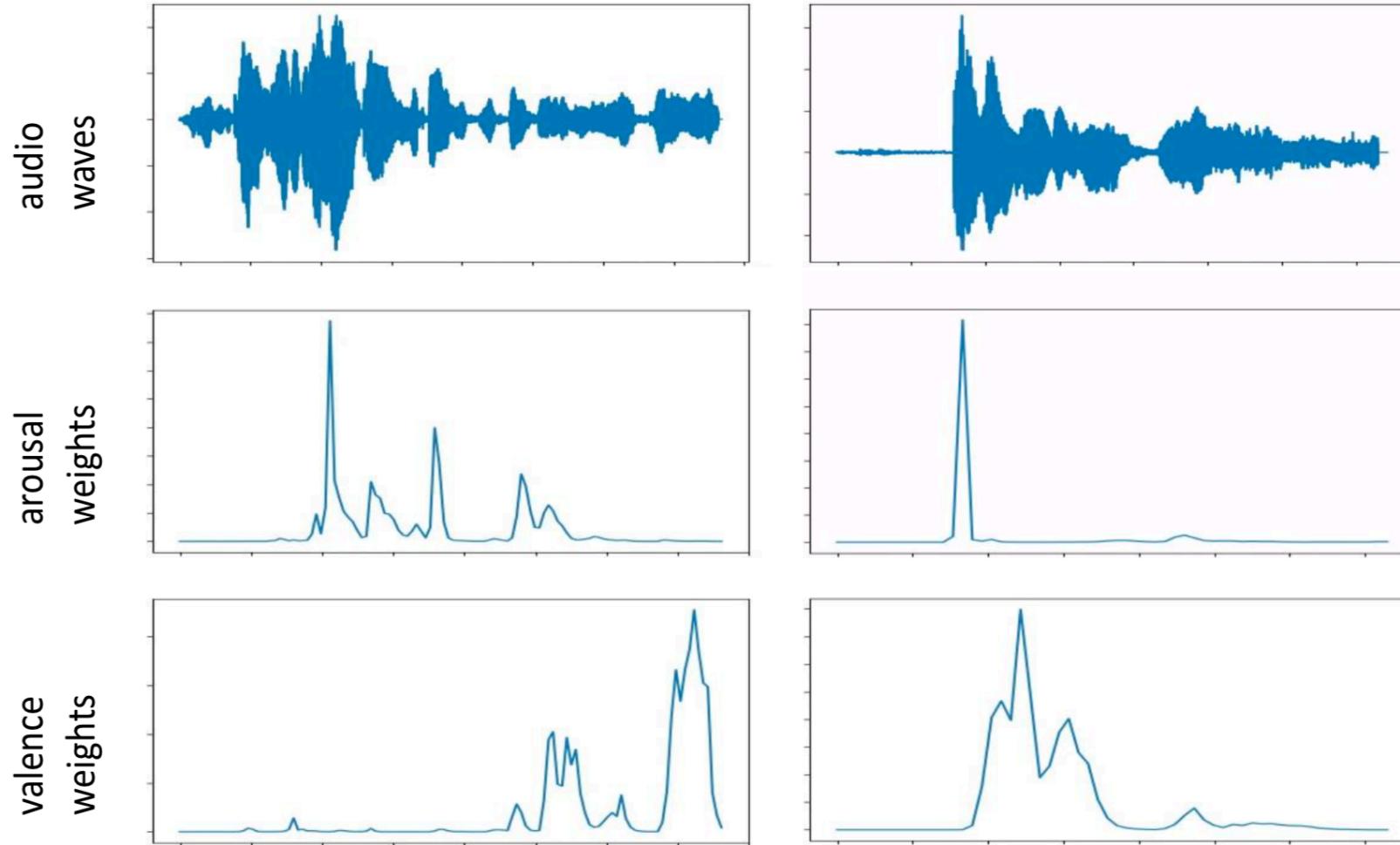
# Attention in CNN

Björn W. Schuller



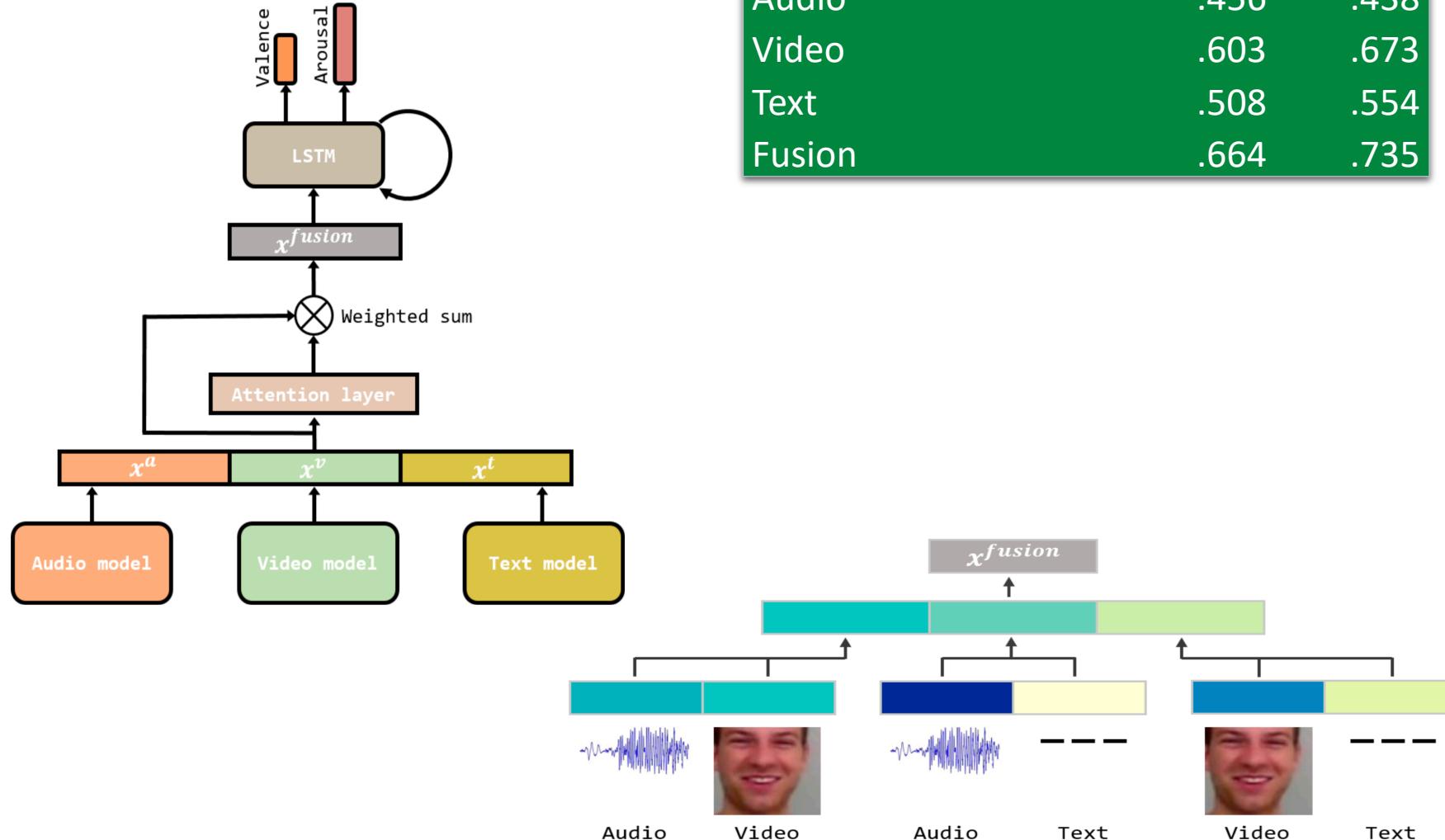
# Attention in RNN

Björn W. Schuller



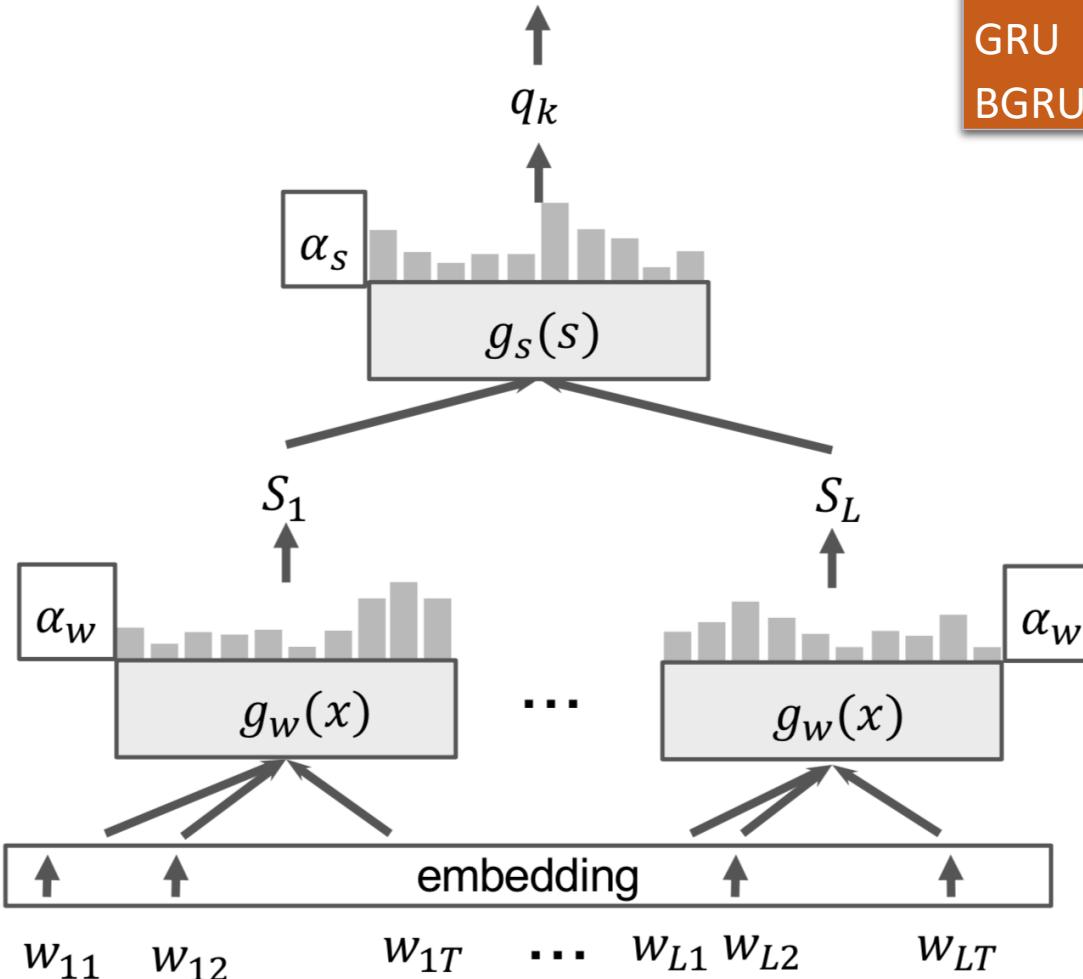
# Attention in NN

Björn W. Schuller



# Hierarchical Attention

Björn W. Schuller



	UAR [%] USoMS +/-	w/o Att.	w/ Att.
FC		81.4	85.2
GRU		86.5	88.5
BGRU		87.8	91.0

# Regularisation.



- **Common Deep Learning regularisation strategies**
  - Mini-Batch Learning
  - Weight Penalties
  - Data Augmentation
  - Training with Noise
  - Dropout
  - Early Stopping

- **Mini-Batch Learning**

- Performs an update for every batch of  $n$  training examples
- Reduces noise in variance of weight updates

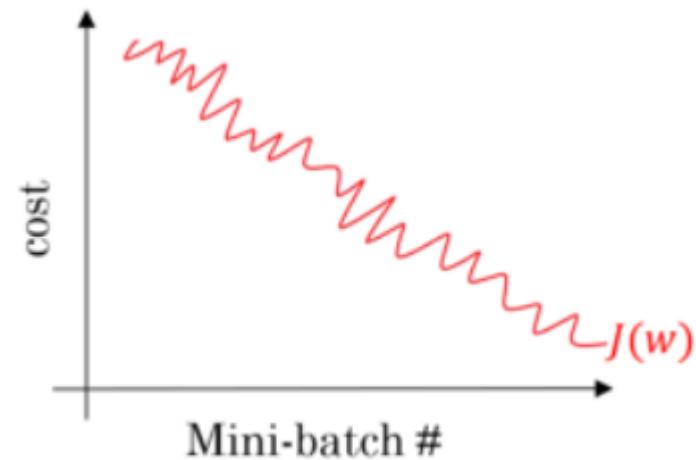
$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla J(\theta^{(t)}; x^{(i:i+n)}; y^{(i:i+n)})$$

- **Advantages**

- Stable convergence
- Good learning speed
- Good approximation minima location

- **Disadvantages**

- Total loss not accumulated



- **Mini-Batch Learning – Key Terminology**

- **Batch size**

- The number of training data points in each mini batch
      - Large enough to give good estimate of gradient
      - Small enough to provide suitable noise into updates

- **Number of batches**

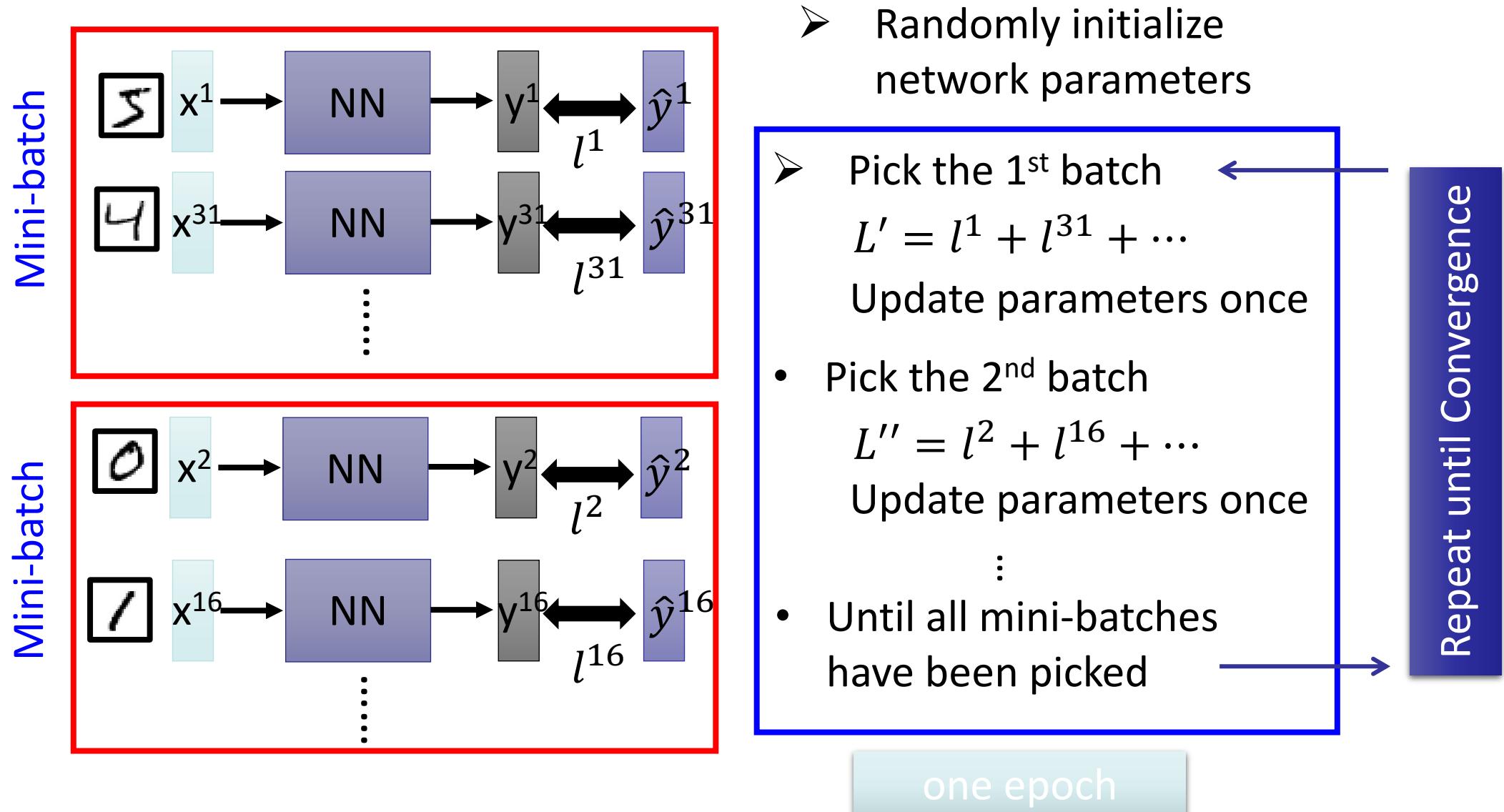
- The total number of mini-batches in the training data

- **Epoch**

- One epoch consists of one full pass of training over the dataset.
    - One epoch would consist of  $n$  number of (forward pass + backpropagation) where  $n$  denotes the number of batches.

# Regularisation – Mini-Batch Learning

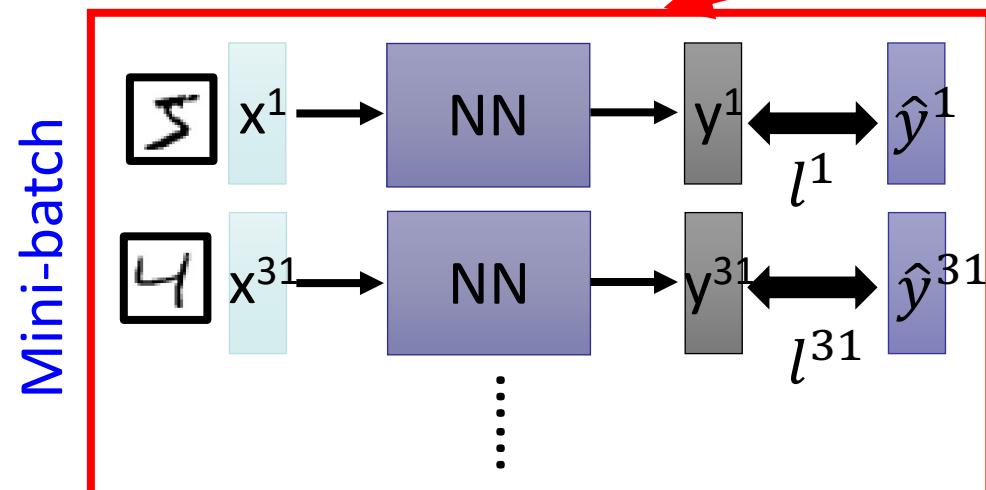
Björn W. Schuller



# Regularisation – Mini-Batch Learning

Björn W. Schuller

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```



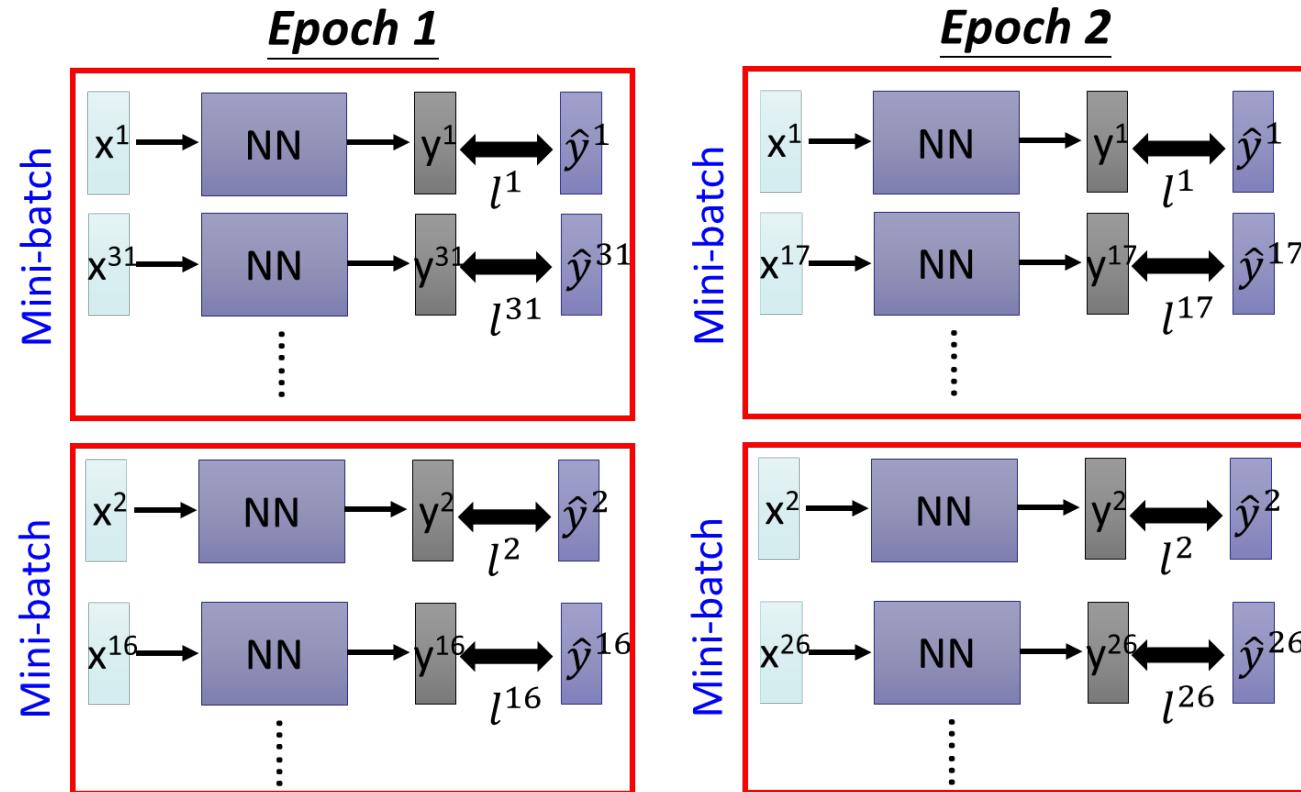
100 examples in a mini-batch

Repeat 20 times

- Pick the 1<sup>st</sup> batch  
 $L' = l^1 + l^{31} + \dots$   
Update parameters once
  - Pick the 2<sup>nd</sup> batch  
 $L'' = l^2 + l^{16} + \dots$   
Update parameters once
  - ⋮
  - Until all mini-batches have been picked
- one epoch

- **Shuffle Training Instances within epochs**

- Help avoid gradient descent getting stuck in local minima



- **Weight Penalties**

- Addition of a weight penalty term to the cost function

$$\tilde{J}(\theta; x, y) = J(\theta; x, y) + \alpha\Omega(\theta)$$

- Large weights make networks unstable
    - Minor variation or statistical noise on the expected inputs will result in large differences in the output
  - Aim of penalty term is to encourage the model to map the inputs to the outputs of the training dataset in such a way that the weights of the model are kept small

- **Common penalty terms**

- **L1 Norm**

- The sum of the absolute values of the weights
    - L1 encourages weights to be zero if possible
    - Resulting in more sparse weights
      - Weights with more zeros values

- **L2 Norm**

- The sum of the squared values of the weights
    - Penalizes larger weights

$$\alpha\Omega(\theta) = \|\theta\|_1$$

$$= \sum_n |\theta_n|$$

$$\alpha\Omega(\theta) = \frac{1}{2} \|\theta\|_2^2$$

$$= \sqrt{\sum_n |\theta_n|^2}$$

- **Data Augmentation**

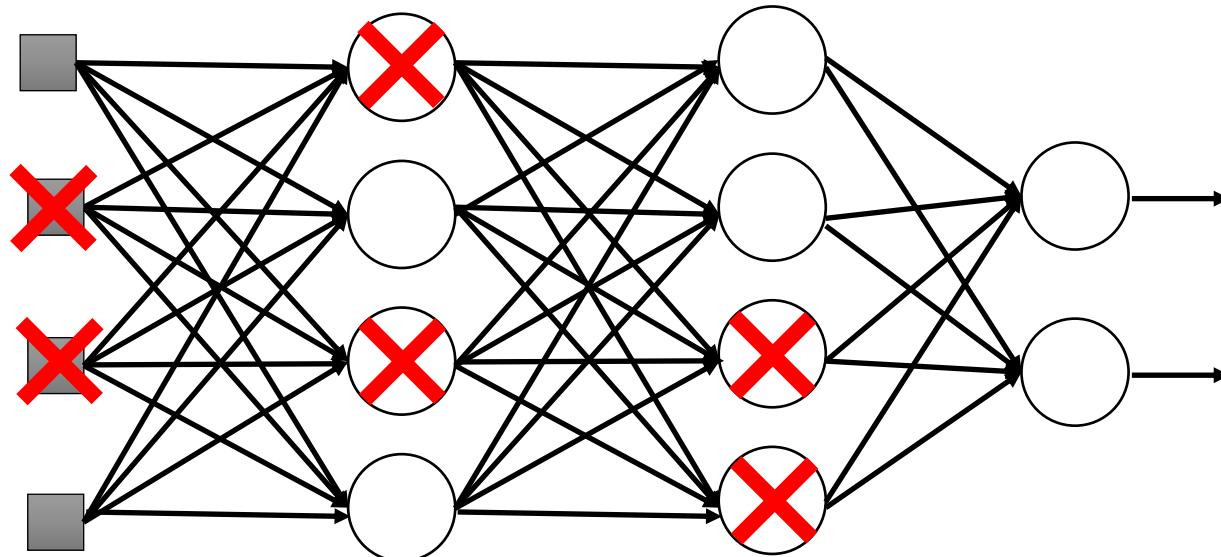
- The best way to make a machine learning model generalise better is to train it on more data
- One way to get around this problem is to create fake data and add it to the training set
  - Trivial with images: shifts, flips, zooms, rotations
  - Adding noise is a form of augmentation
- One must be careful not to apply transformations that would change the correct class.
  - Flips and rotations not useful when recognising the difference between “b” and “d” or the difference between “6” and “9”,

- **Training with noise**
  - **Adding noise means that the network is less able to memorise training samples**
    - Input data is changing all of the time
    - Results in smaller network weights and a more robust network that has lower generalisation error
    - Typical to add noise to input data
      - White Gaussian noise with mean of 0 and a standard deviation of 1
      - Generated as needed using a pseudorandom number generator.
      - We can also add noise activations, to weights, to the gradients and to the outputs

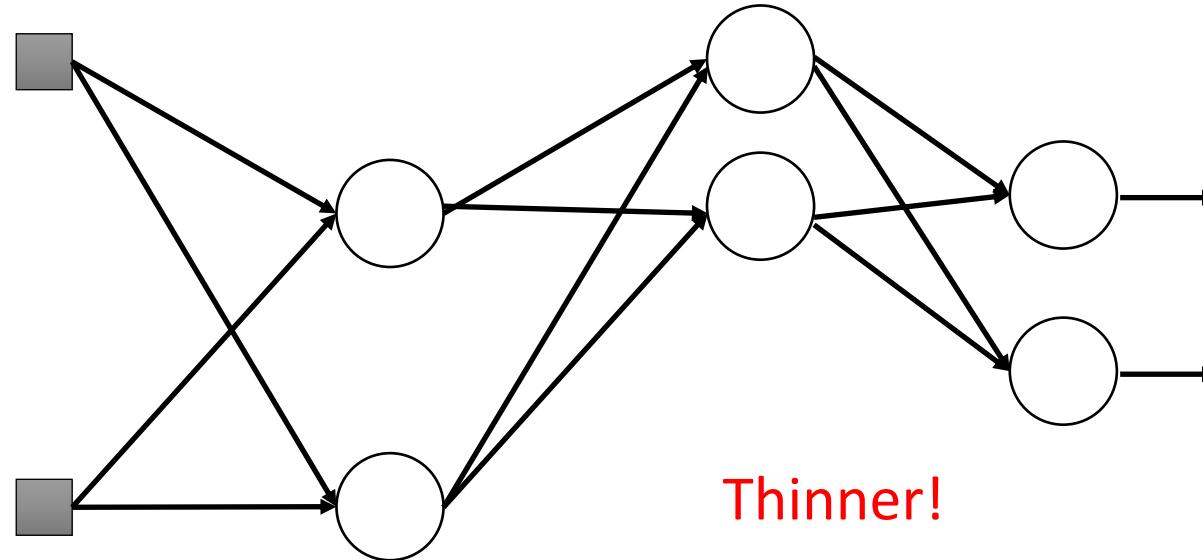
- **Dropout**

- Method for training a ensemble of slightly different networks and averaging them
- **Underlying concept:**
  - Although it's likely that large, unregularized neural networks will overfit to noise, it's unlikely they will overfit to the *same* noise
    - I.e. they will make slightly different mistakes
  - Averaging will cancel out the *differing* mistakes revealing what they all learned in common: *the signal properties*
- The ensemble of subnetworks is formed by randomly removing nonoutput units during training

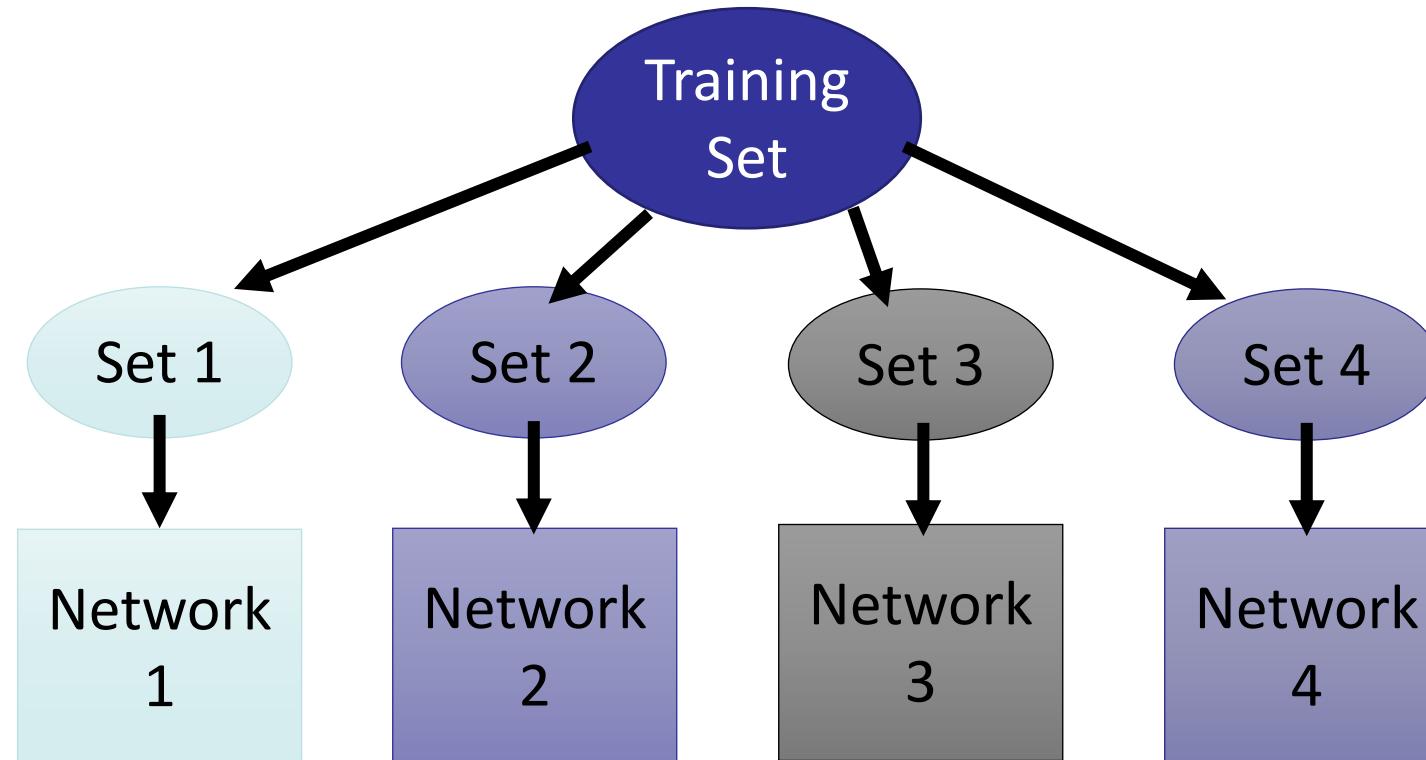
- **Training with Dropout**
  - Random neurons before updating the parameters
  - Each neuron has  $p\%$  to dropout
    - $p$  is a hyperparameter chosen before training



- **Training with Dropout**
  - **The structure of the network is changed**
    - Continue training with this new network
  - **For each mini-batch, we resample the dropout neurons**

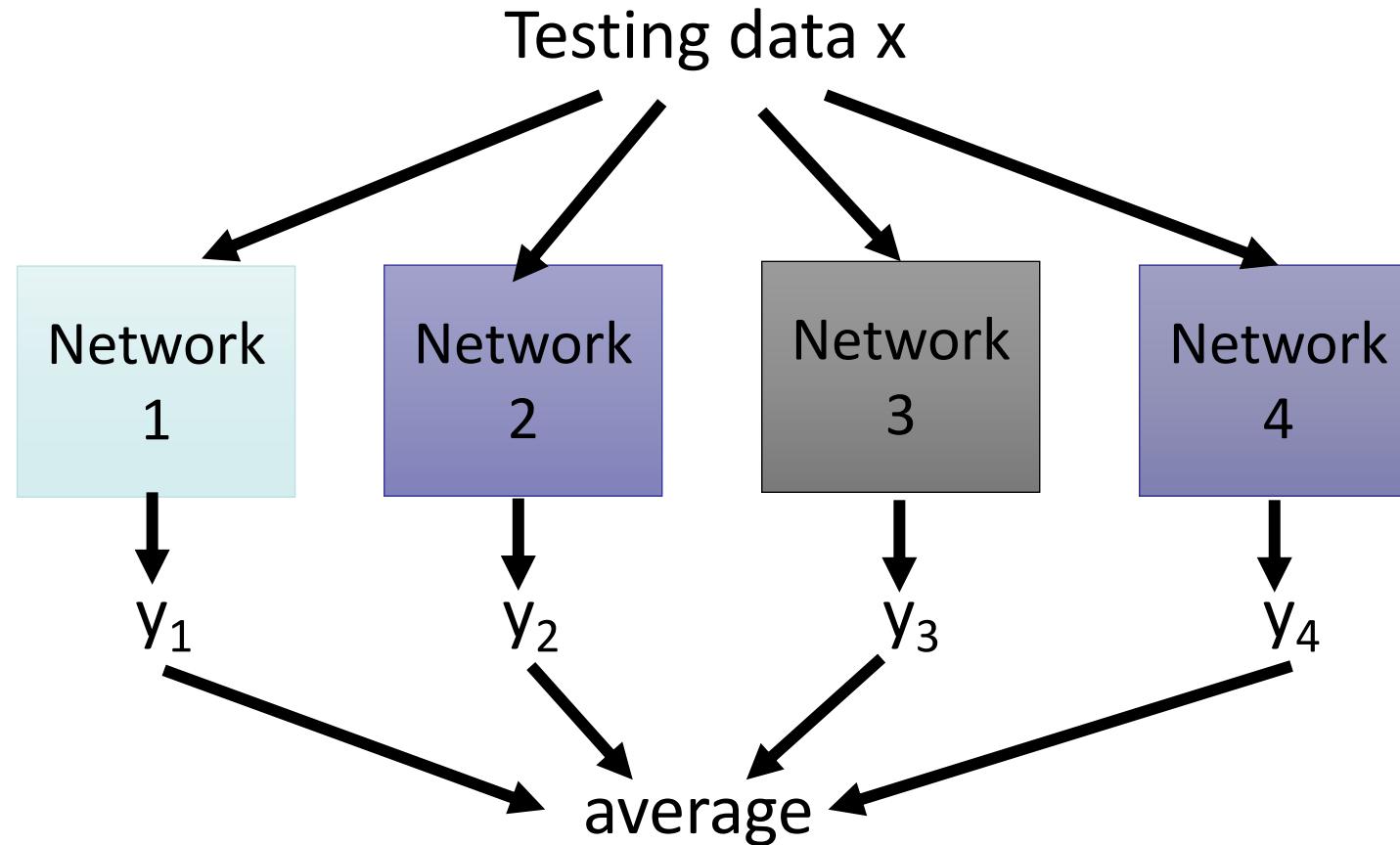


- Dropout is a form of ensemble learning



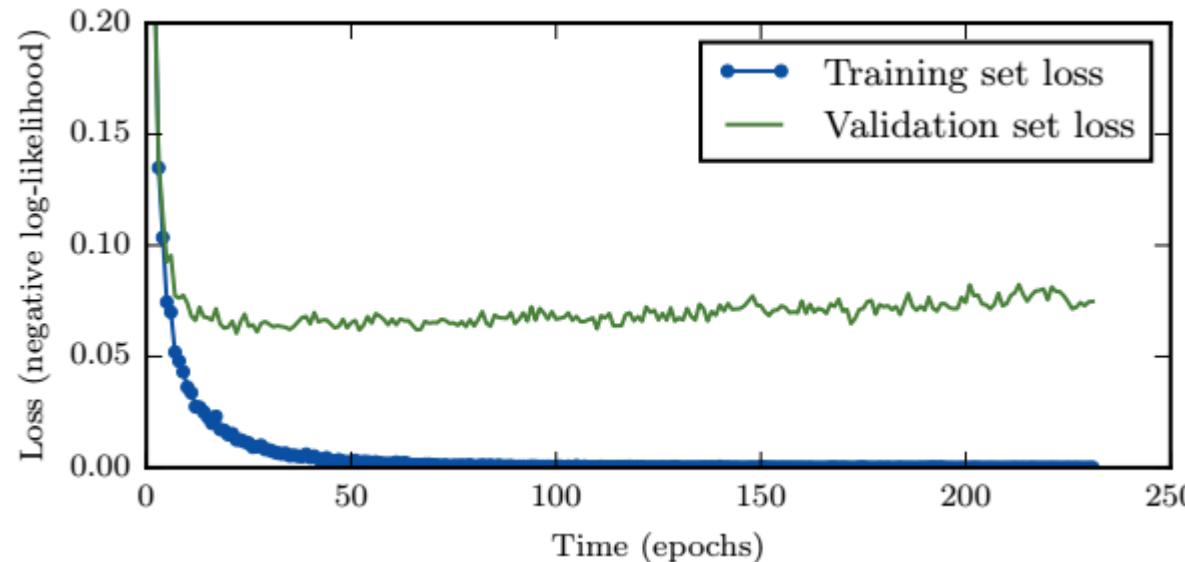
Train a bunch of networks with different structures

- Averaging of ensemble improves generalisability

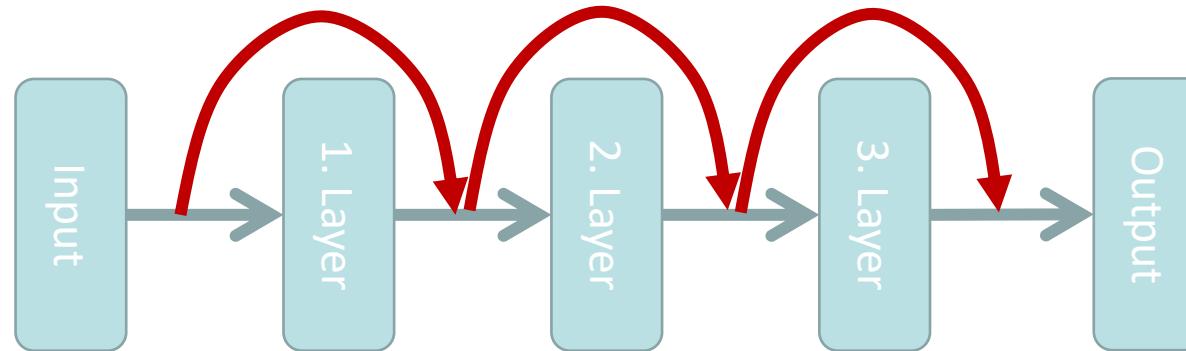


- **Early Stopping**

- During training, the model is evaluated on a holdout validation dataset after each epoch
- If the performance of the model on the validation dataset starts to degrade then the training process is stopped



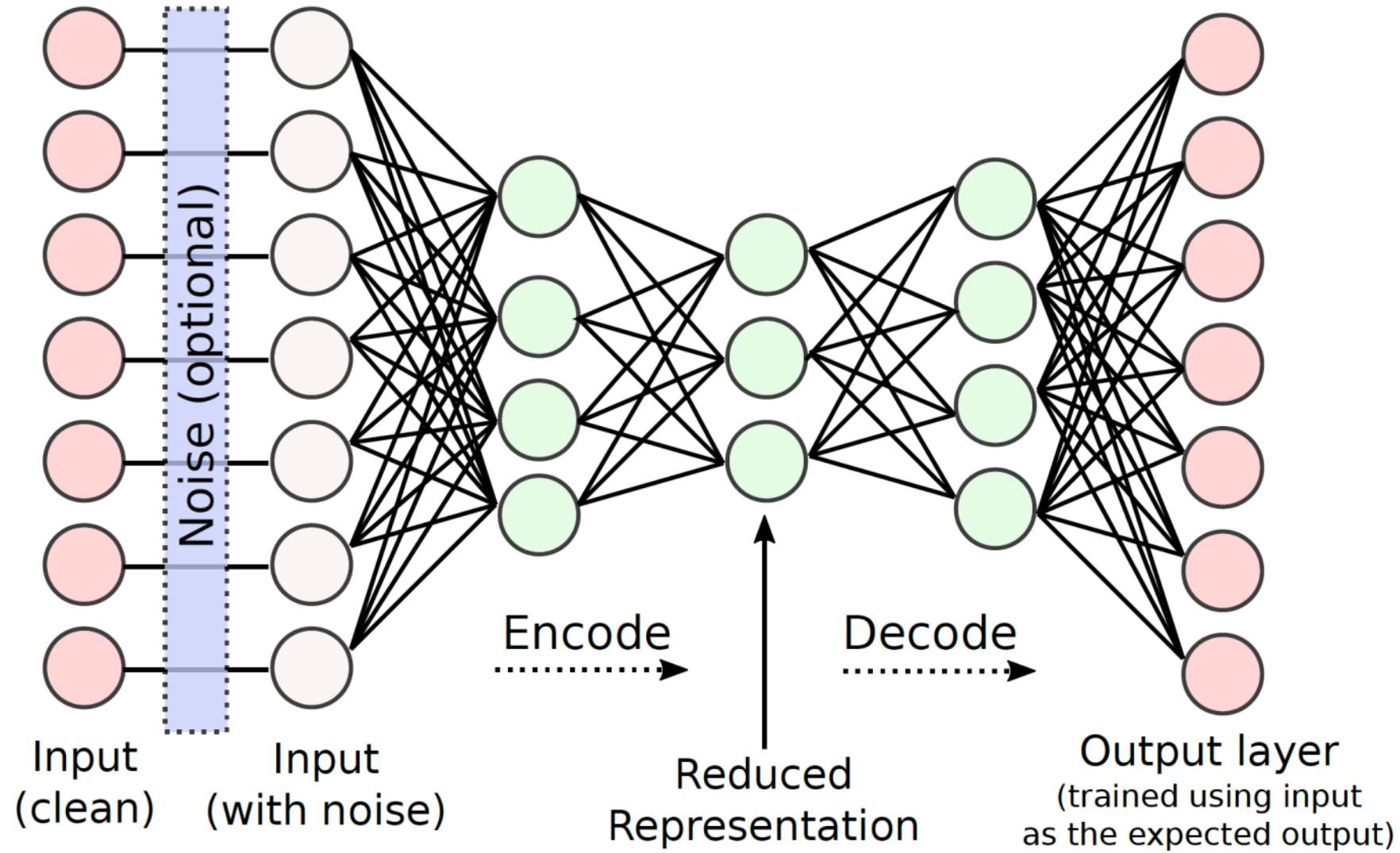
- How can later layers receive useful information already at start of training?
- Solution: Short-cuts for data



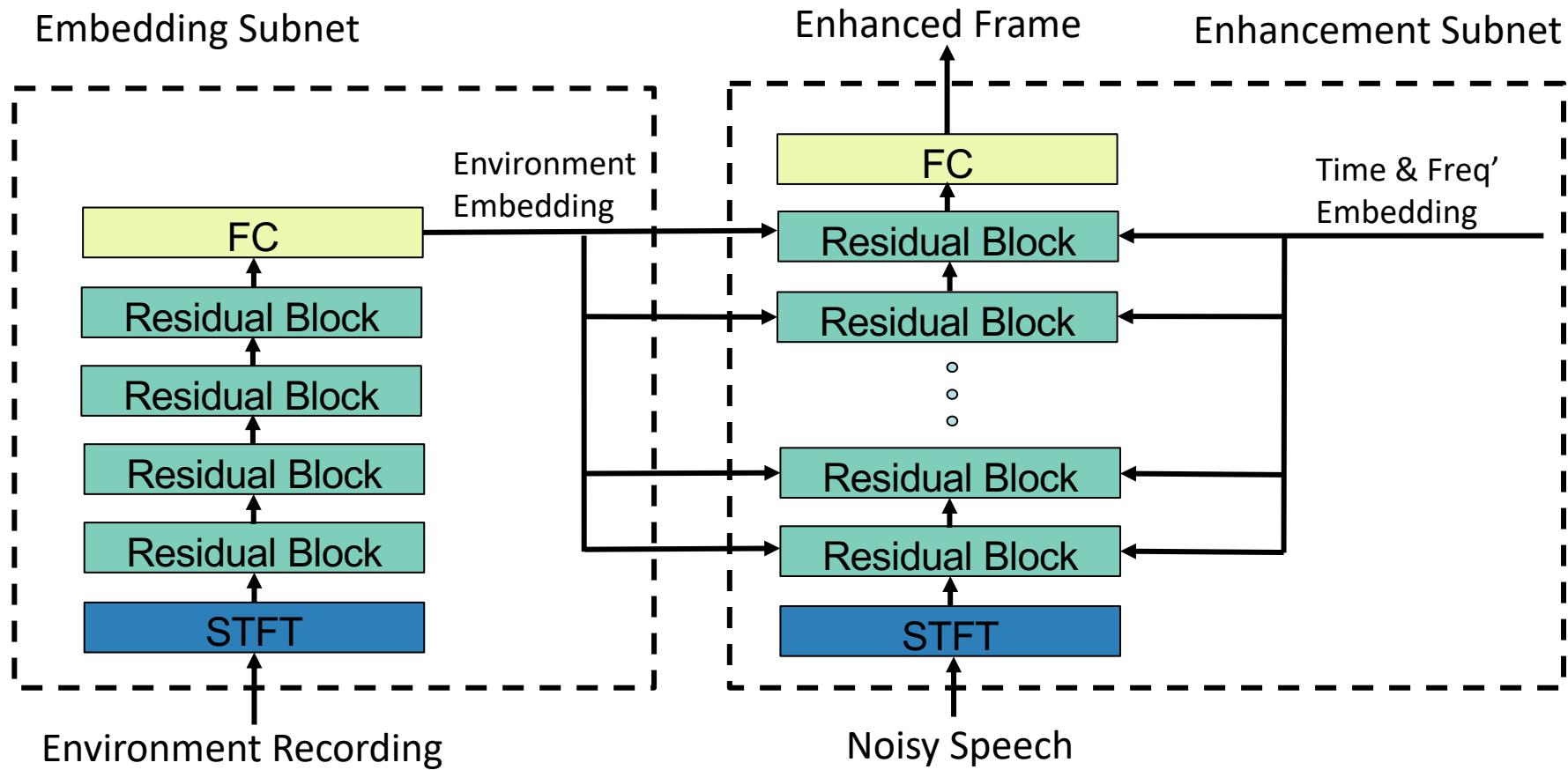
End-to-End .



## 1: Preprocessing: Enter AEs.

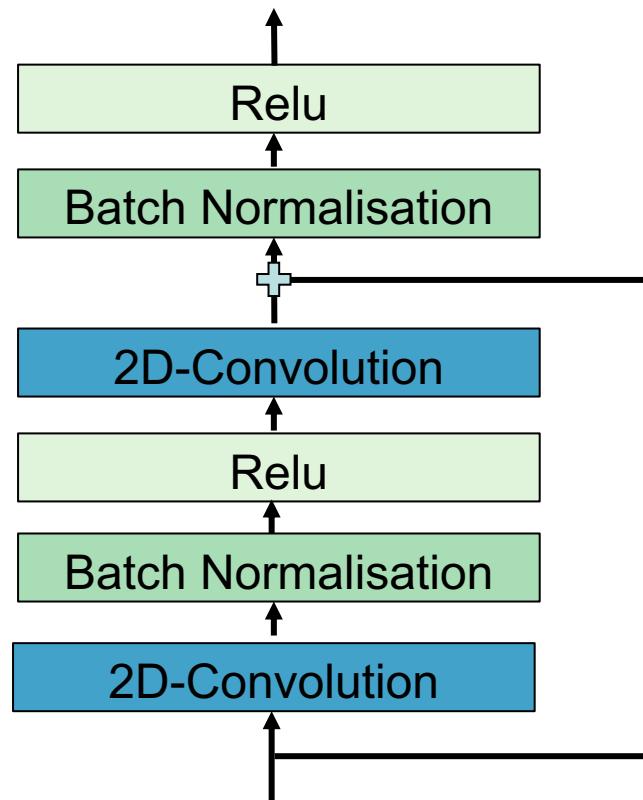


## 1: Enhancing.

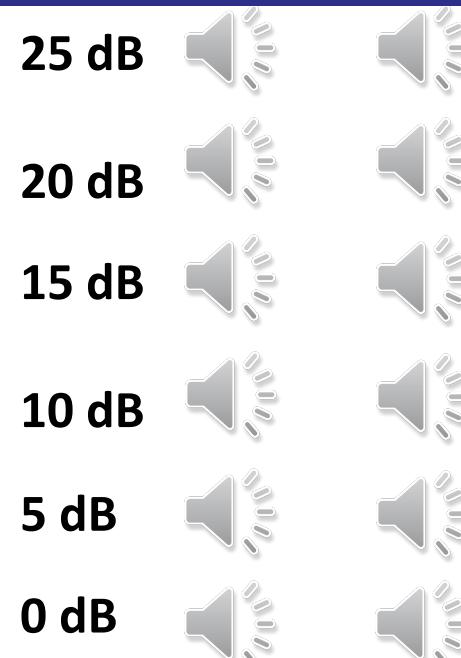


## 1: Enhancing.

A Residual Block:

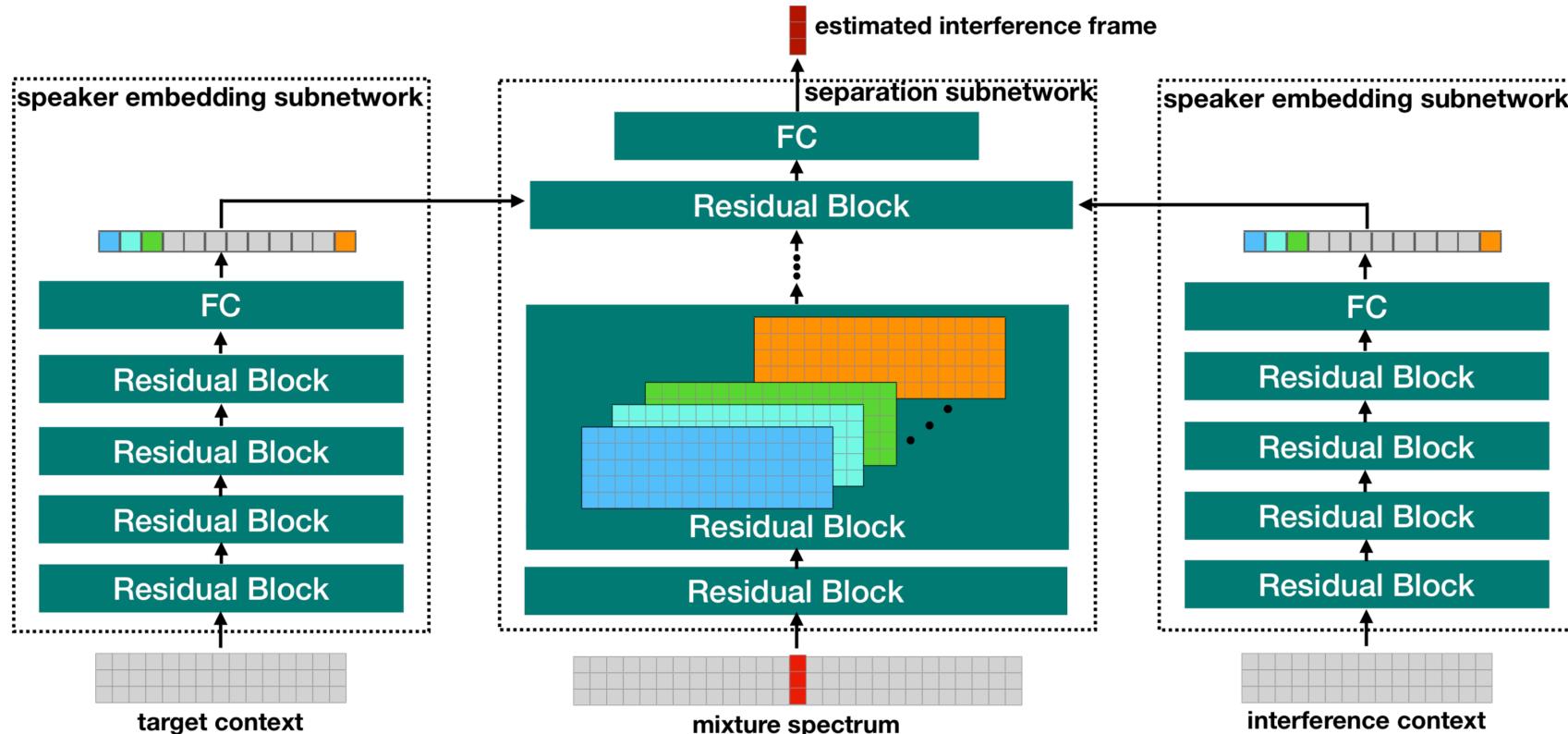


	WER	PESQ	SegSNR	LSD
Clean Speech	4.21	–	–	–
Noisy Speech	34.04	2.59	7.02	0.94
Log-MMSE	35.38	2.66	7.12	0.91
Noise Aware	25.30	2.96	11.01	0.54
w/o Embed.	16.78	3.25	11.71	0.48
w/ Embed.	15.46	3.30	12.99	0.45

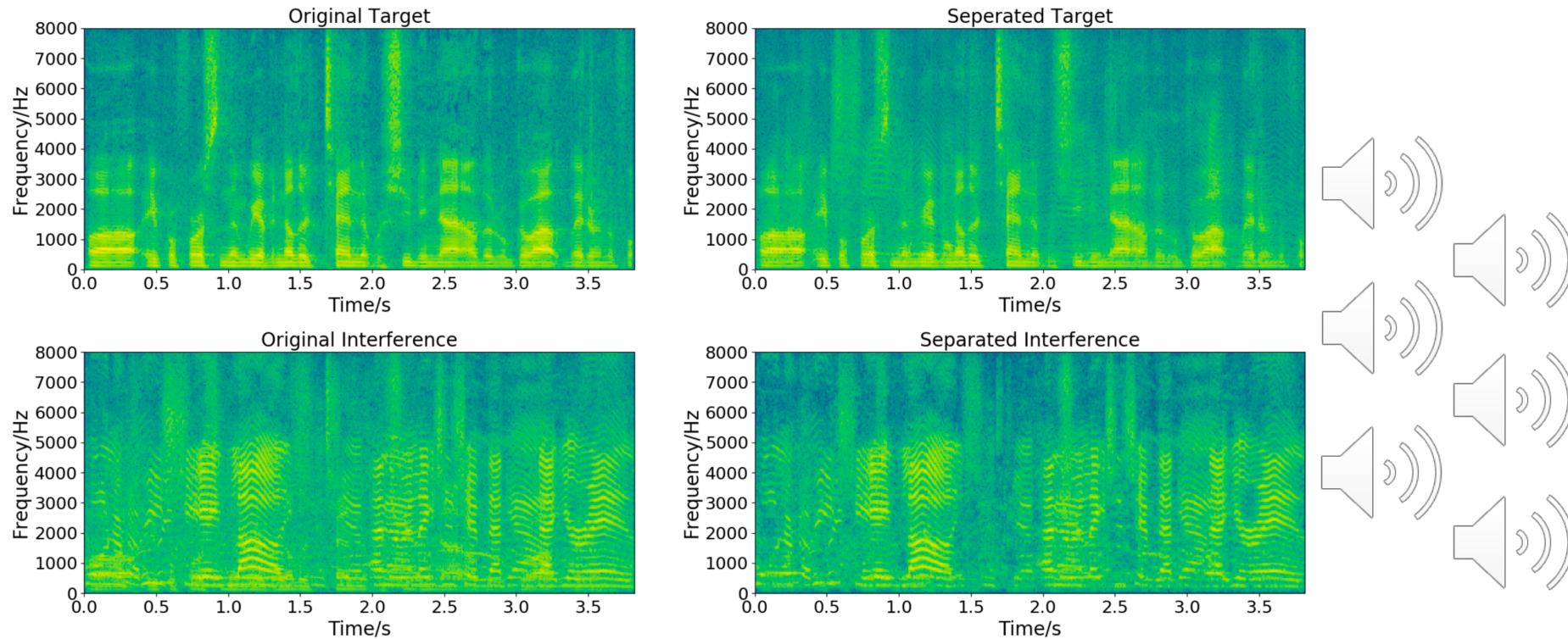


## Enhancement.

	SDR	SAR	SIR
DC	0.84	<b>2.09</b>	6.58
DaNet	1.81	3.29	10.41
Here	4.79	8.44	7.11

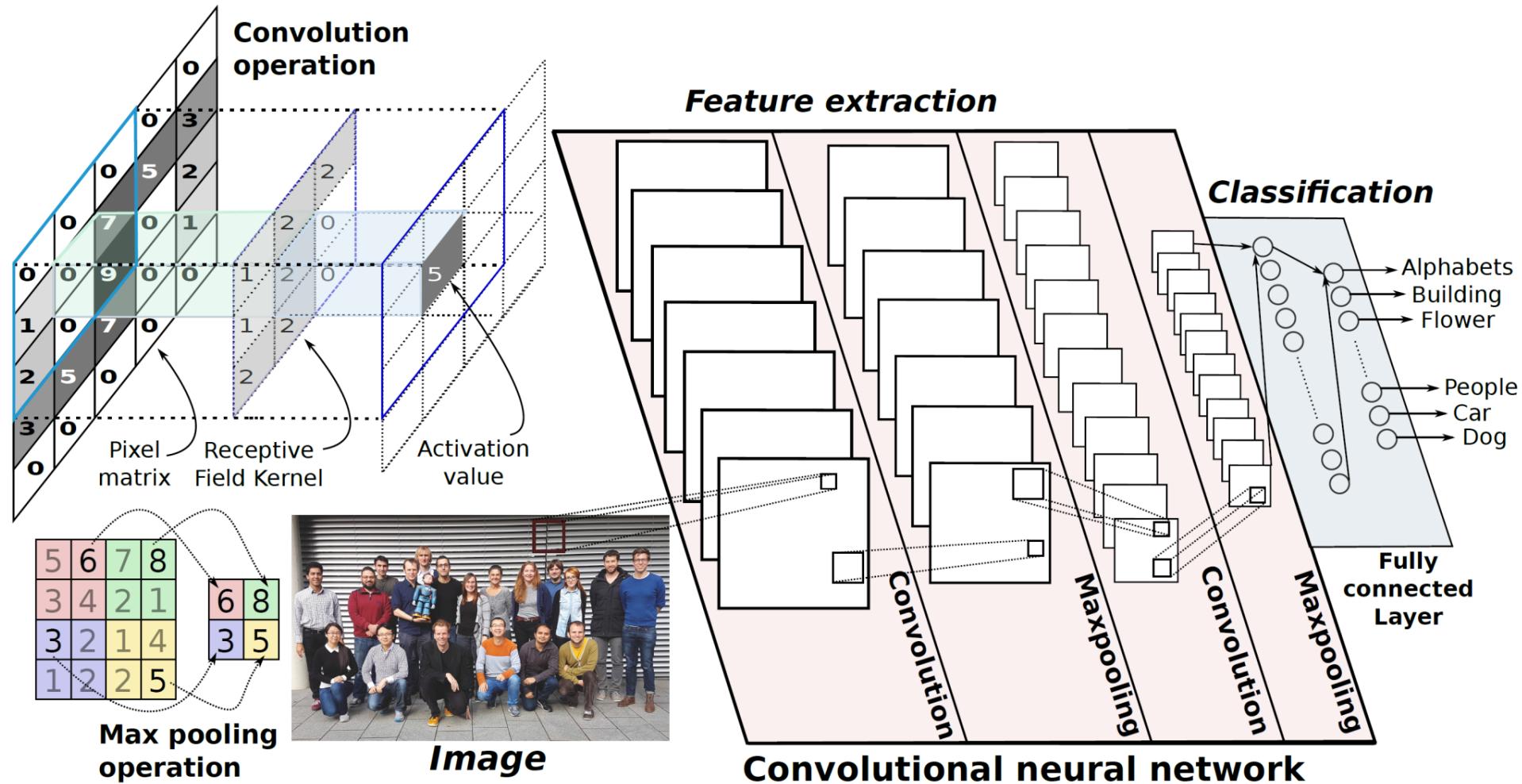


## Enhancement.

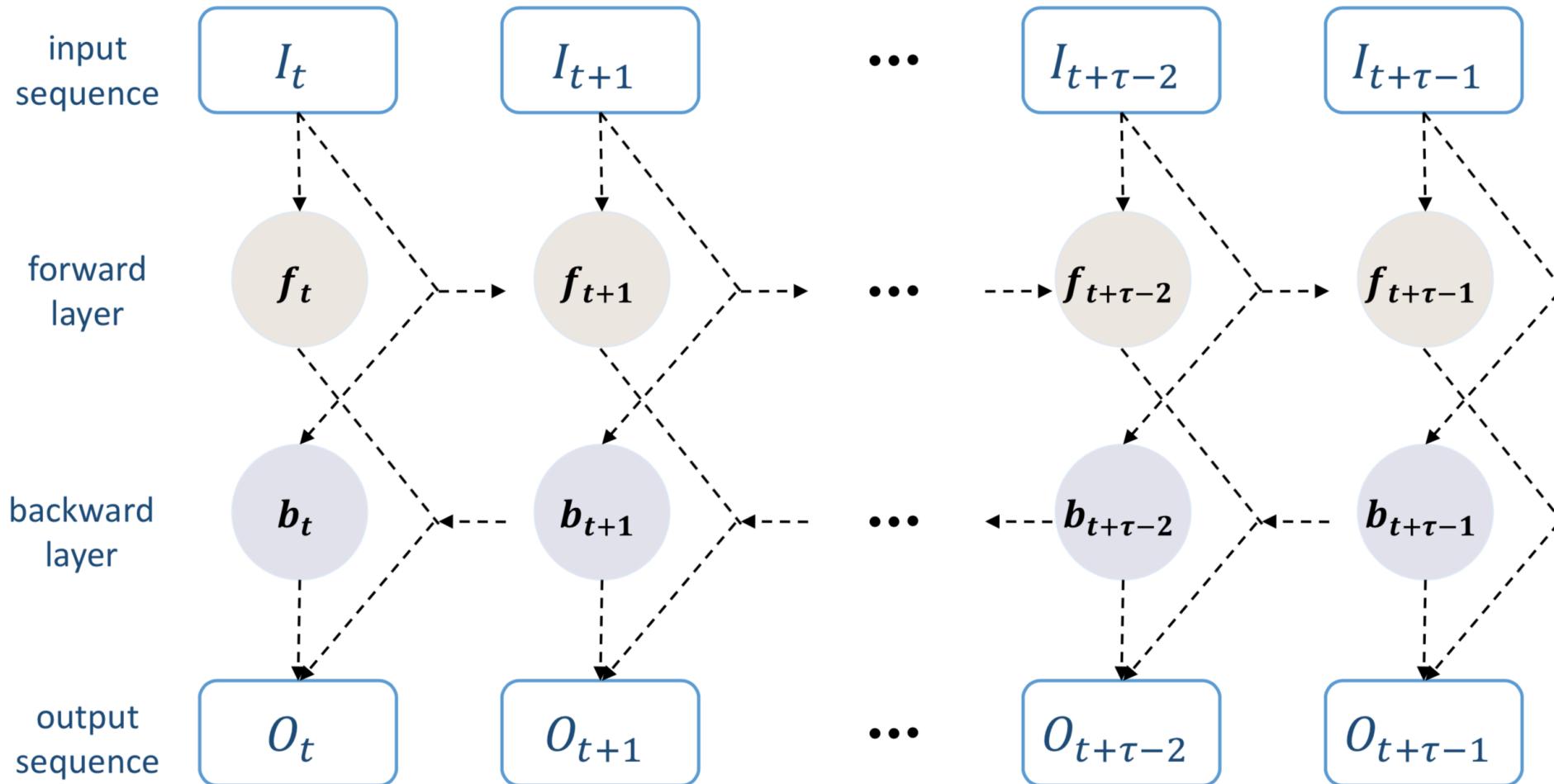


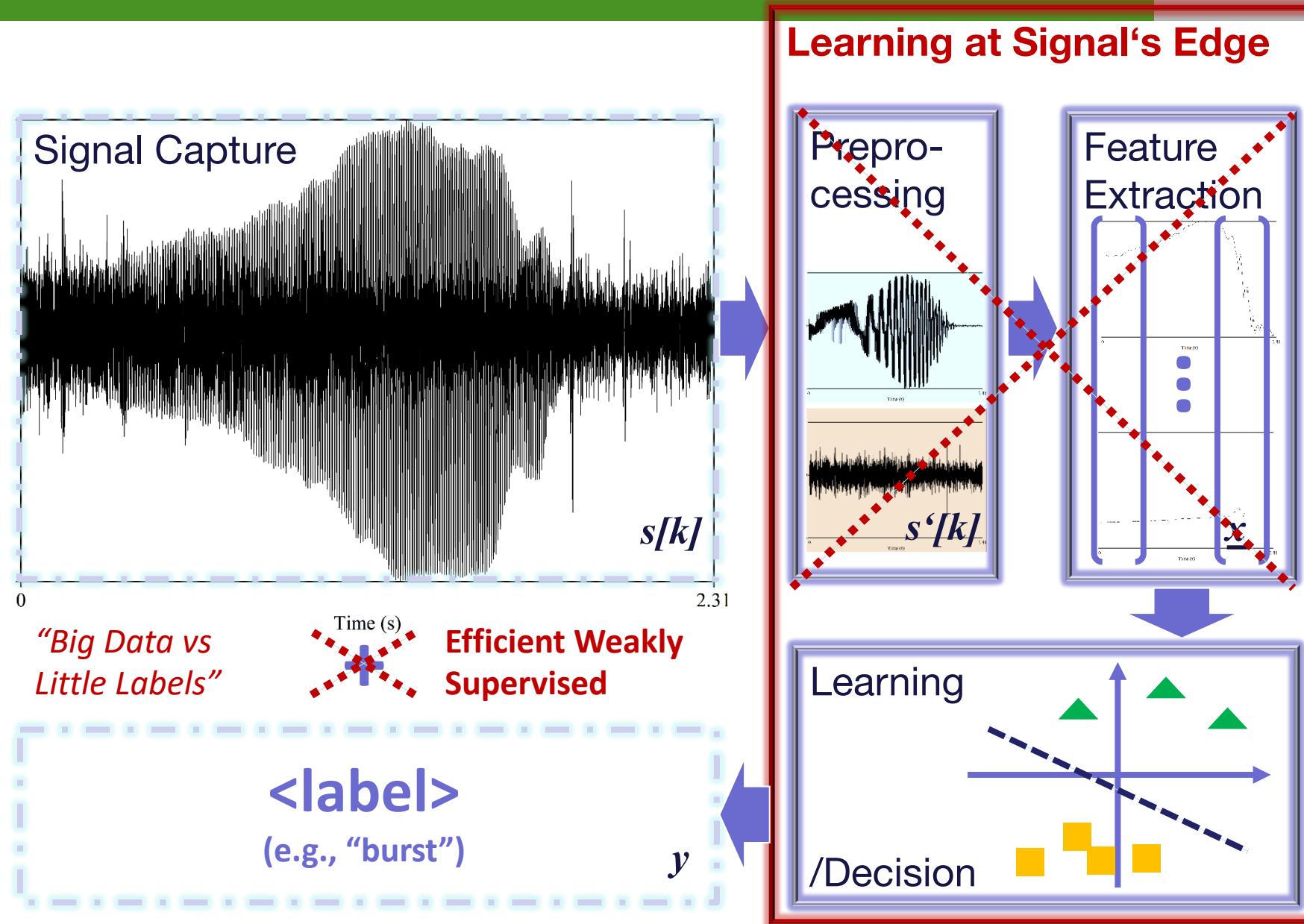
*"Single-Channel Speech Separation with Auxiliary Speaker Embeddings ", submitted.*

## 2: Feature Extraction: Enter CNNs.

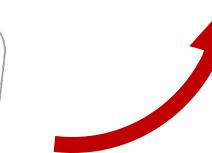
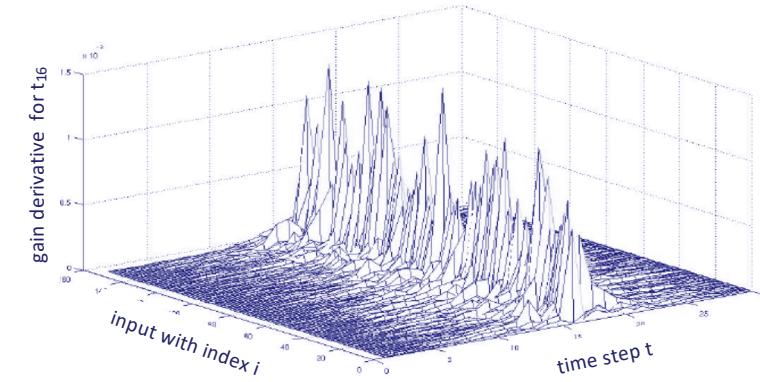
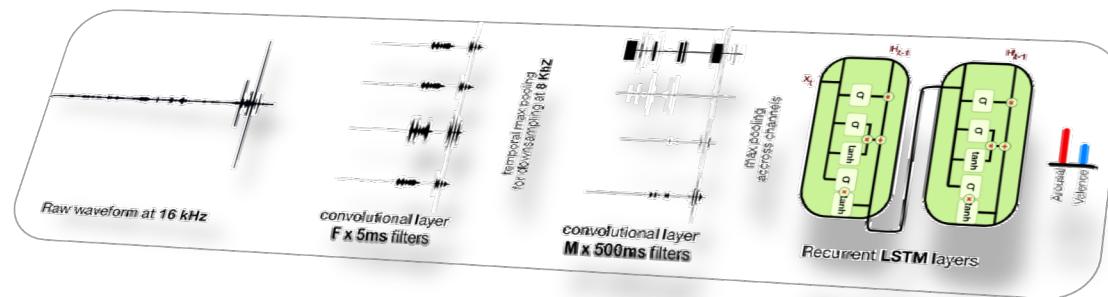


### 3: Decision Making.

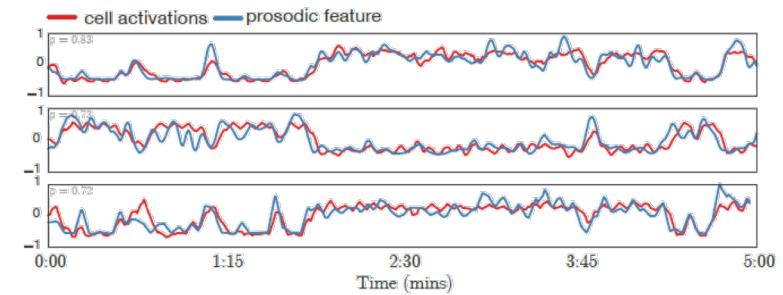




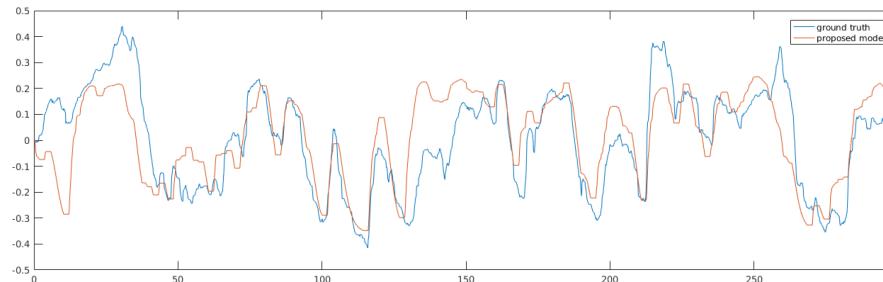
- Black Box?



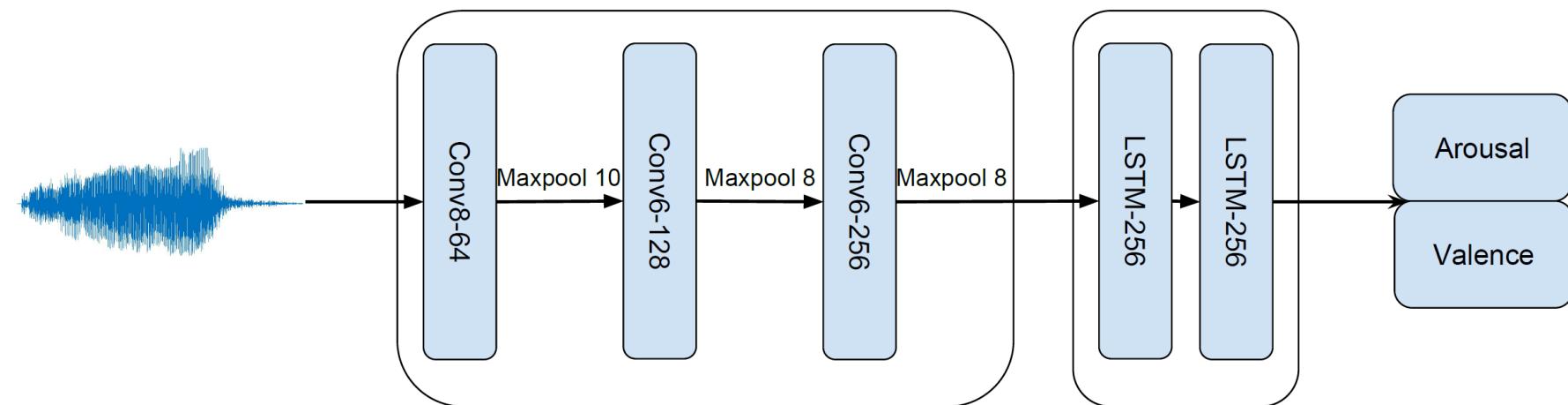
energy range (.77)  
loudness (.73)  
F0 mean (.71)



- CNN + LSTM RNN



	Arousal	Valence
CCC Recola		
ComParE+LSTM	.382	.187
e2e (2016)	.686	.261
BoAW	.753	.430
e2e (2018)	.787	.440



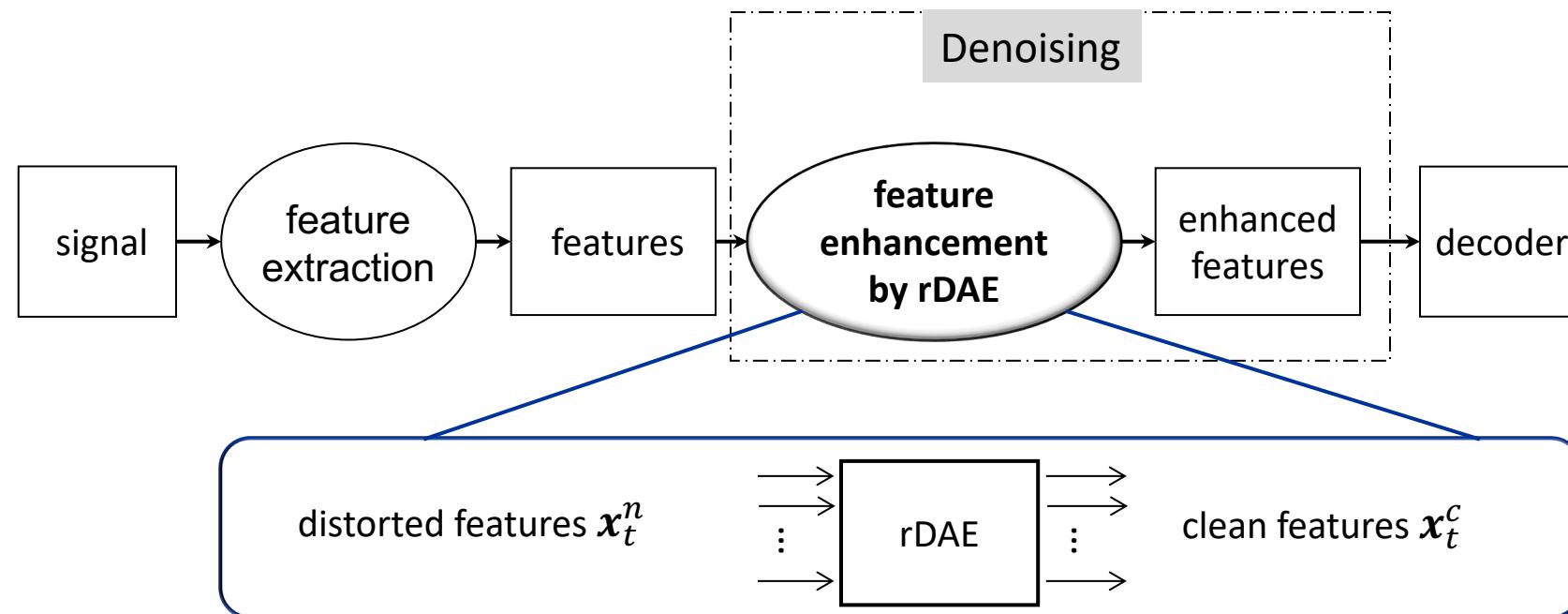
Input raw waveform at 16kHz

Convolution layers

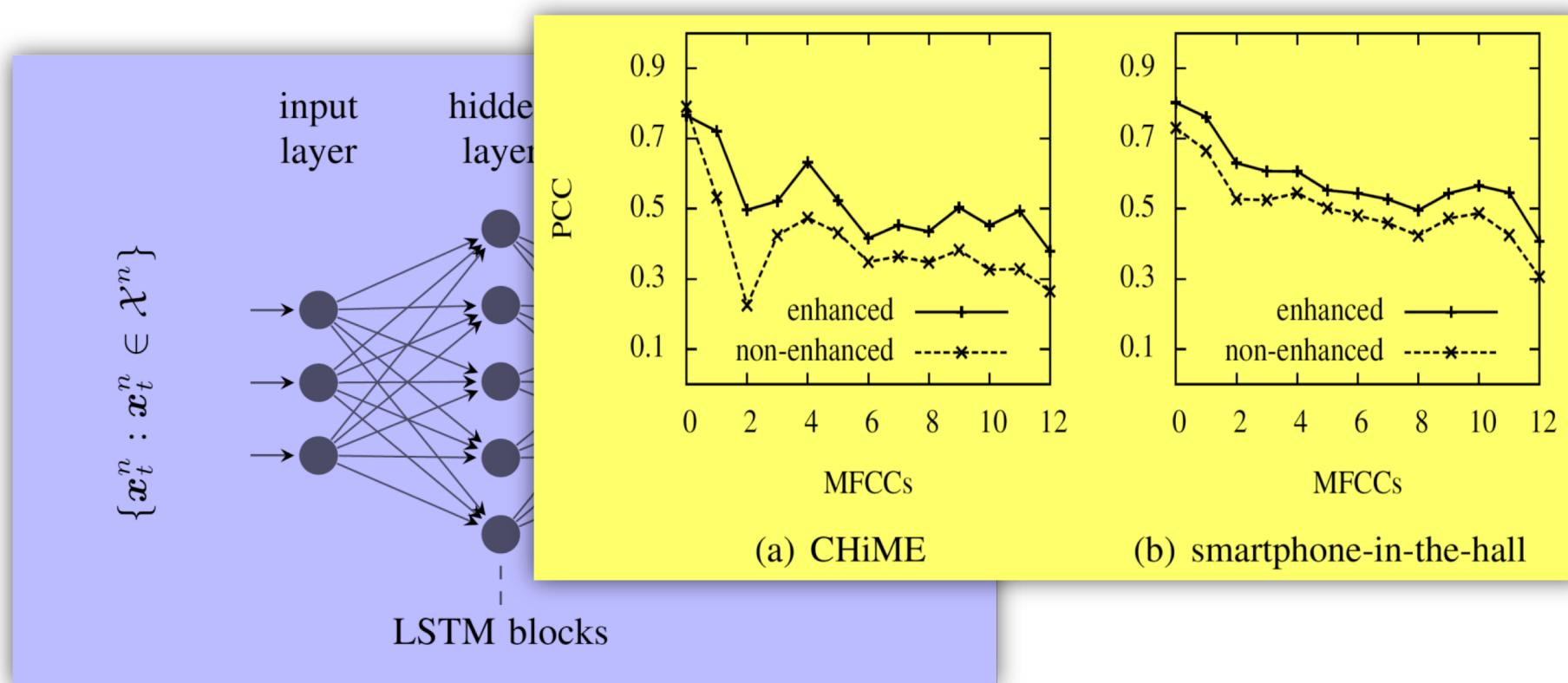
Recurrent layers

Output label at 25Hz

- Feature Enhancement
  - Recurrent Denoising Autoencoder

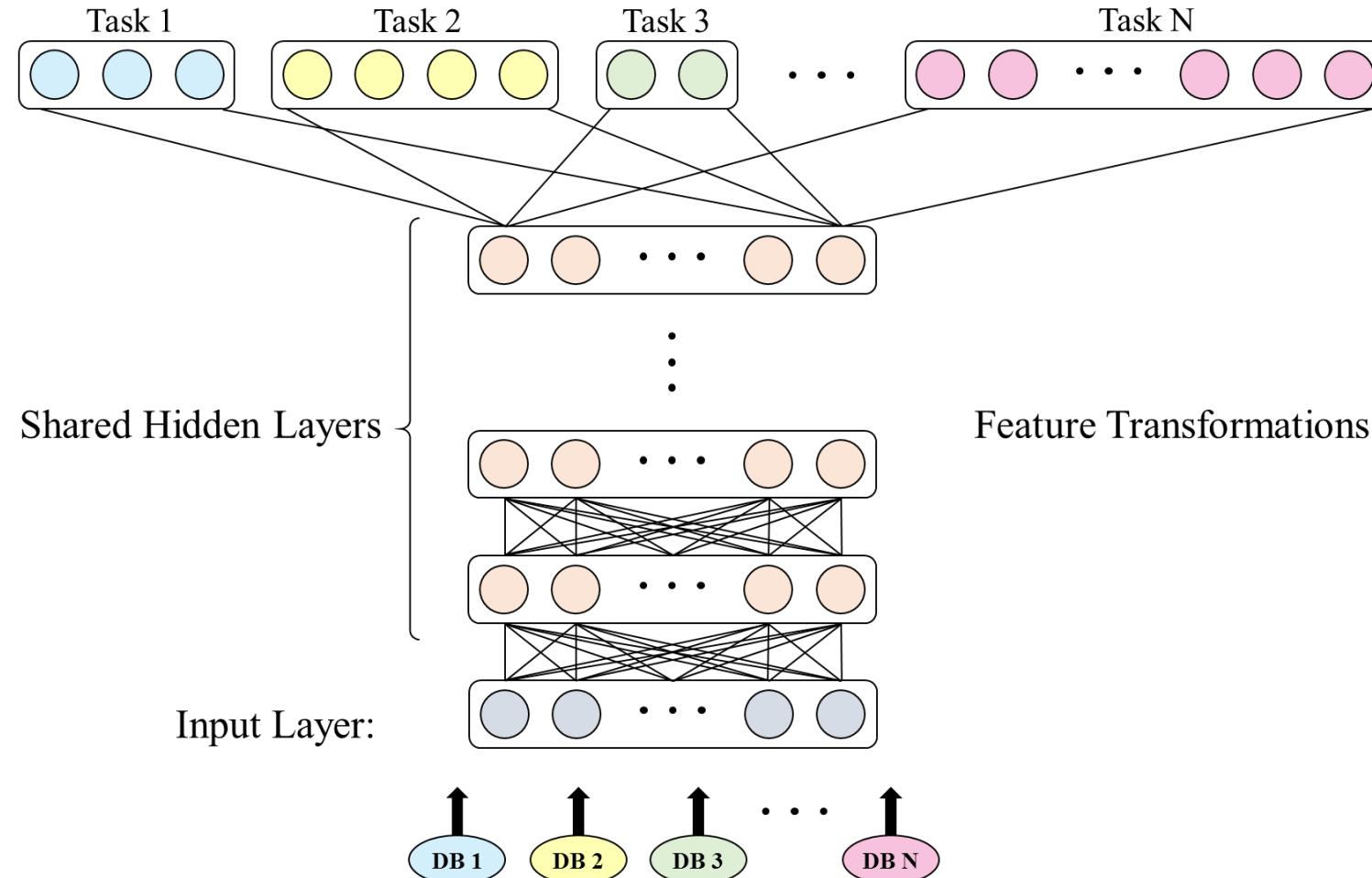


- Autoencoding



# Multi-Task Learning.

Björn W. Schuller

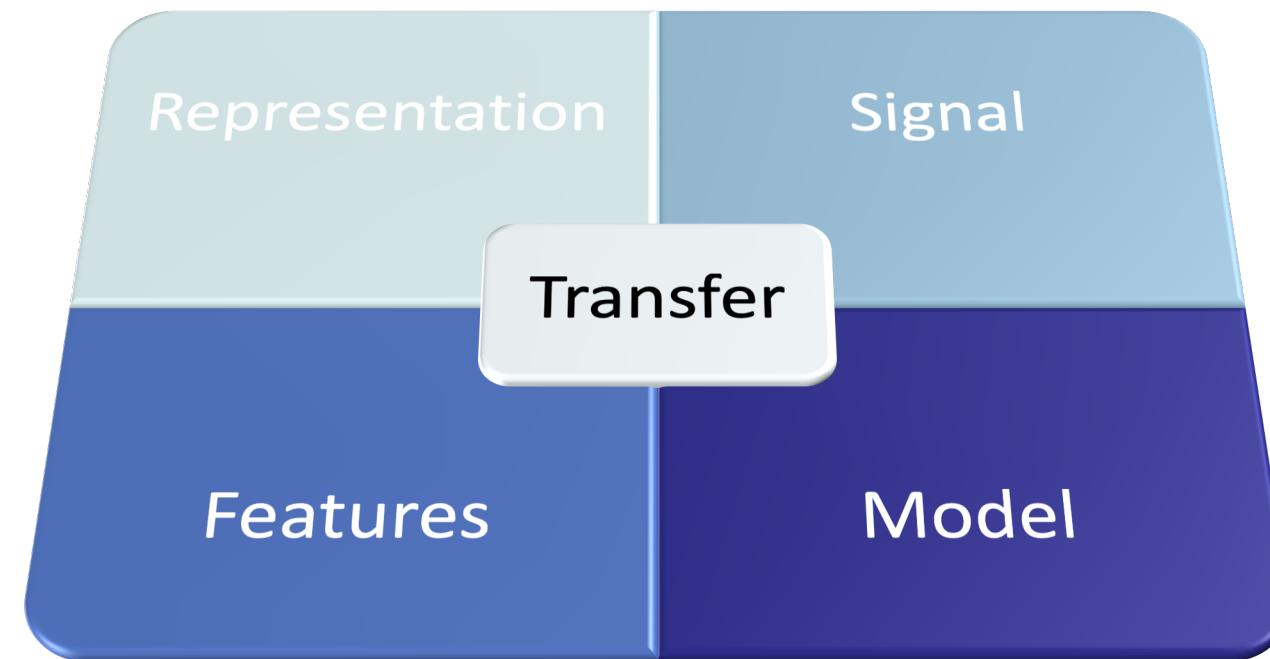


"Multi-task Deep Neural Network with Shared Hidden Layers: Breaking down the Wall between Emotion Representations",  
ICASSP, 2017.

# Transfer Learning.

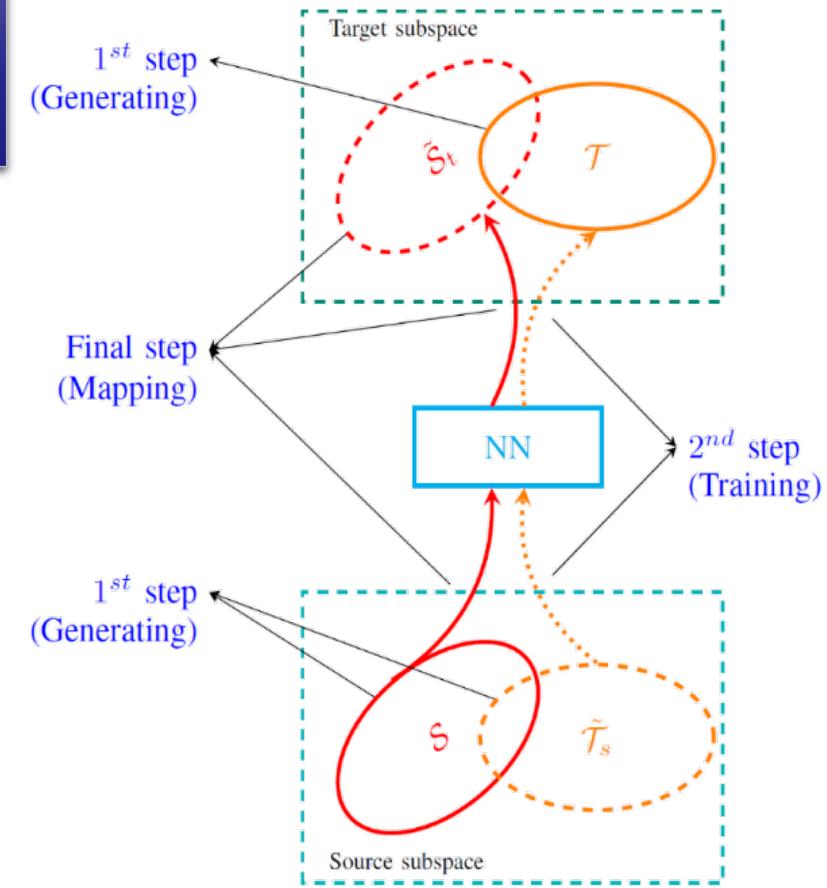
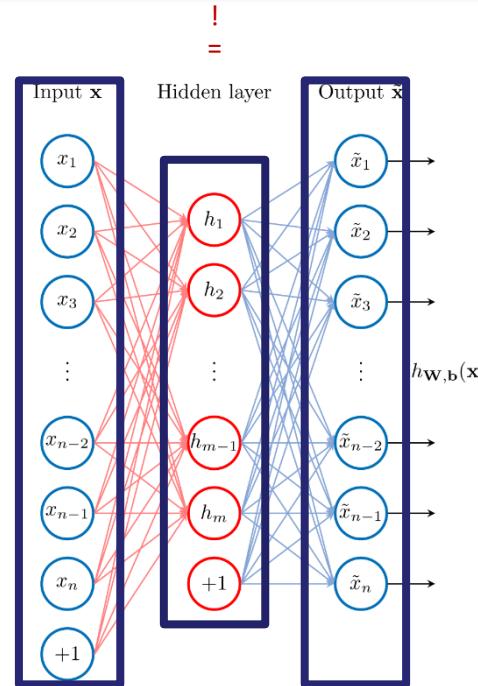


- >2 Decades
- Reuse Data!
- Human: Great @it!  
Know French → Understand some Spanish, Portuguese, etc.



## Shared Hidden Layers?

% UA	Target w/o	DAE	DAE-NN
EC	60.4	56.3	59.2
	!	=	



*"Autoencoder-based Unsupervised Domain Adaptation for Speech Emotion Recognition",  
IEEE Signal Processing Letters, 2014.*

## Audio = Audio ?

- Cross-Domain & Transfer Learning

CCC Speech

Arousal      Valence

S2S

.749

.332

M2S

.545

.164

M2S-TL

.567

.181

CCC Music

Arousal

Valence

M2M

.317

.090

S2M

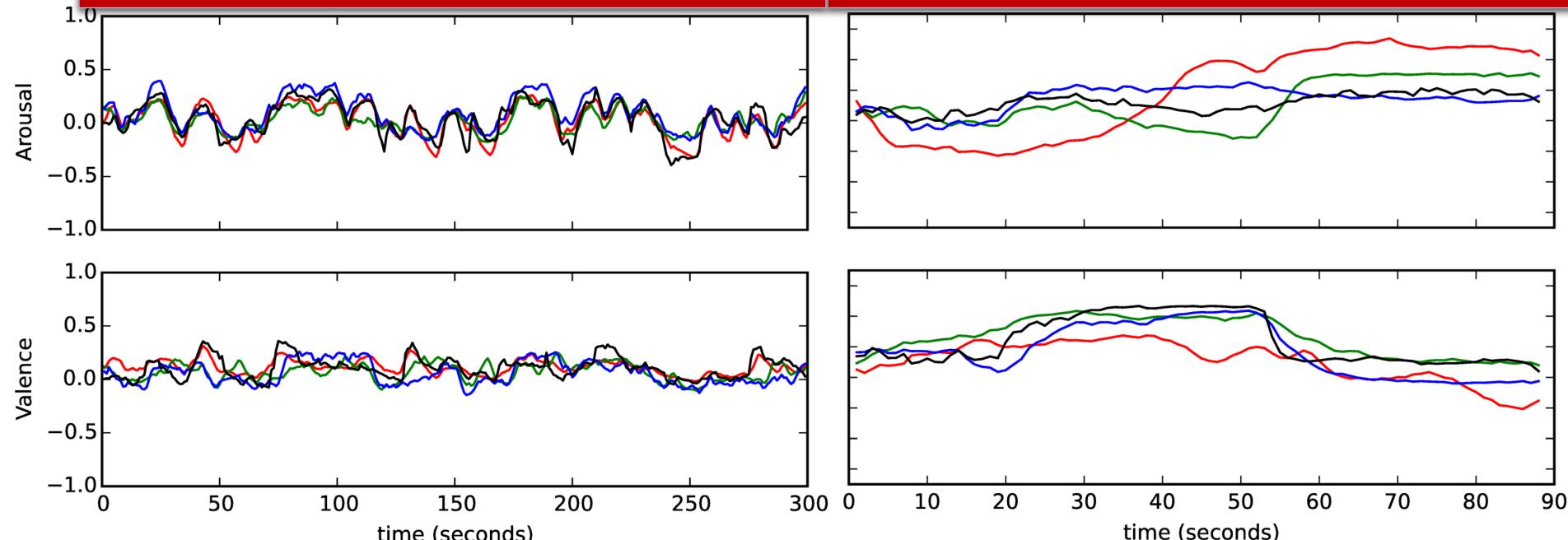
.276

.133

S2M-TL

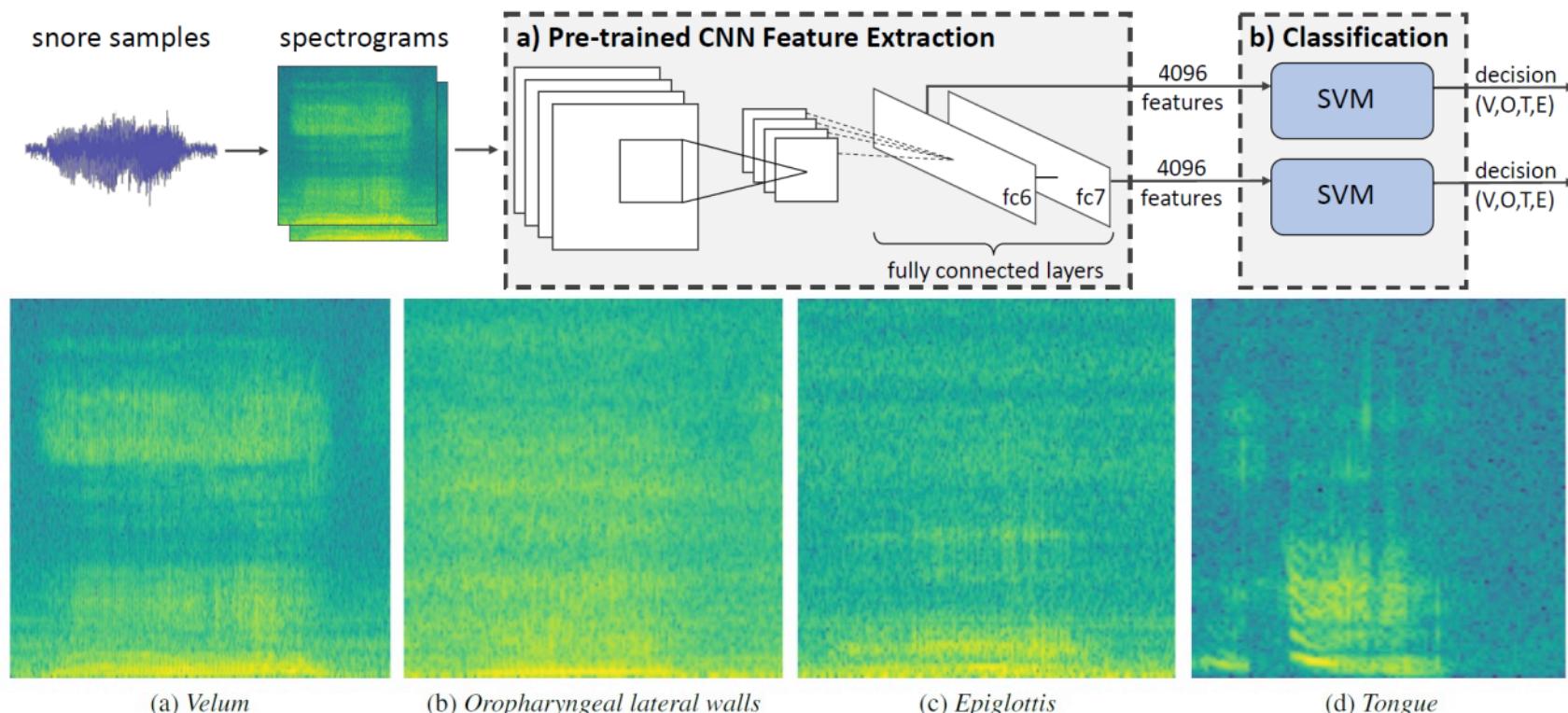
.269

.117



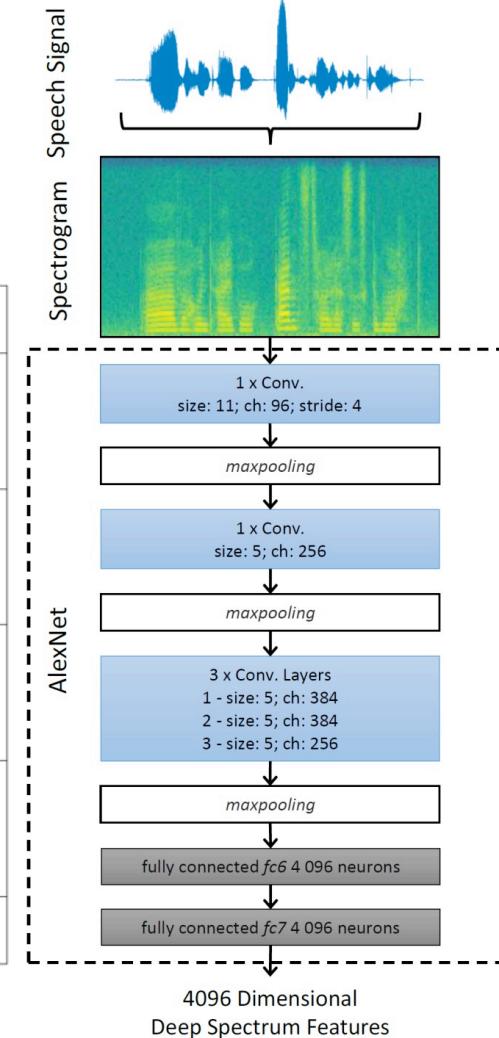
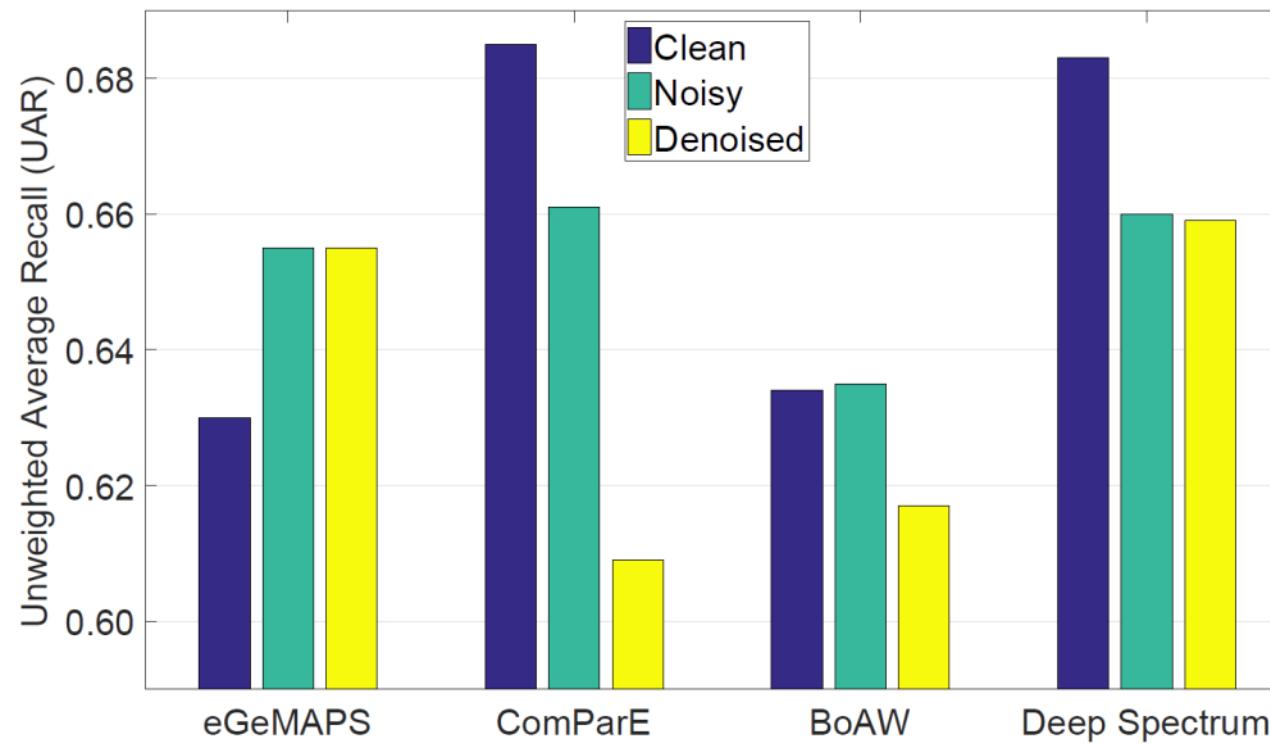
## Audio = Images?

	%UA
CNN+LSTM	40.3
Functionals	58.8
CNN+GRU	63.8
Deep Spec	67.0

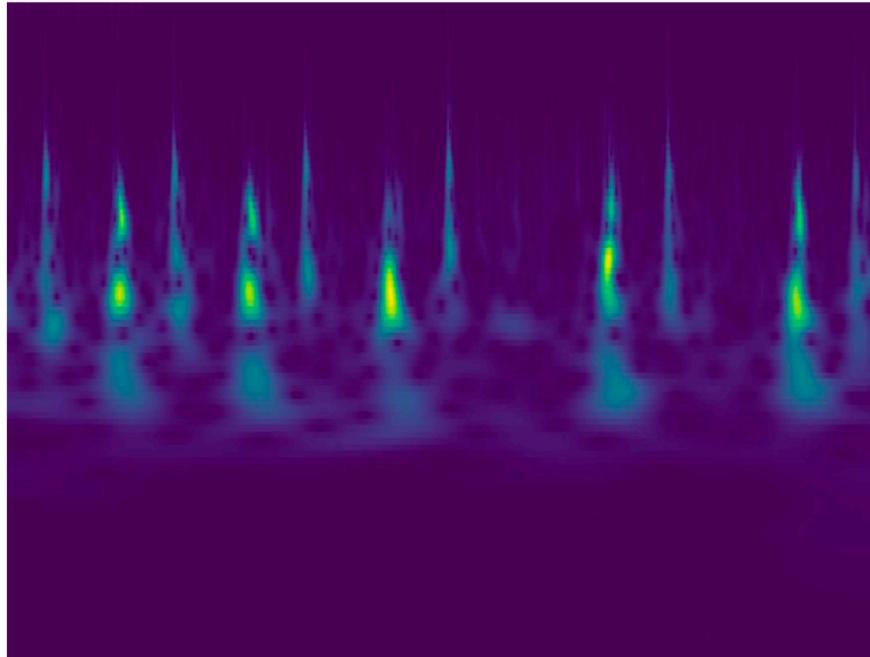


## Speech = Images?

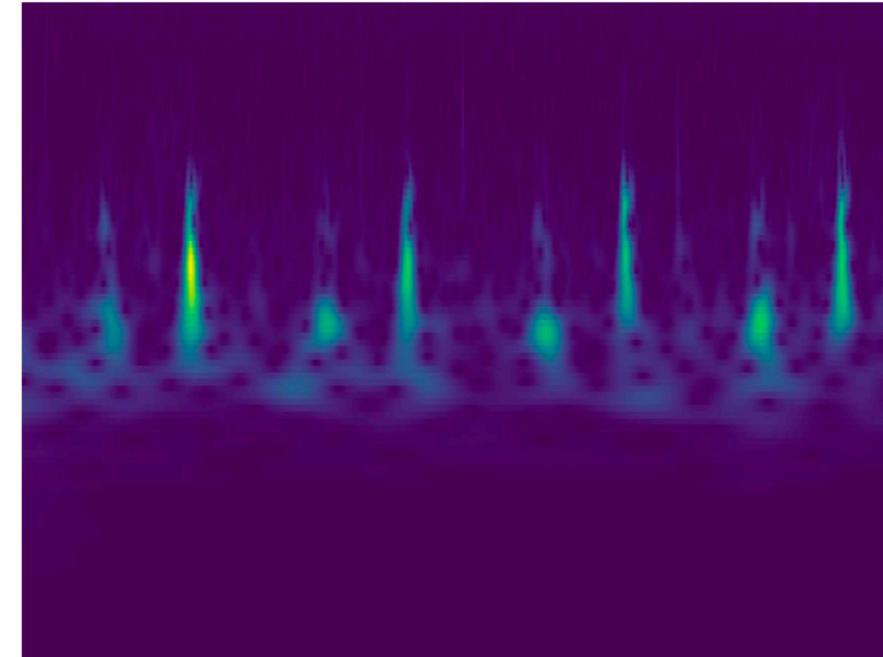
- Emotion with Image Nets
  - IS Emotion Challenge task – 2 classes



## Heart Beats = Images?

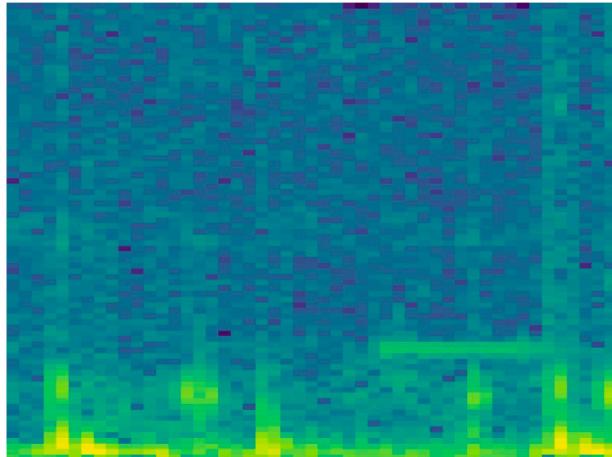


(a) Normal (*a0007.wav*)

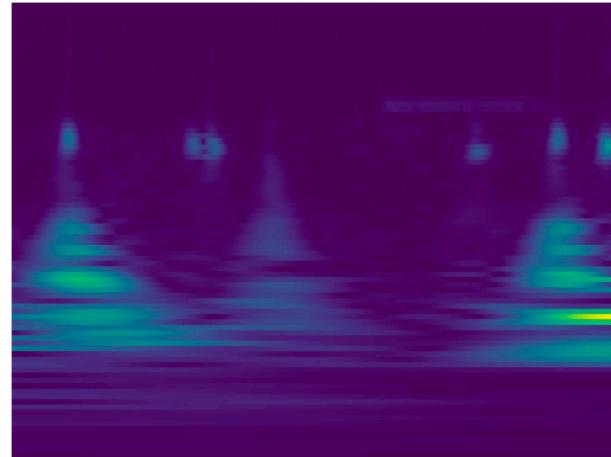


(b) Abnormal (*a0001.wav*)

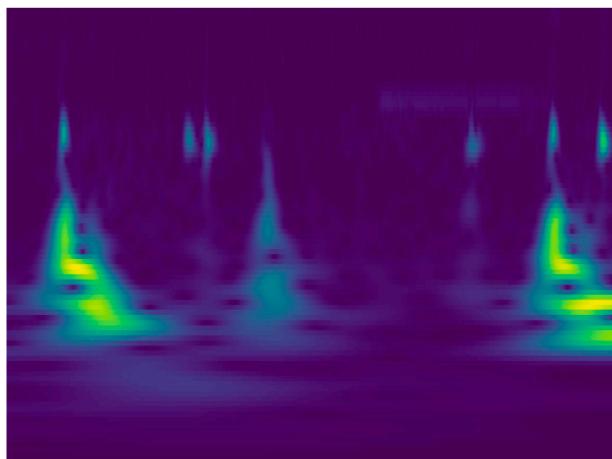
## Heart Beats = Images?



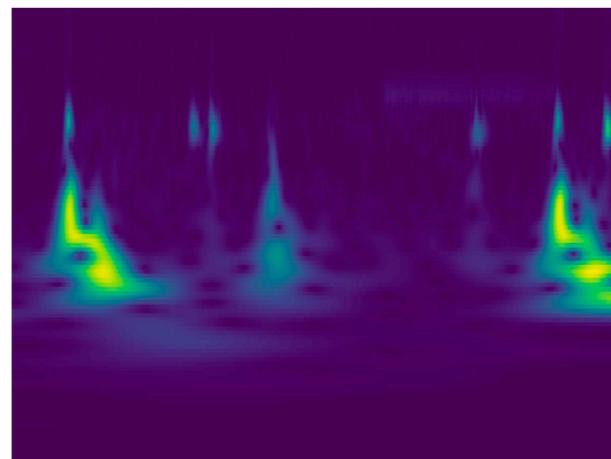
(a) *fft*



(b) *bump*



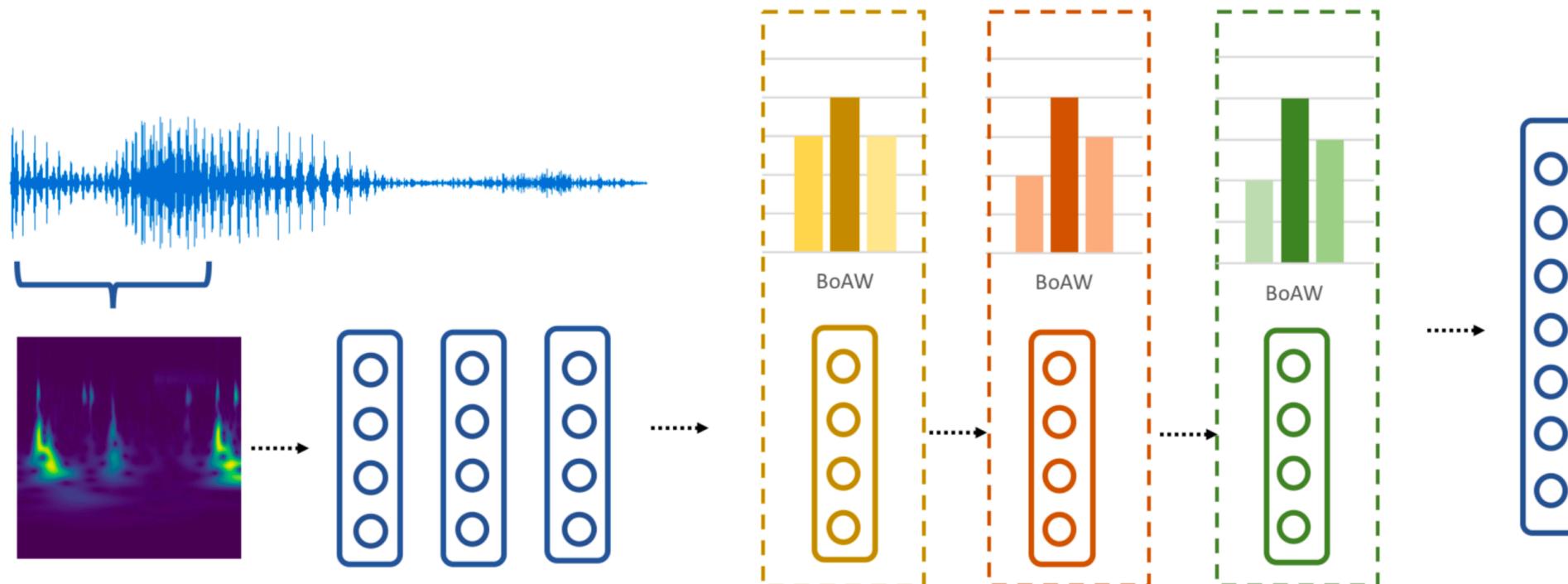
(c) *morse*



(d) *amor*

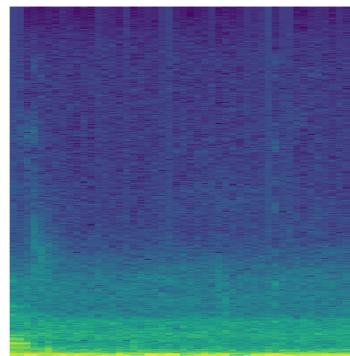
## Heart Beats = Images?

3-way Heart Beat	%UA
Baseline	46.9
Learnt VGG + SVM	56.2

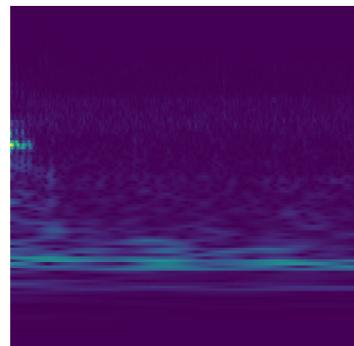


Audio = Images?

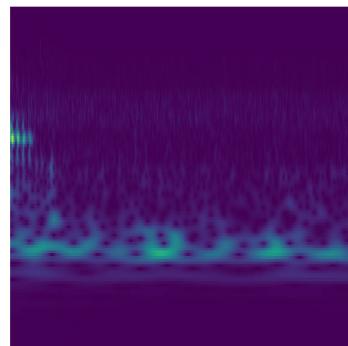
- Wavelets vs STFT via VGG16



(a) STFT



(b) bump wavelet



(c) morse wavelet

---

Input:  $224 \times 224$  RGB image

---

$2 \times$ conv size: 3; ch: 64  
Maxpooling

---

$2 \times$ conv size: 3; ch: 128  
Maxpooling

---

$3 \times$ conv size: 3; ch: 256  
Maxpooling

---

$3 \times$ conv size: 3; ch: 512  
Maxpooling

---

$3 \times$ conv size: 3; ch: 512  
Maxpooling

---

Fully connected layer *fc6* with 4096 neurons  
Fully connected layer *fc7* with 4096 neurons  
Fully connected layer with 1000 neurons

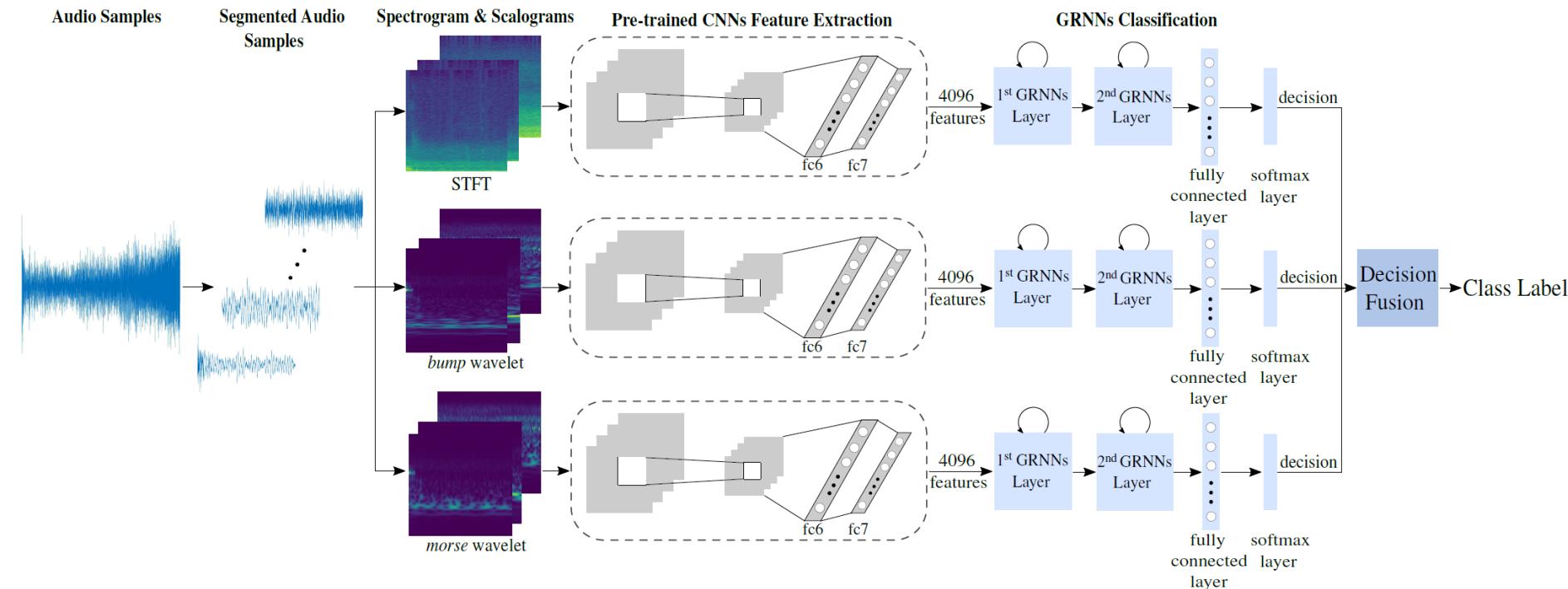
---

Output: softmax layer of probabilities for 1000 classes

---

## Audio = Images? • Wavelets vs STFT via VGG16

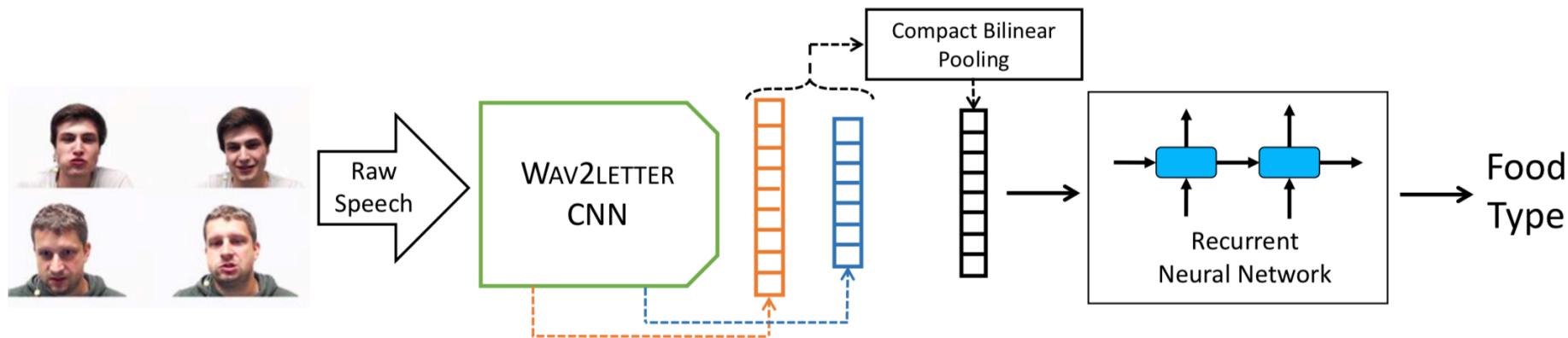
	%WA
DCASE 2017	
STFT	76.5
STFT+bump	79.8
STFT+morse	76.9
All	80.9



*"Deep Sequential Image Features for Acoustic Scene Classification", DCASE, 2018.*

## Speech = Speech ...

- **Wav2Letter (pre-trained on 1000h speech)**



EAT Food-type – UAR [%]	LOSO-CV	Test
Baseline End-to-End CNN-LSTM	-	32.8
Baseline BoAW & SVM	64.3	67.2
2-layer NN + ReLU + momentum [5]	68.6	68.4
Pre-trained CNN & LSTM [6]	67.2	75.9
Proposed method	76.4	TBA

# Reinforcement.



- **Major types of machine learning algorithms**

- Supervised learning
  - Learning on: Labeled data
- Unsupervised learning
  - Learning on: Unlabeled data
- Reinforcement learning
  - Learning on: Reward-based interaction with environment

## ● Case: Interactive Agent

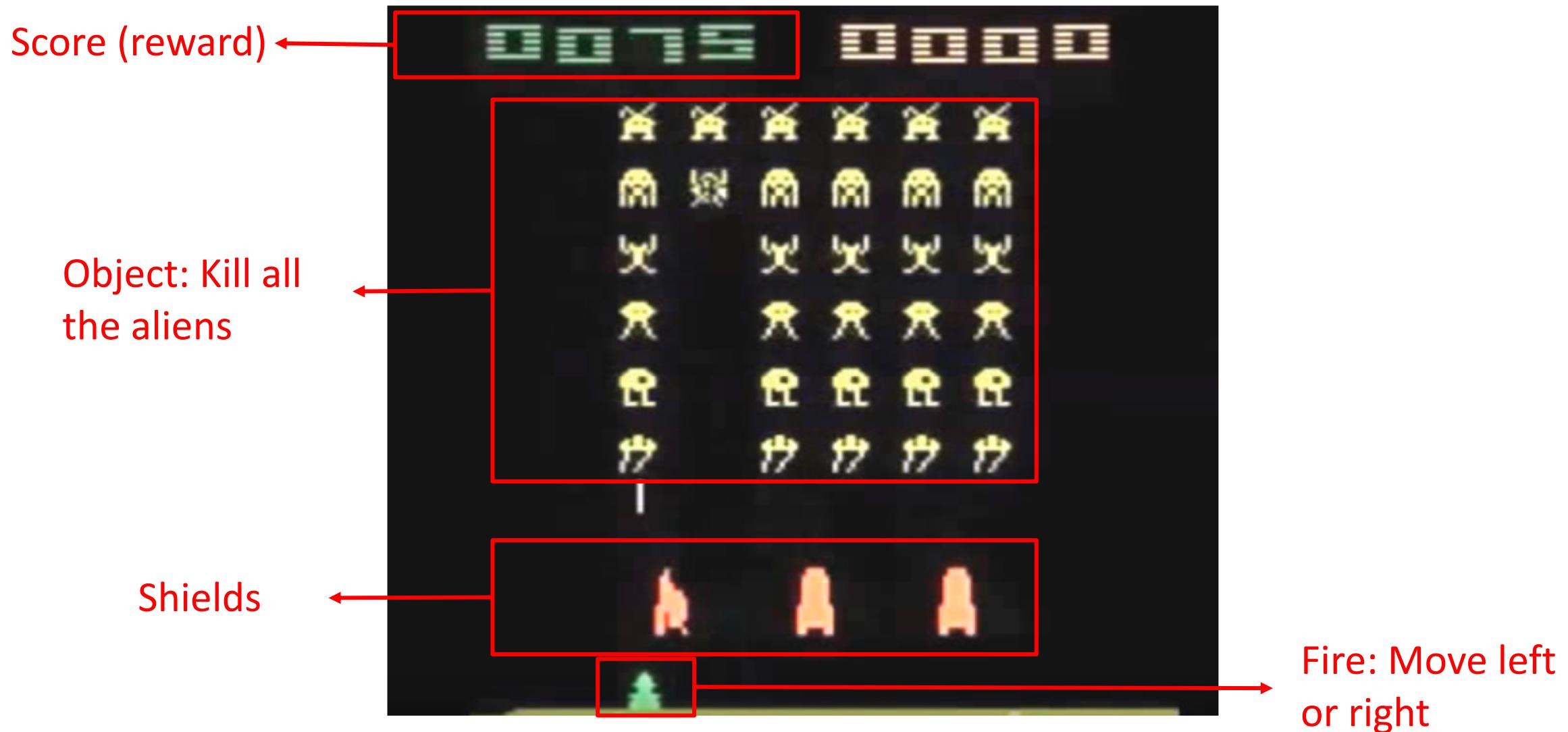
- Input: Environment
  - Described by feature
- Output: Action
  - Given set of possible actions
- Predefined Goal
  - E.g. cross the street (robot), chat-bot

## ● Approaches

- Reinforcement Learning
  - Agent exploring environment
  - Agent obtaining reward for every action
    - Reward based on environmental change to action
  - Agent learning rule system from reward

# Applications of Reinforcement Learning

Björn W. Schuller



- **State**

- Description of environment and agent
- Often hand-crafted features
- Current state  $s_i$  extracted from current situation
- Knowledge of possible state  $s_i \in S$
- Examples
  - Coordinates of Super Mario, distance to closest obstacle  
 $s_i = (x_M, y_M, d_O)$

## ● Action

- Actions available to agent
- Knowledge of available actions  $a_i \in A$
- Examples
  - Jump, Move right, Move left
$$A = \{J, R, L\}$$
- Leading to new situation  $s_{i+1}$

## ● Policy

- Probability of choosing action  $a_i$  in situation  $s_i$   
 $\pi(s_i, a_i) \rightarrow [0, 1]$
- Learning target during training
- Applied at each timestep

## ● Reward

- User rewarded with numerical value  $r_i \in \mathbb{R}$
- Can depend on situation after action  $s_{i+1}$
- Applied at each timestep
- Basis for learning goal

## ● Problem

- State-transitions and rewards difficult to predict
- Non-deterministic environment
  - Same action at different points in time
    - Different reward
    - Different resulting state
- E.g. Football player shooting on goal
  - Hit: High reward
  - Miss: Low reward

## ● **Exploitation vs. Exploration**

- **Exploitation**
  - Choosing most promising action
  - Guaranteed high reward (if well explored)
  - No new information
- **Exploration**
  - Choose less explored action
  - Possibly low reward
  - Gain of information

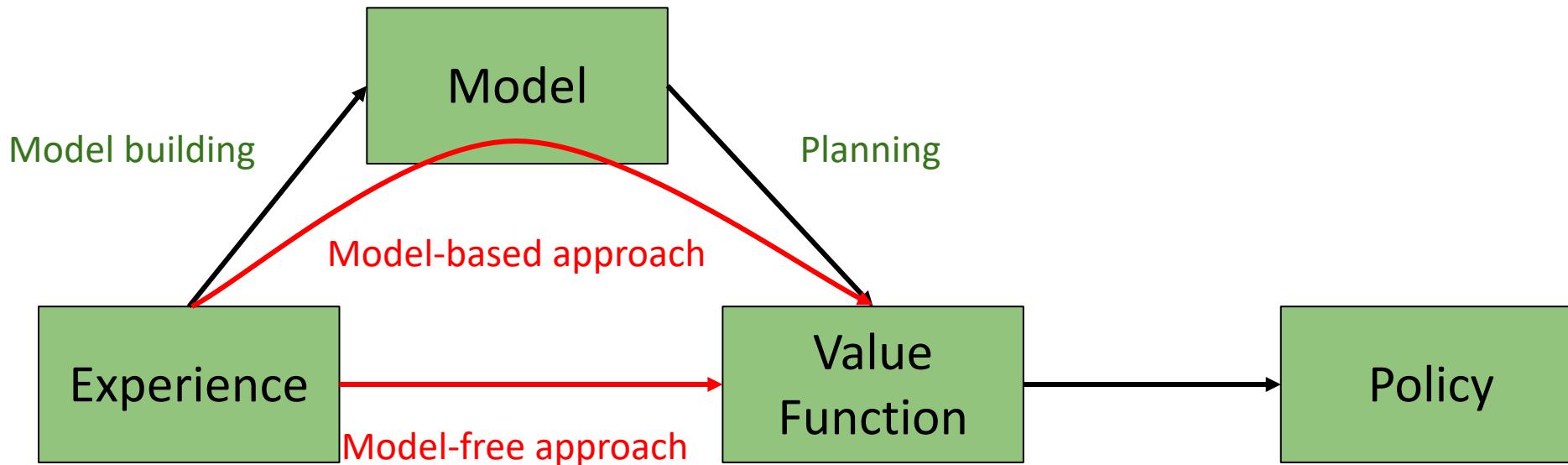
## ● Goal of RL

- Maximizing expected future reward (value)  $V_\pi$ 
  - Expected reward depending on policy  $\pi(s_i, a_i) \rightarrow [0, 1]$   
→ Learn policy to maximize reward
- Problem
  - Infinite runtime of algorithm  
→ Infinite sum of rewards
  - Define expected future reward  $V_\pi$

- **Expected future reward (value) functions**

- Finite-horizon model  $V_\pi = E(\sum_{t=0}^h r_t)$ 
  - Finite horizon of steps
- **Infinite-horizon model**  $V_\pi = E(\sum_{t=0}^{\infty} \gamma^t r_t), 0 \leq \gamma \leq 1$ 
  - $\gamma$  : discount rate
- Average-reward model  $V_\pi = \lim_{h \rightarrow \infty} E\left(\frac{1}{h} \sum_{t=0}^h r_t\right)$

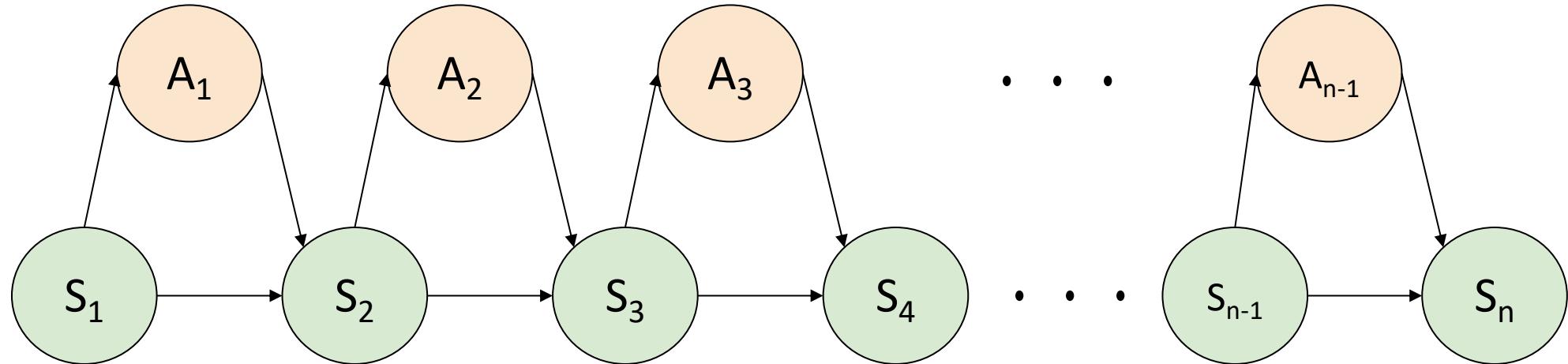
- Two basic reinforcement learning approaches



## Markov Decision Process (MDP)

- Modelling the problem
- MDP defined by
  - Set of possible states  $S$
  - Set of possible actions  $A$
  - Reward function  $R(s_i, a_j) \rightarrow \mathbb{R}$ 
    - Expected reward when choosing action  $a_j$  in situation  $s_i$
  - Transition function  $T(s_i, a_j, s_l) \rightarrow [0, 1]$  Probability of transitioning from state  $s_i$  to state  $s_l$  when choosing action

- **Markov Decision Process (MDP)**



## ● Optimal Value Function

- Assuming MDP model known
- Optimal reward gained from state
  - I.e. using optimal policy
- Infinite-horizon case  $V^*(s) = \max_{\pi} E(\sum_{t=0}^{\infty} \gamma^t r_t)$
- As Bellman equation
  - Optimal solution consisting of optimal solutions for subproblems  $V^*(s) = \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s'))$

## ● Value iteration

- Learn optimal value iteratively

*initialize  $V(s)$  arbitrarily*

*loop until policy good enough*

*loop for  $s \in S$*

*loop for  $a \in A$*

$$Q(s, a) := R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V(s')$$

$$V(s) := \max_a Q(s, a)$$

*end loop*

*end loop*

- Derive optimal policy from optimal values

## ● Problem

- MDP model not known
- Model-based algorithms
  - Learn MDP model → Derive optimal policy
- Model-free algorithms
  - Learn policy without learning model

- **Model-free approach**

- Learn policy rather than MDP model
- Algorithms
  - **Q-learning**
  - Adaptive heuristic critic
  - Model-free learning with average reward
- Model-free vs Model-based
  - Superiority of either heavily discussed

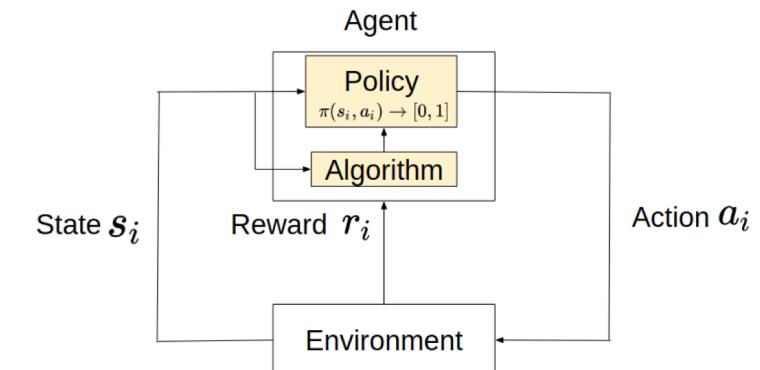
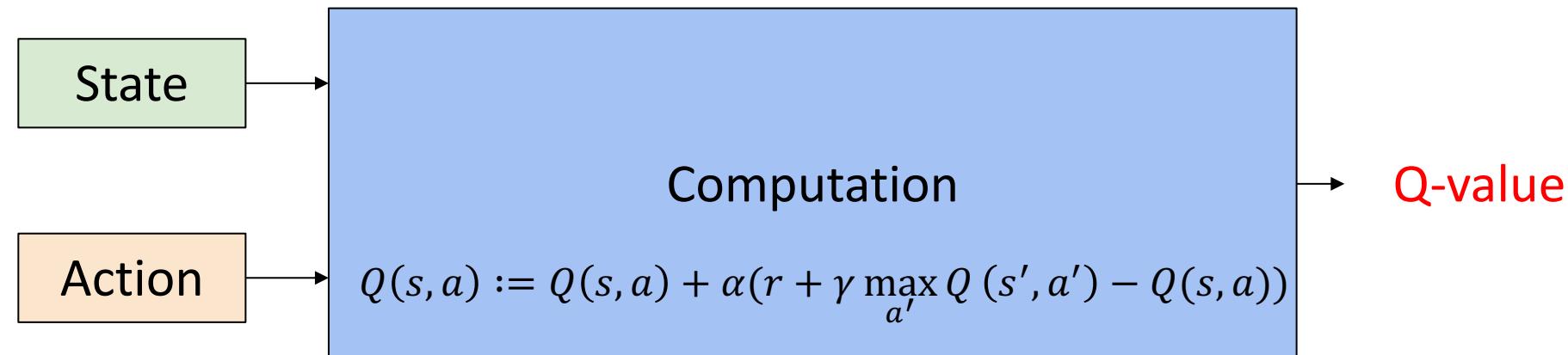
- **Q-learning**

- Popular Algorithm for RL
- Given situation  $s$
- Expected reward  $Q^*(s, a)$ 
  - choosing action  $a$
  - subsequently choosing optimal action
- Optimal Value  $V^*(s) = \max_a Q^*(s, a)$   
 $\rightarrow Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'} Q^*(s', a')$

## ● Q-learning

- Learning  $Q$ -Values
  - Performing action  $a$  in situation  $s$
  - Obtaining reward  $r$  transitioning into state  $s'$   
 $\rightarrow Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
- Exploration: Choose actions (partially randomly)
- Exploitation: Choose action with optimal  $Q$ -Value

## ● Q-learning



## • Q-learning

- The computation part is usually a table – Q-table



## training

## ● Reinforcement Learning

- So far
  - Provided set of situations
  - Provided set of actions
  - Provided reward function
  - Determine optimal policy function
- Problem
  - Where does the set of situations come from?

## ● Situation Sets

- Situation Variables often handcrafted
  - Subjective selection of features
  - Only applicable to one task
- Situation Variables based on sensory input
  - Pre-processing necessary (high dimensionality)
  - Applicable to multiple tasks
  - E.g. computer game: pixels of screen
  - Often used in Deep Reinforcement Learning (DRL)

- General Approach

- Try to learn  $Q$ -Values

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'} Q^*(s', a')$$

- So far linear updates to  $Q$ -Values
  - Guaranteed Convergence
  - No Generalisation between actions and situations

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

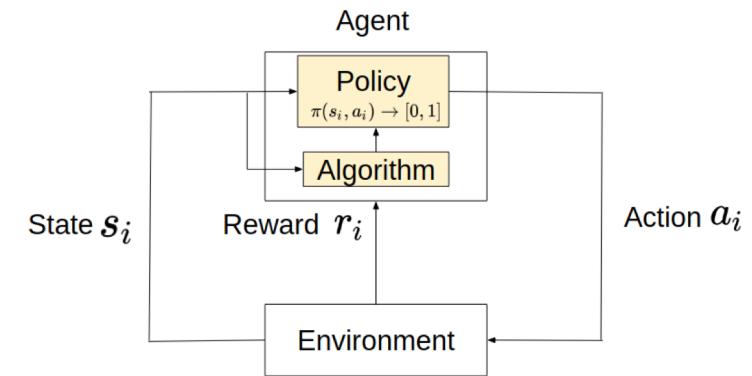
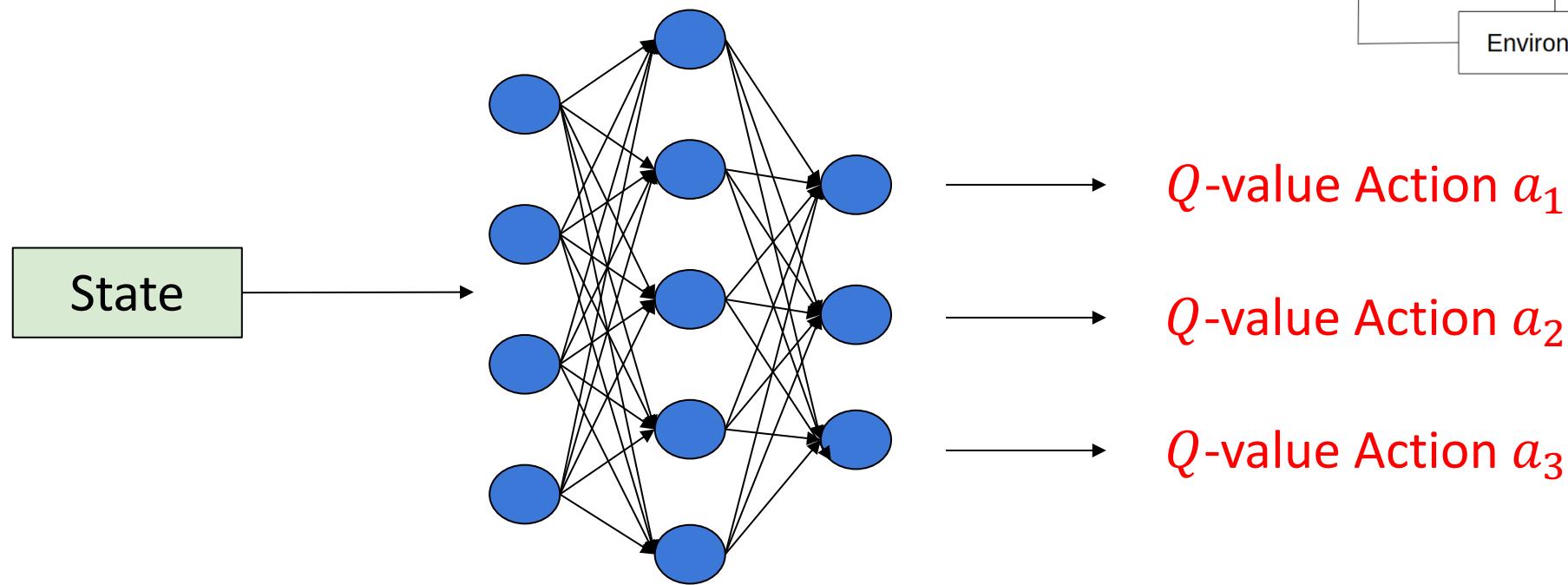
## ● Approximator Approach

- Estimate action Value  $Q(s, a, \theta) \approx Q^*(s, a)$ 
  - Estimation often linear in  $\theta$
- Deep  $Q$ -Network
  - Using DNN for estimation of  $Q^*$
  - Non-linear functions in  $\theta$
  - Convergence not guaranteed
  - Good generalisation

## ● Q-Network

- Prediction of  $Q$ -values
  - Depends on parameters  $\theta_i$  of Neural Net at iteration  $i$
- Training of  $Q$ -Values
  - Given situation  $s$
  - Calculate  $Q$ -Value  $Q(s, a, \theta_i)$  for actions  $a \in A$
  - Perform action  $a$  e.g. with highest  $Q$ -Value (greedy)
  - Get reward  $r$  and get to situation  $s'$

## ● Q-Network



## ● Q-Network

- Training of  $Q$ -Values
  - Compare  $Q$ -Value  $Q(s, a, \theta)$  with  $Q$ -Values for  $s'$
  - Approximating target value

$$y = r + \gamma \max_{a'} Q(s', a', \theta_i^-)$$

- $\theta_i^-$ : DNN parameters of prior iteration (e.g.  $i - 1$ )

## ● Q-Network

- Training of  $Q$ -Values
  - Define loss function
    - For minibatch of tuples  $(s, a, r, s')$
    - Mean squared error

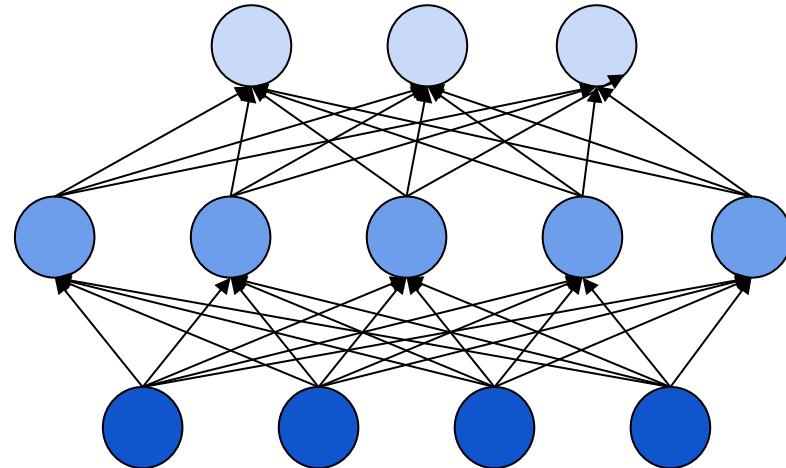
$$L_i(\theta_i) = E_{s,a,r}[(E_{s'}[y] - Q(s, a, \theta_i))^2]$$

- Calculate gradient  $\nabla_{\theta_i} L(\theta_i)$  for update steps

## ● Q-Network

- Summary
    - Deep Reinforcement Learning approach
    - Model-free
    - Learning  $Q$ -Values with DNNs
    - Labels based on previous data
- No supervised learning

- Evolving Learning



Brute-force search

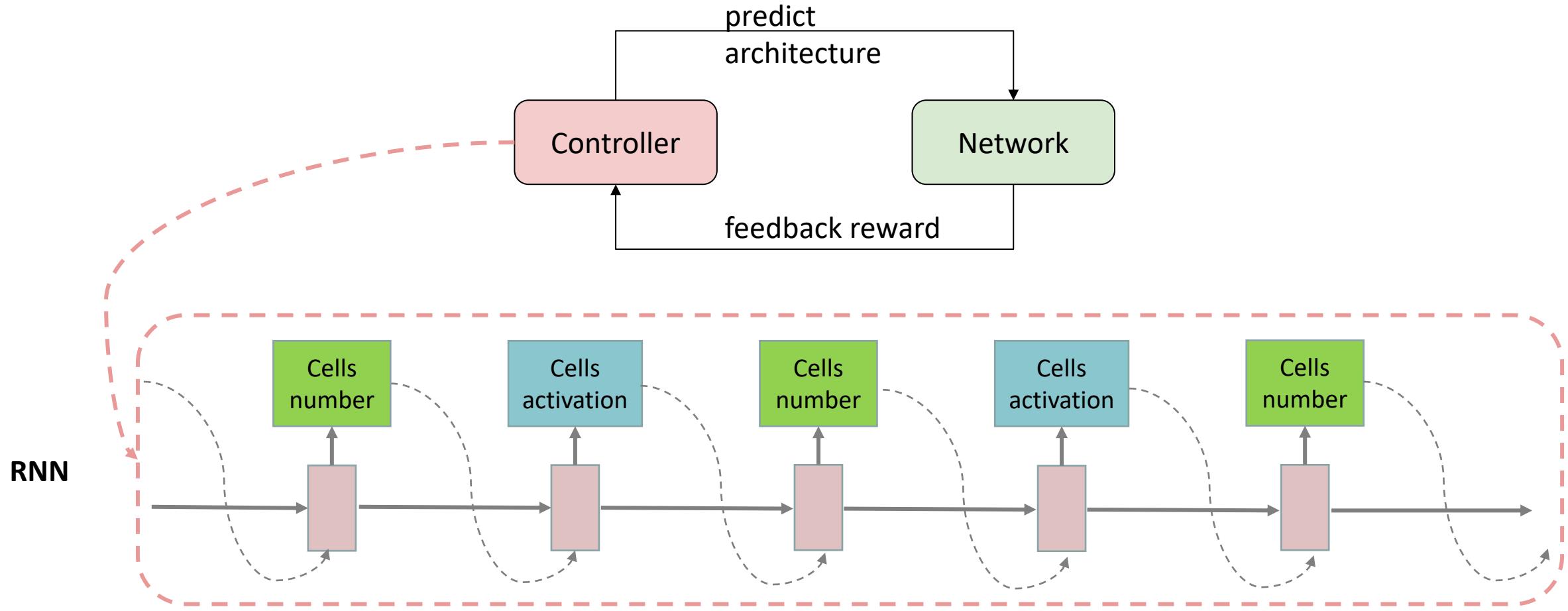
- require high experiences
- time-consuming

Is it possible to use reinforcement learning ?

- Evolving learning

How to select deep learning architectures,  
and hyper-parameters ?

## ● Evolving Learning



- Evolving Algorithms

- Example: RNN for classification problem
  - Goal: Learn hyperparameters for each layer  $i$ 
    - Number of hidden nodes  $h_i$
    - Activation function  $\varphi_i$
  - Given set of hyper-parameters  $\tau = \{h_1, \varphi_1, h_2, \varphi_2, \dots\}$ 
    - Create child-network specified by hyperparameters
    - Train child-network on classification task
    - Obtain reward: Accuracy on development set  $R$

## ● Evolving Algorithms

- Example: RNN for classification problem
  - Probability for child network  $\tau$ :  $p(\tau|\theta)$ 
    - Depends on Parameter of Controller Network  $\theta$
  - For optimal architecture of Controller Network
    - Maximize expected reward

$$J(\theta) = \sum_{\tau} R(\tau)p(\tau|\theta)$$

- Update  $\theta$  with gradient  $\nabla_{\theta} J(\theta)$

## ● Evolving Learning

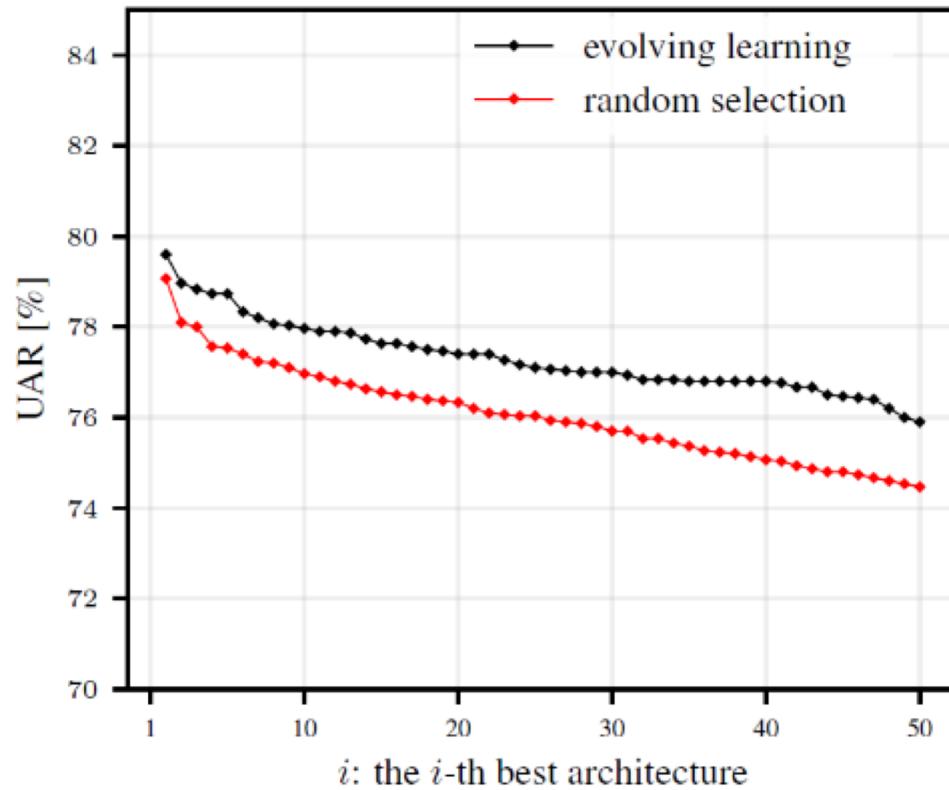
$$\text{reward} \quad J(\theta) = \sum_{\tau} R(\tau) p(\tau | \theta)$$

prediction accuracy controller parameters  
probability network architecture  
network architecture

Search space of hyper-parameters

Types	Hyper-parameters
# layers	1, 2, 3, 4, 5
# node per layer	0, 40, 80, 120, 160, 200
Activation functions	Tanh, ReLU, sigmoid

- Evolving Learning vs other approaches



ComParE Crying Sub-Challenge	UAR[%]
Seq2seq [33]	62.1
End-to-end [34]	63.5
BoAW [35]	67.7
ComParE [28]	71.9
<b>Evolved GRU-RNN</b>	<b>70.1</b>

GANS .



Overview on existing kinds of network structures:

- *Fully connected feedforward networks*
- *Convolutional neural networks (CNNs)*
- *Recurrent neural networks (RNN)*

can take various kinds of inputs/outputs for training

speech, video, text, ...

## Discriminative Models:

$P(Y|X)$  is estimated

for a given sample  $x$ , predict a label  $y$



What I cannot **create**,  
I do not **understand**.

— Richard P. Feynman

## Generative Models:

$P(X)$  is estimated, either *explicitly* or *implicitly*

for given training data  $X$ , generate new samples similar to  $X$



and they can be applied to

- generate realistic meaningful samples (video/image/audio/text)
- for simulation and planning by generating time-series data
- enable statistical inference of latent representations
- expand low-resource dataset
- ... ...

**Problem:**

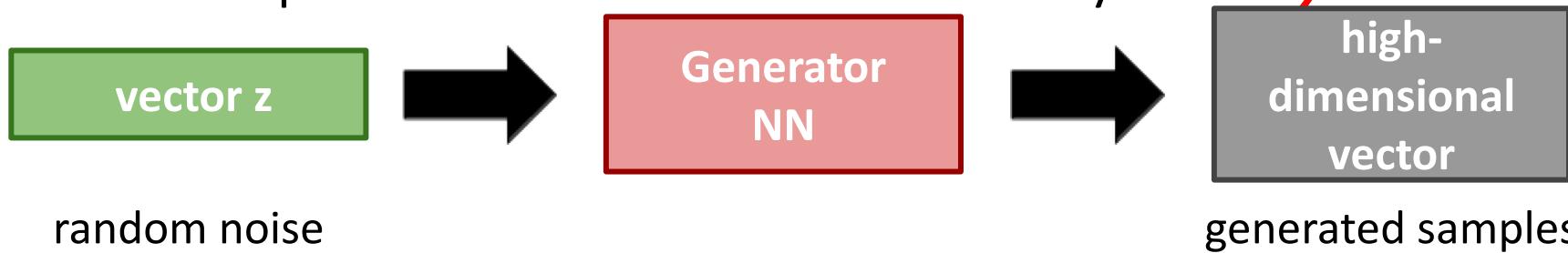
sample from complex, high-dimensional  $P(X)$

**Solution:**

sample from a simple distribution, e.g., random noise  $z$

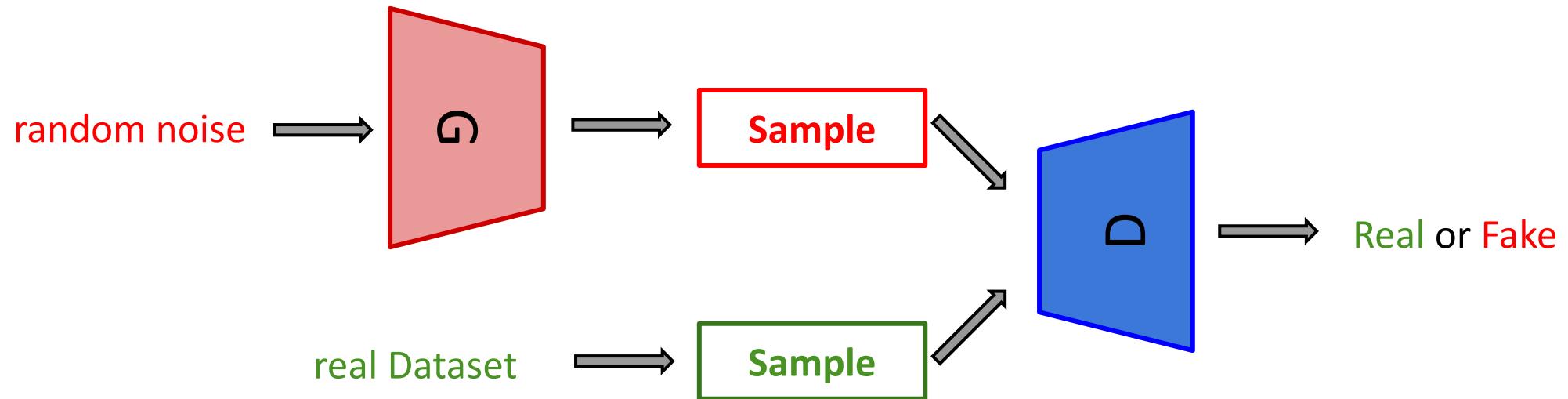
and learn transformation to  $P(X)$  with *a neural network*

& replace the distribution evaluation by *a binary test*



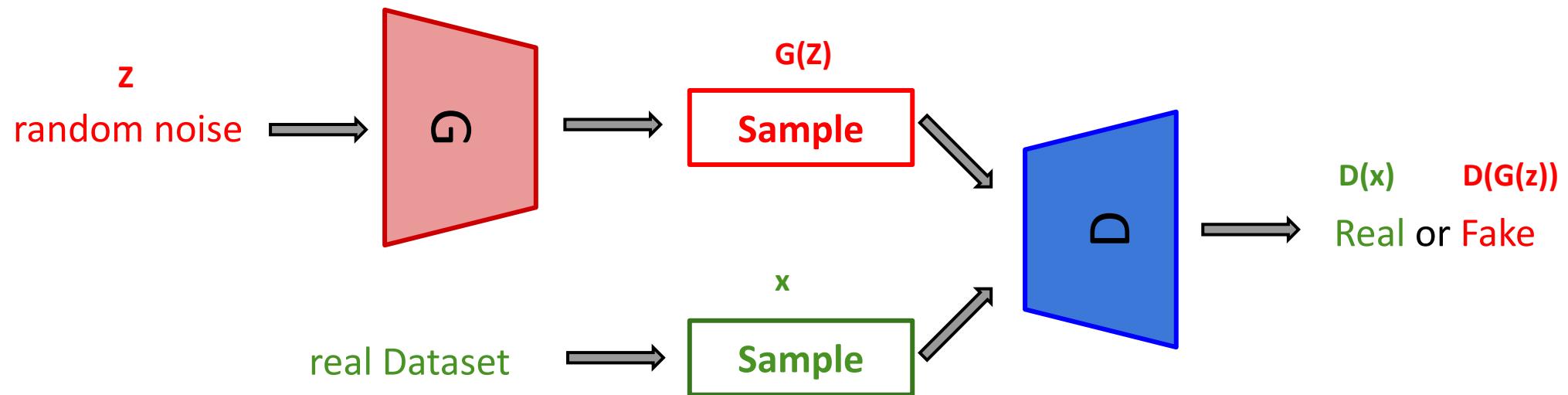
**G**enerator: aim to fool the discriminator by generating real-looking samples

**D**iscriminator: aim to distinguish between real and fake samples



**two-player game between **G** and **D****

## vanilla GAN Architecture

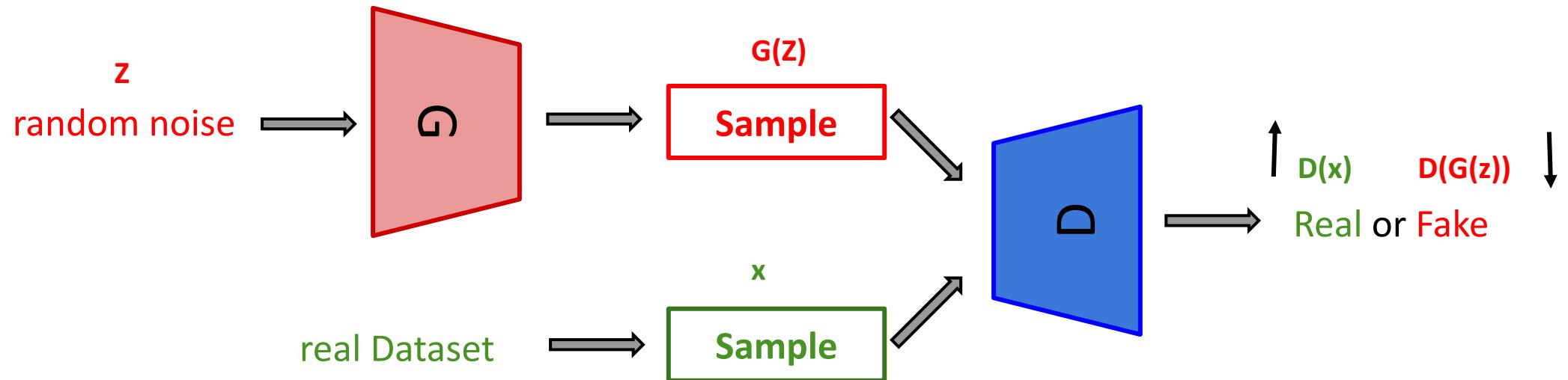


# GAN Architecture and Objective

Björn W. Schuller

**Discriminator:** distinguish between **real** and **fake** samples

→ increase  $D(x)$  to be close to 1 and decrease  $D(G(z))$  to 0

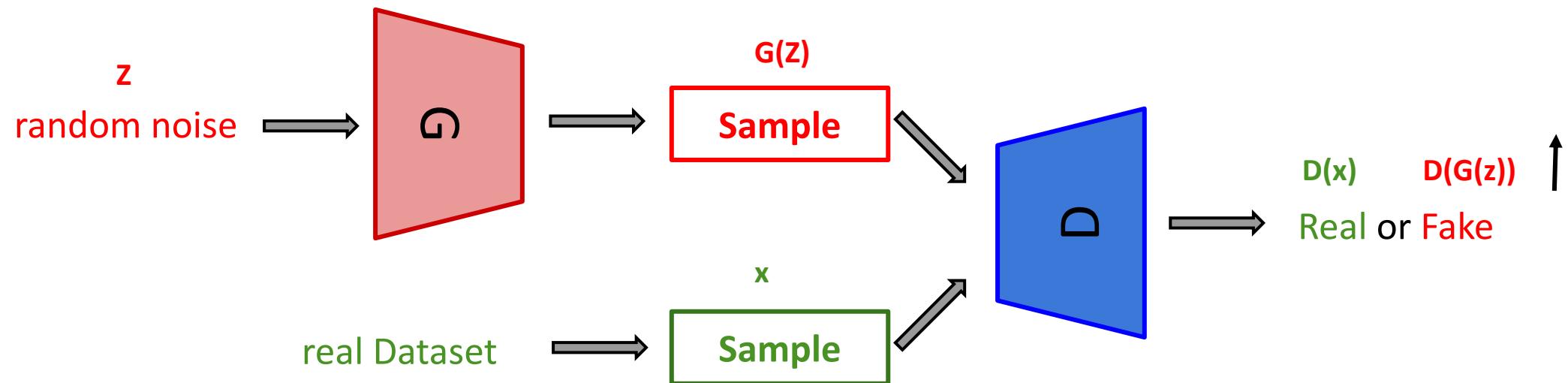


# GAN Architecture and Objective

Björn W. Schuller

**Generator:** fool the discriminator by generating real-looking samples

→ increase  $D(G(z))$  to 1 to fool the discriminator

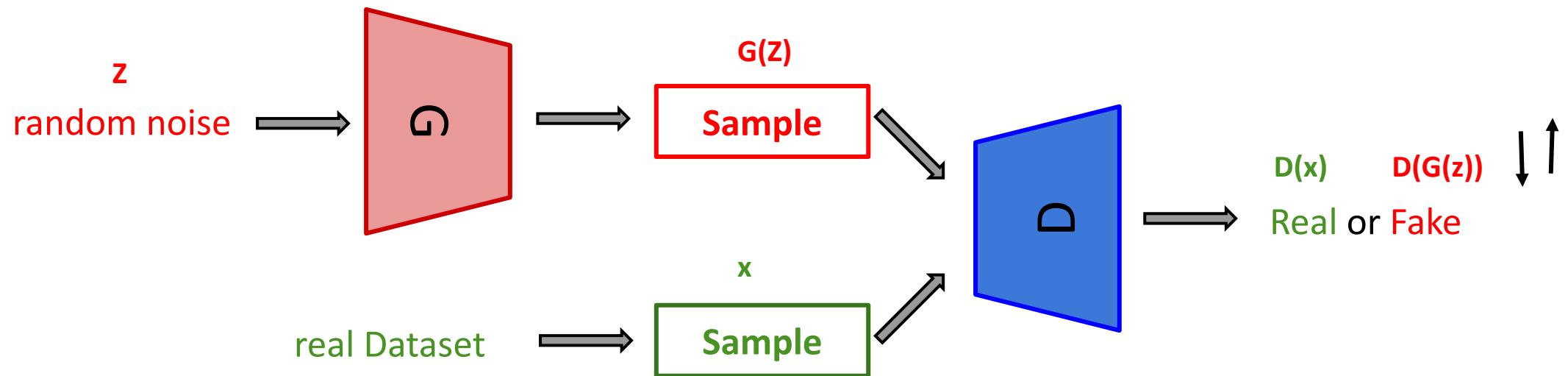


# GAN Architecture and Objective

Björn W. Schuller

**Discriminator:** increase  $D(x)$  to be close to **1** and decrease  $D(G(z))$  to **0**

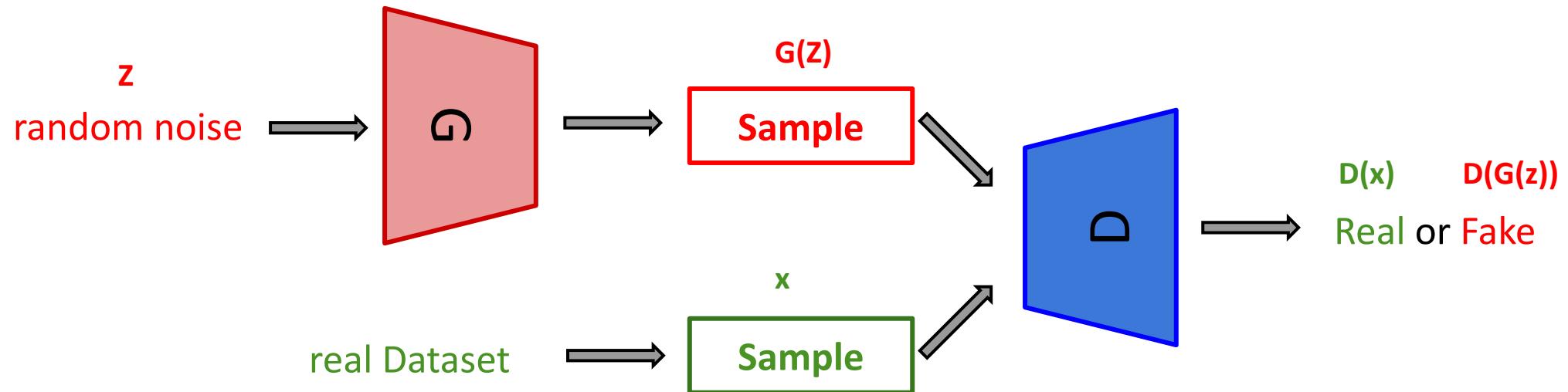
**Generator:** increase  $D(G(z))$  to **1** to fool the discriminator



**G and D trained in a minimax game !**

train G and D with **minimax** objective function:

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim p_z(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$



The Nash equilibrium of this minimax game is achieved at:

- $P_{\text{data}}(x) = P_{\text{gen}}(x) \forall x$
- $D(x) = \frac{1}{2} \forall x$

train G and D with **minimax** objective function:

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\underbrace{\log D_{\theta_d}(x)}_{\text{Discriminator output for real data } x}] + \mathbb{E}_{z \sim p_z(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)})]$$

- Discriminator wants to **maximise objective** such that  $D(x)$  is close to 1 (real) and  $D(G(z))$  is close to 0 (fake)
- Generator wants to **minimise objective** such that  $D(G(z))$  is close to 1, meaning that discriminator is fooled into thinking generated  $G(z)$  is real

train G and D with **minimax** objective function:

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\underbrace{\log D_{\theta_d}(x)}_{\text{Discriminator output for real data } x}] + \mathbb{E}_{z \sim p_z(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)})]$$

- **Gradient ascent** on Discriminator:

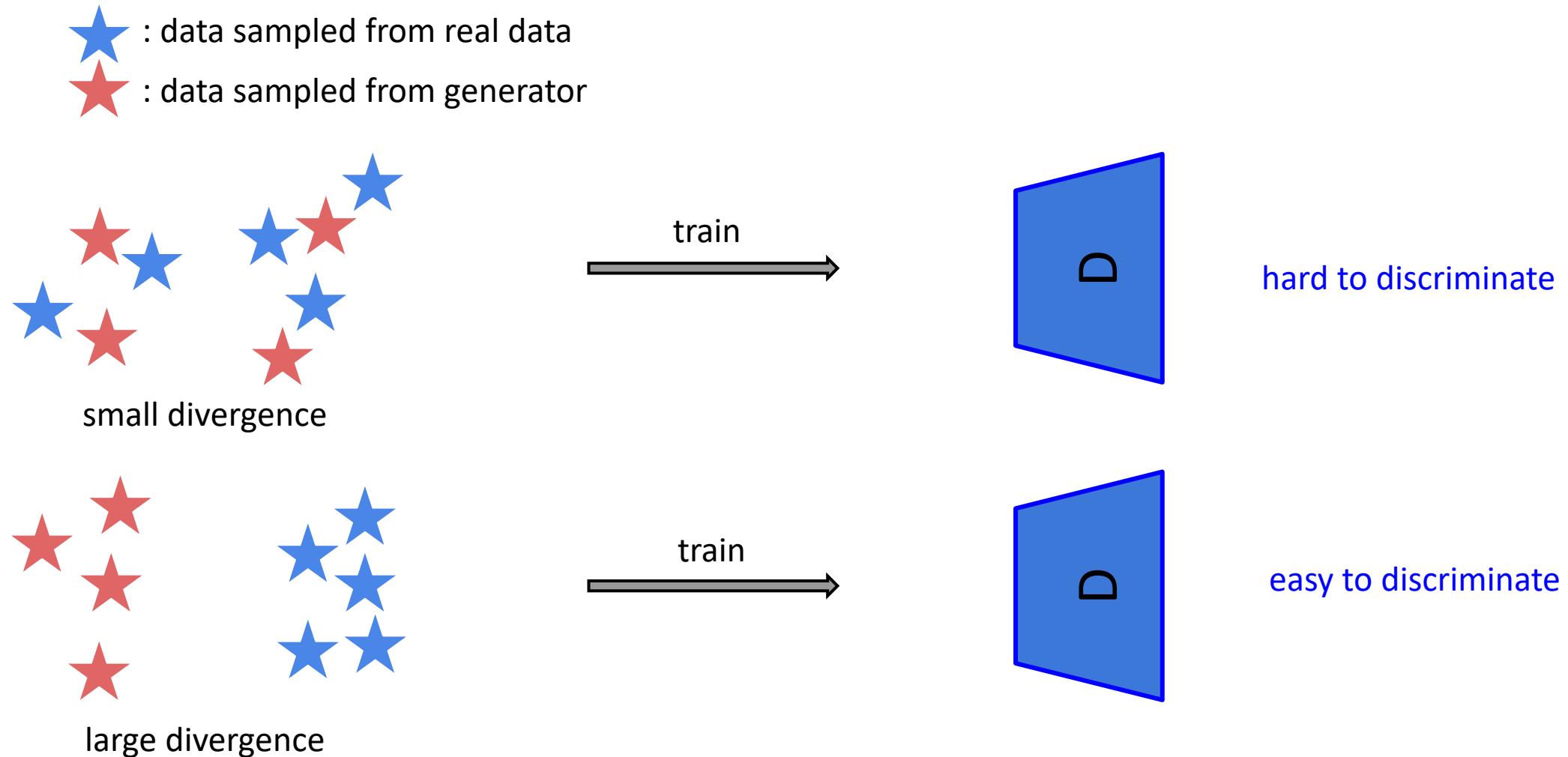
$$\max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim p_z(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$



related to JS divergence

# GAN Architecture and Objective

Björn W. Schuller



train G and D with **minimax** objective function:

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\underbrace{\log D_{\theta_d}(x)}_{\text{Discriminator output for real data } x}] + \mathbb{E}_{z \sim p_z(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)})]$$

- **Gradient ascent** on Discriminator:

$$\max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim p_z(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

- **Gradient descent** on Generator:

$$\min_{\theta_g} \mathbb{E}_{z \sim p_z(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimising this generator objective does not work well!

train G and D with **minimax** objective function:

$$\min_{\theta_g} \max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\underbrace{\log D_{\theta_d}(x)}_{\text{Discriminator output for real data } x}] + \mathbb{E}_{z \sim p_z(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)})]$$

- **Gradient ascent** on Discriminator:

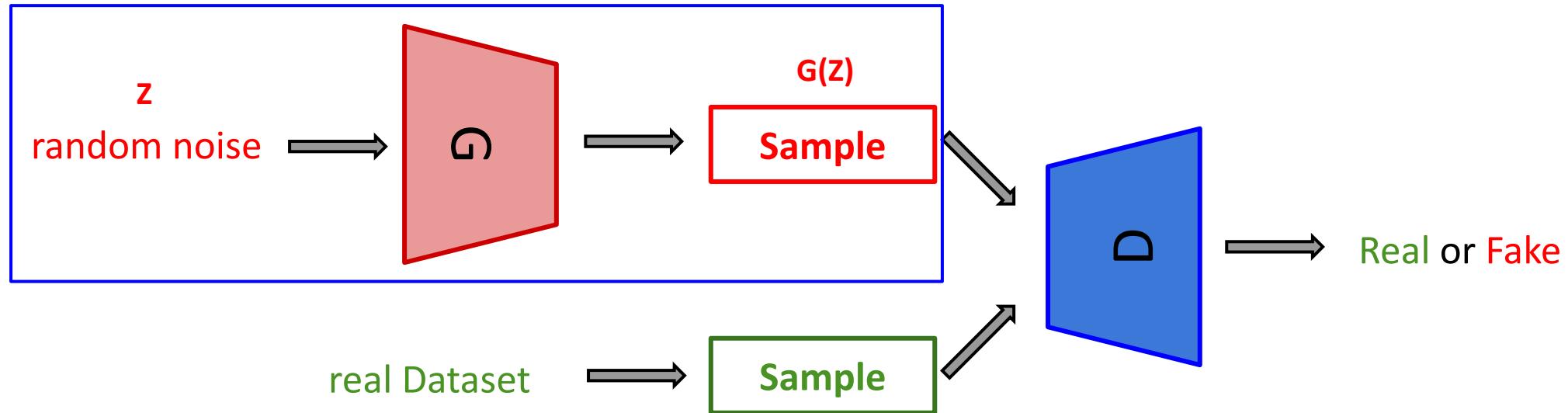
$$\max_{\theta_d} [\mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim p_z(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

- Instead: **Gradient ascent** on Generator, **different objective**:

$$\max_{\theta_g} \mathbb{E}_{z \sim p_z(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

After training

use the **generator** network to generate new samples



## Pros:

- realistic, state-of-the-art samples

## Cons:

- challenge to train (non-convergence, mode-collapse)
- cannot solve inference queries such as  $p(x)$

## Active areas of research:

- better objectives for more stable training
- GANs for all kinds of applications

## Non-convergence:

most deep learning models involve only one single player

- aim to maximise its objective (minimise its loss)
- use SGD with backpropagation to optimise
- therefore it has convergence guarantees

GANs involve two players

- compete with each other, co-evolution of both
- SGD was not designed to find the Nash equilibrium of the game
- might not converge to the Nash equilibrium at all

## Non-convergence:



: data sampled from real data



: data sampled from generator

*JS divergence is log2  
if two distributions do not overlap*

suppose if two distributions do not overlap,  
D achieves 100% accuracy



same objective value is obtained



same divergence

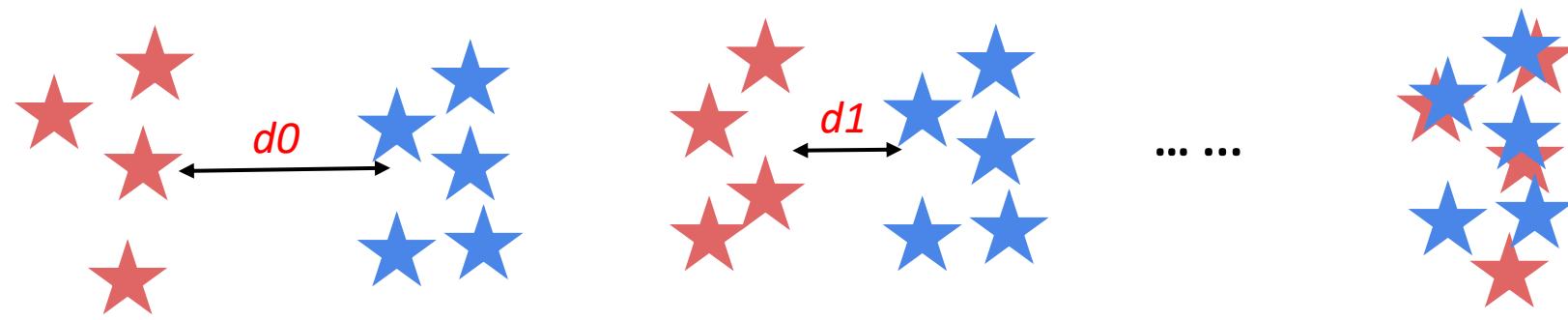


training of GAN **stucks**

## WGAN

- evaluate **wasserstein distance** between  $P_{data}$  and  $P_G$

$$\max_{D \in 1-\text{Lipschitz}} [\mathbb{E}_{x \sim p_{\text{data}}(x)}[D(x)] - \mathbb{E}_{z \sim p_z(z)}[D(G(z))]]$$



$$W(P_{G0}, P_{data}) = d_0$$

$$W(P_{G1}, P_{data}) = d_1$$

... ...

$$W(P_{G99}, P_{data}) = 0$$



- Original WGAN with Weight clipping
  - force the weights  $w$  in range of  $[-c, c]$
  - smooth the training of the discriminator
- WGAN-GP : improved WGAN with Gradient Penalty
  - a penalty on the gradient norm in the objective function
- SN-GAN : improved WGAN with spectral normalisation
  - keep gradient norm smaller than 1 everywhere

*Martin Arjovsky, et al., Wasserstein GAN, arXiv, 2017*

*Ishaan Gulrajani, et al., Improved Training of Wasserstein GANs, NIPS, 2017*

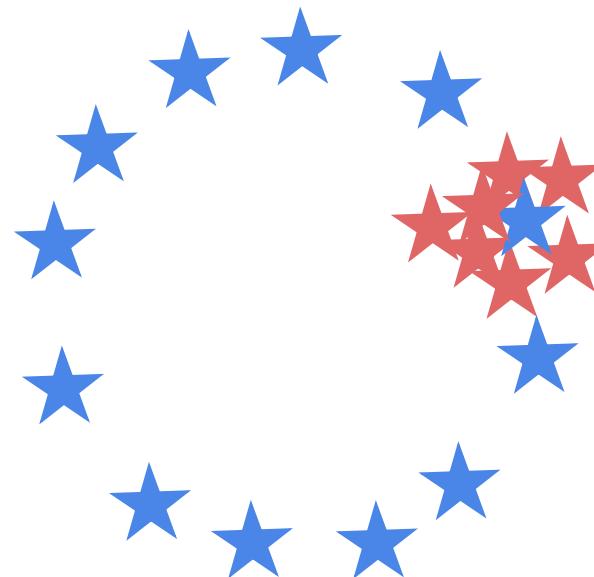
*Takeru Miyato, et al., Spectral Normalization for Generative Adversarial Networks, ICLR, 2018*

## Mode Collapse:

Generator fails to output diverse samples

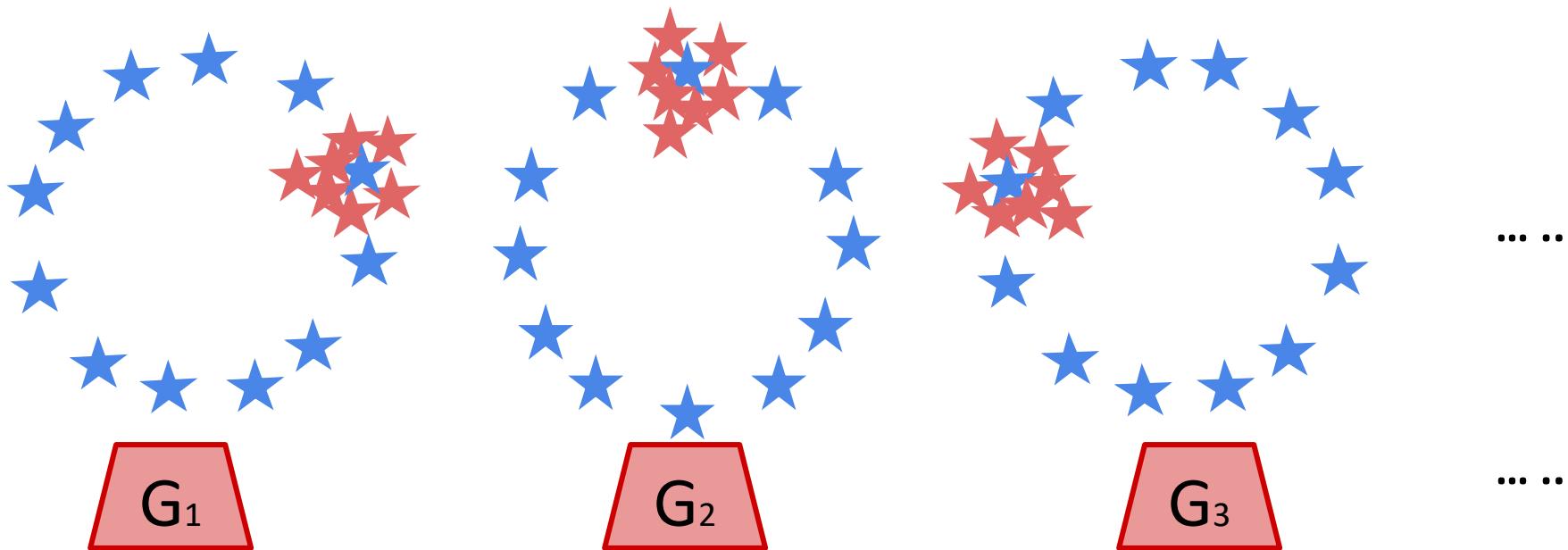
★ : data sampled from real data

★ : data sampled from generator



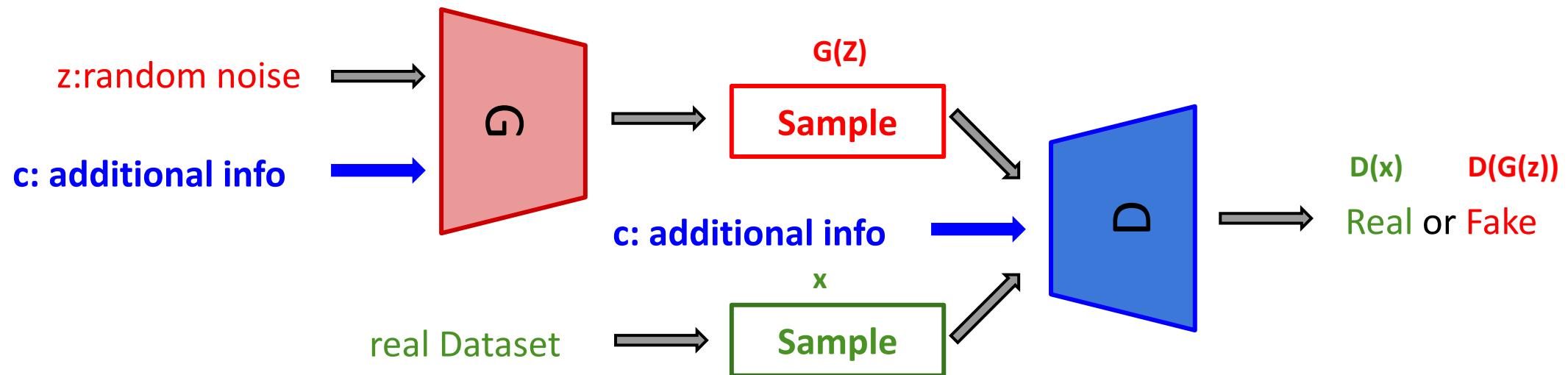
## Ensemble

- train a set of generators:  $\{G_1, G_2, \dots, G_N\}$
- random select one generator when generating a new sample



## Conditional GANs

- generate samples conditioned on *additional information* (e.g., labels)
- better performance with *explicit supervision*



But mainly in **four** types:

- Optimisation-based

*WGAN, Energy Based GAN, Least Squares GAN, Loss-Sensitive GAN, Correlational GAN, ...*

- Structure-based

*cGAN, semi-supervised cGAN, BiGAN, CycleGAN, DiscoGAN, Triple-GAN, ...*

- Network-type-based

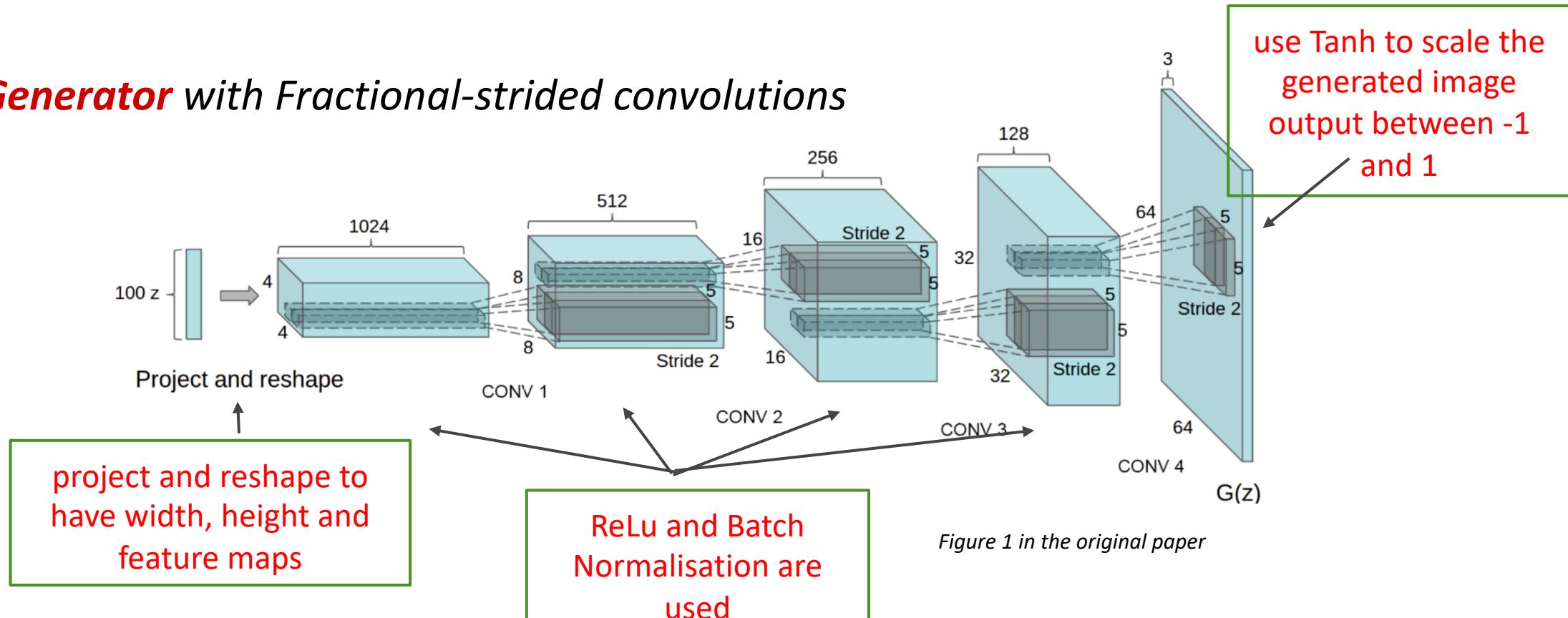
*deep convolutional GAN, C-RNN-GAN, attnGAN, CapsuleGAN, AEGAN with autoencoders, ...*

- Task-oriented

*ArtGAN, WaveGAN, SEGAN for speech enhancement, VoiceGAN for voice impersonation, ...*

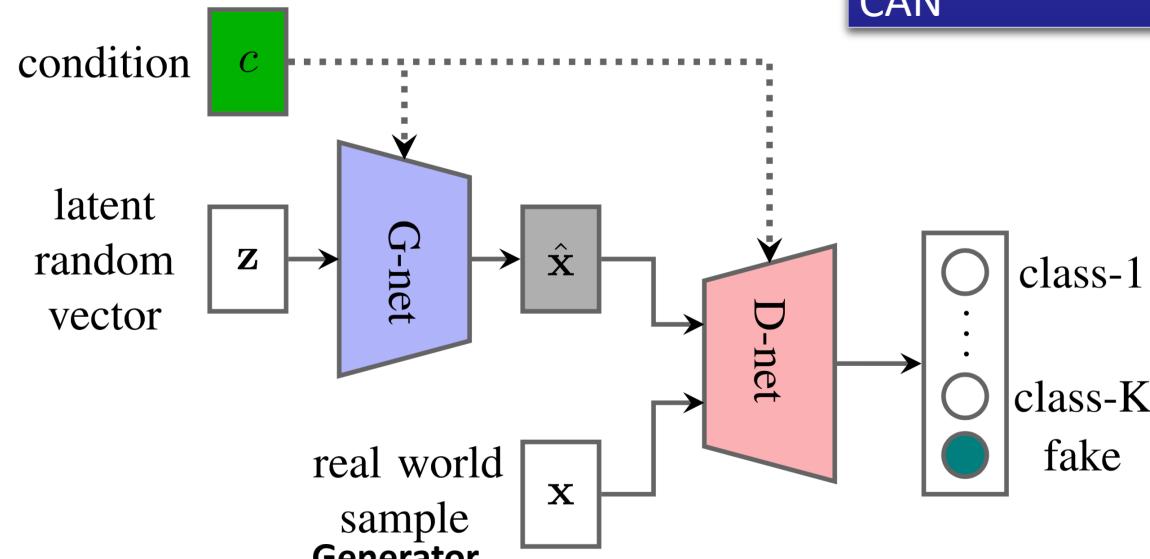
## Deep Convolutional GANs (DCGANs)

### **Generator with Fractional-strided convolutions**

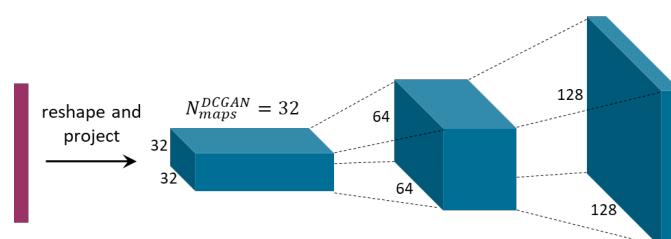


Alec Radford, et al., Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR, 2016

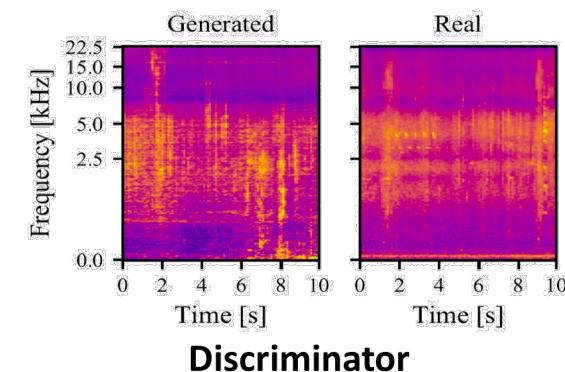
- **Conditional Adversarial Nets**



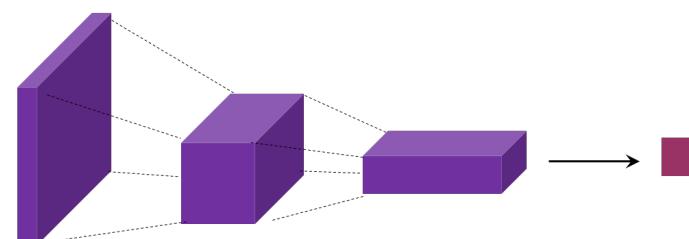
100-D Gaussian Noise    1<sup>st</sup> Convolutional Layer    2<sup>nd</sup> Convolutional Layer    Generator Output



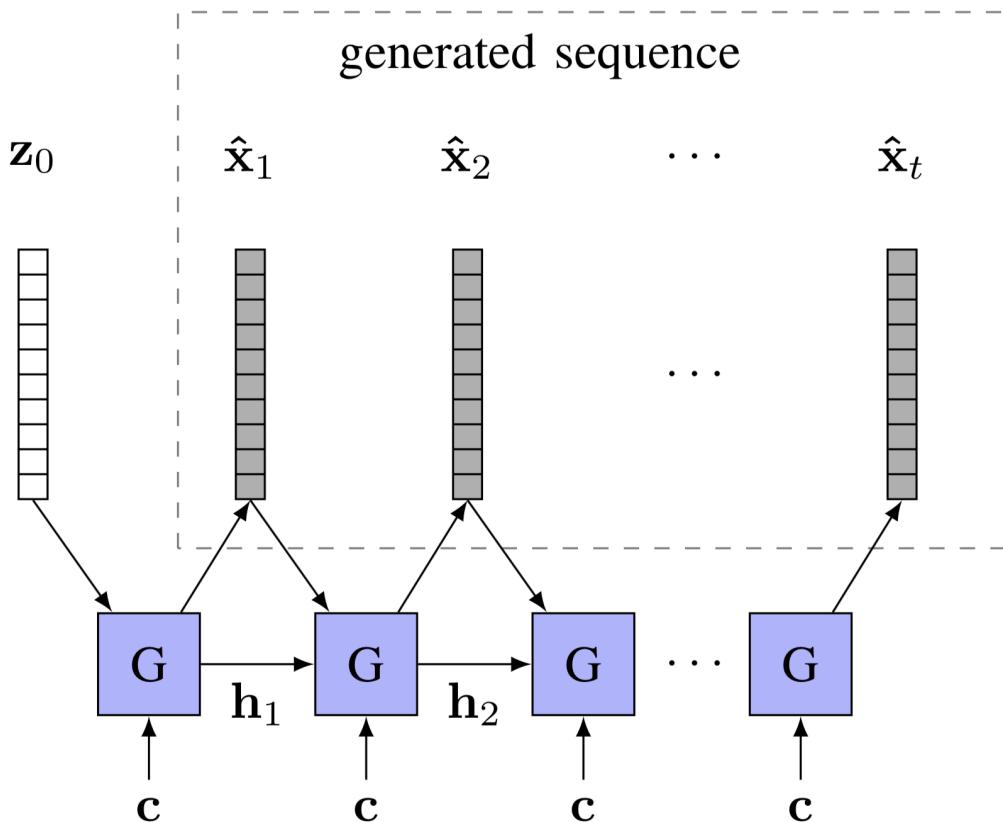
	Arousal	Valence
CCC Recola		
ComParE+LSTM	.382	.187
e2e (2016)	.686	.261
CAN	.737	.455



Discriminator

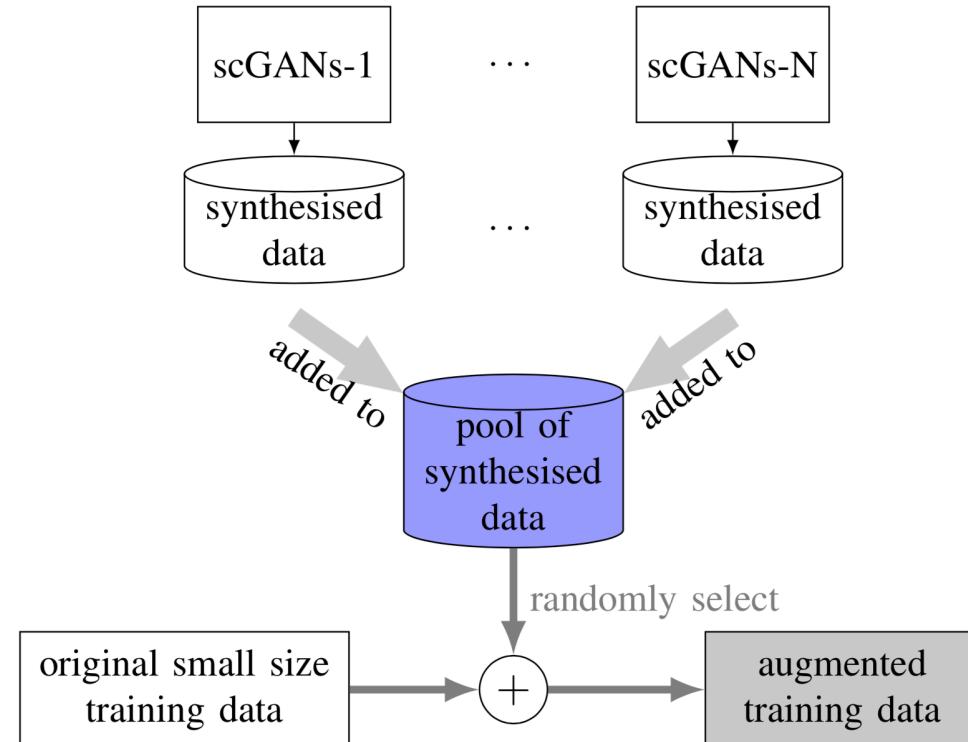


- **Sequence Generation by Recurrent Generator**

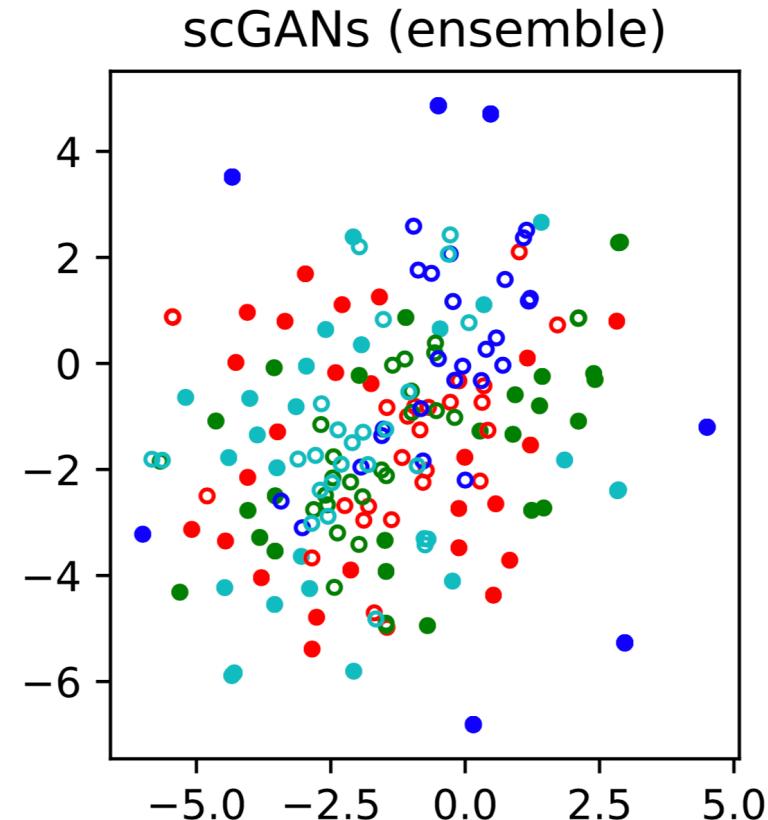
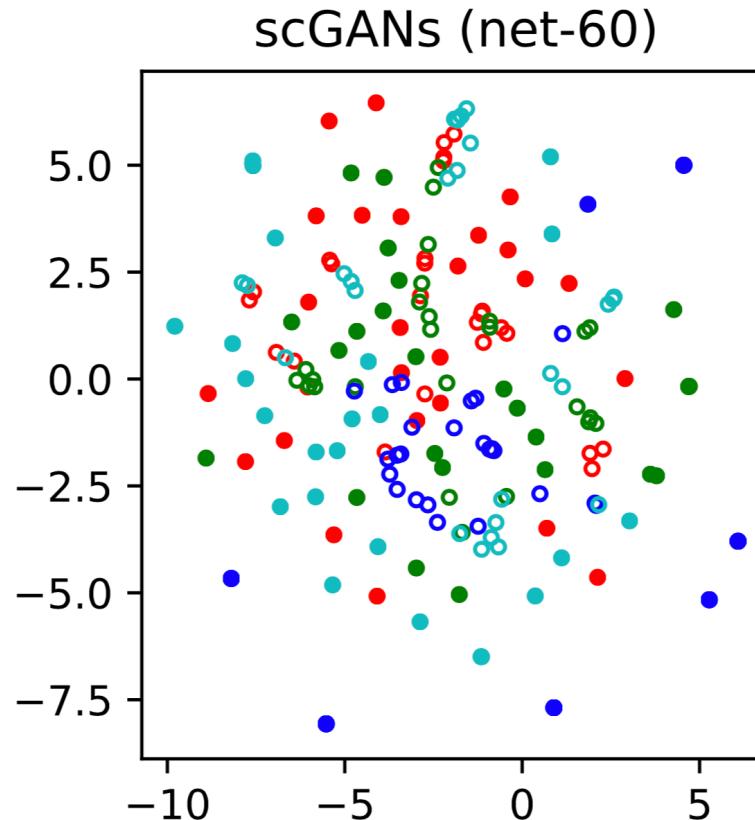


- Ensemble semi-supervised CANs against Model Collapse

ComParE 17 Snoring	%UA
Baseline (BoAW)	48.2
scGAN	54.8
scGAN Ensemble	56.7



- Ensemble GANs against Model Collapse: t-SNE Visualisation



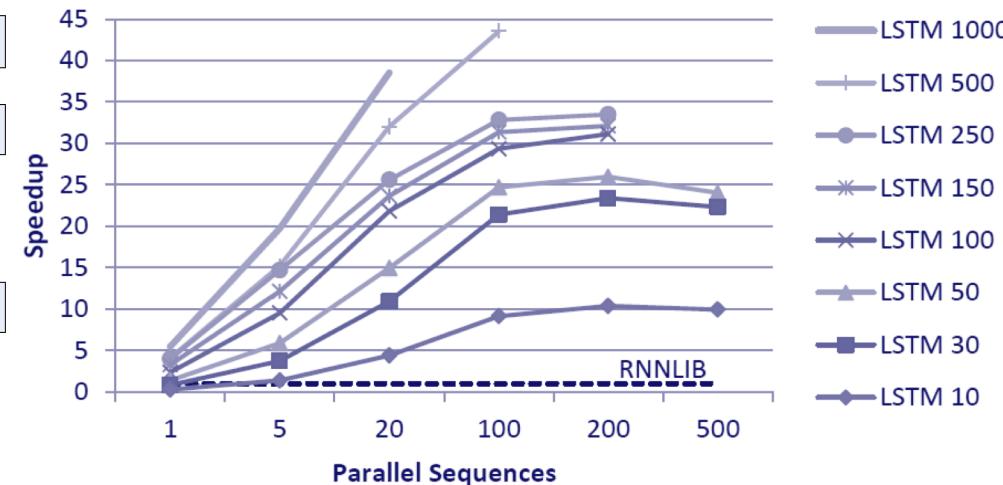
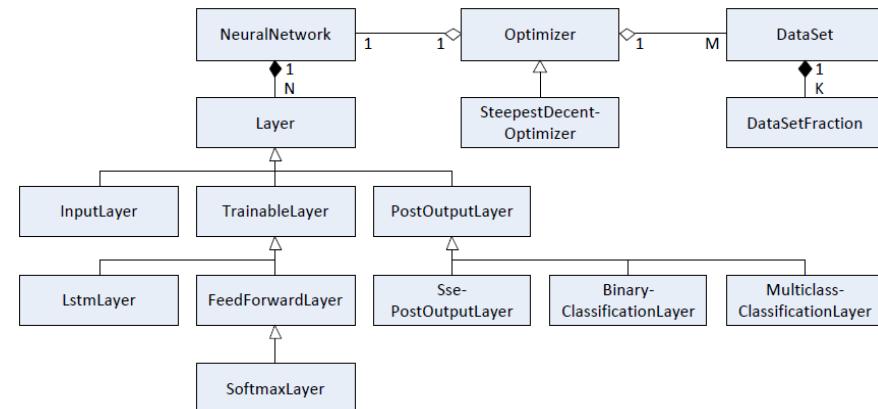
Green?



- GPU-Learning
  - 10 – 1k LSTM cells,
  - 2k – 4Mio parameters
  - GPGPU

## CURRENNT

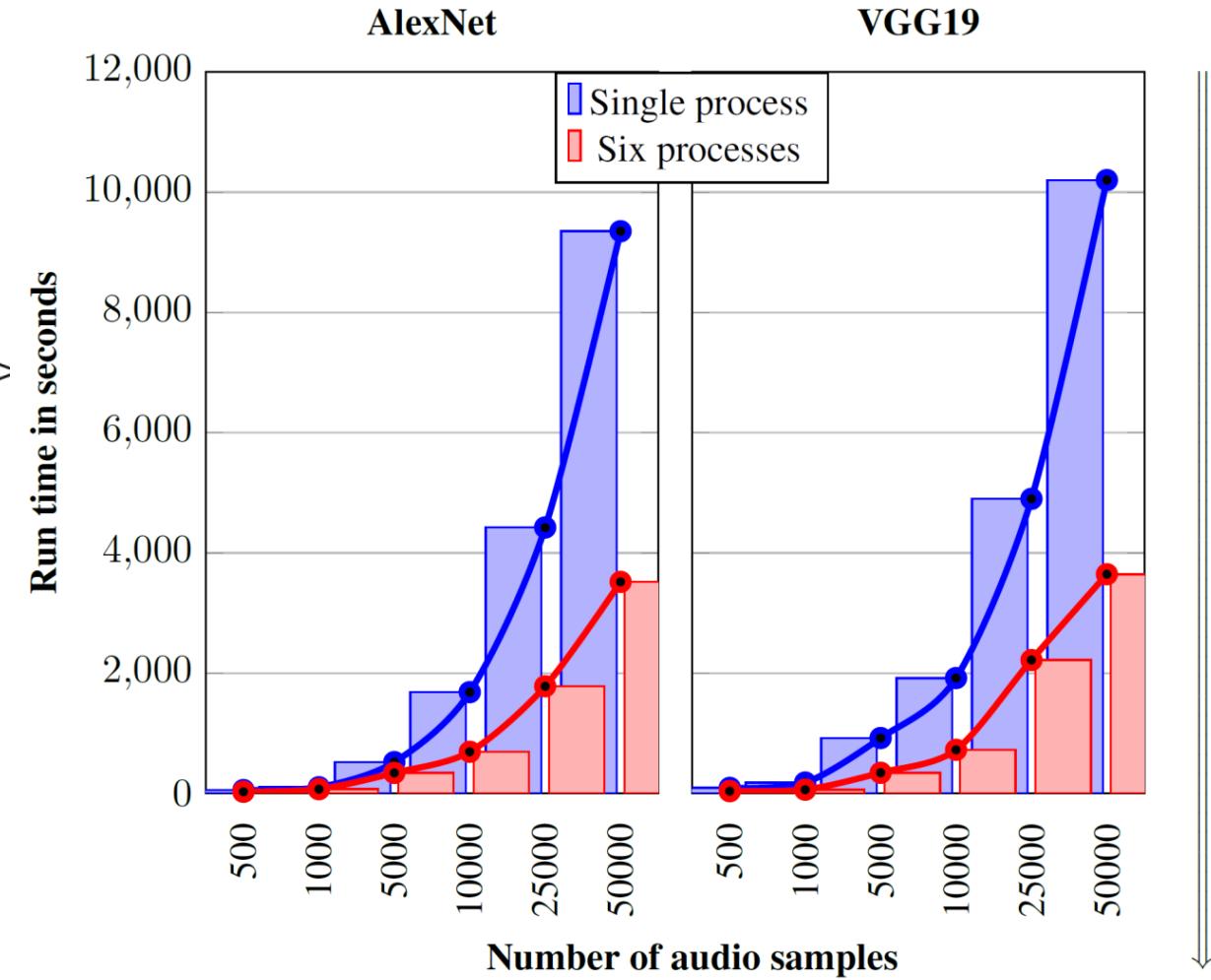
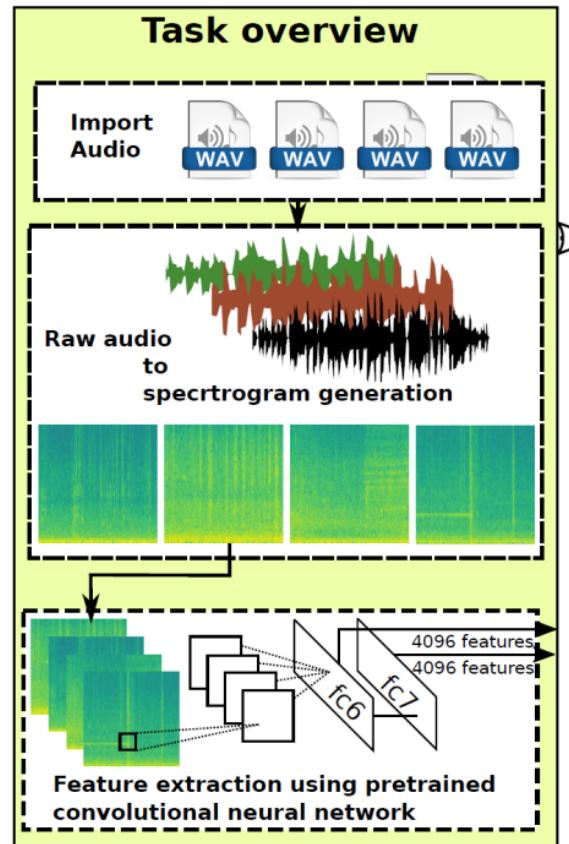
	CHiME 2	RNNLIB	CURRENNT		
#seq.		1	1	10	200
speedup	(1)	2	13	22	



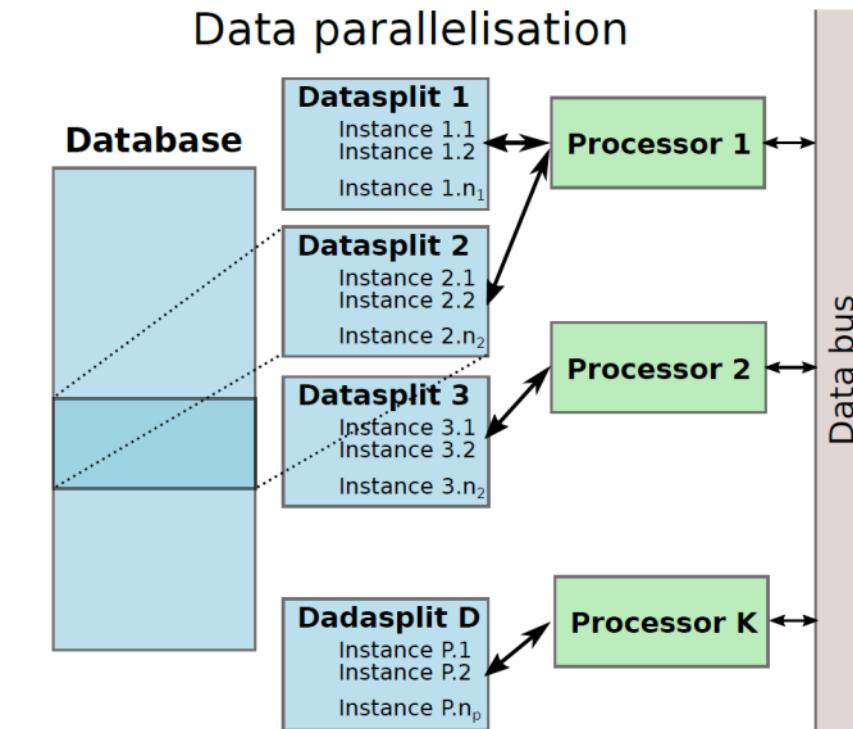
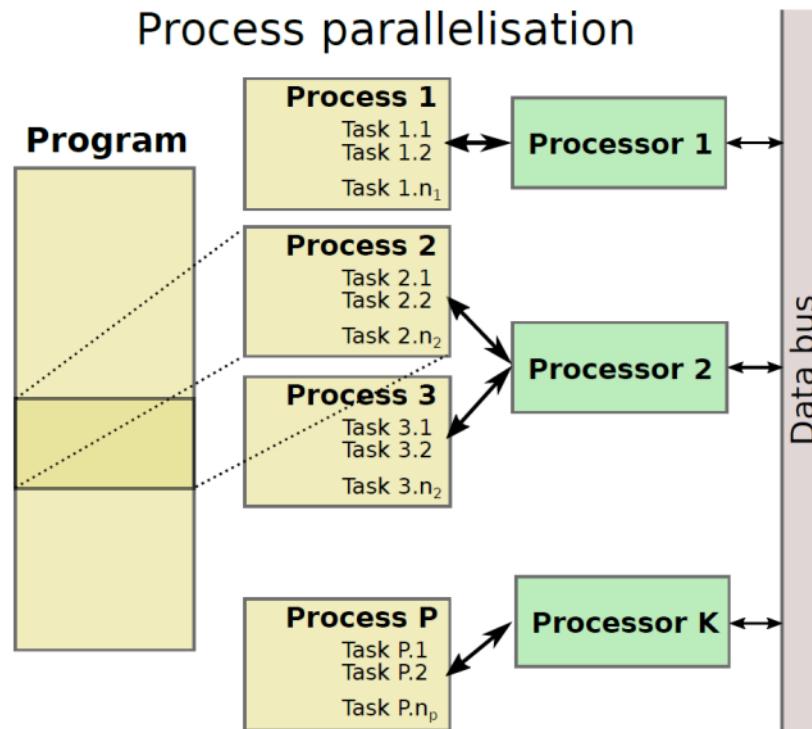
# Speeding Up: GPU.

Björn W. Schuller

- Parallel...

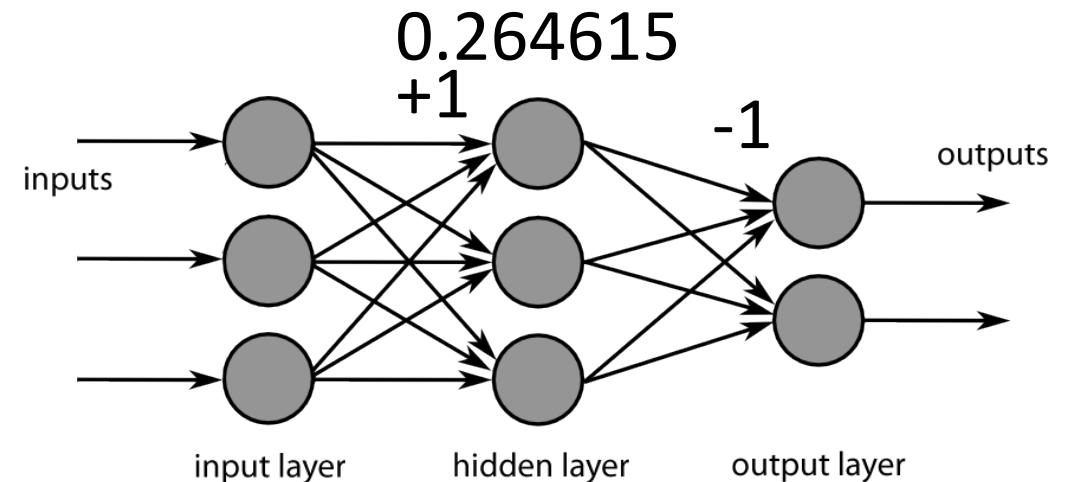


- Parallelisation



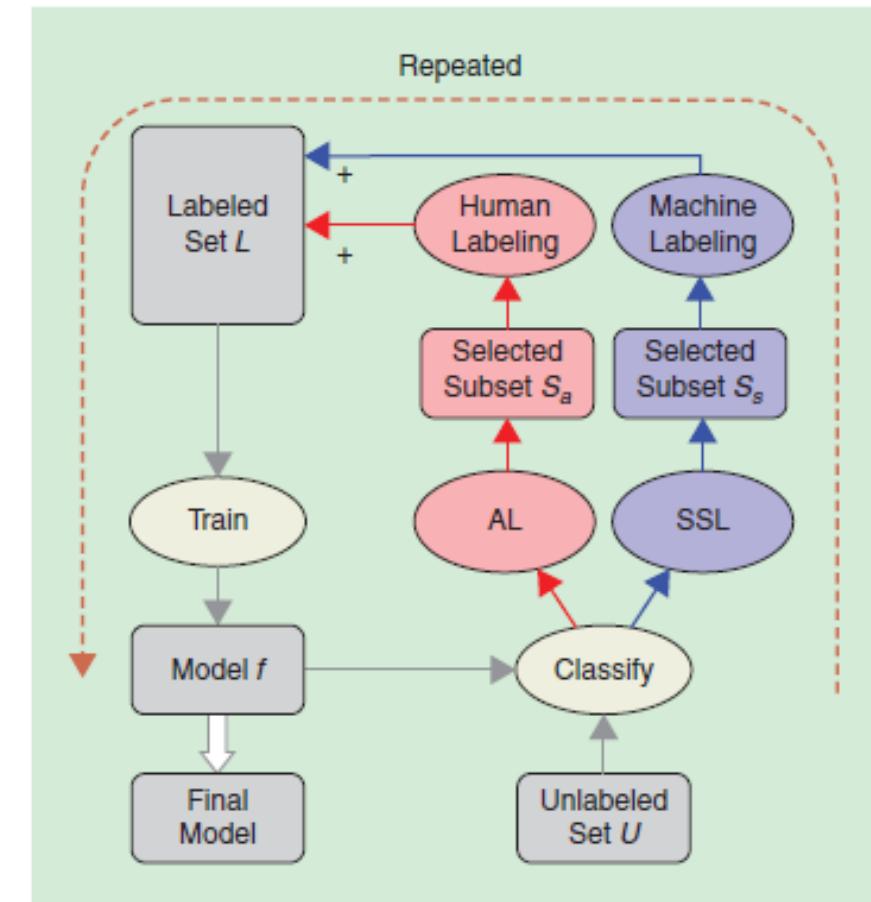
## Alternate Network Architectures

- Low precision networks
  - Less precision in weights and activations
    - Binary Neural Networks {-1,1}
    - Ternary neural networks {-1,0,1}
    - Squeezed Nets...  
e.g., input/activation/weights  
3bit / 3bit / 4bit, etc.



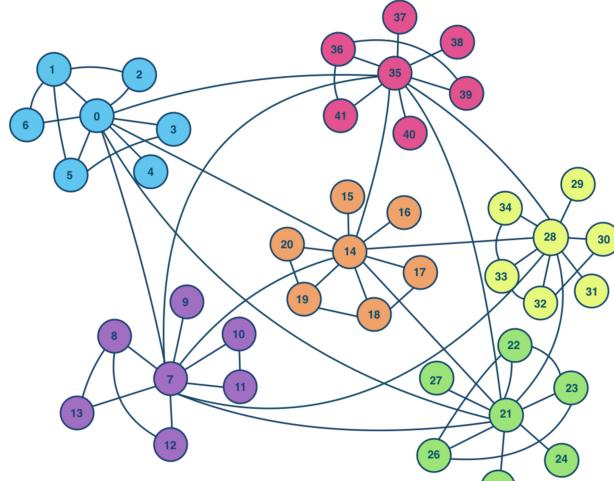
- SSL Techniques
  - Annotation with a bare minimum of human intervention
  - However, the performance of SSL is hampered by error accumulation
- AL techniques
  - Achieve higher accuracy with fewer training labels
    - Actively selecting the data it can best learn from
  - However, requires a considerable human intervention.

- Best of both worlds
  - Successively updating the labelled dataset by AL then by SSL
    - Note: AL first as it reduces error accumulation
  - Shown in literature to generally outperform both methods

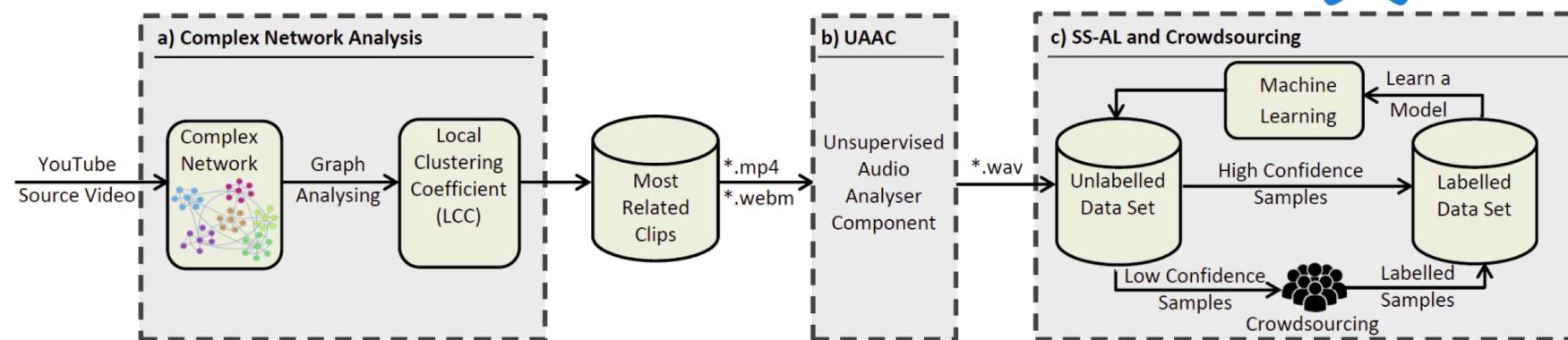


# Cooperative Learning

Björn W. Schuller



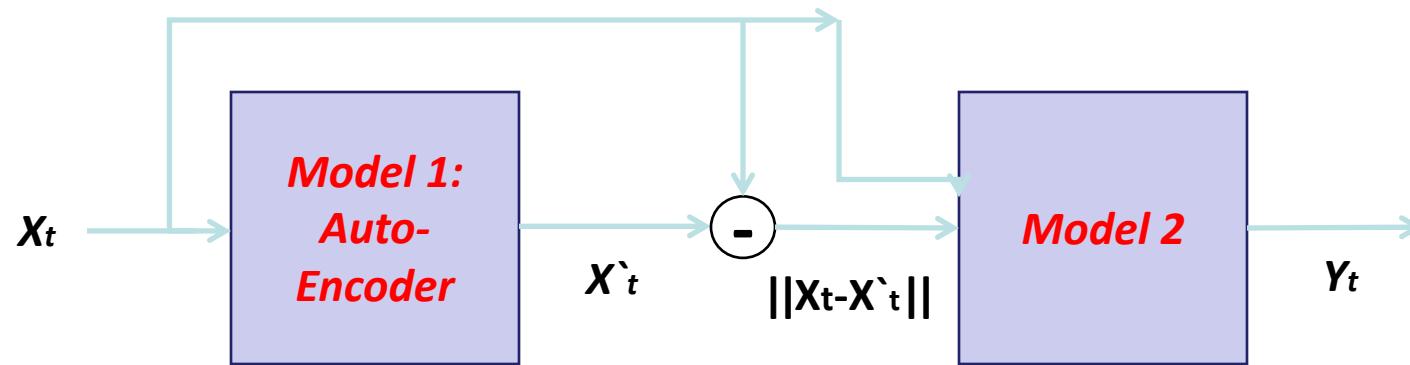
%UA	3 hrs
Freezing	70.2
Intoxication	72.6
Screaming	97.0
Threatening	73.8
Coughing	97.6
Sneezing	85.2



# Hacks?



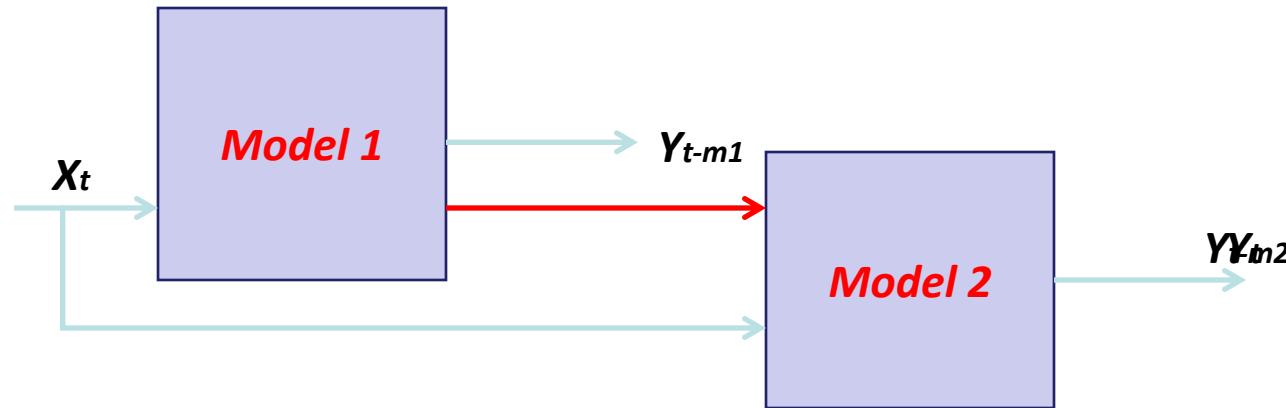
- **Reconstruction Error (RE) in 2 Levels**  
RE of Auto-Encoder as additional input feature



Either: Low Level Descriptors (LLD) or Statistical functionals

Deep BLSTM RNN

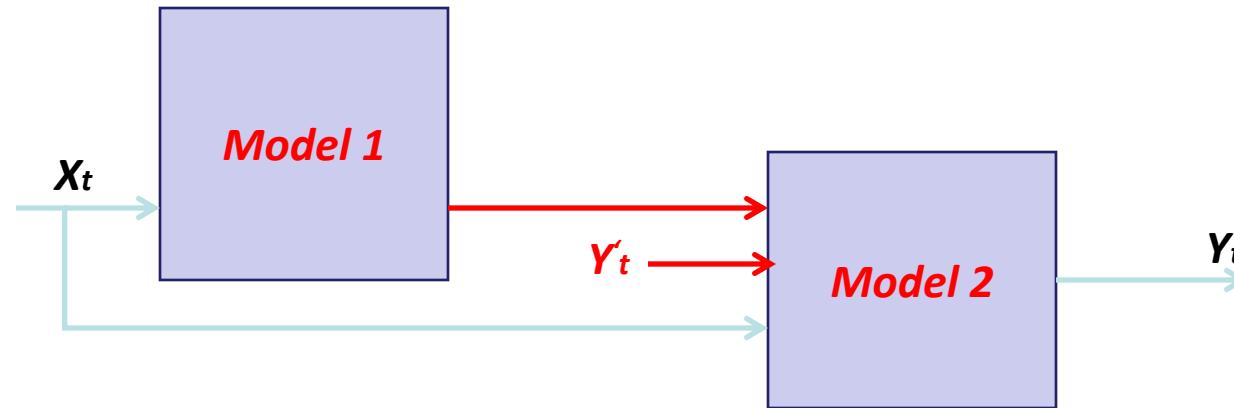
concatenate two models for combined strengths



tandem structure:

outputs predicted by first model combined w/ features as input

(Pseudo prediction: simulated by applying noise to the true label)

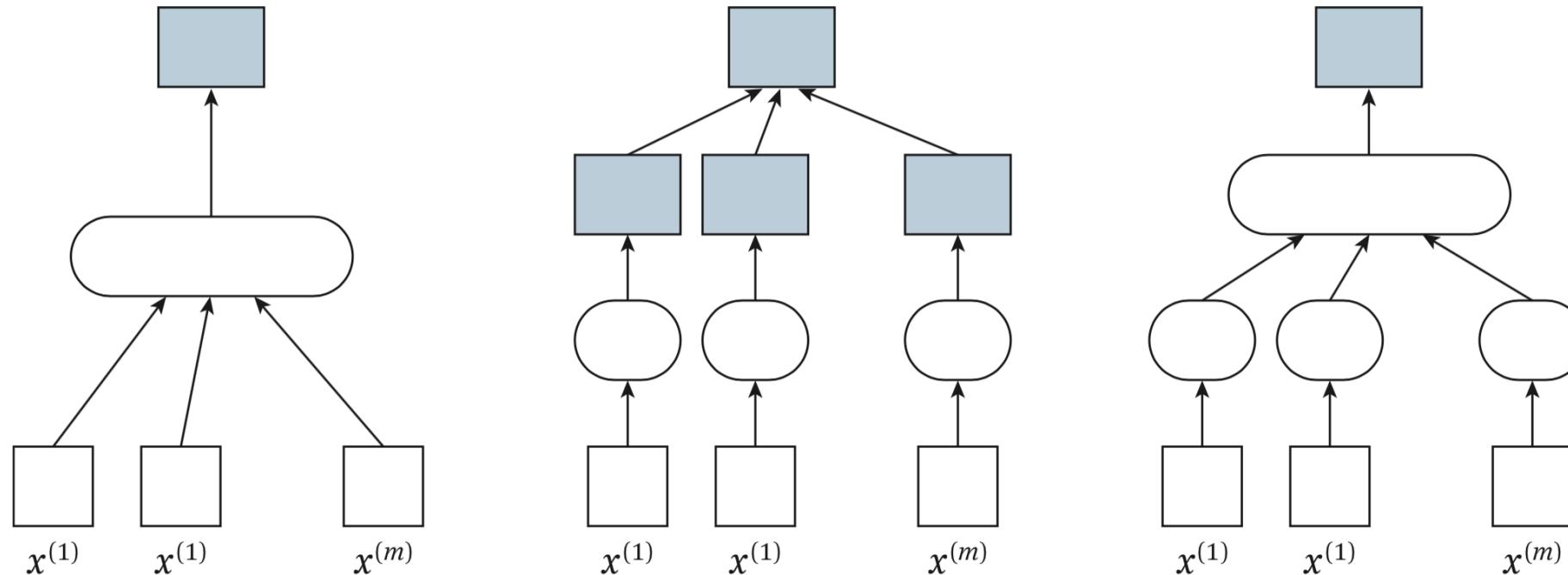


- **SVR**
  - + more likely to achieve global optimal solution
  - Not context-sensitive
- **BLSTM-RNN**
  - + context-sensitive
  - Easily trapped in local minimum and risk of overfitting

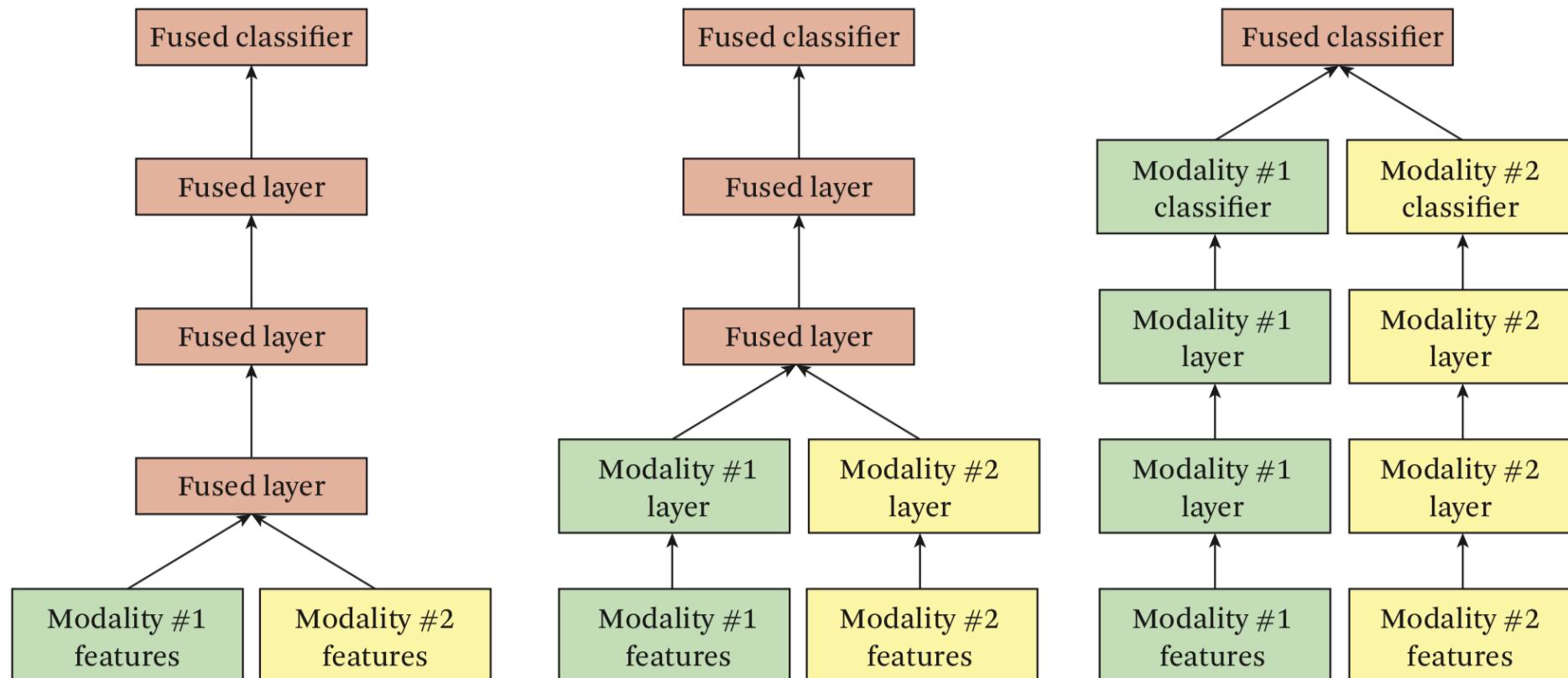
Multimodal Fusion.



- Early, Late & Intermediate Fusion
  - (deep) integration drawn as multi-layer networks
  - Squares: inputs in different modalities (each of which may be a vector)

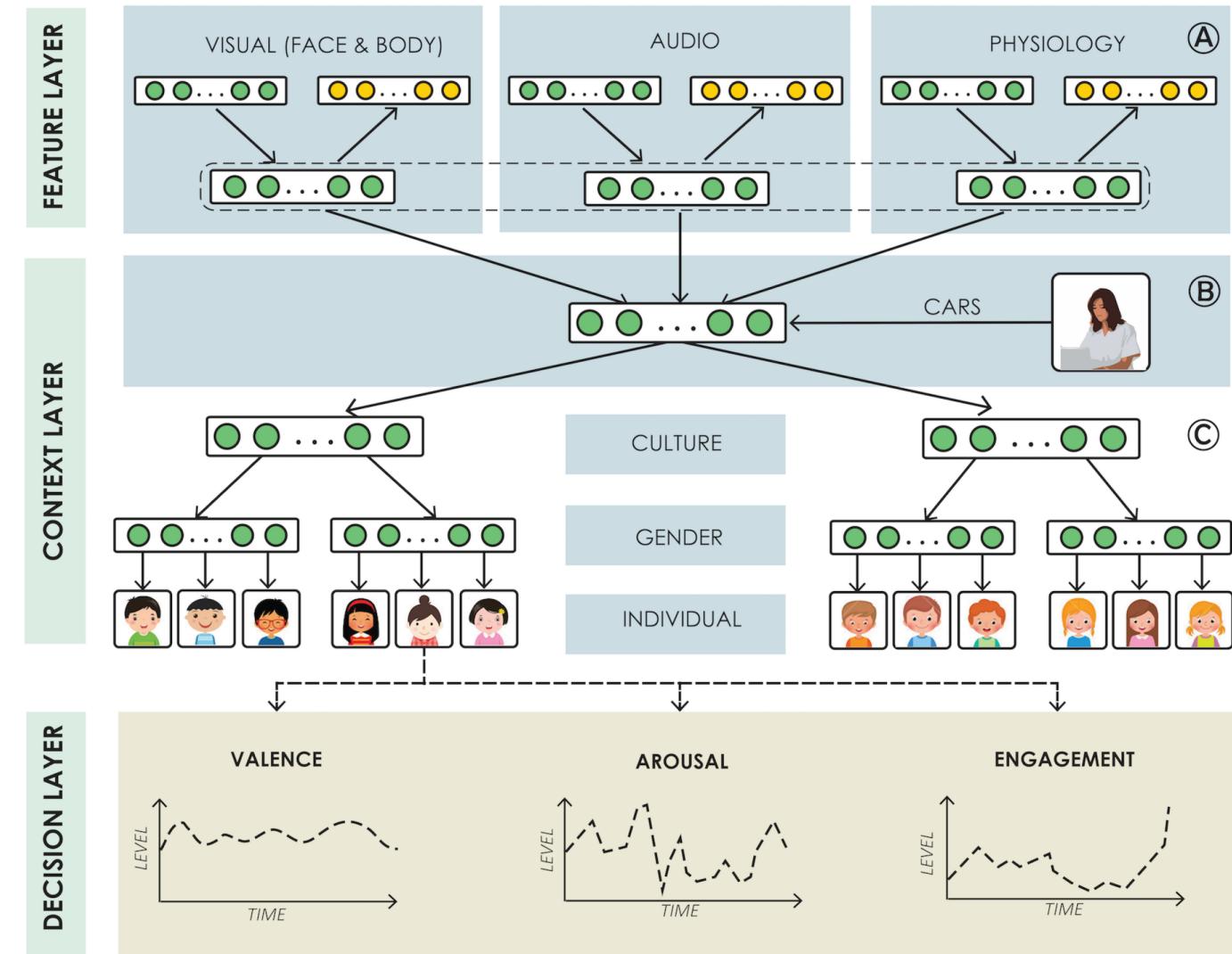


- Early, Intermediate & Late Fusion

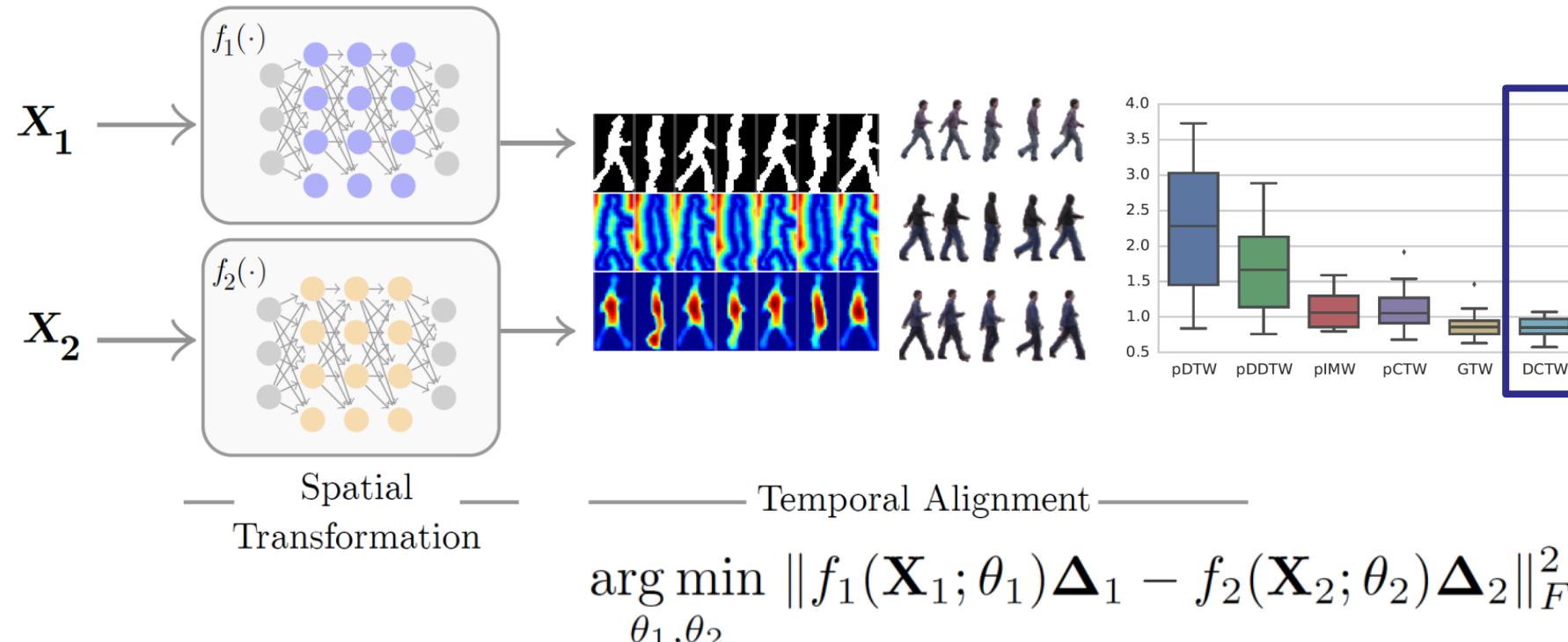


# Multimodal Fusion.

Björn W. Schuller



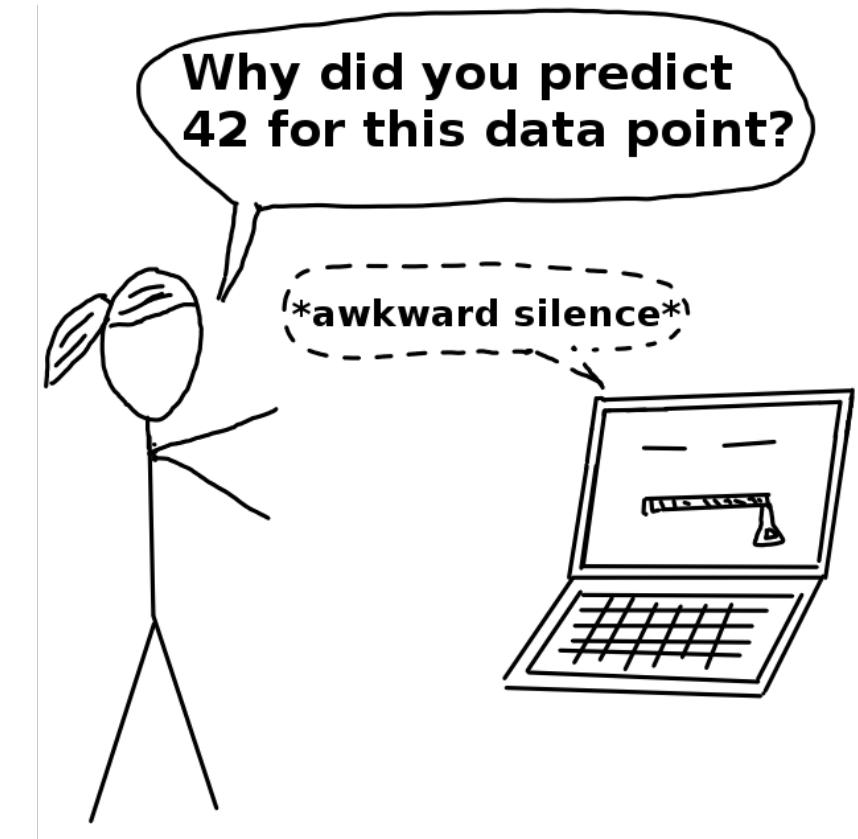
- Deep Cross-Modal Warping
  - Maximise correlation (CCA), introduce hierarchy (deep)



# Explainable?



- **Interpretable Machine Learning**
  - Methods and models that make the behaviour and predictions of machine learning systems understandable to humans
- **Black-Box Systems**
  - Models that cannot be understood by looking at their parameters
  - These models are not interpretable



- **What is an explanation?**
  - An explanation is the **answer to a why-question**
    - E.g., Why did not the treatment work on the patient
- **Properties of good explanations**
  - Selective
  - Contrastive
  - Focus on the abnormal
  - Truthful
  - General and probable
  - Align with our prior beliefs

- **Gain Knowledge**

- The goal of science is to gain knowledge
- Many ML problems are solved with big datasets and black box machine learning models.
- The model itself becomes the source of knowledge instead of the data.
- Interpretability makes it possible to extract this additional knowledge captured by the model.

- **Easier Debugging and Auditing**
  - **Fairness**: Ensuring that predictions are unbiased and do not implicitly or explicitly discriminate against protected groups.
  - **Privacy**: Ensuring that sensitive information is protected
  - **Reliability or Robustness**: Ensuring that small changes in the input do not lead to large changes in the prediction.
  - **Causality**: Check that only causal relationships modelled
  - **Trust**: It is easier for humans to trust a system that explains its decisions compared to a black box

- **Interpretability is not always needed**
  - Interpretability is not required if the model has no significant ‘real-world’ impact
    - E.g., Recommender systems
  - Interpretability is not required when the problem is well studied
    - E.g., Handwritten digit recognition
  - Interpretability might enable people or programs to manipulate the system

- **Intrinsic or post hoc?**

- Is interpretability achieved by restricting the complexity of the machine learning model (intrinsic) or by applying methods that analyse the model after training (post hoc).
- Intrinsic interpretability refers to machine learning models that are considered interpretable due to their simple structure, such as short decision trees or sparse linear models.
- Post hoc interpretability refers to the application of interpretation methods after model training.

- **Types**

- **Feature summary statistic:** Methods which provide summary statistics for each feature
- **Feature summary visualization:** visualisation of feature summary statistics
- **Model internals:** visualisation of summaries of model internals e.g. learnt weights
- **Data point:** This category includes all methods that return data points to make a model interpretable.
- **Intrinsically interpretable model:** Approximate black-box models with an interpretable model

- **Model-specific or model-agnostic?**
  - **Model-specific** interpretation tools are limited to specific model classes.
  - **Model-agnostic** tools can be used on any machine learning model and are applied after the model has been trained
- **Local or global?**
  - Does the interpretation method explain an individual prediction or the entire model behaviour

- **Separating the explanations from the machine learning model**
  - Main advantage of this approach is flexibility
    - Interpretation methods can be applied to any model
  - Greater predictive performance
    - Complex models can still be used
  - Allows for easier comparisons
    - Same explanation method can be applied to all models under consideration

