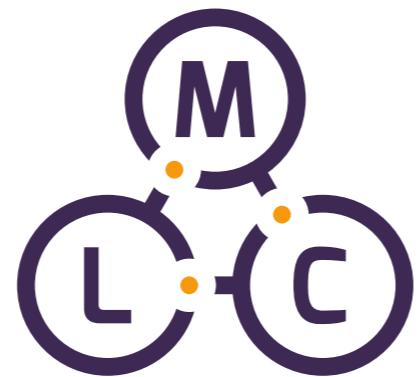


# Advanced Deep Learning Techniques

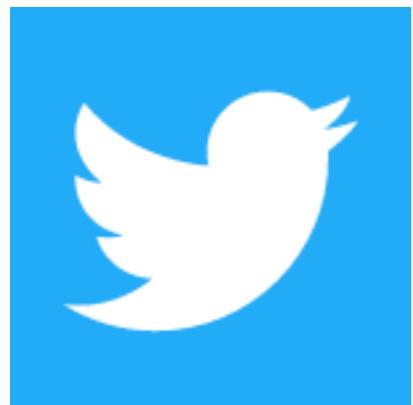
Jiří Materna



Machine  
Learning  
College



@mlcollegecom



@mlcollegecom



#mlcollege

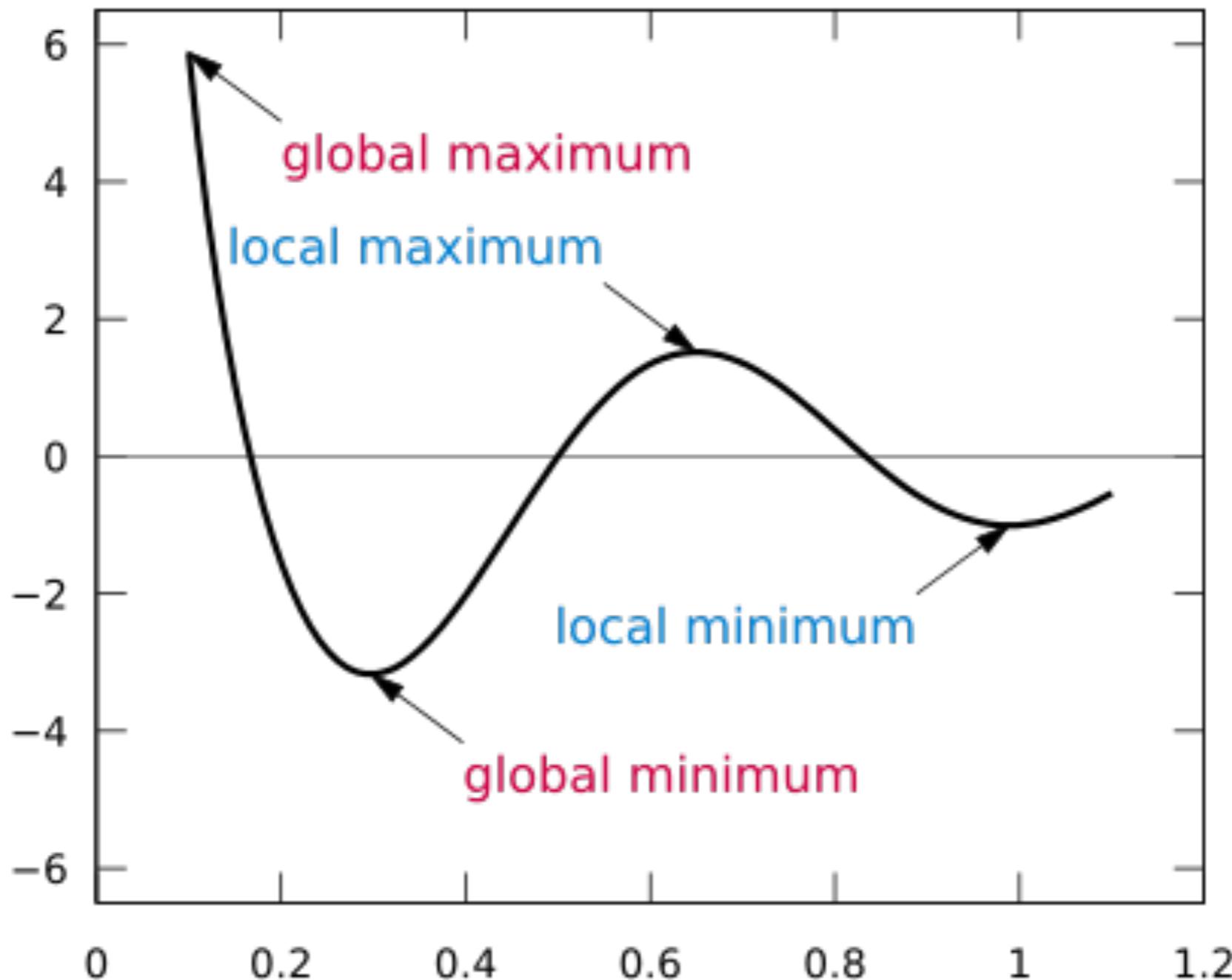
# About me

- Ph.D. in Natural Language Processing and Artificial Intelligence at Masaryk University
- 10 years at seznam.cz (last 8 years as Head Of Research)
- Founder and co-organizer of ML Prague
- Mentor at StartupYard
- ML Freelancer and consultant

# Outline

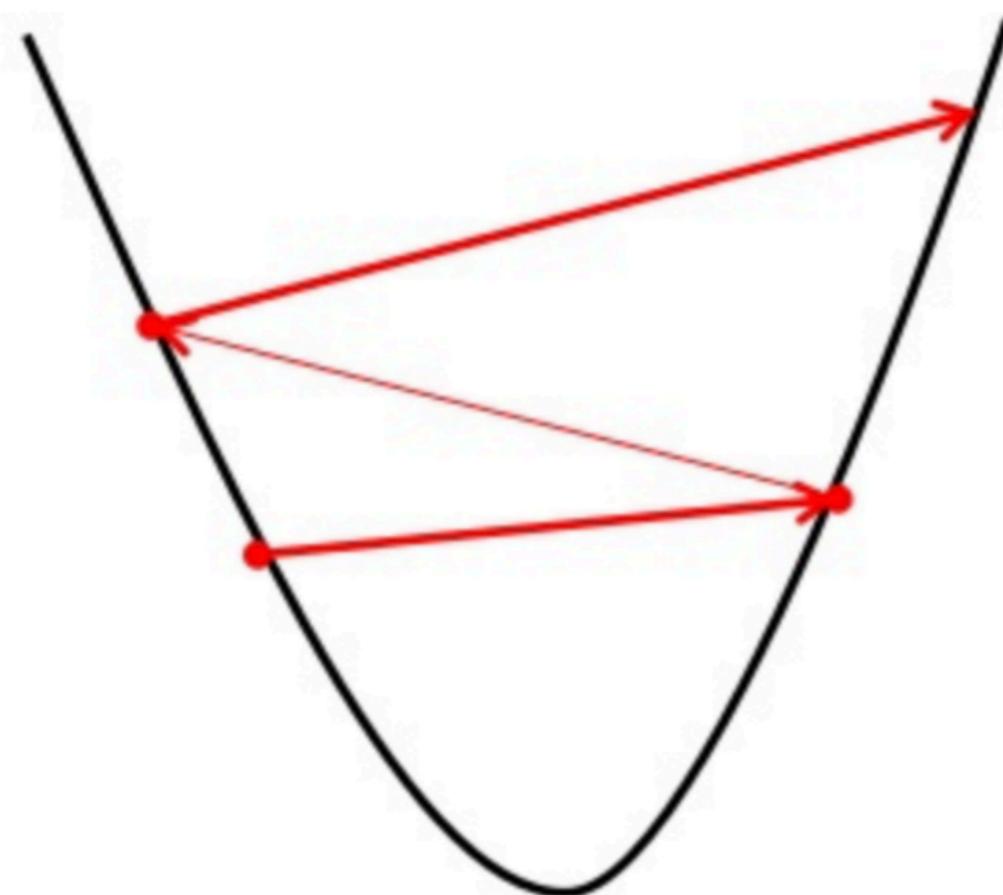
- Optimizers and their evolution
- Loss functions and their properties
- Practical classification task
- Practical regression task
- Initialization of weights in neural networks
- Normalization and Regularization in neural networks
- Functional model definition in Keras
- Semisupervised learning
- Practical example of a complex neural network
- Selected Neural Network architectures

# Parameter optimization strategies

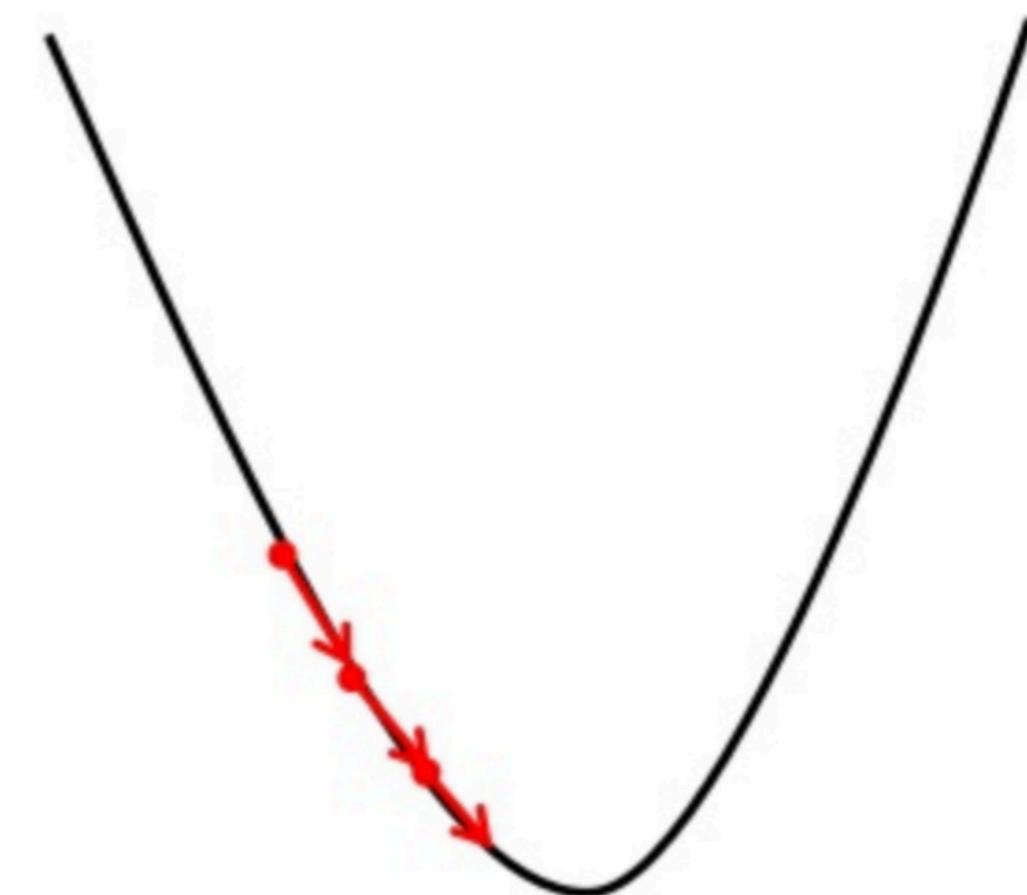


# Learning rate tuning

Big learning rate



Small learning rate



# Gradient Descent Variants

**(Batch) Gradient Descent**

$$w_{t+1} = w_t - \lambda \frac{\partial e(X, y)}{\partial w_t}$$

**Stochastic Gradient Descent**

$$w_{t+1} = w_t - \lambda \frac{\partial e(X^i, y^i)}{\partial w_t}$$

**Mini-Batch Gradient Descent**

$$w_{t+1} = w_t - \lambda \frac{\partial e(X^{(i,i+n)}, y^{(i,i+n)})}{\partial w_t}$$

# Momentum and Nesterov Accelerated Gradient

$$v_t = \gamma v_{t-1} + \lambda \frac{\partial e(w_t)}{\partial w_t}$$

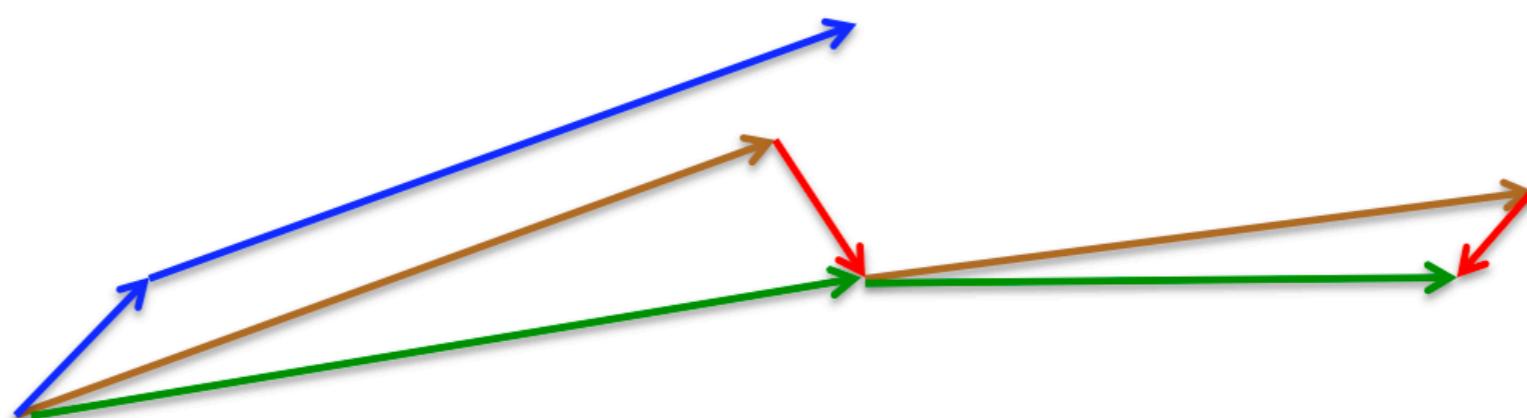
**Naive momentum**

$$w_{t+1} = w_t - v_t$$

**Nesterov Accelerated Gradient**

$$v_t = \gamma v_{t-1} + \lambda \frac{\partial e(w_t - \gamma v_{t-1})}{\partial w_t}$$

$$w_{t+1} = w_t - v_t$$



# Adaptive Gradient Algorithms

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\sum_{i=1}^t g_i^2 + \epsilon}} g_t$$

**Adagrad**

$$g_i = \frac{\partial e(w_i)}{\partial w_i}$$

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{\mathbb{E}[g^2]_t - \epsilon}} g_t^2$$

**RMSProp**

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1 - \gamma) g_t^2$$

# Adam and Nadam

**Adam**

Combination of RMSProp with momentum

**Nadam**

Combination of RMSProp with Nesterov momentum

# Loss functions for deep learning

**Mean Squared Error**

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

**Mean Absolute Error**

$$MSE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

# Cross Entropy (Negative Log Likelihood)

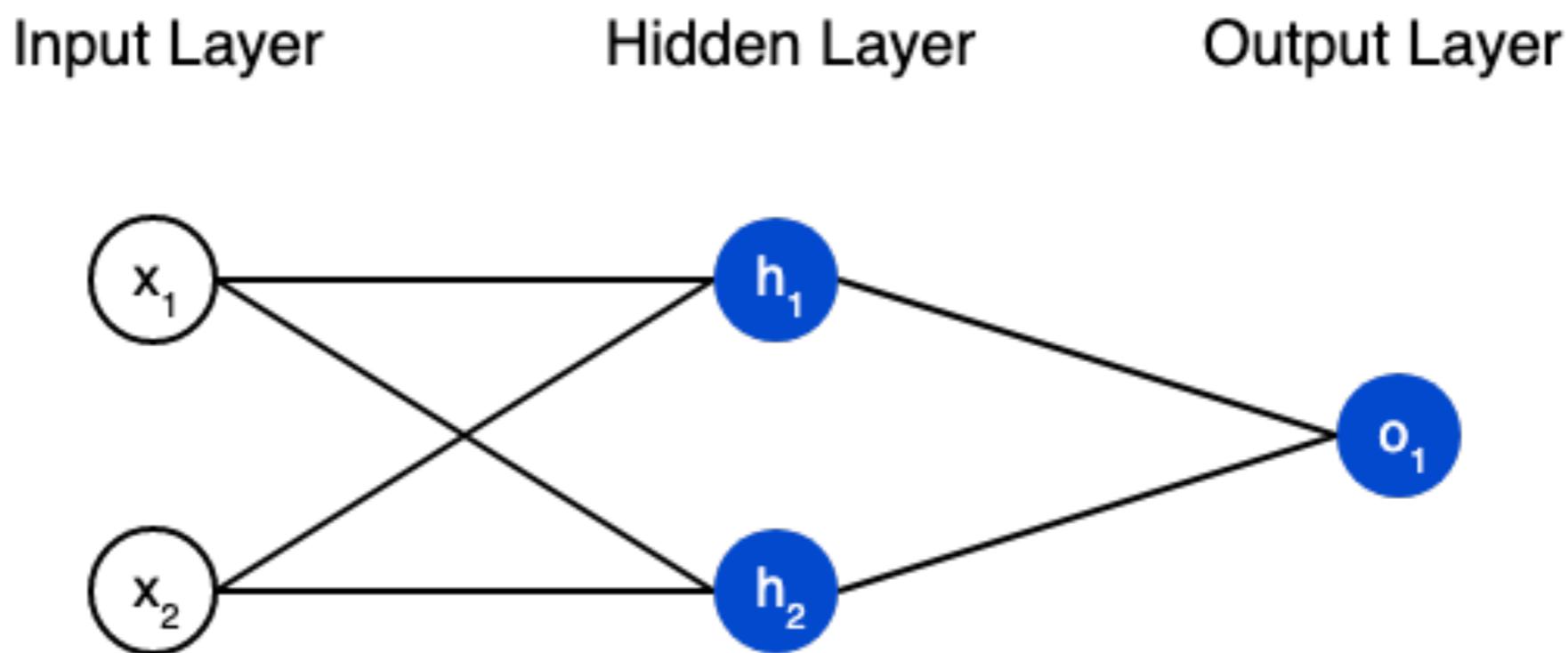
**Categorical Cross Entropy** 
$$CCE = -\frac{\sum_{i=1}^n \sum_{j=1}^c y_{i,j} \log(\hat{y}_{i,j})}{n}$$

## Binary Cross Entropy

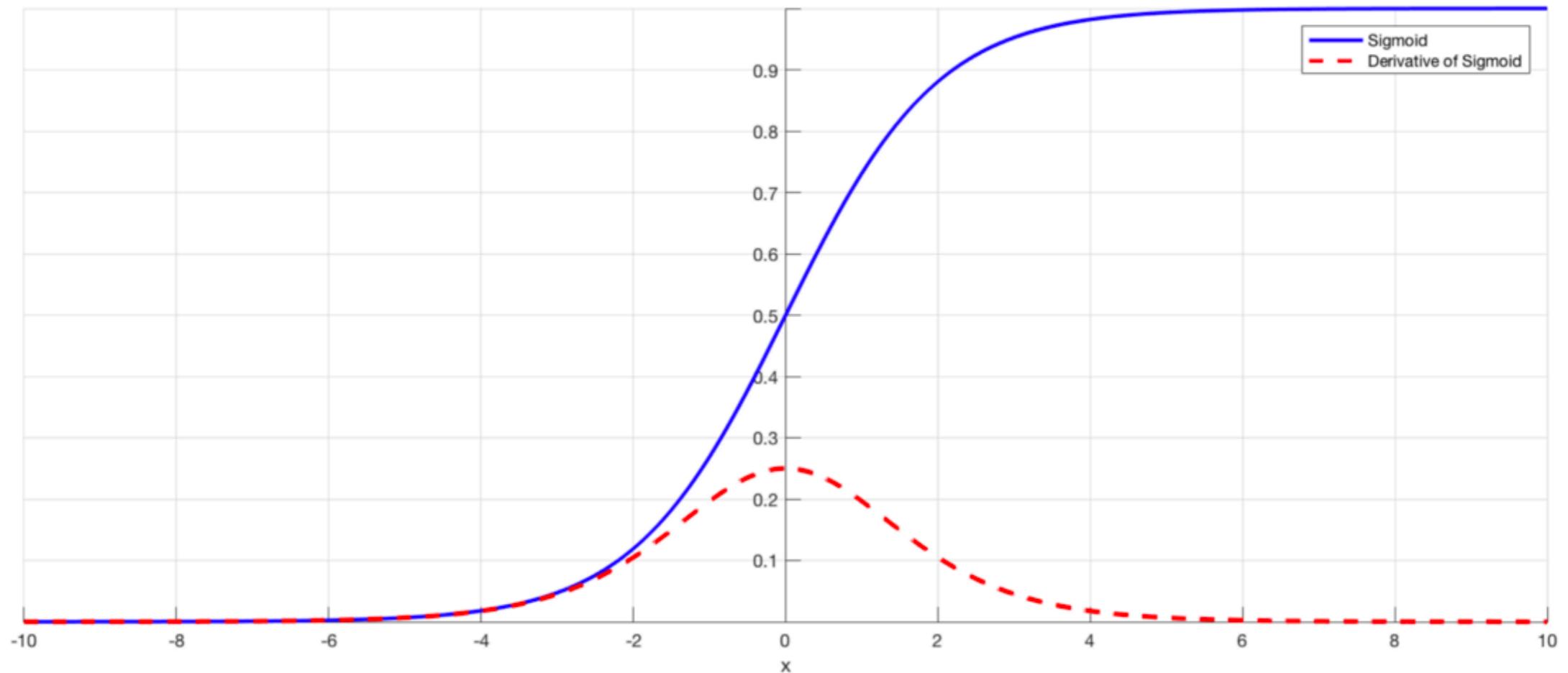
$$BCE = -\frac{\sum_{i=1}^n [y_{i,j} \log(\hat{y}_{i,j}) + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j})]}{n}$$

# Weight initialization

**Zero or constant initialization**



# Too low or too high initialization



# Xavier and He initializers

1. The mean of the activations should be zero
2. The variance of the activations should stay the same across every layer

**Xavier (Glorot) initialization  
for tanh**

$$\mathbf{W}^l \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{l-1}})$$

$$b^l = 0$$

**He (Kaiming) initialization  
for relu**

$$\mathbf{W}^l \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{2}{n^{l-1}})$$

$$b^l = 0$$

# Experiment with various initializations for a deep network

[01-Regression-nn-assignment.ipynb](#)

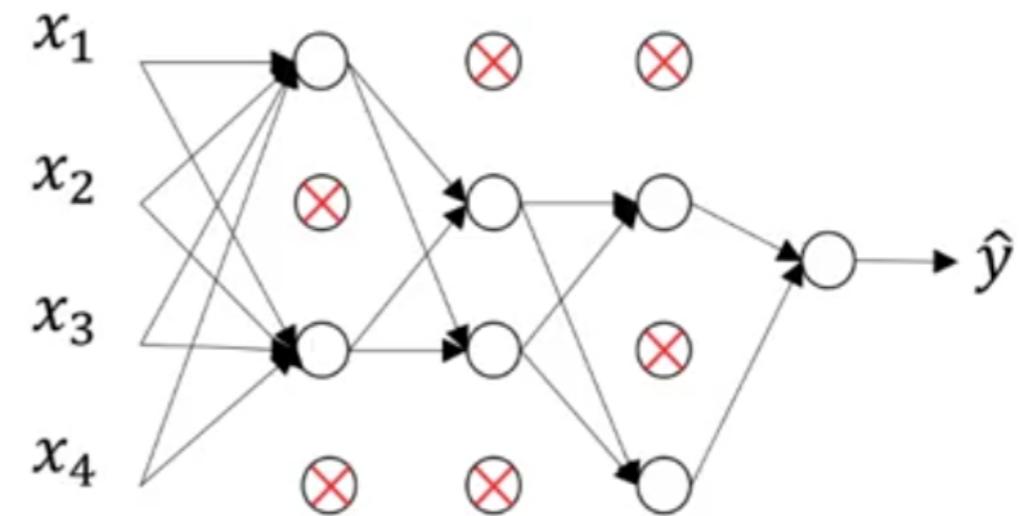
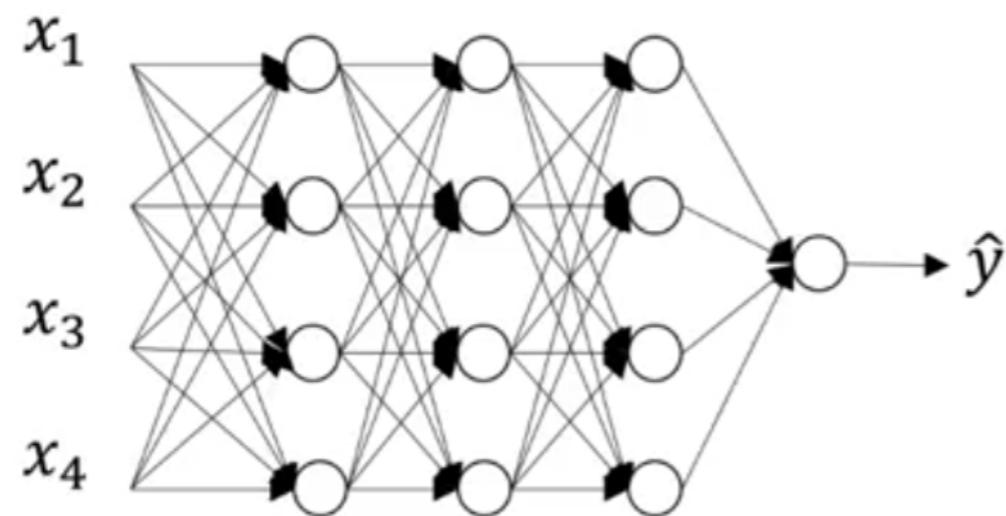
# L2 Regularization in deep learning

$$cost(w^1, b^1, \dots, w^L, b^L) = \frac{1}{n} \sum_{i=1}^n Loss(y_i, \hat{y}_i) + \frac{\lambda}{2n} \sum_{l=1}^L \|w^l\|_F^2$$

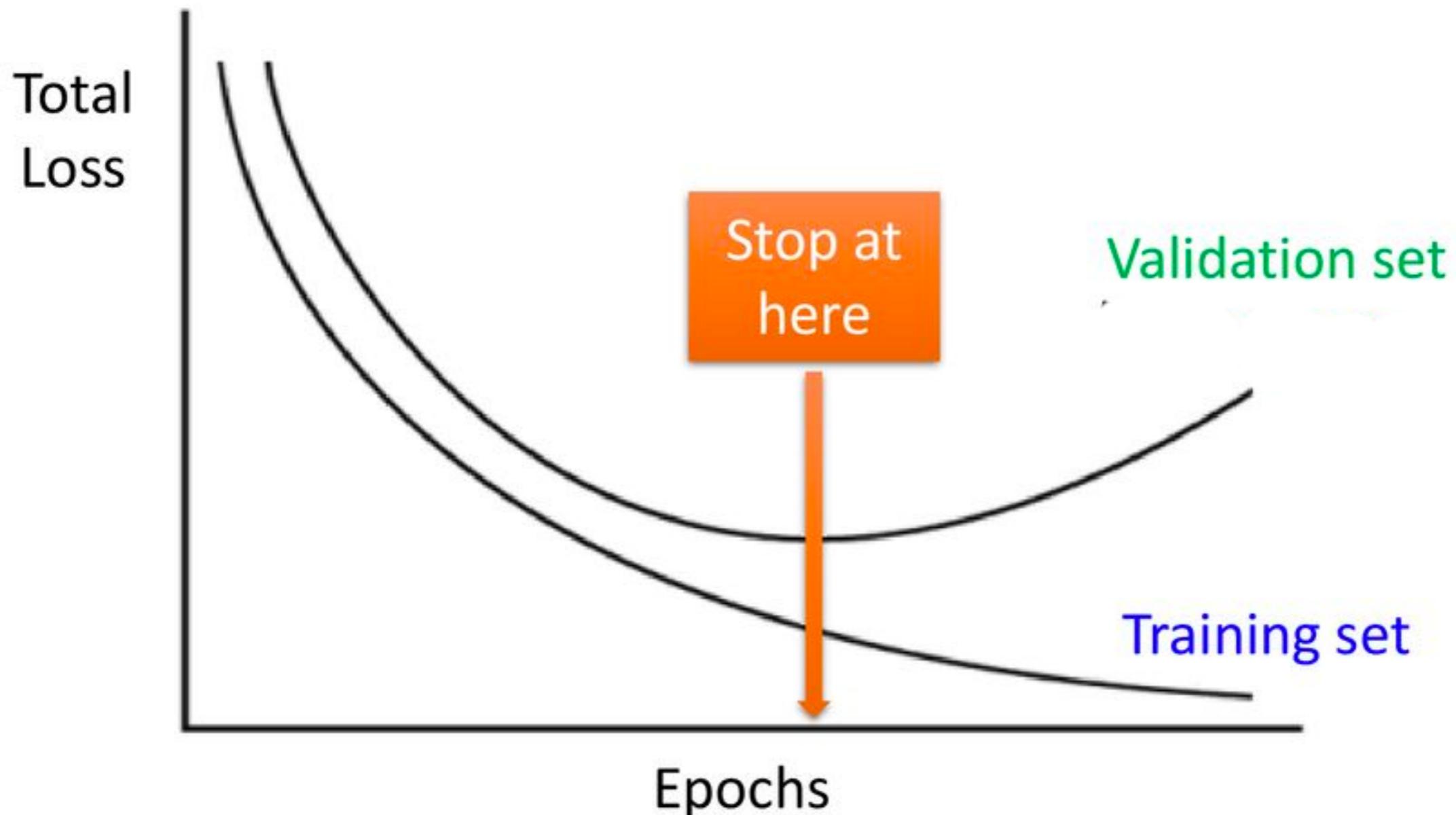
**Frobenius norm**

$$\|w\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m |w_{i,j}|^2}$$

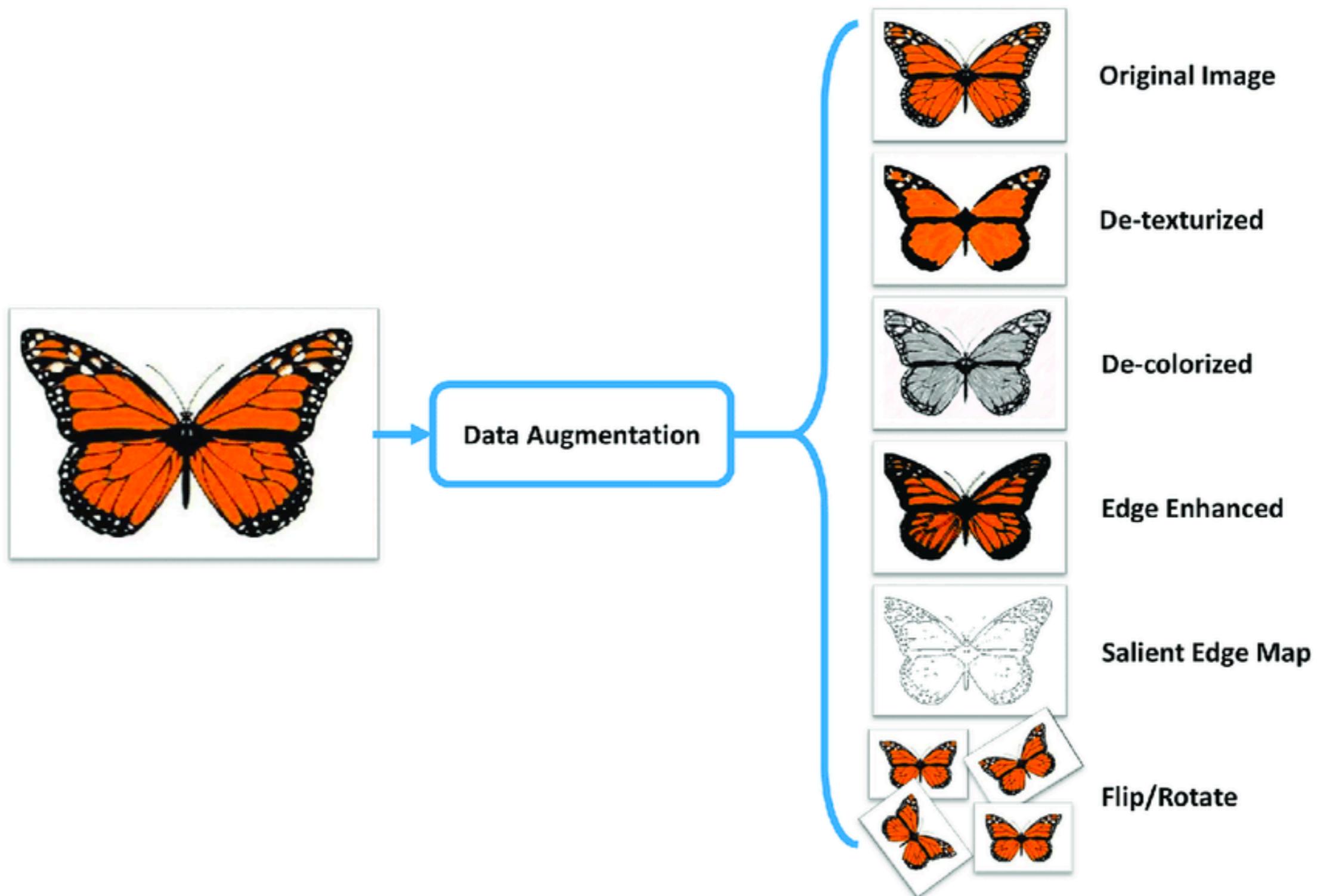
# Dropout



# Early stopping



# Data augmentation



# Batch normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

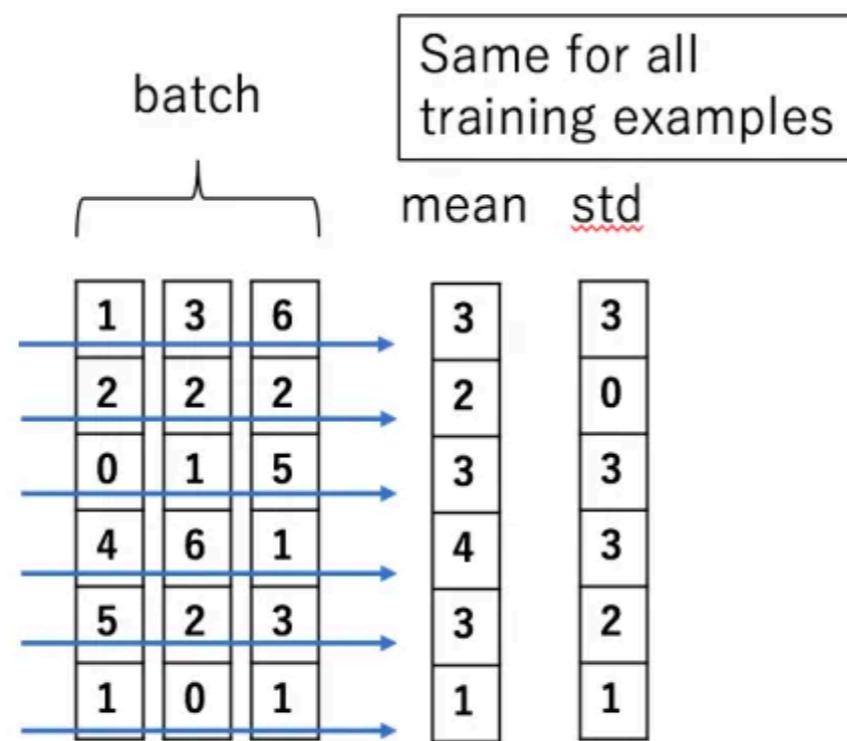
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

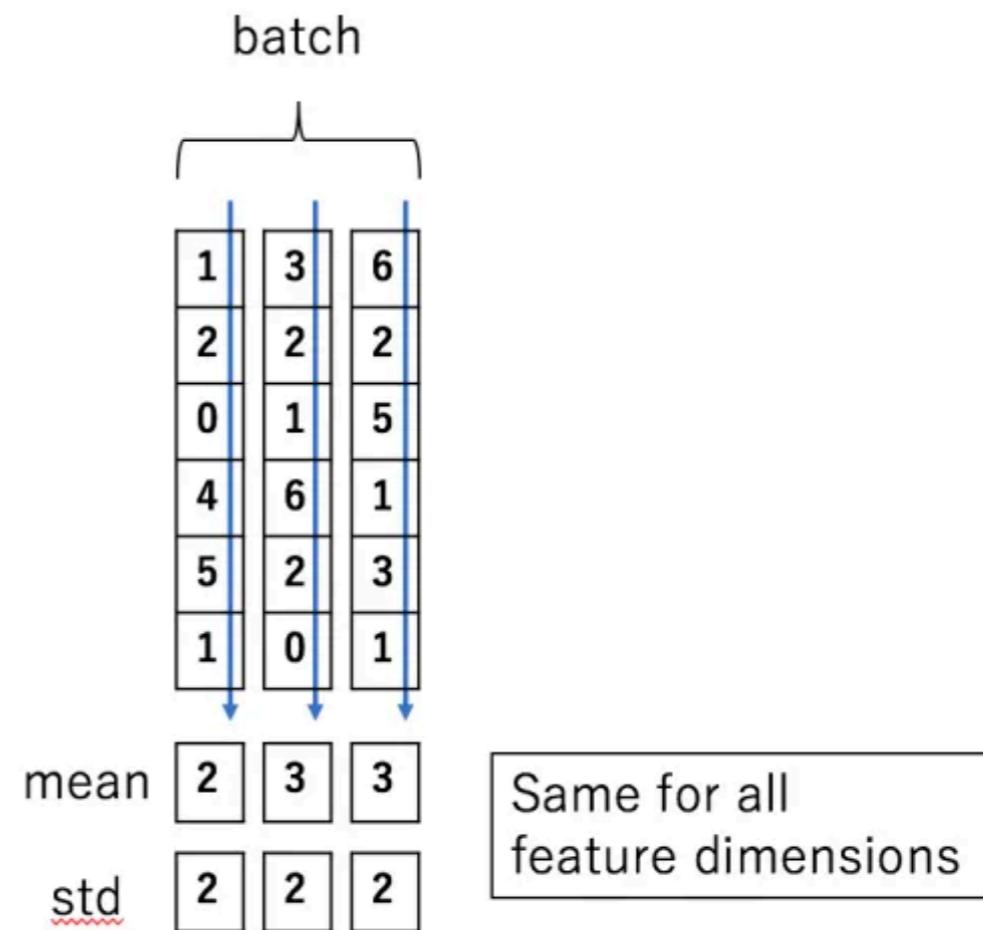
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

# Layer normalization

Batch Normalization



Layer Normalization



# Functional API in Keras

```
1 # Sequential model
2 from keras.models import Sequential
3 from keras.layers import Dense
4
5 model = Sequential()
6 model.add(10, input_shape=(10,), activation='relu')
7 model.add(Dense(20, activation='relu'))
8 model.add(Dense(10, activation='relu'))
9 model.add(Dense(1, activation='sigmoid'))
```

```
1 # Functional model
2 from keras.models import Model
3 from keras.layers import Input, Dense
4
5 visible = Input(shape=(10,))
6 hidden1 = Dense(10, activation='relu')(visible)
7 hidden2 = Dense(20, activation='relu')(hidden1)
8 hidden3 = Dense(10, activation='relu')(hidden2)
9 output = Dense(1, activation='sigmoid')(hidden3)
10 model = Model(inputs=visible, outputs=output)
```

# Shared Input

```
1 # Shared Input Layer
2 from keras.utils import plot_model
3 from keras.models import Model
4 from keras.layers import Input, Dense, Flatten
5 from keras.layers.convolutional import Conv2D
6 from keras.layers.pooling import MaxPooling2D
7 from keras.layers.merge import concatenate
8 # input layer
9 visible = Input(shape=(64,64,1))
10 # first feature extractor
11 conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
12 pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
13 flat1 = Flatten()(pool1)
14 # second feature extractor
15 conv2 = Conv2D(16, kernel_size=8, activation='relu')(visible)
16 pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
17 flat2 = Flatten()(pool2)
18 # merge feature extractors
19 merge = concatenate([flat1, flat2])
20 # interpretation layer
21 hidden1 = Dense(10, activation='relu')(merge)
22 # prediction output
23 output = Dense(1, activation='sigmoid')(hidden1)
24 model = Model(inputs=visible, outputs=output)
```

# Multiple inputs (outputs)

```
1 # Multiple Inputs
2 from keras.utils import plot_model
3 from keras.models import Model
4 from keras.layers import Input
5 from keras.layers import Dense
6 from keras.layers import Flatten
7 from keras.layers.convolutional import Conv2D
8 from keras.layers.pooling import MaxPooling2D
9 from keras.layers.merge import concatenate
10 # first input model
11 visible1 = Input(shape=(64,64,1))
12 conv11 = Conv2D(32, kernel_size=4, activation='relu')(visible1)
13 pool11 = MaxPooling2D(pool_size=(2, 2))(conv11)
14 conv12 = Conv2D(16, kernel_size=4, activation='relu')(pool11)
15 pool12 = MaxPooling2D(pool_size=(2, 2))(conv12)
16 flat1 = Flatten()(pool12)
17 # second input model
18 visible2 = Input(shape=(32,32,3))
19 conv21 = Conv2D(32, kernel_size=4, activation='relu')(visible2)
20 pool21 = MaxPooling2D(pool_size=(2, 2))(conv21)
21 conv22 = Conv2D(16, kernel_size=4, activation='relu')(pool21)
22 pool22 = MaxPooling2D(pool_size=(2, 2))(conv22)
23 flat2 = Flatten()(pool22)
24 # merge input models
25 merge = concatenate([flat1, flat2])
26 # interpretation model
27 hidden1 = Dense(10, activation='relu')(merge)
28 hidden2 = Dense(10, activation='relu')(hidden1)
29 output = Dense(1, activation='sigmoid')(hidden2)
30 model = Model(inputs=[visible1, visible2], outputs=output)
```

# Practical example on regularization and normalization

[\*\*02-Normalization-and-regularization-assignment.ipynb\*\*](#)

# Neural Network architectures design



# Neural Network design best practices

- ★ Start from simple architectures
- ★ Get inspiration from architectures for similar problems
- ★ Change one parameter only and then validate

# Most common architectures

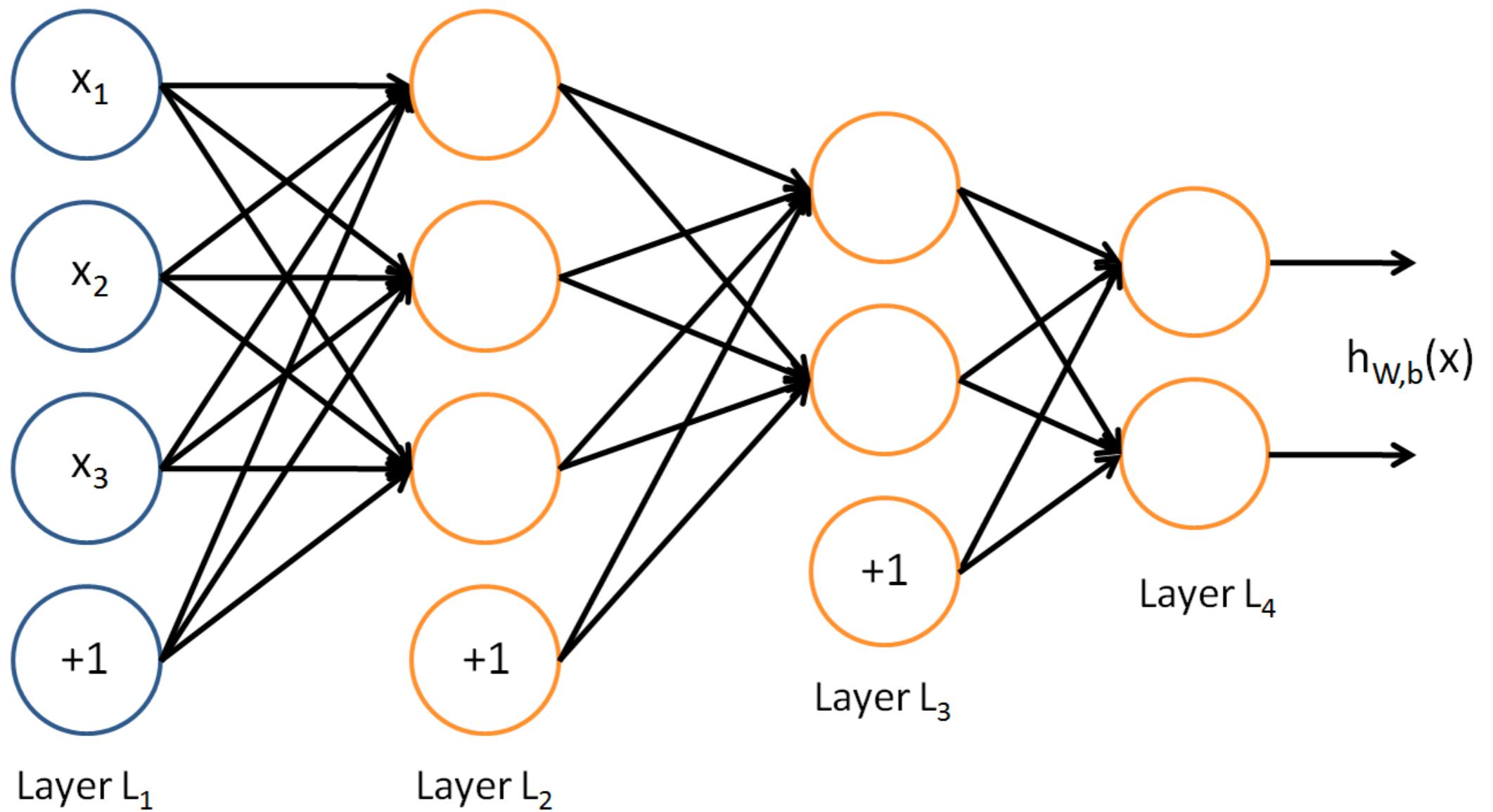
**Feed forward network**

**Convolutional network**

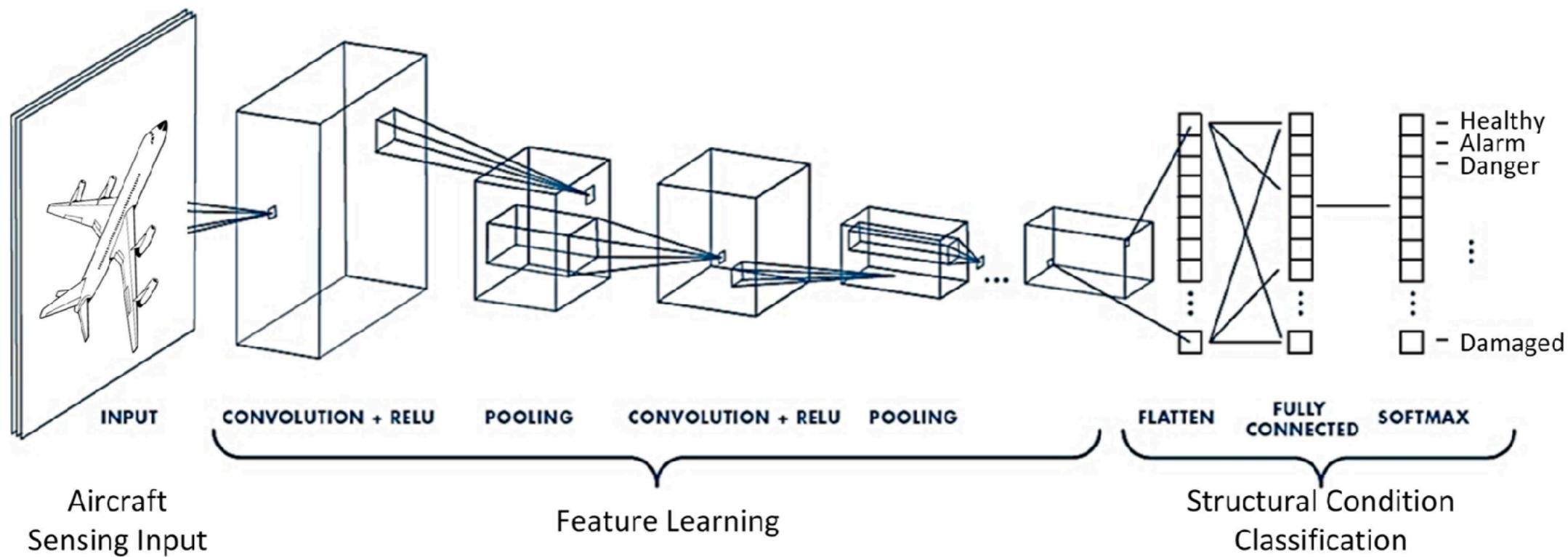
**Recurrent network**

**Generative adversarial network**

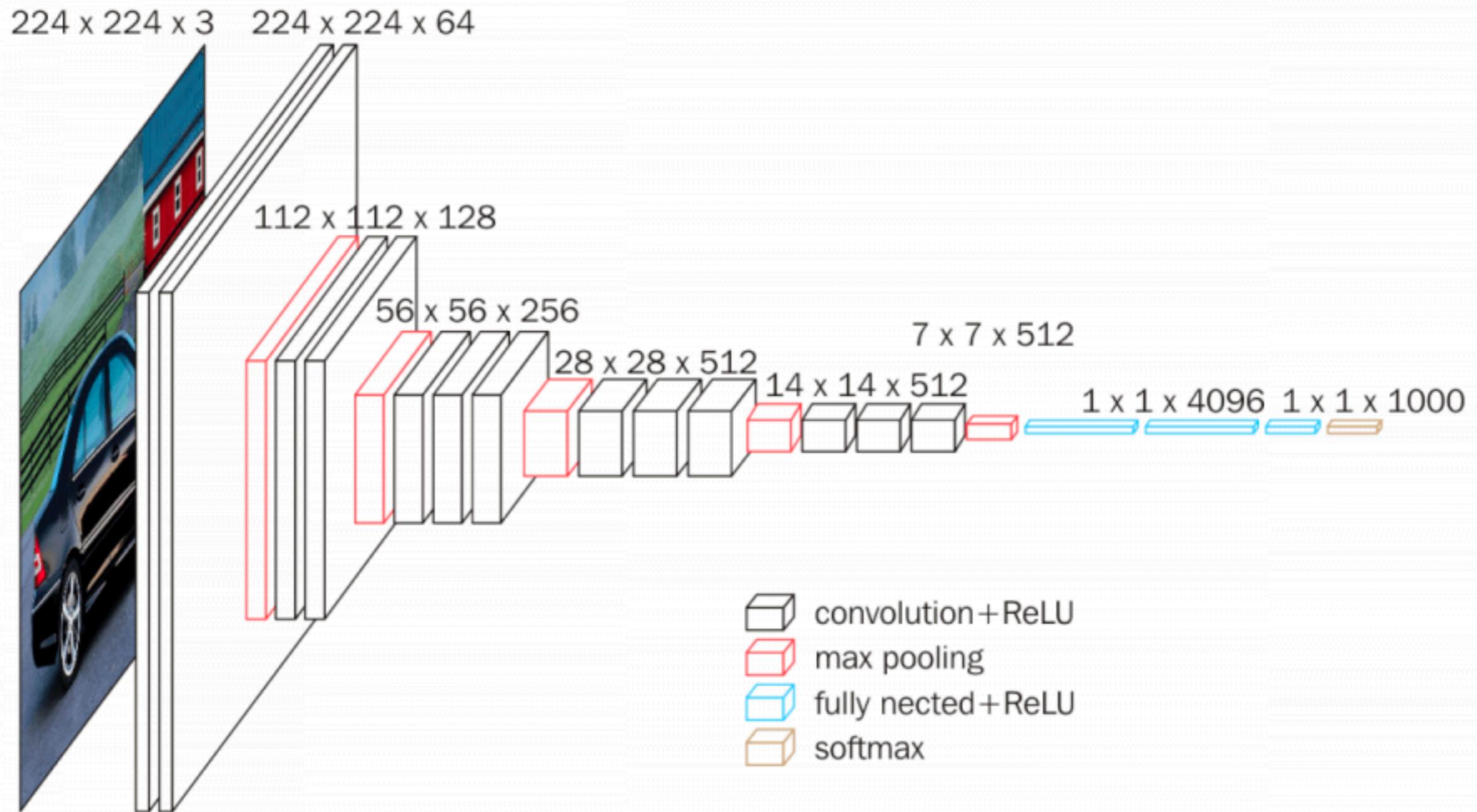
# Feed Forward Network



# Convolutional Neural Network

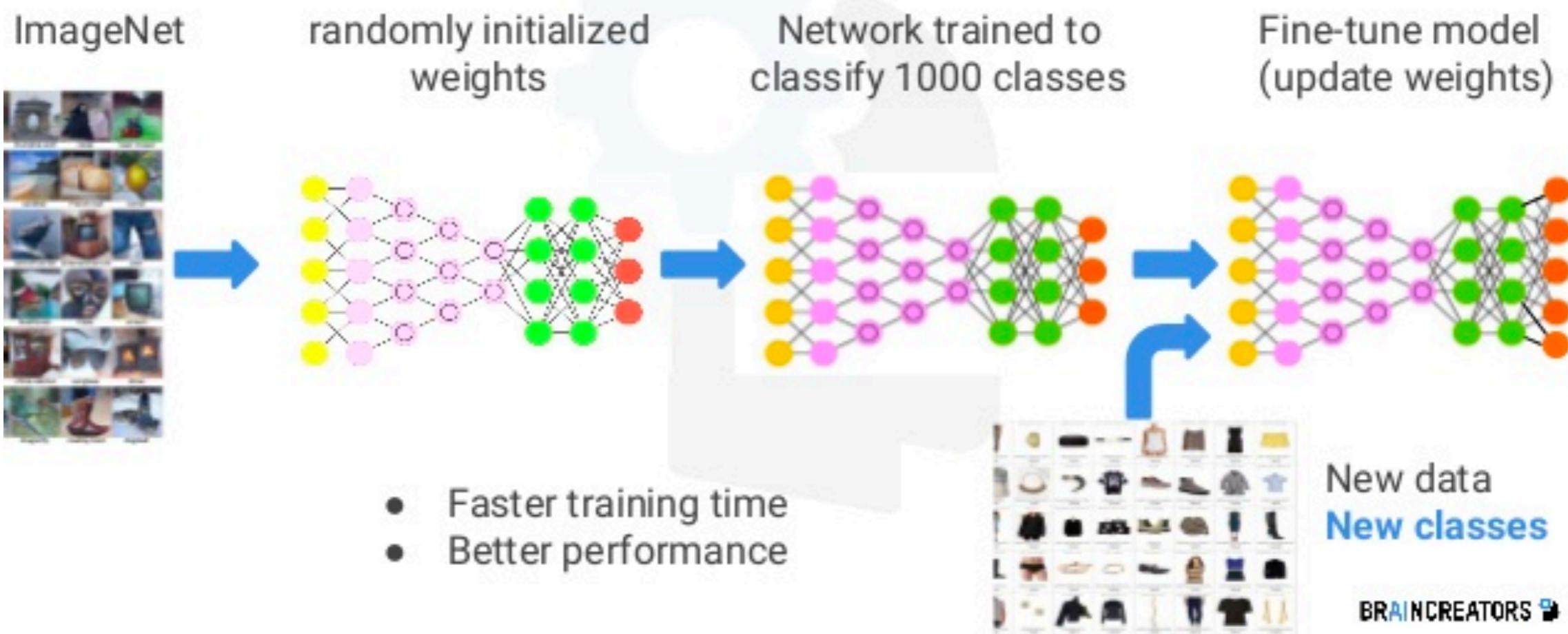


# VGG 16



# Finetuning

## Transfer Learning



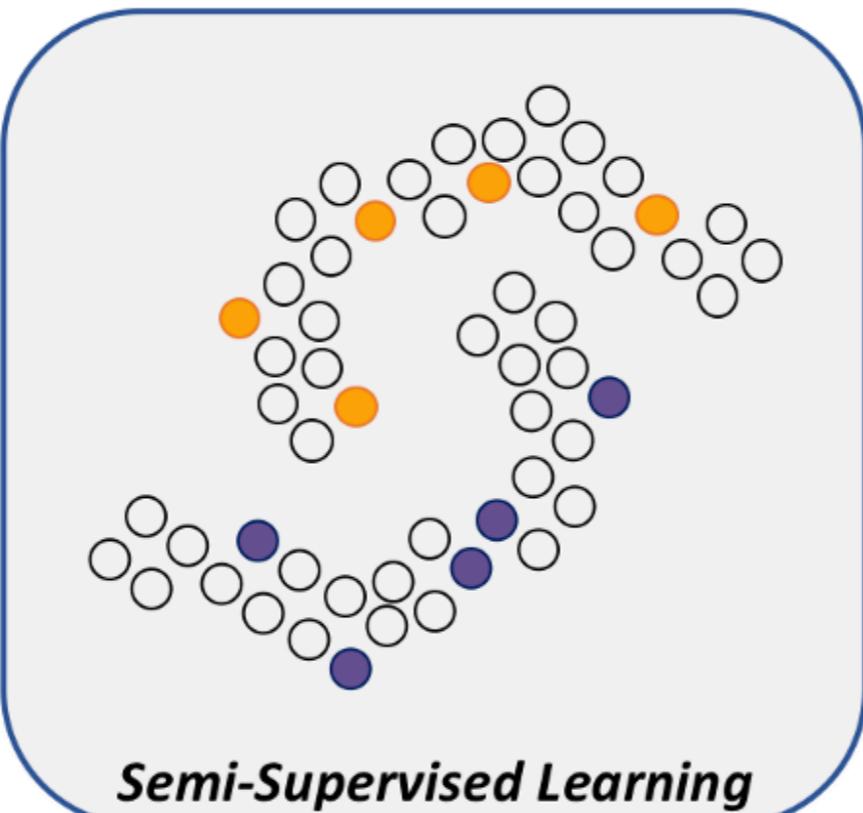
# **Transfer learning example**

**03-Transfer\_learning.ipynb**

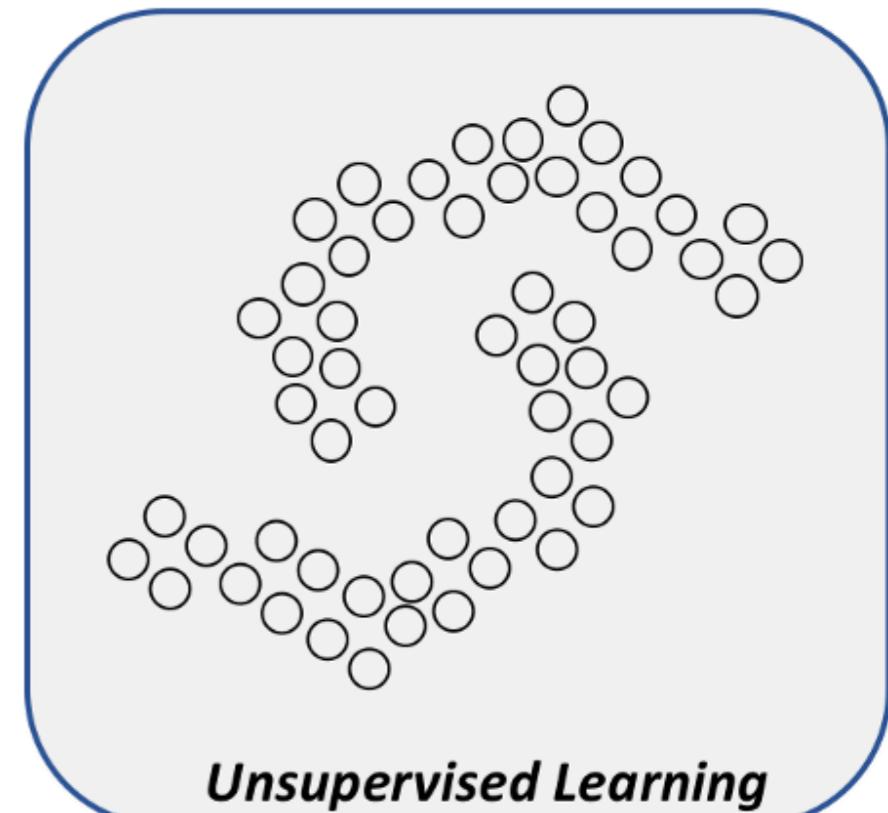
# Semi-supervised learning



*Supervised Learning*

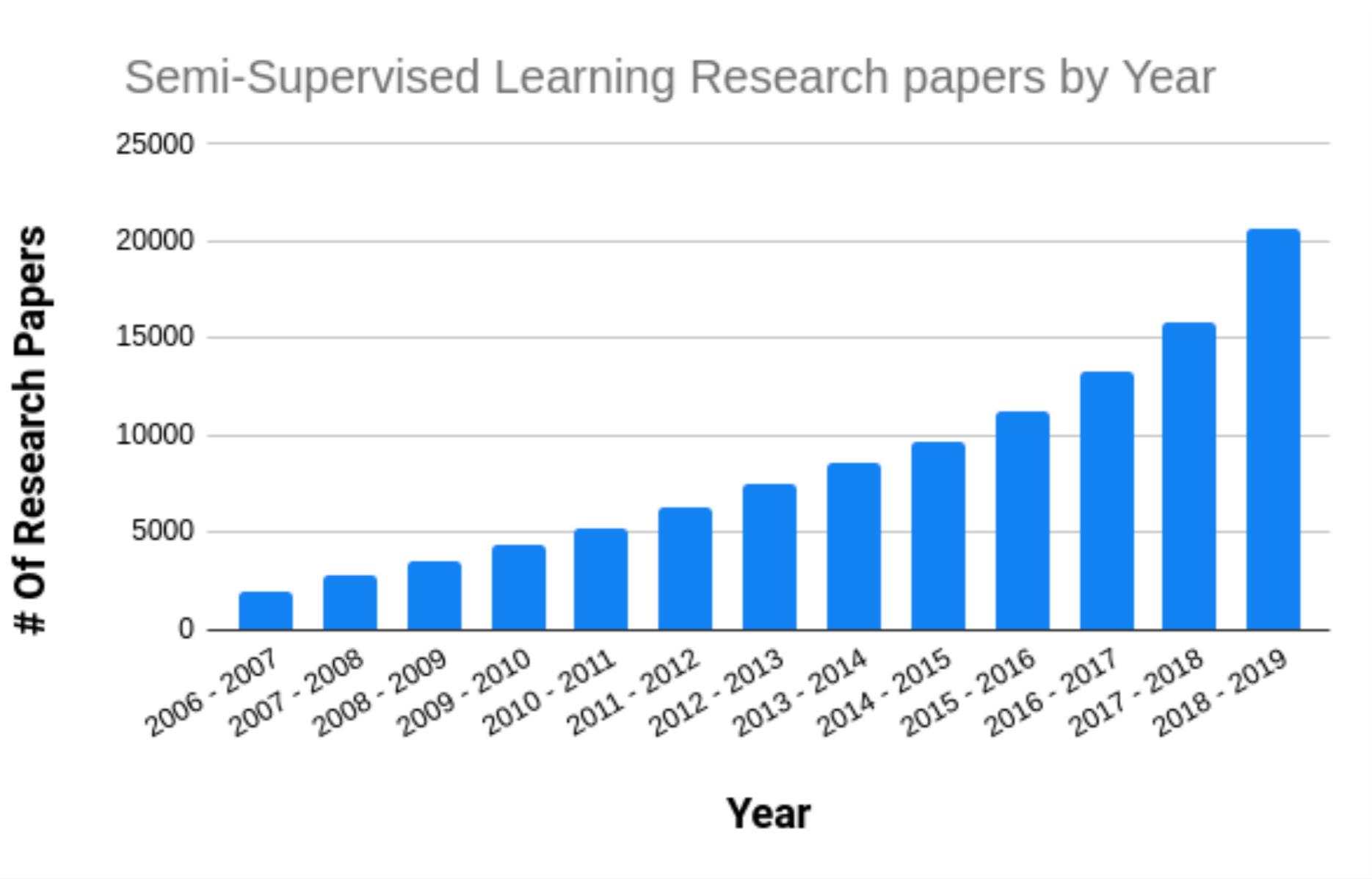


*Semi-Supervised Learning*

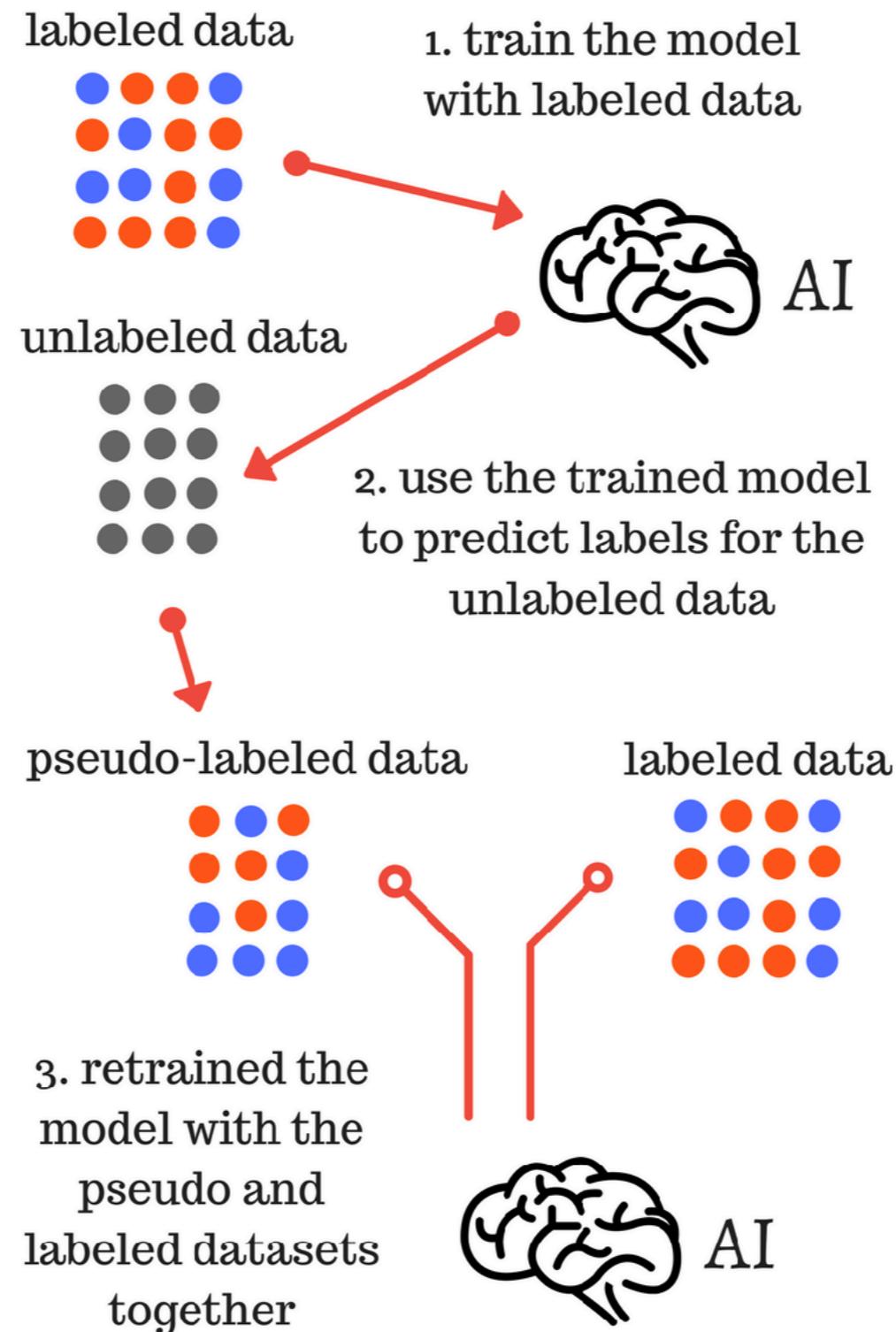


*Unsupervised Learning*

# Small Data is the new Big Data

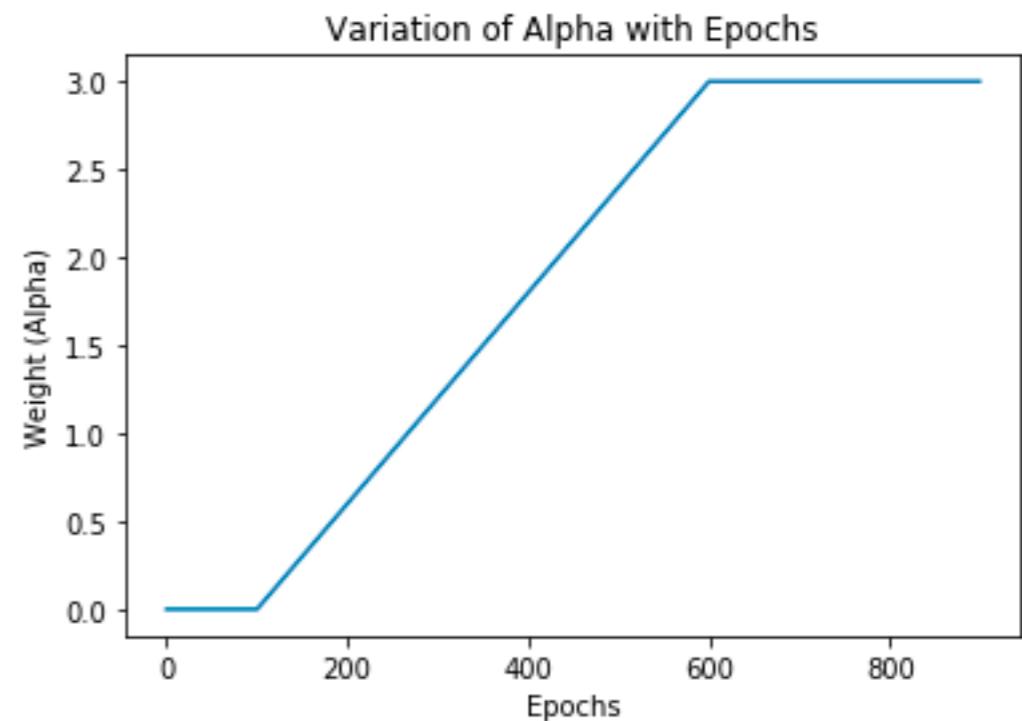


# Pseudo-labeling

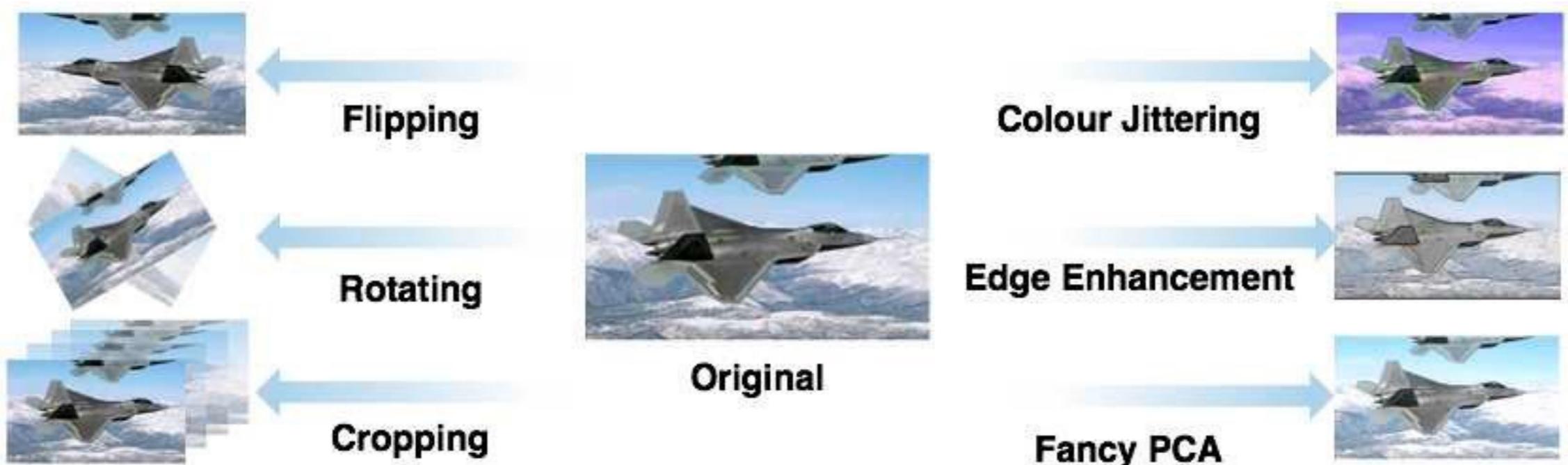


# Pseudo-labeling with neural networks

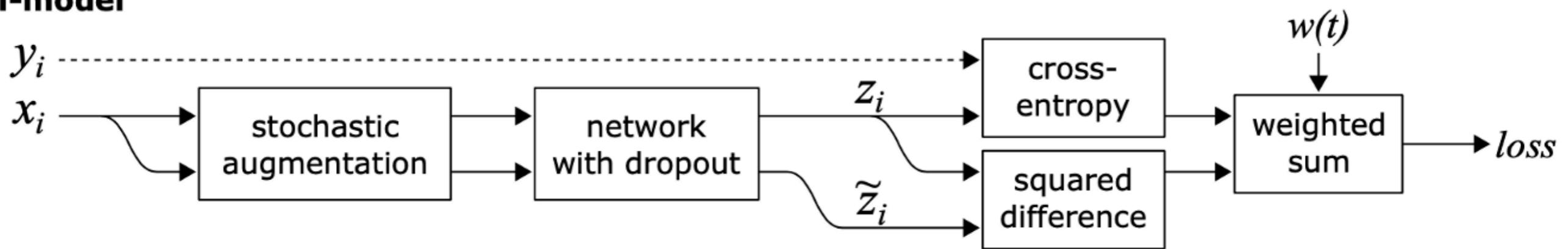
- ★ Train model on a batch of labeled data
- ★ Use the trained model to predict labels on a batch of unlabeled data
- ★ Use the predicted labels to calculate the loss on unlabeled data
- ★ Combine labeled loss with  $\alpha$ -weighted unlabeled loss and backpropagate
- ★ Repeat



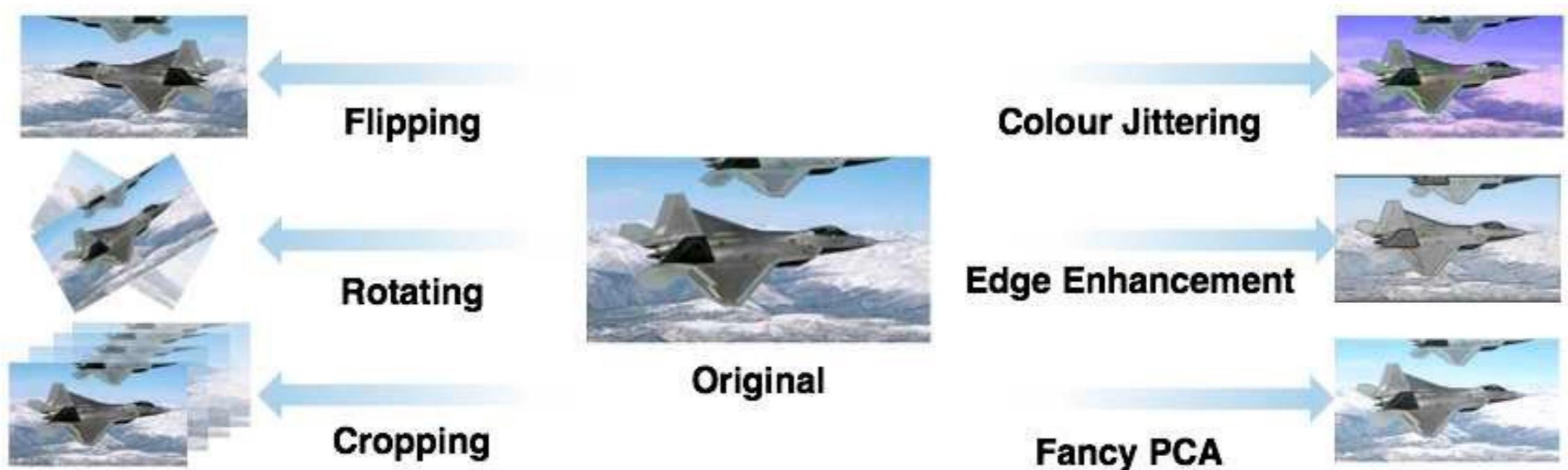
# PI-model



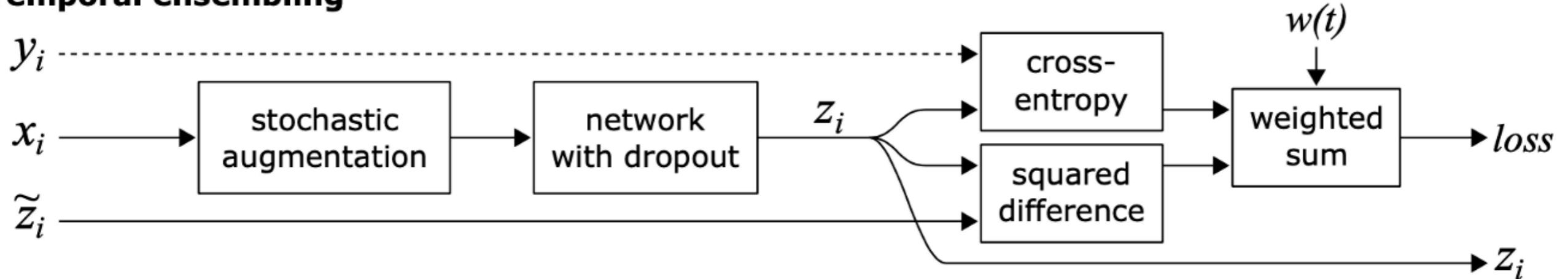
## Π-model



# Temporal ensembling



## Temporal ensembling

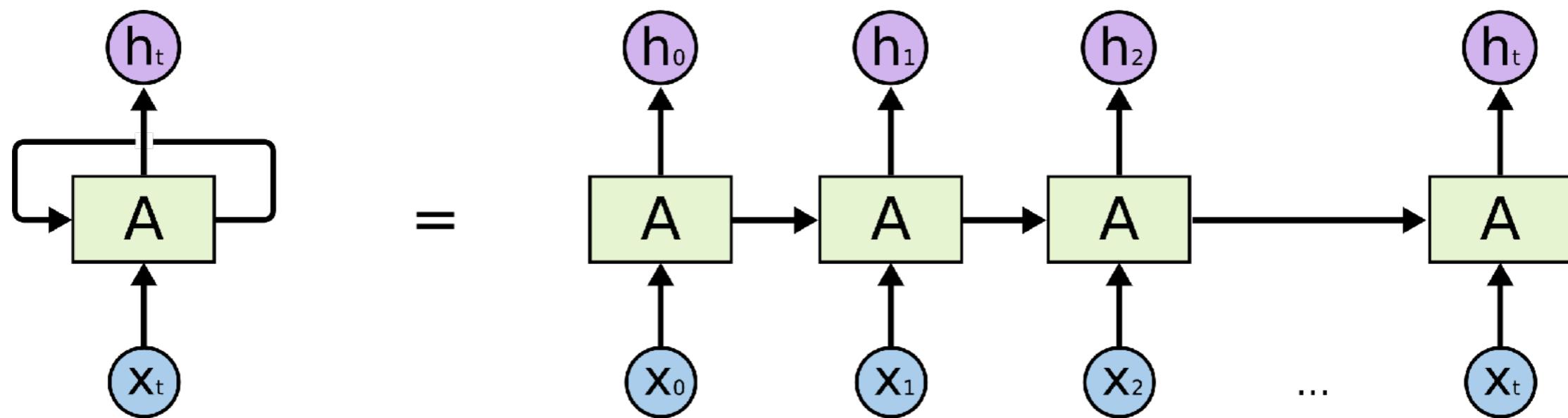


# Clustering as semi-supervised learning

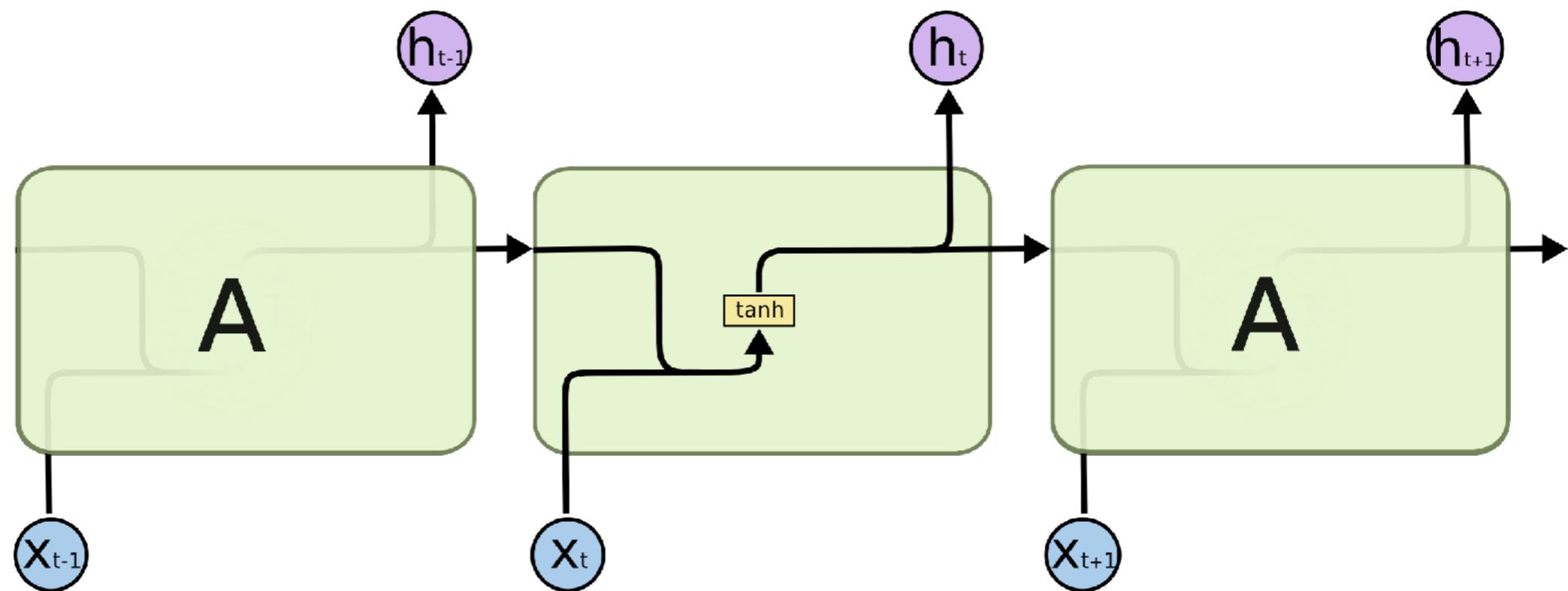
**04-clustering-assignment.ipynb**

# **Selected Neural Network Architectures**

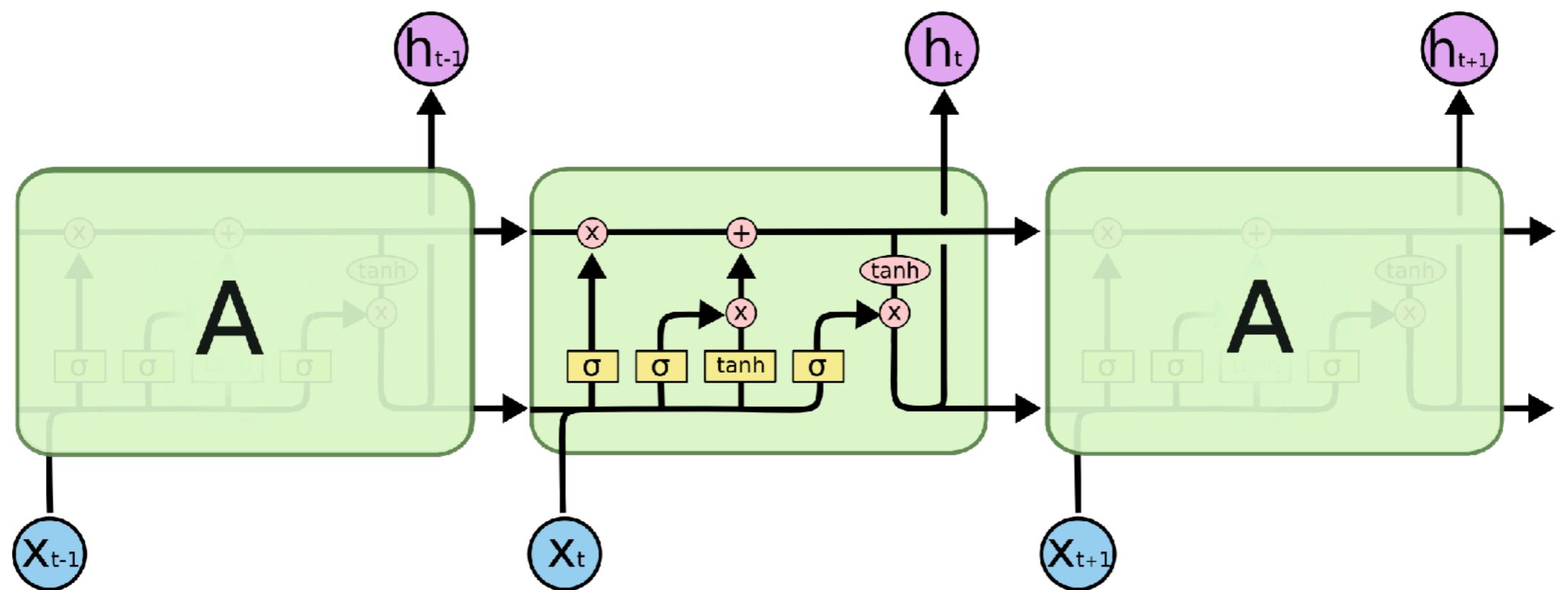
# Recurrent Neural Network



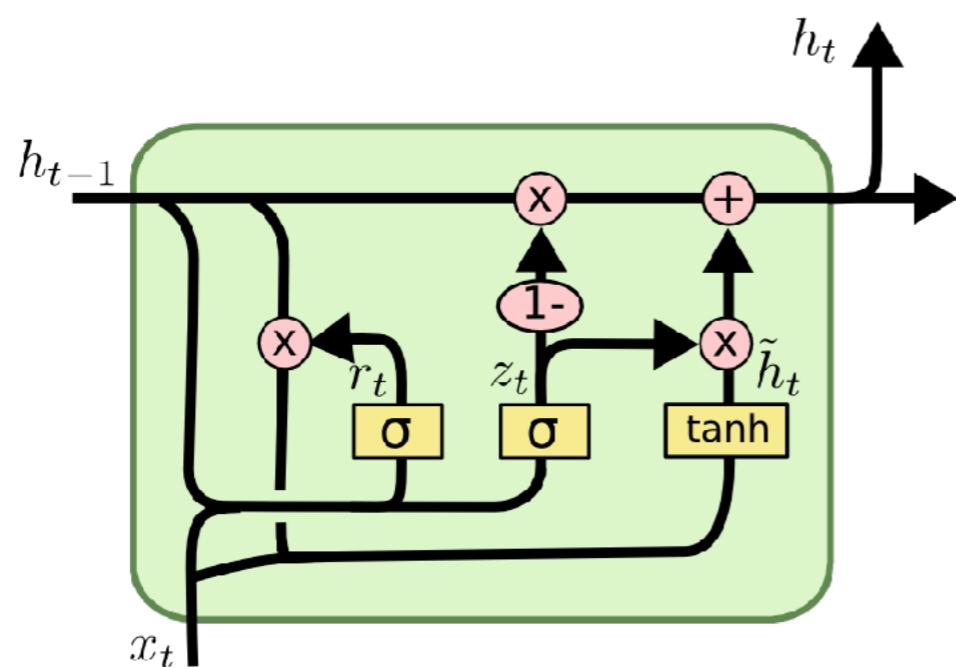
# Recurrent Neural Network



# Long Short Term Memory



# Gated Recurrent Unit



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

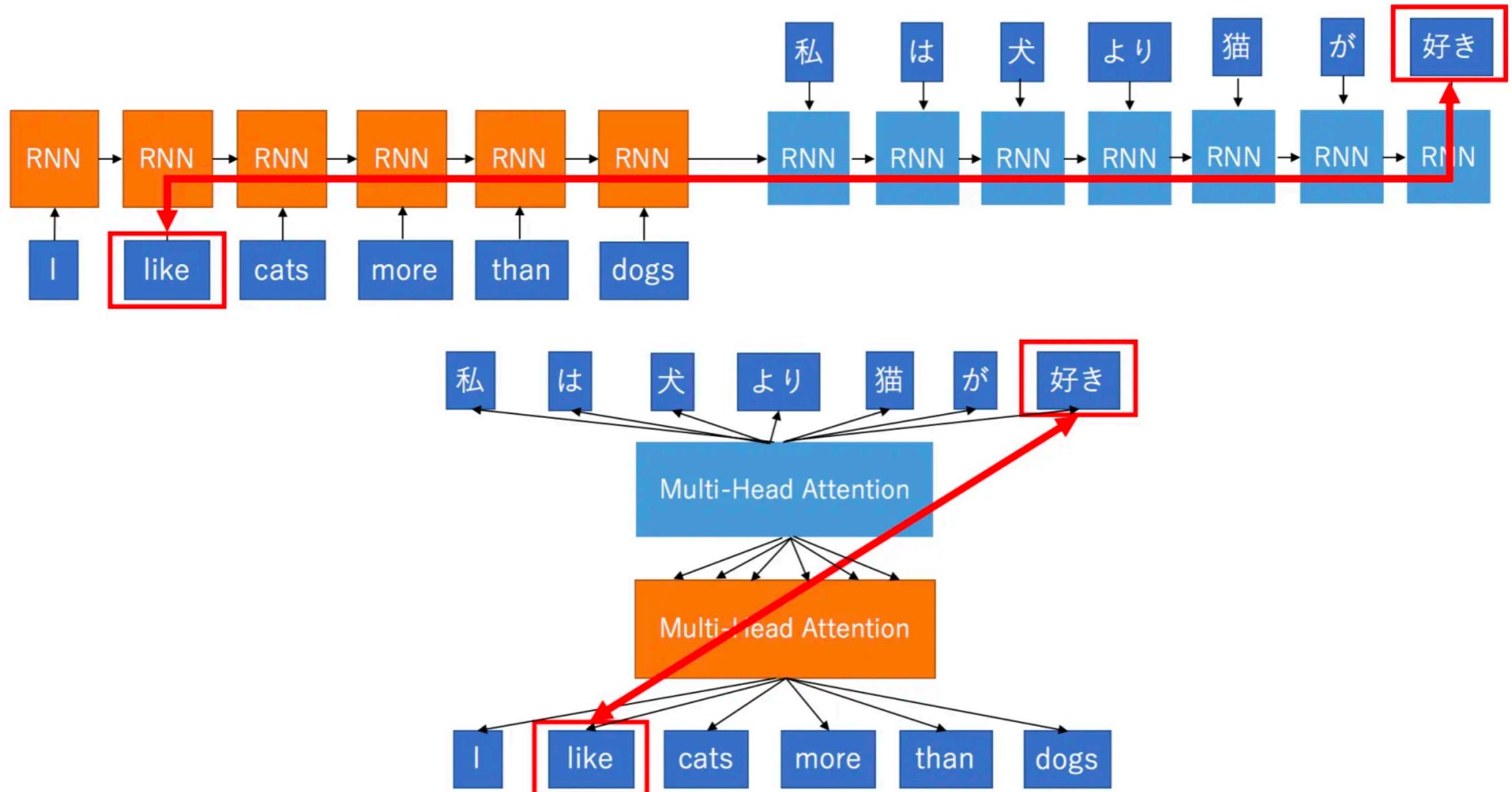
$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# **Regression using RNN**

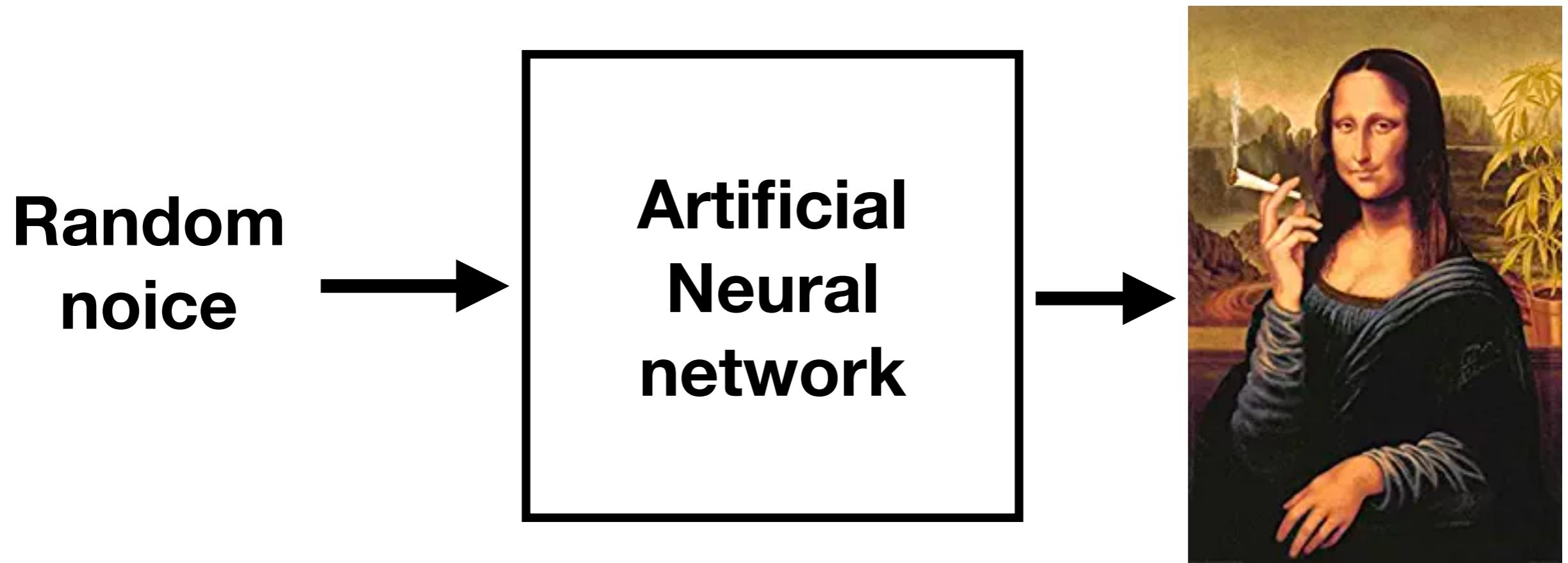
**05-Regression-RNN-assignment.ipynb**

# Transformer

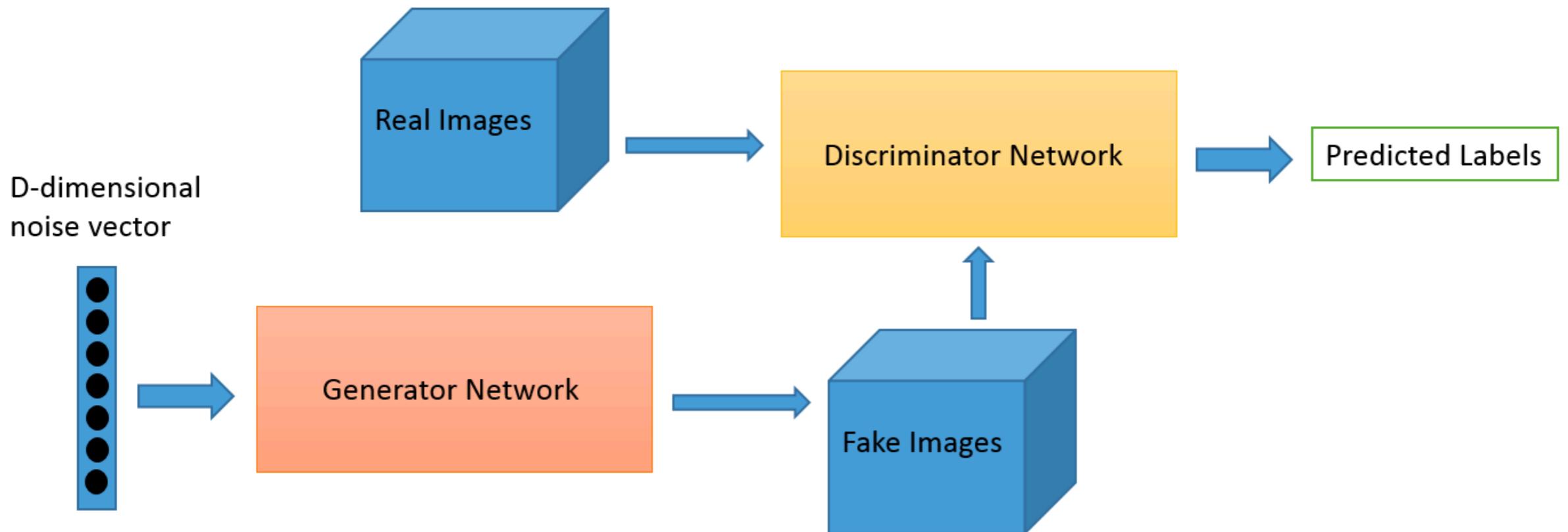


source: [www.mlexplained.com](http://www.mlexplained.com)

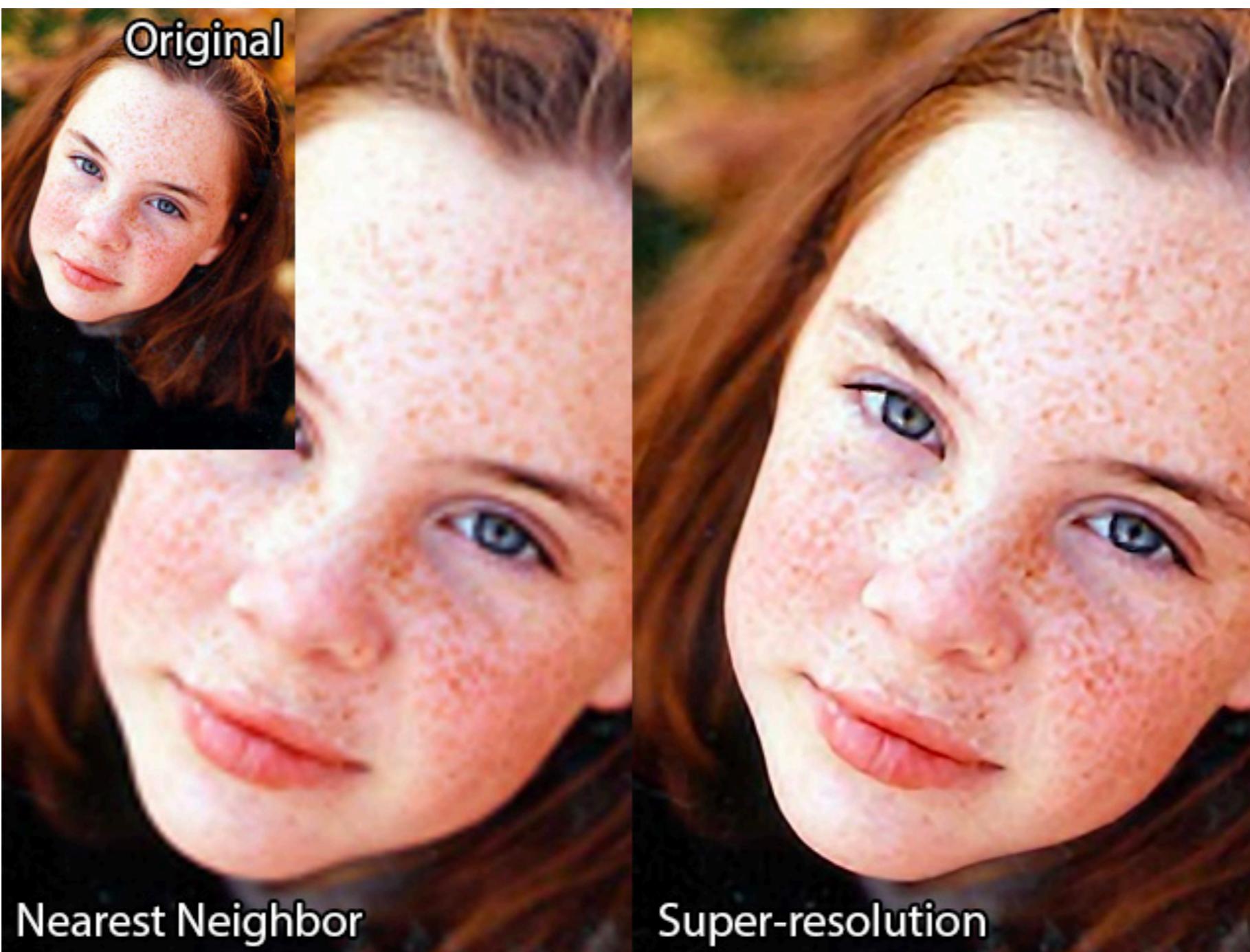
# Generative models with neural networks



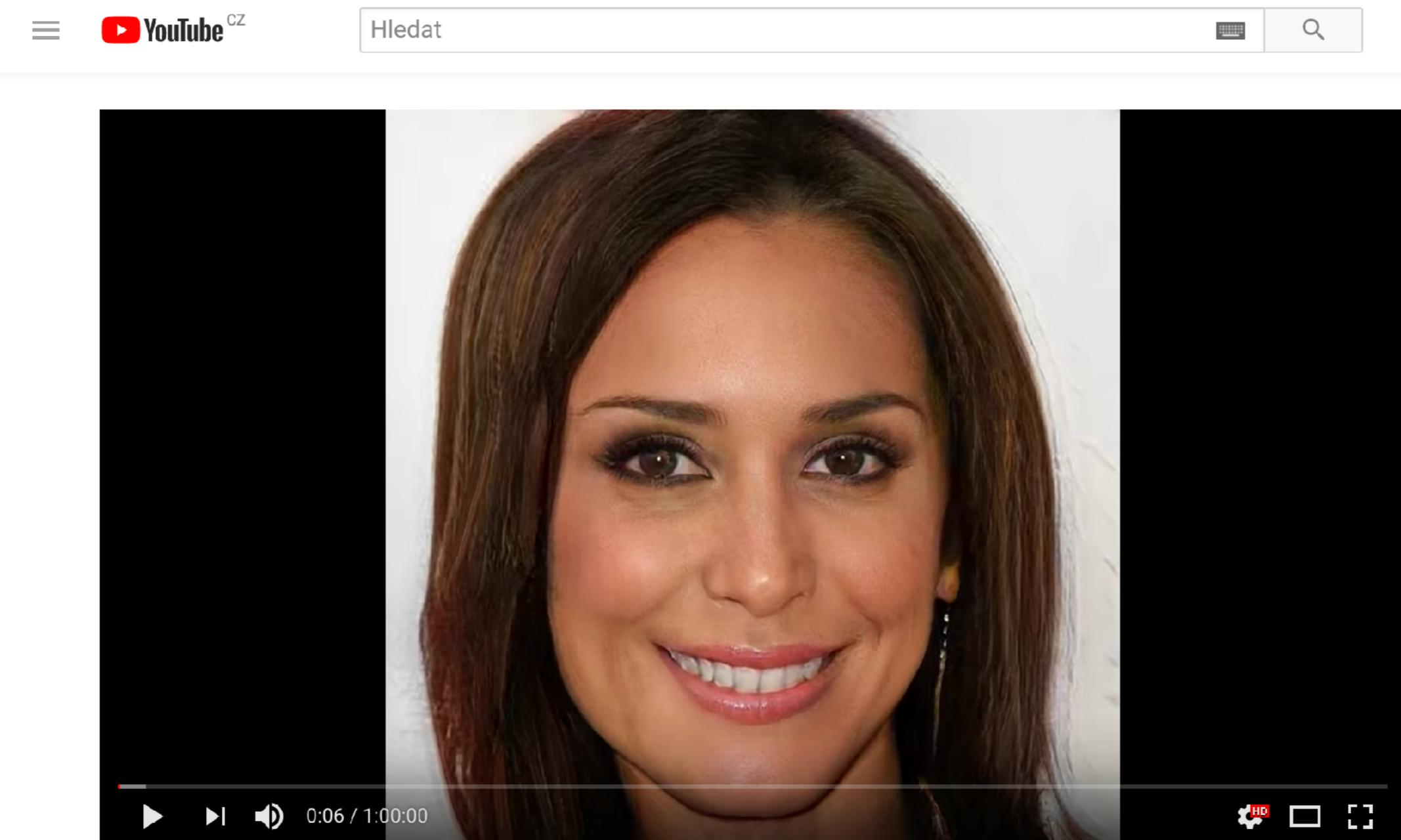
# Generative Adversarial Networks



# Superresolution



# Image synthesis



One hour of imaginary celebrities

95 832 zhlédnutí

TO SE MI LÍBÍ NELÍBÍ SE SDÍLET ...

# Which one is fake?



# Thank you for your attention

**e-mail:** jiri@mlguru.com

**Web:** www.mlguru.com

**Twitter:** @JiriMaterna

**Facebook:** <https://www.facebook.com/maternajiri>

**LinkedIn:** <https://www.linkedin.com/in/jirimaterna/>