Advanced Deep Learning techniques

Jiří Materna

Machine Learning College

Outline

- Neural Network architectures
- Optimizers and their evolution
- Loss functions and their properties
- Initialization of weights in neural networks
- Normalization and Regularization in neural networks
- Functional model definition in Keras
- Semi-supervised learning
- AutoML approaches

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

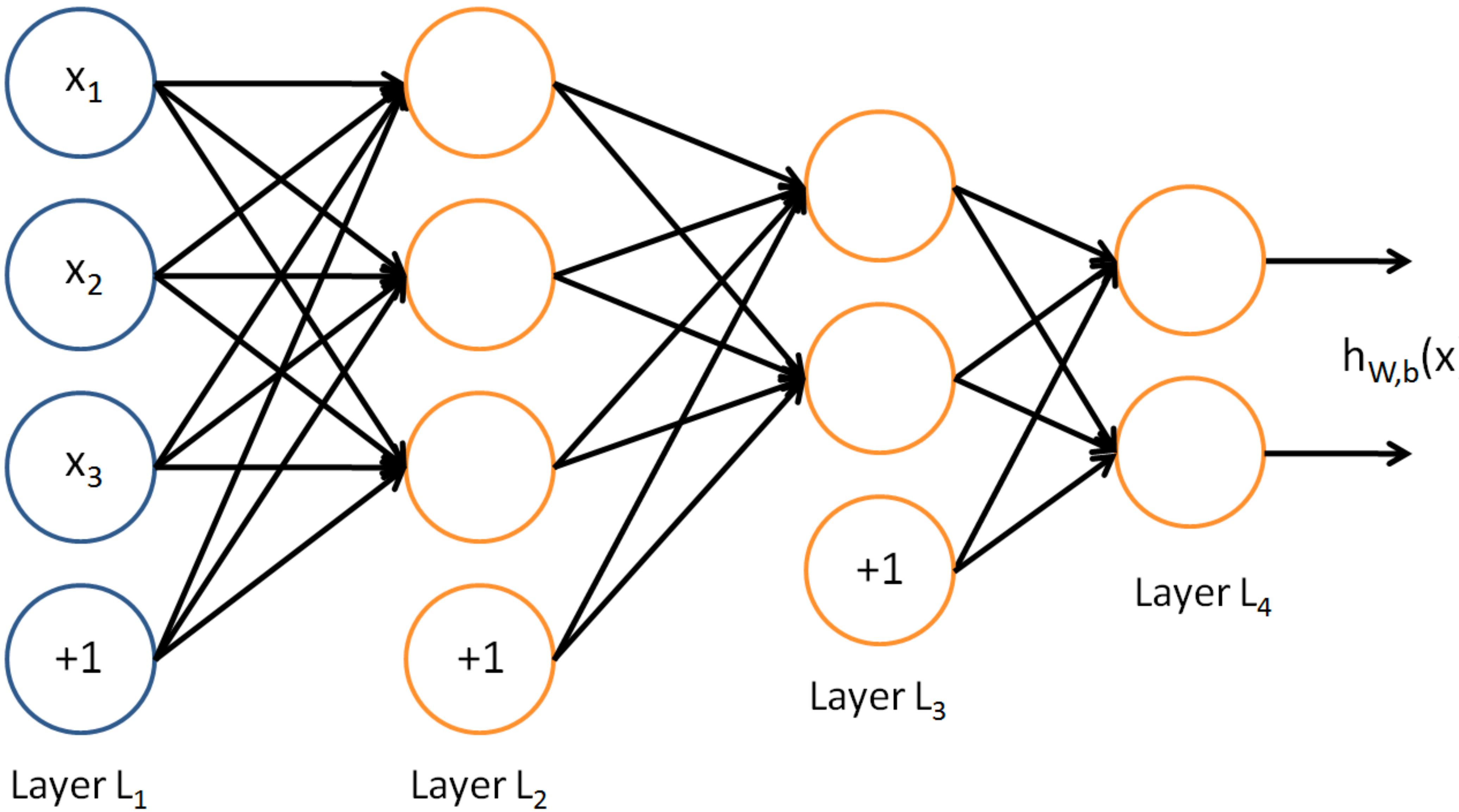
Autoencoder network and Restricted Blotzmann Machines

Transformer network

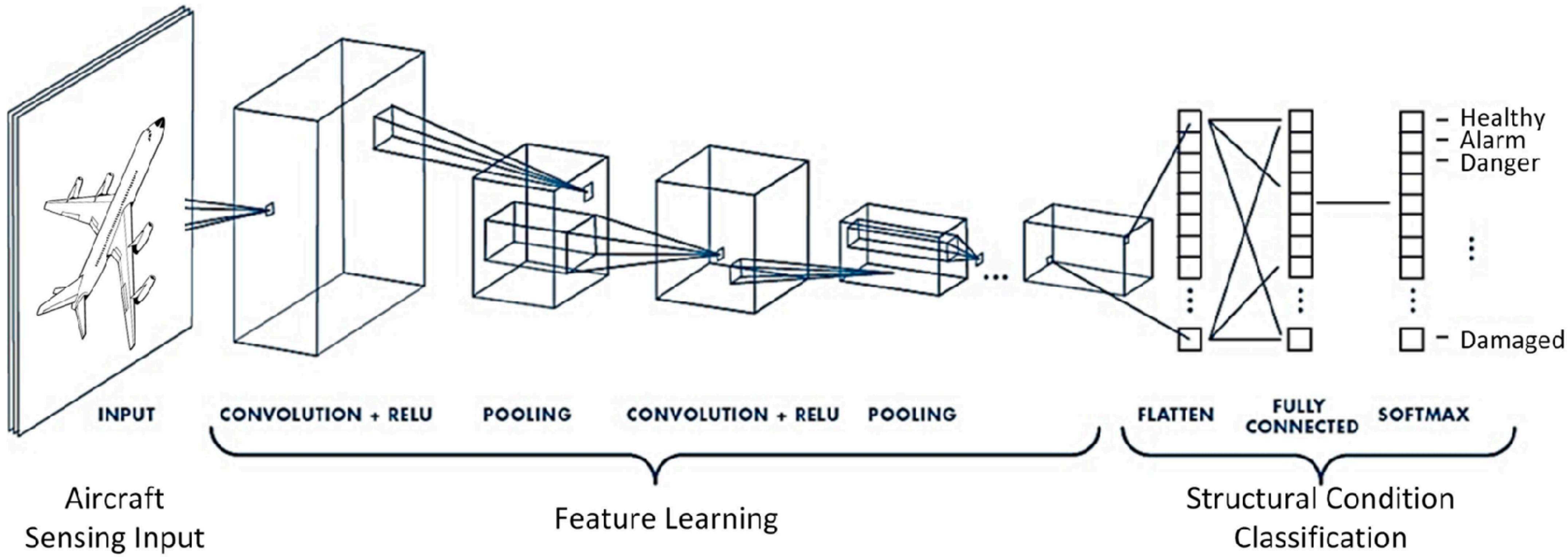
U-Net

Generative adversarial network

Feed Forward Network

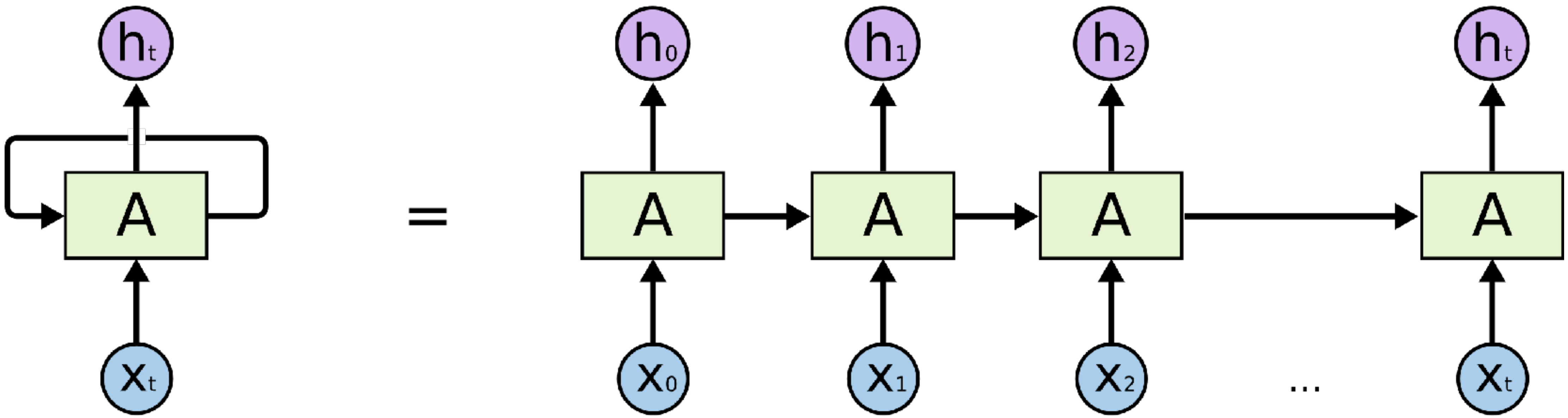


Convolutional Neural Network



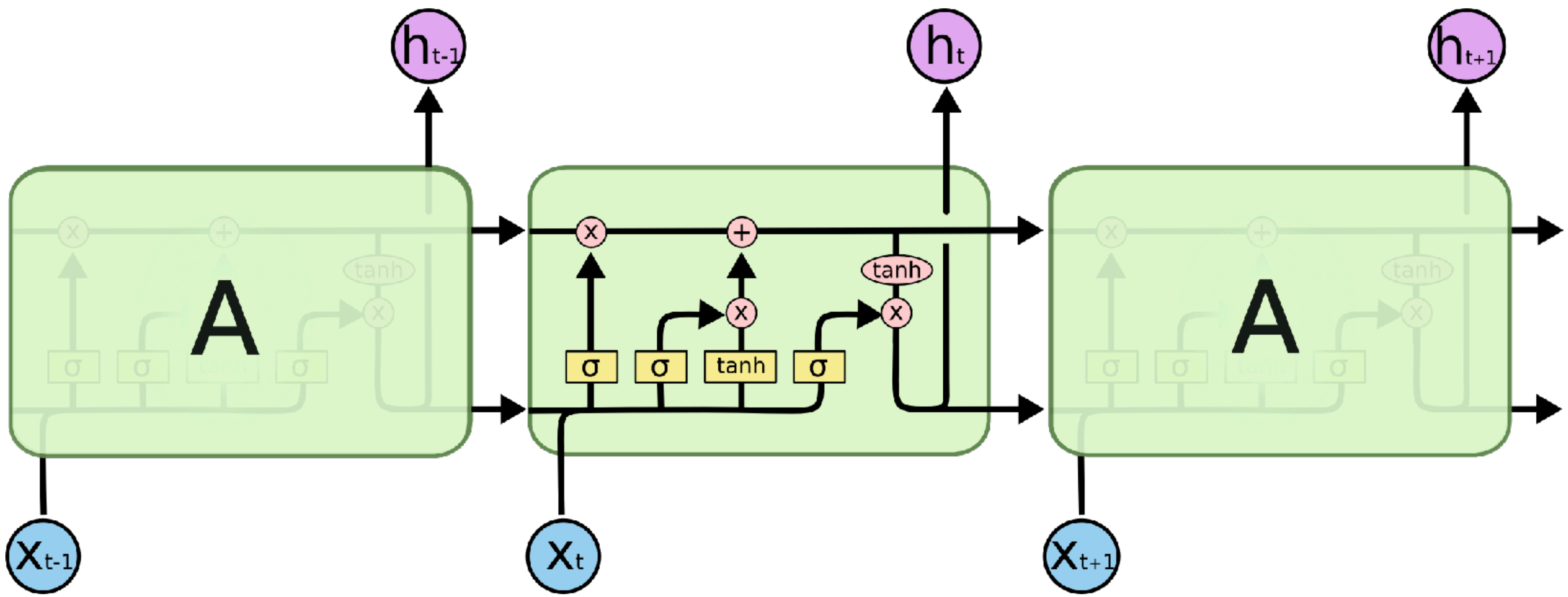
https://www.mdpi.com/sensors/sensors-19-04933/article_deploy/html/images/sensors-19-04933-g001.png

Recurrent Neural Network



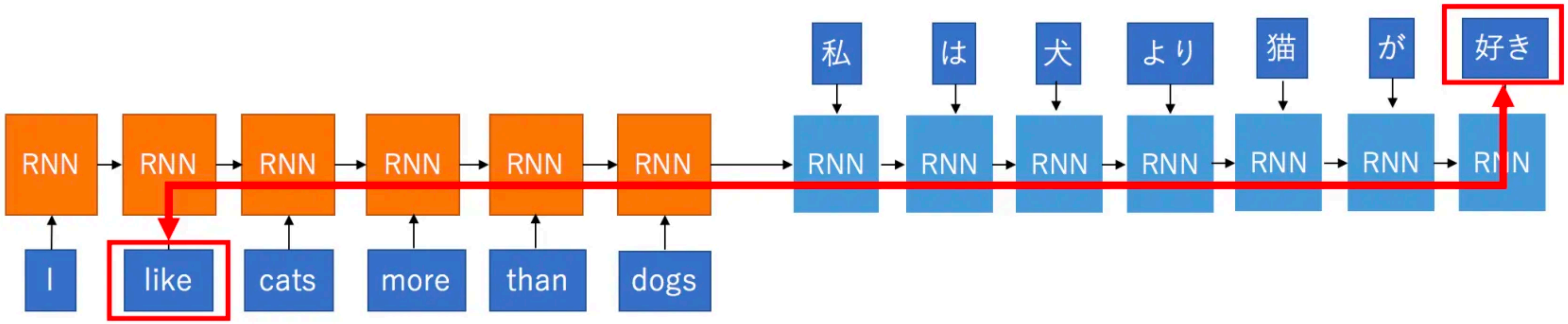
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

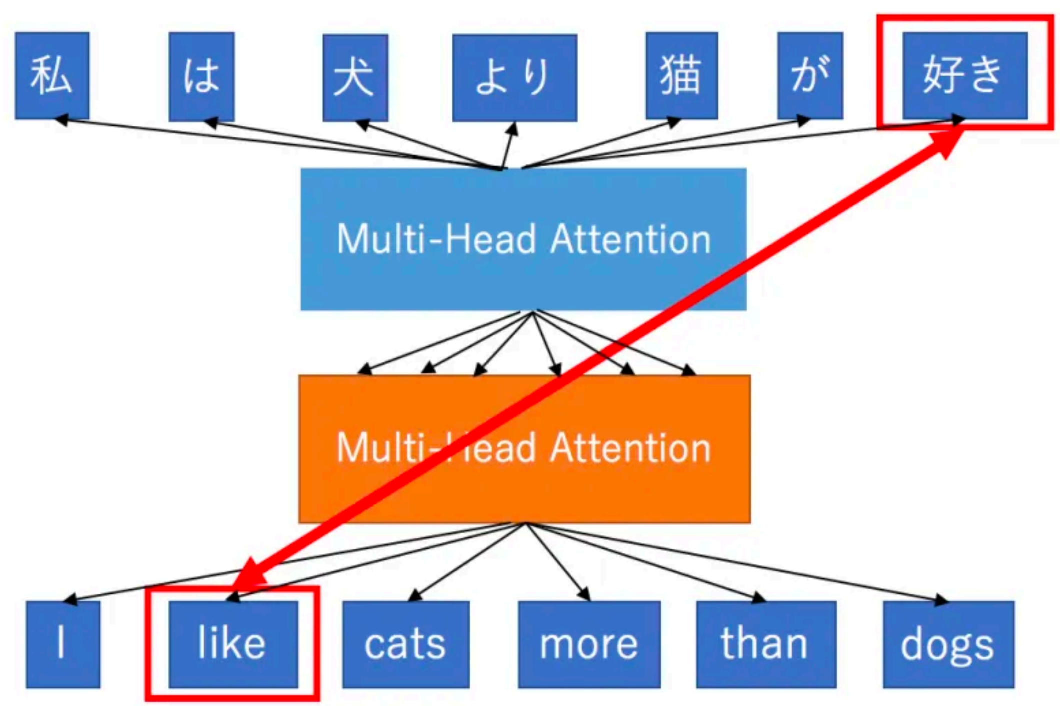
Long Short Term Memory



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

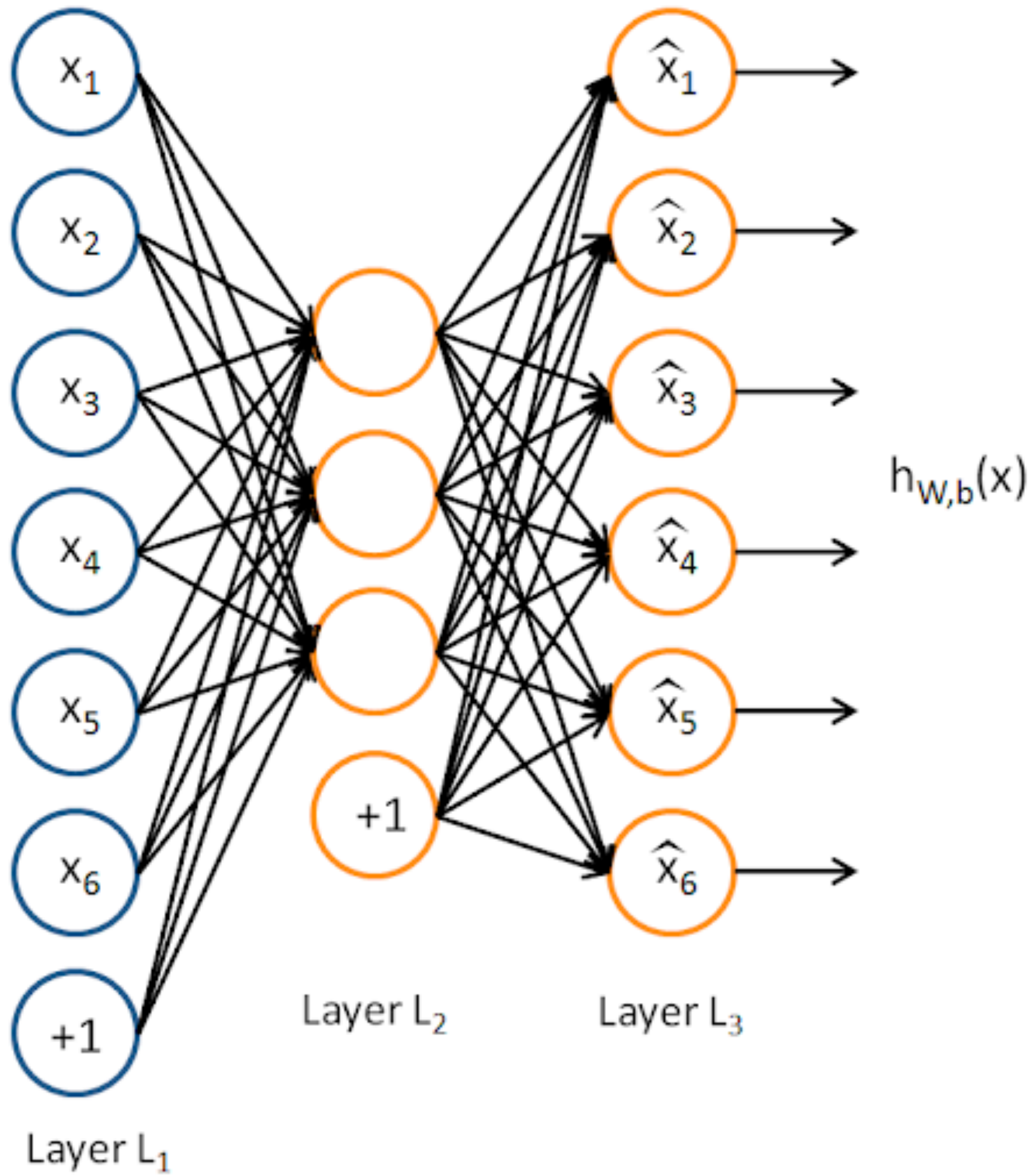
Transformer





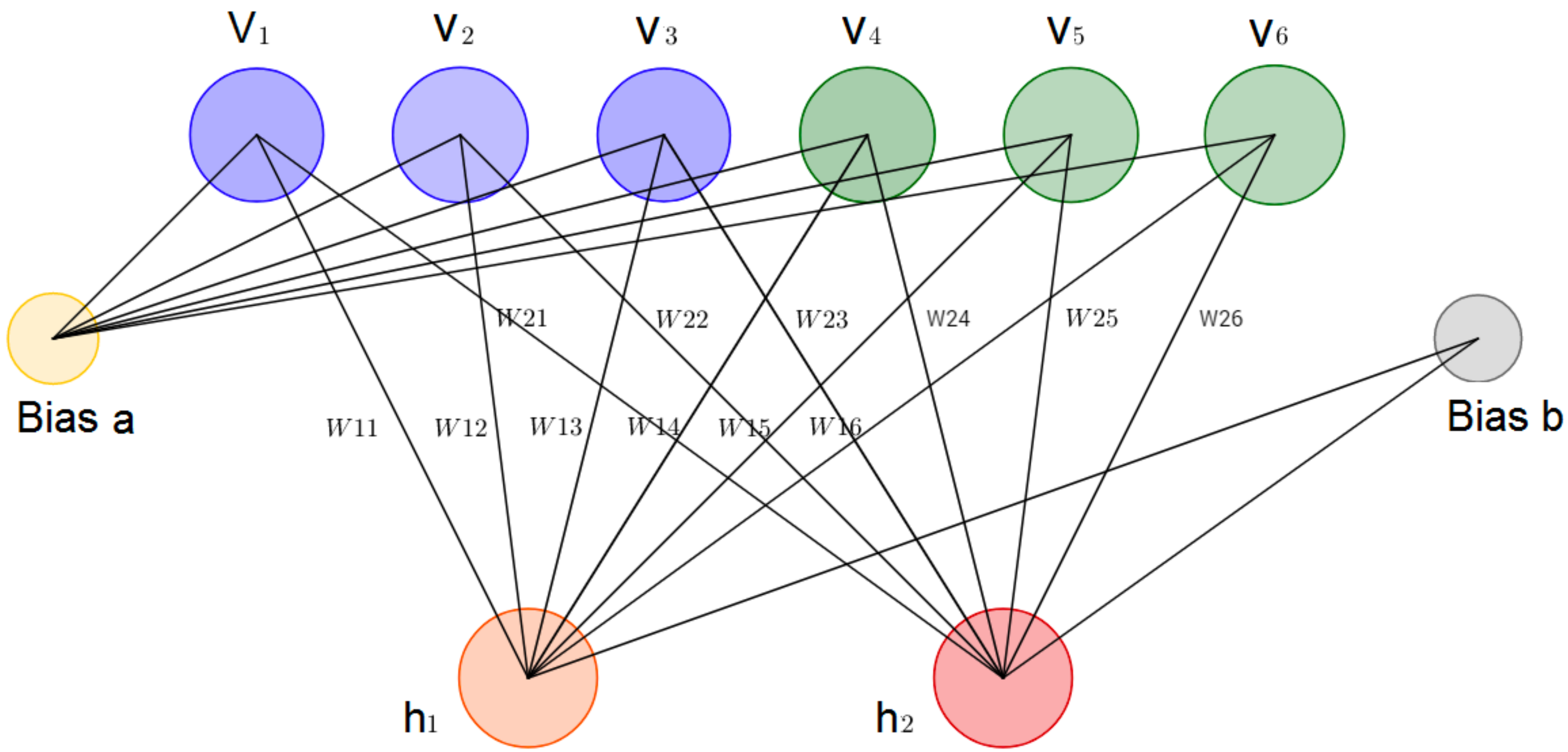
source: www.mlexplained.com

Autoencoder Network



http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

Restricted Boltzmann Machine



$$E(\mathbf{v},\mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

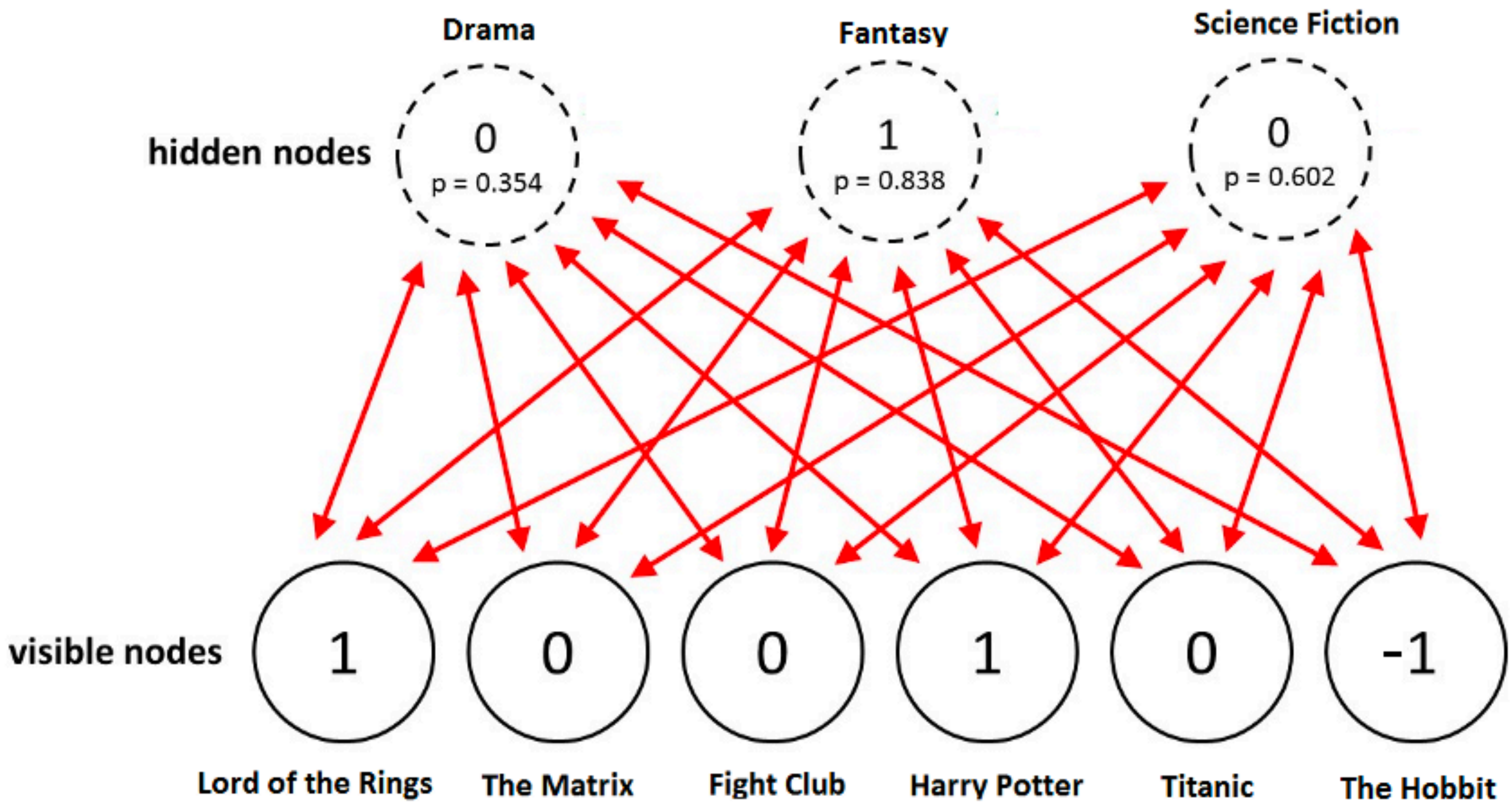
$$p(h_j = 1|\mathbf{v}) = \frac{1}{1+e^{(-(b_j+W_j v_i))}} = \sigma(b_j + \sum_i v_i w_{ij})$$

$$p(v_i = 1|\mathbf{h}) = \frac{1}{1+e^{(-(a_i+W_i h_j))}} = \sigma(a_i + \sum_j h_j w_{ij})$$

Gibbs sampling + contrastive divergence

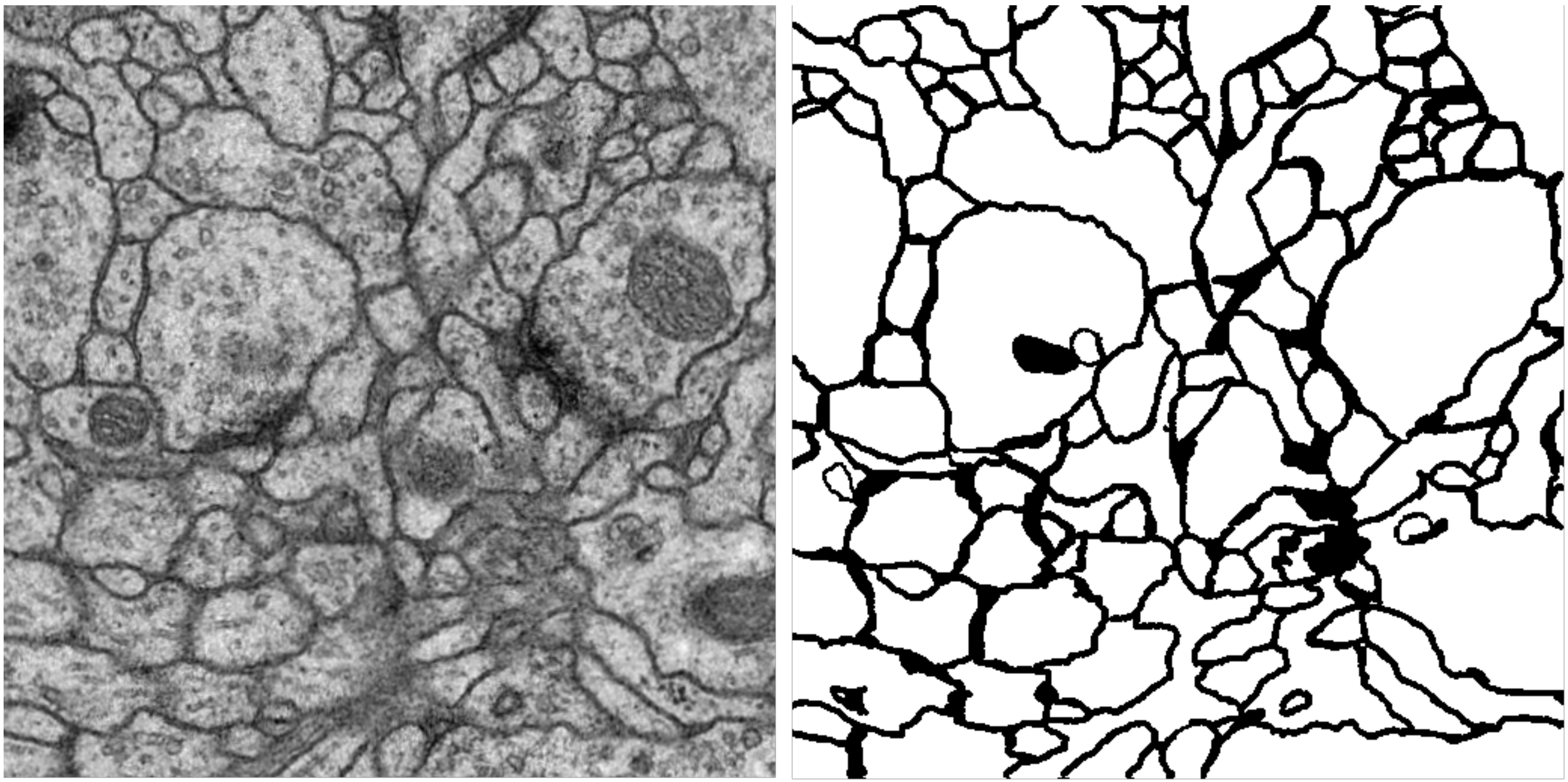
https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15

Collaborative filtering with Restricted Boltzmann Machine



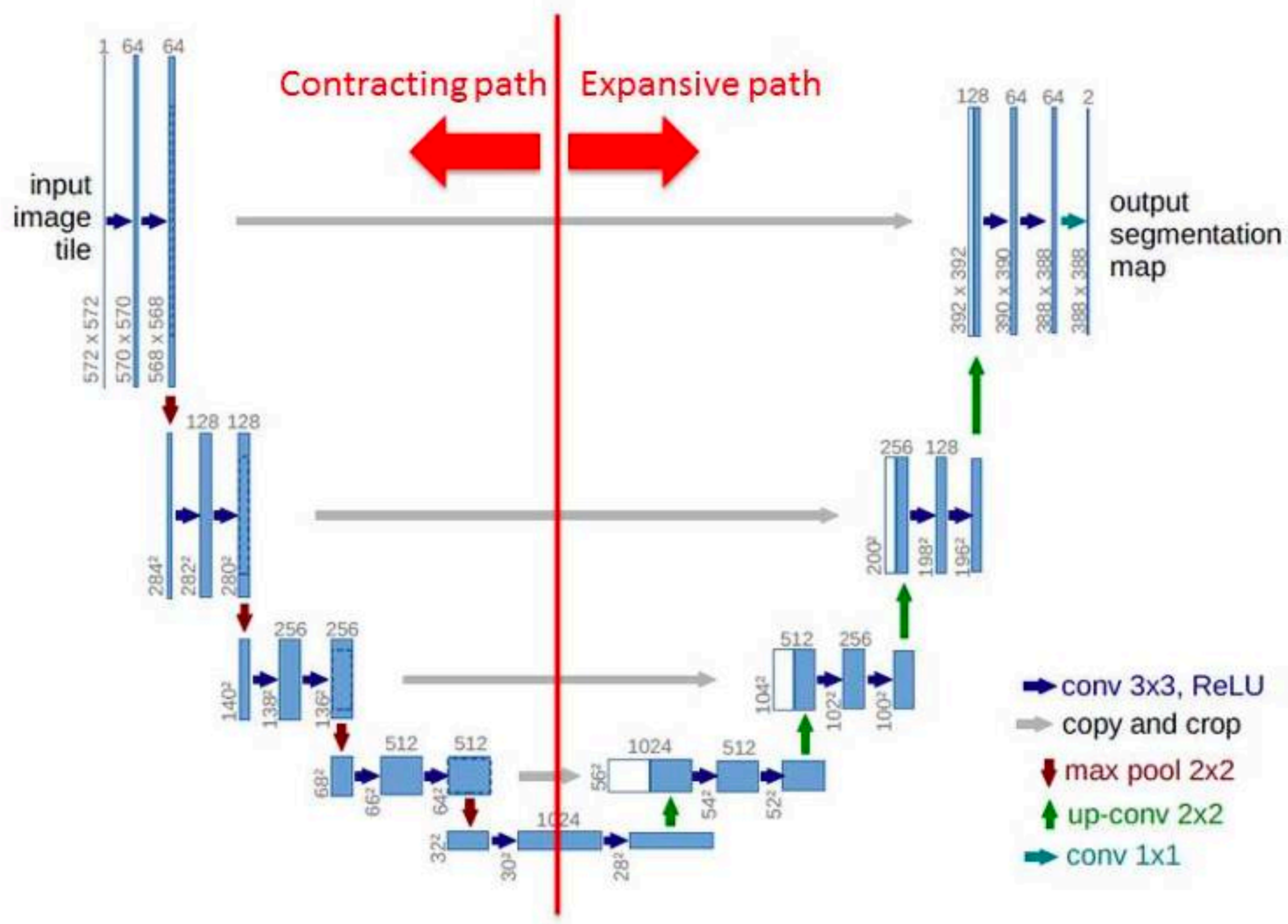
https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15

Image segmentation



U-Net

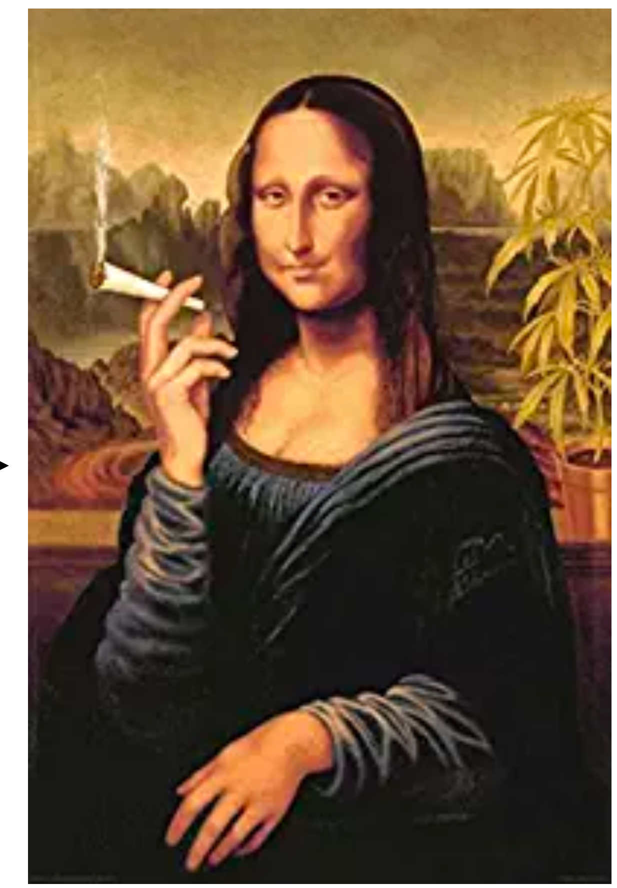
Network Architecture



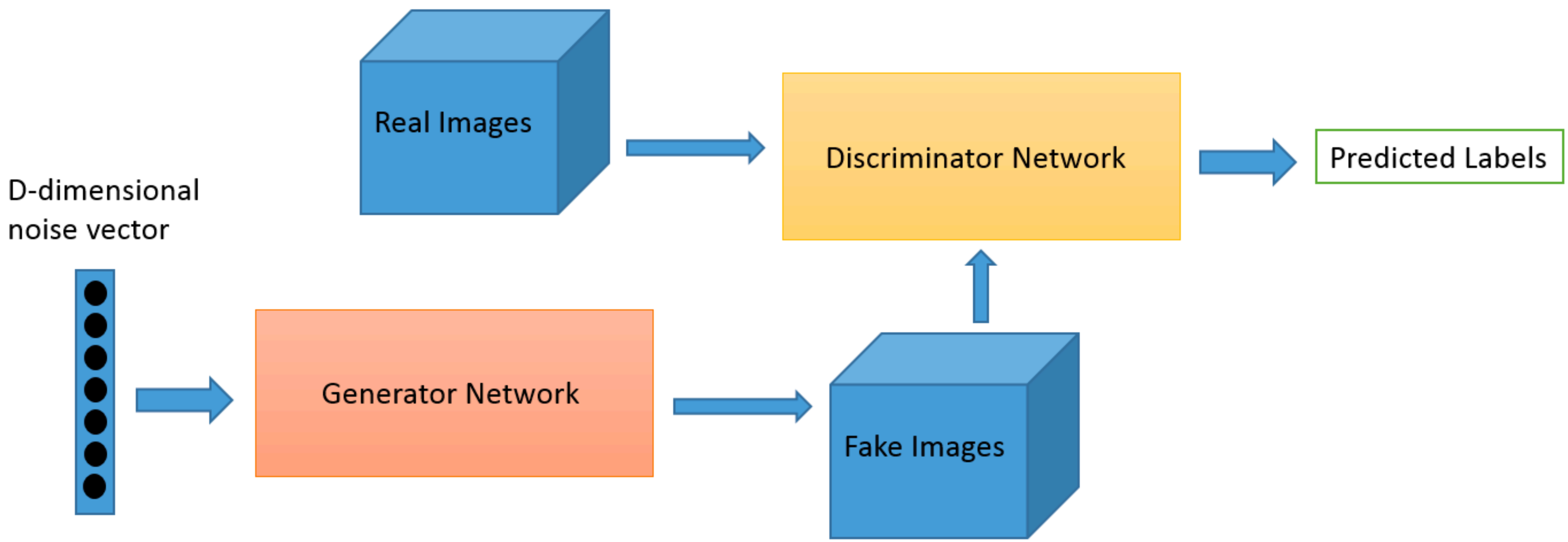
Generative models with neural networks

Random noice

Artificial Neural network



Generative Adversarial Networks



Superresolution

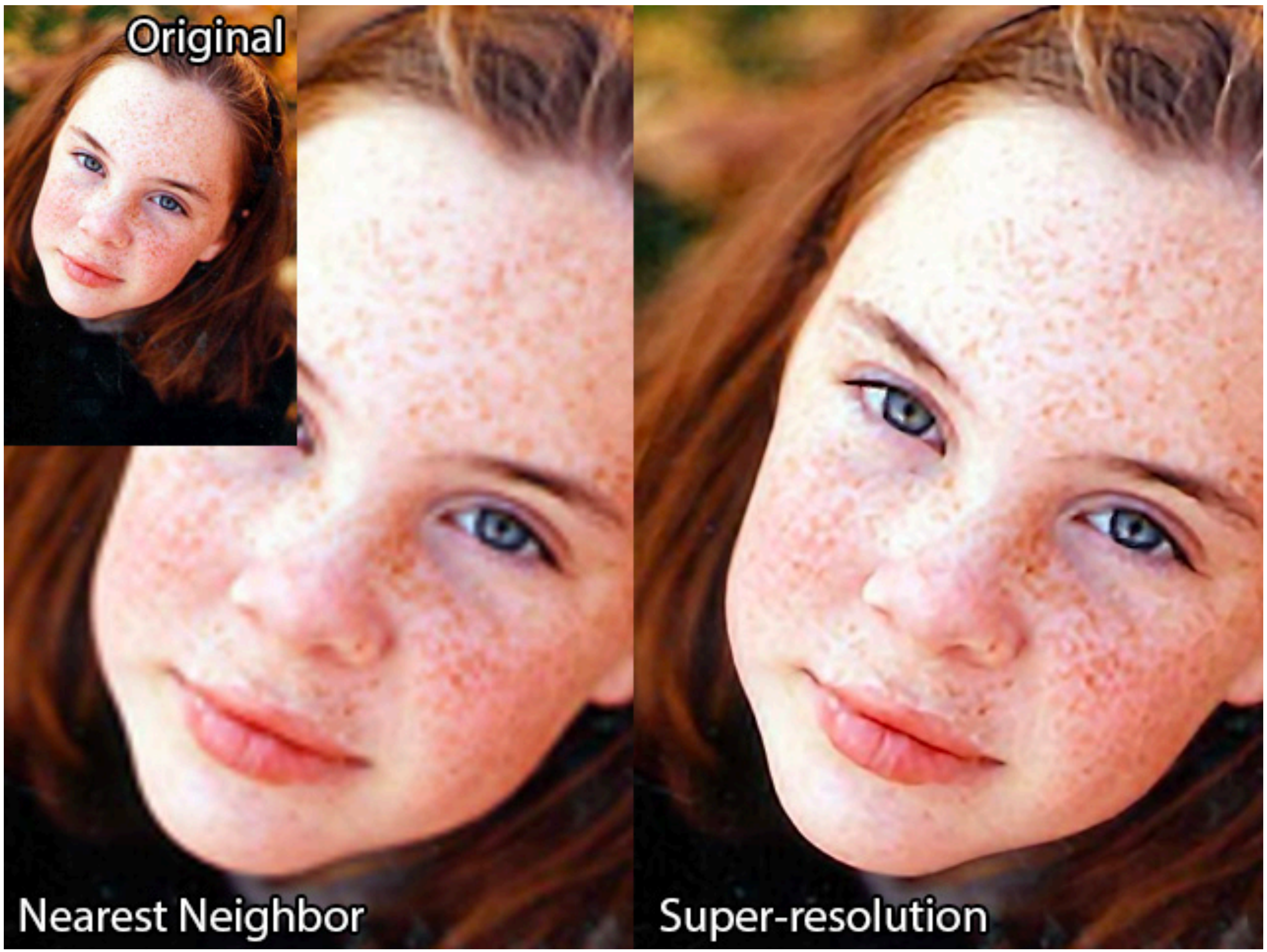
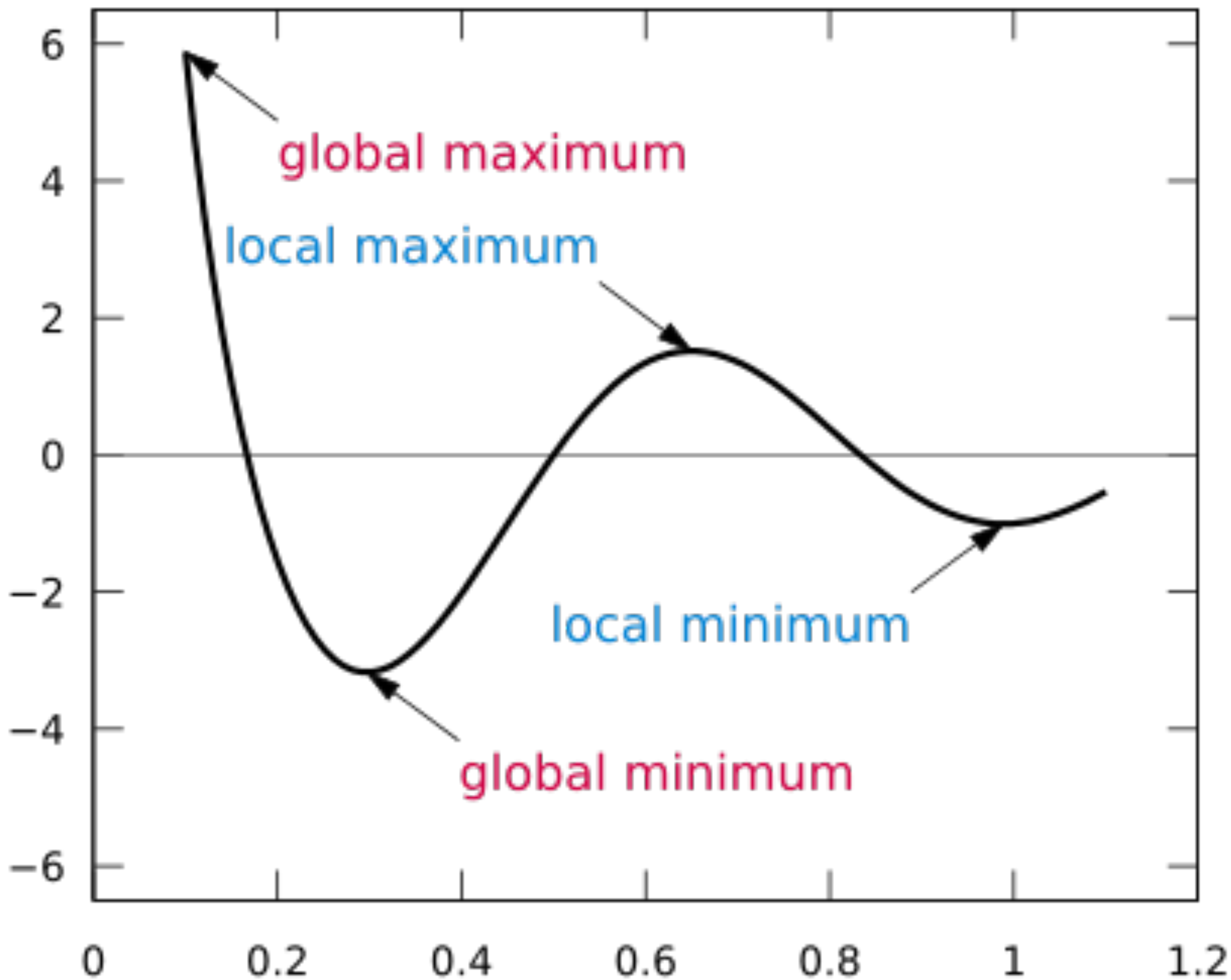


Image manipulation

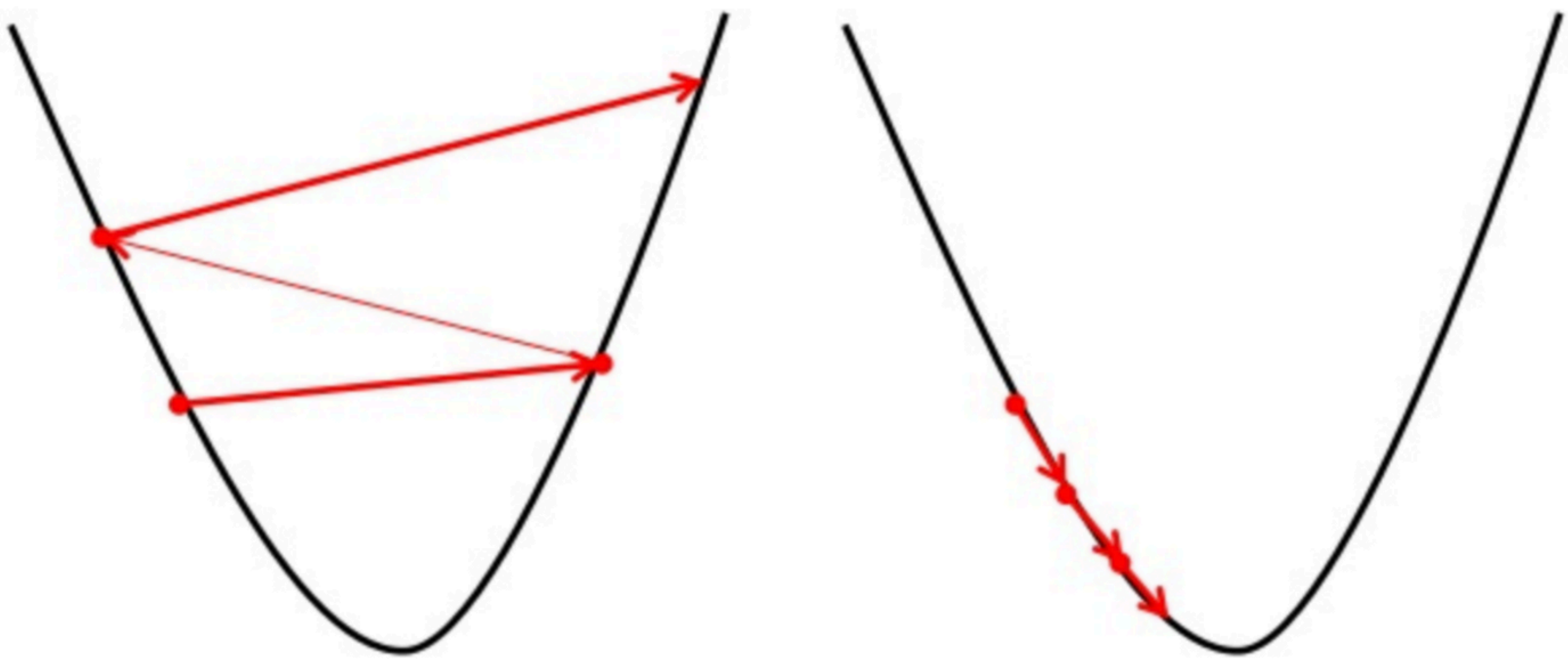
Parameter optimization strategies



https://heartbeat.fritz.ai/an-empirical-comparison-of-optimizers-for-machine-learning-models-b86f29957050

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

Naive momentum

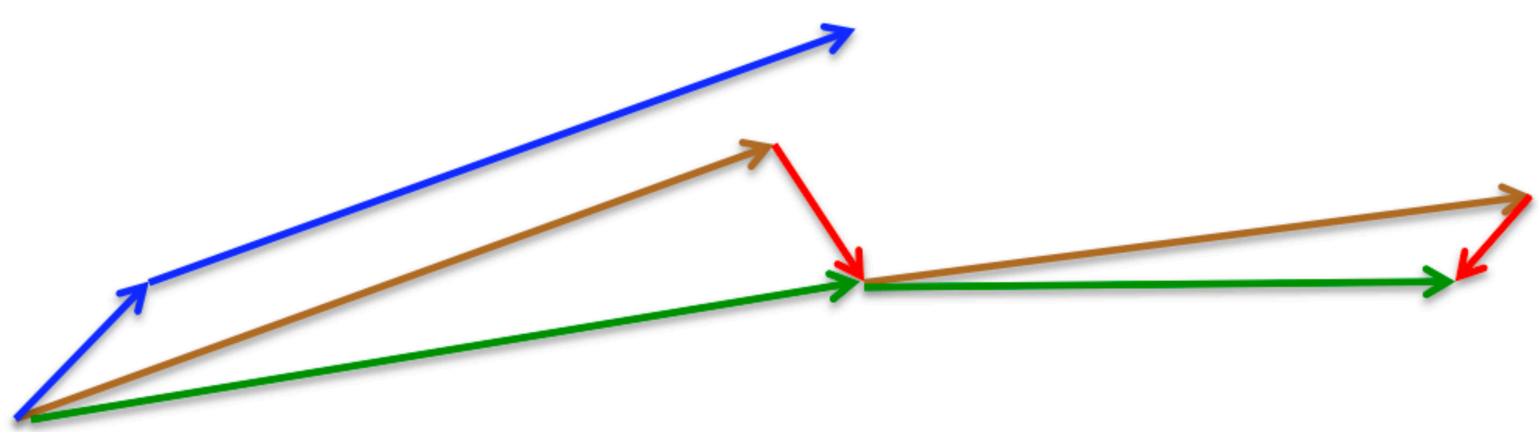
$$v_t = \gamma v_{t-1} + \lambda \frac{\partial e(w_t)}{\partial w_t}$$

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

Nesterov Accelerated Gradient

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

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



https://ruder.io/optimizing-gradient-descent/index.html#nesterovacceleratedgradient

Adaptive Gradient Algorithms

Adagrad

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

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

RMSProp

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

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

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)

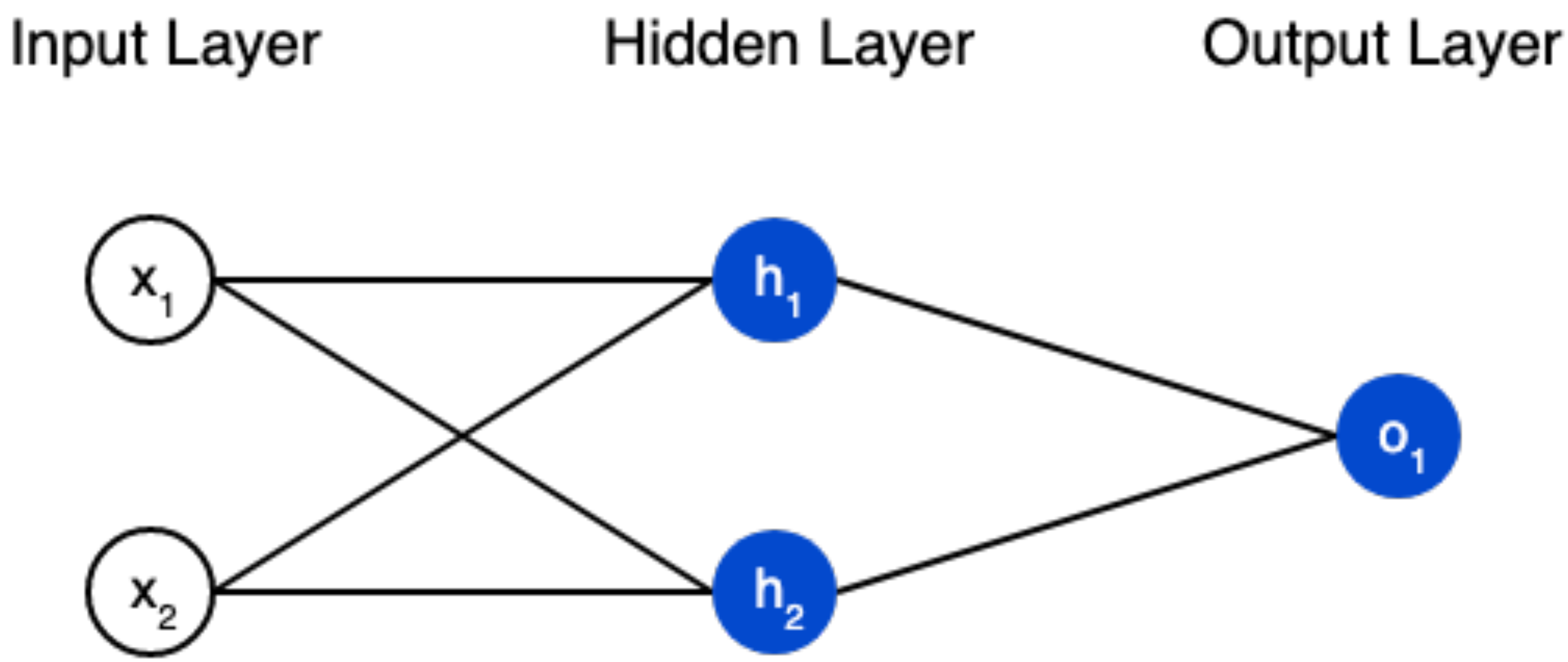
Categorical Cross Entropy $$CCE = -\frac{\sum_{i=1}^{n}\sum_{j=1}^{c} y_{i,j} log(\hat{y}_{i,j})}{n}$$

Binary Cross Entropy

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

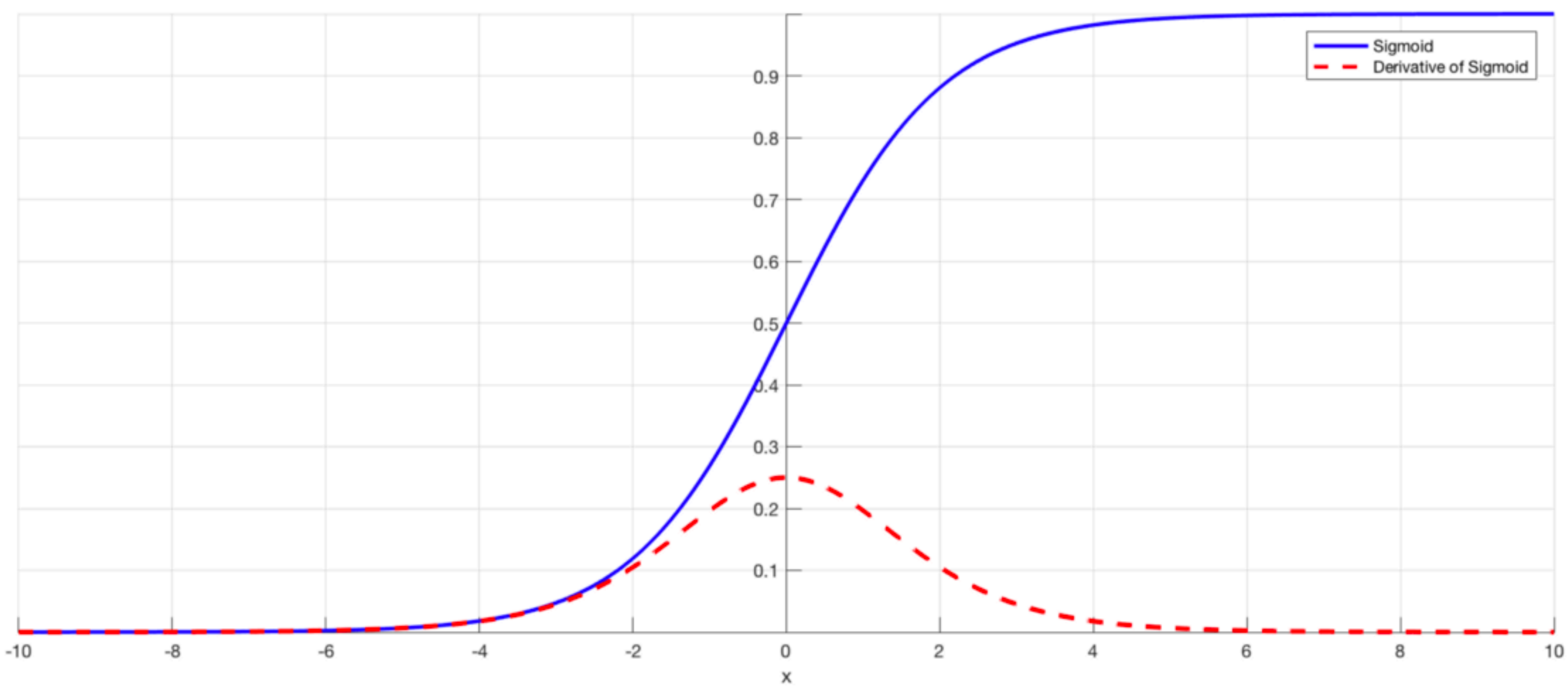
Weight initialization

Zero or constant initialization



https://victorzhou.com/blog/intro-to-neural-networks/

Too low or too high initialization



https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e

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

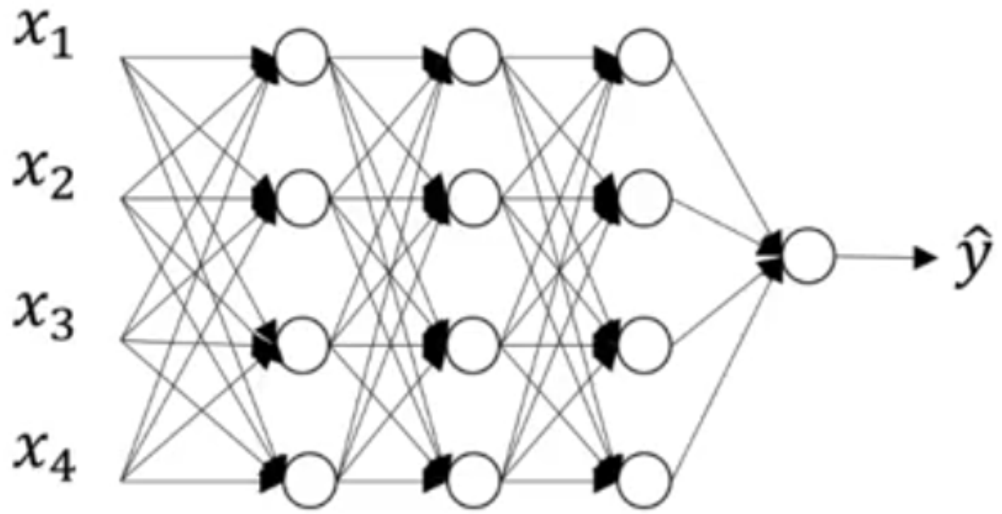
04-Regression-nn-assignment.ipynb

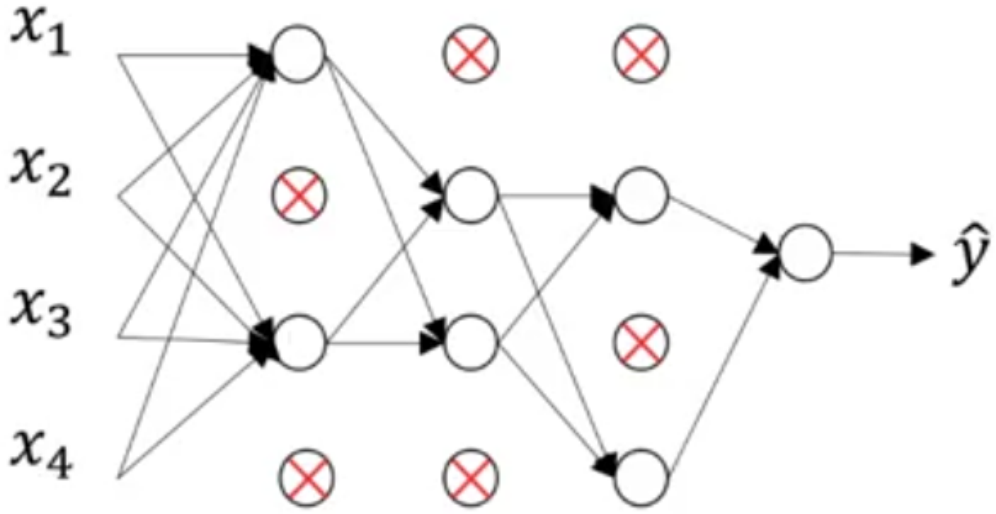
L2 Regularization in deep learning

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

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

Dropout





https://towardsdatascience.com/how-to-improve-a-neural-network-with-regularization-8a18ecda9fe3

Functional API in Keras

```
# Sequential model
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(10, input_shape=(10,),activation=‘relu')
model.add(Dense(20, activation='relu')
model.add(Dense(10, activation=‘relu')
model.add(Dense(1, activation=‘sigmoid')
```

```
# Functional model
from keras.models import Model
from keras.layers import Input, Dense

visible = Input(shape=(10,))
hidden1 = Dense(10, activation='relu')(visible)
hidden2 = Dense(20, activation='relu')(hidden1)
hidden3 = Dense(10, activation='relu')(hidden2)
output = Dense(1, activation='sigmoid')(hidden3)
model = Model(inputs=visible, outputs=output)
```

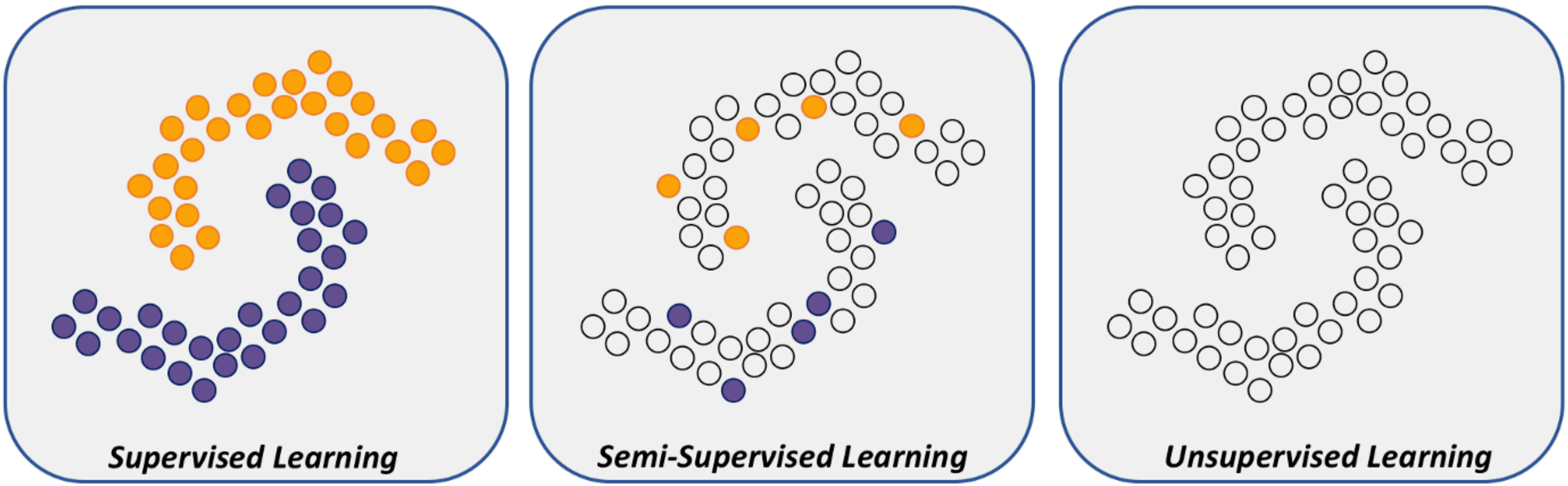
Shared Input

```
# Shared Input Layer
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input, Dense, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
# input layer
visible = Input(shape=(64,64,1))
# first feature extractor
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
flat1 = Flatten()(pool1)
# second feature extractor
conv2 = Conv2D(16, kernel_size=8, activation='relu')(visible)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat2 = Flatten()(pool2)
# merge feature extractors
merge = concatenate([flat1, flat2])
# interpretation layer
hidden1 = Dense(10, activation='relu')(merge)
# prediction output
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```

Multiple inputs (outputs)

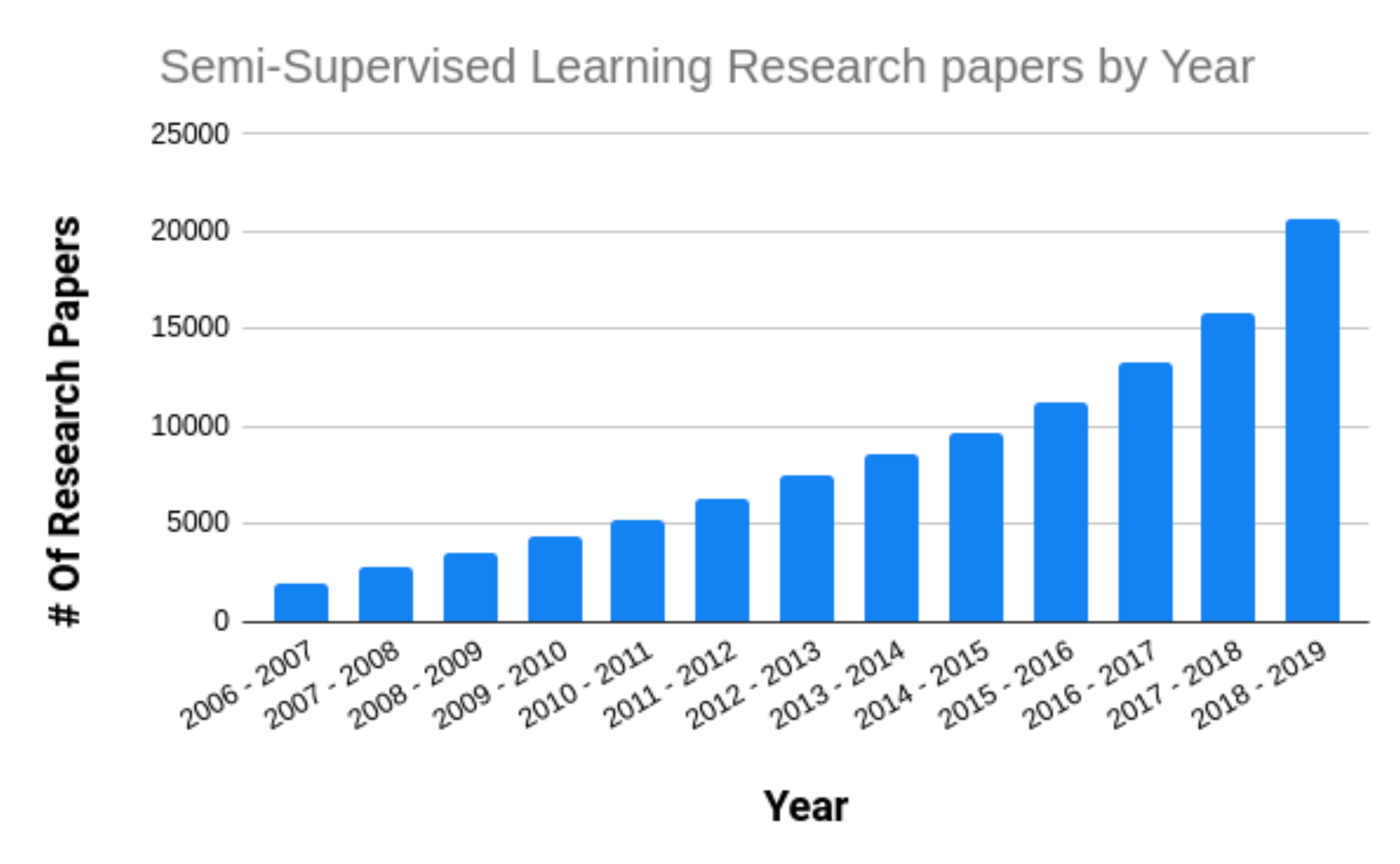
```
# Multiple Inputs
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
# first input model
visible1 = Input(shape=(64,64,1))
conv11 = Conv2D(32, kernel_size=4, activation='relu')(visible1)
pool11 = MaxPooling2D(pool_size=(2, 2))(conv11)
conv12 = Conv2D(16, kernel_size=4, activation='relu')(pool11)
pool12 = MaxPooling2D(pool_size=(2, 2))(conv12)
flat1 = Flatten()(pool12)
# second input model
visible2 = Input(shape=(32,32,3))
conv21 = Conv2D(32, kernel_size=4, activation='relu')(visible2)
pool21 = MaxPooling2D(pool_size=(2, 2))(conv21)
conv22 = Conv2D(16, kernel_size=4, activation='relu')(pool21)
pool22 = MaxPooling2D(pool_size=(2, 2))(conv22)
flat2 = Flatten()(pool22)
# merge input models
merge = concatenate([flat1, flat2])
# interpretation model
hidden1 = Dense(10, activation='relu')(merge)
hidden2 = Dense(10, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
model = Model(inputs=[visible1, visible2], outputs=output)
```

Semi-supervised learning



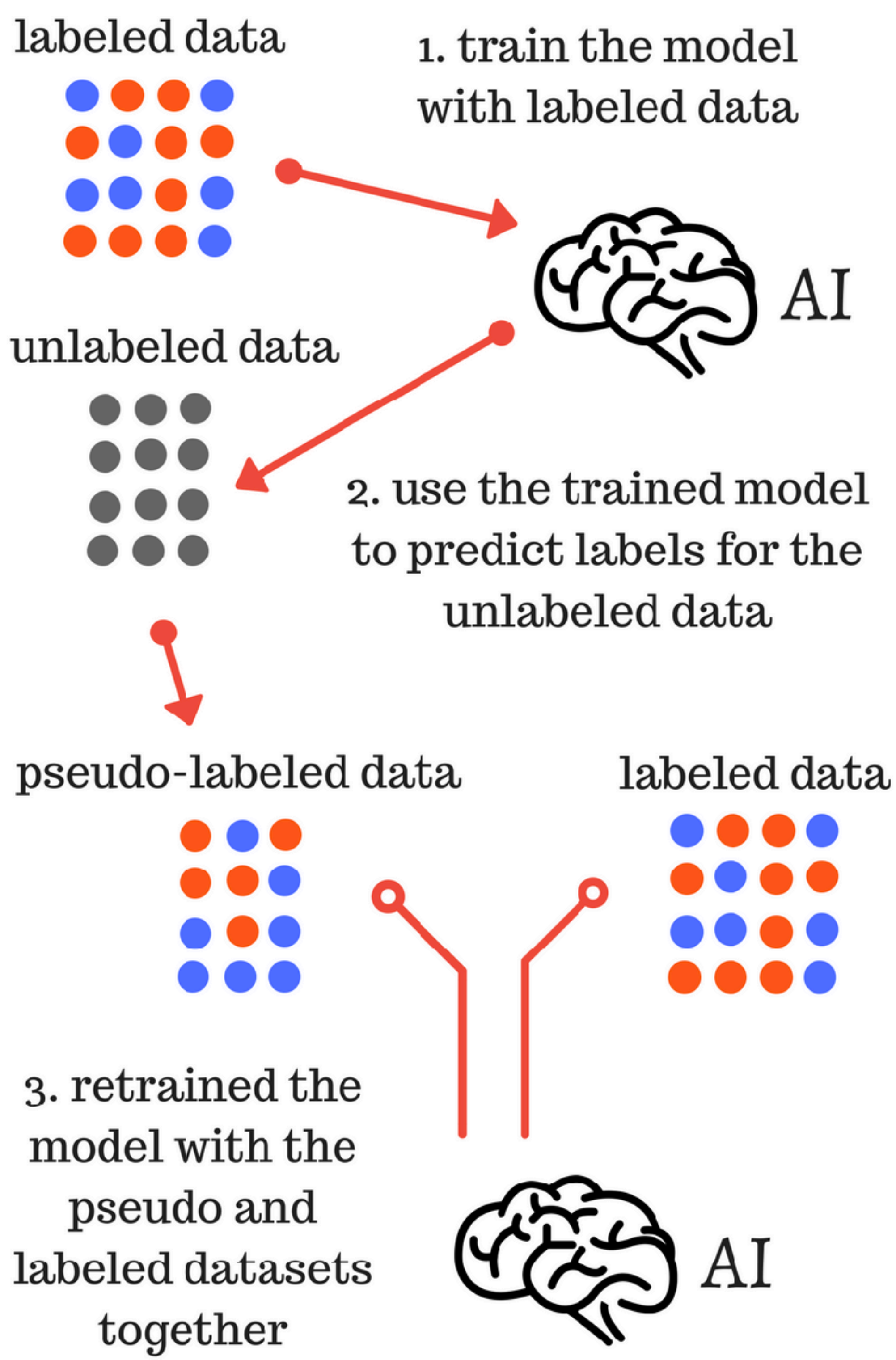
Source: KDnuggets.

Small Data is the new Big Data



https://towardsdatascience.com/pseudo-labeling-to-deal-with-small-datasets-what-why-how-fd6f903213af

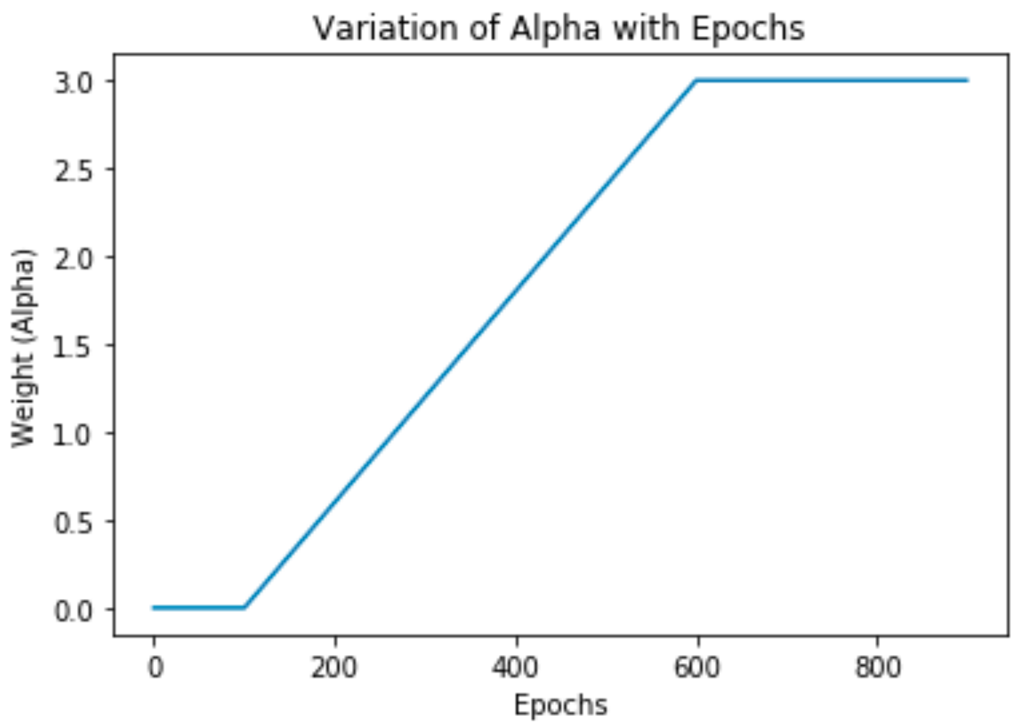
Pseudo-labeling



https://datawhatnow.com/pseudo-labeling-semi-supervised-learning/

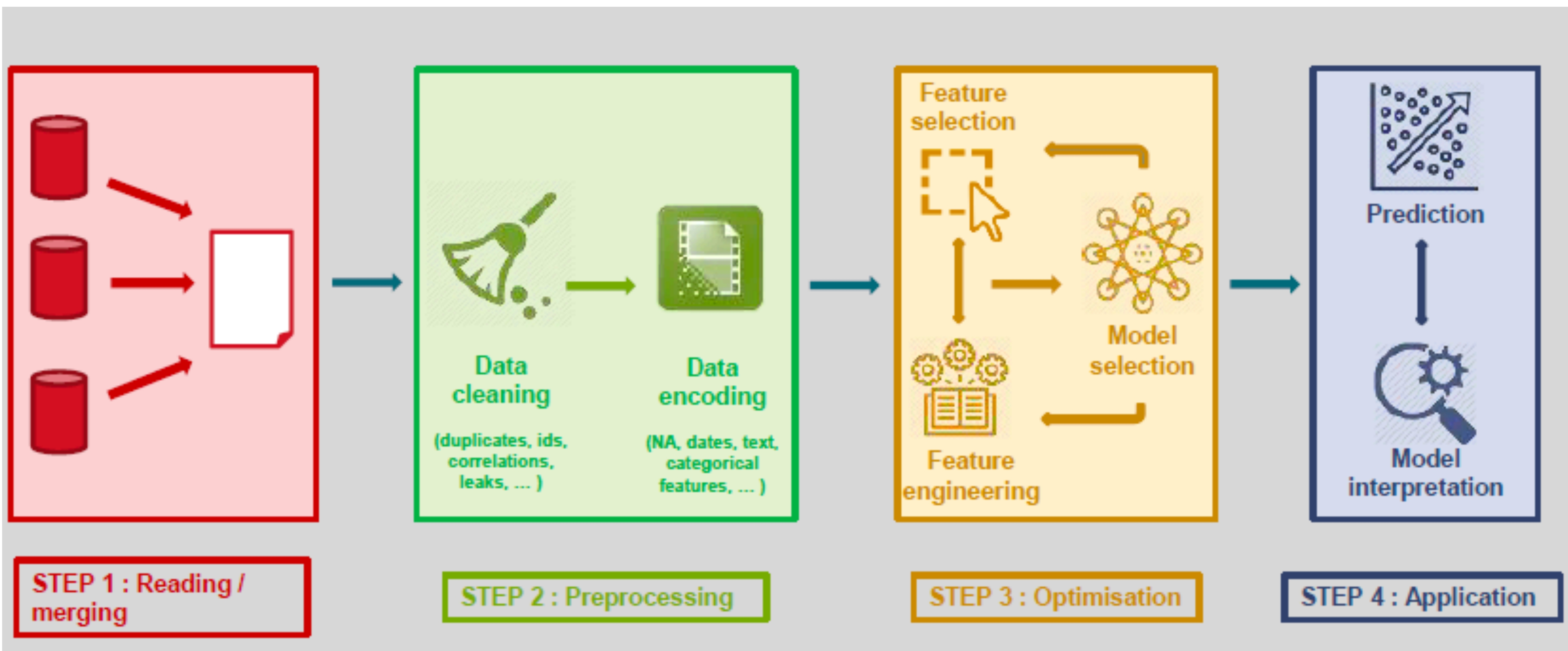
Pseudo-labeling with neural networks

- 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 α-weighted unlabeled loss and backpropagate
- Repeat



https://towardsdatascience.com/pseudo-labeling-to-deal-with-small-datasets-what-why-how-fd6f903213af

AutoML approaches



https://www.slideshare.net/AxeldeRomblay?utm_campaign=profiletracking&utm_medium=sssite&utm_source=ssslideview

AutoML frameworks

H2O AutoML

Google's AutoML

AutoGluon

bigml

Microsoft Azure

HYPEROPT

MLBox, Machine Learning Box

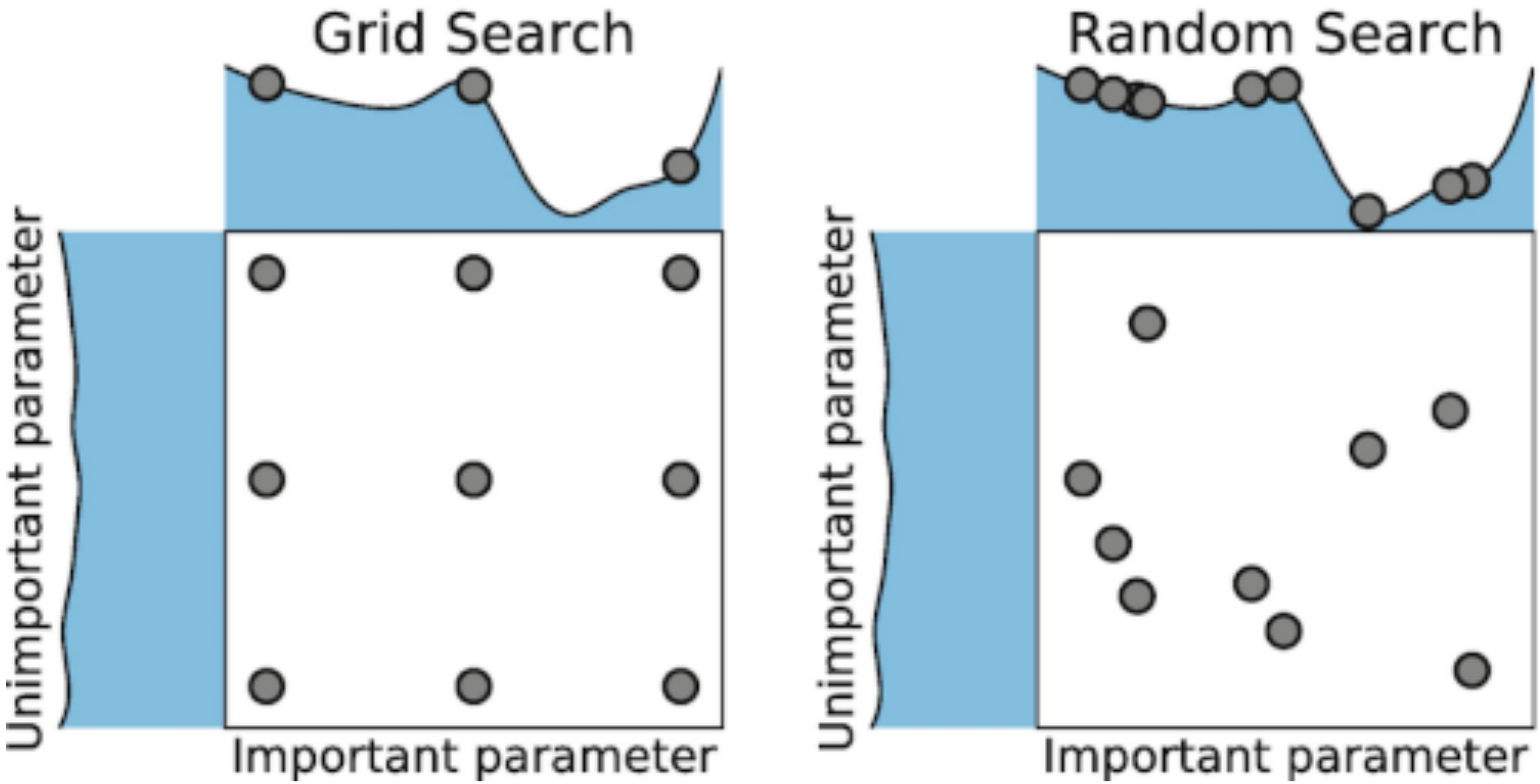
AutoKeras

Hyper-parameters tuning

Parameters - learnable weights of the neural network

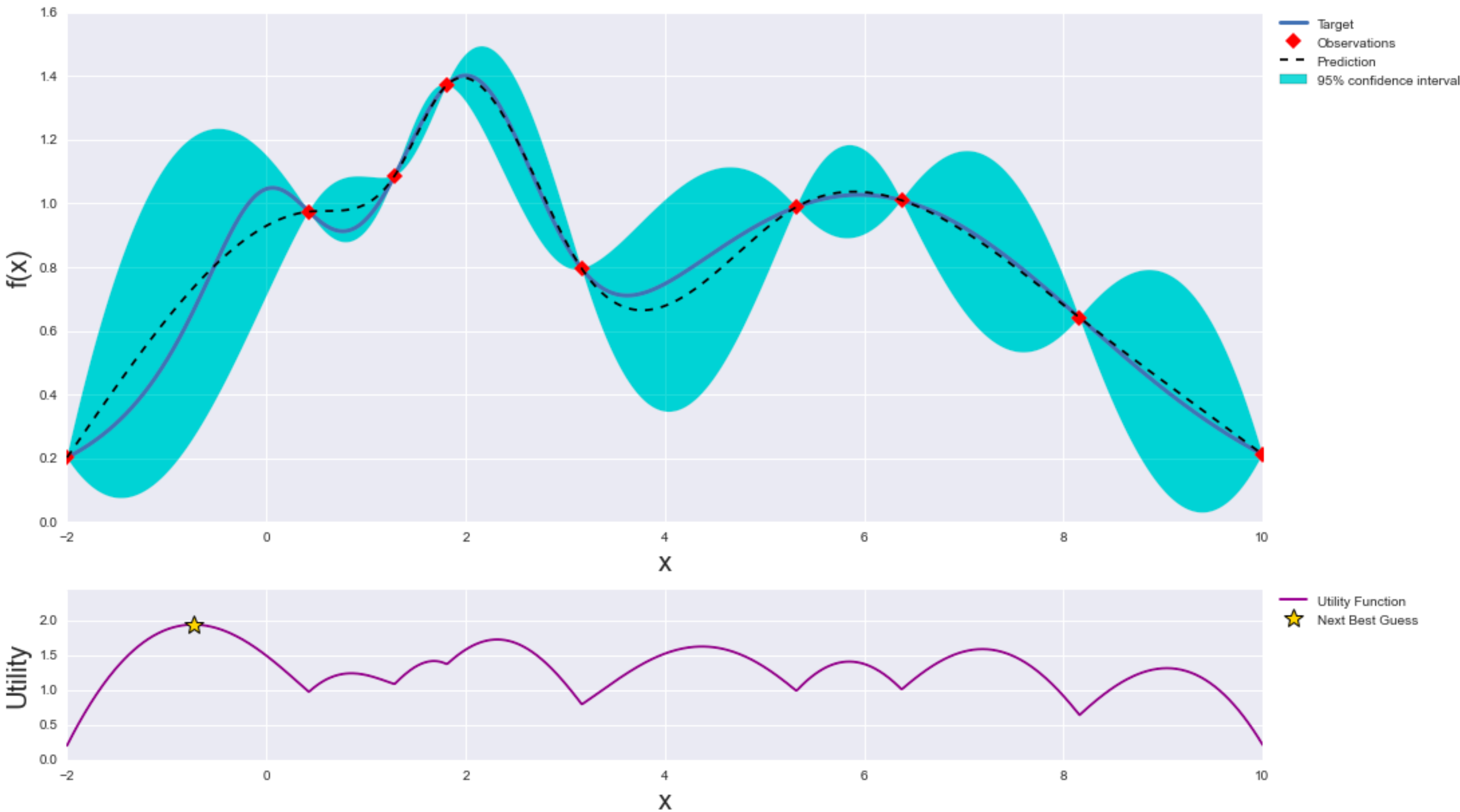
Hyper-parameters - parameters set before training (learning rate, momentum, dropout rate, etc.)

Grid search vs random search



https://medium.com/@cjl2fv/an-intro-to-hyper-parameter-optimization-using-grid-search-and-random-search-d73b9834ca0a

Bayesian optimization using Gaussian processes



https://medium.com/@hiepnguyen034/improving-neural-networks-performance-with-bayesian-optimization-efbaa801ad26

Neural Architecture Search (NAS)

https://towardsdatascience.com/neural-architecture-search-nas-the-future-of-deep-learning-c99356351136

AutoML with AutoKeras

06-Autokeras.ipynb