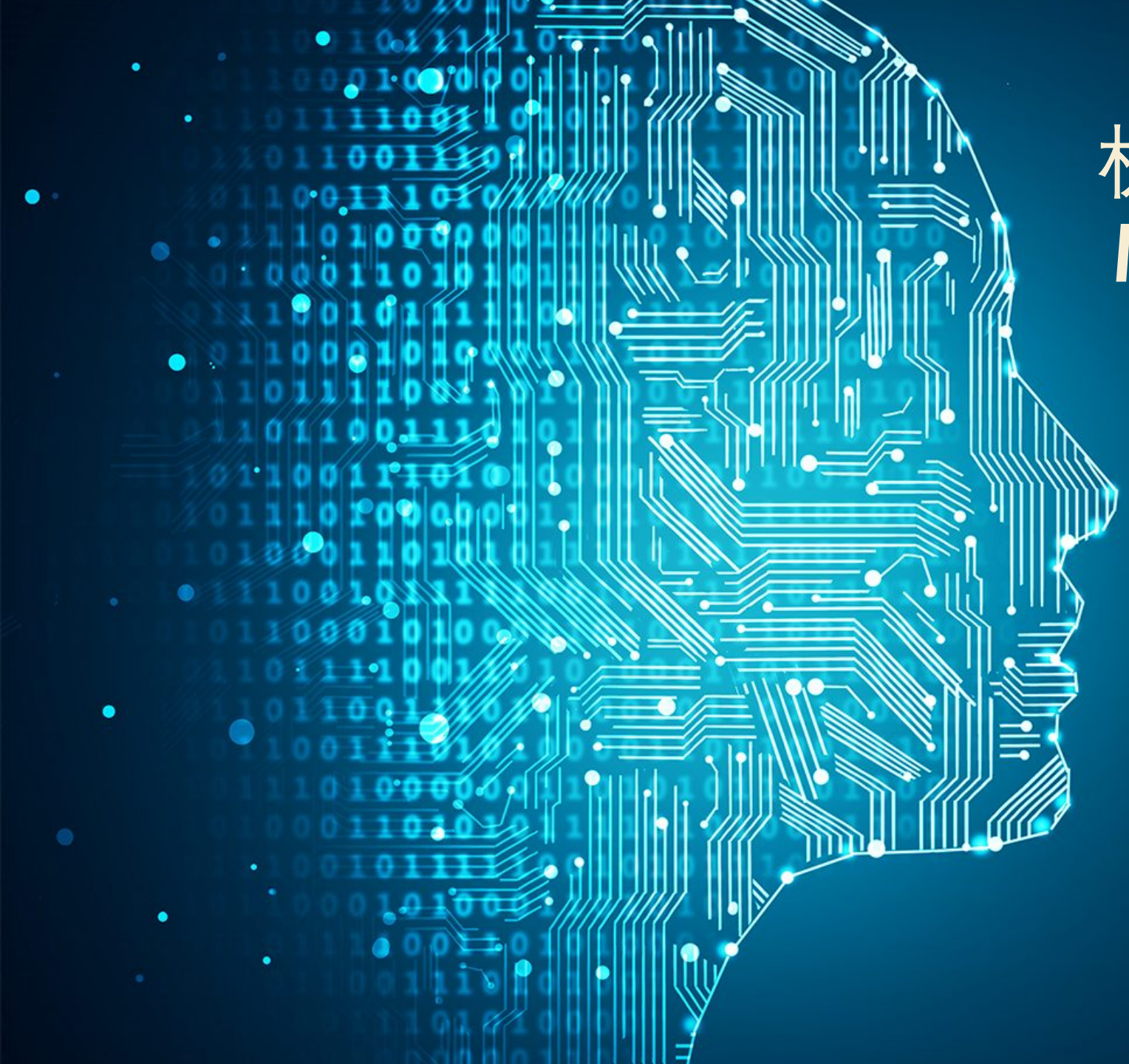


机器学习与人工智能
Machine Learning and Artificial Intelligence

Deep Learning

Yingjie Zhang (张颖婕)
Peking University
yingjiezhang@gsm.pku.edu.cn

2021 Fall

This is an INTRODUCTION!!

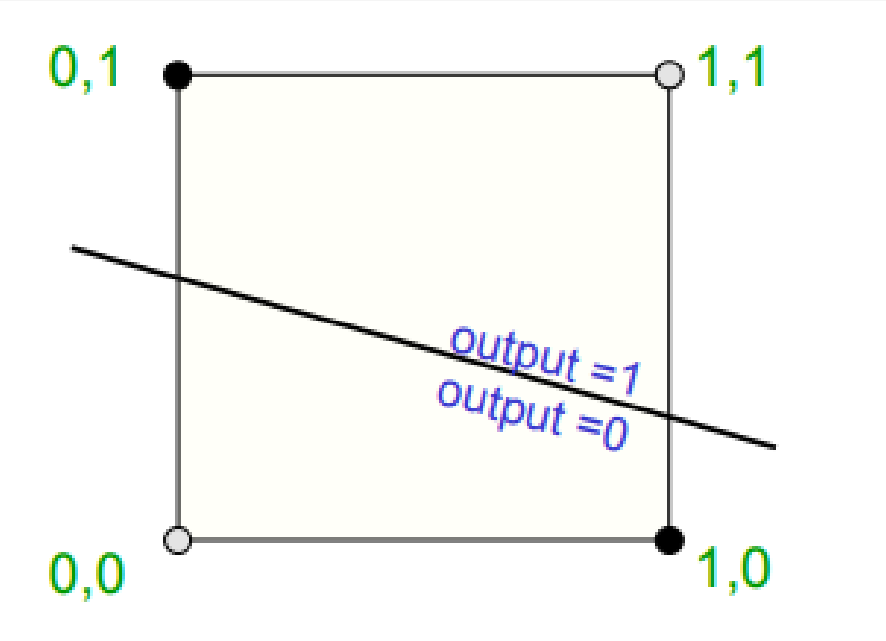
Agenda

- Neural Networks
- Backpropagation
- Deep Learning
- Python Practice
- Convolutional Neural Networks
- Recurrent Neural Networks

Neural Network

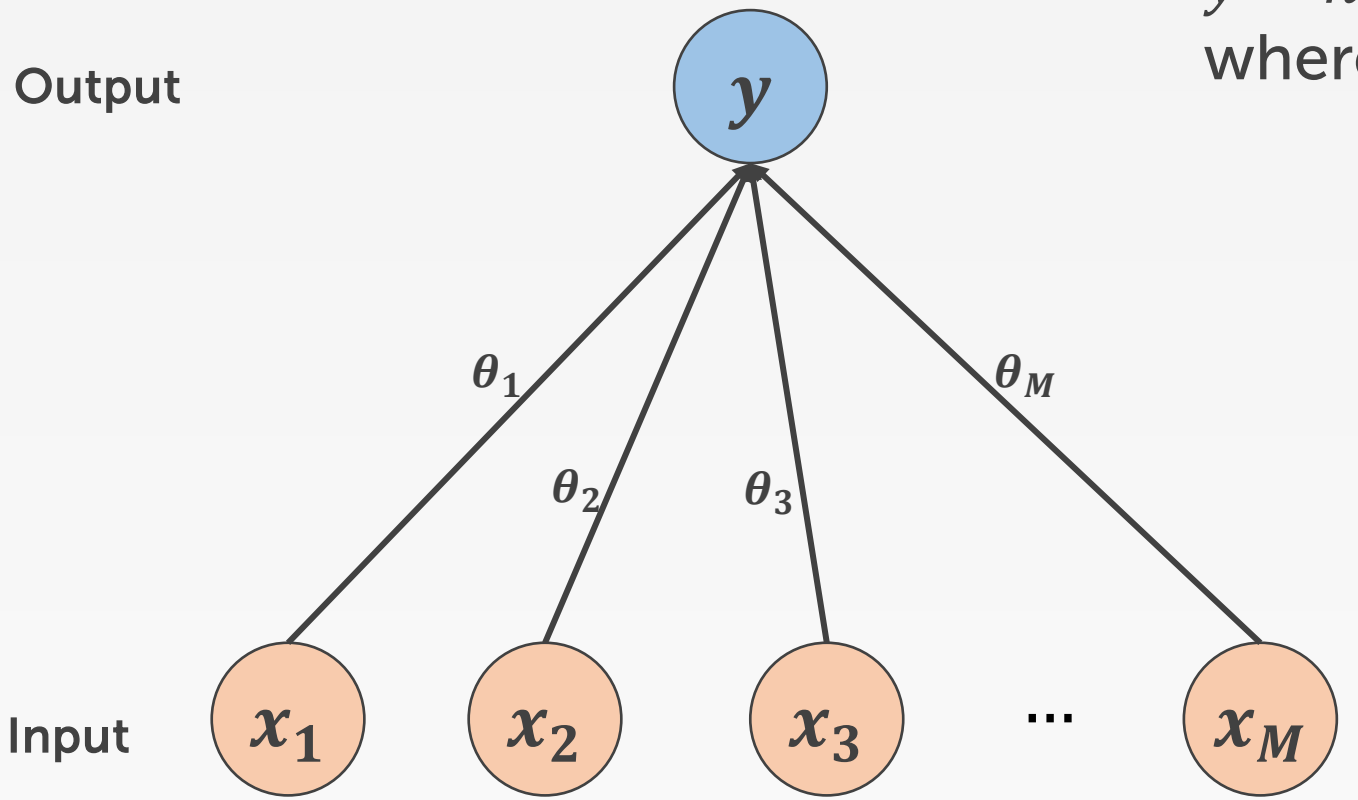
Limitation of Linear Classifier

- Linear classifiers (e.g., logistic regression) classify inputs based on linear combinations of features x_i
- Many decisions involve non-linear functions of the input



Linear Regression

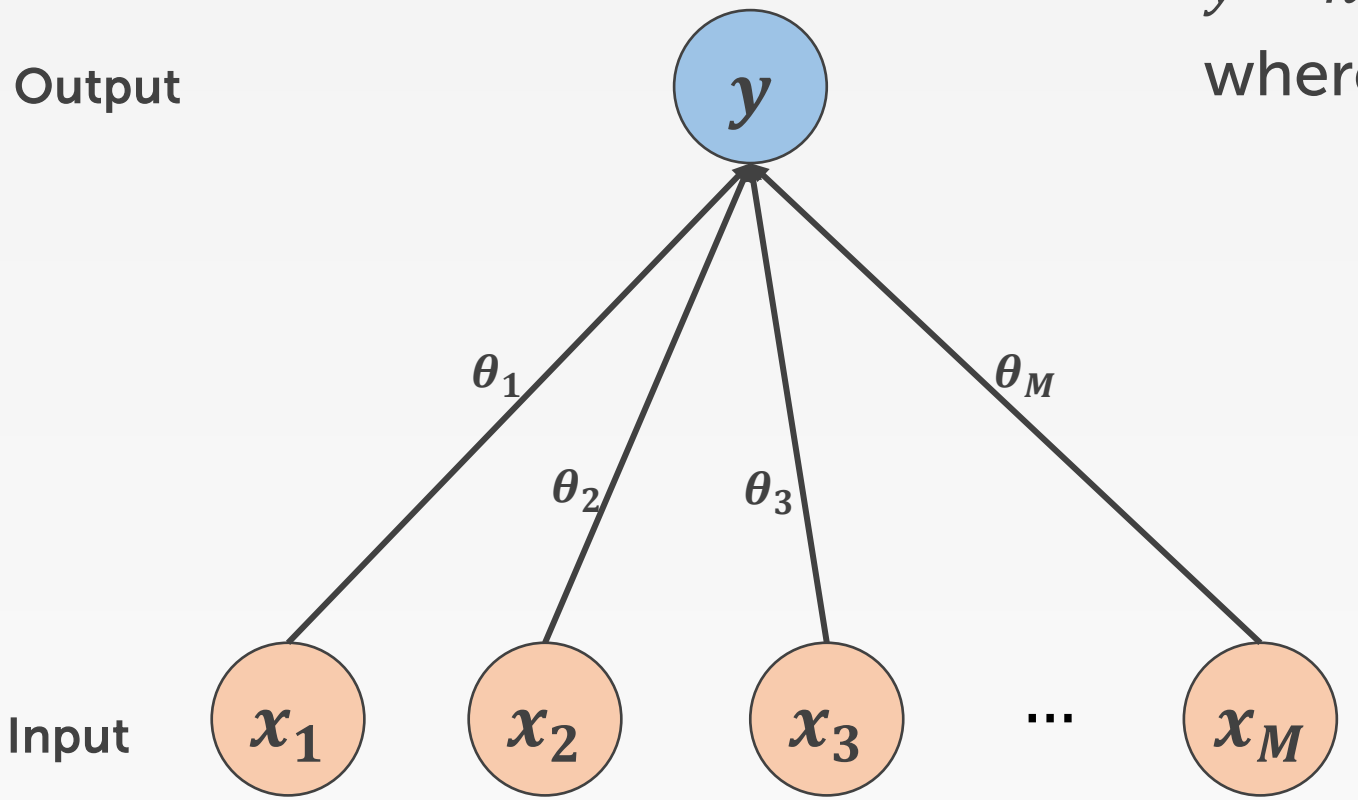
$$y = h_{\boldsymbol{\theta}}(x) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$
where $\sigma(a) = a$

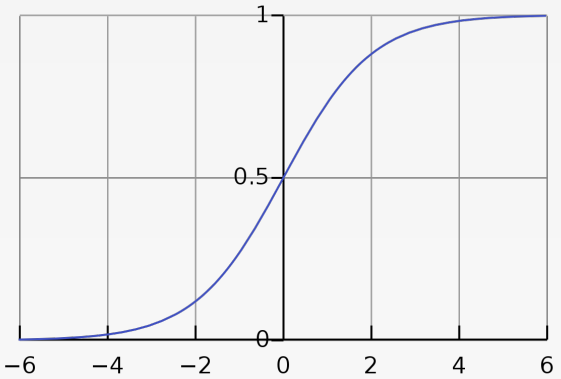


Logistic Regression

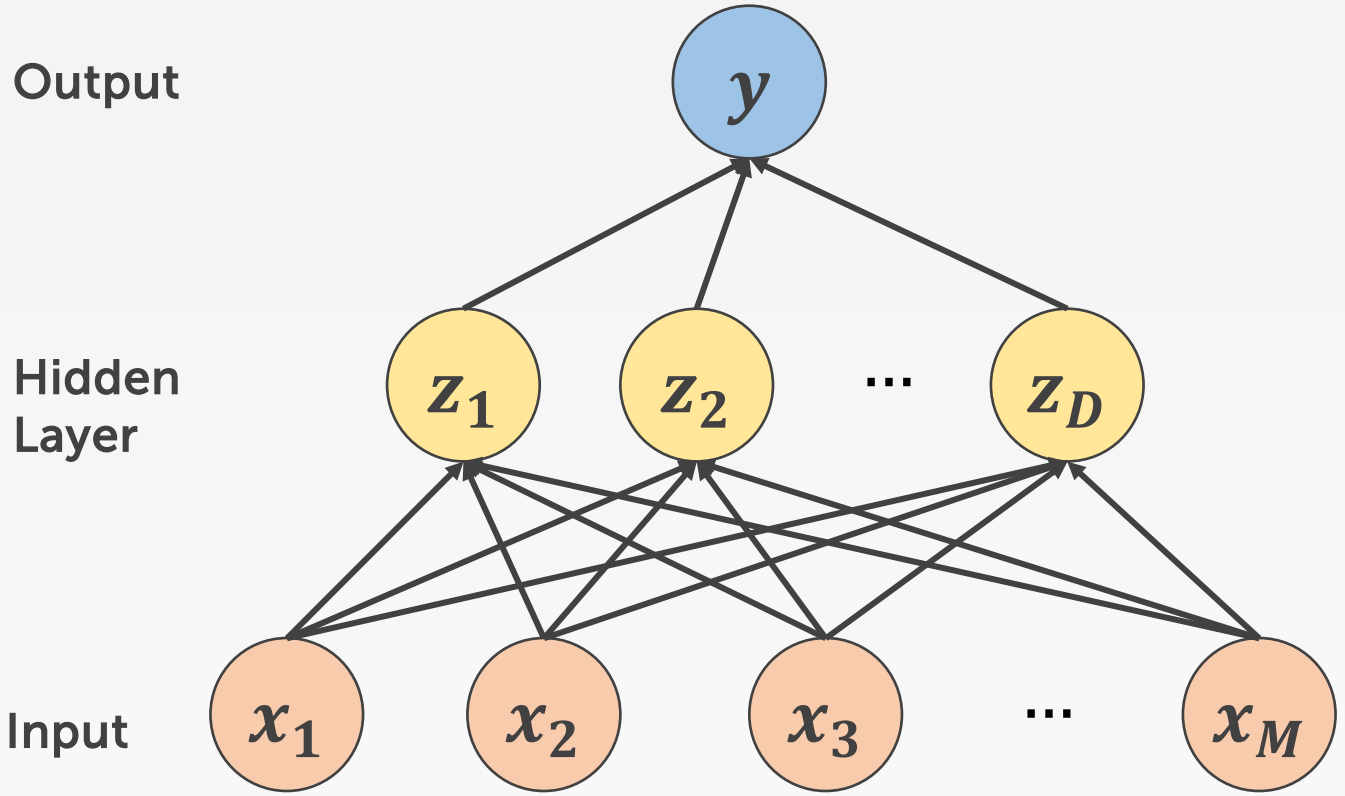
$$y = h_{\boldsymbol{\theta}}(x) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \frac{1}{1+\exp(-a)}$

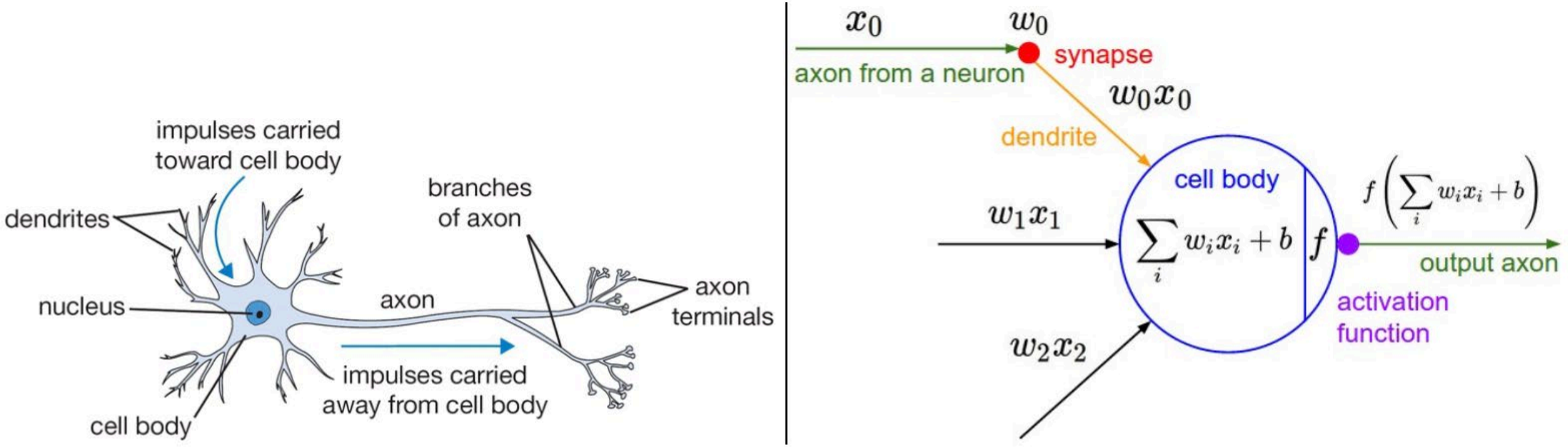




Neural Network

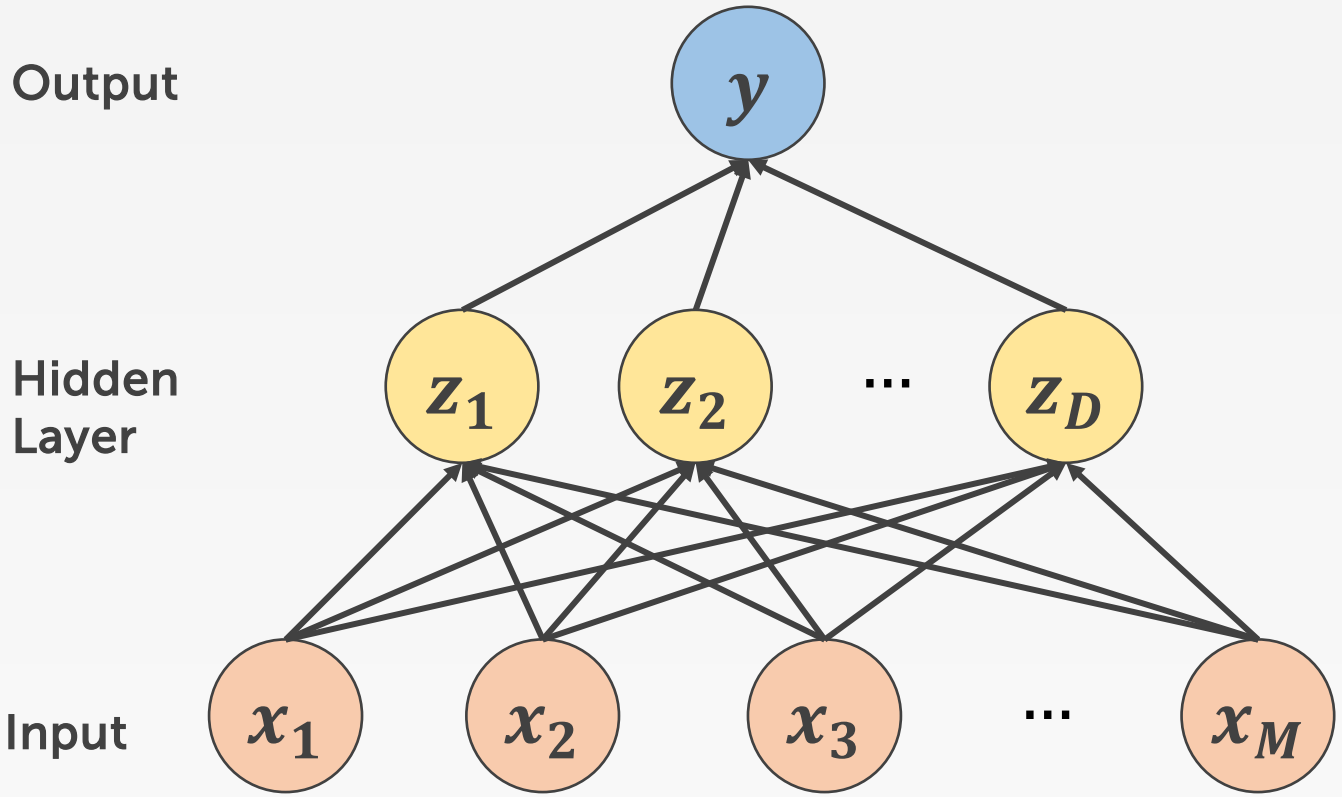


Inspiration: The Brain



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

Neural Network



(E) Output (sigmoid)

$$y = \frac{1}{1+\exp(-b)}$$

(D) Output (linear)

$$b = \sum_{j=1}^{D} \beta_j z_j$$

(C) Hidden (sigmoid)

$$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$$

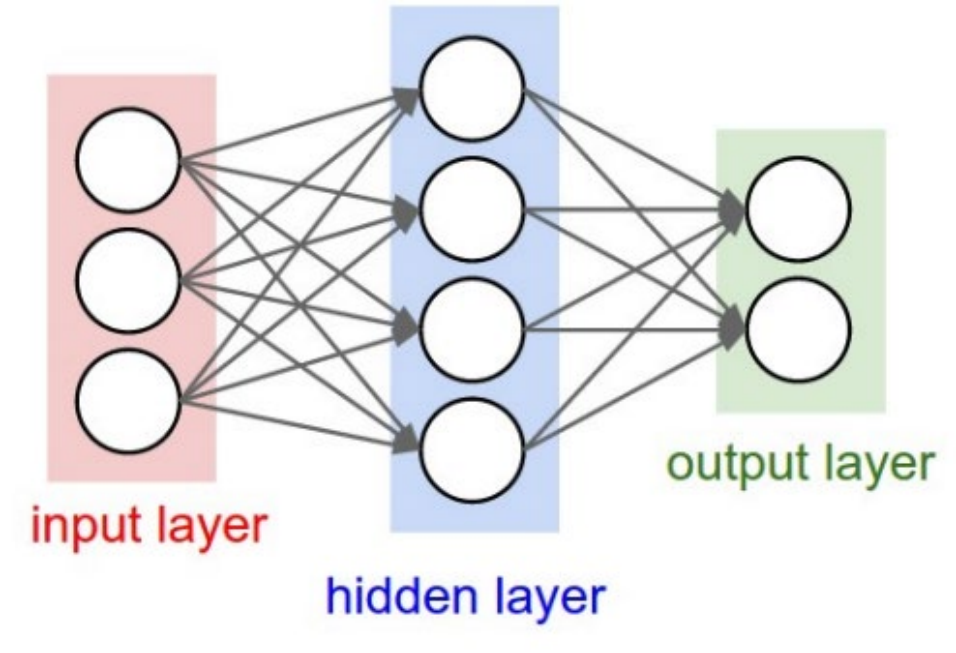
(B) Hidden (linear)

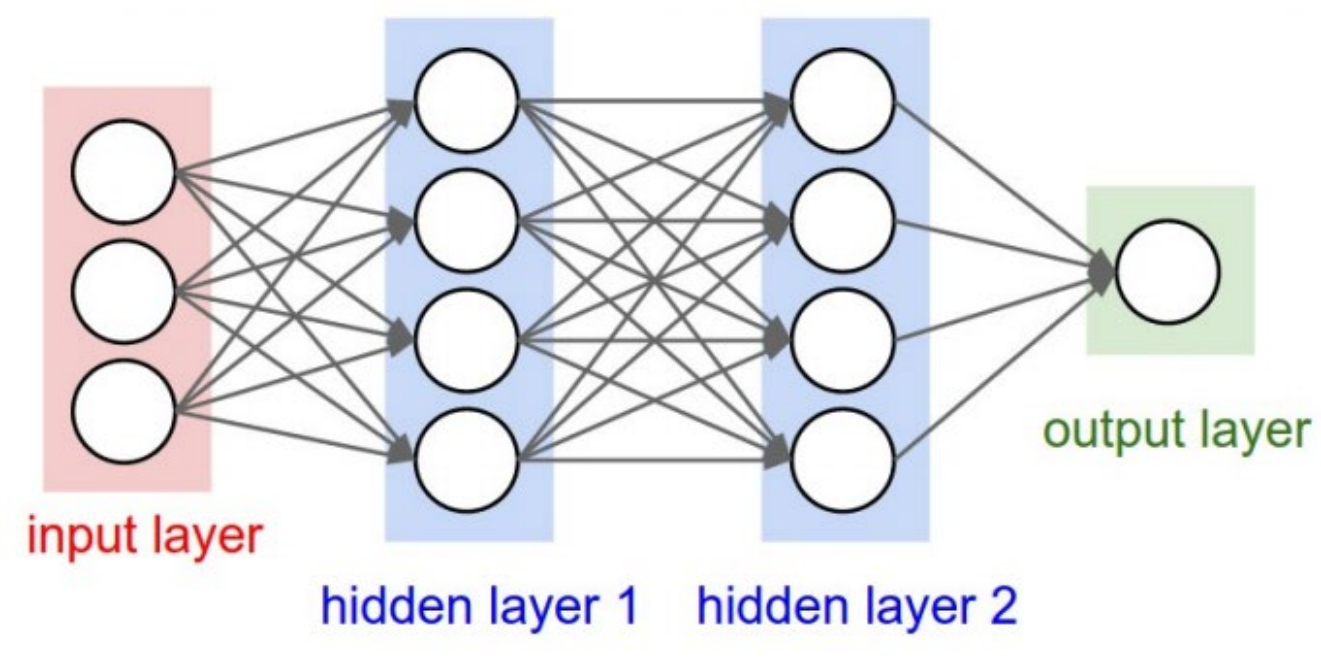
$$a_j = \sum_{i=1}^{M} \alpha_{ji} x_i, \forall j$$

(A) Input

Given $x_i, \forall i$

Multi-Layer Neural Network

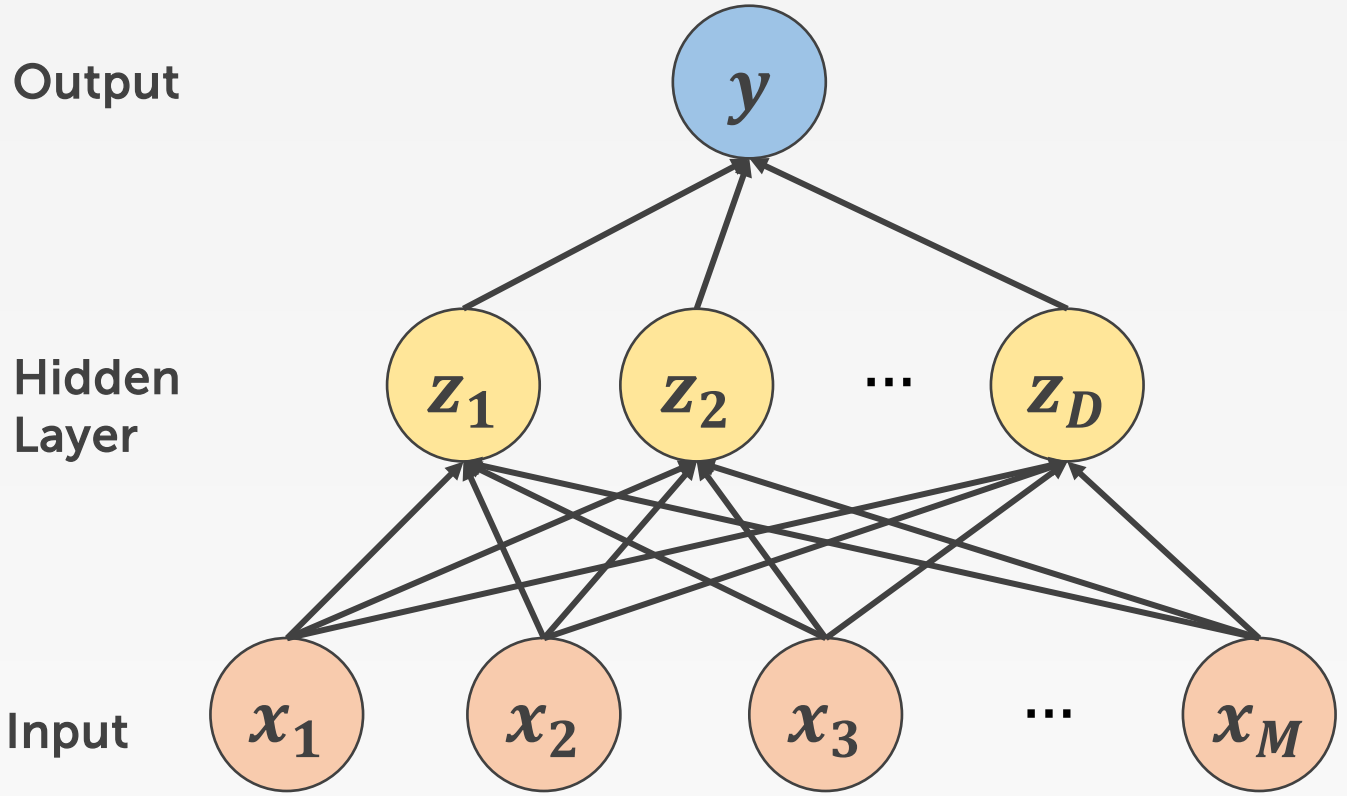




Architecture

- # of hidden layers (depth)
- # of units per hidden layer (width)
- Type of activation function (nonlinearity)
- Form of objective function

How many hidden units?



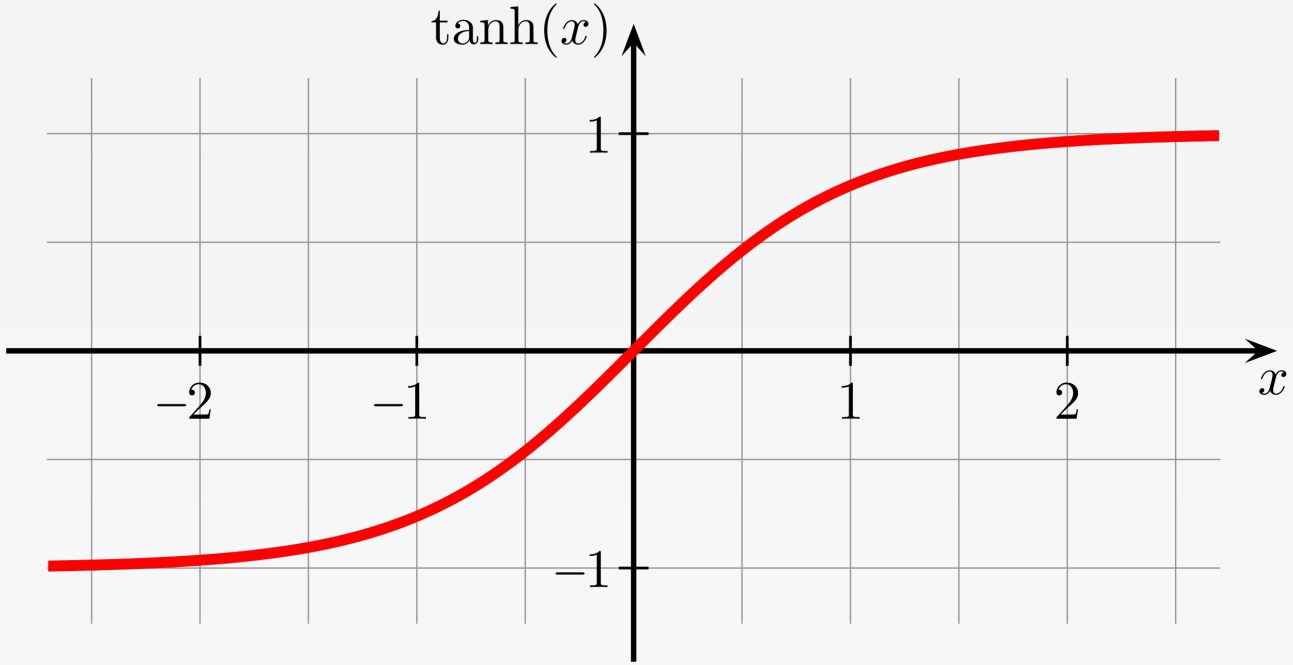
D=M?
D<M?
D>M?

How many layers should we use?

- Theoretical answer:
  - Cybenko (1989): For any continuous function $g(x)$, there exists a 1-hidden layer neural network $h_\theta(x)$
    s.t., $|h_\theta(x) - g(x)| < \epsilon$ for all x, assuming sigmoid activation functions.
- Empirical answer:
  - Before 2006: “Depp networks (e.g., 3 or more hidden layers) are too hard to train”
  - After 2006: “Deep networks are easier to train than shallow networks for many problems”

From Matt Gormley @CMU

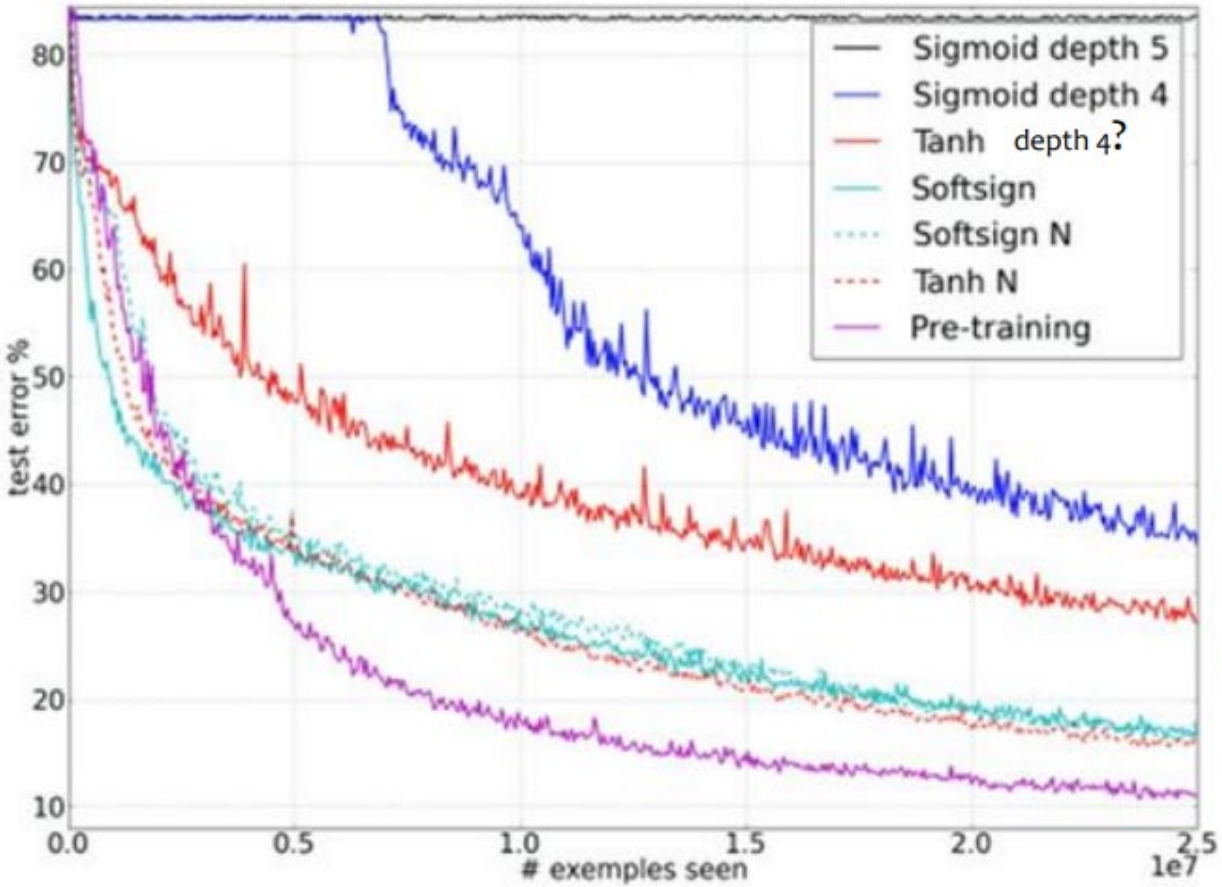
Alternative 1: tanh



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

Understanding the difficulty of training deep feedforward neural networks

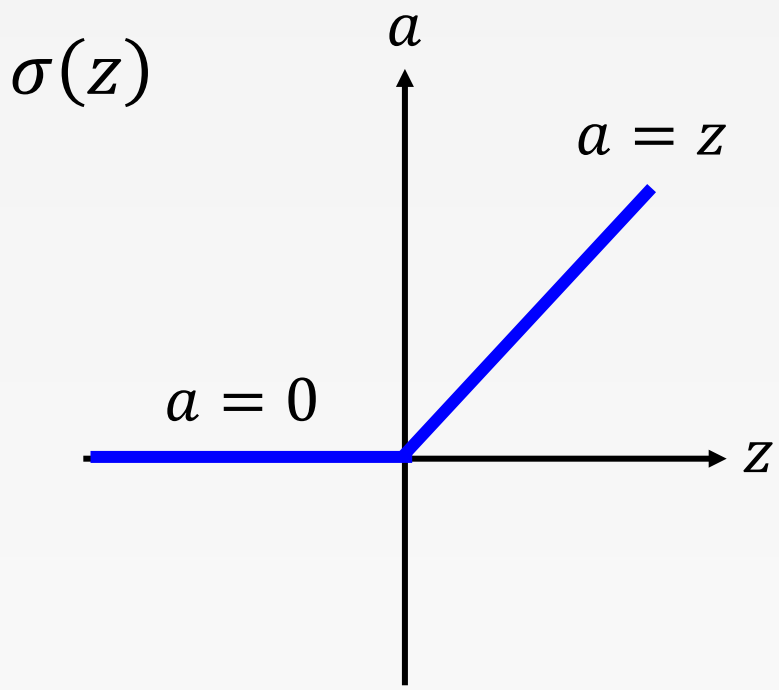
AI Stats 2010



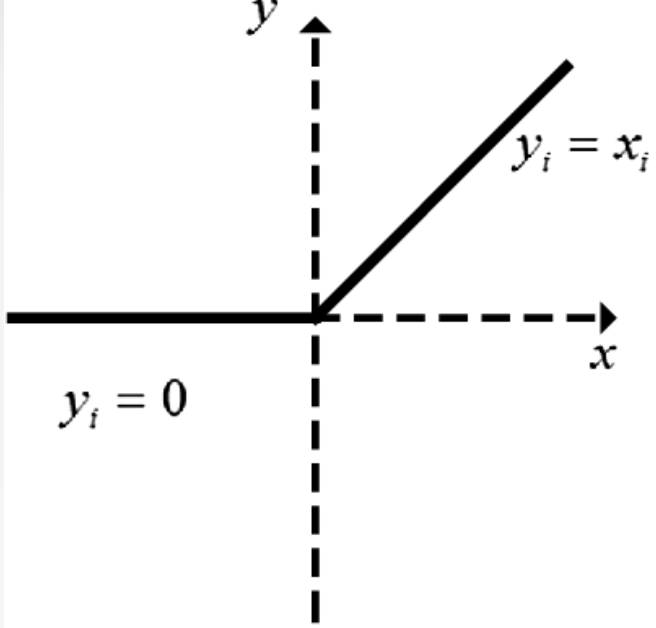
sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

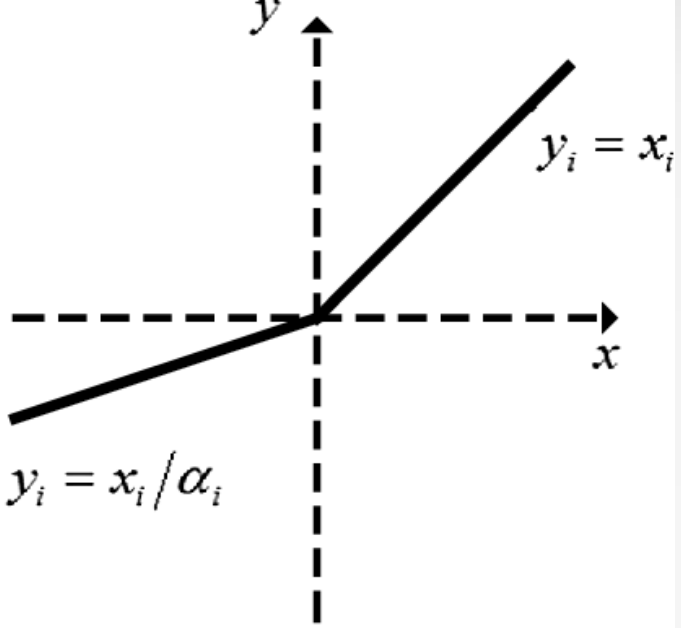
Alternative 2: ReLU



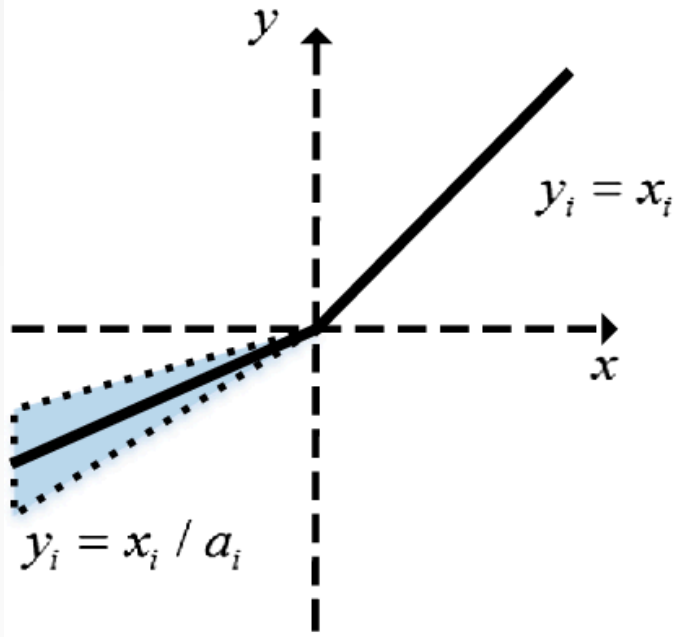
- Rectified Linear Unit (ReLU)



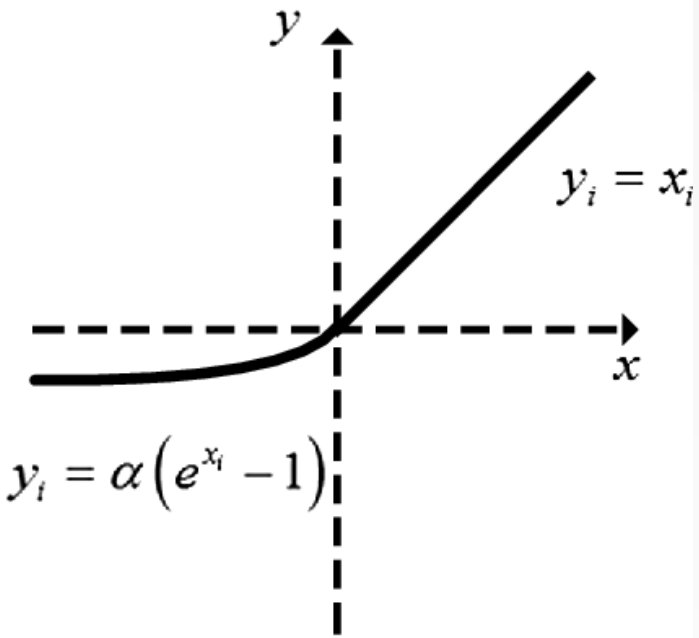
(a) ReLU



(b) LReLU/PReLU

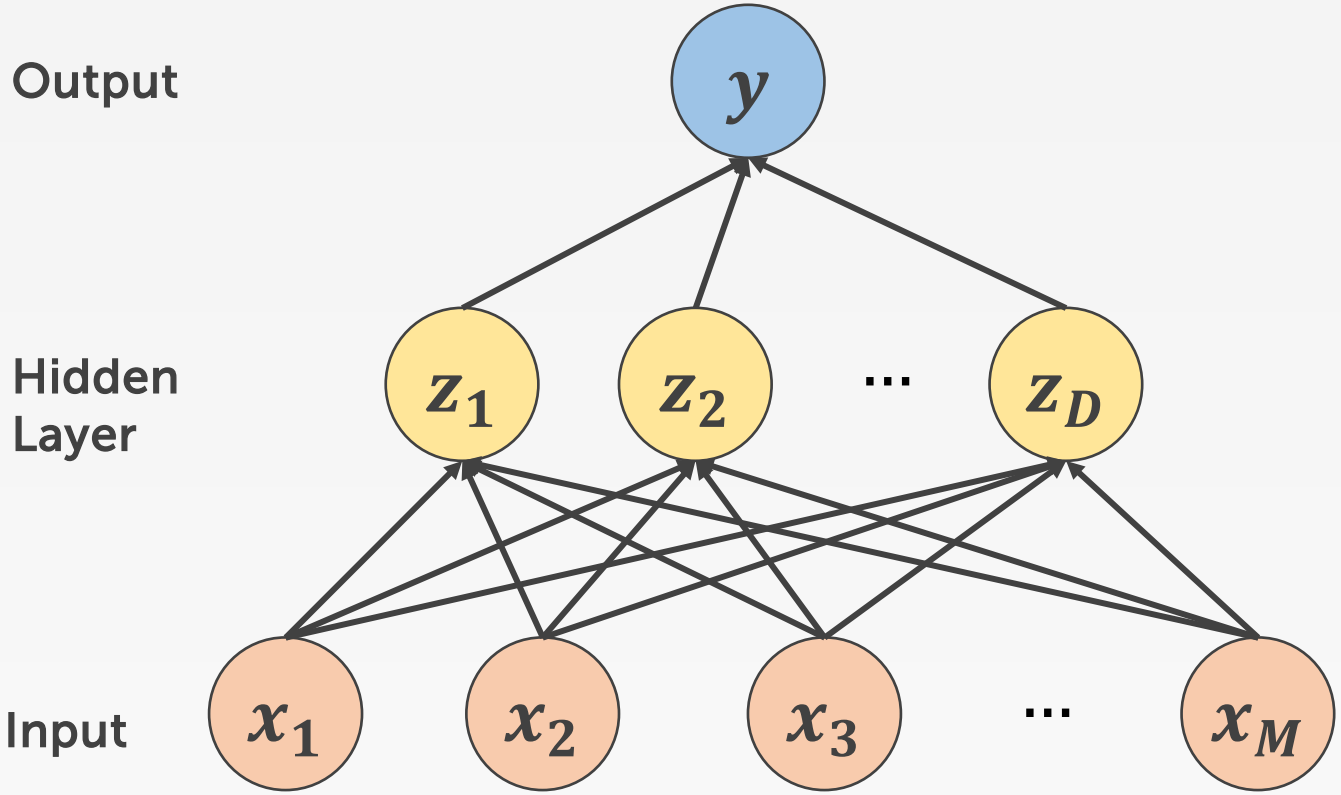


(c) RReLU



(d) ELU

Neural Network for Regression



(D) Output (linear)

$b = \sum_{j=1}^{D} \beta_j z_j$

(C) Hidden (sigmoid)

$z_j = \frac{1}{1+\exp(-a_j)}, \forall j$

(B) Hidden (linear)

$a_j = \sum_{i=1}^{M} \alpha_{ji} x_i, \forall j$

(A) Input

Given $x_i, \forall i$

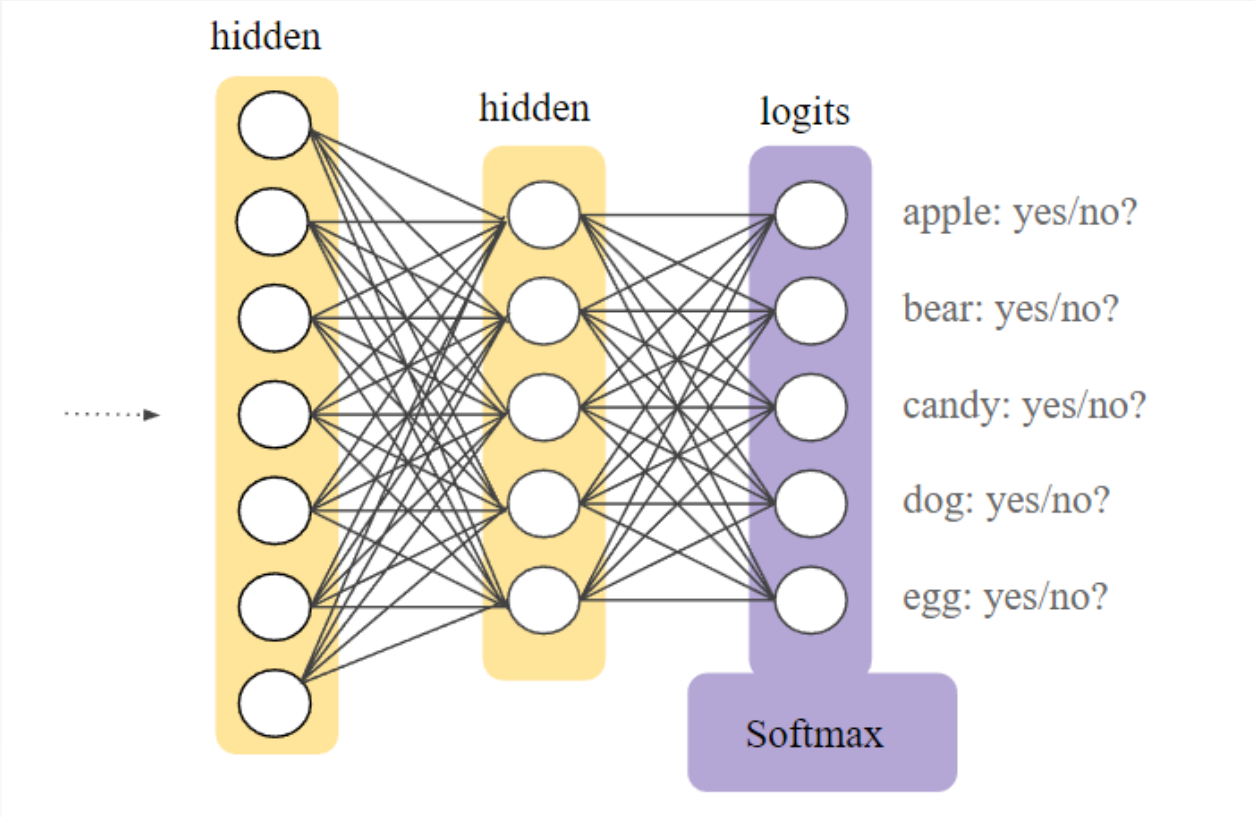
Objective Functions for NNs

- Quadratic Loss:
  - the same objective as Linear Regression
  - i.e. mean squared error
- Cross-Entropy:
  - the same objective as Logistic Regression
  - i.e. negative log likelihood

Forward	Backward
Quadratic: $J = \frac{1}{2}(y - y^*)^2$ Cross Entropy: $J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y - y^*$ $\frac{dJ}{dy} = \mathrm{y}^* \frac{1}{y} + (1 - \mathrm{y}^*) \frac{1}{1 - y}$

Softmax

- Softmax (multinomial choice model)
- $y_1 = e^{z_1} / \sum_j e^{z_j}$



Backpropagation Algorithm

Recall: SGD

- Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

- Loss function: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$, where $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$
- Define goal:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

- Train with SGD:

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Chain Rule

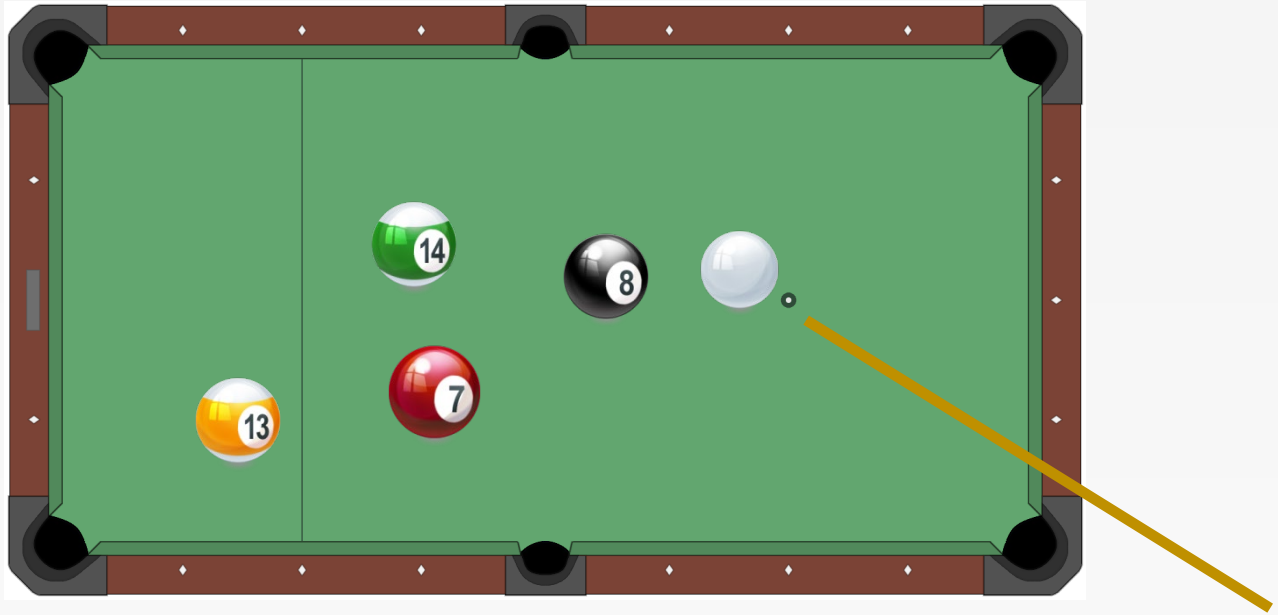
Given: $y = g(u)$ and $u = h(x)$

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \forall i, k$$

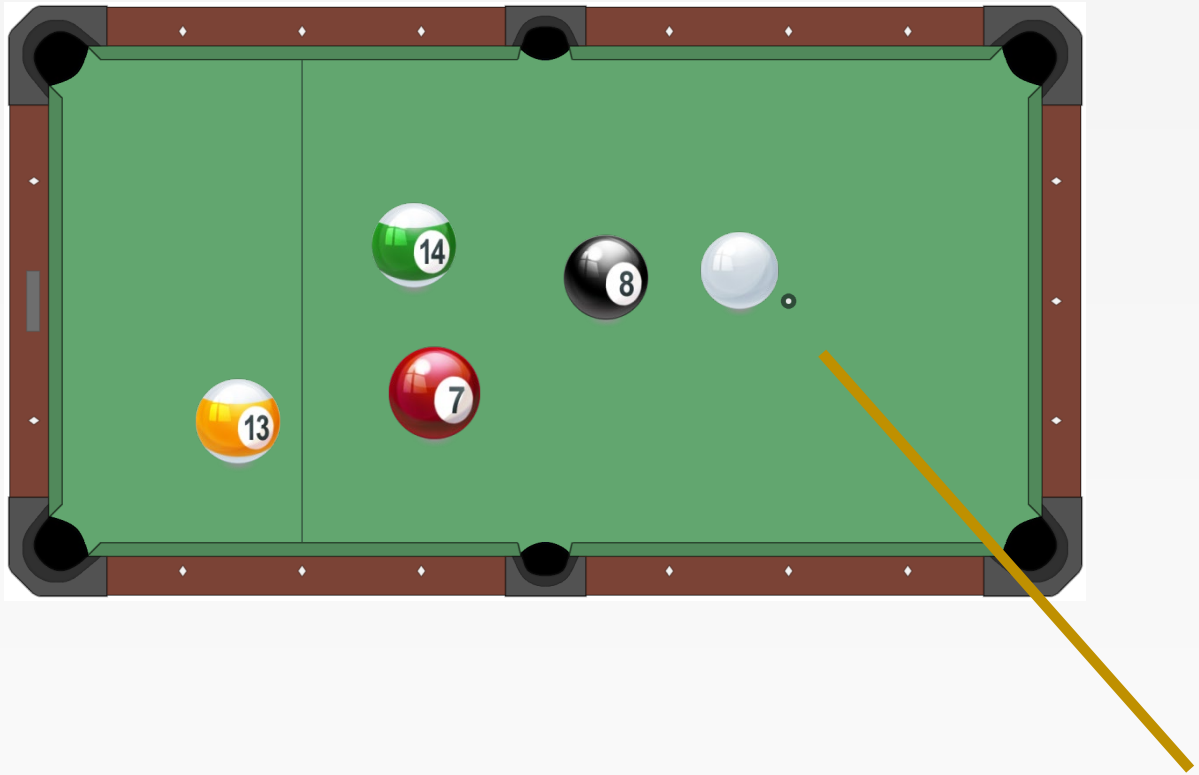
❖Backpropagation is just repeated application of the chain rule from Calculus course.

Error Back-Propagation



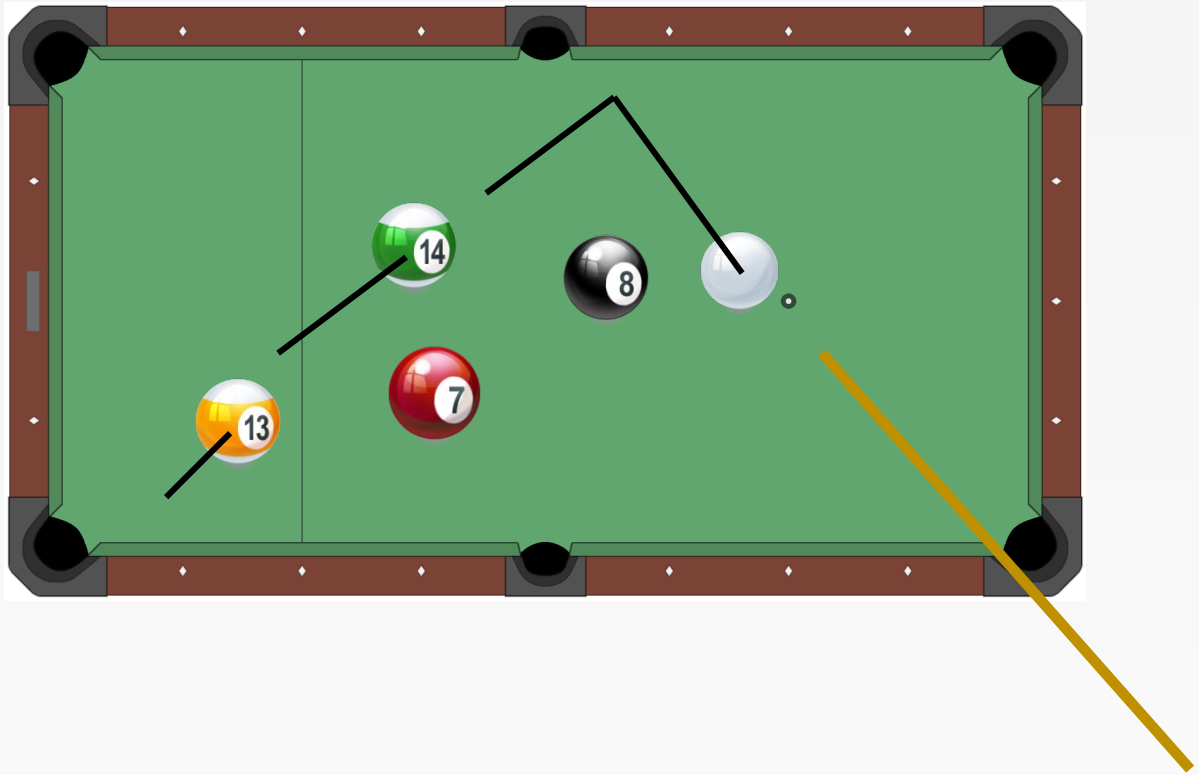
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



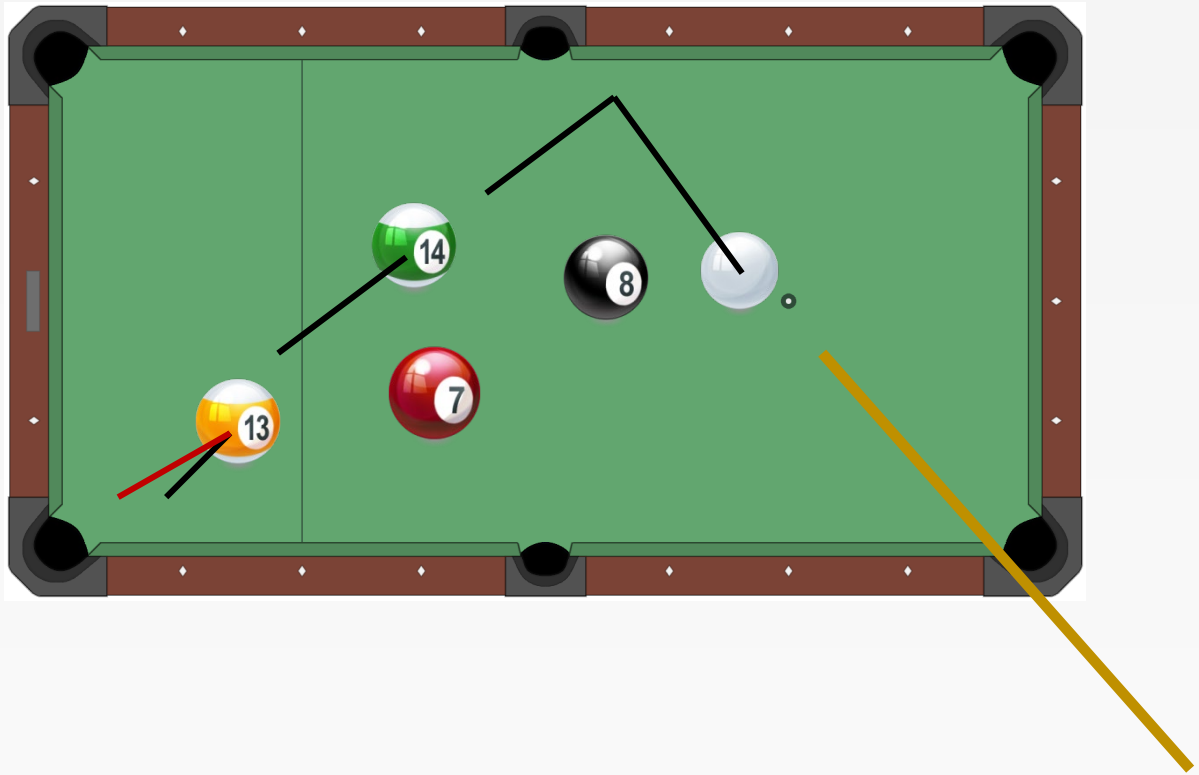
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



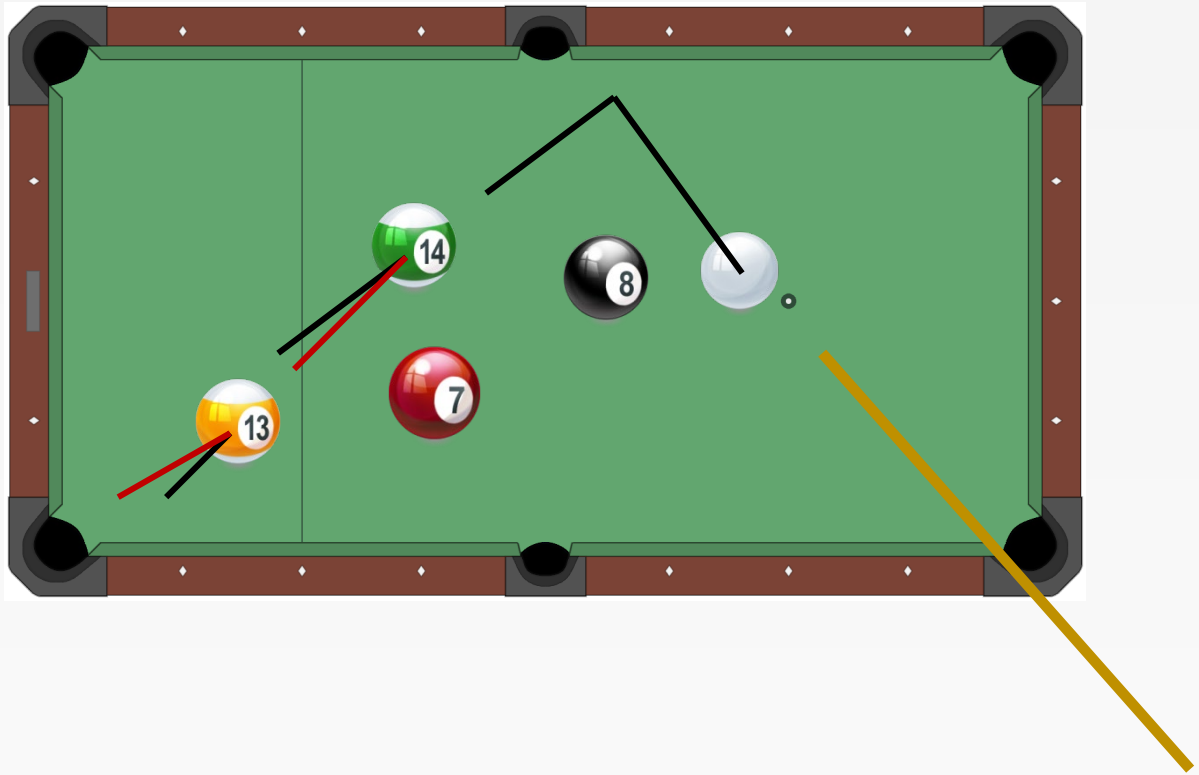
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



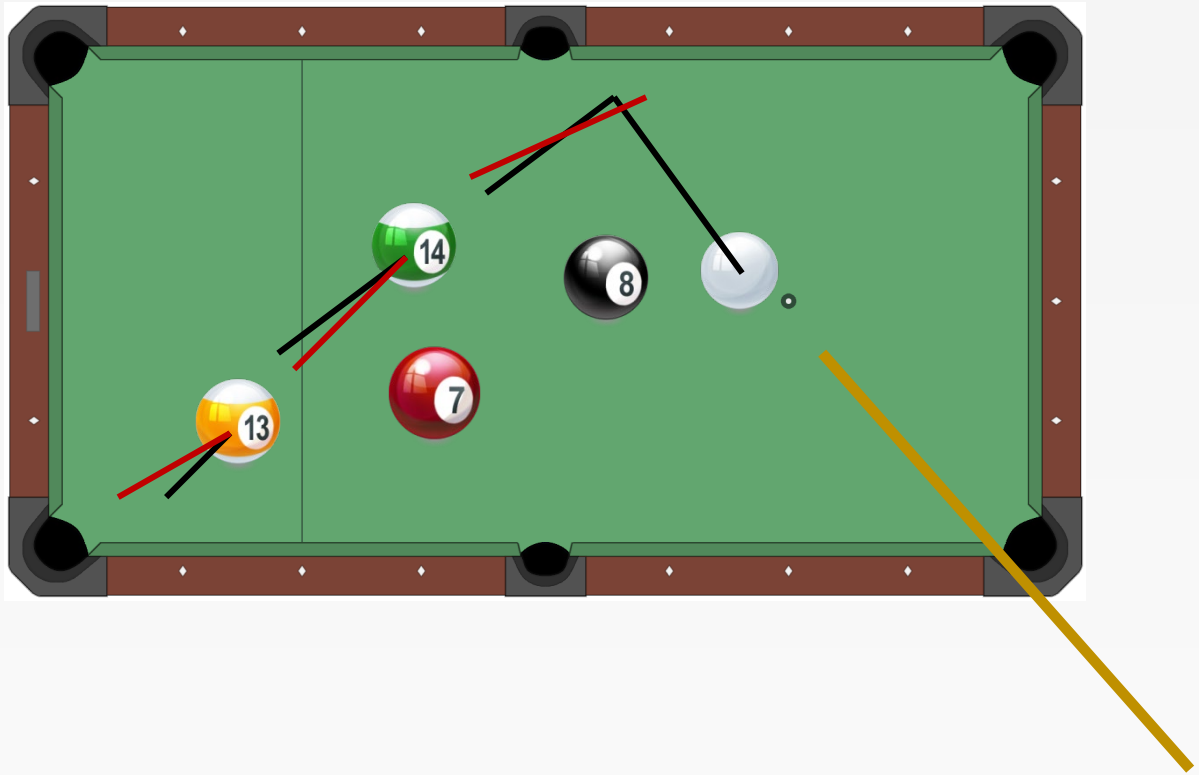
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



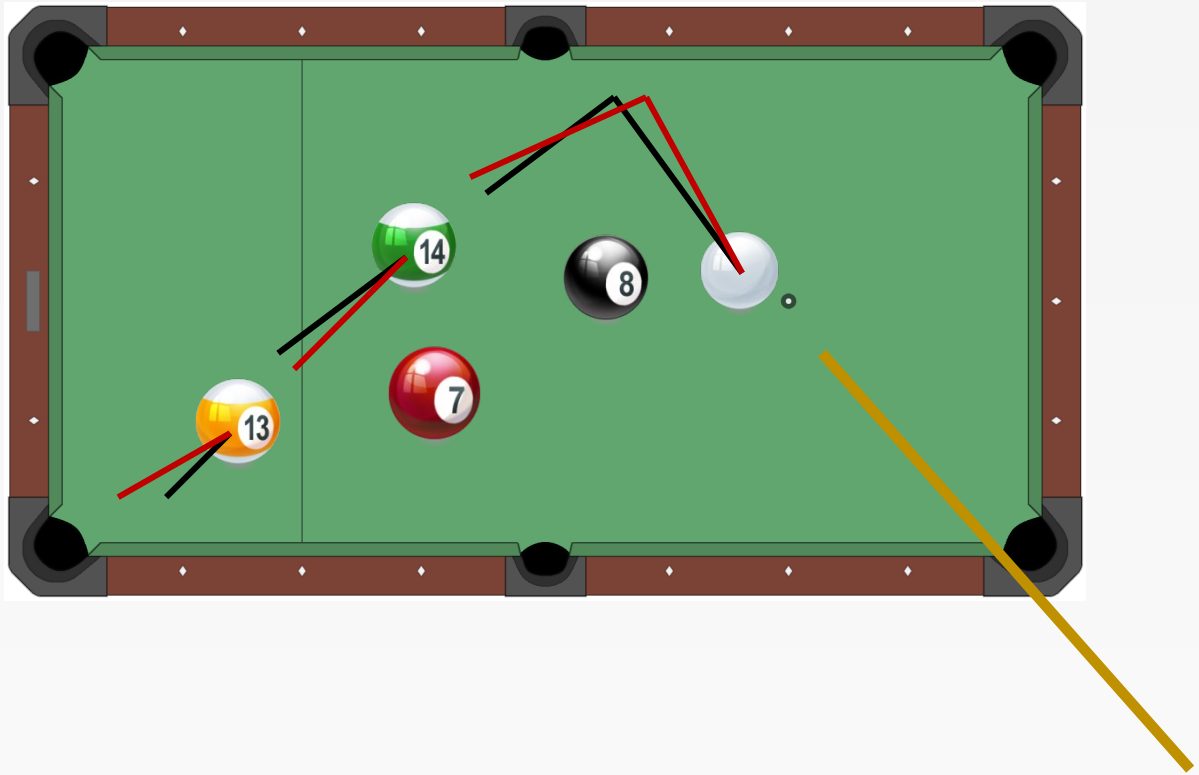
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



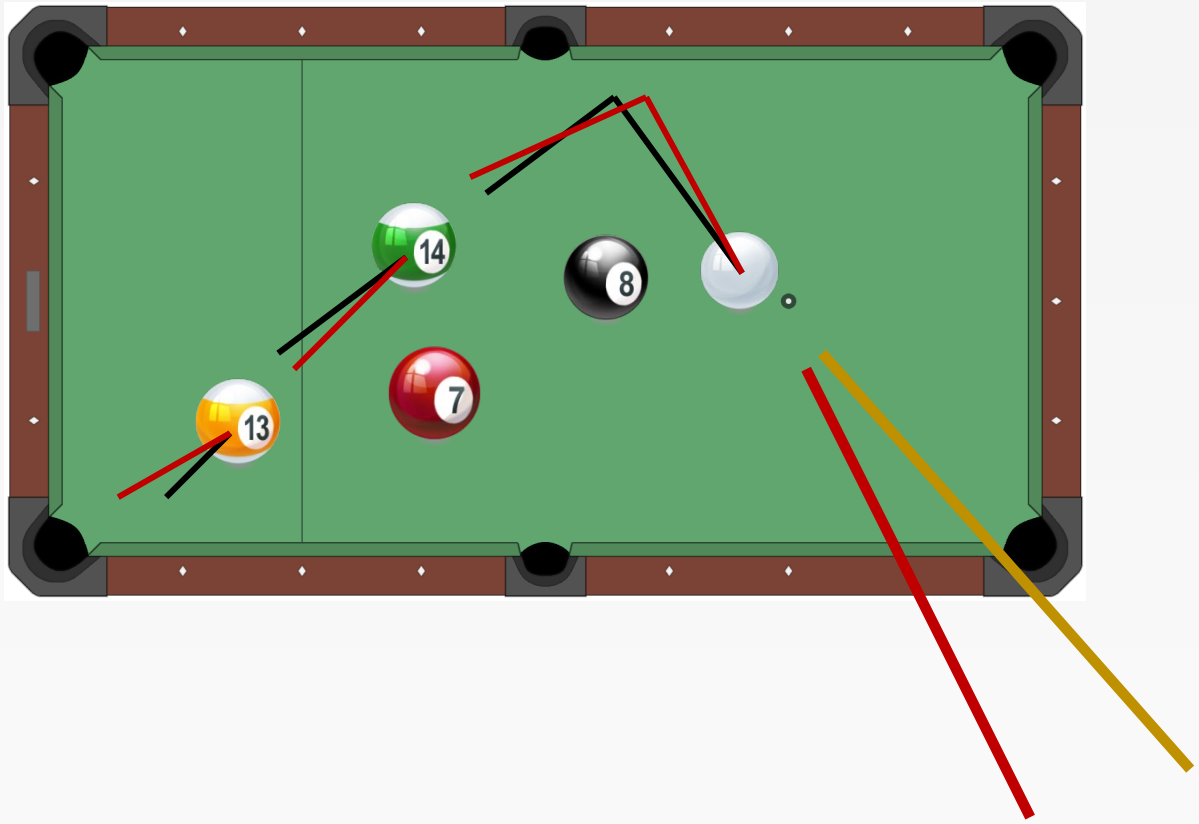
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



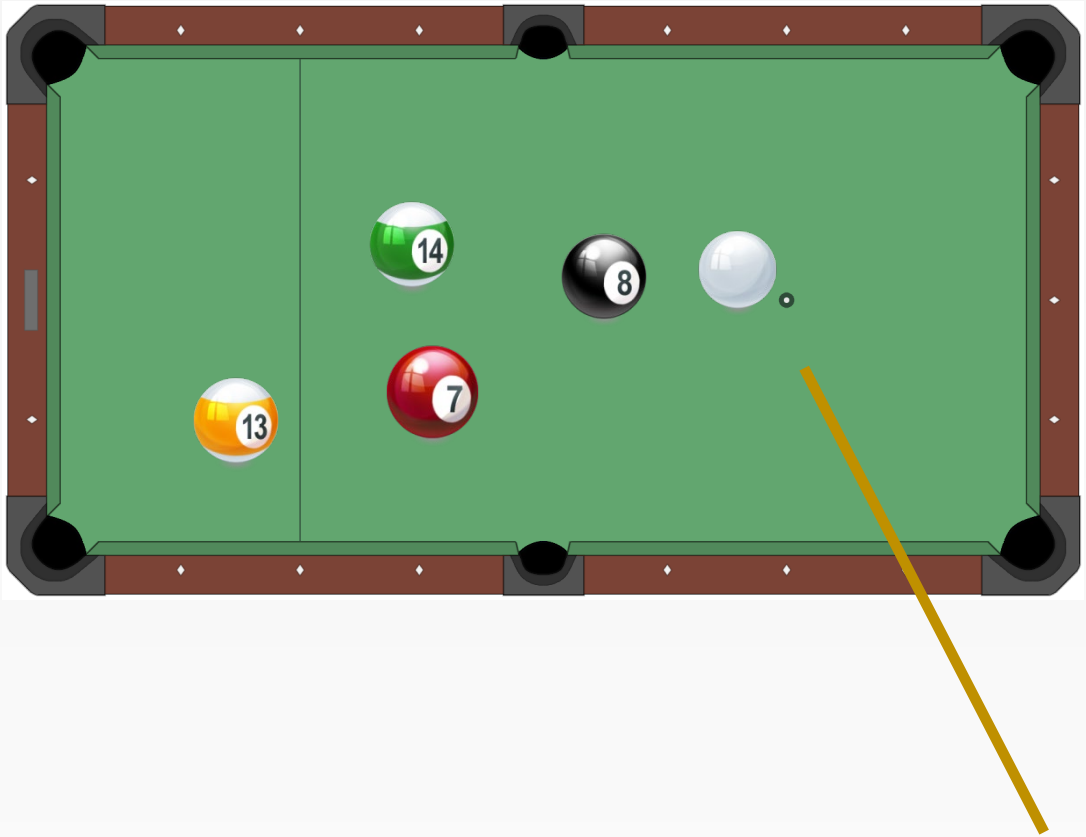
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



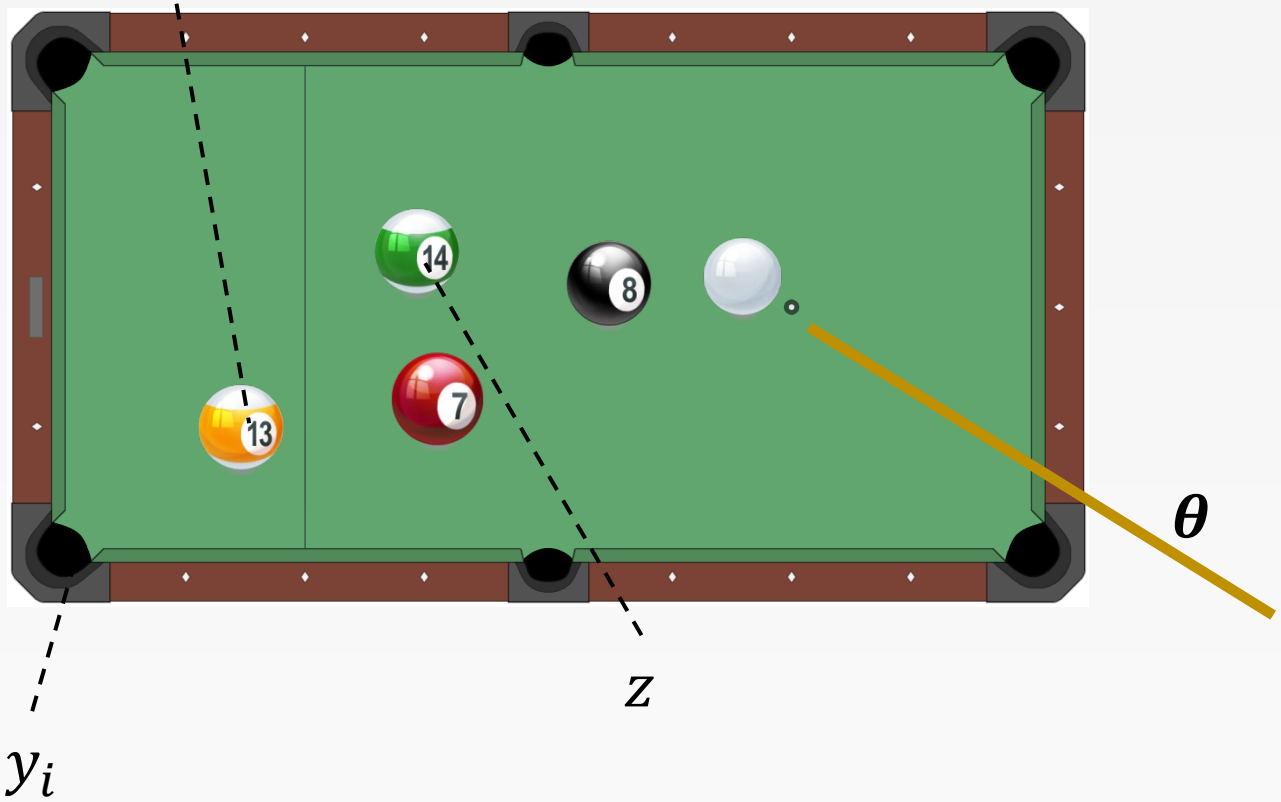
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Simple Idea

For each training instance:

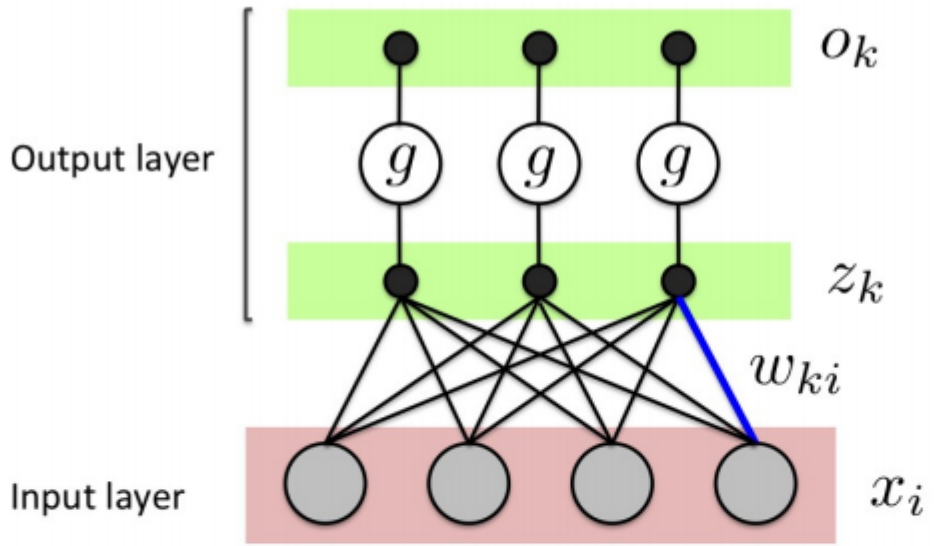
Make a prediction (forward pass)

Measure the error

Go through each layer in reverse to measure the error contribution from each connection (reverse pass)

Slightly tweaks the connection weights to reduce error (Gradient Descent)

Compute Gradient



- Error gradients for single layer network:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial z_k} \frac{\partial z_k}{\partial w_{ki}}$$

- Error gradient is computable for any continuous activation function $g()$, and any continuous error function

An Example

	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$

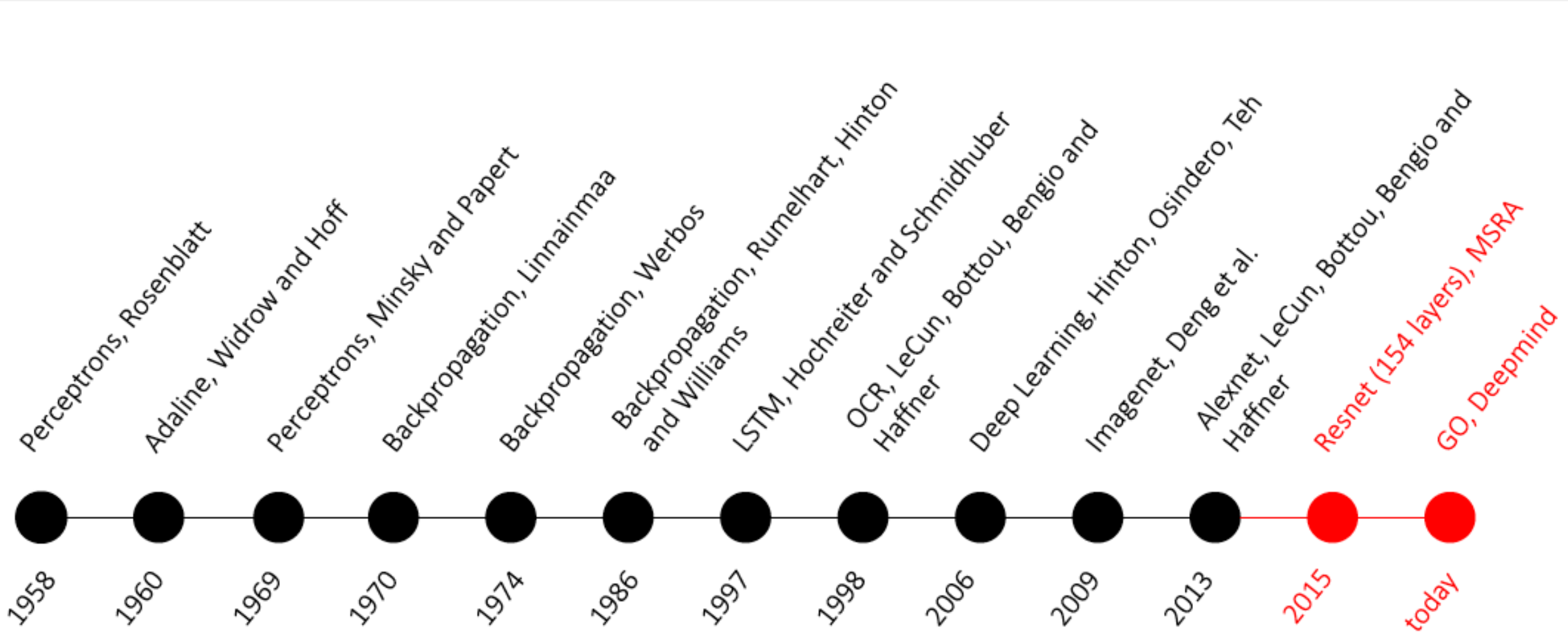
Slide from Matt Gormley @ CMU

Backpropagation

- Training neural nets:
- Loop until convergence:
  - for each example n
    - Given input, propagate activity forward ($x \rightarrow h \rightarrow \ell$) (**forward** pass)
    - Propagate gradients backward (**backward** pass)
    - Update each weight (via **gradient** descent)

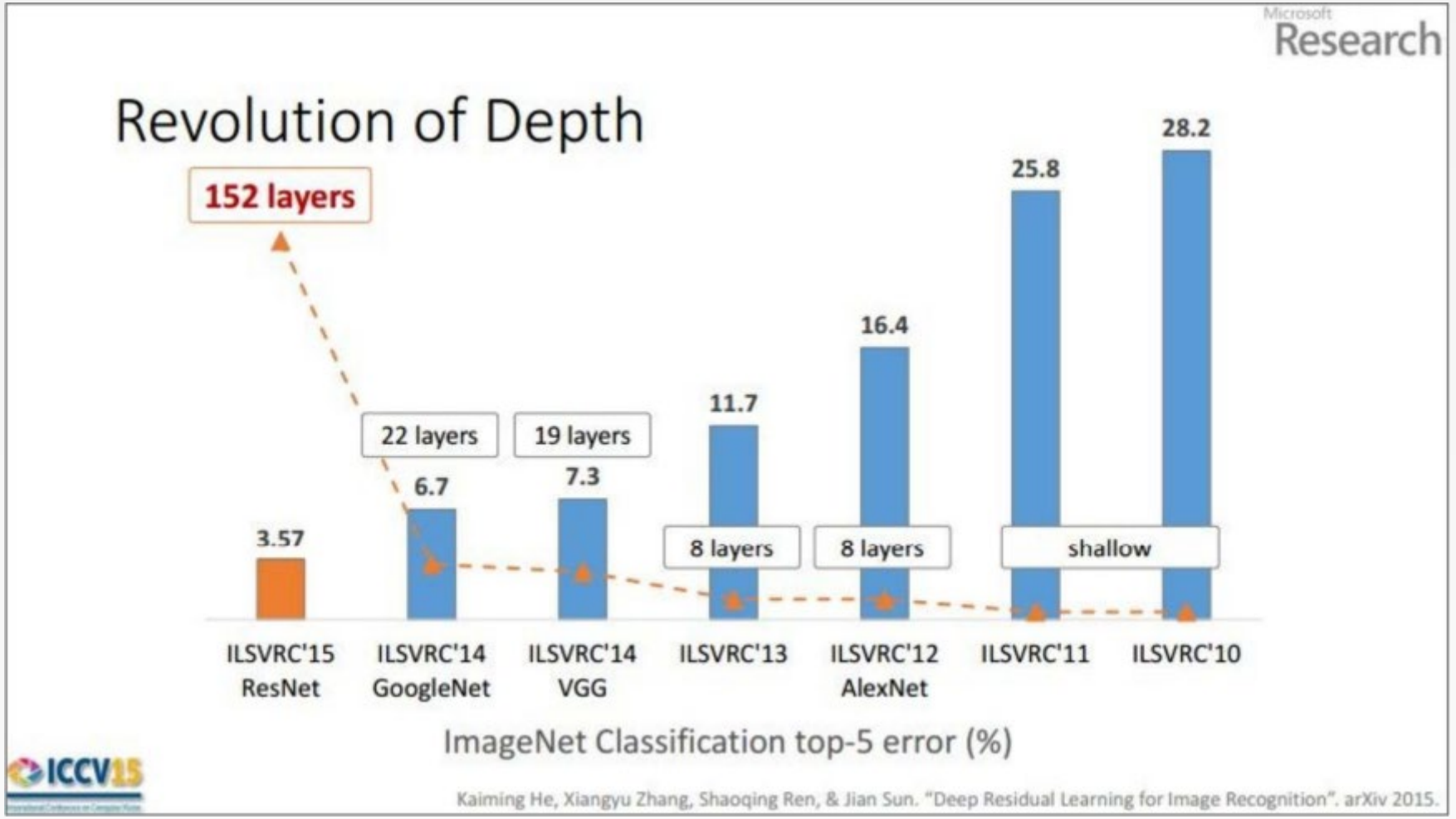
Deep Learning

Why is everyone talking about Deep Learning?

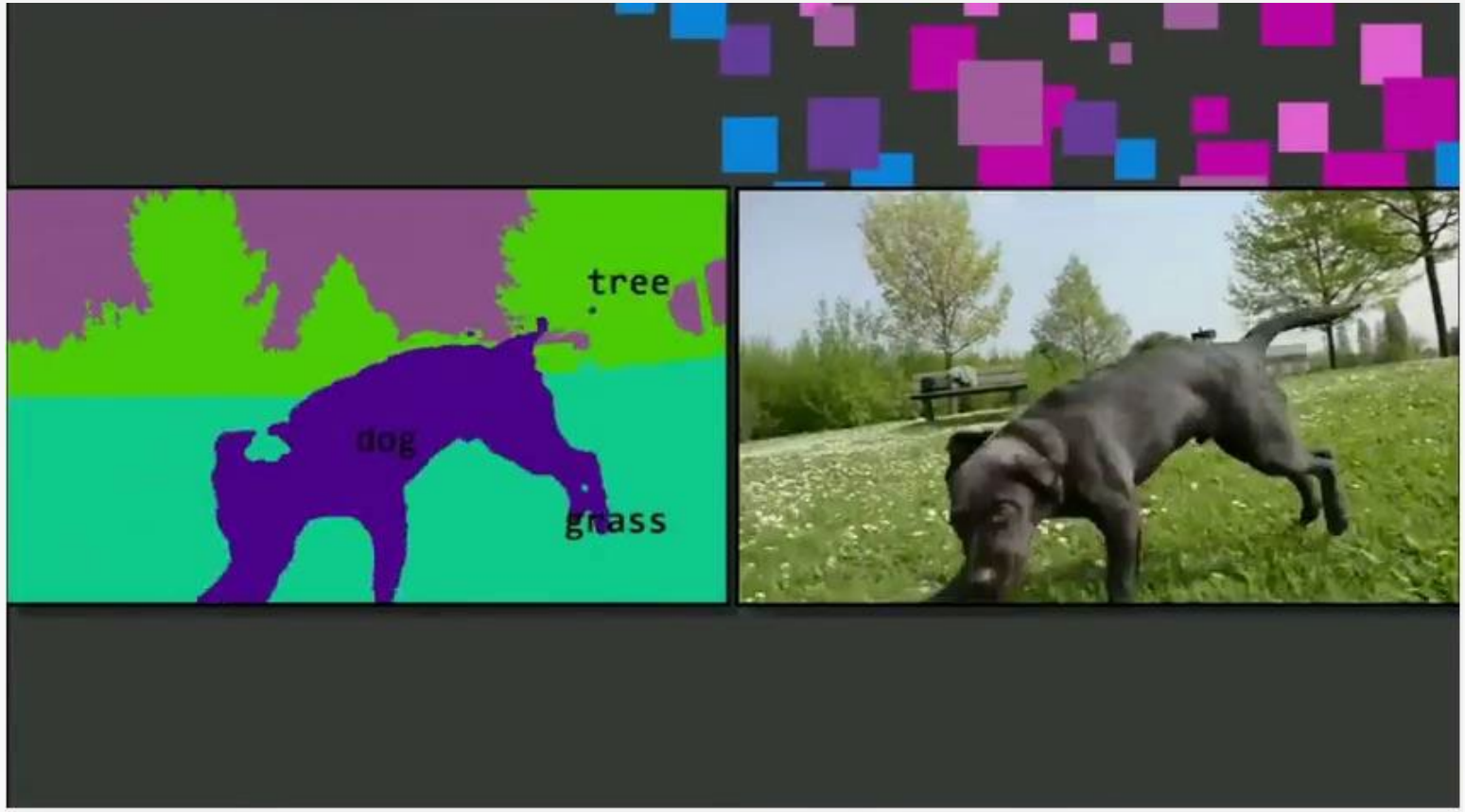


Source: Deep learning @ UVA

CNN for Image Recognition



Object Detection



Microsoft Deep Learning Semantic Image Segmentation

Captioning

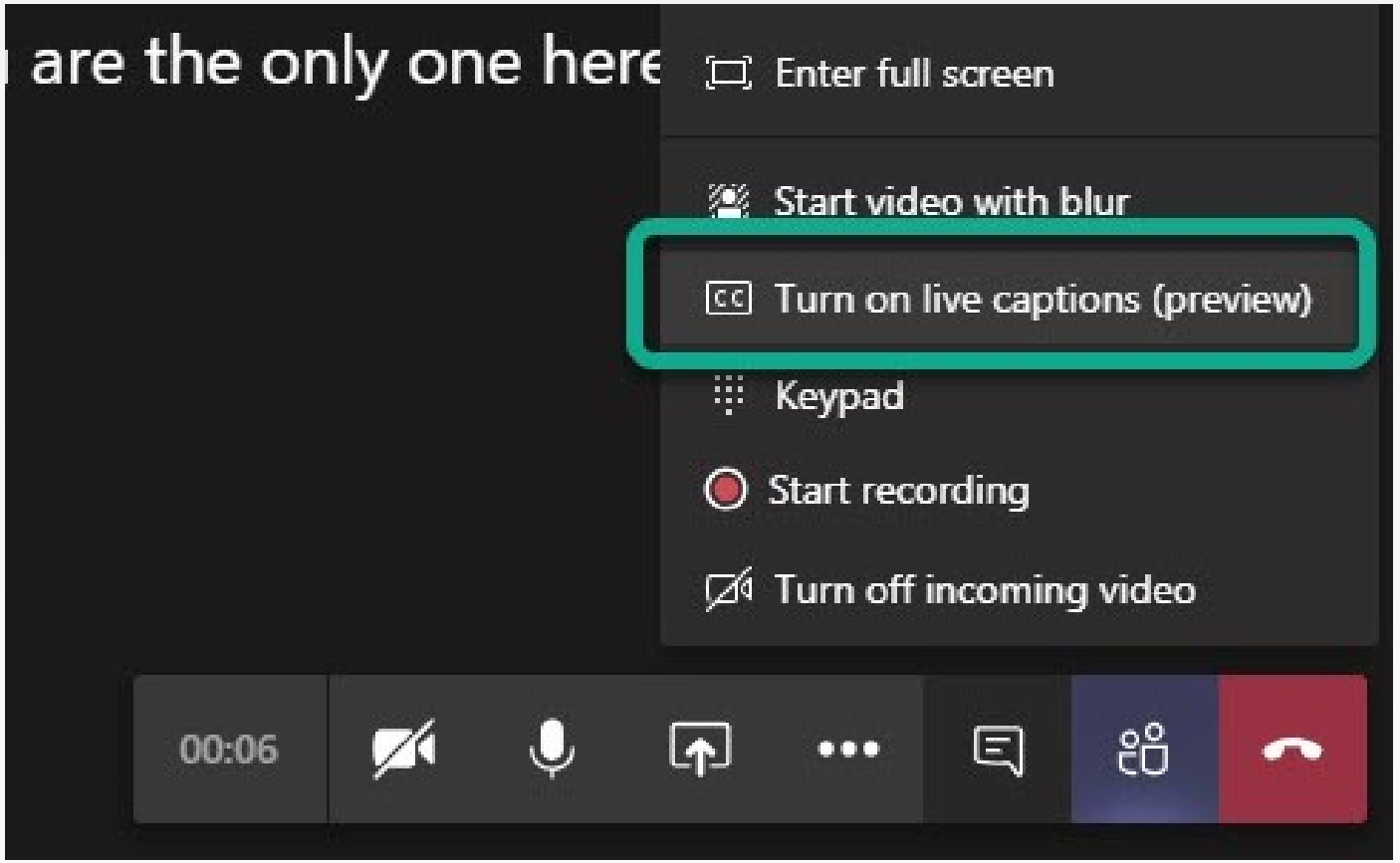

are the only one here
Enter full screen
Start video with blur
Turn on live captions (preview)
Keypad
Start recording
Turn off incoming video
00:06

Image Captioning

"man in black shirt is playing guitar."

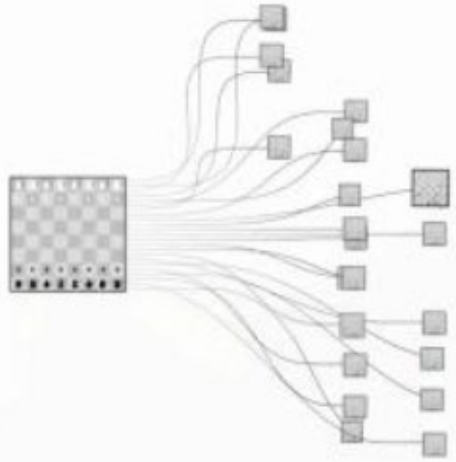
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

Self-driving Cars

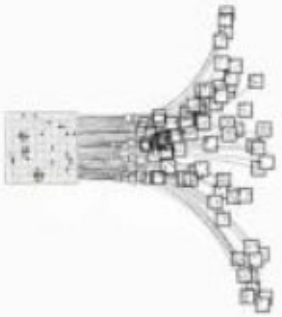
Self Driving Cars HD

Game AI



Chess: 10^{47}

Deep Blue, Feb 10, 1996



Go: 10^{170}

AlphaGo, March, 2016

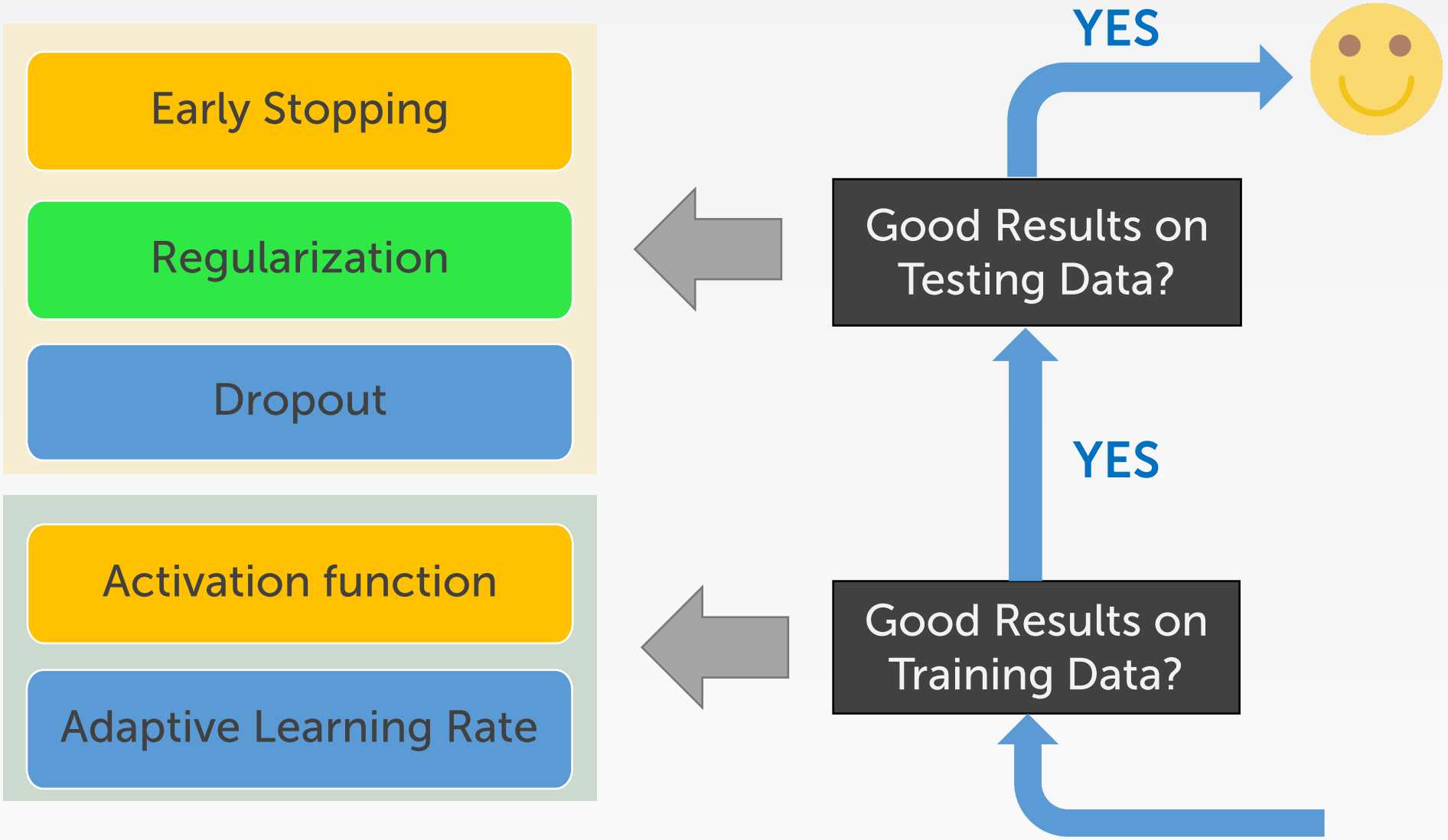
Let's see how it works!

Comparison

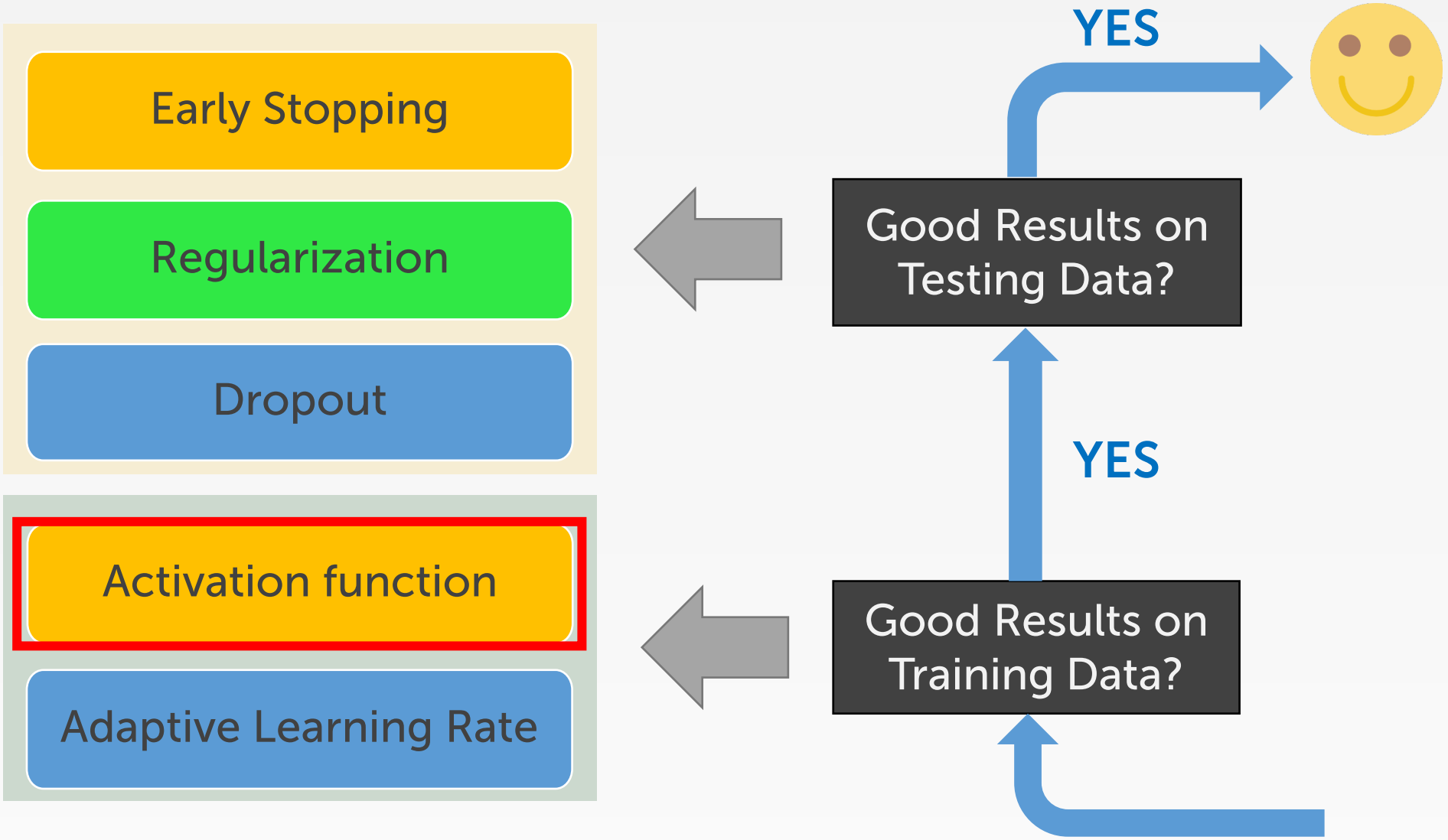
- Traditional pattern recognition

- End-to-end learning → Features are also learned from data

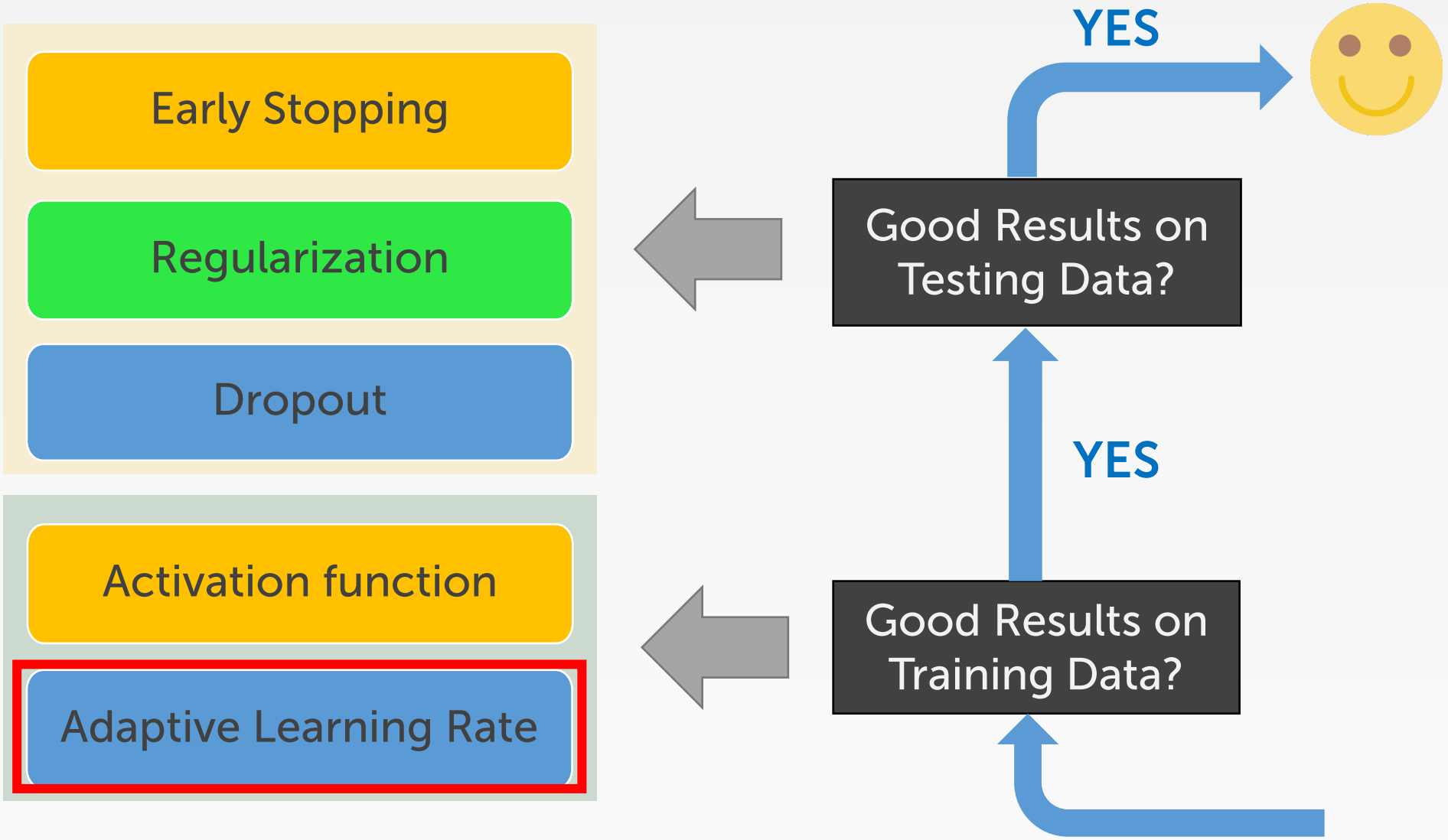
Recipe of Deep Learning



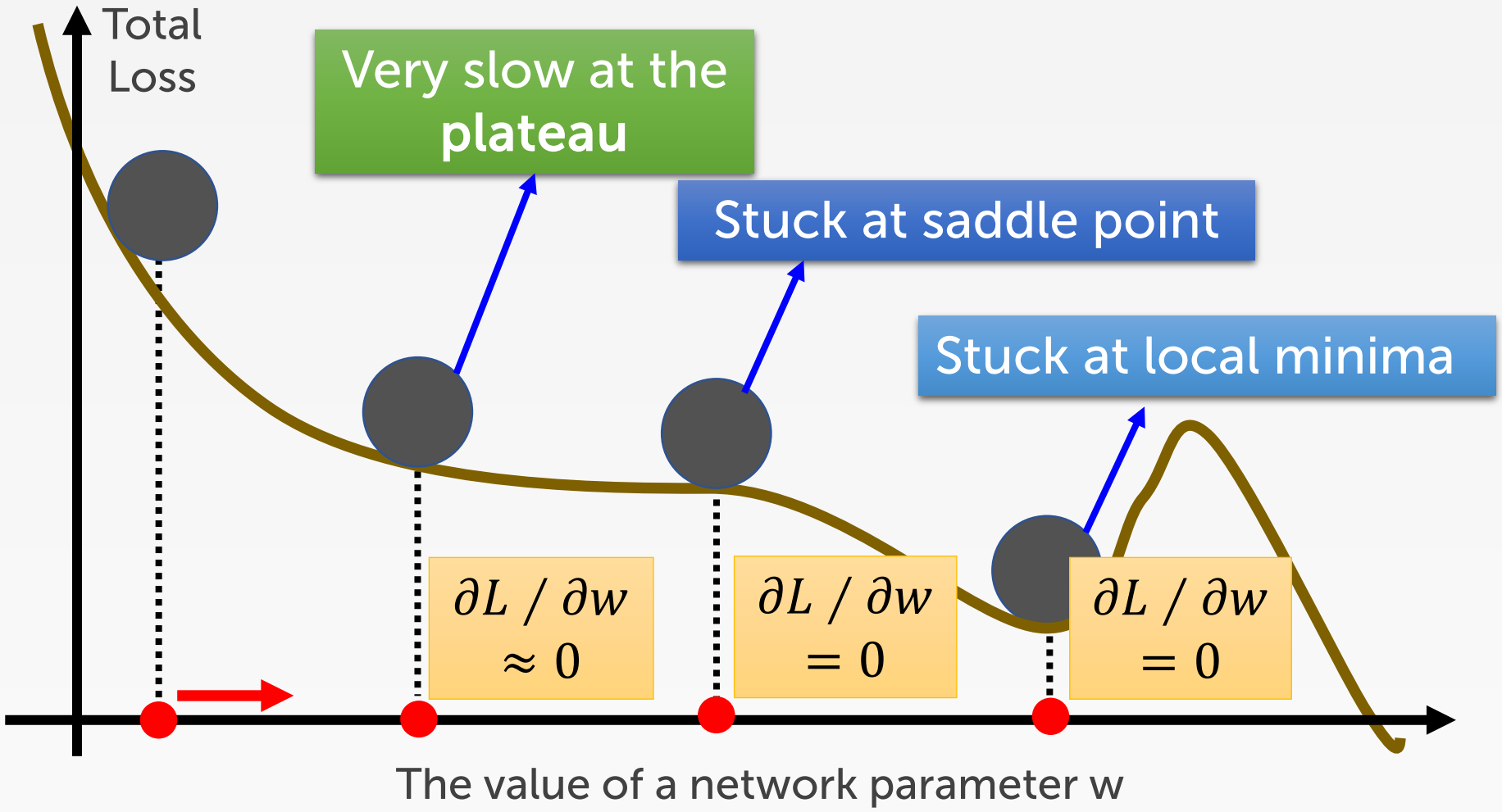
Recipe of Deep Learning



Recipe of Deep Learning



Hard to find optimal network parameters



Learning Rate Scheduling

- Predetermined piecewise constant learning rate

  For example, set the learning rate to η_0= 0.1 at first, then to η_1 = 0.001 after 50 epochs. Although this solution can work very well, it often requires fiddling around to figure out the right learning rates and when to use them.

- Performance scheduling

  Measure the validation error every N steps (just like for early stopping) and reduce the learning rate by a factor of λ when the error stops dropping.

- Exponential scheduling

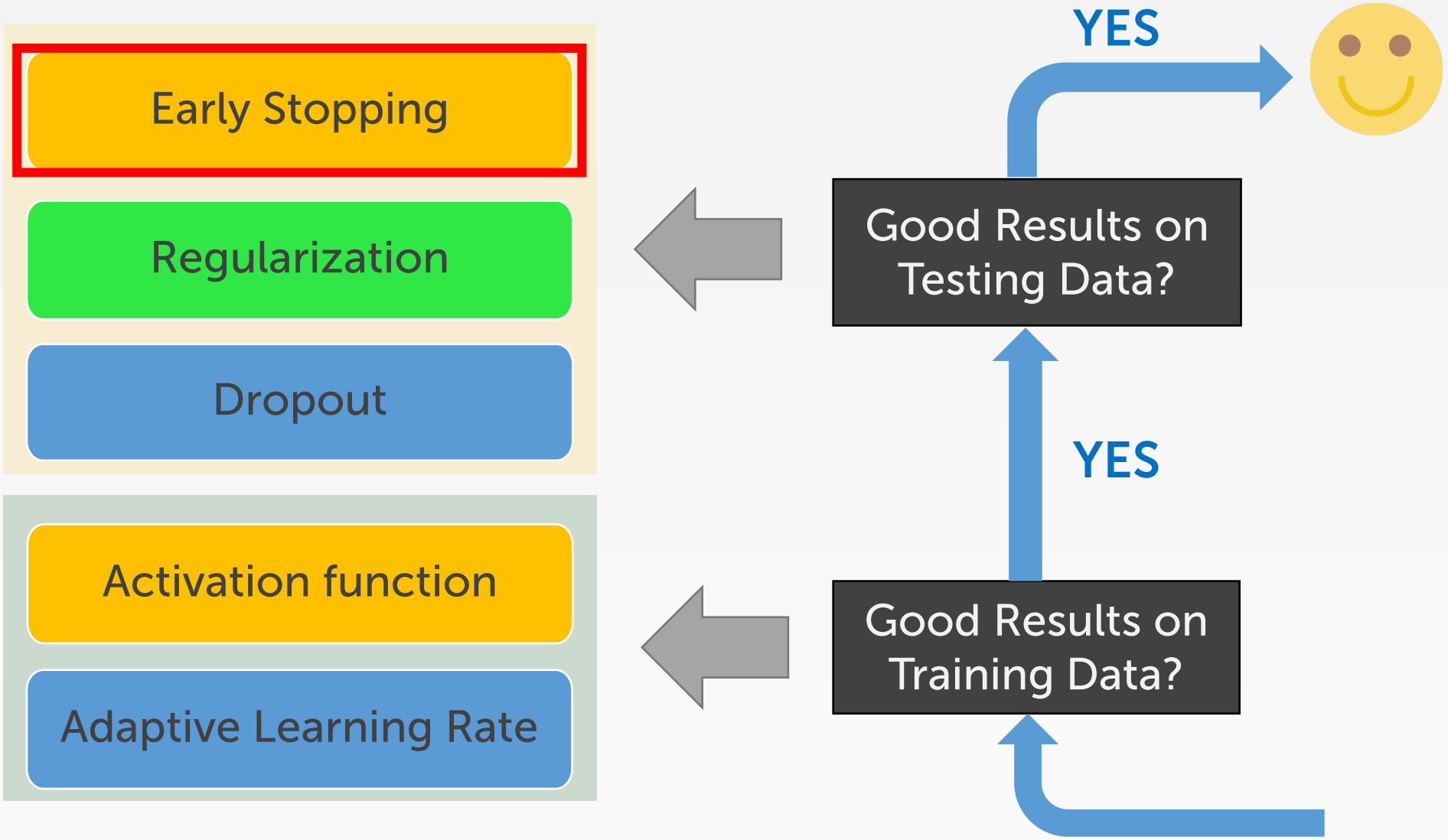
  Set the learning rate to a function of the iteration number t: $\eta(t) = \eta_0 10^{-t/r}$. This works great, but it requires tuning η_0 and r. The learning rate will drop by a factor of 10 every r steps.

- Power scheduling

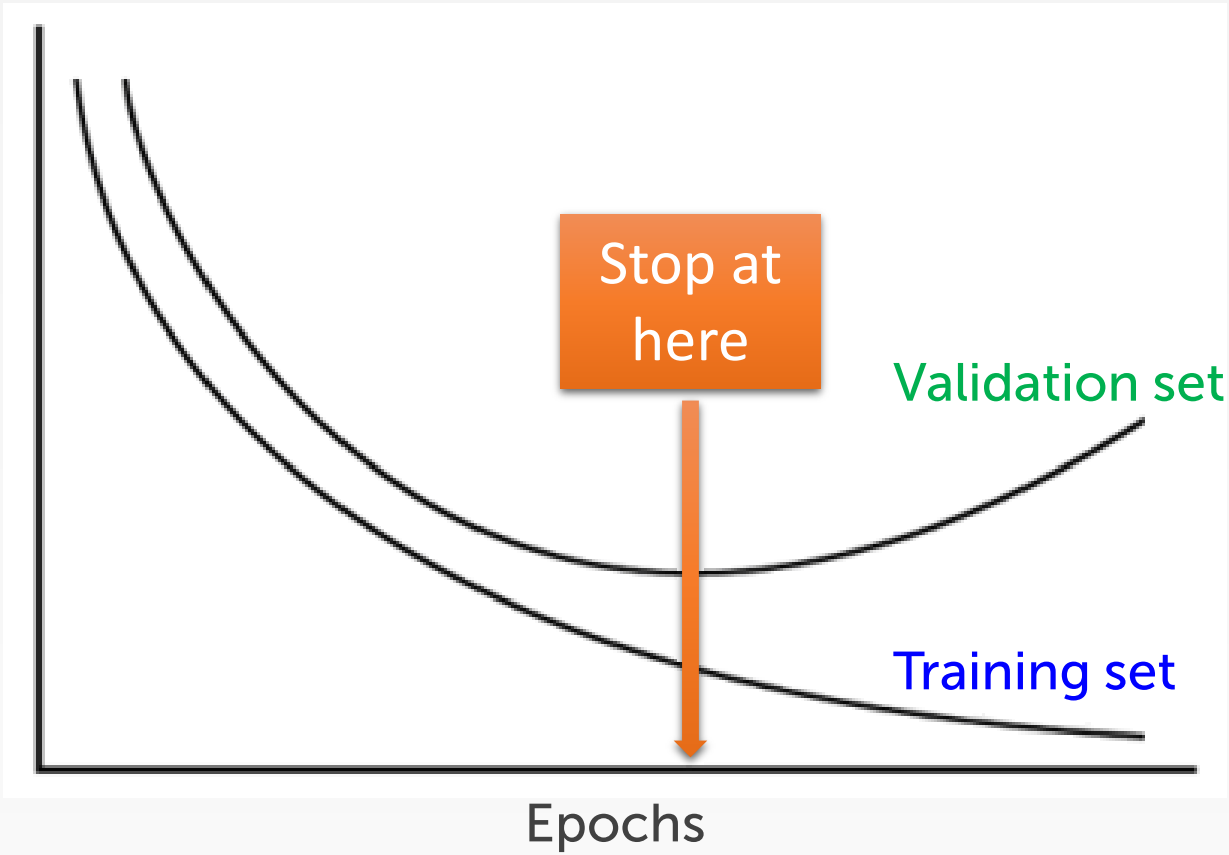
  Set the learning rate to $\eta(t) = \eta_0 \left(1 + \frac{t}{r}\right)^{-c}$. The hyperparameter c is typically set to 1. This is similar to exponential scheduling, but the learning rate drops much more slowly.

Avoid Overfitting

Recipe of Deep Learning



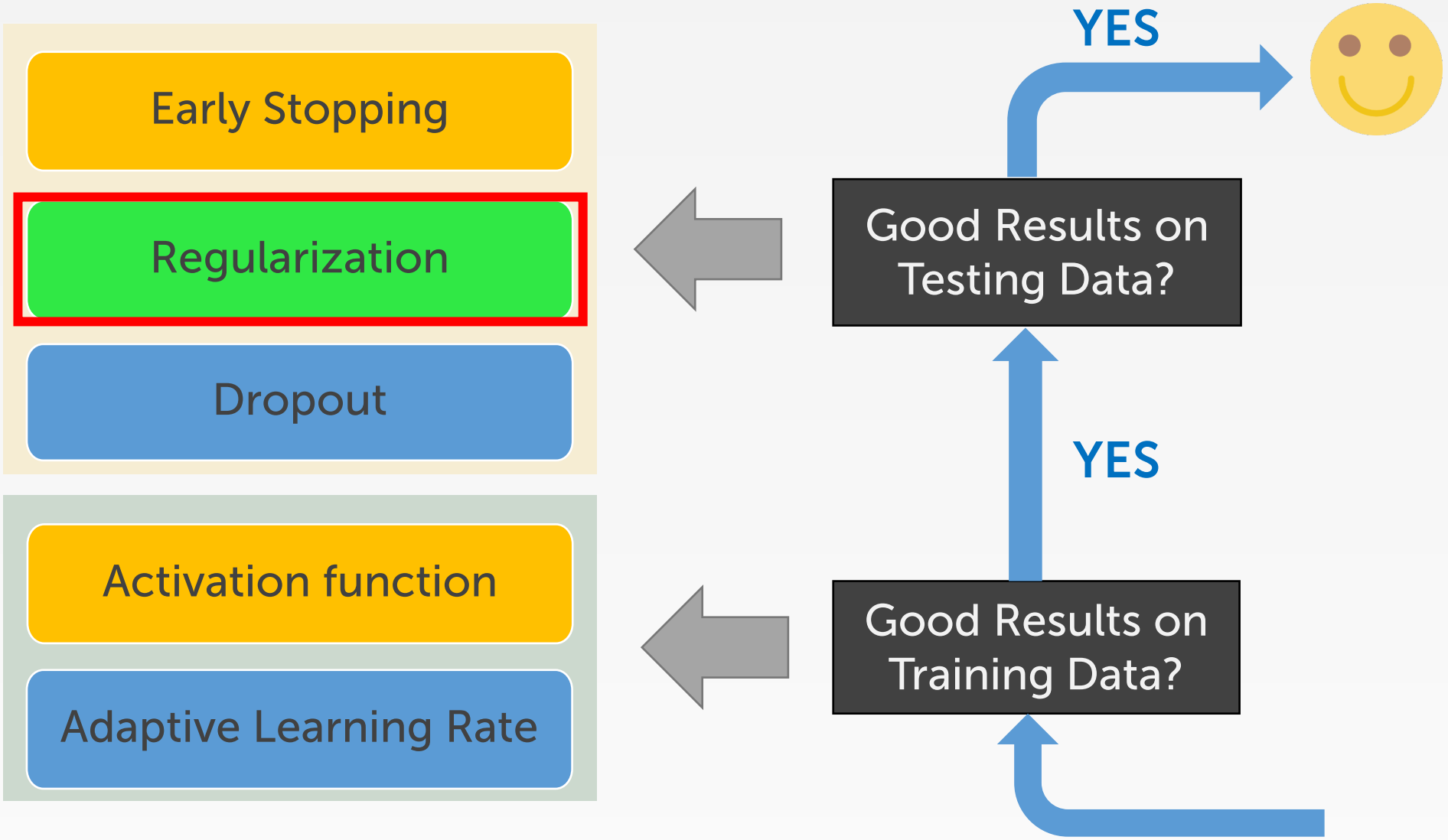
Early Stopping



Example:

- evaluate the model on a validation set at regular intervals (e.g., every 50 steps)
- save a "winner" snapshot if it outperforms previous "winner" snapshots
- Count the number of steps since the last "winner" snapshot was saved, and interrupt training when this number reaches some limit (e.g., 2,000 steps)
- restore the last "winner" snapshot.

Recipe of Deep Learning

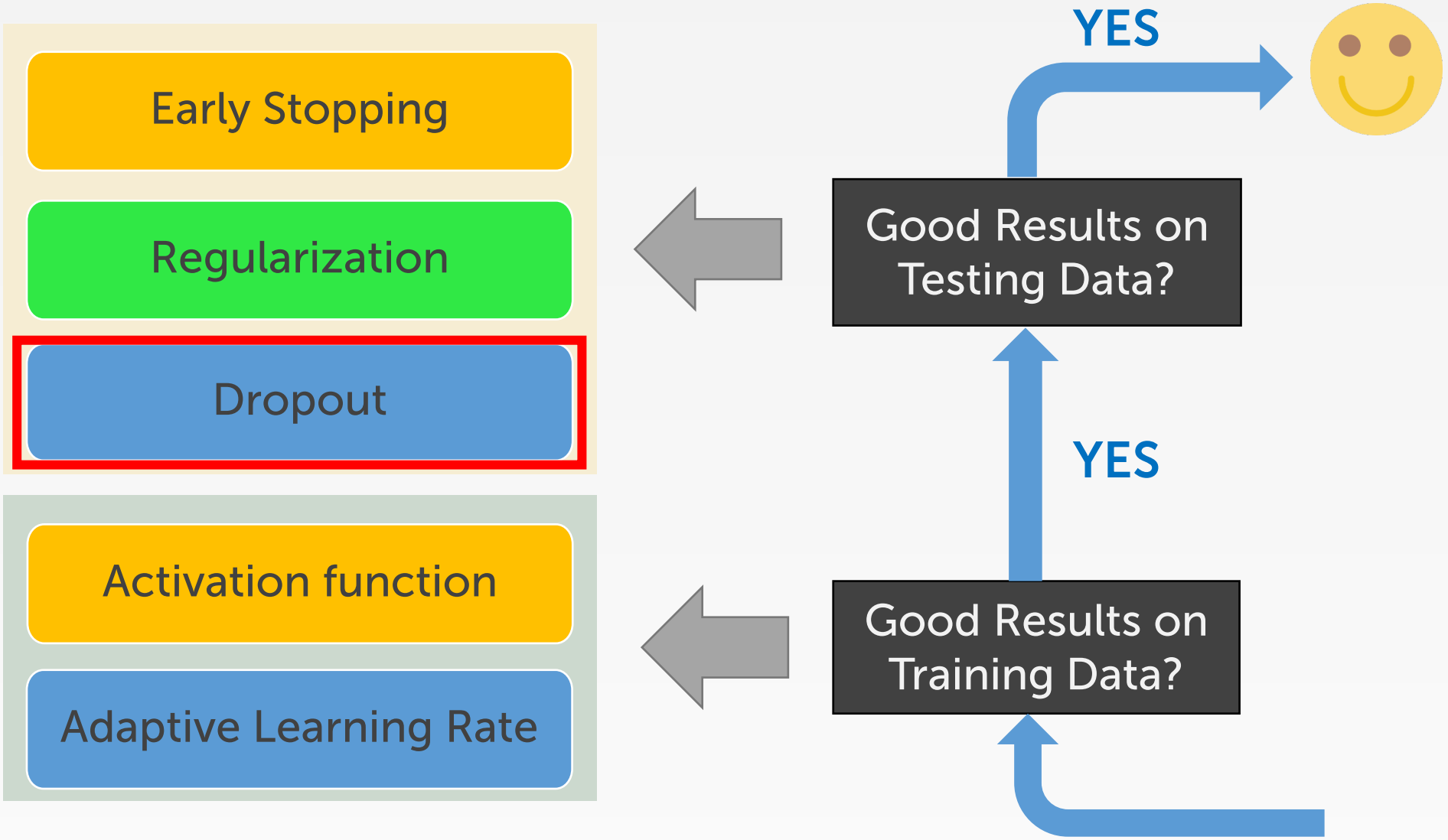


Regulation

- New loss function:
  - $L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2$

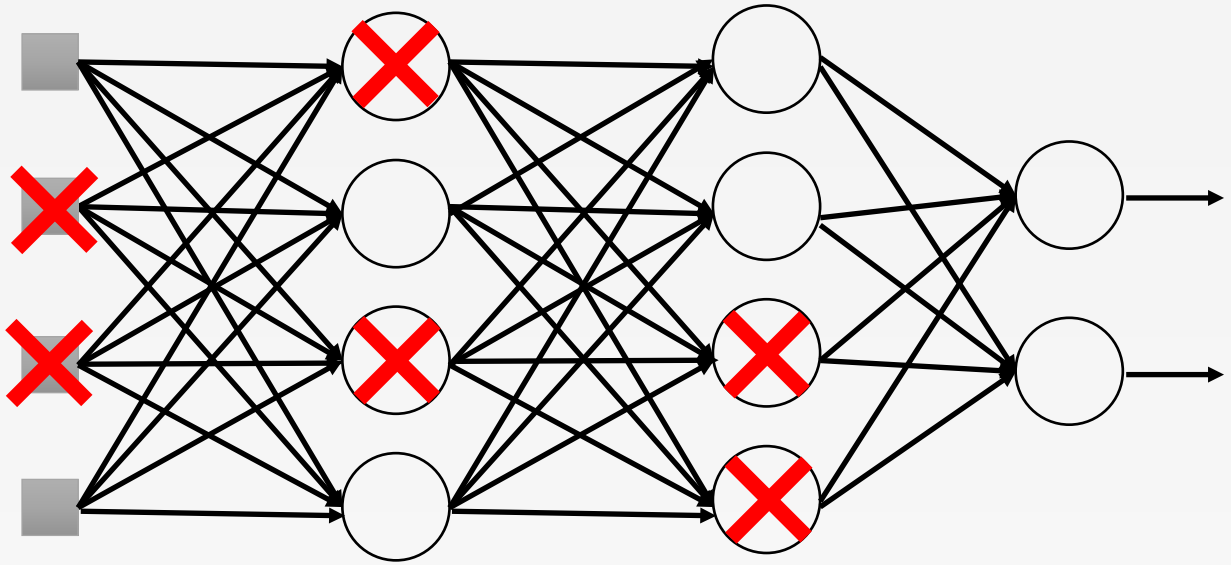
- $l2$ normalization:
  - $\|\theta\|_2 = (\omega_1)^2 + (\omega_2)^2 + \cdots$
- $l1$ normalization:
  - $\|\theta\|_2 = |\omega_1| + |\omega_2| + \cdots$

Recipe of Deep Learning



Dropout

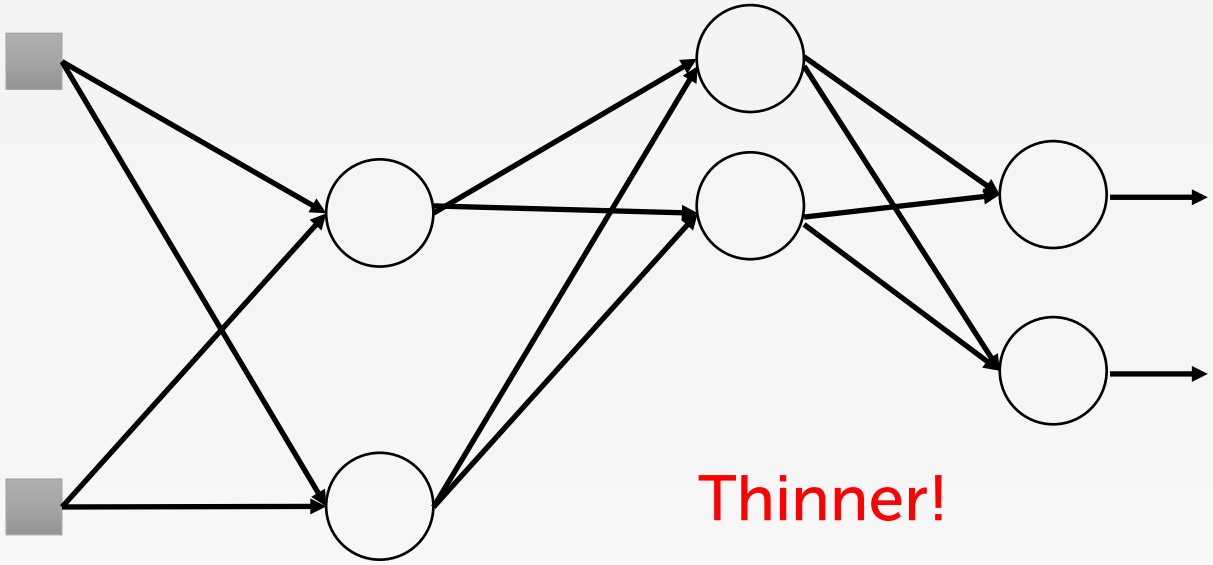
Training:



- **Each time before updating the parameters**
  - Each neuron has p% to dropout

Dropout

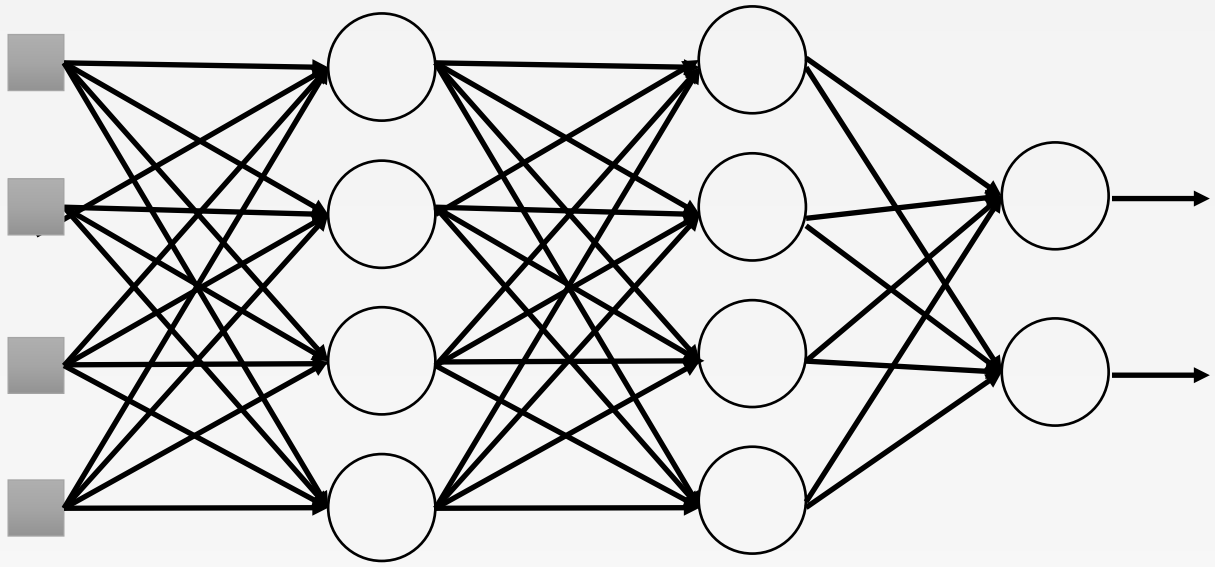
Training:



Thinner!

Dropout

Testing:



➢ **No dropout**

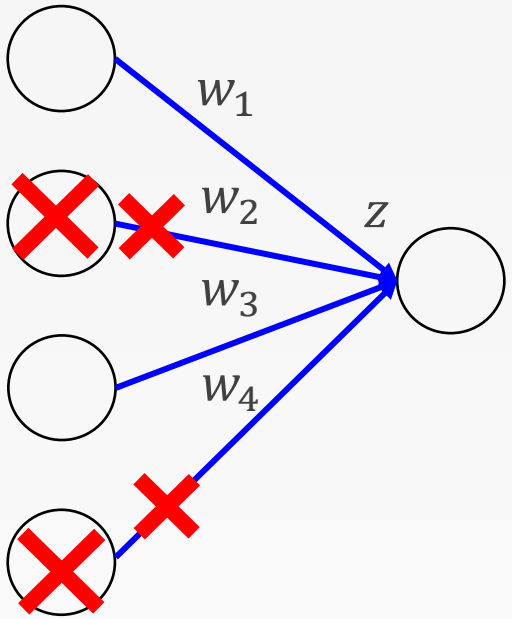
- If the dropout rate at training is p%, all the weights times 1-p%
- Assume that the dropout rate is 50%. If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout - Intuitive Reason

- Why the weights should multiply (1-p)% (dropout rate) when testing?

Training of Dropout

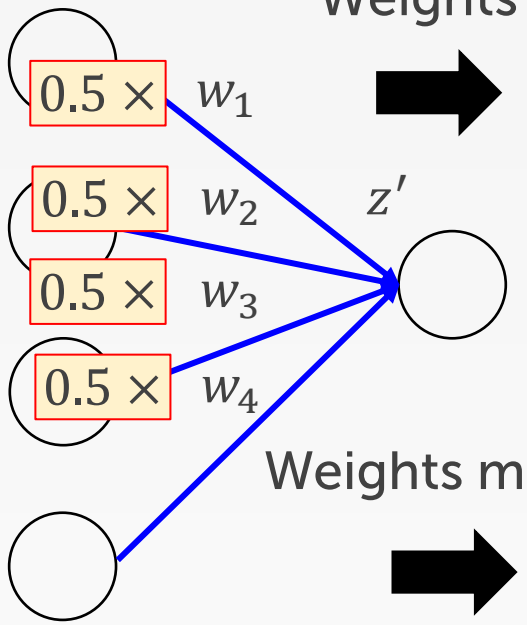
Assume dropout rate is 50%



Testing of Dropout

No dropout

Weights from training

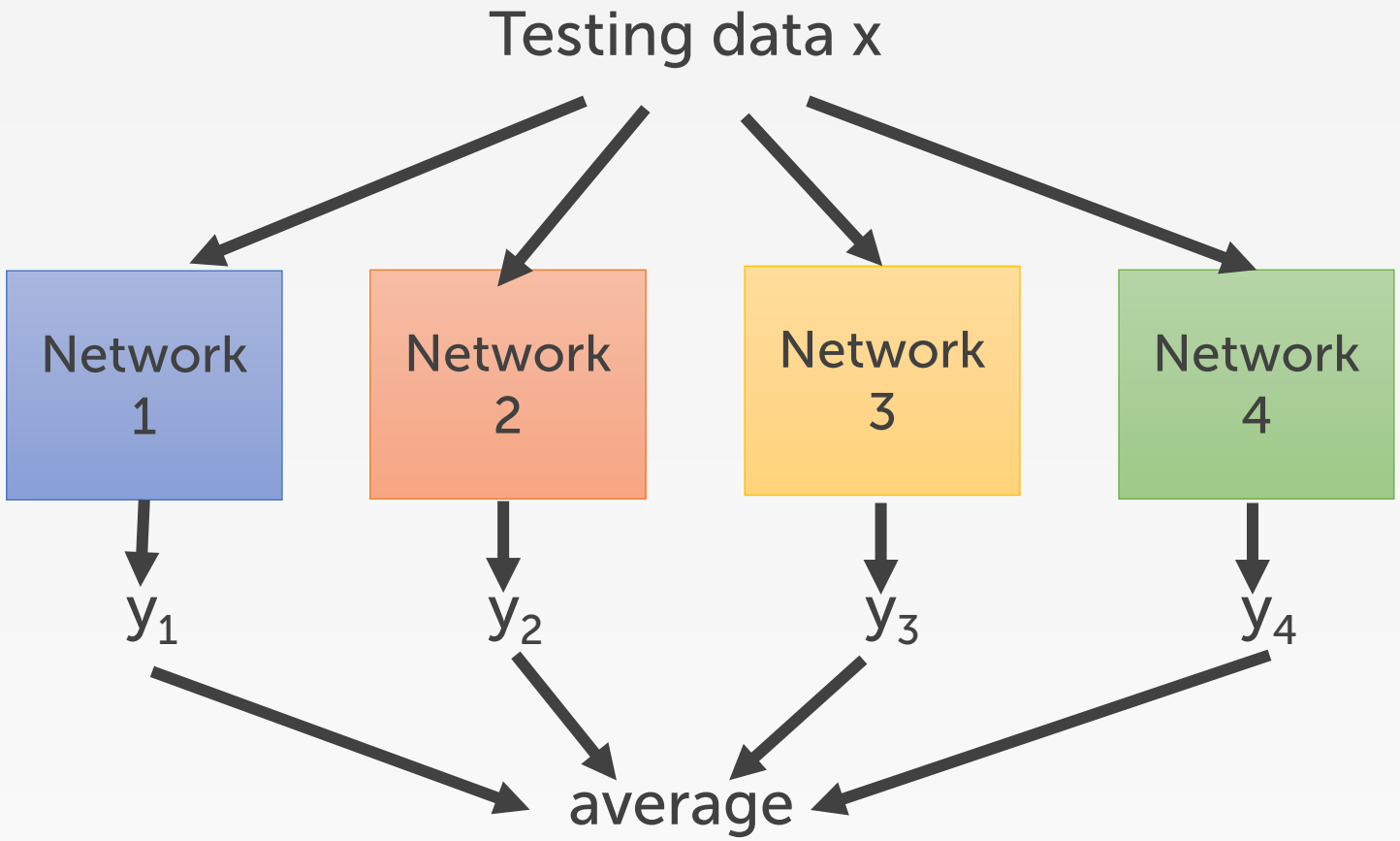


$z' \approx 2z$

Weights multiply 1-p%

$z' \approx z$

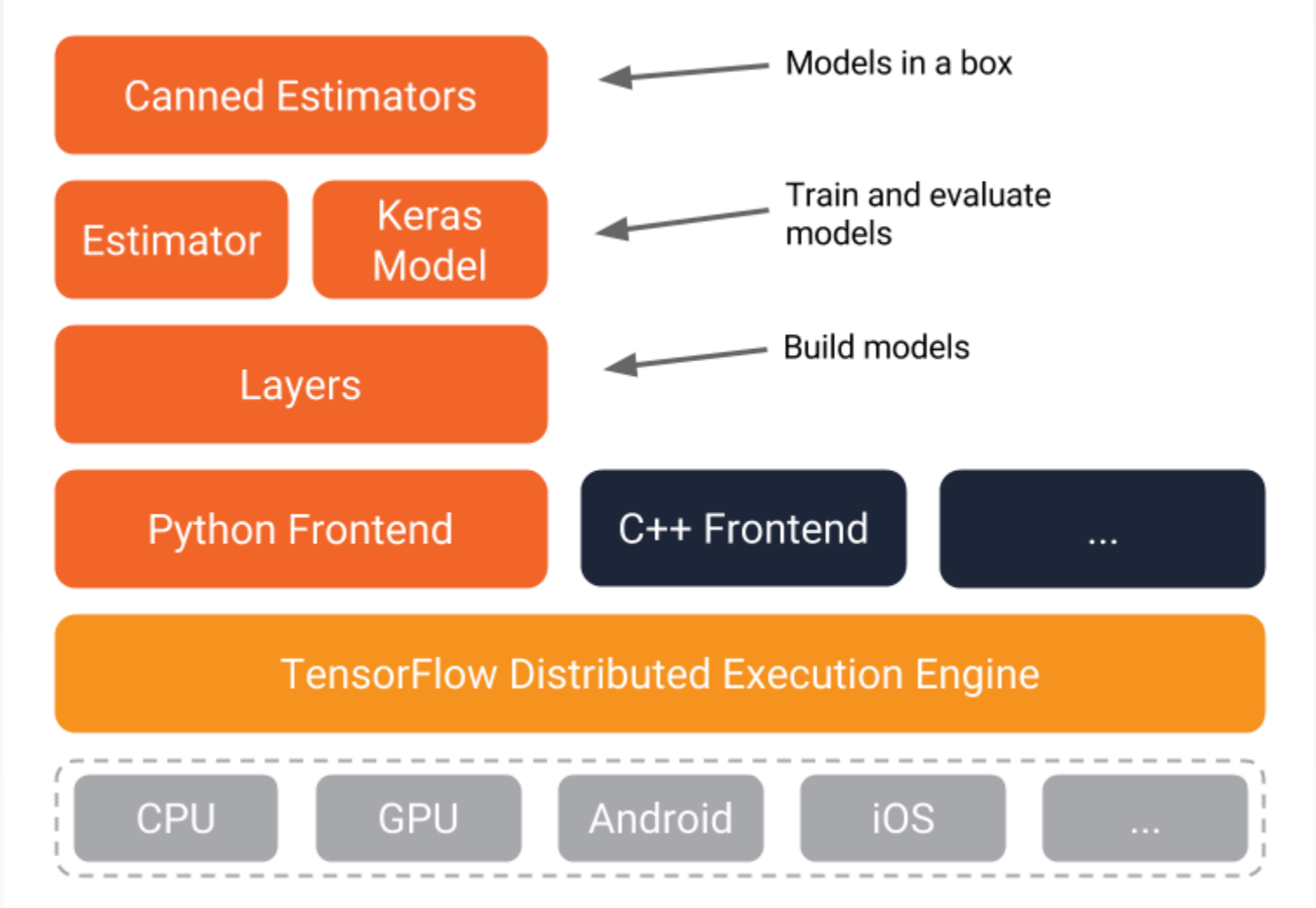
Dropout is a kind of ensemble.

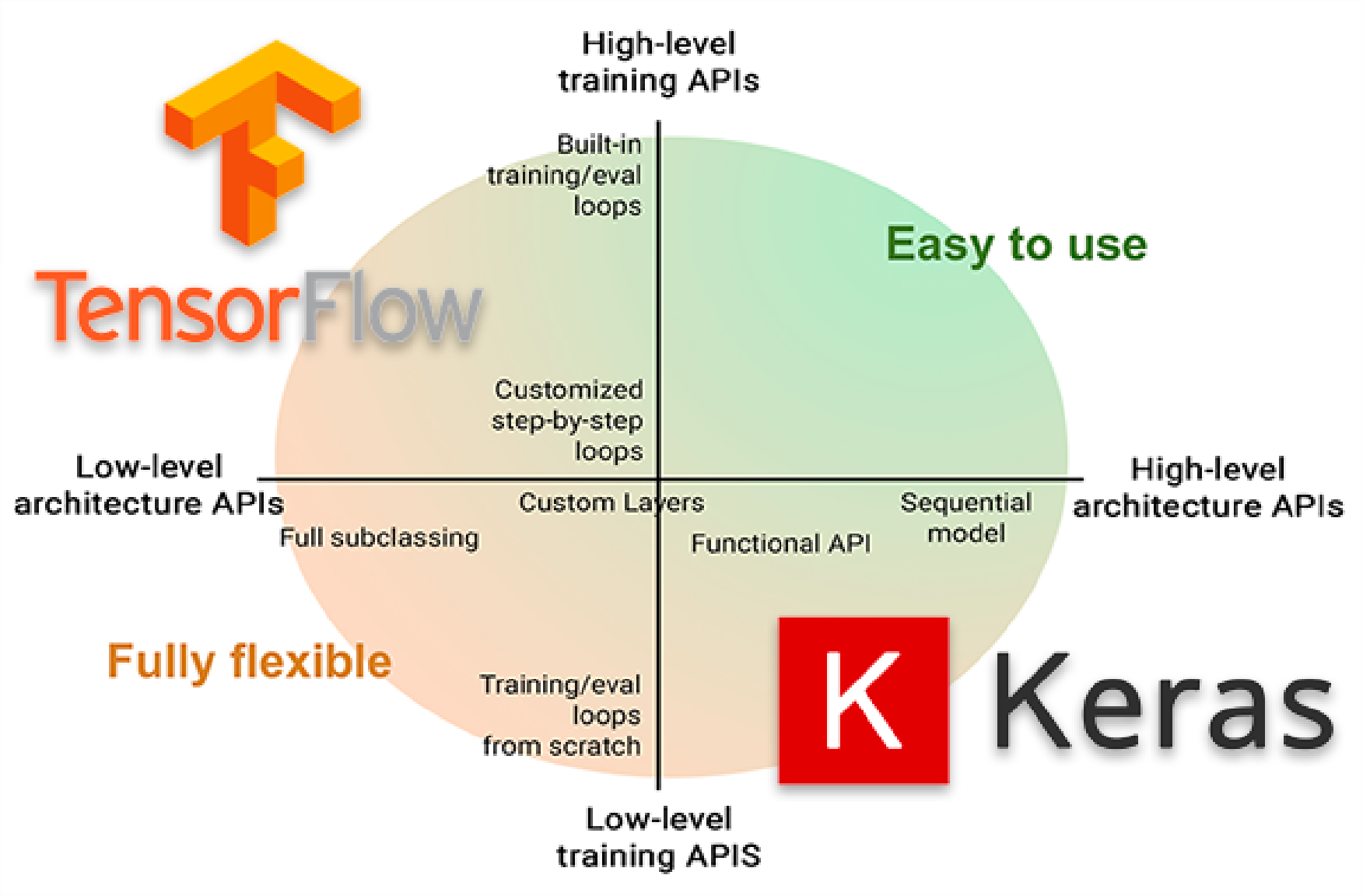


"Hello World"

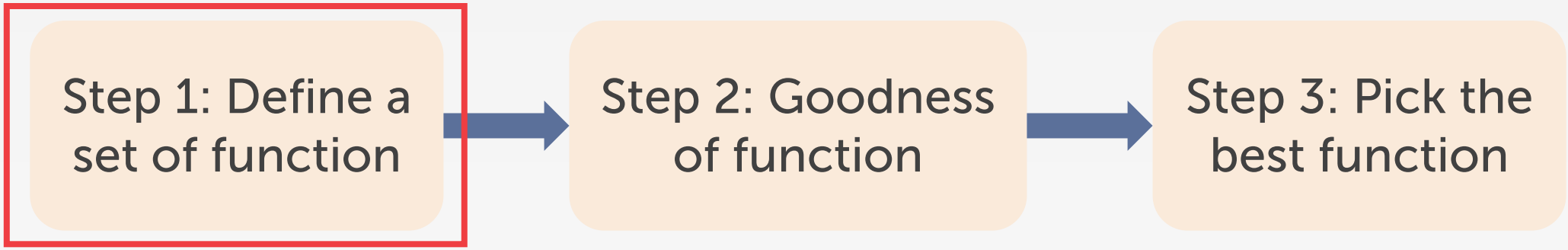
Source: YichengSong@UMN

TensorFlow

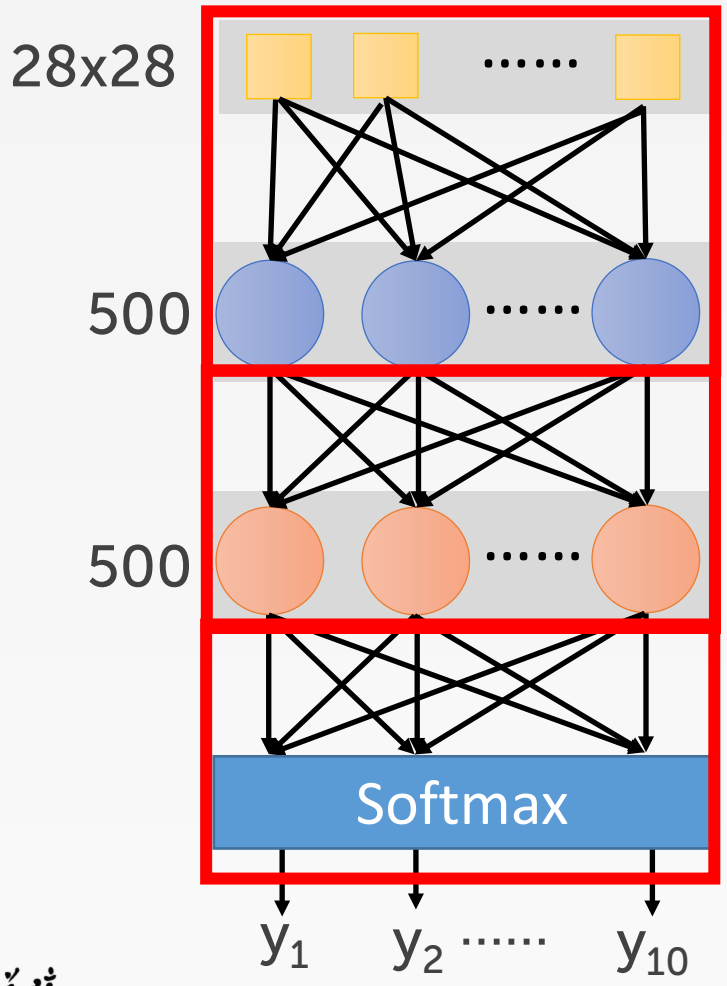

Canned Estimators
Models in a box
Estimator
Keras Model
Train and evaluate models
Layers
Build models
Python Frontend
C++ Frontend
...
TensorFlow Distributed Execution Engine
CPU
GPU
Android
iOS
...


High-level
training APIs
Built-in
training/eval
loops
Easy to use
TensorFlow
Customized
step-by-step
loops
Low-level
architecture APIs
High-level
architecture APIs
Custom Layers
Sequential
model
Full subclassing
Functional API
Fully flexible
Training/eval
loops
from scratch
K
Keras
Low-level
training APIS

Keras


Step 1: Define a set of function
Step 2: Goodness of function
Step 3: Pick the best function

Keras: Define a Function




```
model = Sequential()
```

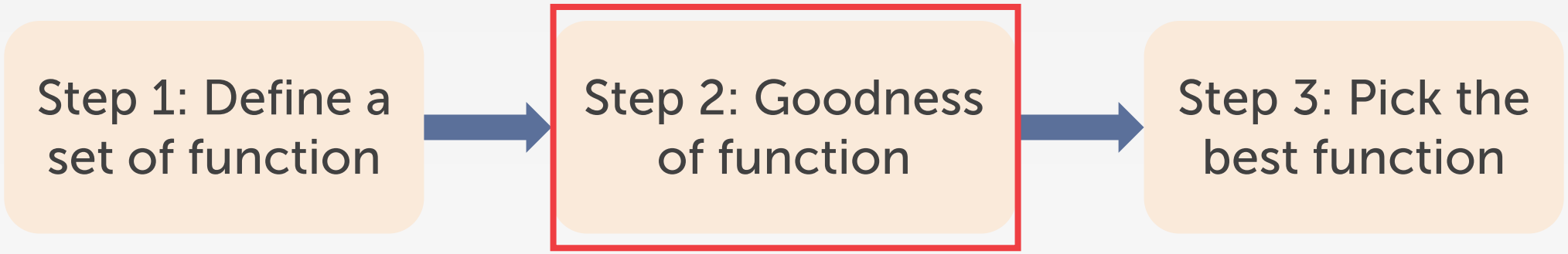
```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

softplus, softsign, relu, tanh, hard_sigmoid, linear

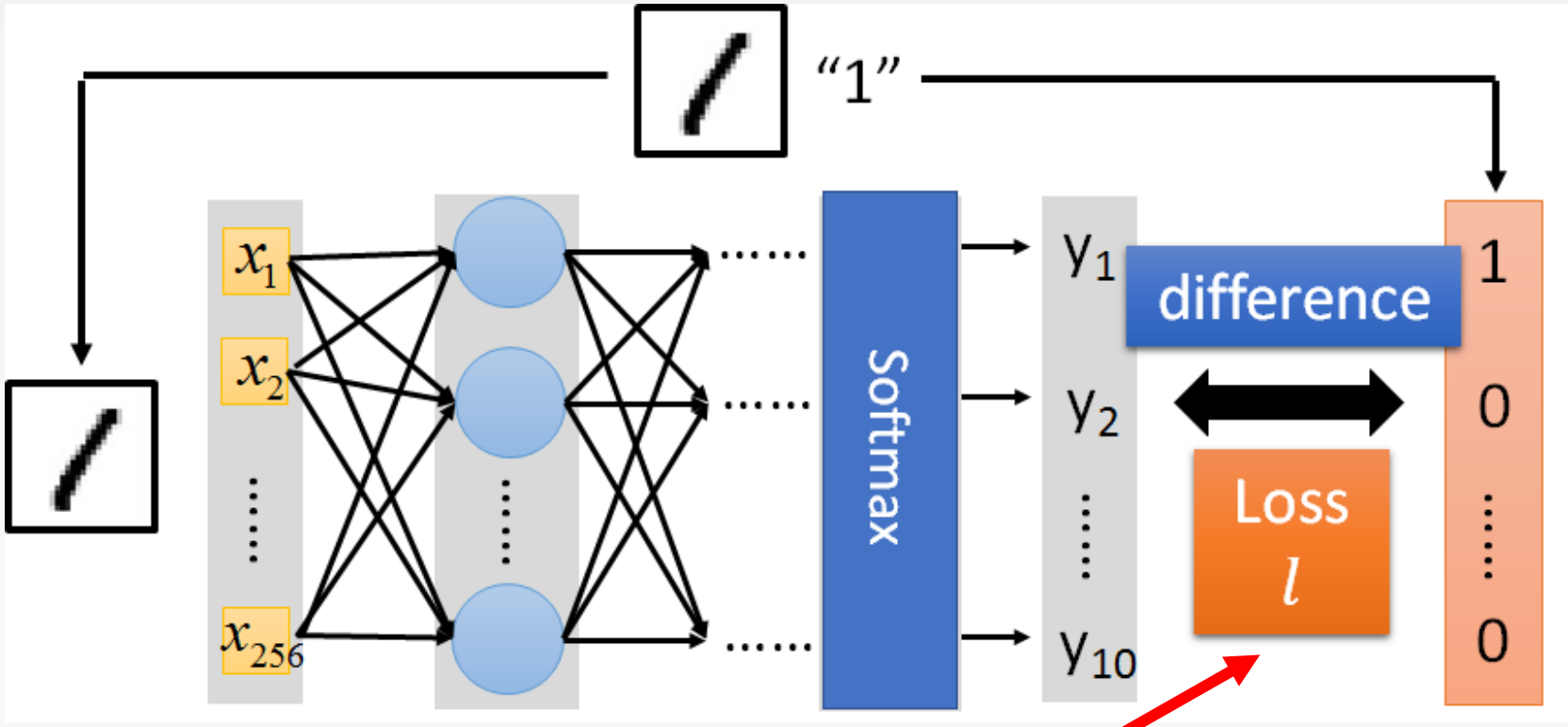
```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

Keras


Step 1: Define a set of function
Step 2: Goodness of function
Step 3: Pick the best function

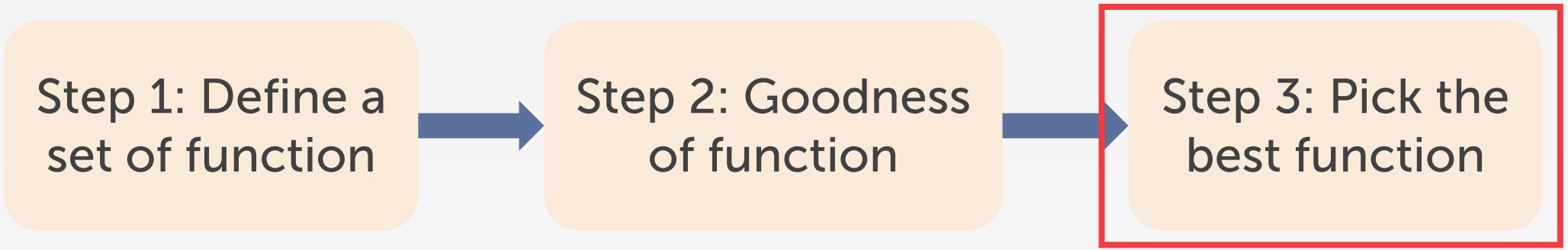
Keras: Goodness of Function




```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Several alternatives: https://keras.io/objectives/

Keras


Step 1: Define a set of function
Step 2: Goodness of function
Step 3: Pick the best function

Keras: Pick the Best

Step 3.1: Configuration

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam

Step 3.2: Find the optimal network parameters

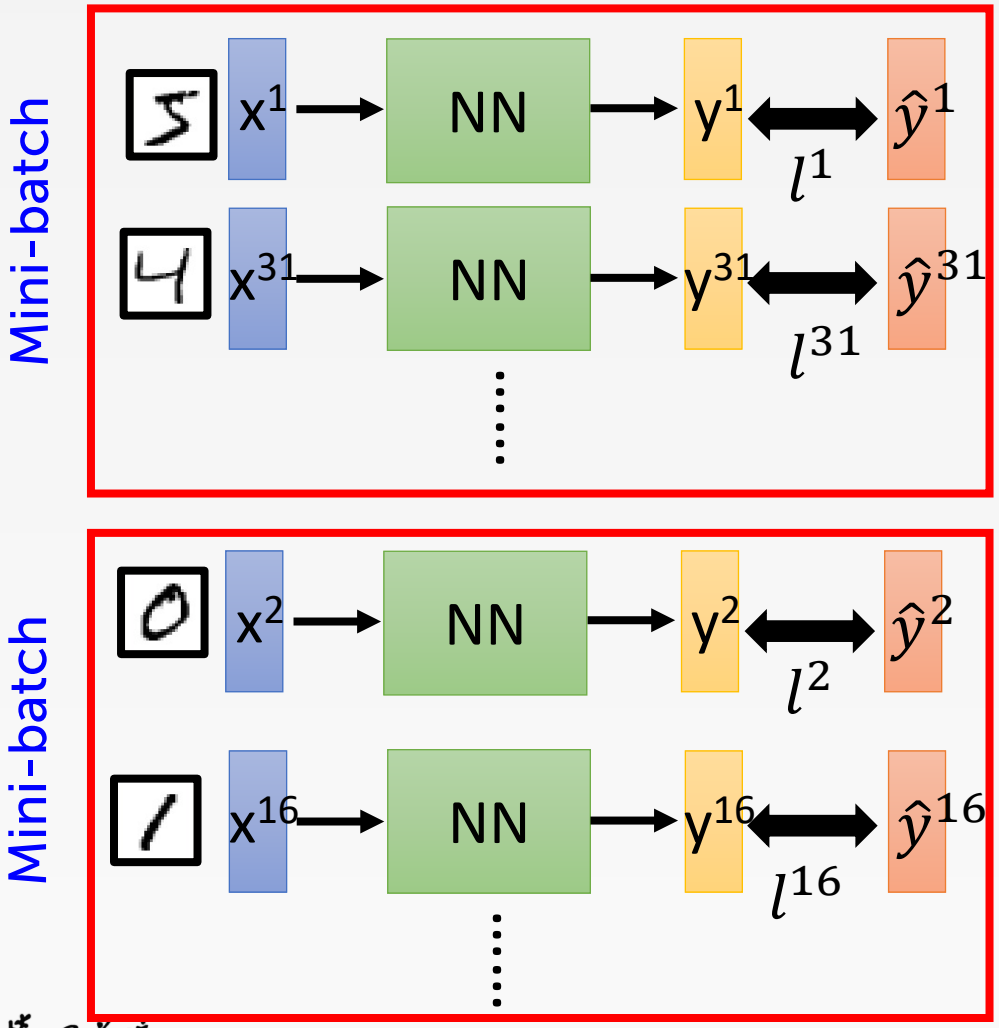
```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Training data (Images)

Labels

In the following slides

Mini-batch



- Randomly initialize network parameters

- Pick the 1st batch

  $L' = l^1 + l^{31} + \cdots$

  Update parameters once

- Pick the 2nd batch

  $L'' = l^2 + l^{16} + \cdots$

  Update parameters once

  ⋮

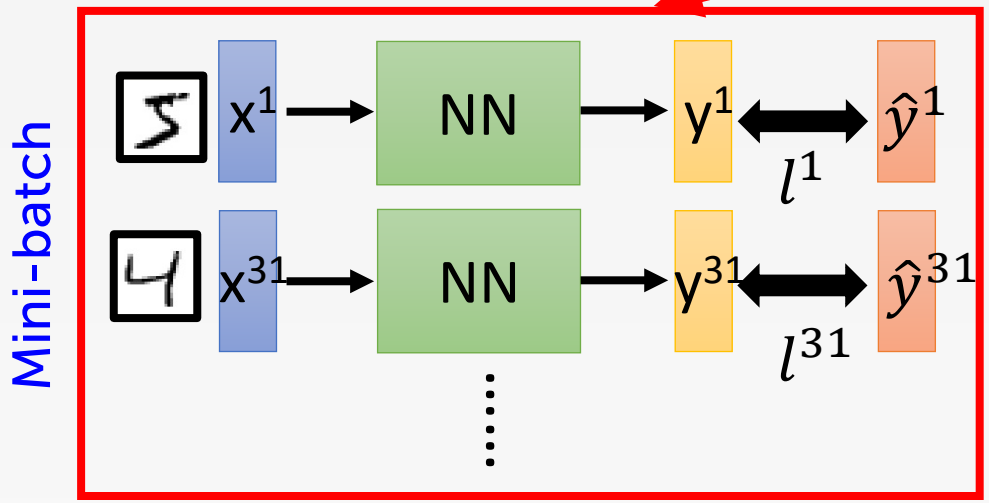
- Until all mini-batches have been picked

One epoch

Repeat the above process

Mini-batch

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



100 examples in a mini-batch

- Pick the 1st batch

  $L' = l^1 + l^{31} + \cdots$

  Update parameters once

- Pick the 2nd batch

  $L'' = l^2 + l^{16} + \cdots$

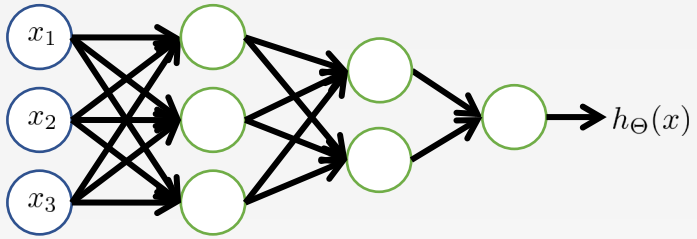
  Update parameters once

  ⋮

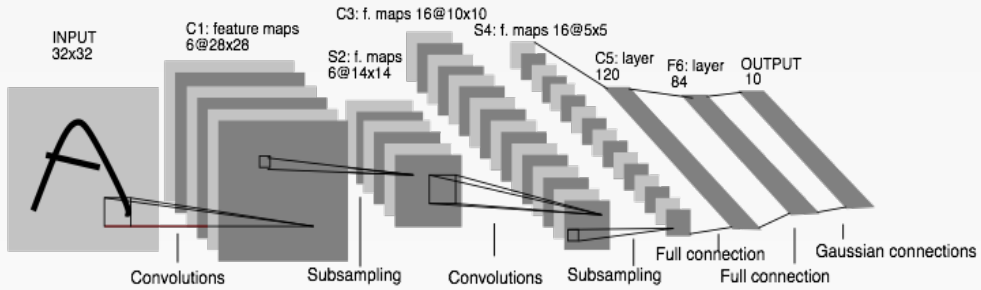
- Until all mini-batches have been picked

Network Architectures

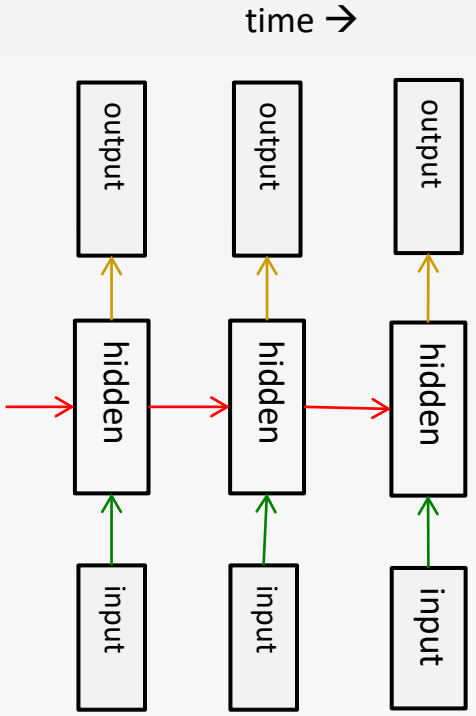
Fully connected



Convolutional



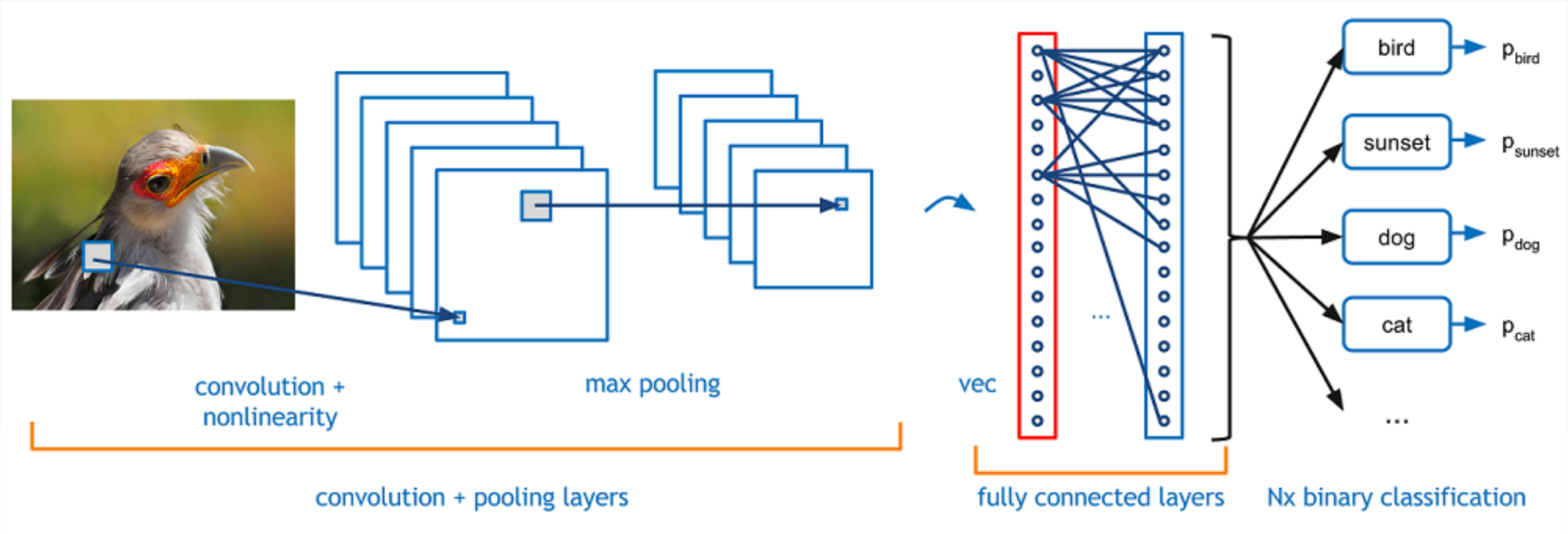
Recurrent



Convolutional Neural Network

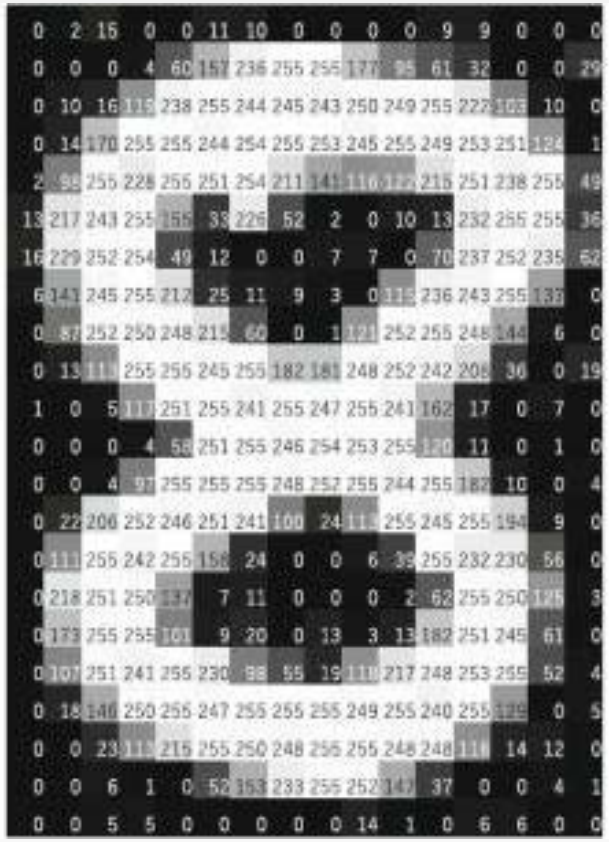
Source: https://www.deeplearningbook.org/slides/09_conv.pdf

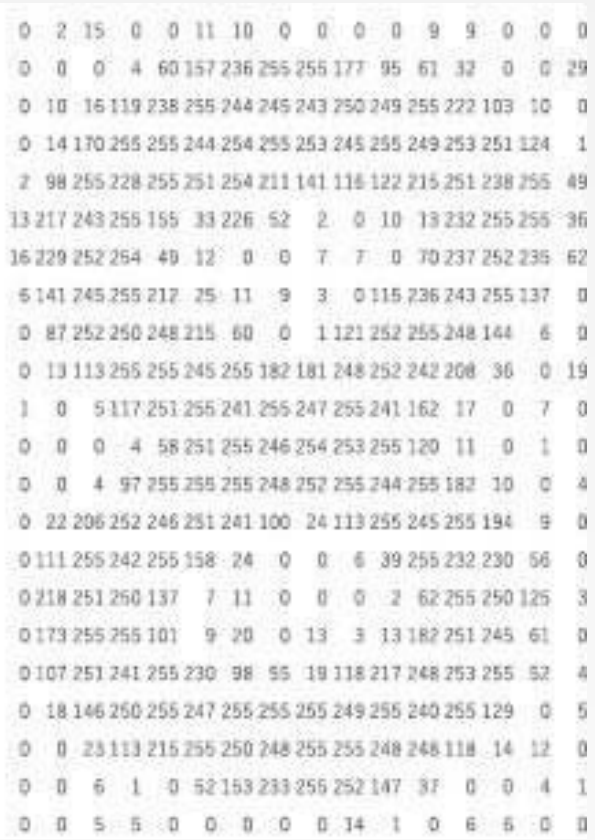
CNN Architecture



Key Ideas

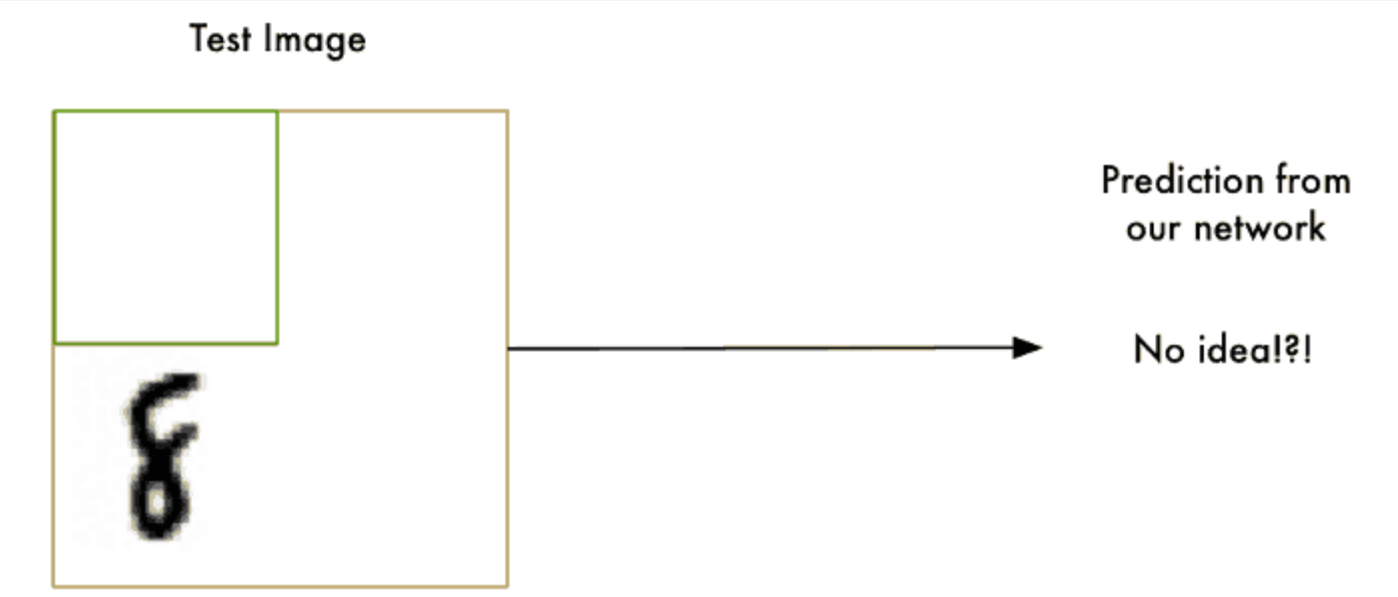
- Replace matrix multiplication in neural nets with convolution
- Parameter Sharing
  - Parameter sharing scheme is used in Convolutional Layers to control the number of parameters
- Image processing:
  - Not connected to every single pixel in the input image, but only to pixels in their receptive fields.

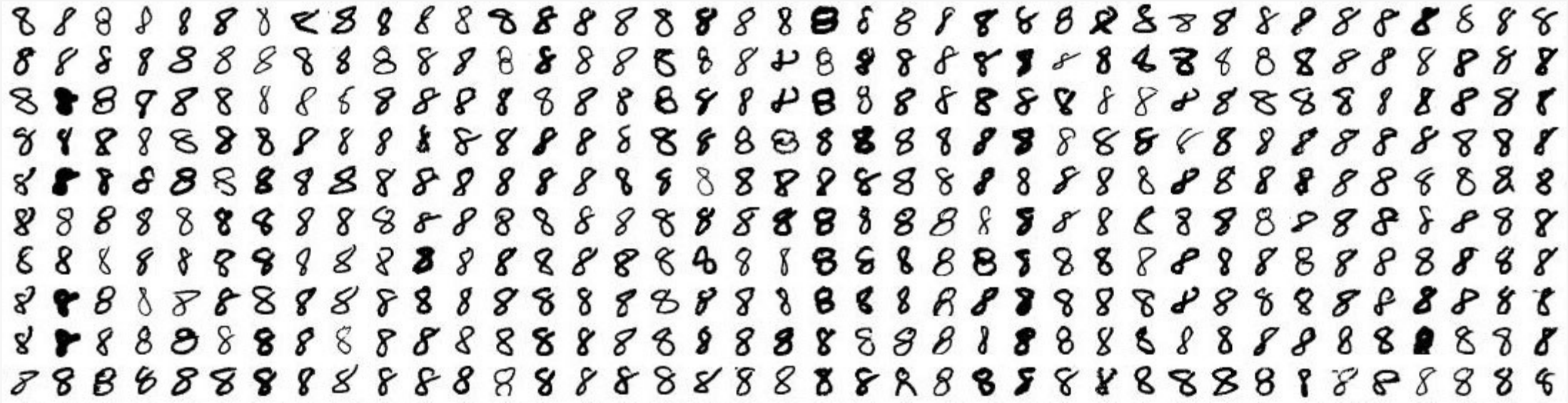




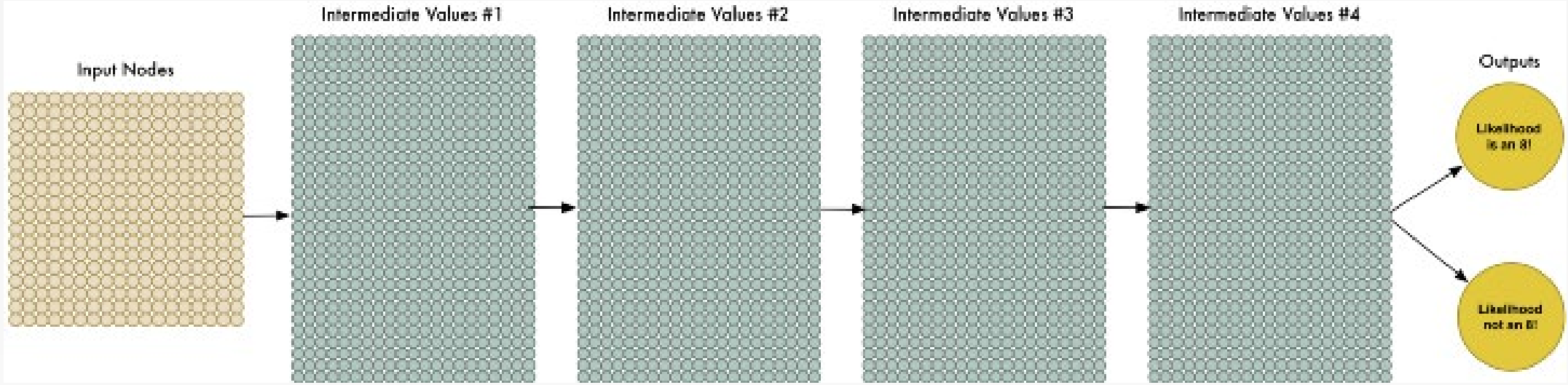
Preprocessing of pixel values

Search with a Sliding Window

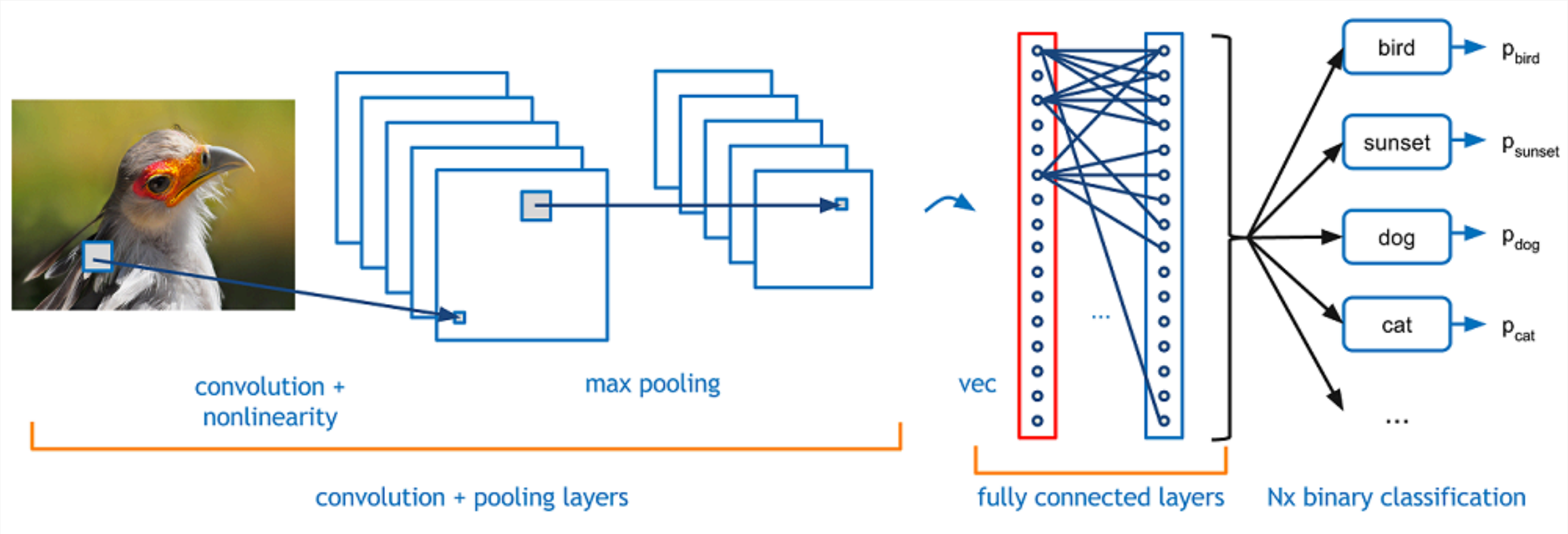




A Deep Neural Net?

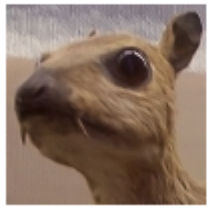


CNN Architecture



Convolution Layer

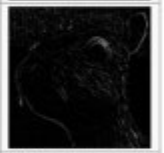
Input image



Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

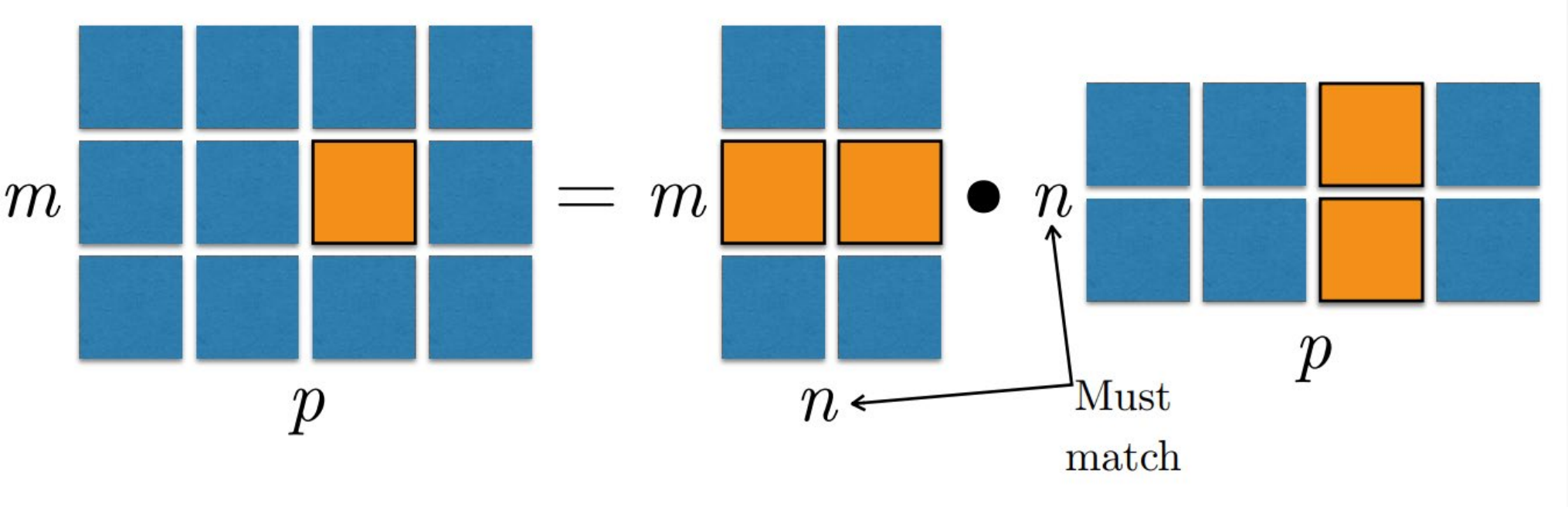
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\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

[CNN Explainer]

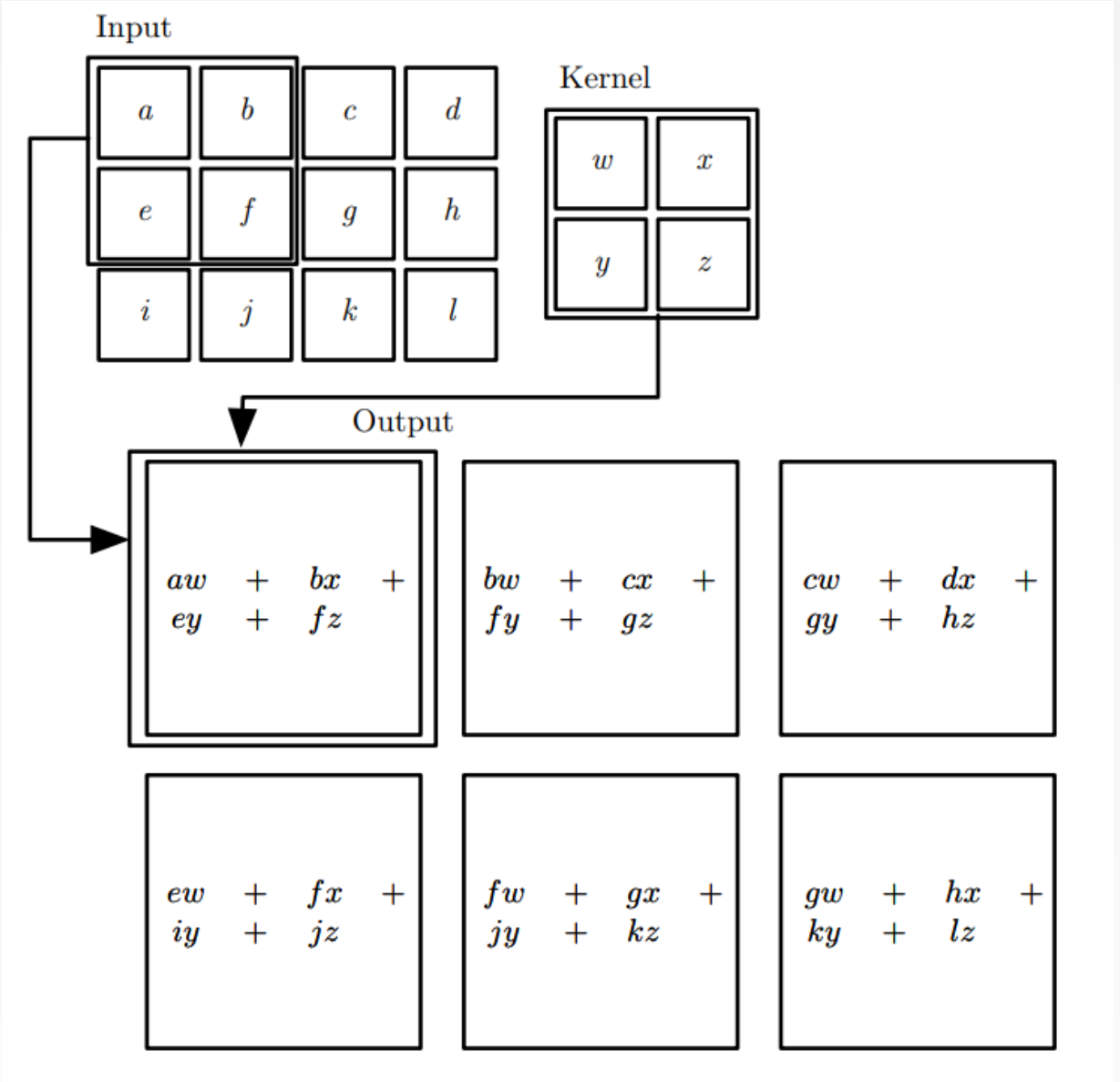
Matrix Product

$$\boldsymbol{C = AB}$$

$$C_{ij} = \sum_k A_{i,k} B_{k,j}$$



2D Convolution



Convolutional Layer

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

6 x 6 image

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

Filter 1

Matrix

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

Filter 2

Matrix

⋮ ⋮

Those are the network parameters to be learned.

Convolutional Layer

stride=1

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

6 x 6 image

3 -1

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

Filter 1

Convolutional Layer

stride=2

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

6 x 6 image

3 -3

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

Filter 1

We set stride=1 below

Convolutional Layer

stride=1

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

6 x 6 image

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

Filter 1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolutional Layer

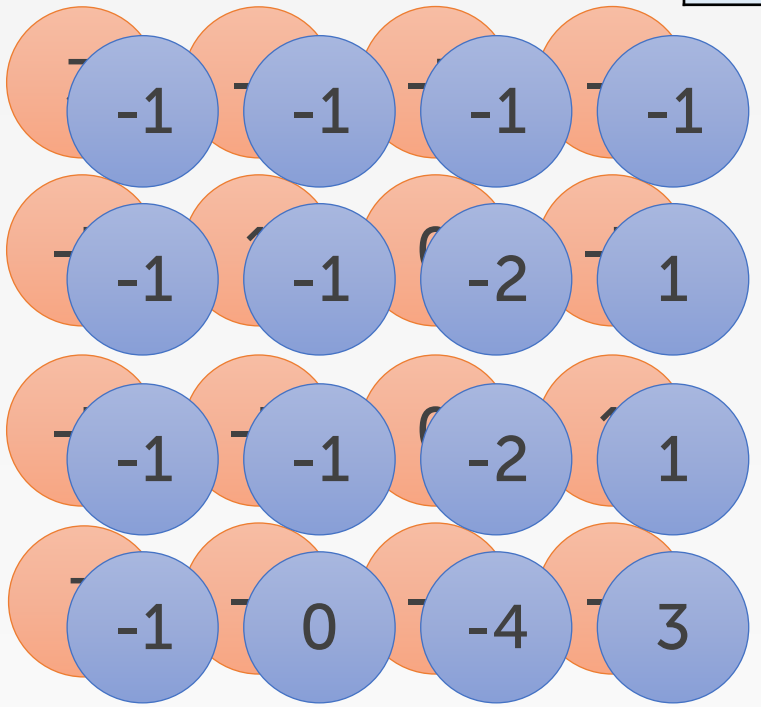
stride=1

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

6 x 6 image

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

Filter 2



Padding

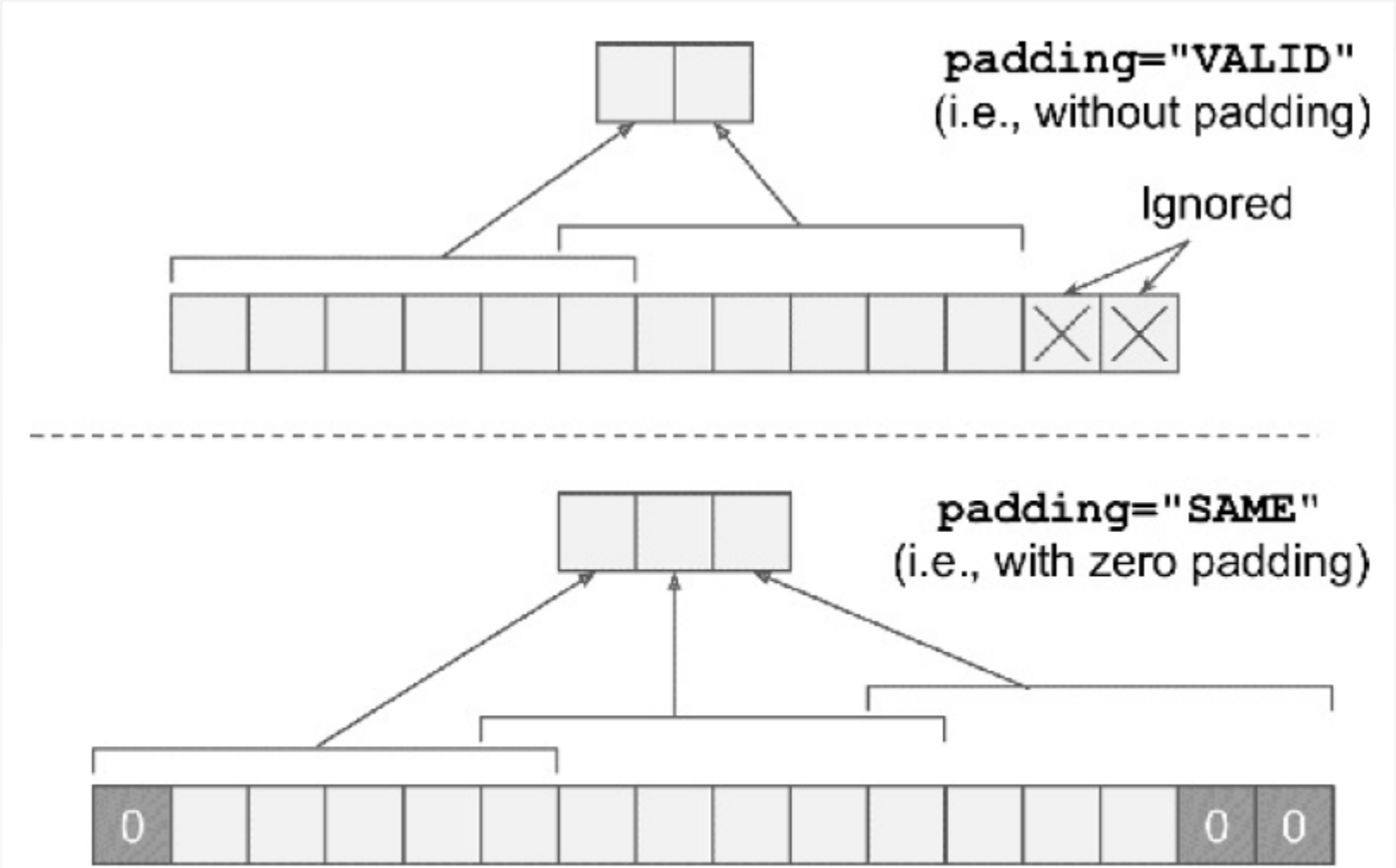
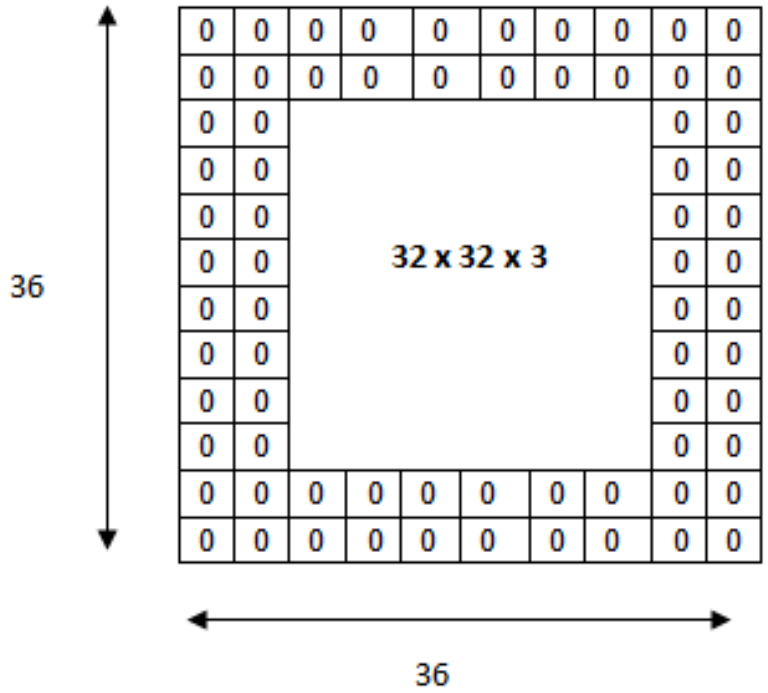


Figure 13-7. Padding options — input width: 13, filter width: 6, stride: 5

https://keras.io/api/layers/convolution_layers/convolution2d/

Padding

- If you have a stride of 1 and if you set the size of zero padding to $\frac{K-1}{2}$, where K is the filter size, then the input and output volume will always have the same spatial dimensions.



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

Padding

$$O = \frac{W-K+2P}{S} + 1$$

O: output height/length

W: input height/length

K: filter size

P: padding

S: stride

Real-world Example

- Input image: $227 \times 227 \times 3$
- Filter size: $11 \times 11 \times 3$; Stride = 4; no zero padding
- Number of filters = 96

- Output size: $55 \times 55 \times 96$

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

  - max-pooling: chose the max in a window
  - mean-pooling: take the average

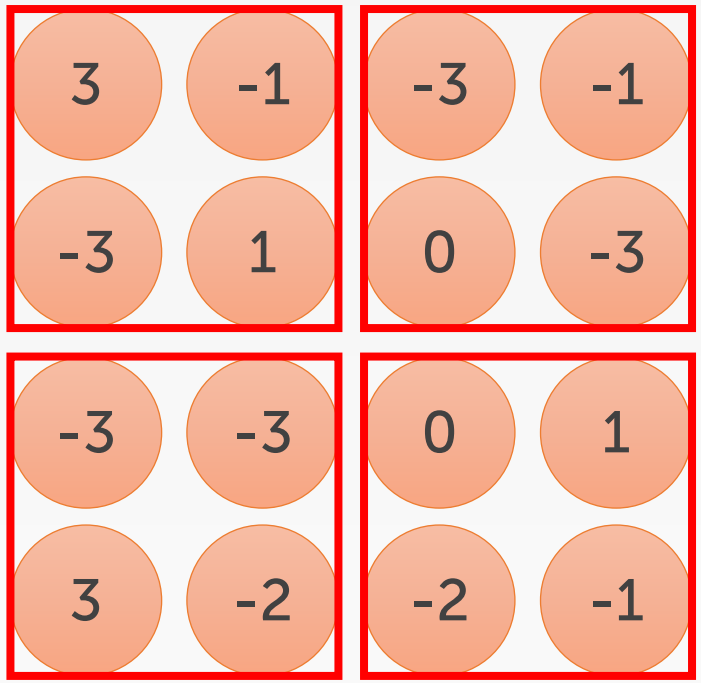
Max Pooling

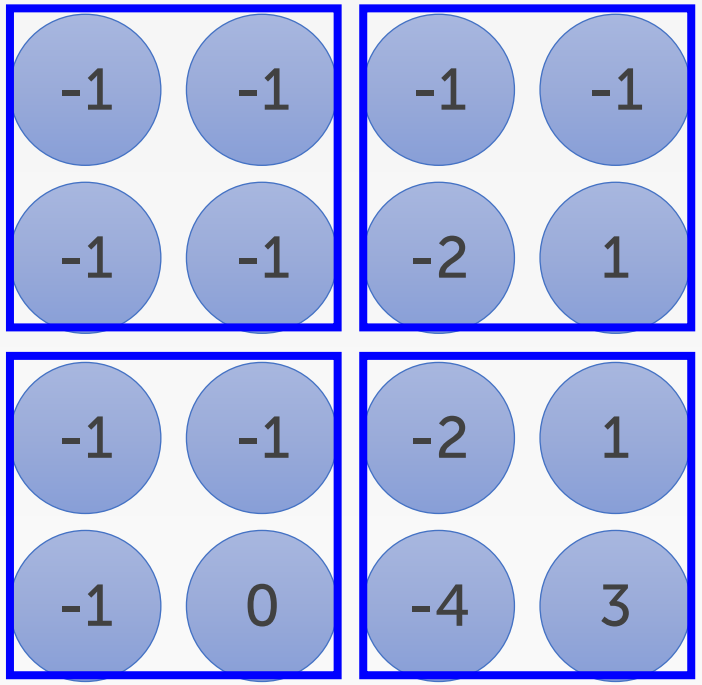
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

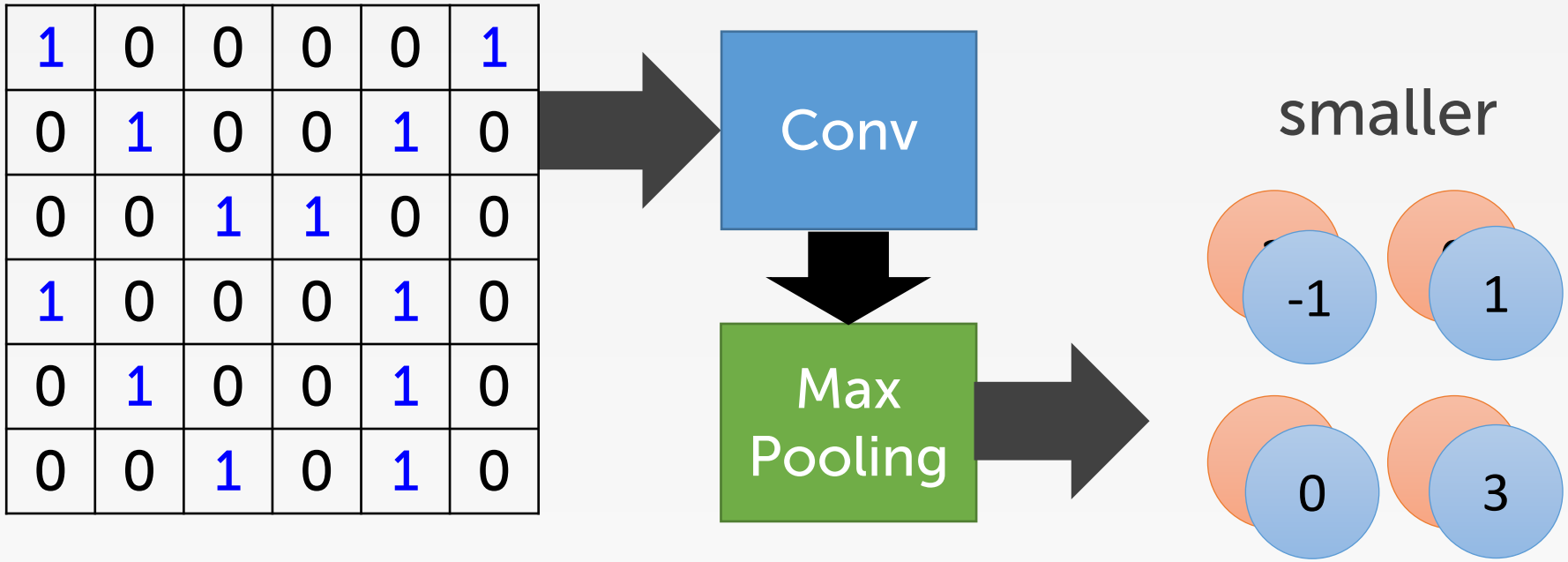
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



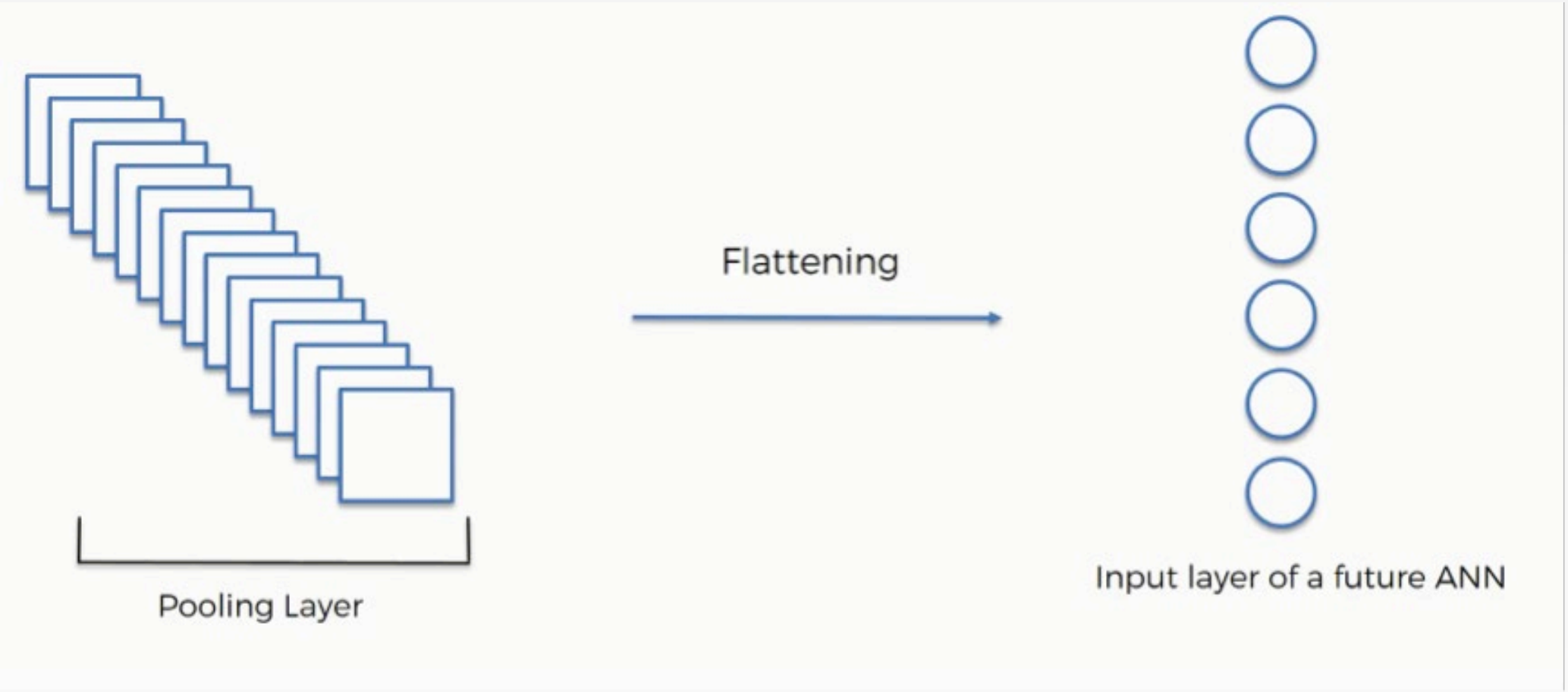


Max Pooling

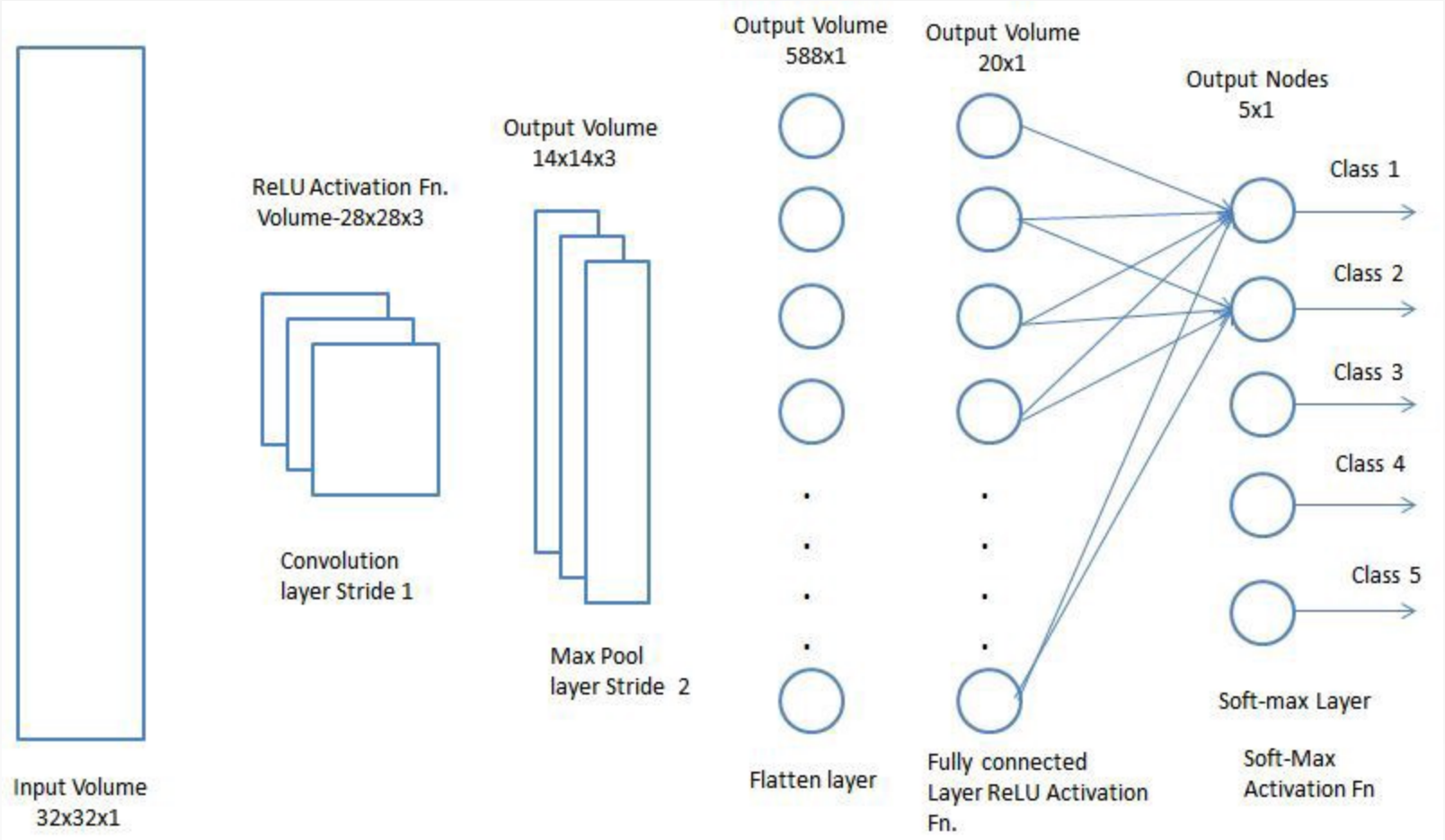


6 x 6 image

Fully Connected Layer



Fully Connected Layer



Popular CNN Architectures

- LeNet
- AlexNet
- GoogLeNet
- ResNet

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

LeNet-5 Architecture

LeNet-5

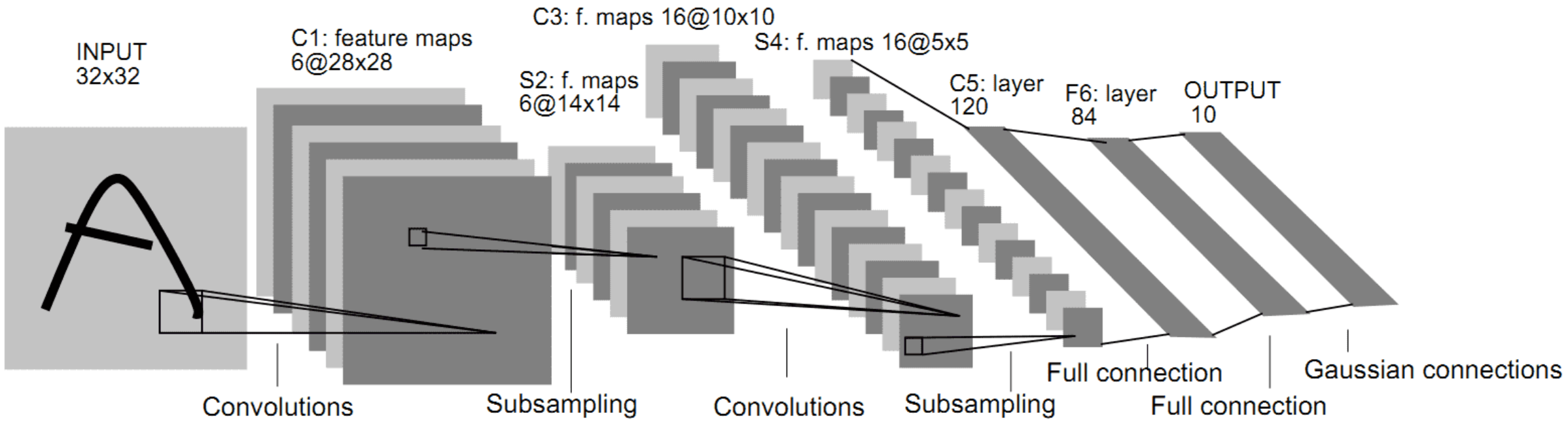


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

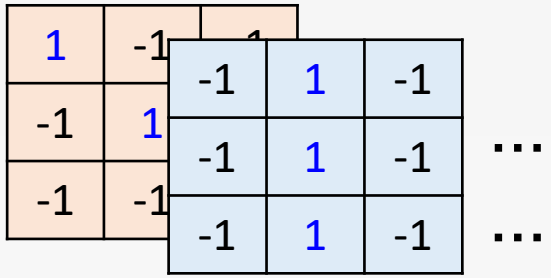
Implementation

```
keras.Input(shape=(28,28,1)),
layers.Conv2D(25, kernel_size=(3, 3), activation="relu"),
```

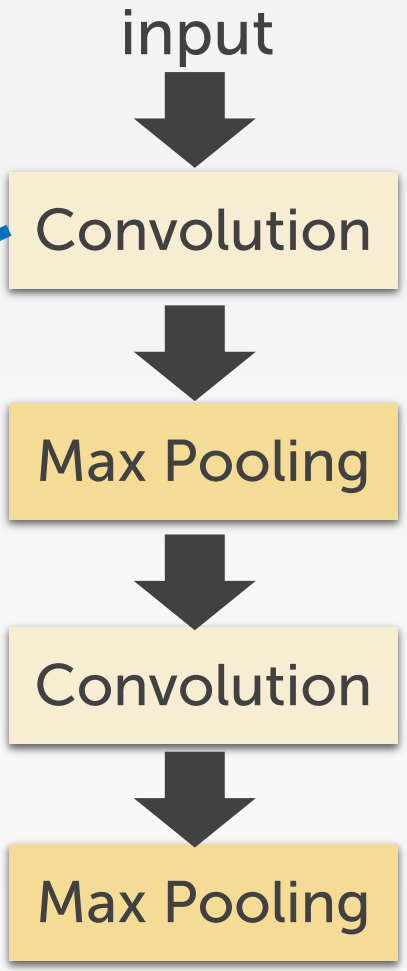
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

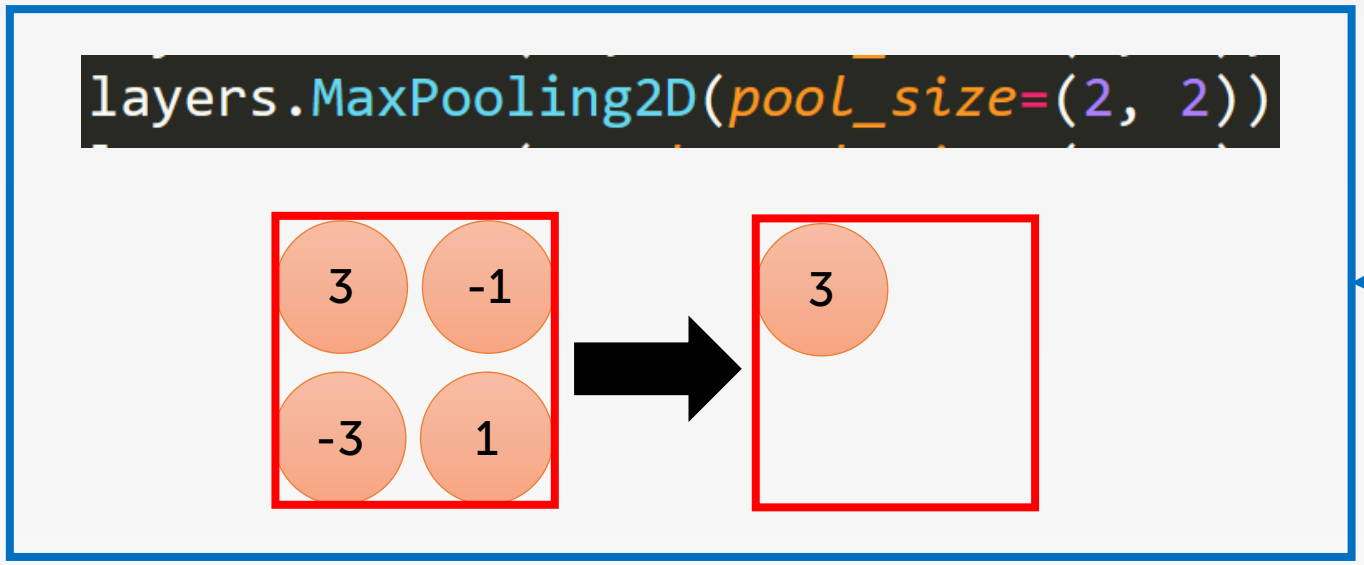


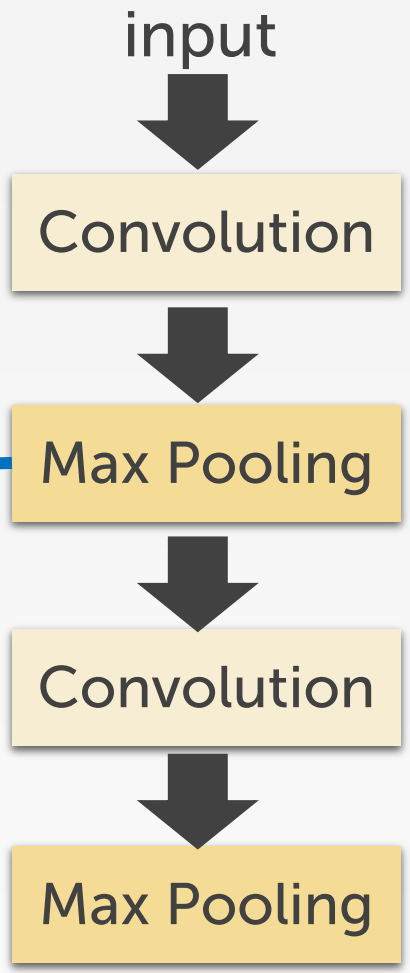
There are 25 3x3 filters.



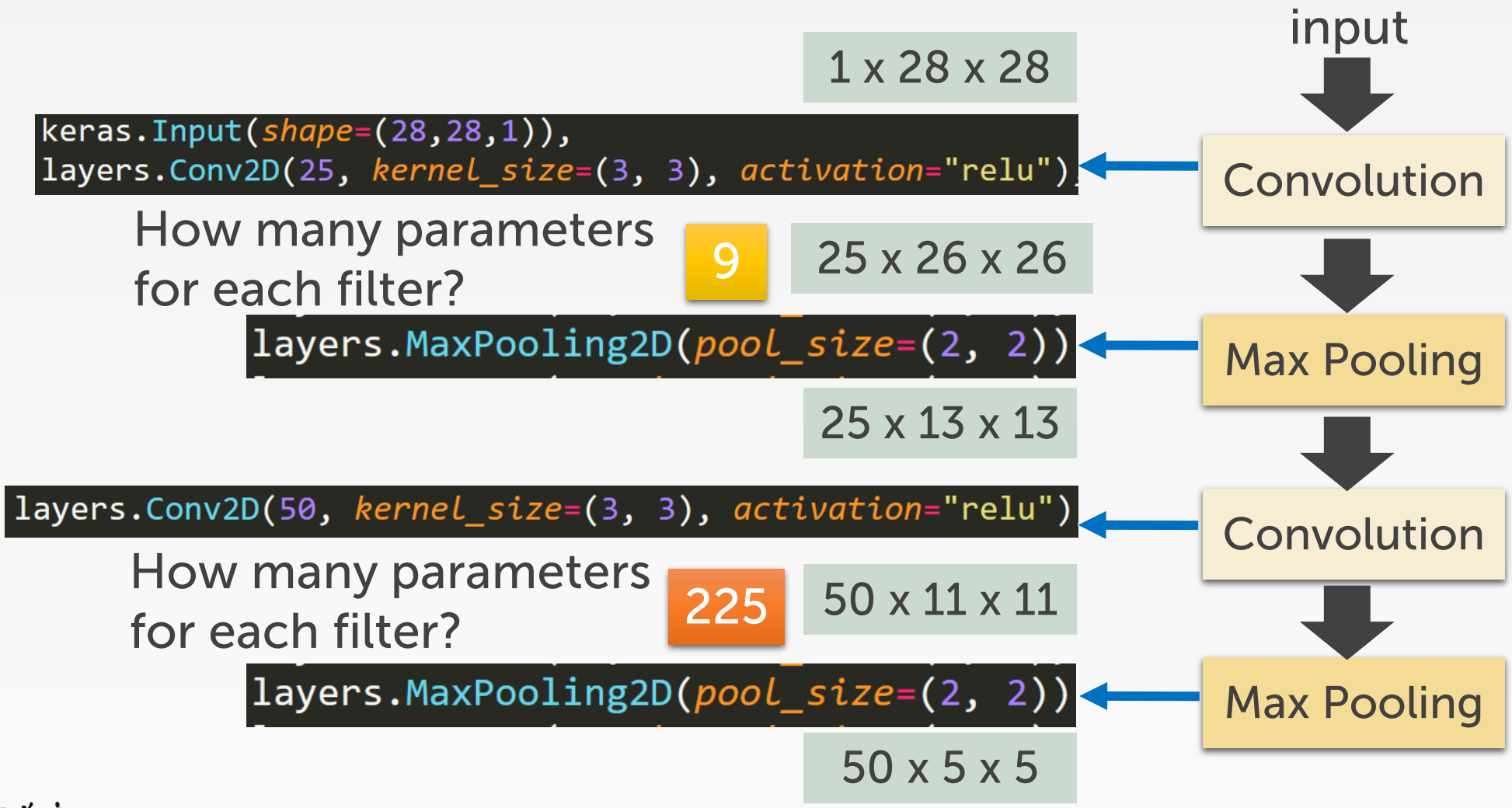
Implementation

```
layers.MaxPooling2D(pool_size=(2, 2))
```





Implementation



Implementation

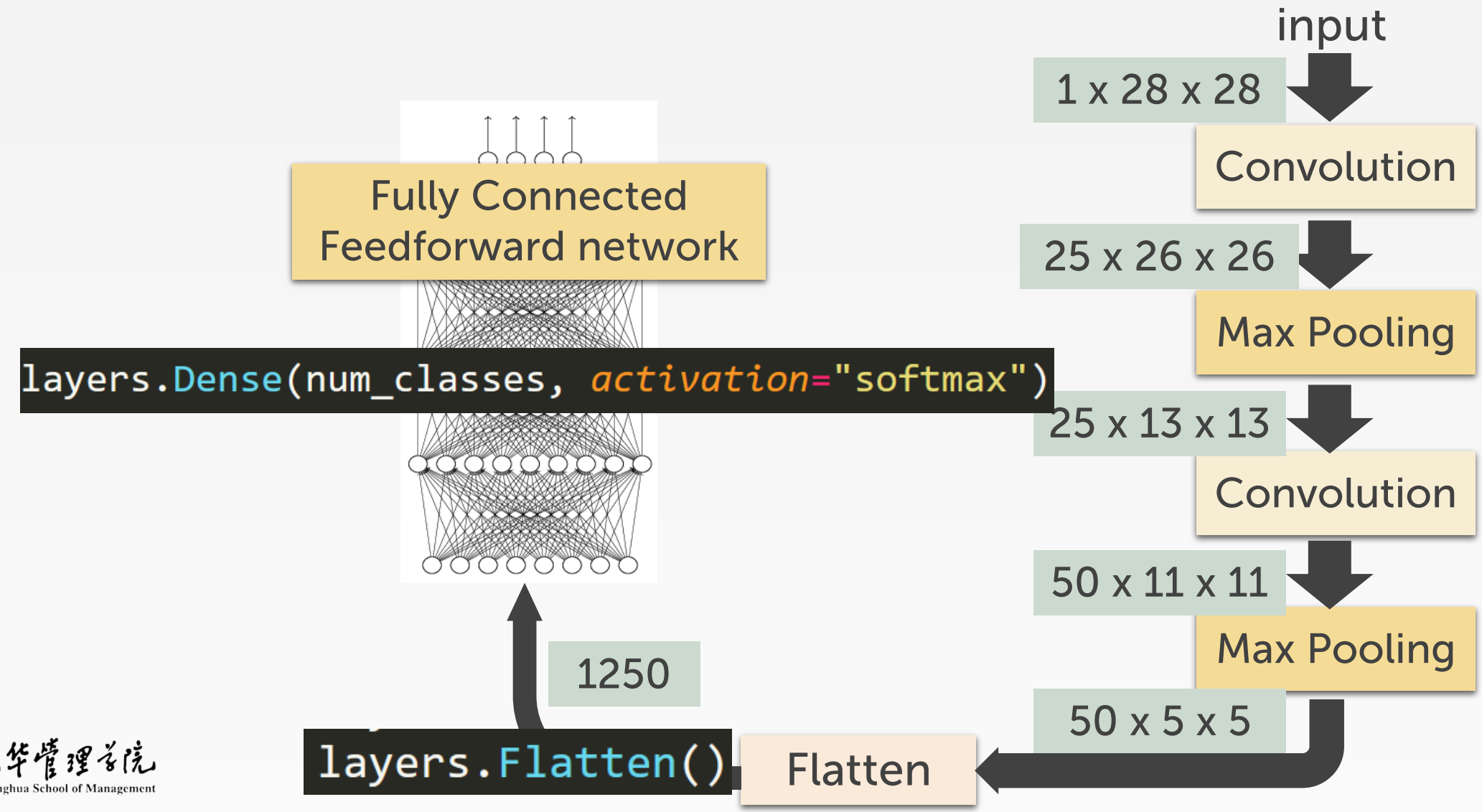


Figure 1. Comparison of Unverified and Verified Photos

Zhang, S., Lee, D., Singh, P.V. and Srinivasan, K.. How much is an image worth? Airbnb property demand estimation leveraging large scale image analytics. *Management Science (2021)*

12 Human-Interpretable Image Attributes				
Composition	1	*DIAGONAL_DOMINANCE*	2.516**	0.945
	2	*VISUAL_BALANCE_INTENSITY*	4.618***	1.350
	3	*VISUAL_BALANCE_COLOR*	8.869***	2.143
	4	*RULE_OF_THIRDS*	3.537**	1.106
Color	5	*WARM_HUE*	4.715*	2.363
	6	*SATURATION*	3.920*	1.927
	7	*BRIGHTNESS*	3.434*	1.679
	8	*CONTRAST_OF_BRIGHTNESS*	-4.897*	2.411
	9	*IMAGE_CLARITY*	6.212**	2.175
Figure-Ground Relationship	10	*SIZE_DIFFERENCE*	3.807*	1.541
	11	*COLOR_DIFFERENCE*	2.728*	1.372
	12	*TEXTURE_DIFFERENCE*	2.313*	1.090
Fixed Effect			Property	
Seasonality			City-Year-Month	
Num. Observations			76,901	
R-squared			0.6670	

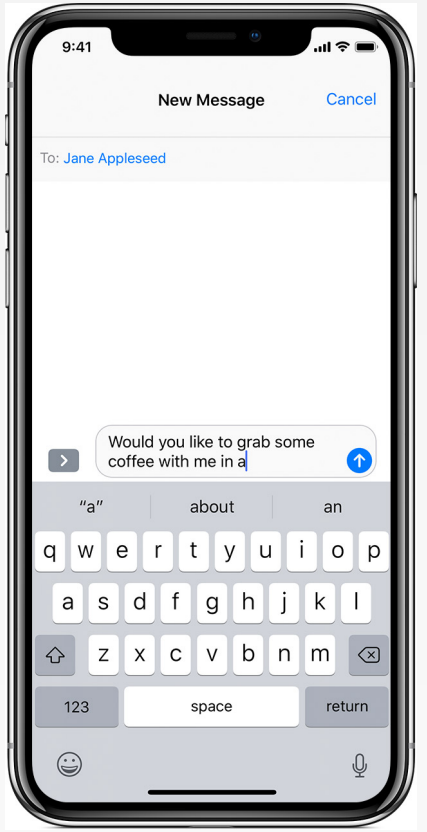
Note: *$p < 0.05$, **$p < 0.01$, ***$p < 0.001$.

Recurrent Neural Network

Source: Roi Yehoshua@CMU

Sequence of Data

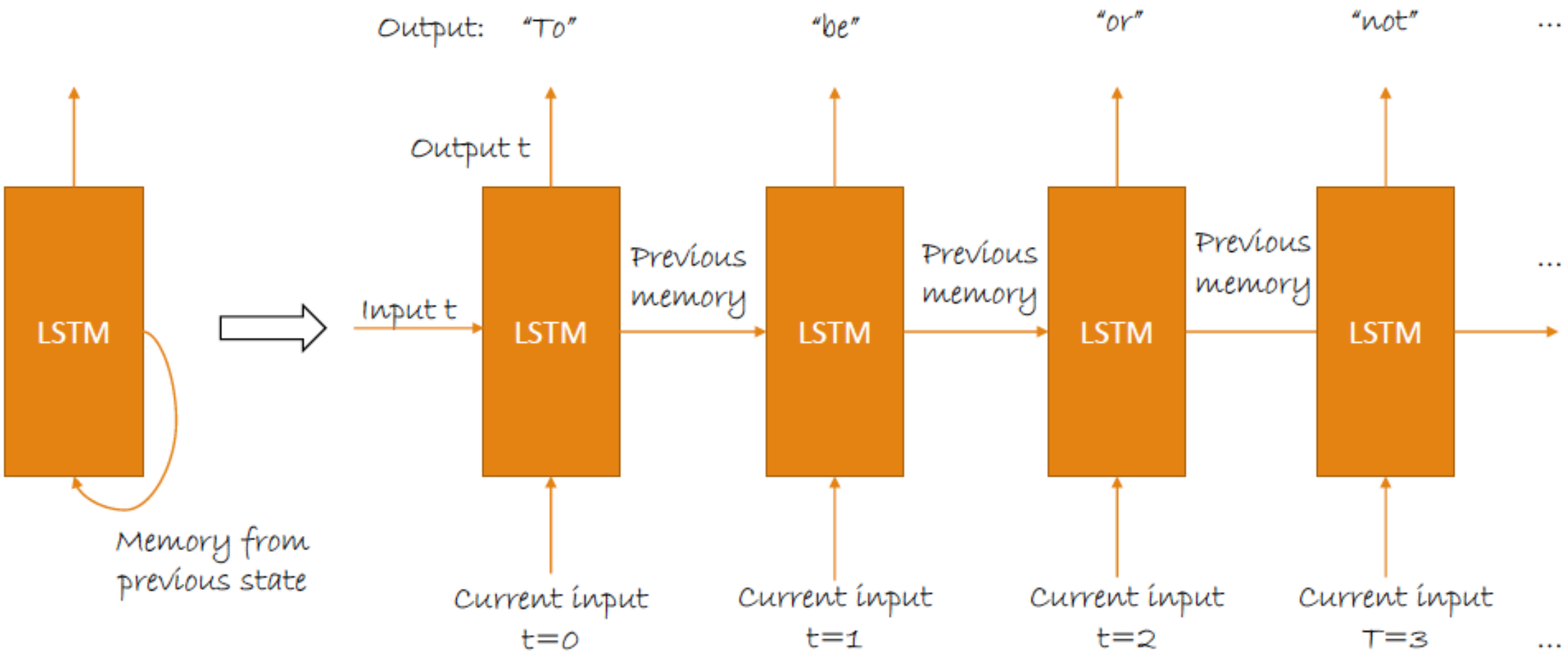
- Sequences in our world:
  - Audio
  - Text
  - Video
  - Weather
  - Stock market
- Sequential data is why we build RNN architectures
- RNNs are tools for making predictions about sequences



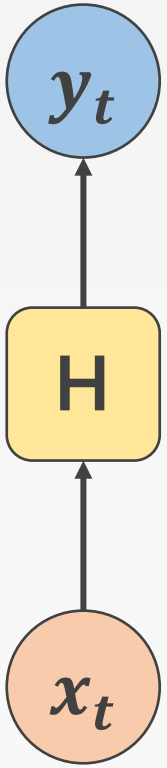
Limitations of Previous Networks

- Fixed Length:
  - Inputs and outputs are of fixed lengths
- Independence
  - Data (e.g., images) are independent of one another

Nutshell: RNN



Neural Network



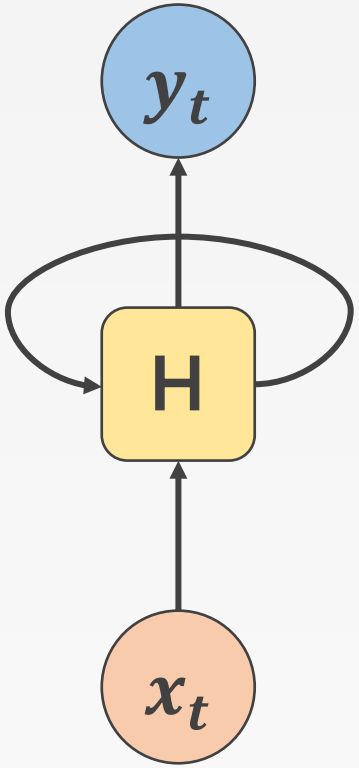
x_t: Input

y_t: Output/Prediction

H: chunk of NN

Every input is treated independently

RNN

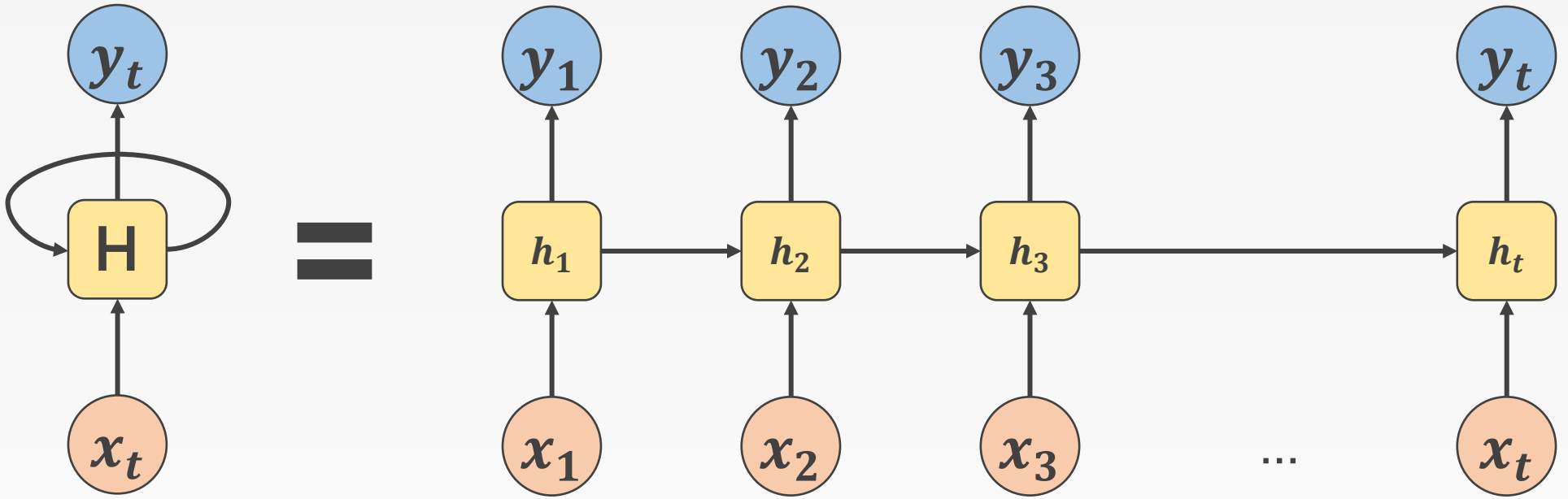


The loop allows information to be passed from one time step to the next.

Now we are modeling the dynamics

RNN

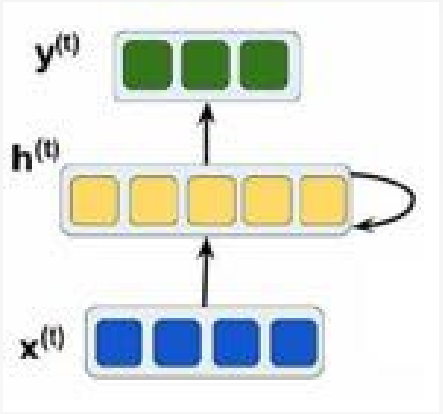
- A recurrent neural network can be thought of as multiple copies of the network, each passing a message to a successor.

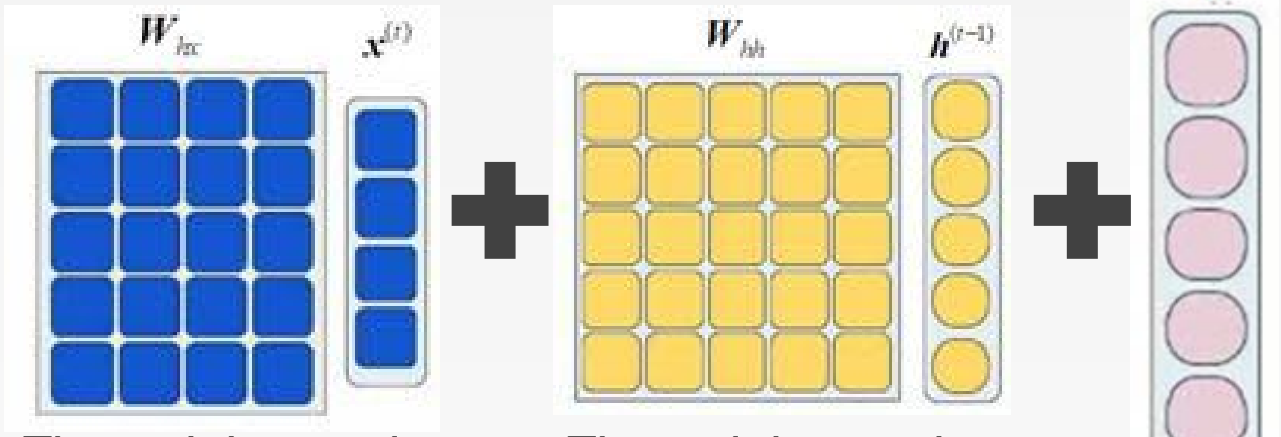


	x - input		y - output
Speech recognition		⟹	"Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair."
Music generation	∅	⟹	
Sentiment classification	"Decent effort. The plot could have been better."	⟹	
DNA sequence analysis	ACTGTACCCATGTGACTGCCC	⟹	ACTGTACCCATGTGACTGCCC
Machine translation	"El que no arriesga, no gana."	⟹	"If you don't take risks, you cannot win."
Video activity recognition		⟹	Running
Name entity recognition	"Ygritte says Jon Snow knows nothing."	⟹	"Ygritte says Jon Snow knows nothing."

RNN Structure

- New hidden state: $h^{(t)} = \phi_h\left(W_{hx}x^{(t)} + W_{hh}h^{(t-1)} + b_h\right)$





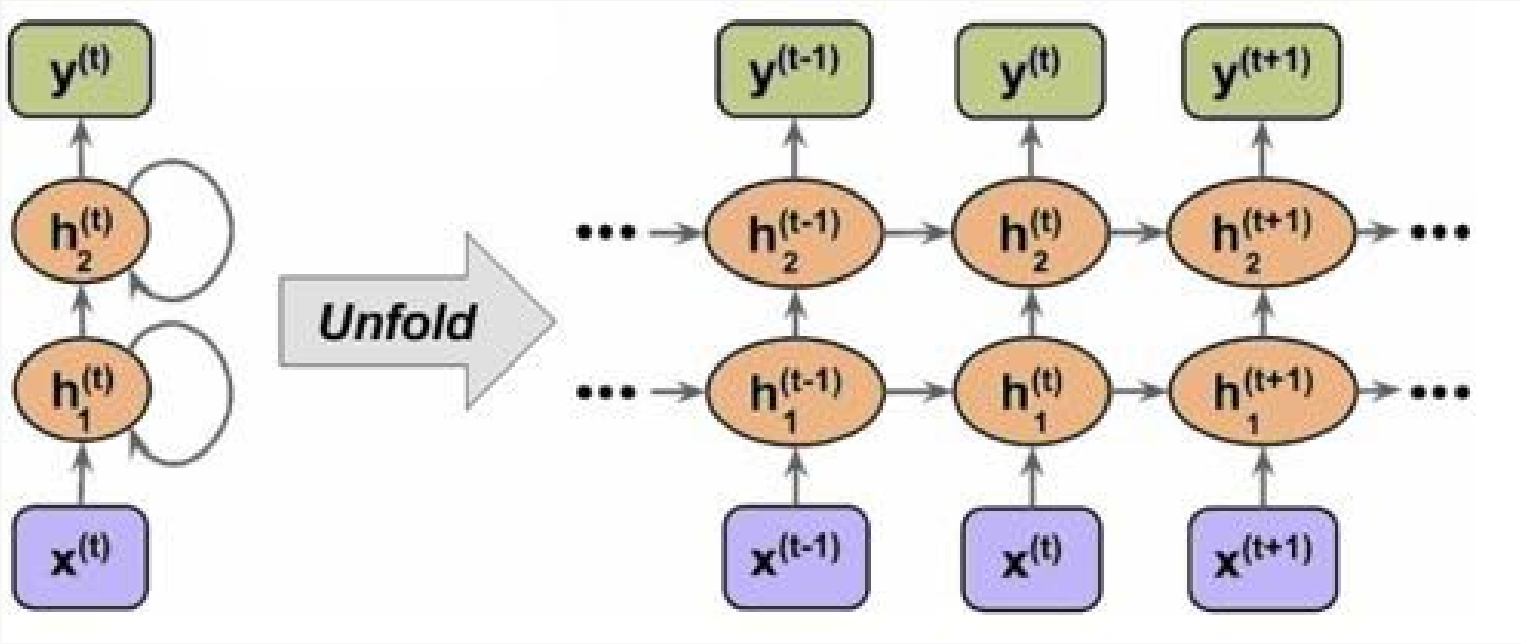
The weight matrix between the input and the hidden layer

The weight matrix associated with the recurrent edges

Bias vector for the hidden units

- Output: $y^{(t)} = \phi_y\left(W_{yh}h^{(t)} + b_y\right)$

Multilayer RNN



Deep RNNs

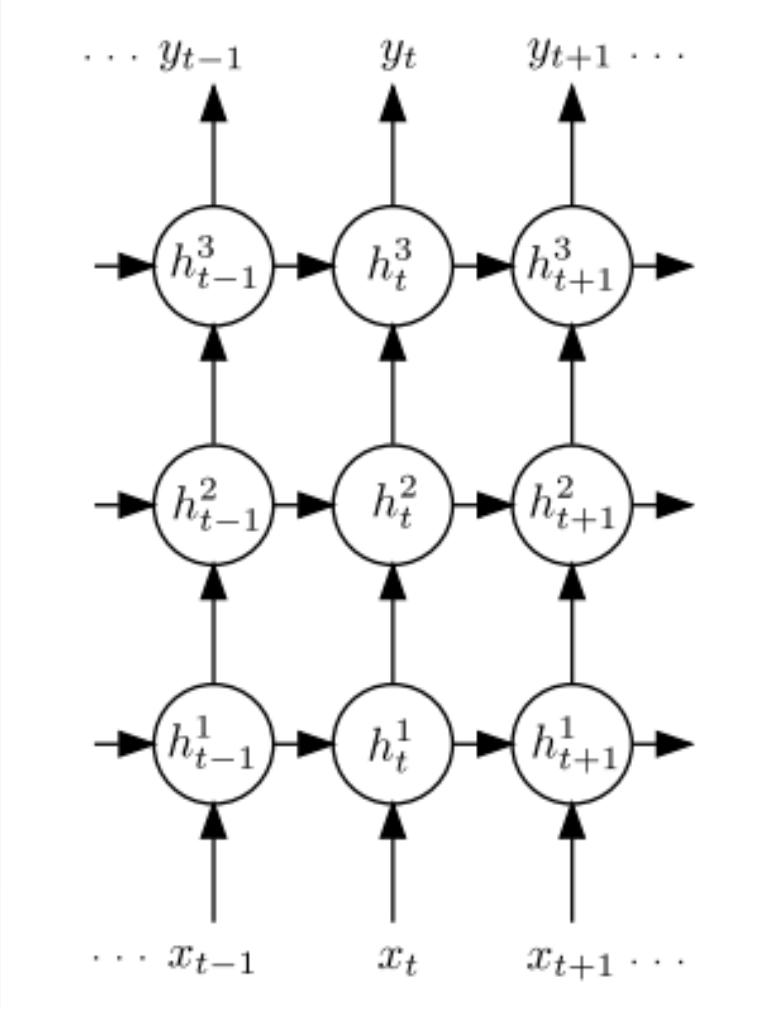


Figure from (Graves et al., 2013)

Long Short-Time Memory

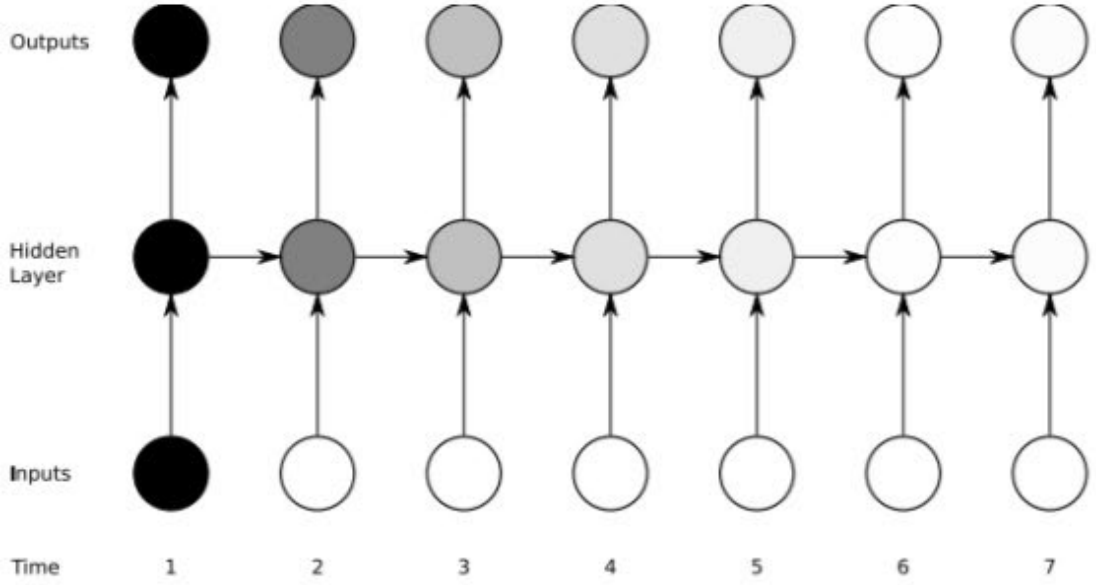
- Motivation:
  - Standard RNNs have trouble learning long distance dependencies

EX #1: The cat, which already ate a bunch of food, was full.
EX #2: The cats, which already ate a bunch of food, were full.

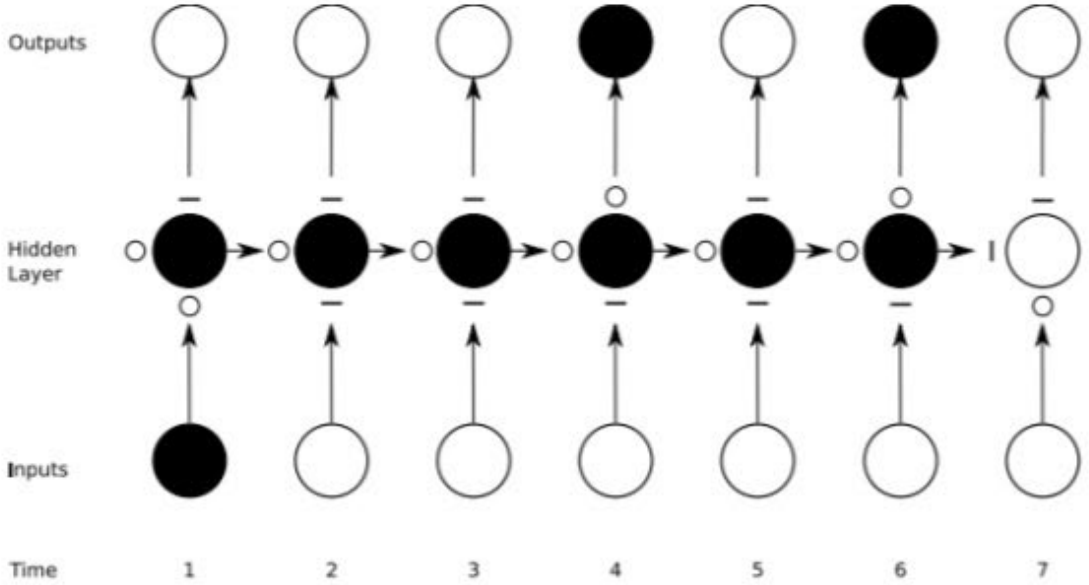
- Motivation:
  - Vanishing gradient problem for standard RNNs

LSTMs reduce vanishing gradient problem

Standard Recurrent Network



LSTM Network



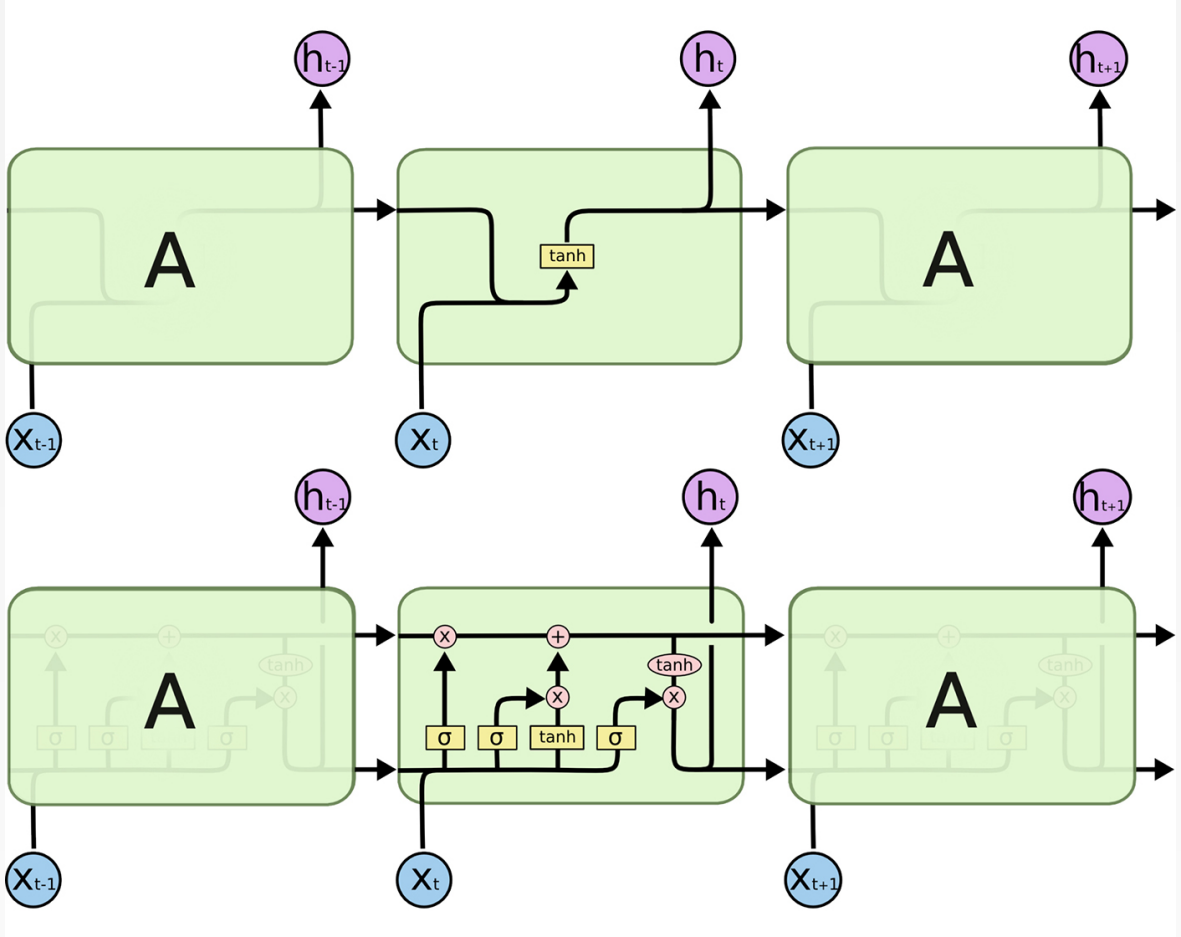
Graves et al 2013

- The darker the shade, the greater the sensitivity

The various “gates” determine the propagation of information and can choose to “remember” or “forget” information

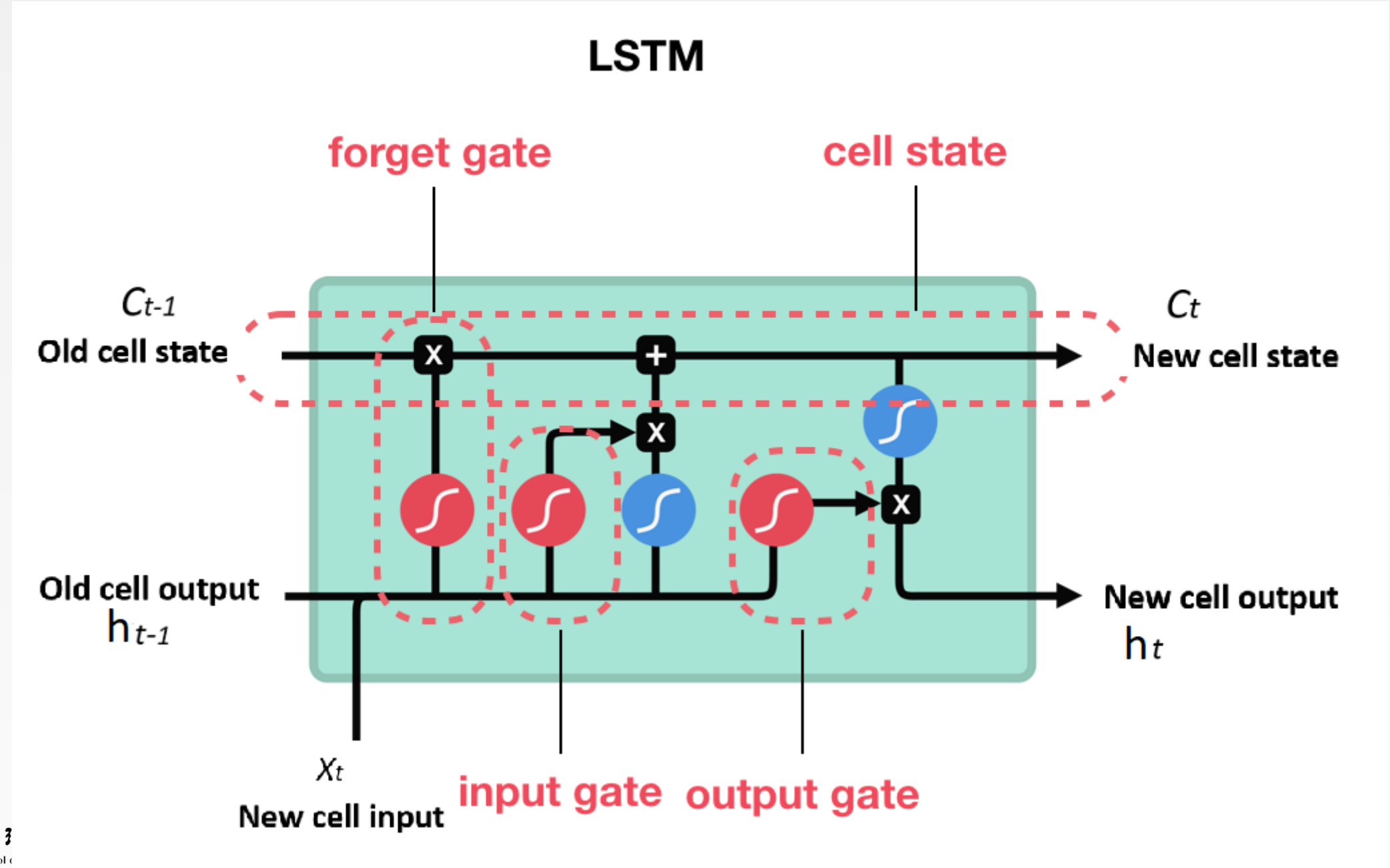
LSTM (Long Short Term Memory)

- LSTM is a RNN architecture
- Suggested in 1997 by Hochreiter and Schmidhuber as a solution to the vanishing gradient problem
- An LSTM cell stores a value (state) for either long or short time periods



Gate

- Sigmoid: outputs numbers between:
  - Zero “let nothing through” and
  - One “let everything through!”


LSTM
forget gate
cell state
C_{t-1}
Old cell state
C_t
New cell state
Old cell output
h_{t-1}
New cell output
h_t
X_t
New cell input
input gate
output gate

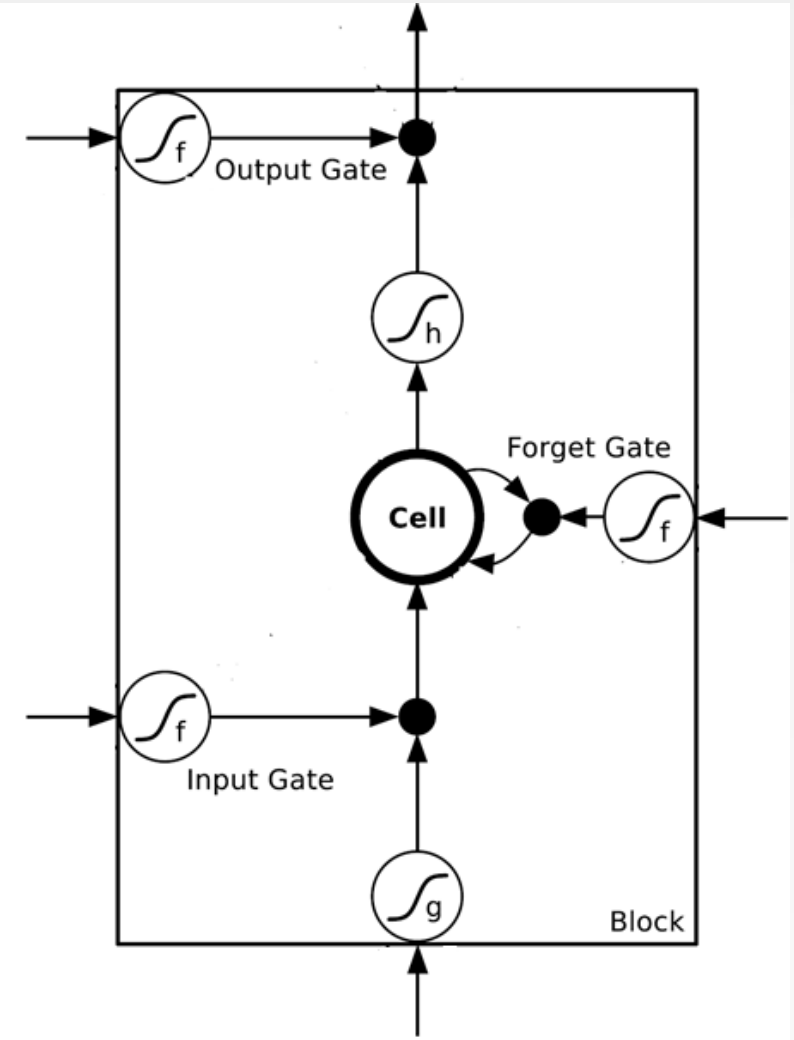
LSTM - Example

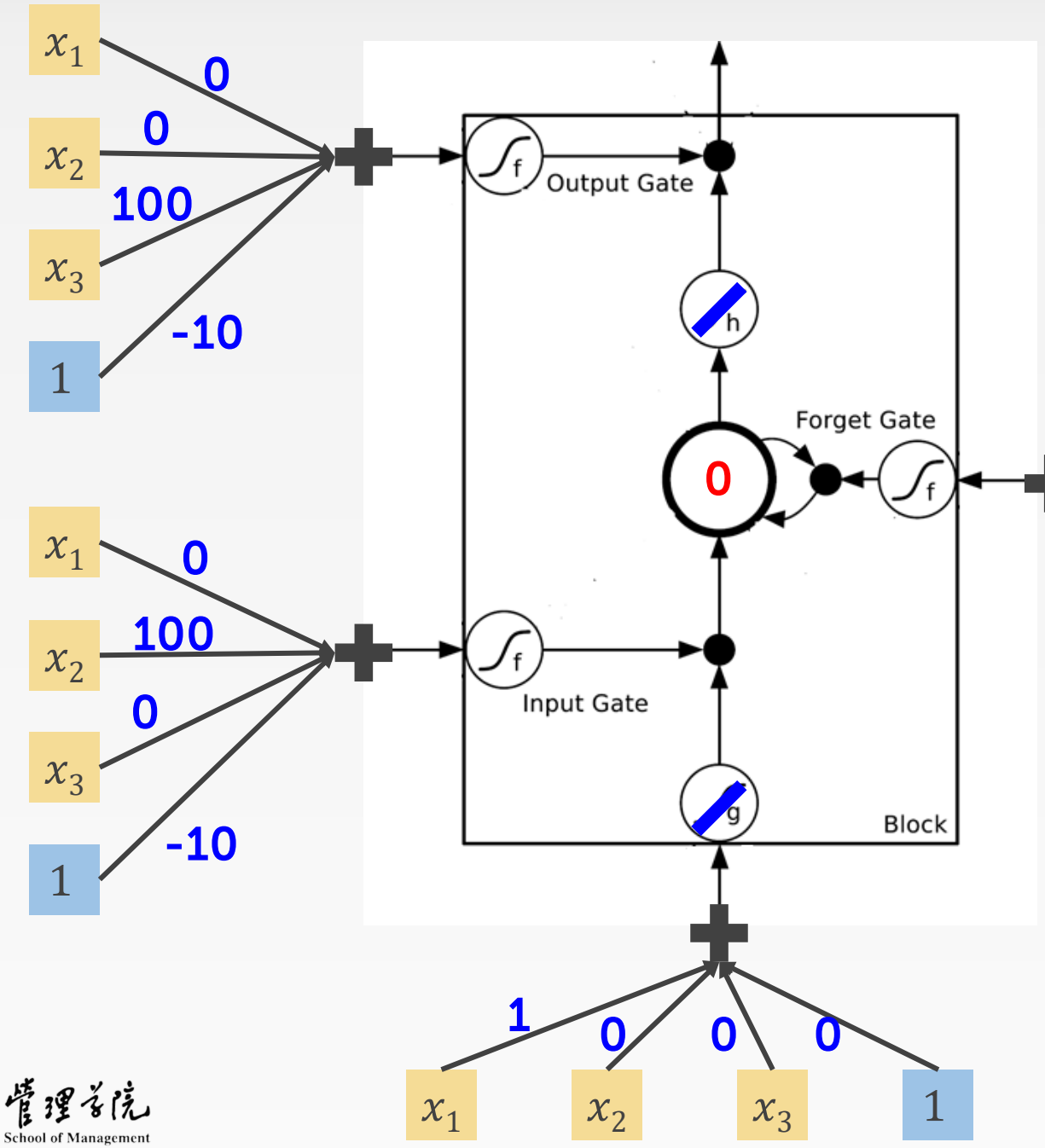
	0	0	3	3	7	7	7	0	6
x_1	1	3	2	4	2	1	3	6	1
x_2	0	1	0	1	0	0	-1	1	0
x_3	0	0	0	0	0	1	0	0	1
y	0	0	0	0	0	7	0	0	6

When x_2 = 1, add the numbers of x_1 into the memory

When x_2 = -1, reset the memory

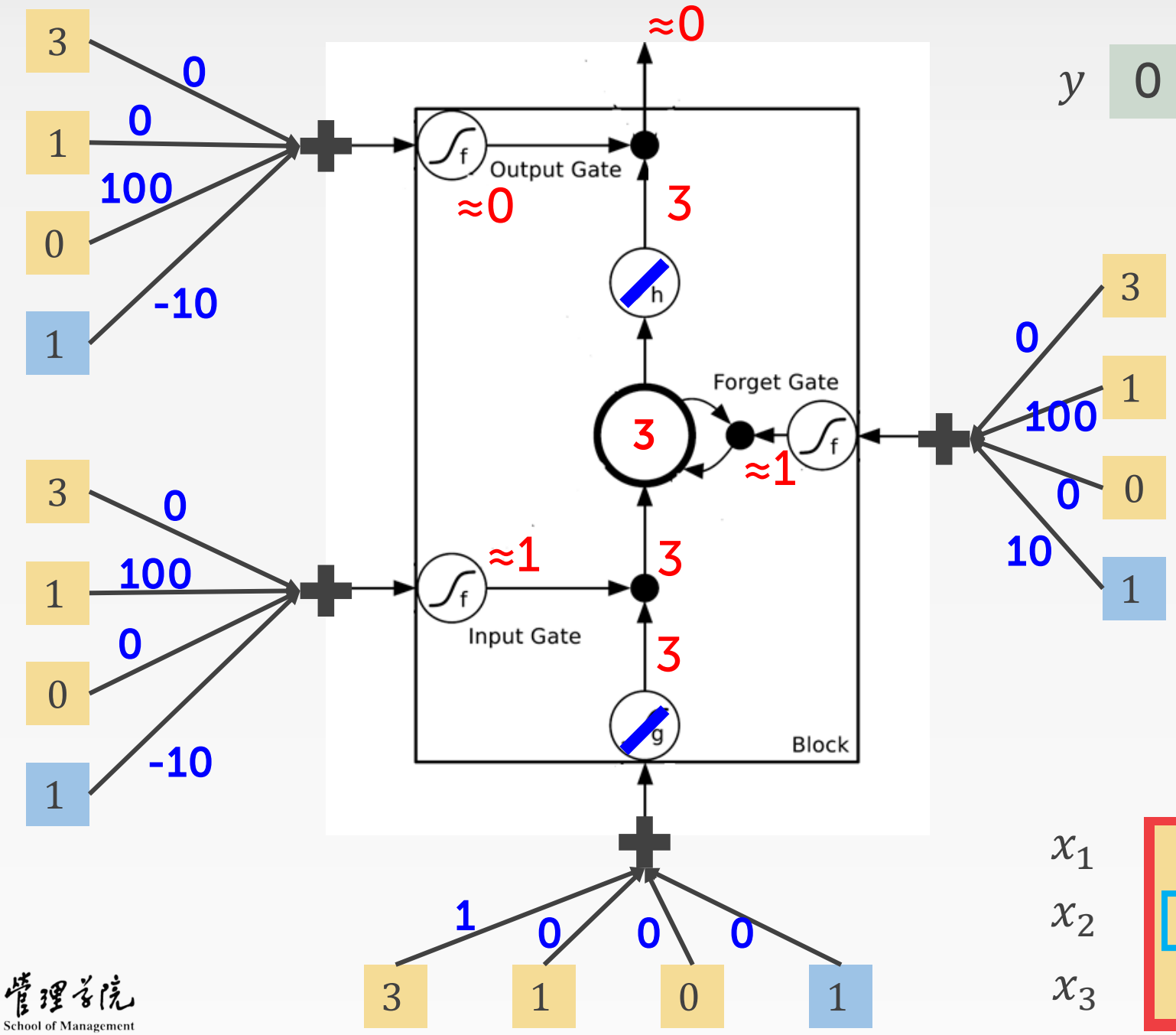
When x_3 = 1, output the number in the memory.


f
Output Gate
h
Forget Gate
Cell
f
f
Input Gate
g
Block



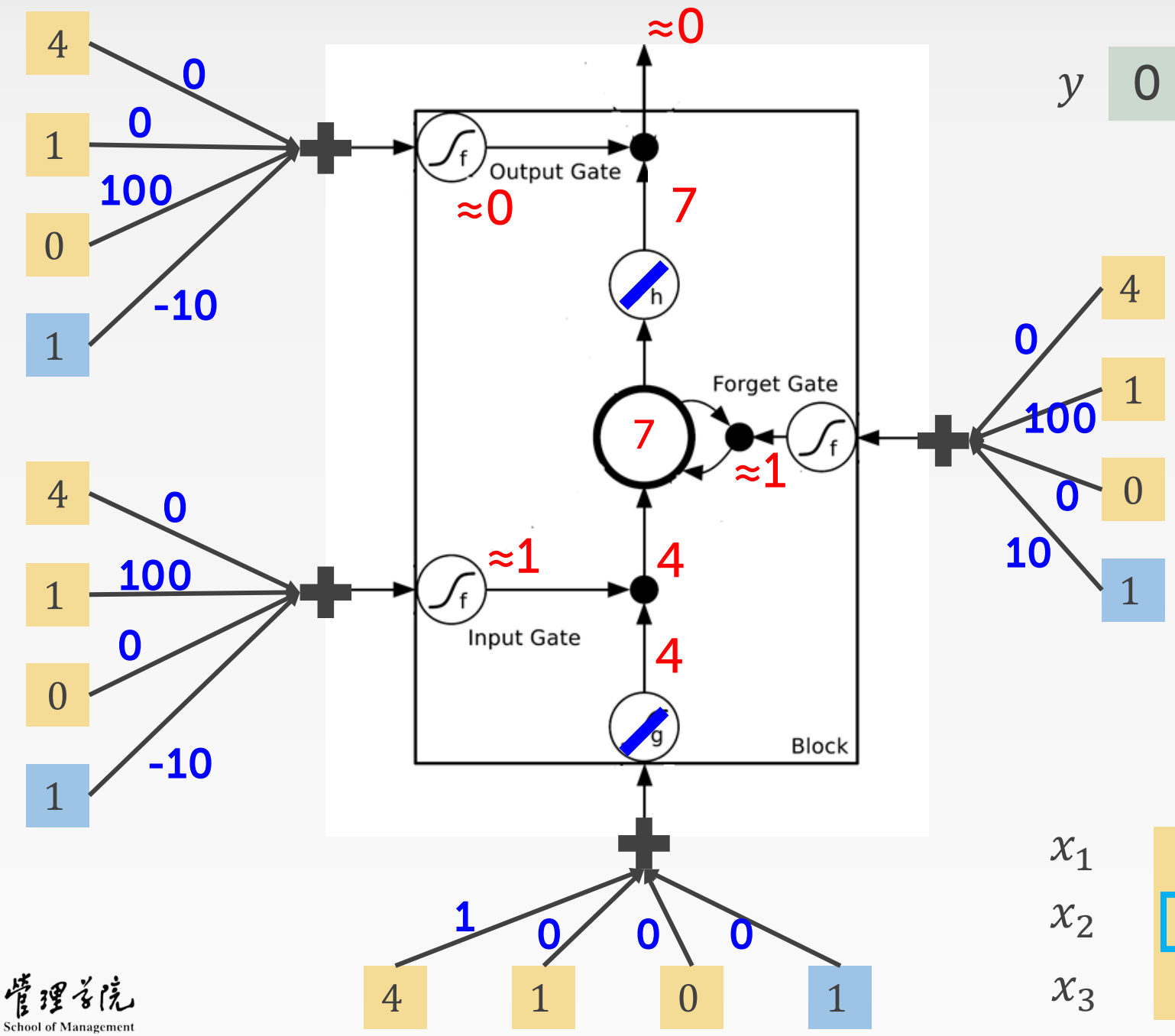
y	0	0	0	7	0

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0



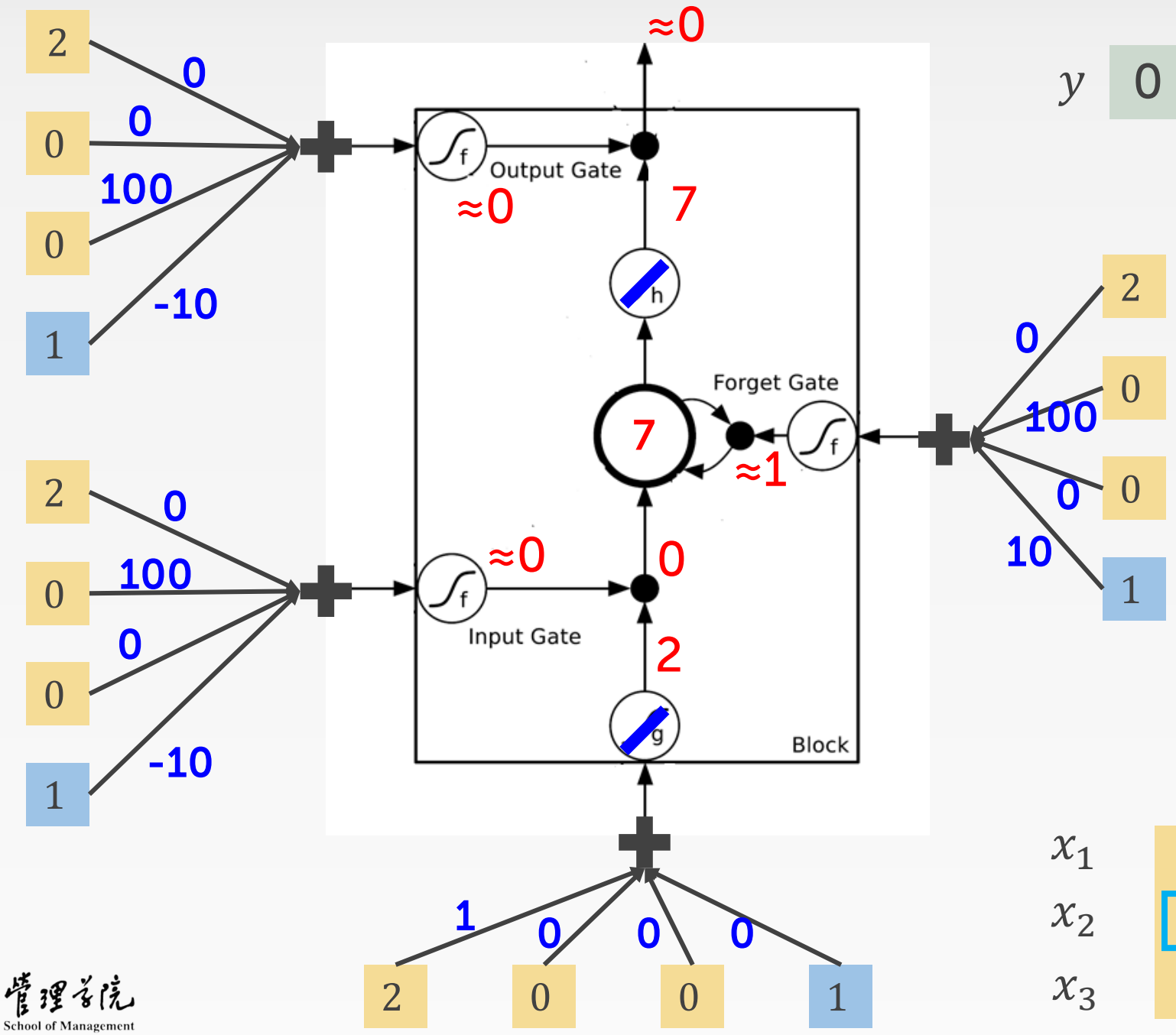
y 0 0 0 7 0

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0



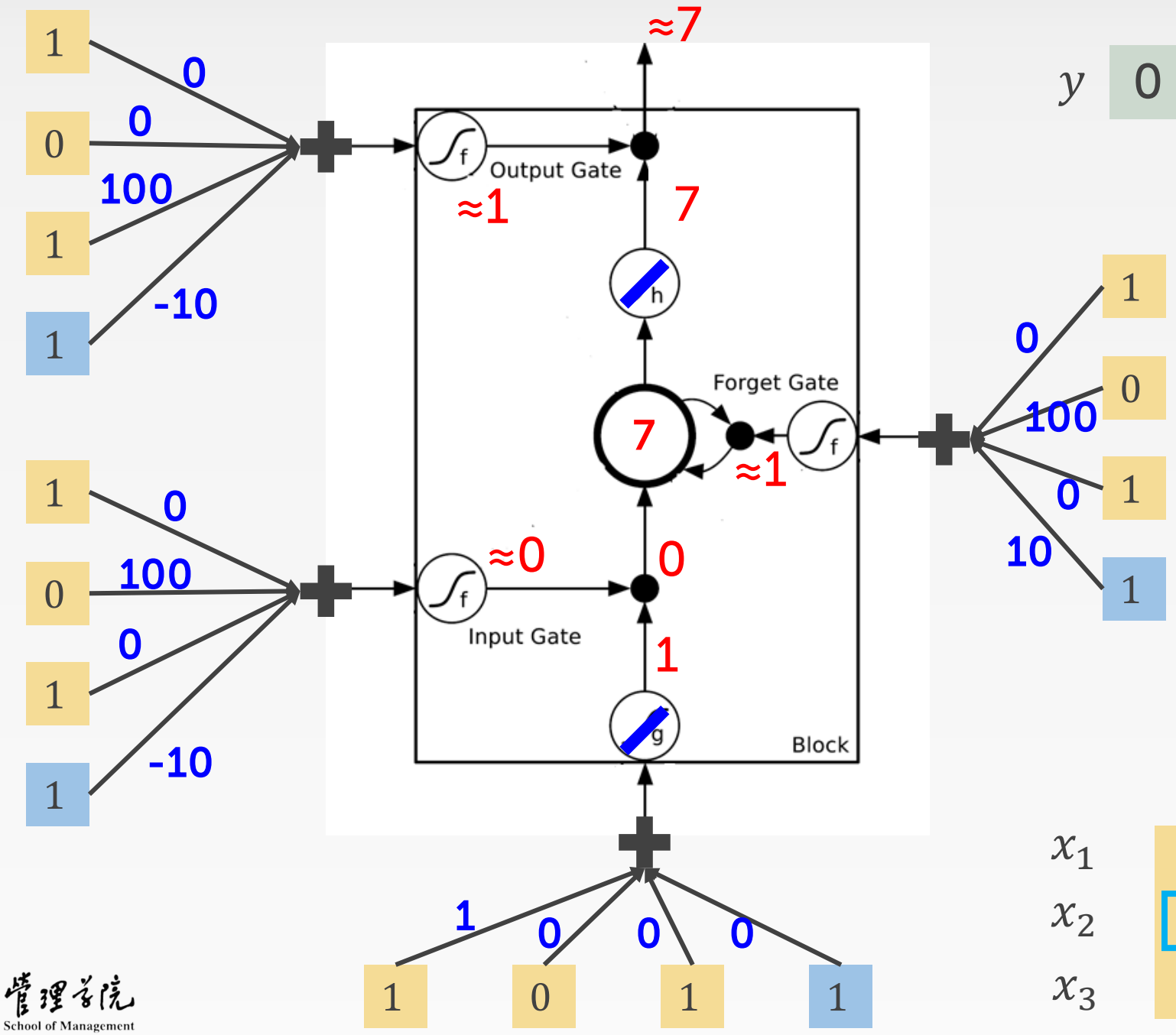
y	0	0	0	7	0

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0



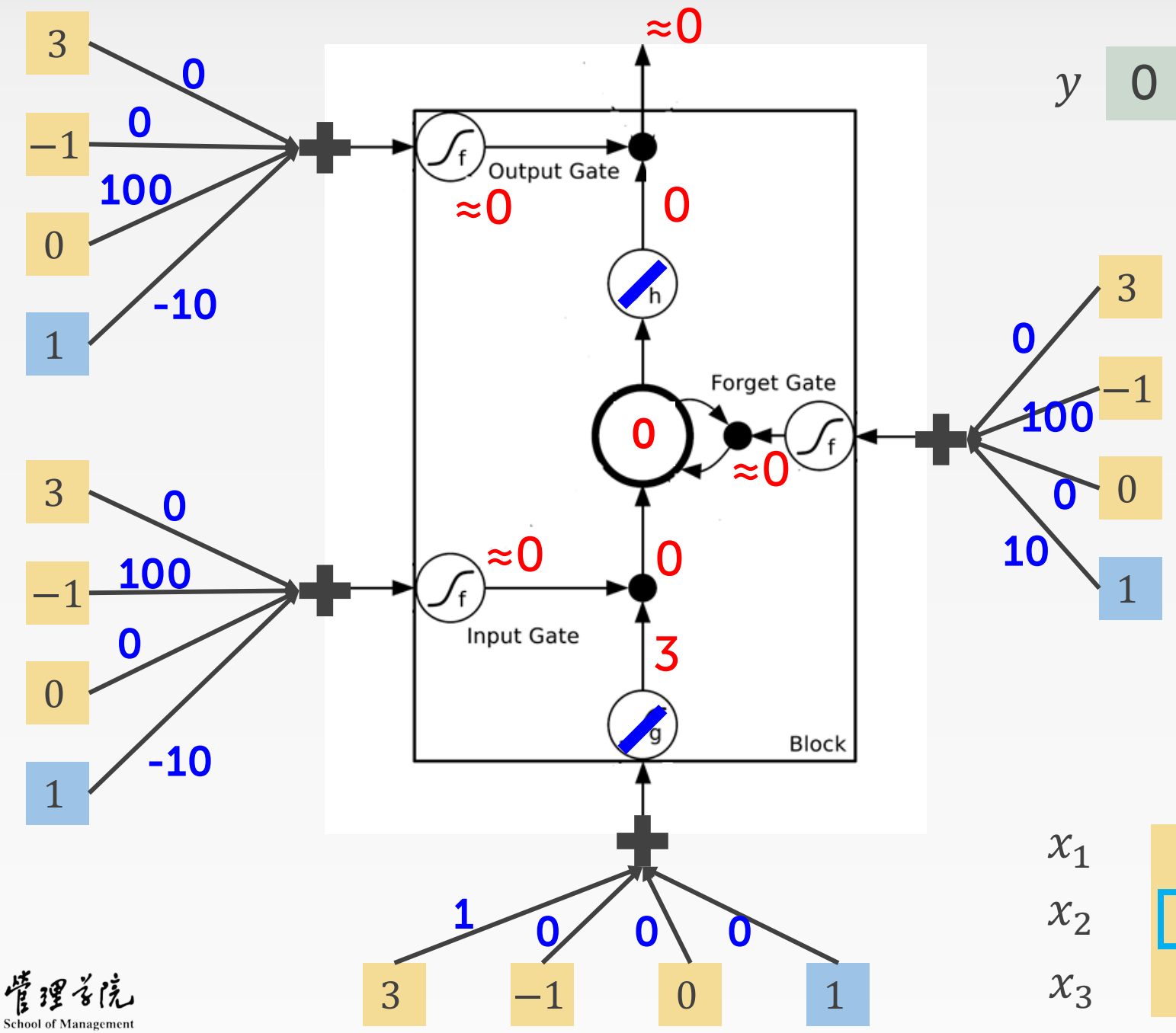
y 0 0 0 7 0

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0



y	0	0	0	7	0

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0



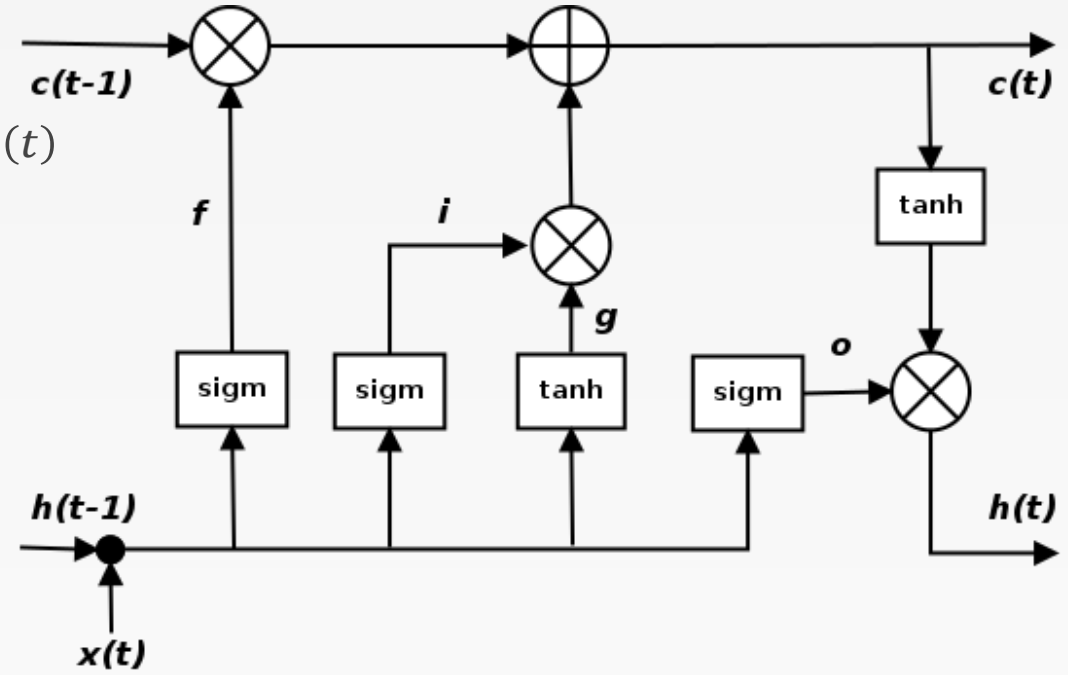
y 0 0 0 7 0

x_1	3	4	2	1	3
x_2	1	1	0	0	-1
x_3	0	0	0	1	0

LSTM Computations

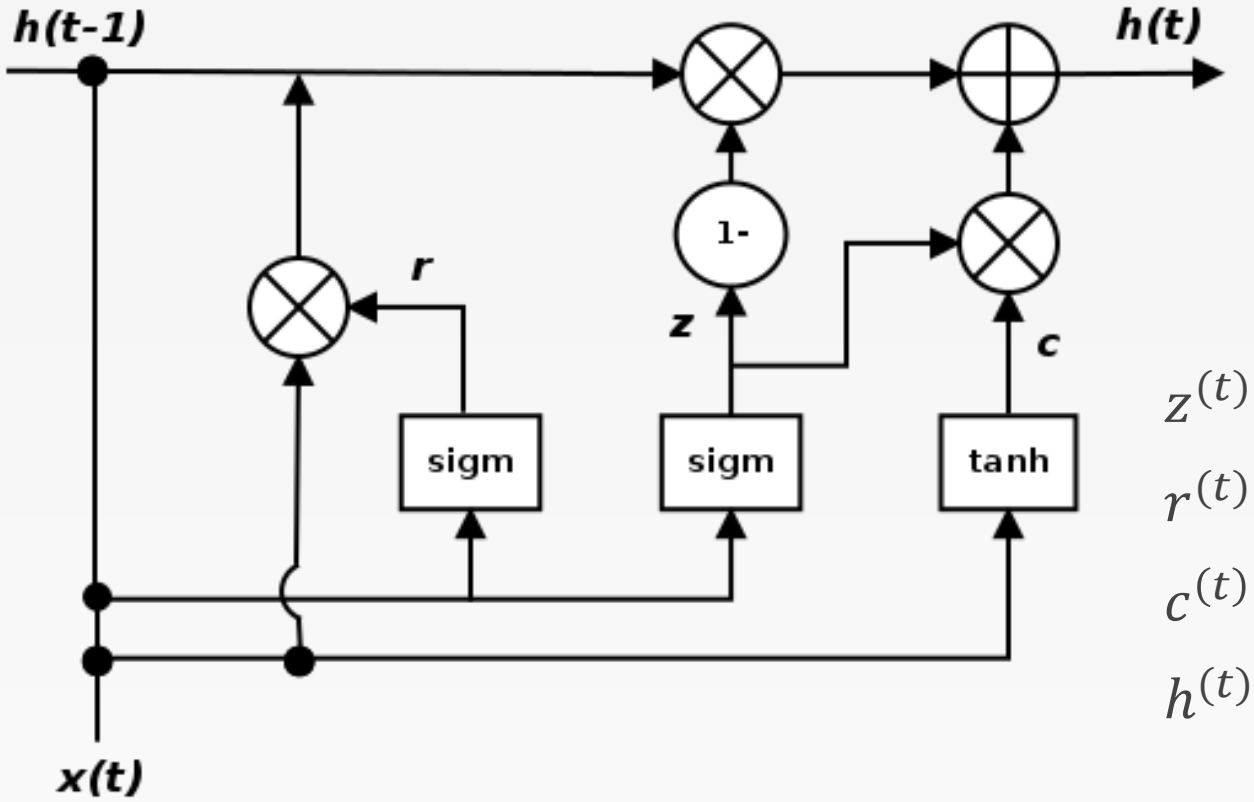
- Forget gate: $f^{(t)} = \sigma\left(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f\right)$
- Input gate: $i^{(t)} = \sigma\left(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i\right)$
- Output gate: $o^{(t)} = \sigma\left(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o\right)$
- $g^{(t)} = \tanh\left(W_{gx}x^{(t)} + W_{gh}h^{(t-1)} + b_g\right)$
- New cell gate: $c^{(t)} = f^{(t)} \otimes c^{(t-1)} + i^{(t)} \otimes g^{(t)}$
- $y^{(t)} = h^{(t)} = o^{(t)} \otimes \tanh\left(c^{(t)}\right)$

σ is the sigmoid function
$\otimes$ refers to element-wise product



Gated Recurrent Unit (GRU)

- Similar performance as LSTM with less computation
- They have fewer parameters than LSTM, as they lack an output gate



$$z^{(t)} = \sigma\left(W_{zx}x^{(t)} + W_{zh}h^{(t-1)} + b_z\right)$$

$$r^{(t)} = \sigma\left(W_{rx}x^{(t)} + W_{rh}h^{(t-1)} + b_r\right)$$

$$c^{(t)} = \tanh\left(W_{cx}x^{(t)} + W_{ch}(r^{(t)} \otimes h^{(t-1)}) + b_c\right)$$

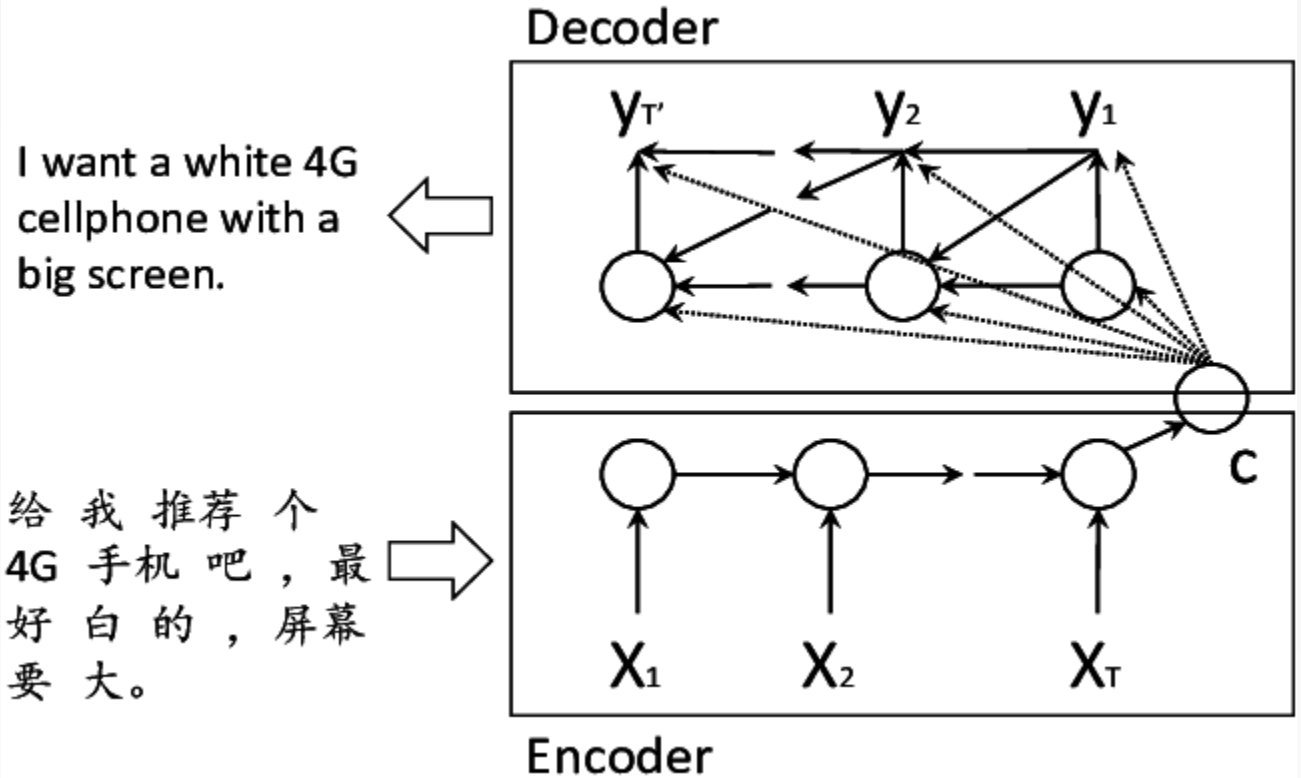
$$h^{(t)} = z^{(t)} \otimes h^{(t-1)} + (1 - z^{(t)}) \otimes c^{(t)}$$

RNN Applications

- Stock Price Prediction using LSTM
  - Project
  - Python Demo
- Machine Translation
- Visual Question Answering

Machine Translation

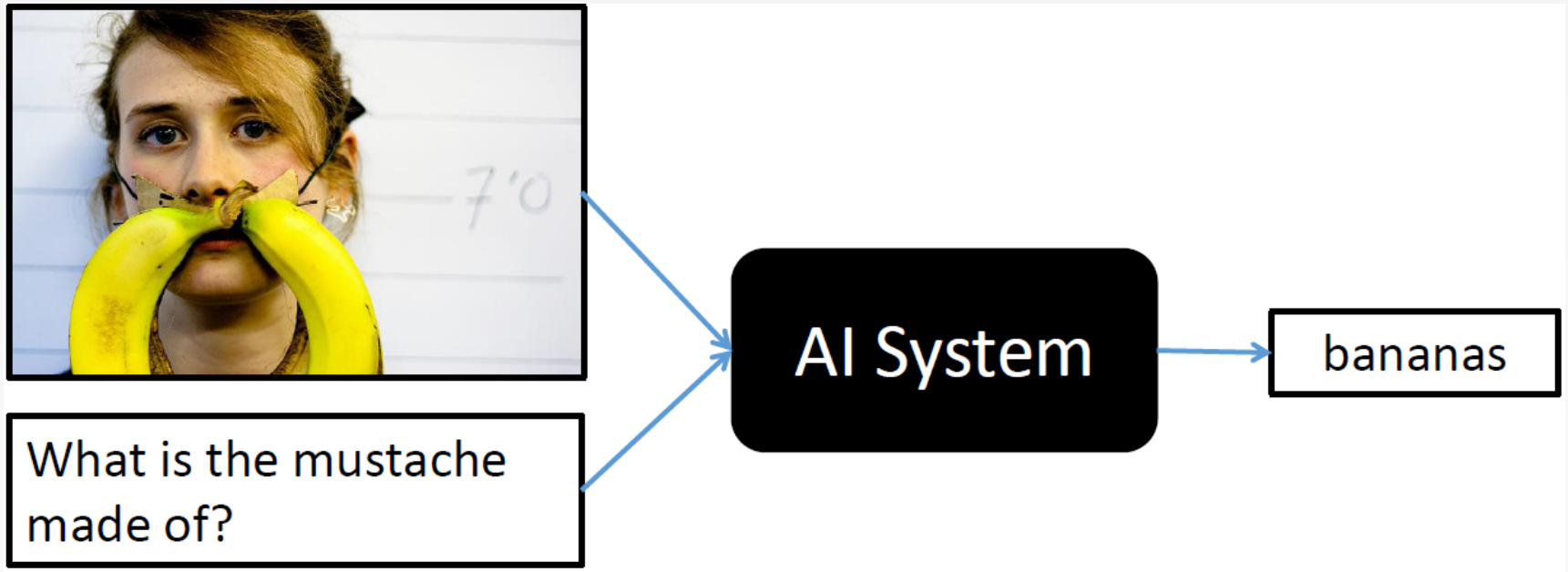
- In machine translation, the input is a sequence of words in source language, and the output is a sequence of words in target language.



Visual Question Answering

- VQA: Given an image and a natural language question about the image, the task is to provide an accurate natural language answer

DEMO



Picture from (Antol et al., 2015)