Data analysis and visualization (DAV)

Lecture 11
Statistics & machine learning
Part 3

Łukasz P. Kozłowski

Warsaw, 2025

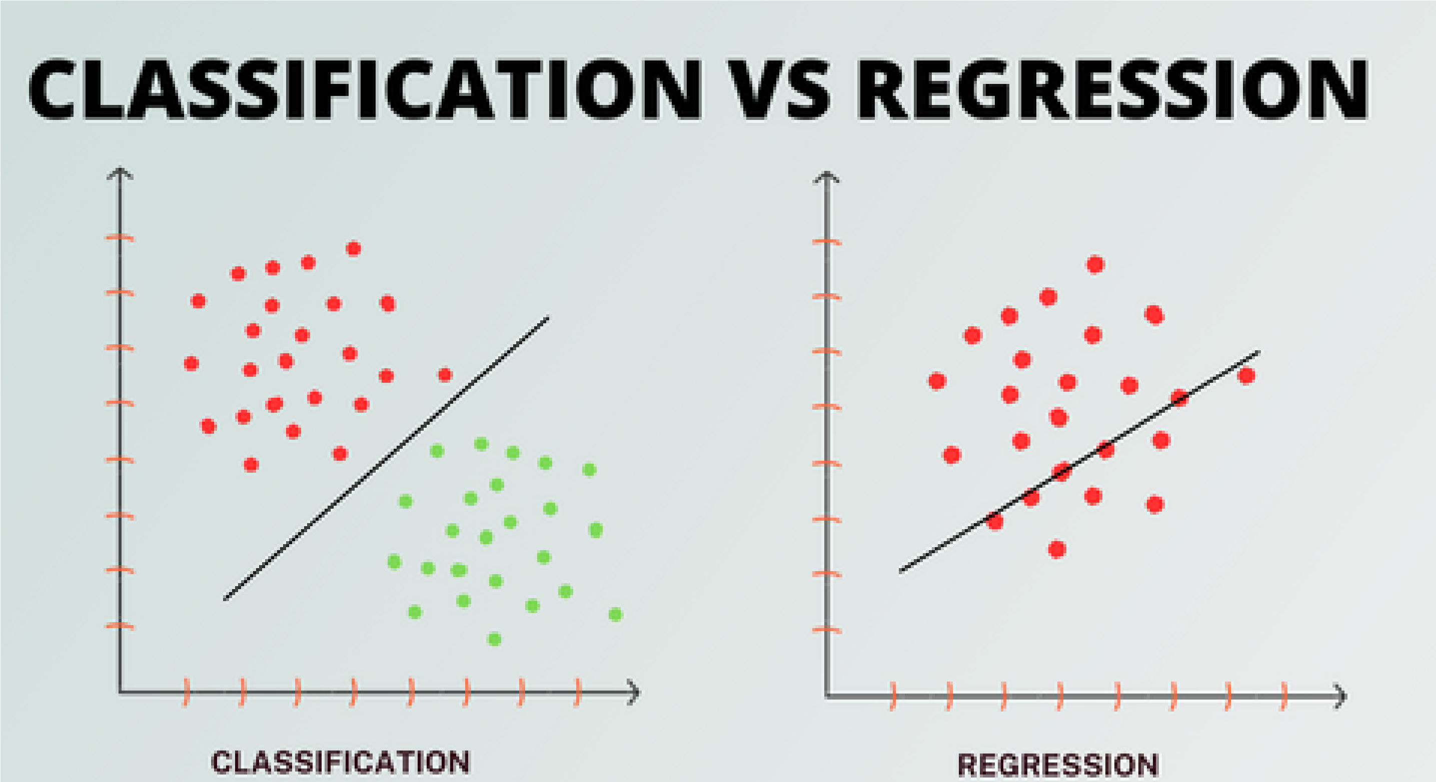
l.kozlowski@mimuw.edu.pl

Supervised Machine Learning Algorithms

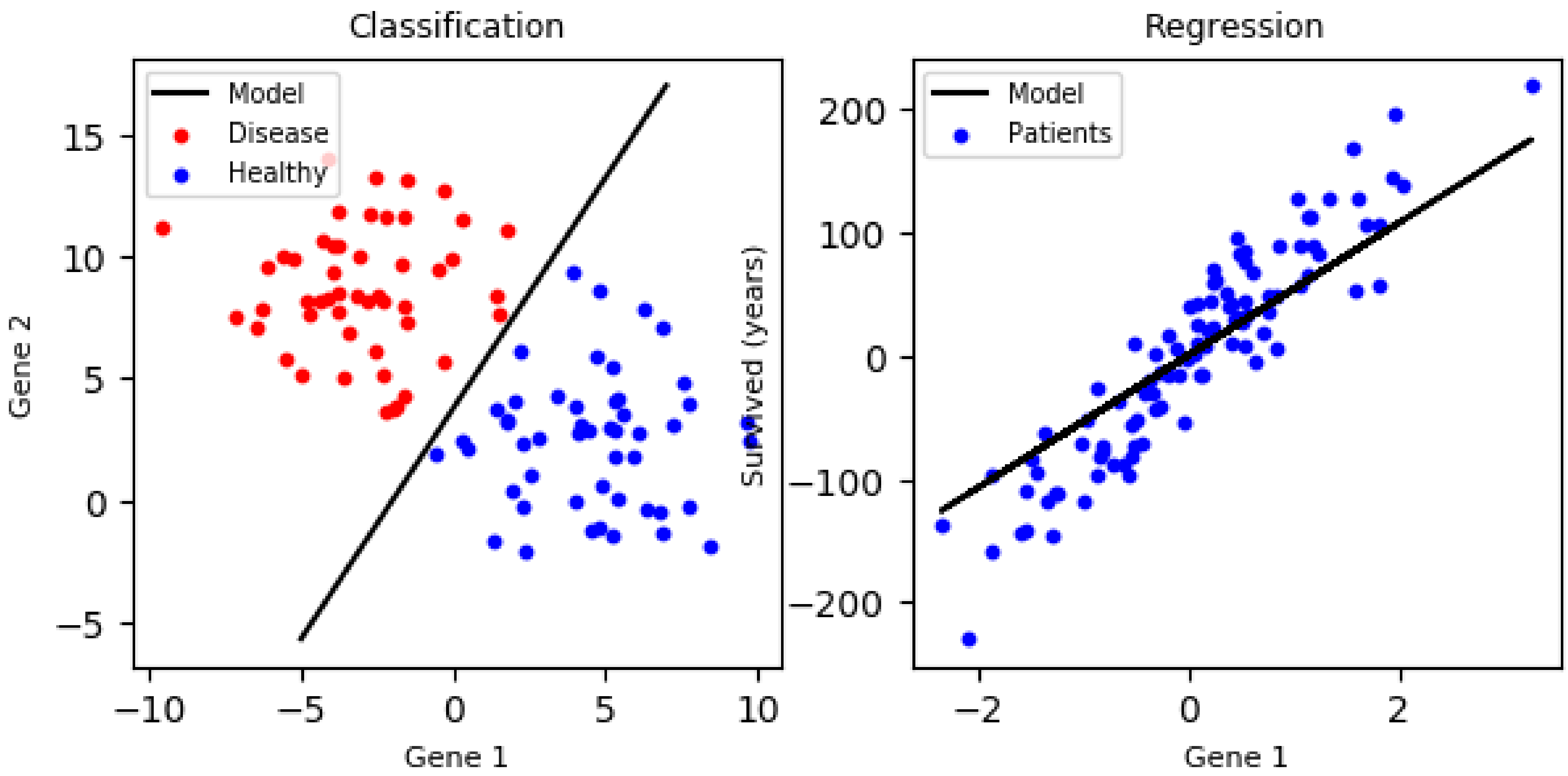
This lecture covers monst commonly used algorithms (obviously the list is not complete and subjective). We will cover and briefly describe:

- Decision tree
- Random forest
- Nearest Neighbors
- Support Vector Machines
- Neural networks (deep learning)

Note: we will focus rather on practical issues than mathematical formulas



	Regression	Classification
Description	A a regression model seeks to predict a continuous quantity.	A classification model seeks to predict some class label.
Type of algorithm	Supervised learning algorithm	Supervised learning algorithm
Type of response variable	Continuous	Categorial
How to assess model fit	Root mean squared error	Percentage of correct classifications



Classification
Model
Disease
Healthy
Gene 2
Gene 1
Regression
Model
Patients
Survived (years)
Gene 1

Converting Regression into Classification

A regression problem can be converted into a classification problem by simply **discretizing** the response variable into **buckets**.

- $80k$–160k: “Low selling price”
- $161k$–240k: “Medium selling price”
- $241k$–320k: “High selling price”

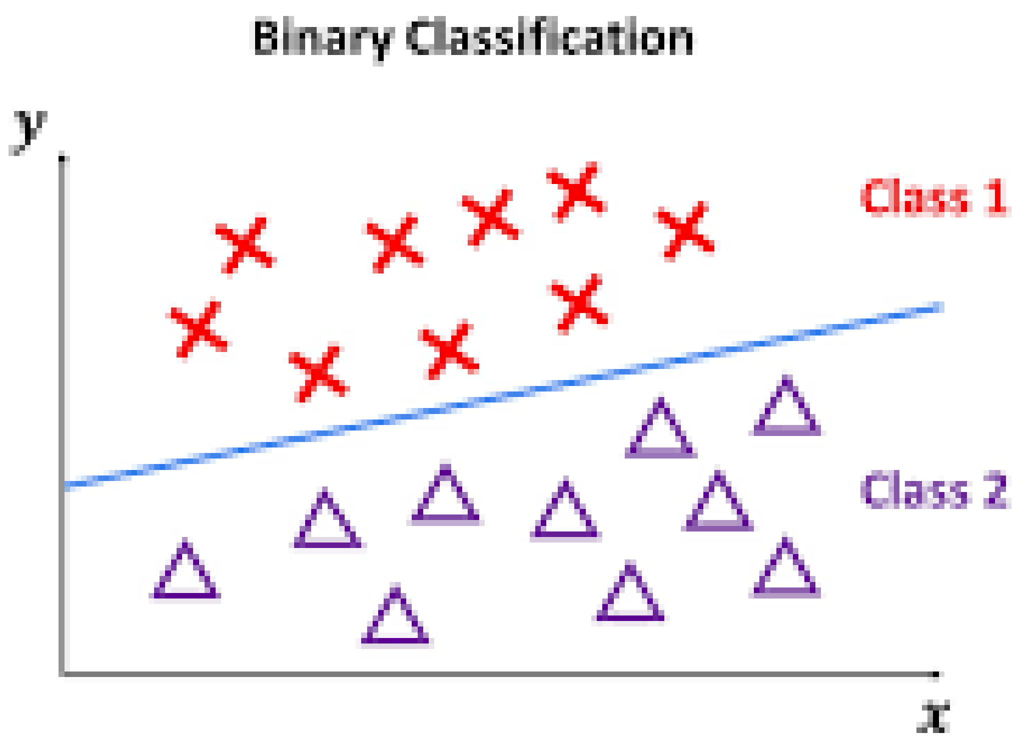
https://www.statology.org/regression-vs-classification/

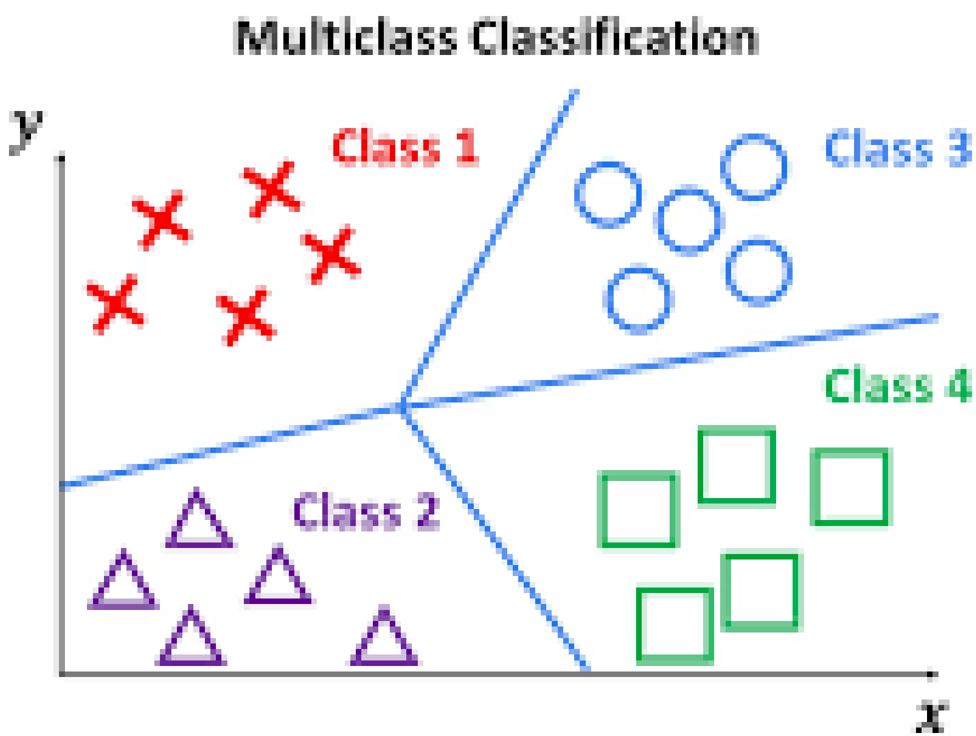
Binary and Multiclass Classification

Application	Observation	0	1
Medical Diagnosis	Patient	Healthy	Diseased
Email Analysis	Email	Not Spam	Spam
Financial Data Analysis	Transaction	Not Fraud	Fraud
Marketing	Website visitor	Won't Buy	Will Buy
Image Classification	Image	Hotdog	Not Hotdog

Binary and Multiclass Classification

Application	Observation	0	1
Medical Diagnosis	Patient	Healthy	Diseased
Email Analysis	Email	Not Spam	Spam
Financial Data Analysis	Transaction	Not Fraud	Fraud
Marketing	Website visitor	Won't Buy	Will Buy





Three Type of Classification Tasks

Binary Classification

- Spam
- Not spam

Multiclass Classification

- Dog
- Cat
- Horse
- Fish
- Bird
- …

Multi-label Classification

- Dog
- Cat
- Horse
- Fish
- Bird
- …

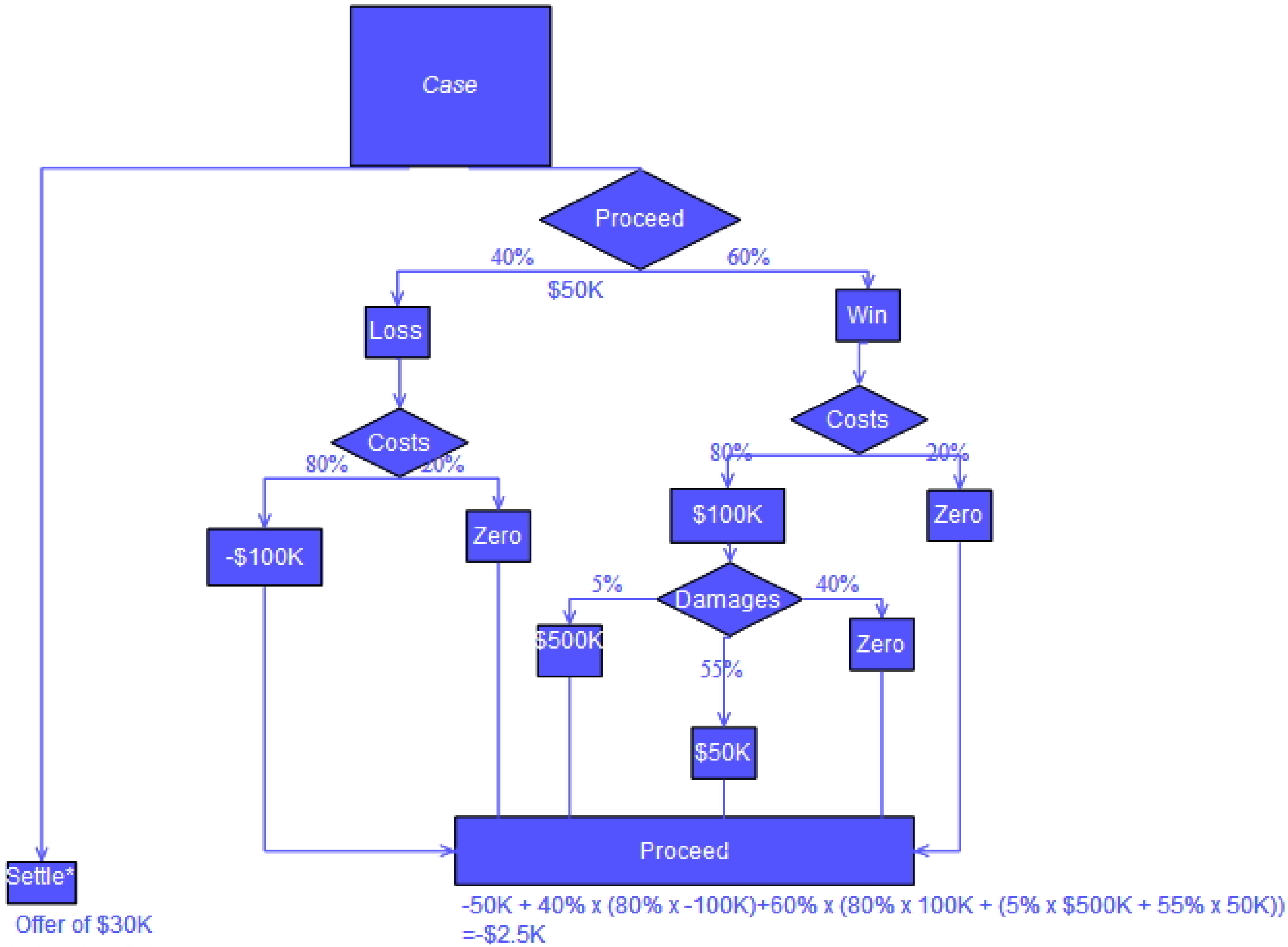
Decision tree

It is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility

A decision tree consists of three types of nodes:

- Decision nodes – typically represented by squares
- Chance nodes – typically represented by circles
- End nodes – typically represented by triangles

Decision tree



Decision tree

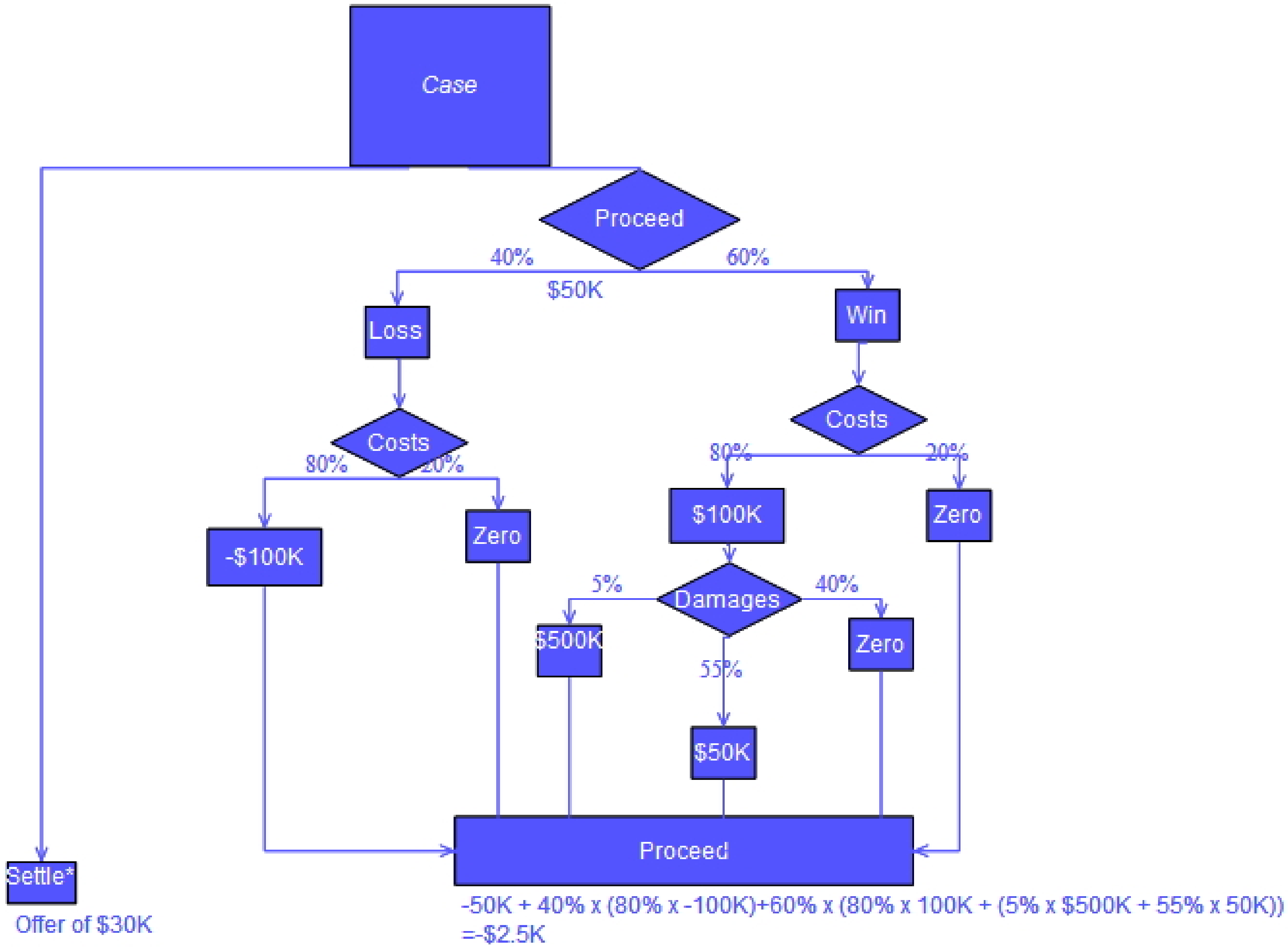
Pros:

- Simple to understand and to interpret ("white box", not "black box")
- Trees can be visualised
- Requires little data preparation
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree
- Can handle both numerical and categorical data
- Can be used for multi-output problems

Cons:

- The result can be over-complex tree that do not generalise the data well (overfitting)
- Can be unstable (small variations in the data might result in a completely different tree).
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts
- Not applicable for some problems(XOR, parity or multiplexer)
- Not applicable for unbalanced data (biased trees if some classes dominate)

Decision tree



Decision tree

PYTHON: sklearn.tree.DecisionTreeClassifier

For broad documentation with examples see:

https://scikit-learn.org/stable/modules/tree.html

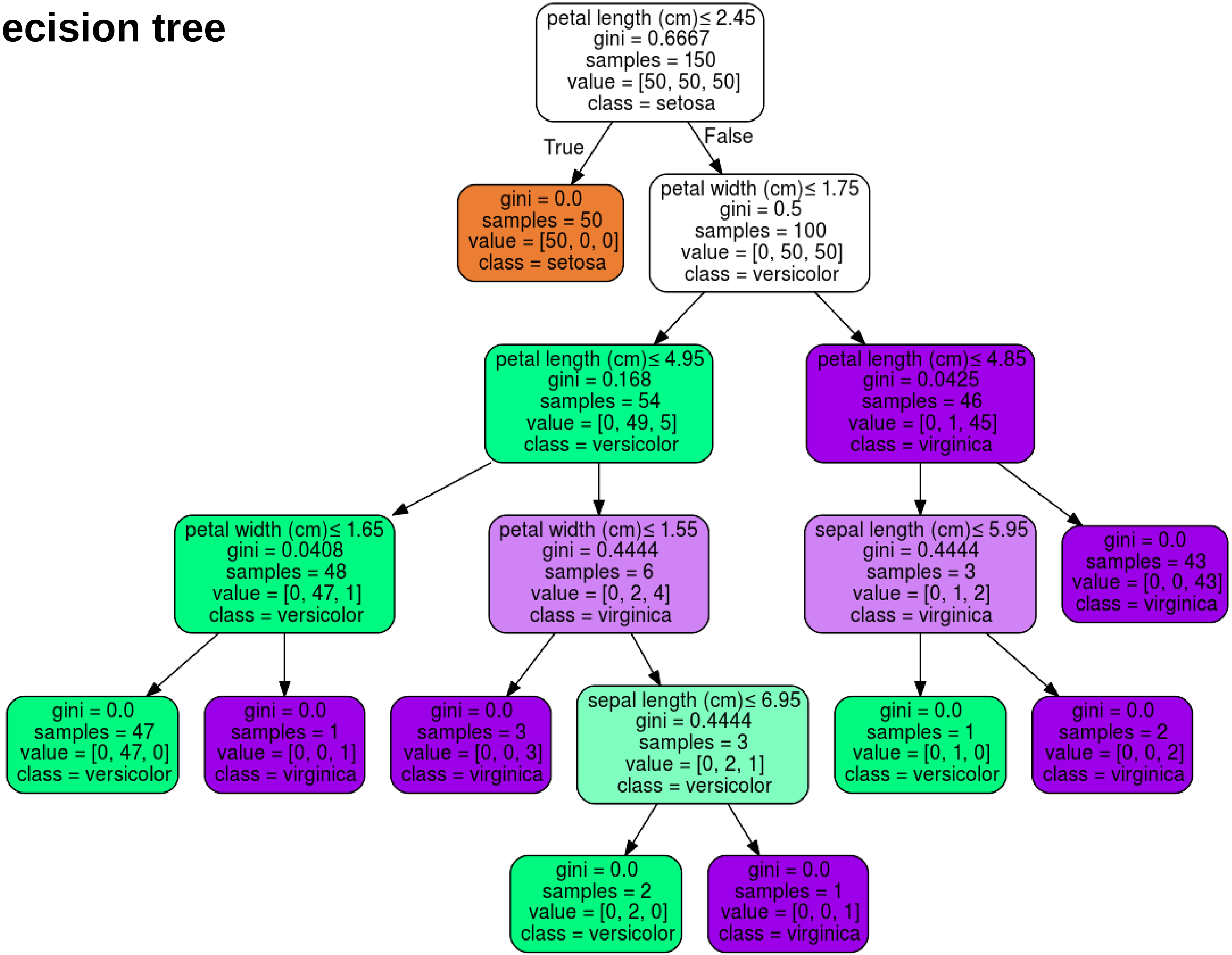
Using the Iris dataset, we can construct a tree as follows:

```
from sklearn.datasets import load_iris
from sklearn import tree
X, y = load_iris(return_X_y=True)
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, y)
```

Once trained, you can plot the tree with the plot_tree function:

```
tree.plot_tree(clf.fit(iris.data, iris.target))
```

Decision tree

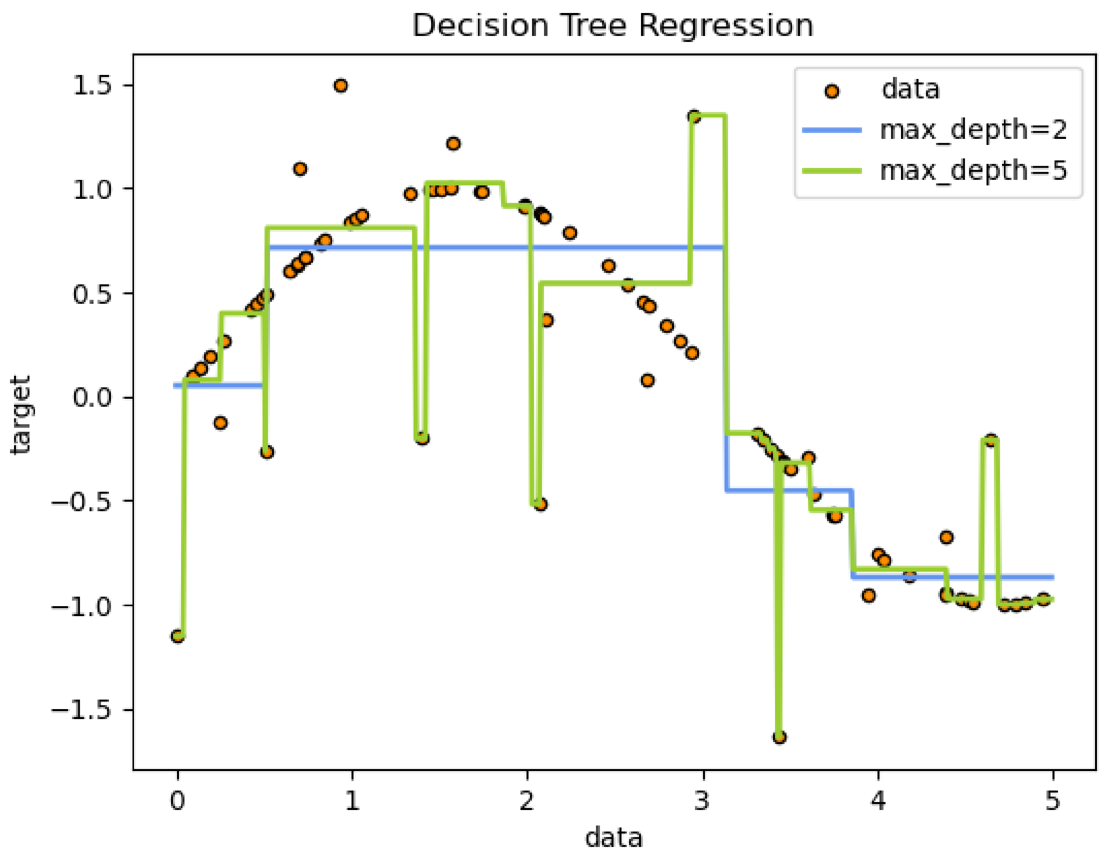


Iris decision tree

Fit regression model

Here we fit two models with different maximum depths

```
from sklearn.tree import DecisionTreeRegressor

regr_1 = DecisionTreeRegressor(max_depth=2)
regr_2 = DecisionTreeRegressor(max_depth=5)
regr_1.fit(X, y)
regr_2.fit(X, y)
```



Decision tree learning

The crucial step in DT is the learning. There are many algorithms:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (extension of ID3)
- CART (Classification and regression trees)
- CHAID (Chi-square automatic interaction detection)
- MARS (multivariate adaptive regression splines)

For details see:

https://en.wikipedia.org/wiki/Decision_tree_learning

https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart

Decision tree learning

The crucial step in DT is the learning. There are many algorithms:

- ID3 (Iterative Dichotomiser 3)
- C4.5 (extension of ID3)
- CART (Classification and regression trees)
- CHAID (Chi-square automatic interaction detection)
- MARS (multivariate adaptive regression splines)

For details see:

https://en.wikipedia.org/wiki/Decision_tree_learning

https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart

Random forest

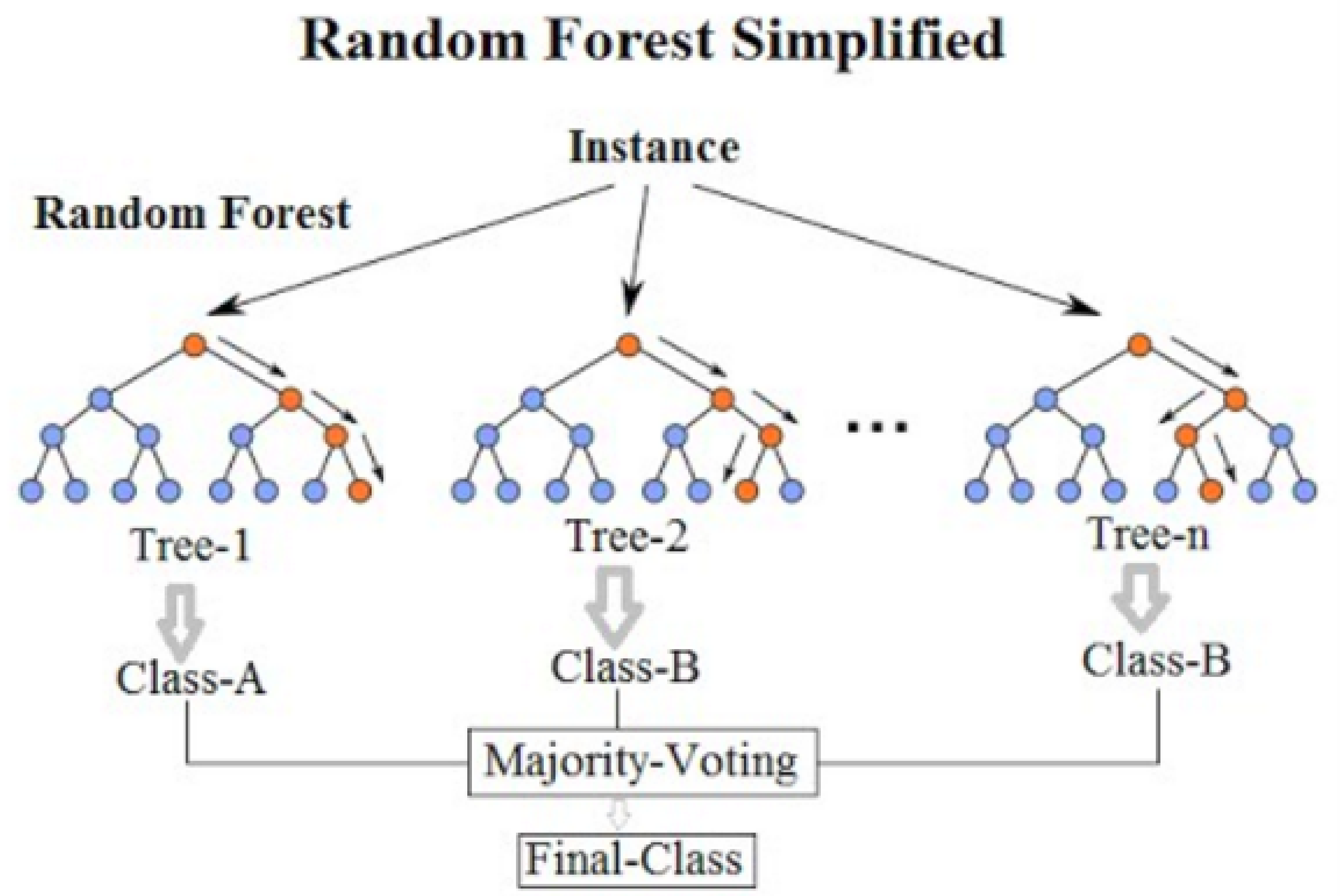
Random forest is an ensemble learning method constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

Random decision forests correct for decision trees' habit of overfitting to their training set

PYTHON: sklearn.ensemble.RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
```

```
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(X, y)

print(clf.feature_importances_)

print(clf.predict([[0, 0, 0, 0]]))
```



https://upload.wikimedia.org/wikipedia/commons/7/76/Random_forest_diagram_complete.png

Nearest Neighbors

Also called k-nearest neighbors algorithm (k-NN) - a non-parametric method used for classification and regression

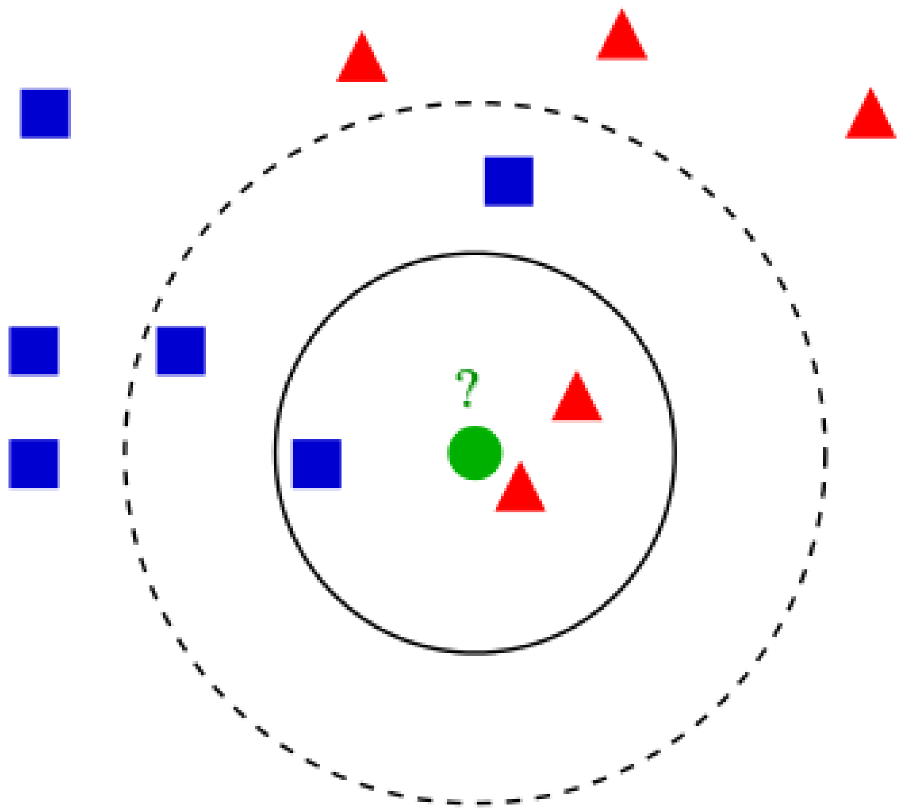
The input consists of the k closest training examples in the feature space

The output:

In k-NN classification is a class membership (an object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors; k is a positive integer, typically small)

In k-NN regression is the property value for the object (the average of the values of k nearest neighbors)

Nearest Neighbors



Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

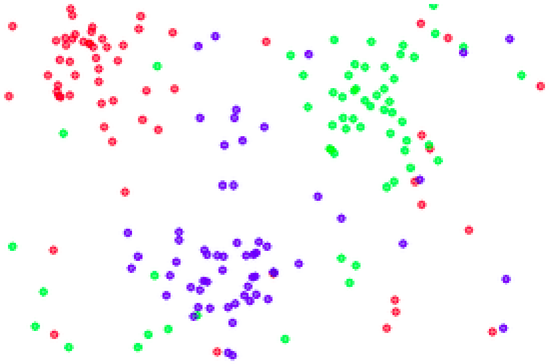
https://upload.wikimedia.org/wikipedia/commons/e/e7/KnnClassification.svg

Nearest Neighbors

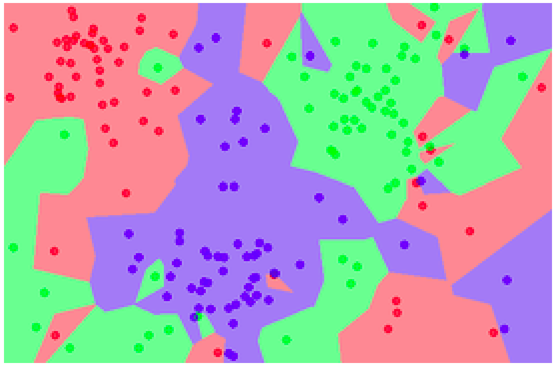
PYTHON: sklearn.neighbors

For broad documentation with examples see:

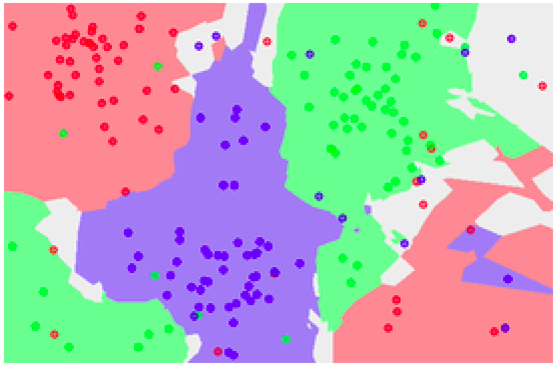
https://scikit-learn.org/stable/modules/neighbors.html



Dataset



1NN



5NN

Support Vector Machines

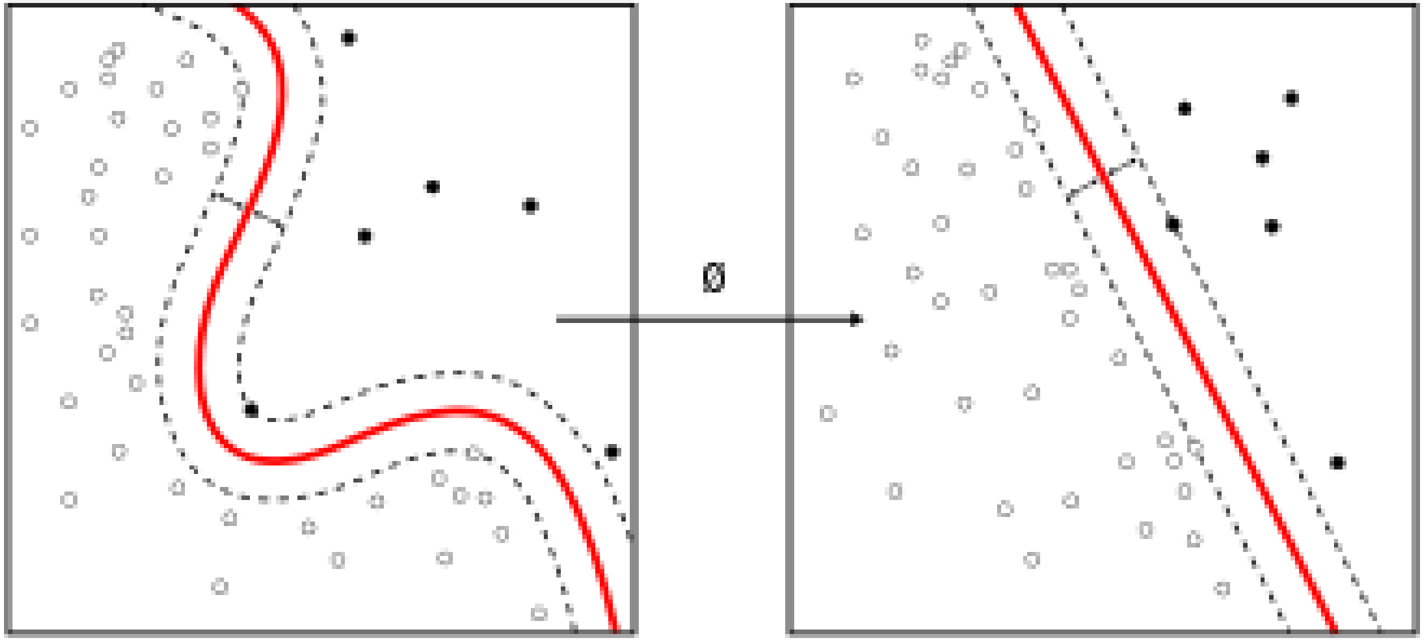
Pros:

- effective in high dimensional spaces
- still effective in cases where number of dimensions is greater than the number of samples.
- uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- versatile: different Kernel functions can be specified for the decision function

Cons:

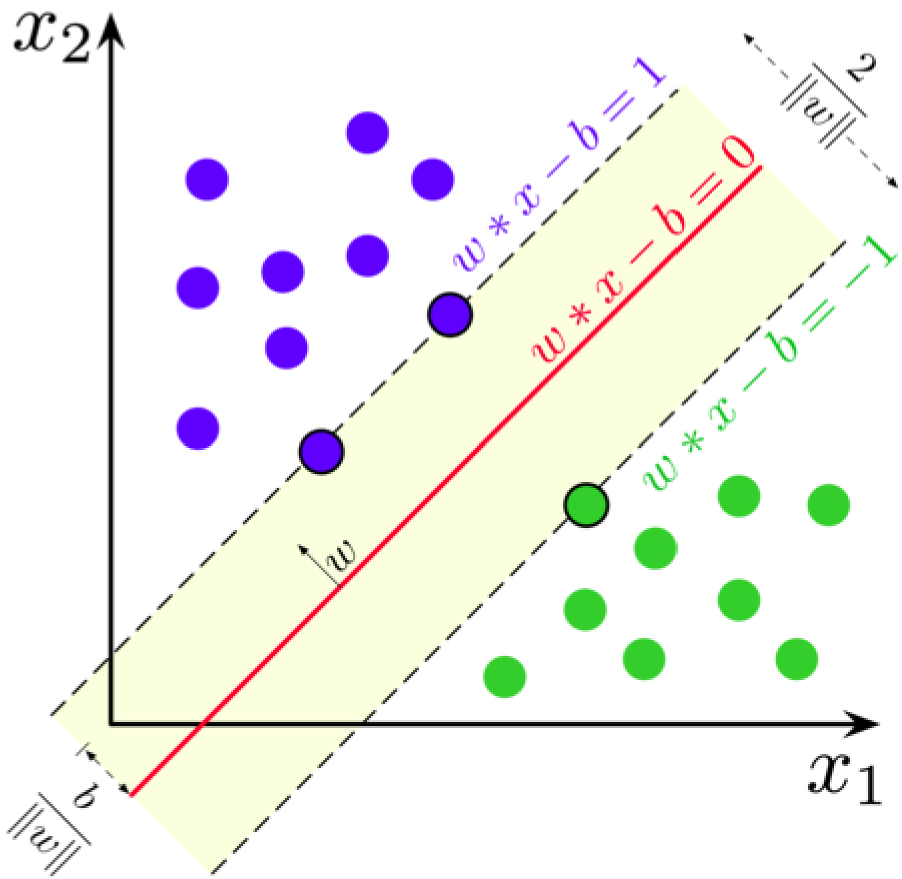
- if the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

Support Vector Machines



It uses so called kernels

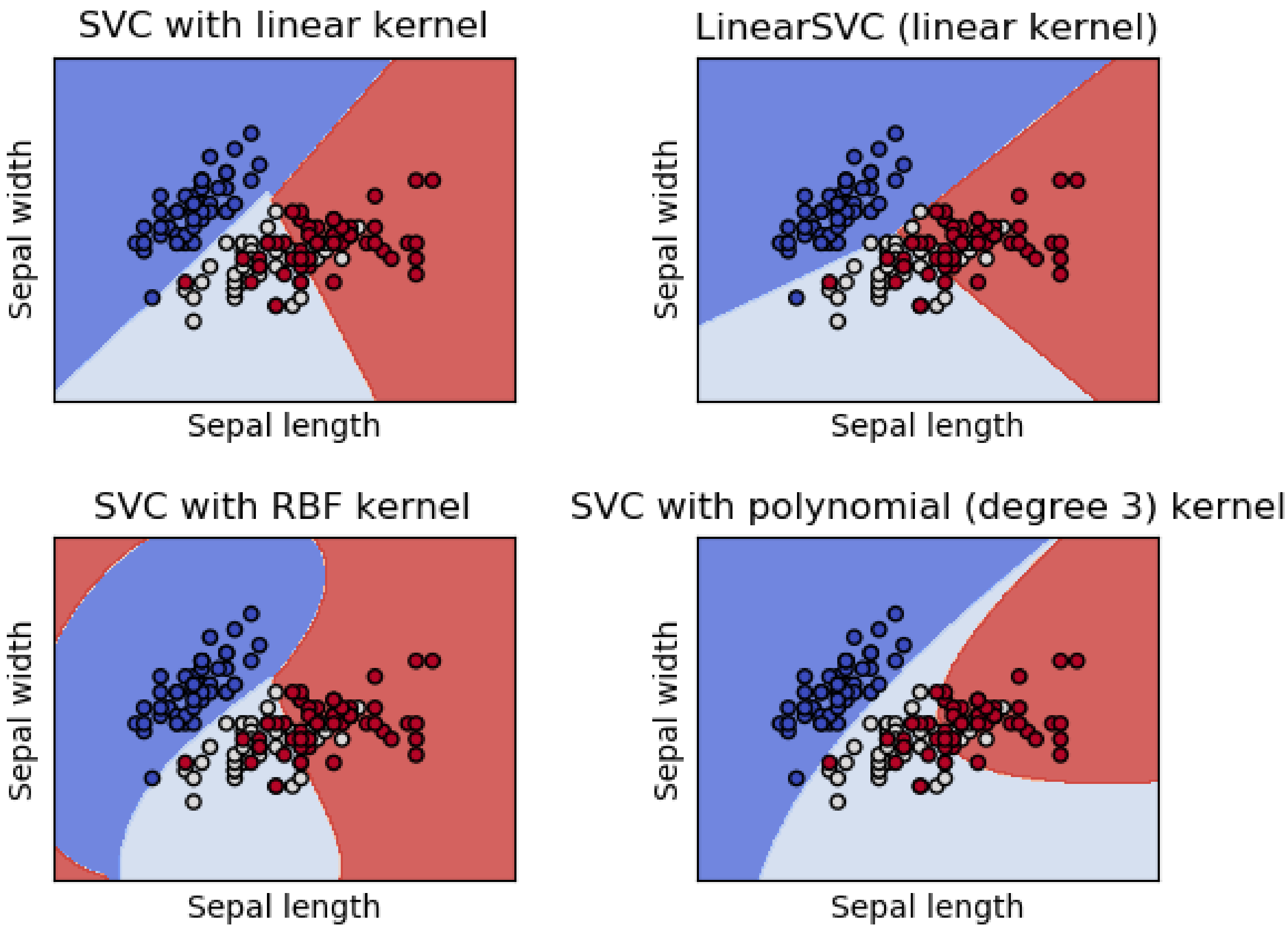
Support Vector Machines



Or more formaly: maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Kernel functions: linear, polynomial, rbf, sigmoid

Support Vector Machines



Iris example

Support Vector Machines

PYTHON: sklearn.svm

For broad documentation with examples see:

https://scikit-learn.org/stable/modules/svm.html

Kernel functions: linear, polynomial, rbf, sigmoid

If you need to remember one thing about SVMs:

always use rbf (radical basis function) kernel with GridSearchCV optimization of gamma and C

Support Vector Regression (SVR) using linear and non-linear kernels

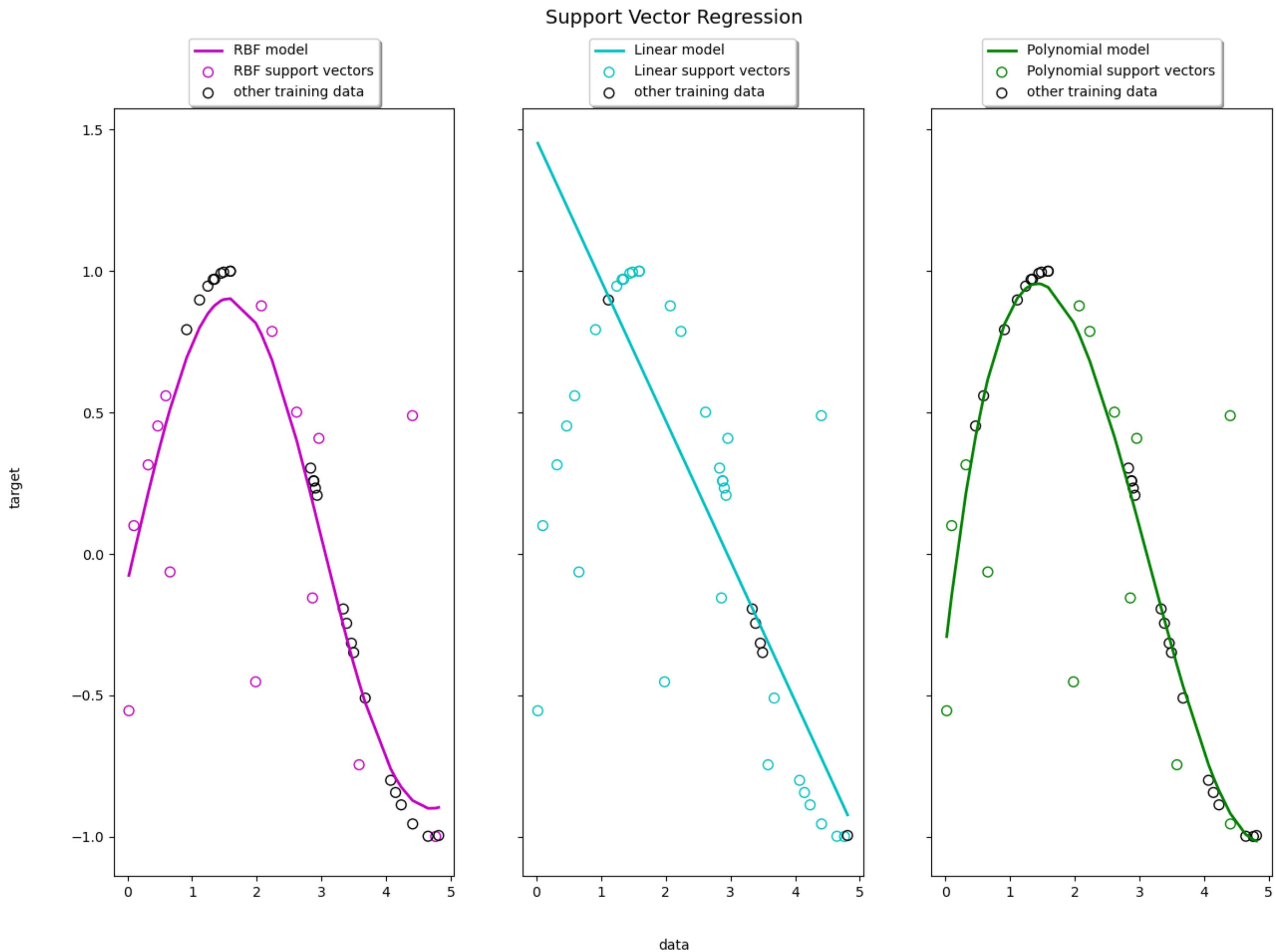
Toy example of 1D regression using linear, polynomial and RBF kernels.

```
# Authors: The scikit-learn developers
# SPDX-License-Identifier: BSD-3-Clause

import matplotlib.pyplot as plt
import numpy as np

from sklearn.svm import SVR
```

Fit regression model

```
svr_rbf = SVR(kernel="rbf", C=100, gamma=0.1, epsilon=0.1)
svr_lin = SVR(kernel="linear", C=100, gamma="auto")
svr_poly = SVR(kernel="poly", C=100, gamma="auto", degree=3, epsilon=0.1, coef0=
```

Support Vector Regression (SVR)



Support Vector Machines

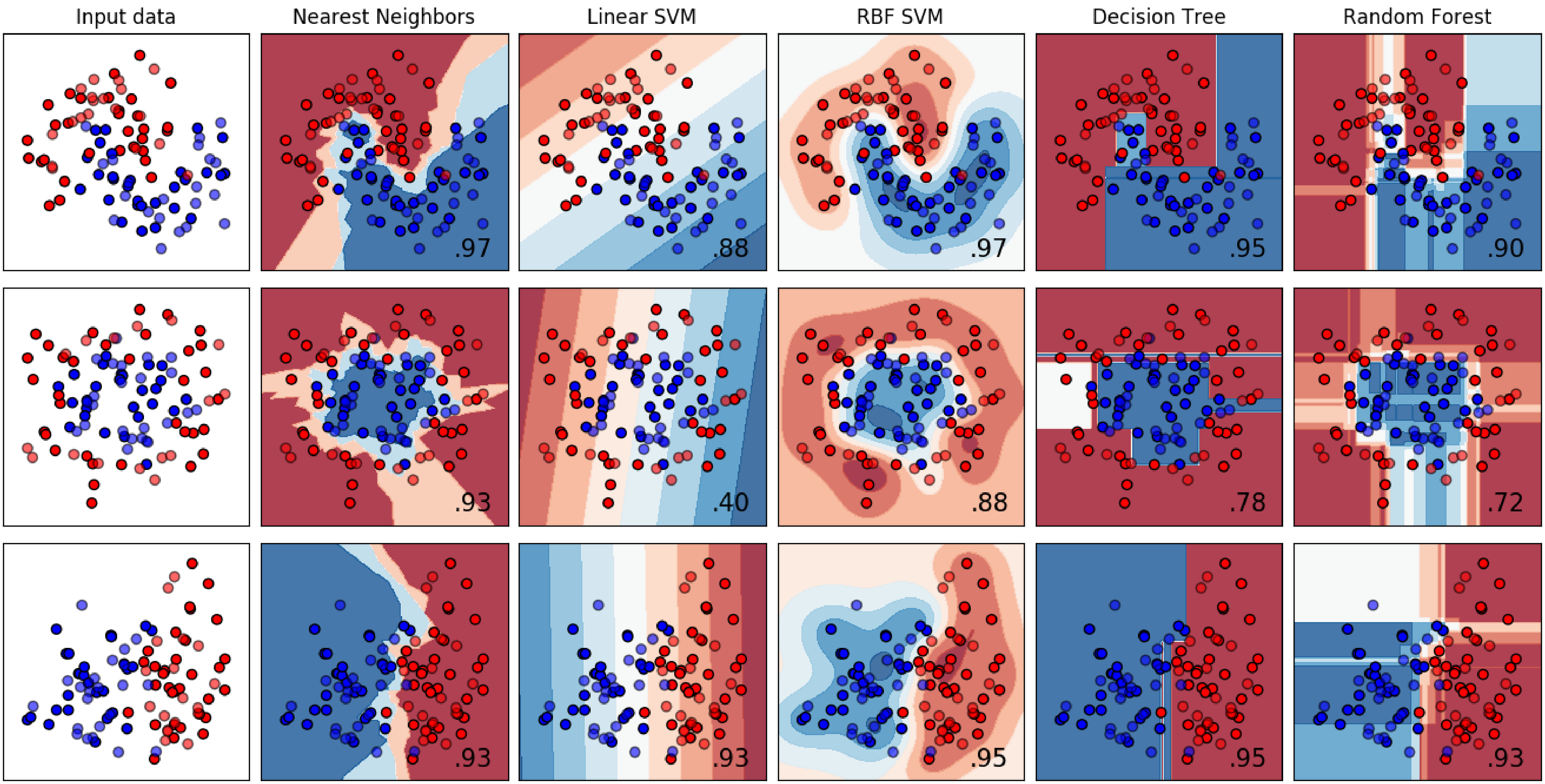
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
```

If you need to remember one thing about SVMs:

always use rbf (radical basis function) kernel with GridSearchCV optimization of gamma and C

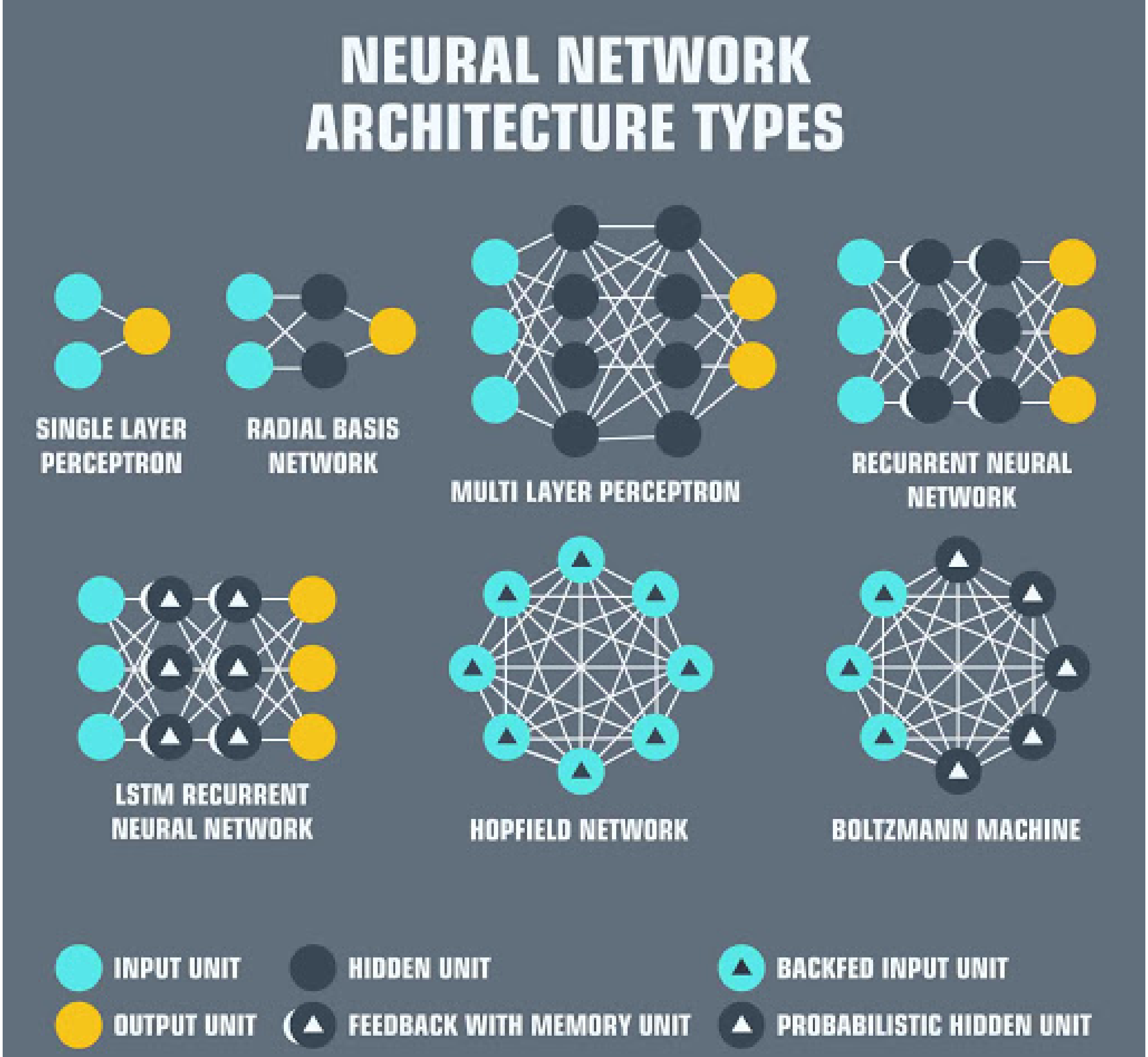
It has been shown that rbf with grid optimization can substitute all other kernels

Comparing algorithms

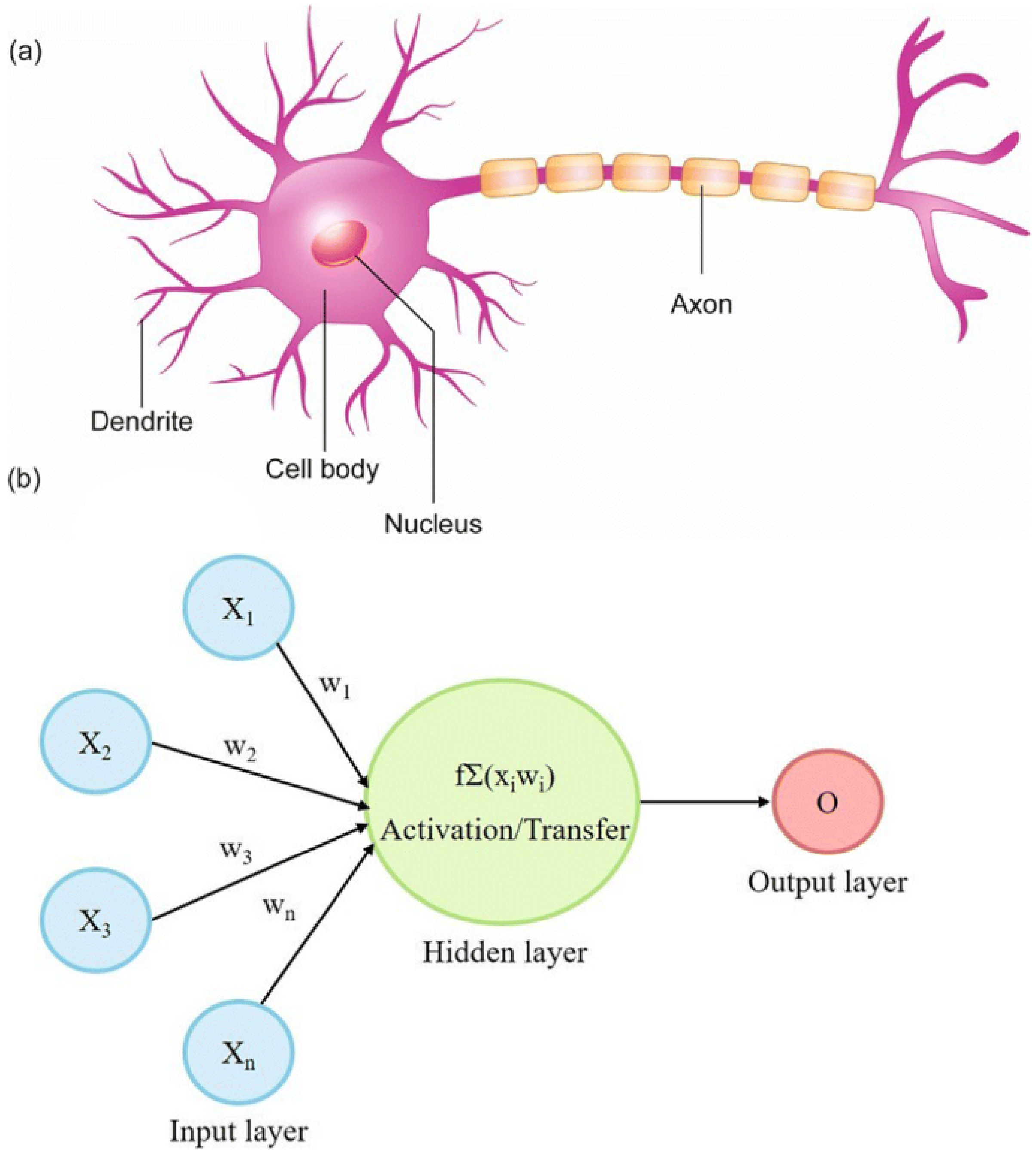


https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

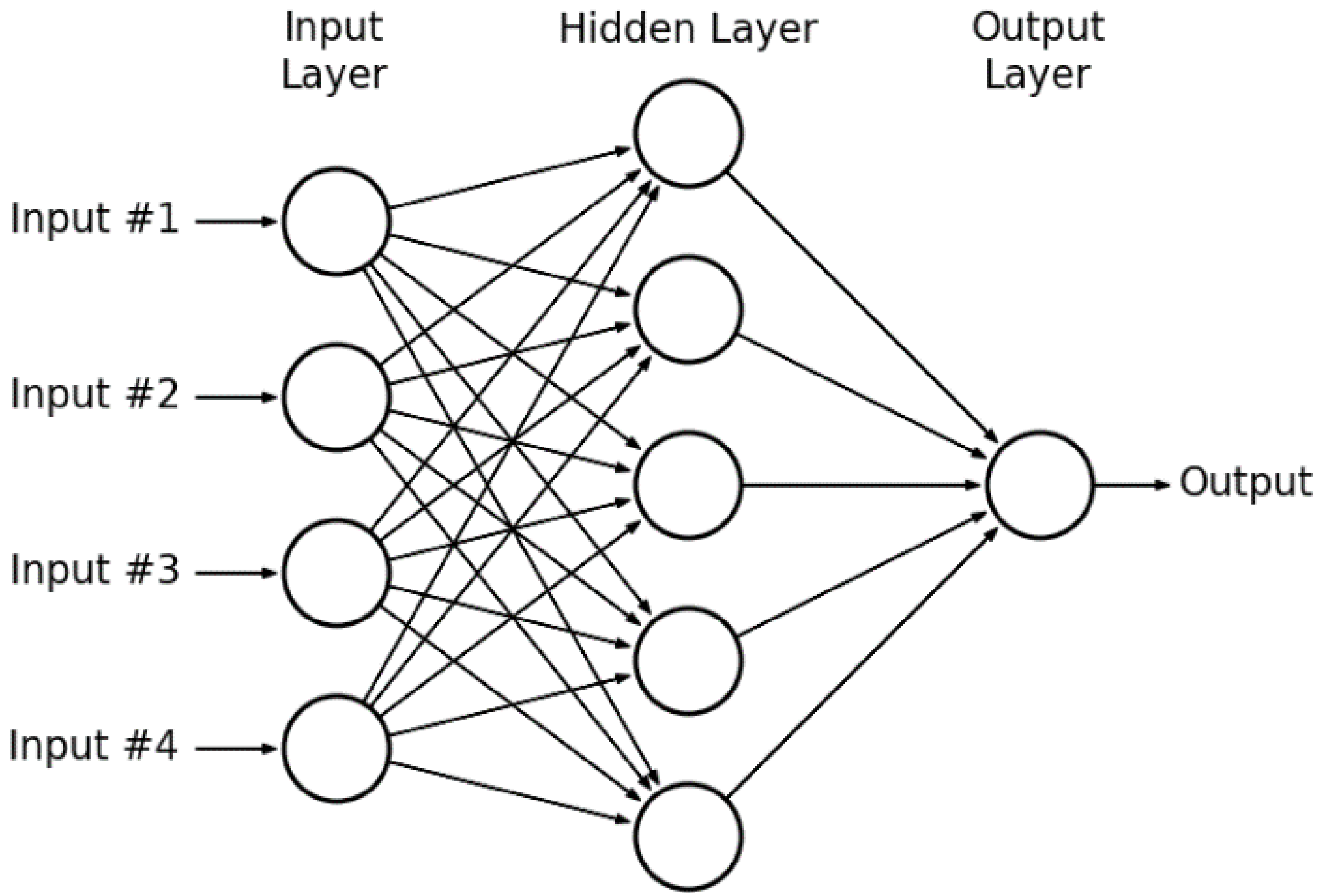
Deep learning


NEURAL NETWORK
ARCHITECTURE TYPES
SINGLE LAYER
PERCEPTRON
RADIAL BASIS
NETWORK
MULTI LAYER PERCEPTRON
RECURRENT NEURAL
NETWORK
LSTM RECURRENT
NEURAL NETWORK
HOPFIELD NETWORK
BOLTZMANN MACHINE
INPUT UNIT
HIDDEN UNIT
BACKFED INPUT UNIT
OUTPUT UNIT
FEEDBACK WITH MEMORY UNIT
PROBABILISTIC HIDDEN UNIT

Neural networks (deep learning)

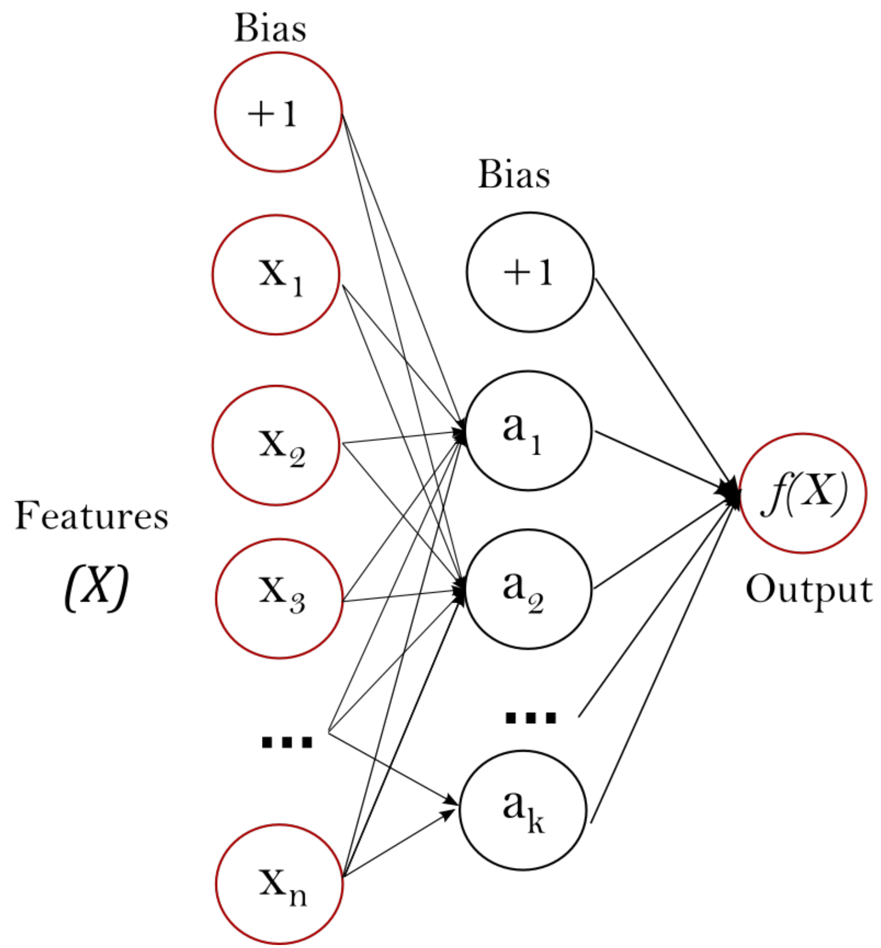


Neural networks (deep learning)



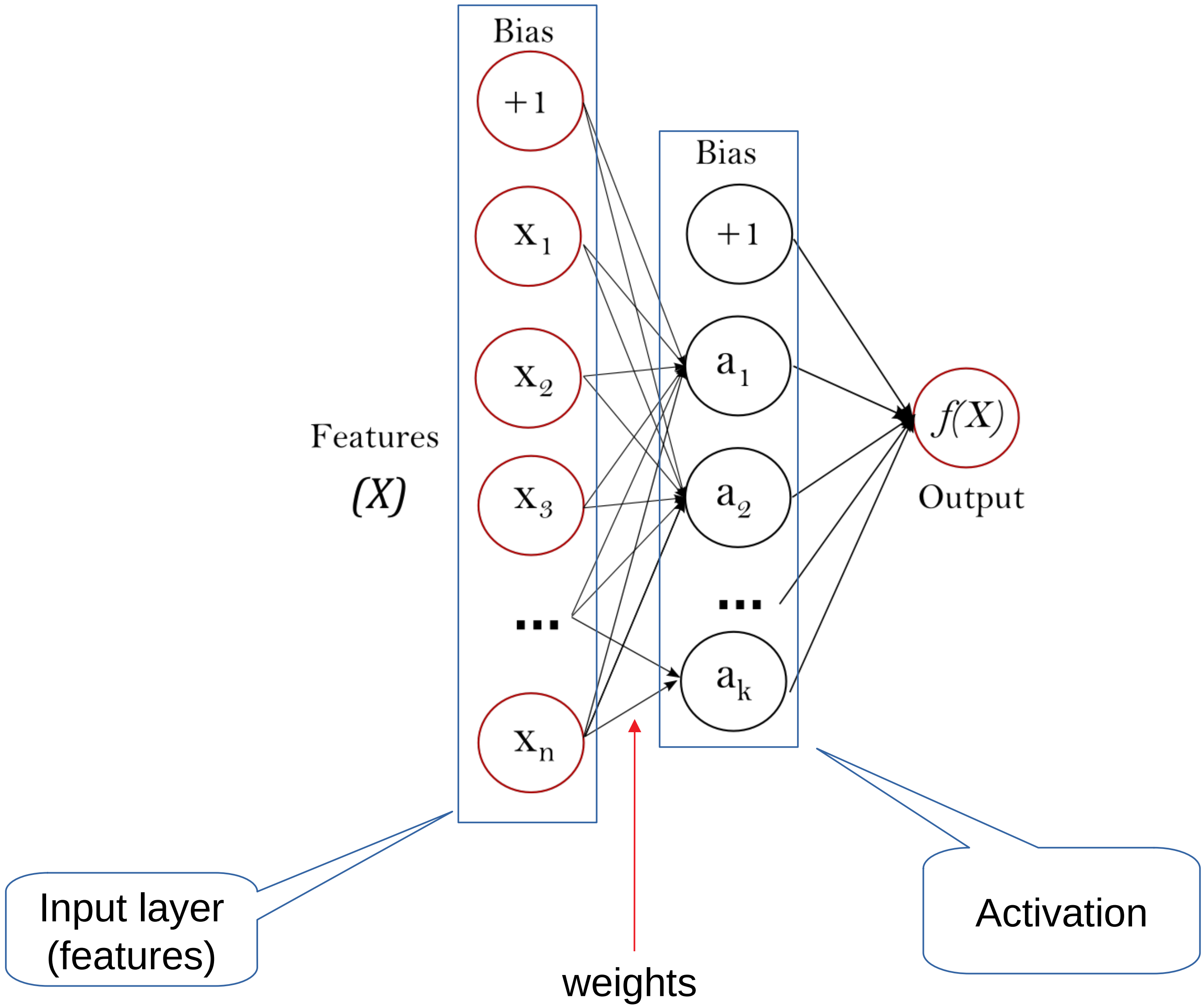
Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, \ldots, x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.

Neural networks (deep learning)



Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(\cdot): R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, \ldots, x_m$ and a target y, it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.

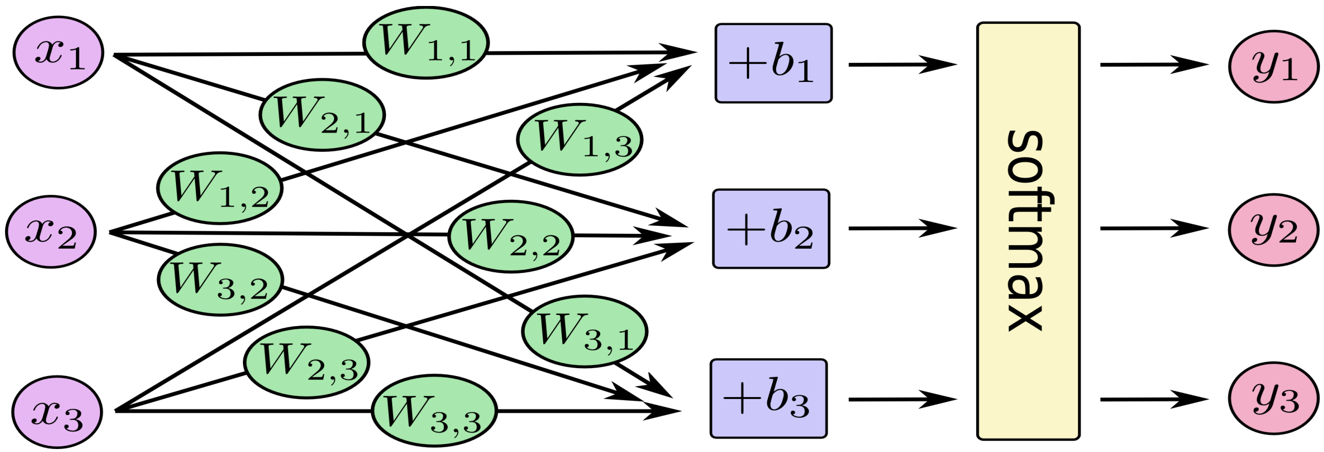
Neural networks (deep learning)



$w_1x_1 + w_2x_2 + \ldots + w_mx_m,$ AND $g(\cdot) : R \rightarrow R$

Deep Learning - basics

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$



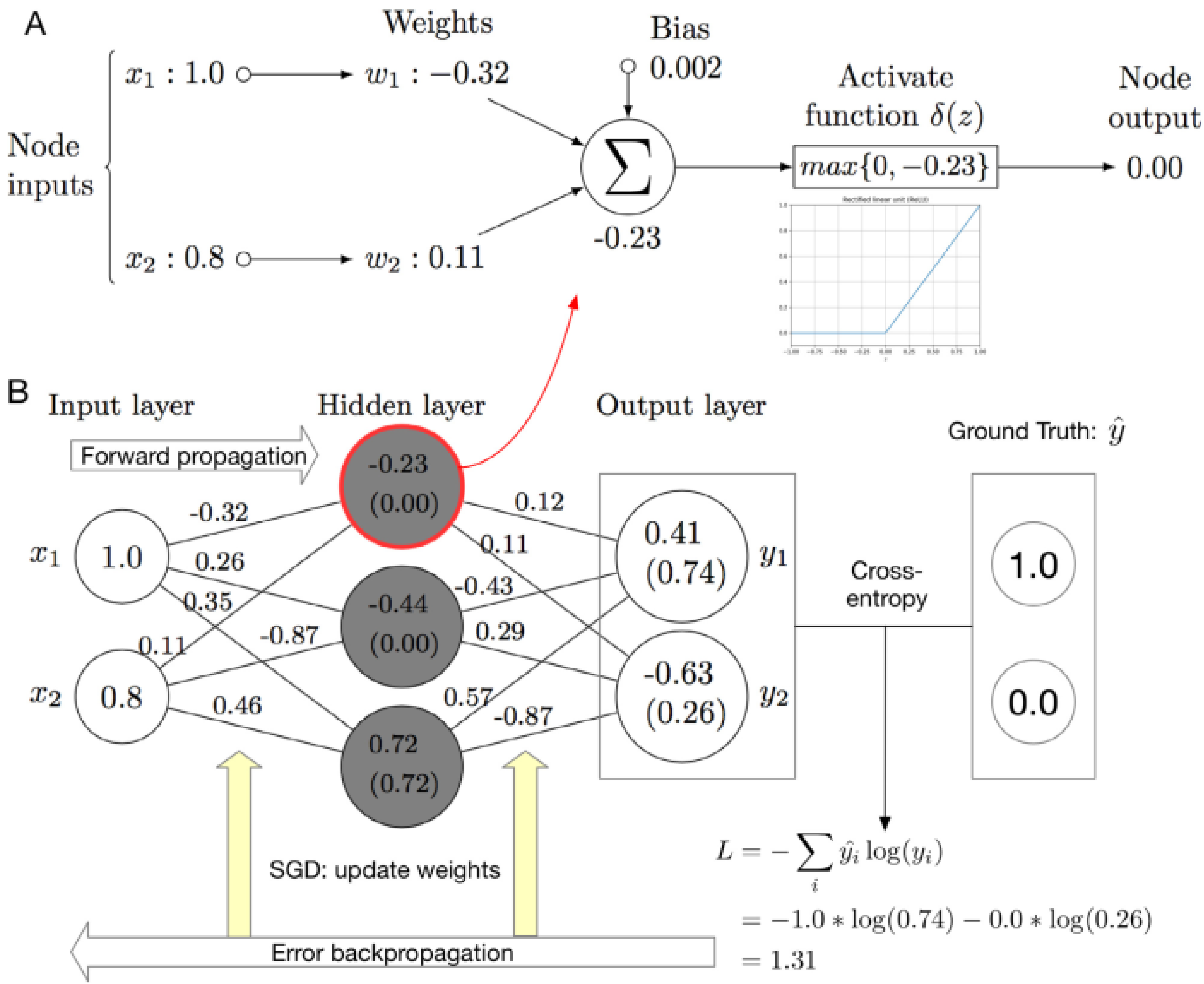
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$

y = softmax(Wx+b)

https://www.tensorflow.org/tutorials/

Deep Learning - basics



10.1016/j.ymeth.2019.04.008

Neural networks (deep learning)

sklearn.neural_network.MLPClassifier

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
```

solver{‘lbfgs’, ‘sgd’, ‘adam’}

The solver for weight optimization.

- ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
- ‘sgd’ refers to stochastic gradient descent.
- ‘adam’ refers to a stochastic gradient-based optimizer

Neural networks (deep learning)

sklearn.neural_network.MLPClassifier

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
```

solver{‘lbfgs’, ‘sgd’, ‘adam’}

For big datasets Adam
For small ones lbfgs

The solver for weight optimization.

- ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
- ‘sgd’ refers to stochastic gradient descent.
- ‘adam’ refers to a stochastic gradient-based optimizer

Neural networks (deep learning)

sklearn.neural_network.MLPClassifier

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
```

activation{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default=’relu’

Activation function for the hidden layer.

- ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x
- ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
- ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)

Neural networks (deep learning)

sklearn.neural_network.MLPClassifier

```
from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
```

activation{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default=’relu’

Equently worth to check other than ReLu

Activation function for the hidden layer.

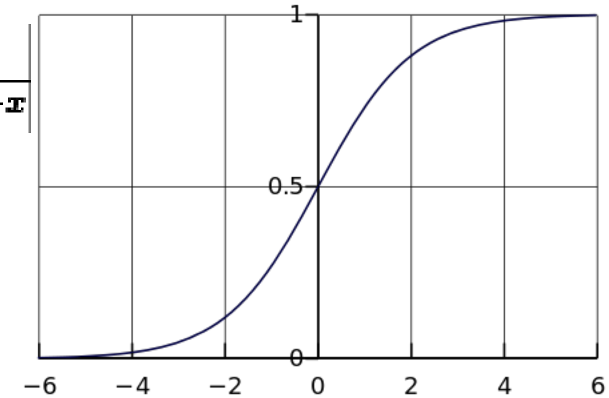
- ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x
- ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
- ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)

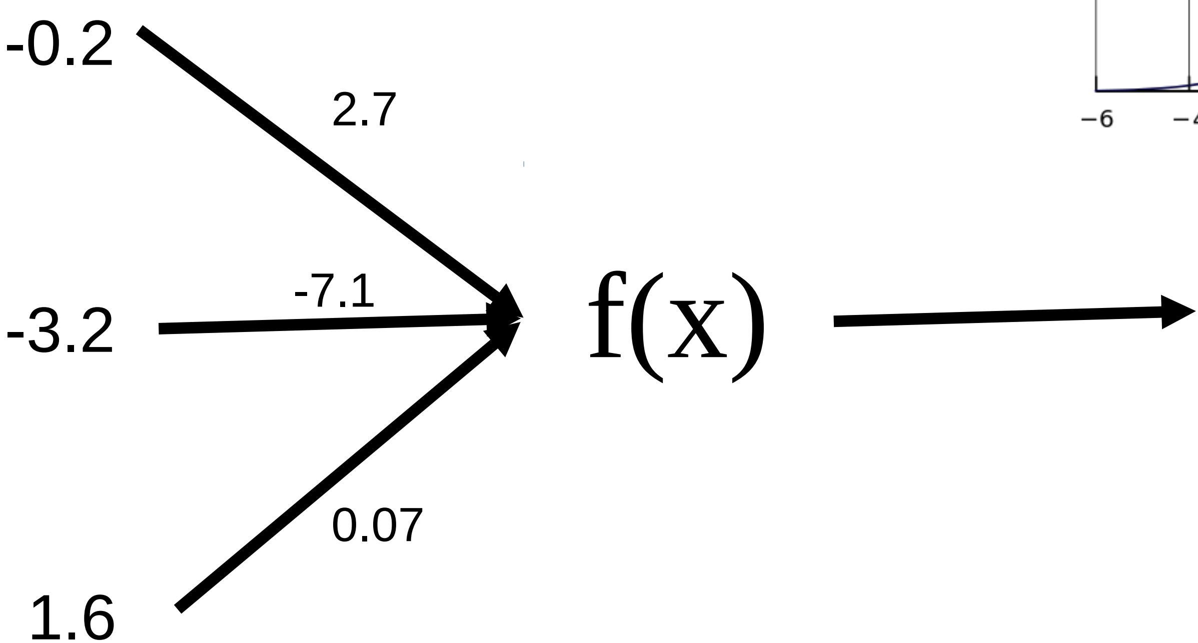
Nonlinear activation functions

Activation functions in neural networks are mathematical equations that determine the output of a neuron, effectively deciding whether it should be activated or not. They introduce non-linearity into the network, enabling it to learn complex patterns in data. Without activation functions, a neural network would behave like a simple linear model, regardless of its depth

Nonlinear activation functions

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





$X = -0.2\times2.7 + 3.2\times7.1 + 1.6\times0.007 = 22.19$

Nonlinear activation functions

activation function of a node defines the output of that node given an input or set of inputs

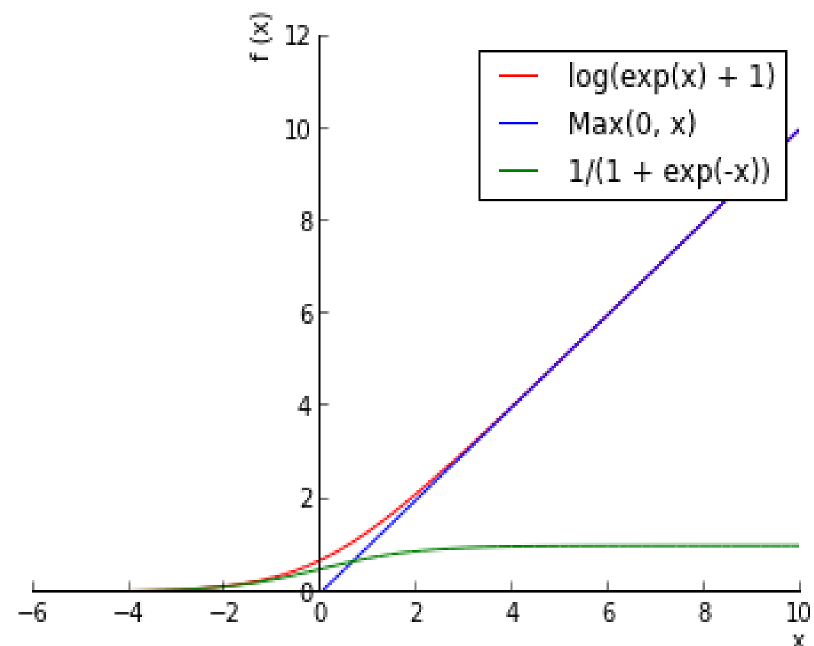
Sigmoid unit :
$f(x) = \frac{1}{1+exp(-x)}$

Tanh unit:
$f(x) = tanh(x)$

Rectified linear unit (ReLU):
$f(x) = \sum_{i=1}^{\inf} \sigma(x - i + 0.5) \approx log(1 + e^{x})$

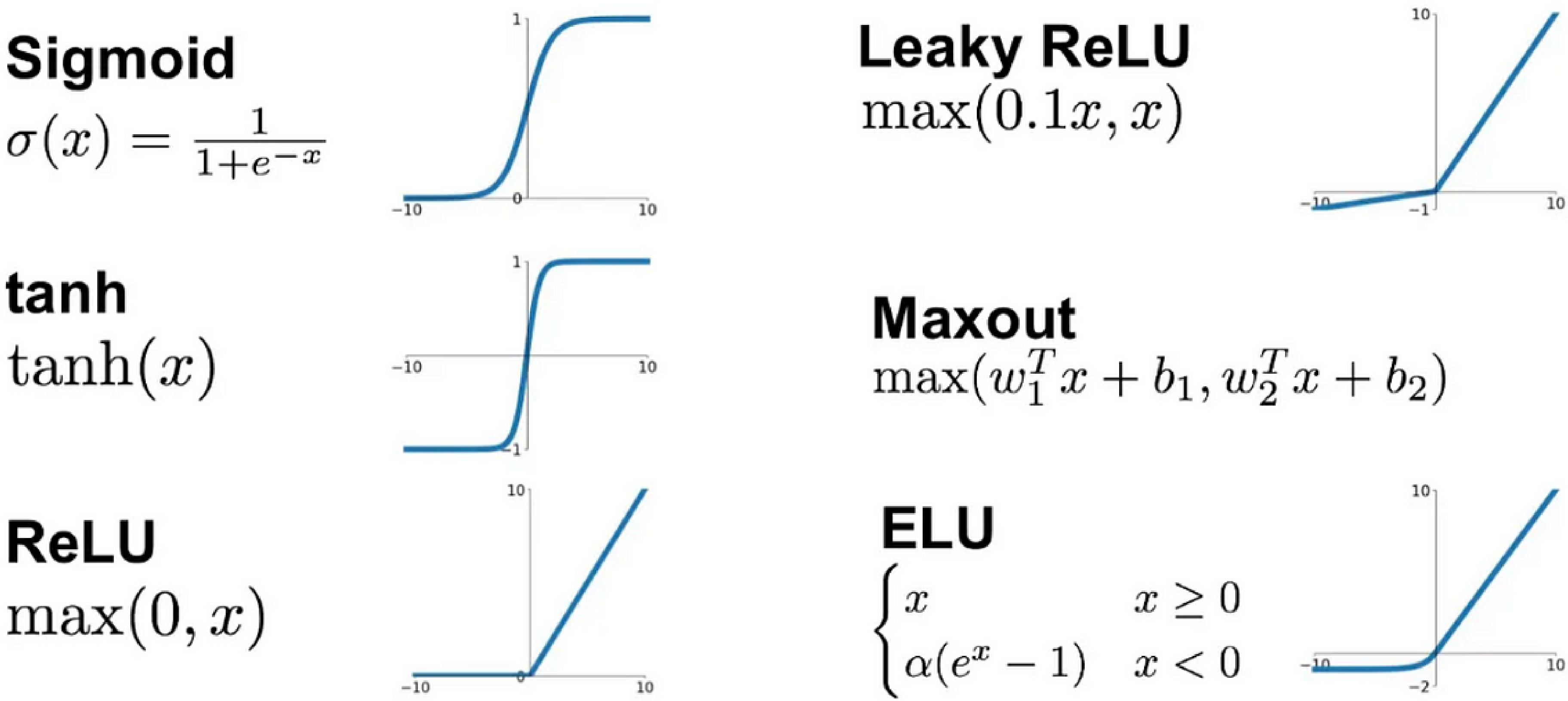
we refer

- $\sum_{i=1}^{\inf} \sigma(x - i + 0.5)$ as **stepped sigmoid**
- $log(1 + e^{x})$ as **softplus function**

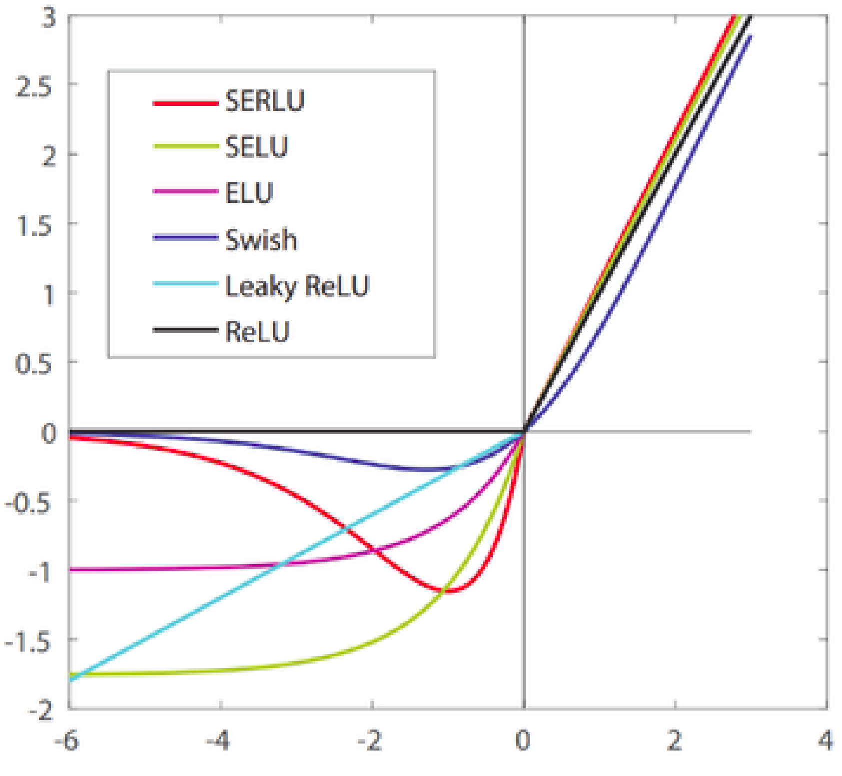


Nonlinear activation functions

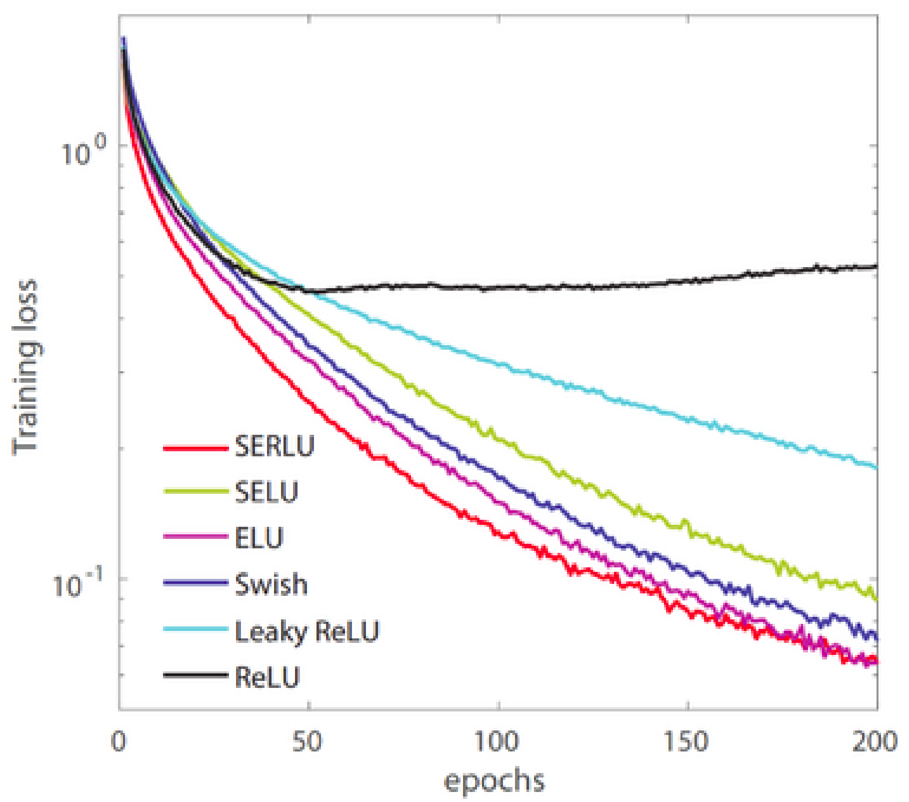
activation function of a node defines the output of that node given an input or set of inputs



Neural networks (deep learning)



(a): Different activation functions



(b): Performance on CIFAR10 without dropout

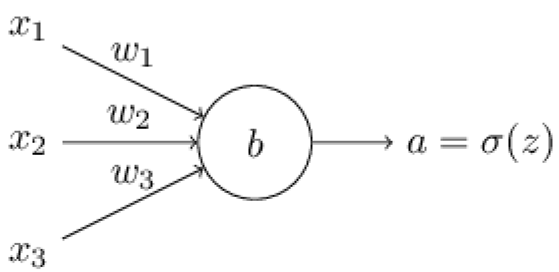
activation{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default=’relu’

Equently worth to check other than ReLu

Activation function for the hidden layer.

- ‘identity’, no-op activation, useful to implement linear bottleneck, returns f(x) = x
- ‘logistic’, the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- ‘tanh’, the hyperbolic tan function, returns f(x) = tanh(x).
- ‘relu’, the rectified linear unit function, returns f(x) = max(0, x)

Deep Learning – loss function: cross-entropy



$$z = \sum_j w_j x_j + b$$

the weighted sum of the inputs

$$C = -\frac{1}{n} \sum_x \left[y \ln a + (1 - y) \ln(1 - a) \right]$$

where **n** is the total number of items of training data,
the sum is **over all** training inputs **x,** and **y** is the corresponding desired output.

Some features: non-negative and non-symetric

Neural networks (deep learning)

+

Front-end **Back-end**

Neural networks (deep learning)

For more complicated architectures

Neural networks (deep learning)

TensorFlow

PyTorch

scikit learn

theano

plaidML

Popular back-ends
(low or medium level programming libraries for neural networks)

Neural networks (deep learning)

ONNX

mxnet

Caffe2

Microsoft CNTK

GLUON

Keras

Chainer

Popular front-ends
(high level programming libraries for neural networks)

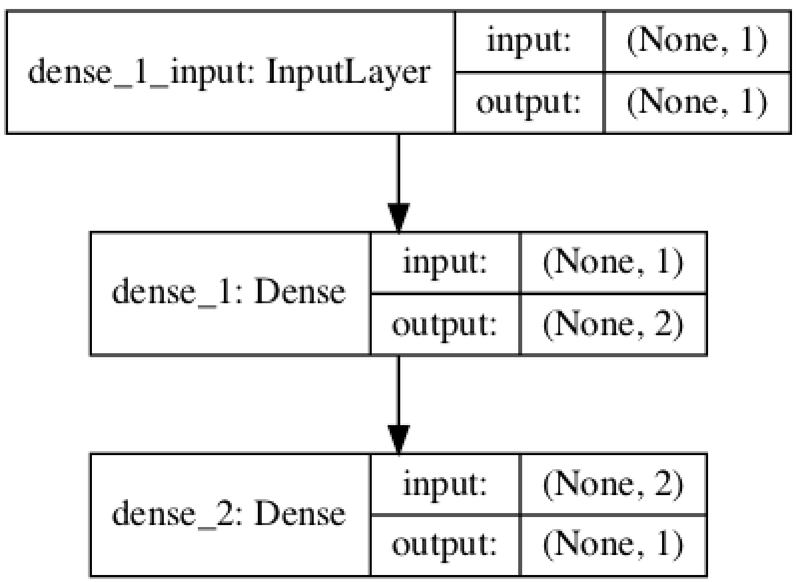
Neural networks (deep learning)

Recommended setting

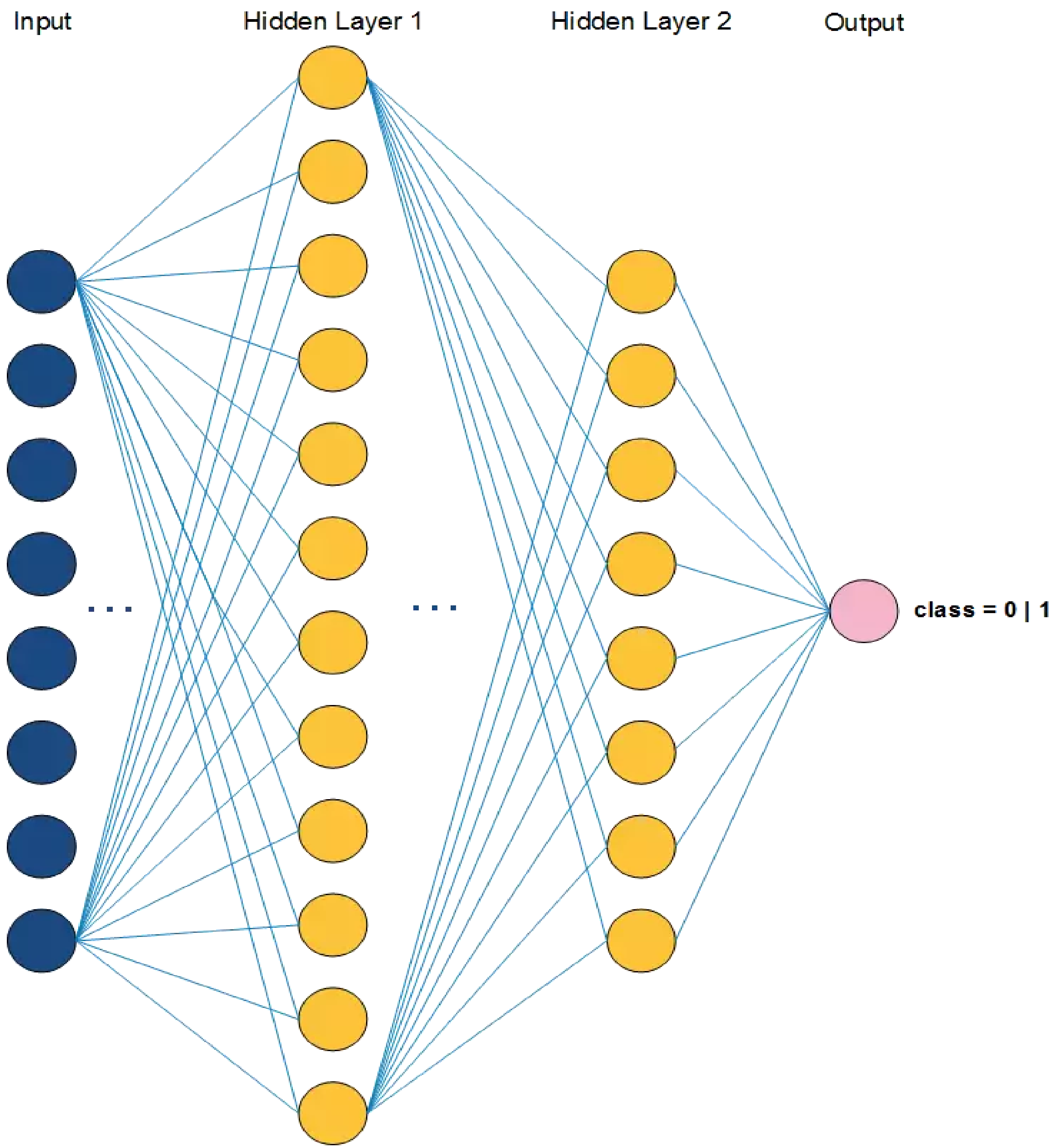
```python
from tensorflow import keras
from tensorflow.keras import layers

# Instantiate a trained vision model
vision_model = keras.applications.ResNet50()

# This is our video.encoding branch using the trained vision_model
video_input = keras.Input(shape=(100, None, None, 3))
encoded_frame_sequence = layers.TimeDistributed(vision_model)(video_input)
encoded_video = layers.LSTM(256)(encoded_frame_sequence)

# This is our text-processing branch for the question input
question_input = keras.Input(shape=(100,), dtype='int32')
embedded_question = layers.Embedding(10000, 256)(question_input)
encoded_question = layers.LSTM(256)(embedded_question)

# And this is our video question answering model:
merged = keras.layers.concatenate([encoded_video, encoded_question])
output = keras.layers.Dense(1000, activation='softmax')(merged)
video_qa_model = keras.Model(inputs=[video_input, question_input],
                             outputs=output)
```

Neural networks (deep learning)

```python
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model
model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True
```



Dense model - Multilayer Perceptron (MLP) in Keras



Dense model - Multilayer Perceptron (MLP) in Keras

```
model = Sequential()


model.add(Dense(DENSE_1ST_SIZE, input_shape=(ROW_LENGTH,), init='uniform',
            activation='softplus', W_constraint=maxnorm(3)))
model.add(Dropout(DROPOUT1))


model.add(Dense(DENSE_2ST_SIZE, init='uniform', activation='softsign'))
model.add(Dropout(DROPOUT2))


model.add(Dense(1, init='uniform', activation='sigmoid'))

print(model.summary())

optimizer = optimizers.adam(lr=1e-03, epsilon=1e-06)

model.compile(loss='binary_crossentropy',
        optimizer=OPTIMIZER,
        metrics=['accuracy'])
```

Dense model - Multilayer Perceptron (MLP) in Keras

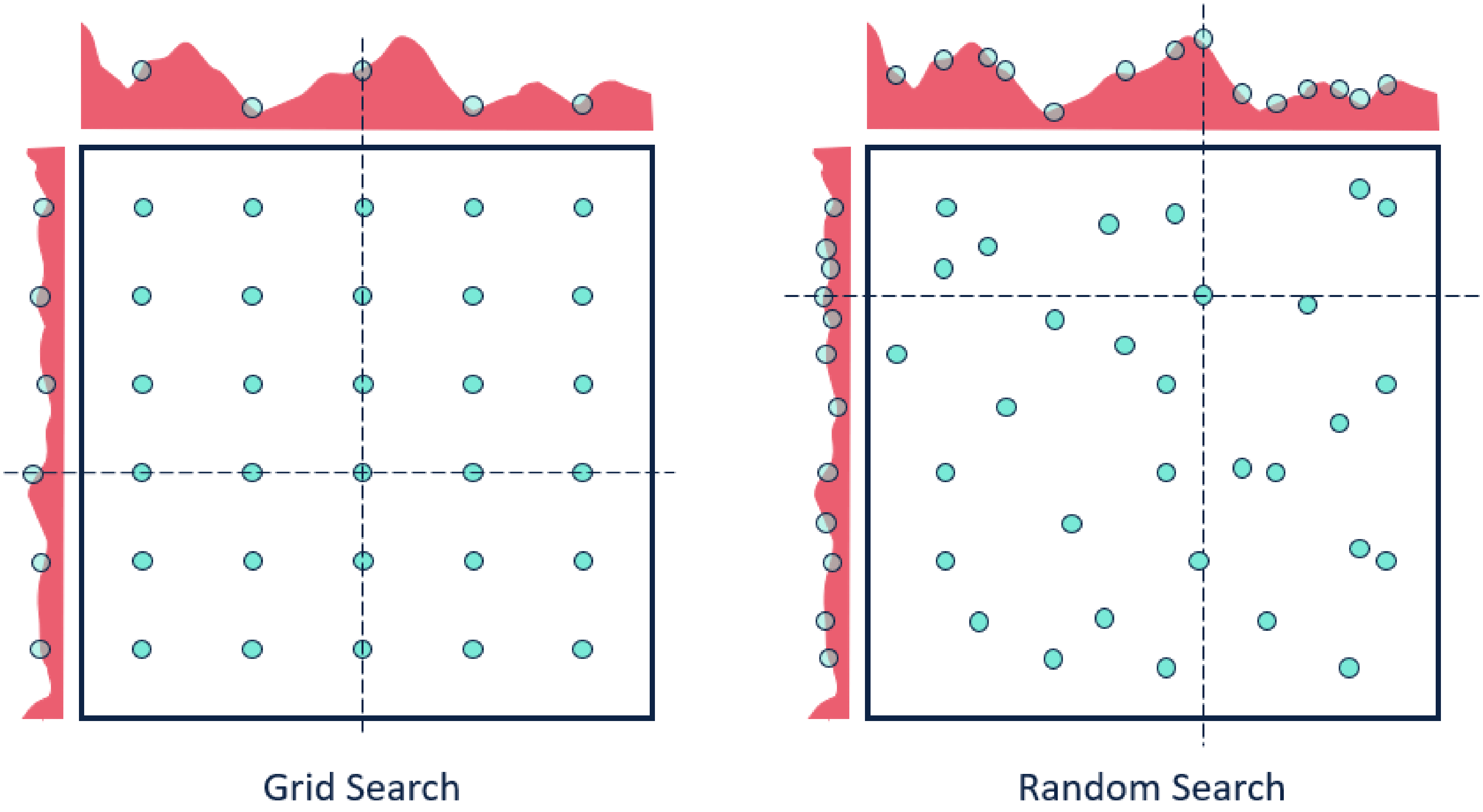
Parameters to tune

```
model = Sequential()

model.add(Dense(DENSE_1ST_SIZE, input_shape=(ROW_LENGTH,), init='uniform',
              activation='softplus', W_constraint=maxnorm(3)))
model.add(Dropout(DROPOUT1))

model.add(Dense(DENSE_2ST_SIZE, init='uniform', activation='softsign'))
model.add(Dropout(DROPOUT2))

model.add(Dense(1, init='uniform', activation='sigmoid'))

print(model.summary())

optimizer = optimizers.adam(lr=1e-03, epsilon=1e-06)

model.compile(loss='binary_crossentropy',
        optimizer=OPTIMIZER,
        metrics=['accuracy'])
```

Dense model - Multilayer Perceptron (MLP)

```
model = Sequential()

model.add(Dense(DENSE_1ST_SIZE)
model.add(Dropout(DROPOUT1))

model.add(Dense(DENSE_2ST_SIZE)
model.add(Dropout(DROPOUT2))

model.add(Dense(1, init='uniform', activation='sigmoid'))

print(model.summary())

model.compile(loss='binary_crossentropy')
```

Hyperparameters tuning



Hyperparameters tuning

Keras Tuner
Hyperas
Ray-Tune
Optuna
Hyperopt
mlmachine
Polyaxon
BayesianOptimization
Talos
SHERPA
Scikit-Optimize
GpyOpt
…

Deep Learning

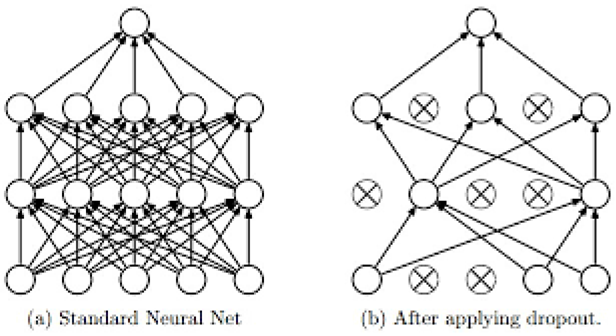
Dense model - Multilayer Perceptron (MLP)

Pros: extremely fast to train and easy to interpret

Cons:
- do not generalize well (no matter how many layers you use)
- prone to overfitting

Avoiding overfitting:

a) dropout (random killing of some neurons)



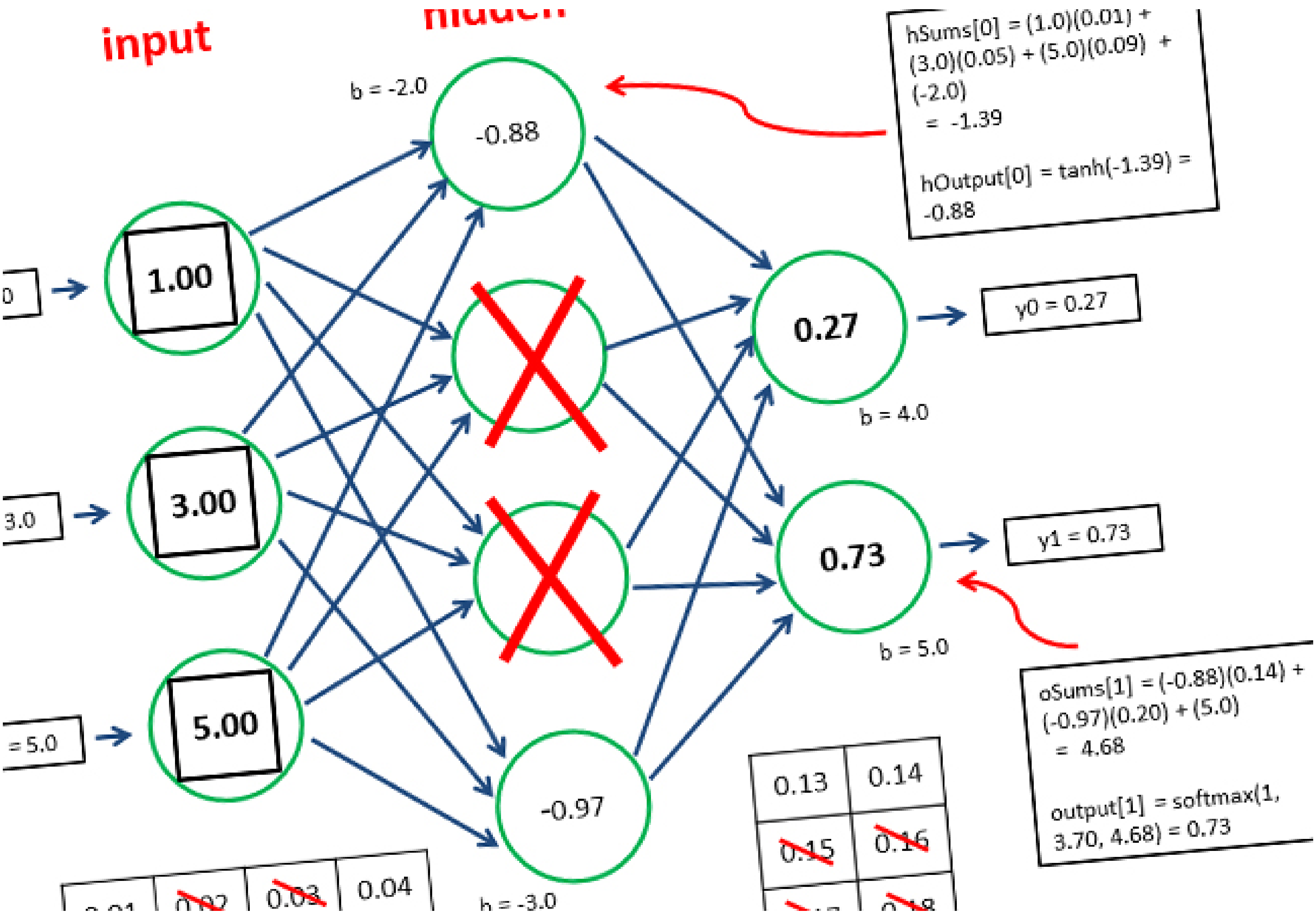
b) batching

c) early stopping

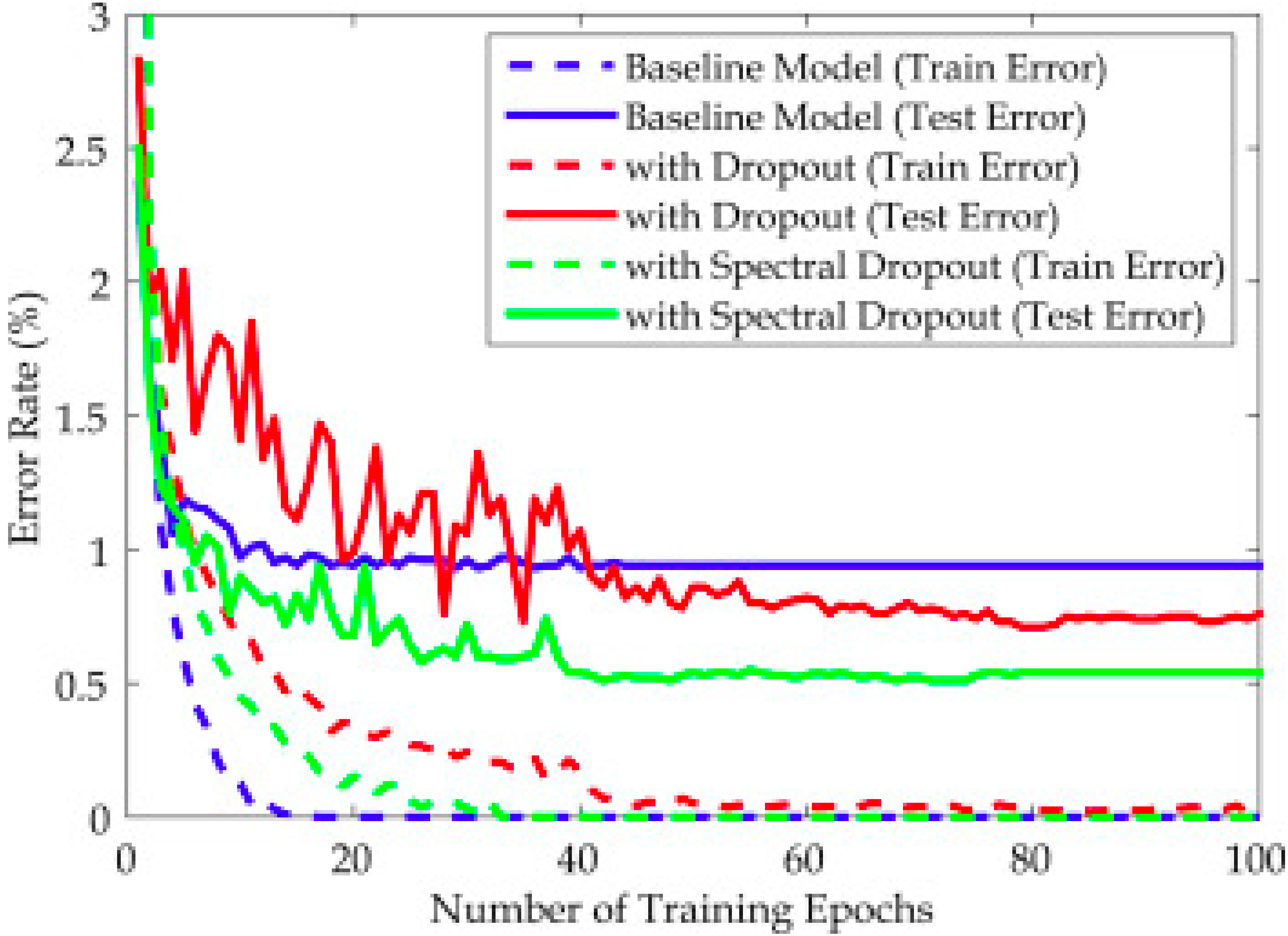
Deep Learning

DROPOUT

Dense model - Multilayer Perceptron (MLP)

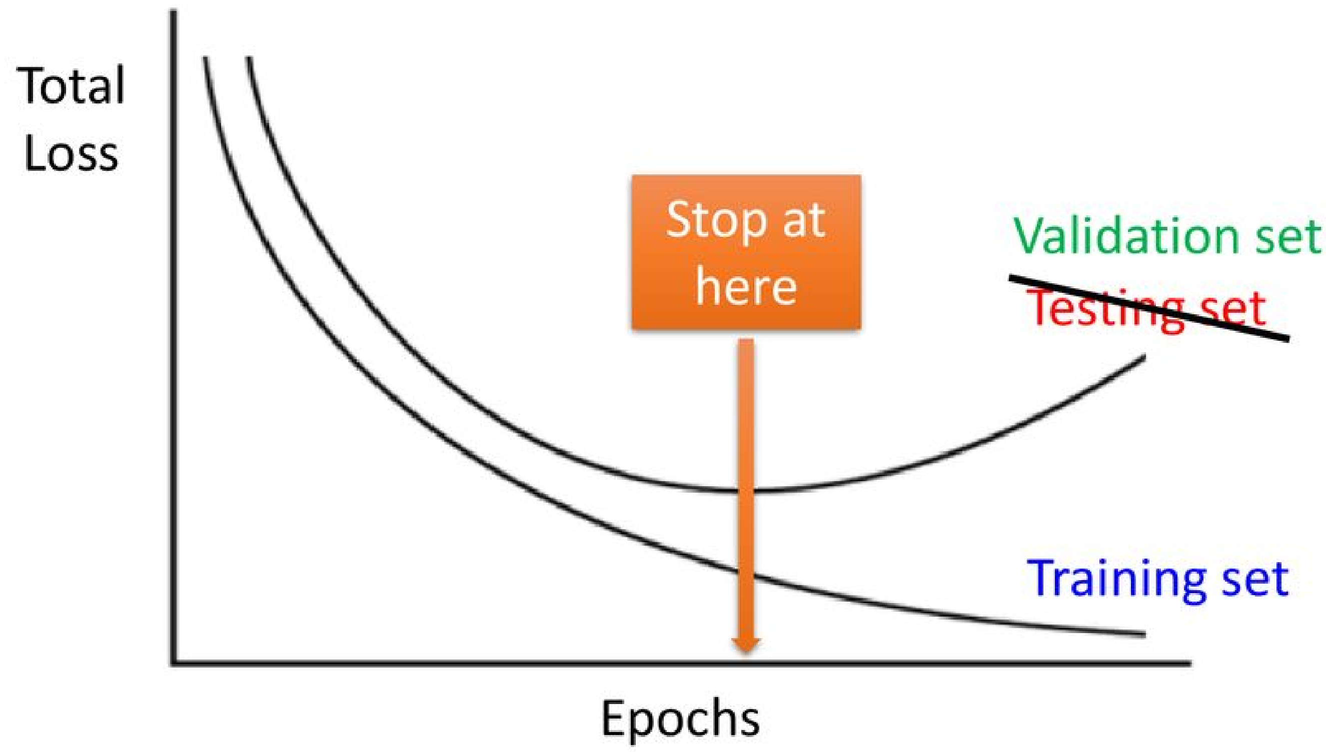


Dense model - Multilayer Perceptron (MLP)



Dense model - Multilayer Perceptron (MLP)

Early Stopping



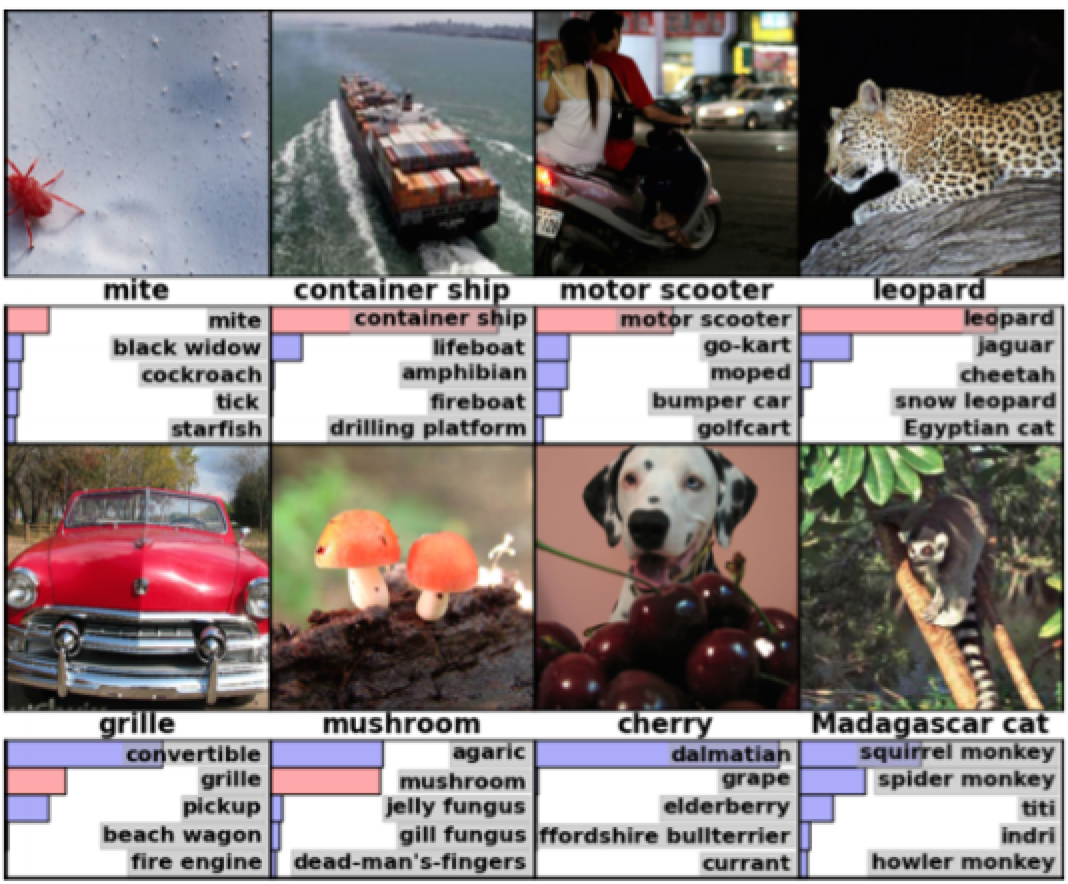
Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

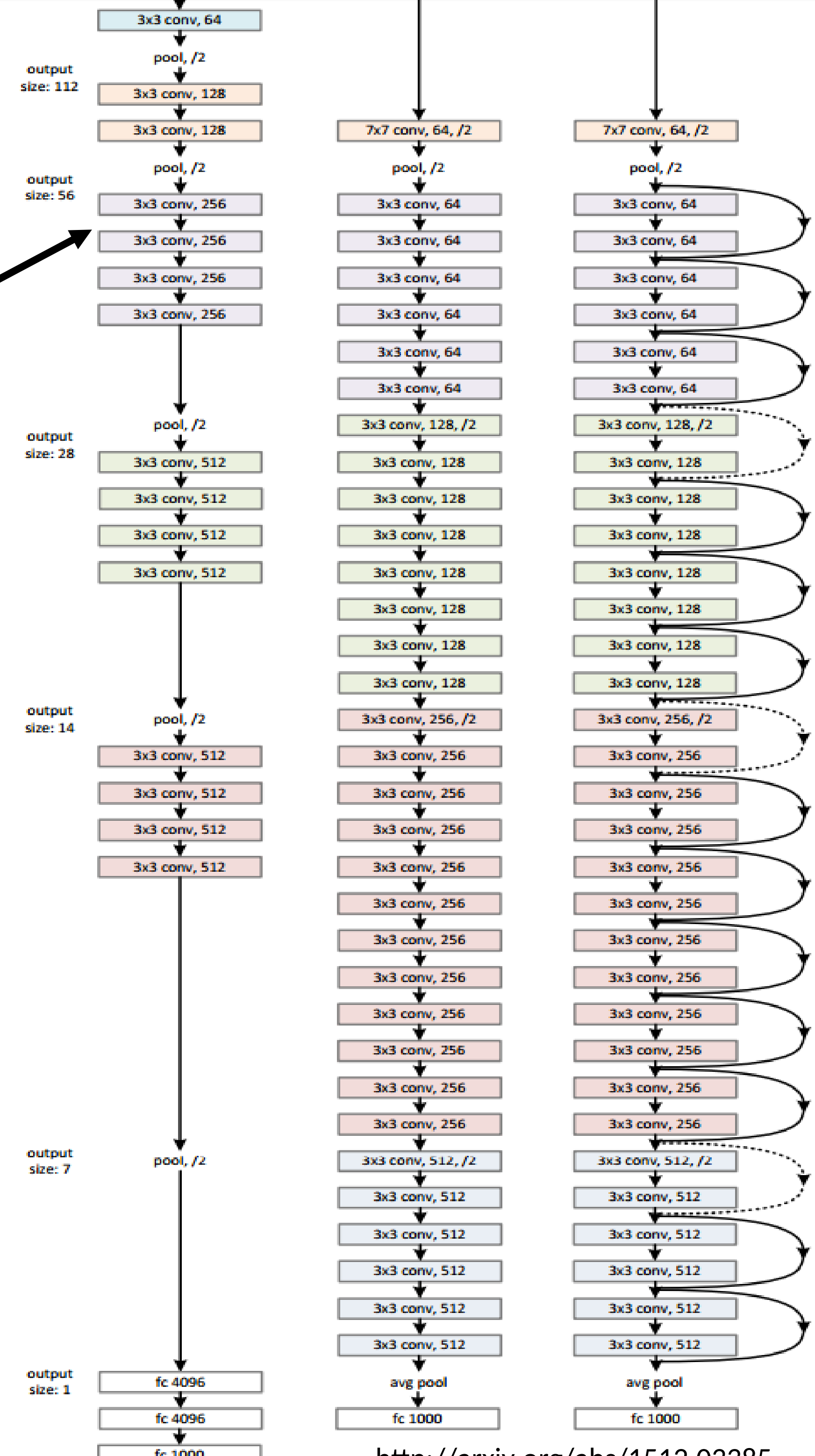
Deep Learning

More sophisticated model

VGG-19 model
19 layers
138M parameters
19,6 billion FLOPs

To

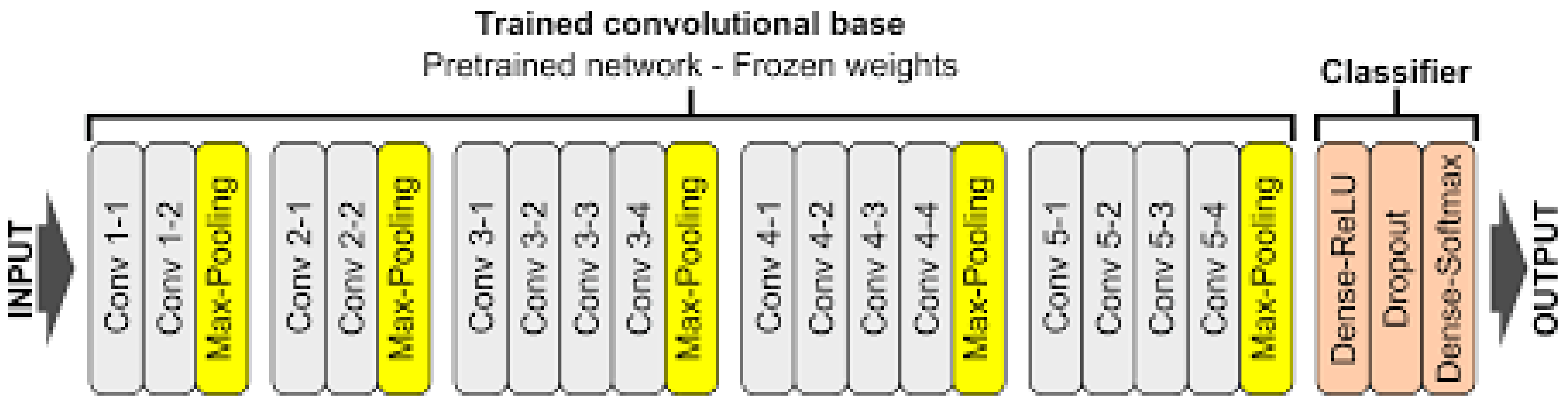




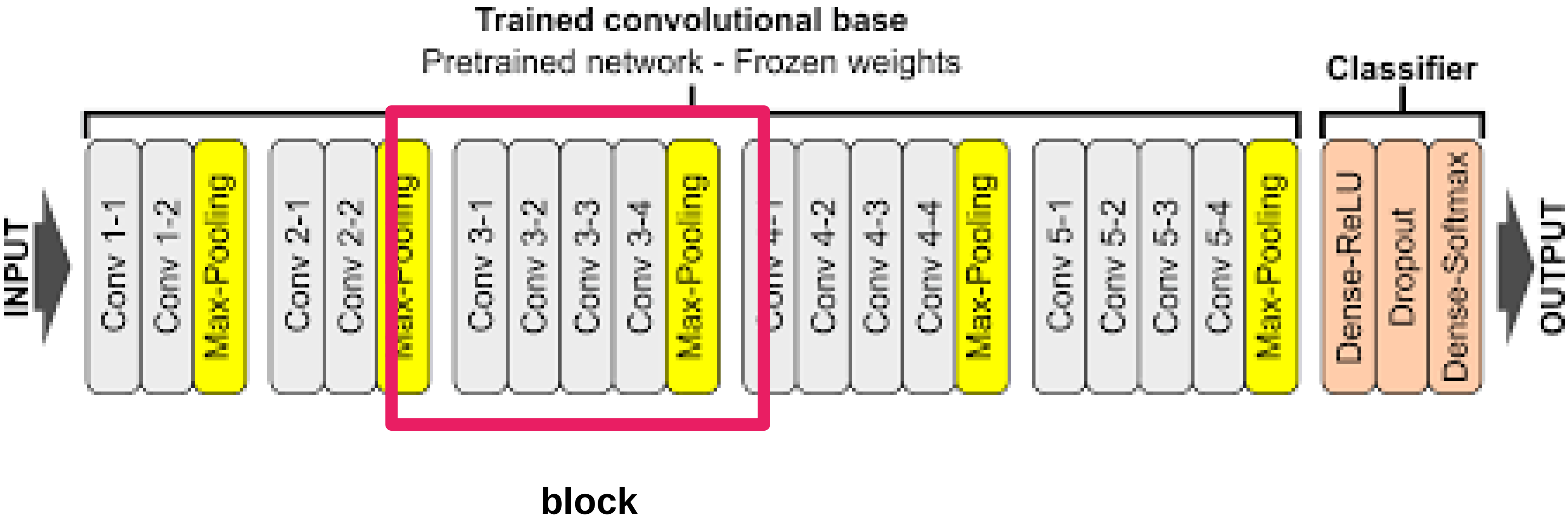
http://arxiv.org/abs/1512.03385

Deep Learning

More sophisticated model



More sophisticated model



Convolution1d + MaxPooling + LSTM + dense

1	0	-1
1	0	-1
1	0	-1

Kernel
(mask)

Convolution1d + MaxPooling + LSTM + dense

1	0	-1
1	0	-1
1	0	-1

Kernel
(mask)

0	1	1
0	1	0
1	-1	1

Kernel

-1	-1	1
0	1	-1
0	1	1

Kernel

Convolution1d + MaxPooling + LSTM + dense

```
[ 0  0  0 ]
[ 0  0  0 ]
[ 0  0  0 ]

[ 0  0  0 ]
[ 0  1  0 ]   (center pixel-ish)
[ 0  0  0 ]

[ 0  1  0 ]
[ 0  1  0 ]   (Vertical line-ish)
[ 0  1  0 ]

[ 0  0  0 ]
[ 1  1  1 ]   (Horizontal line-ish)
[ 0  0  0 ]

… (2^9 = 512 possibilities)
```

1	0	-1
1	0	-1
1	0	-1

Kernel

0	1	1
0	1	0
1	-1	1

Kernel

-1	-1	1
0	1	-1
0	1	1

Kernel

Convolution1d + MaxPooling + LSTM + dense

$$\begin{bmatrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 6 & & \\ & & \\ & & \end{bmatrix}$$

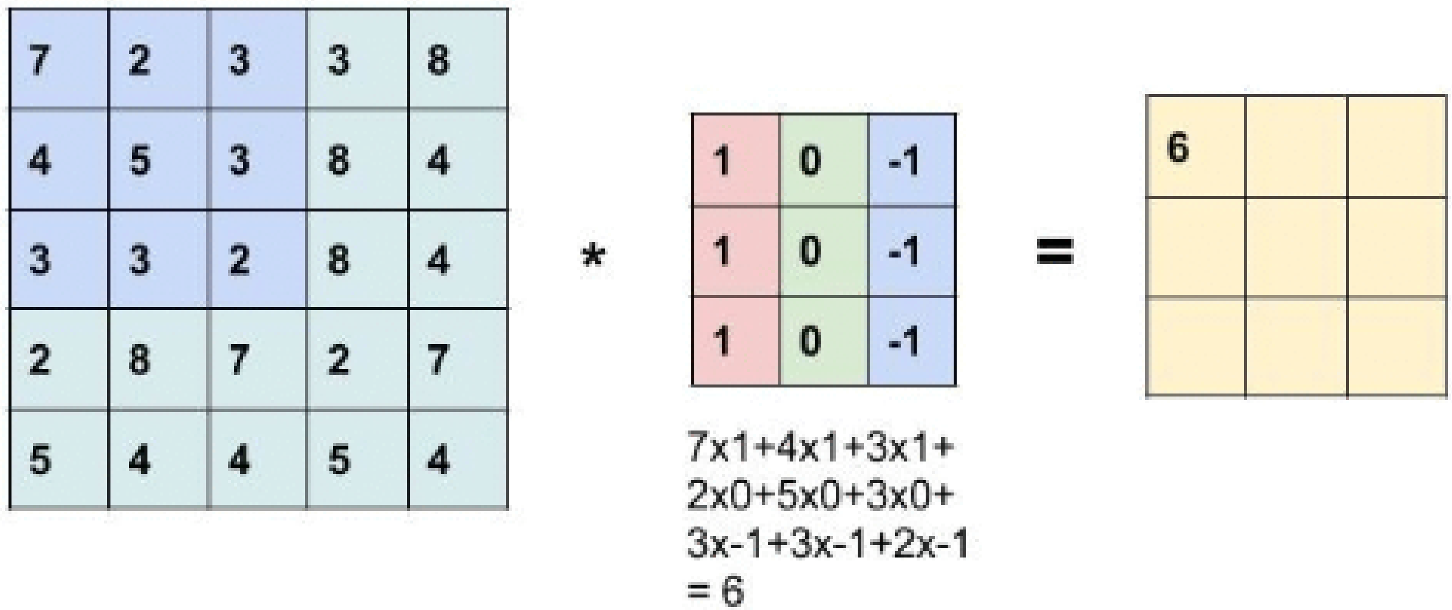
The general expression of a convolution is

$$g_{x,y} = \omega * f_{x,y} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega_{i,j} f_{x-i,y-j},$$

where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image, ω is the filter kernel. Every element of the filter kernel is considered by $-a \leq i \leq a$ and $-b \leq j \leq b$.

Deep Learning

Convolution1d + MaxPooling + LSTM + dense



The general expression of a convolution is

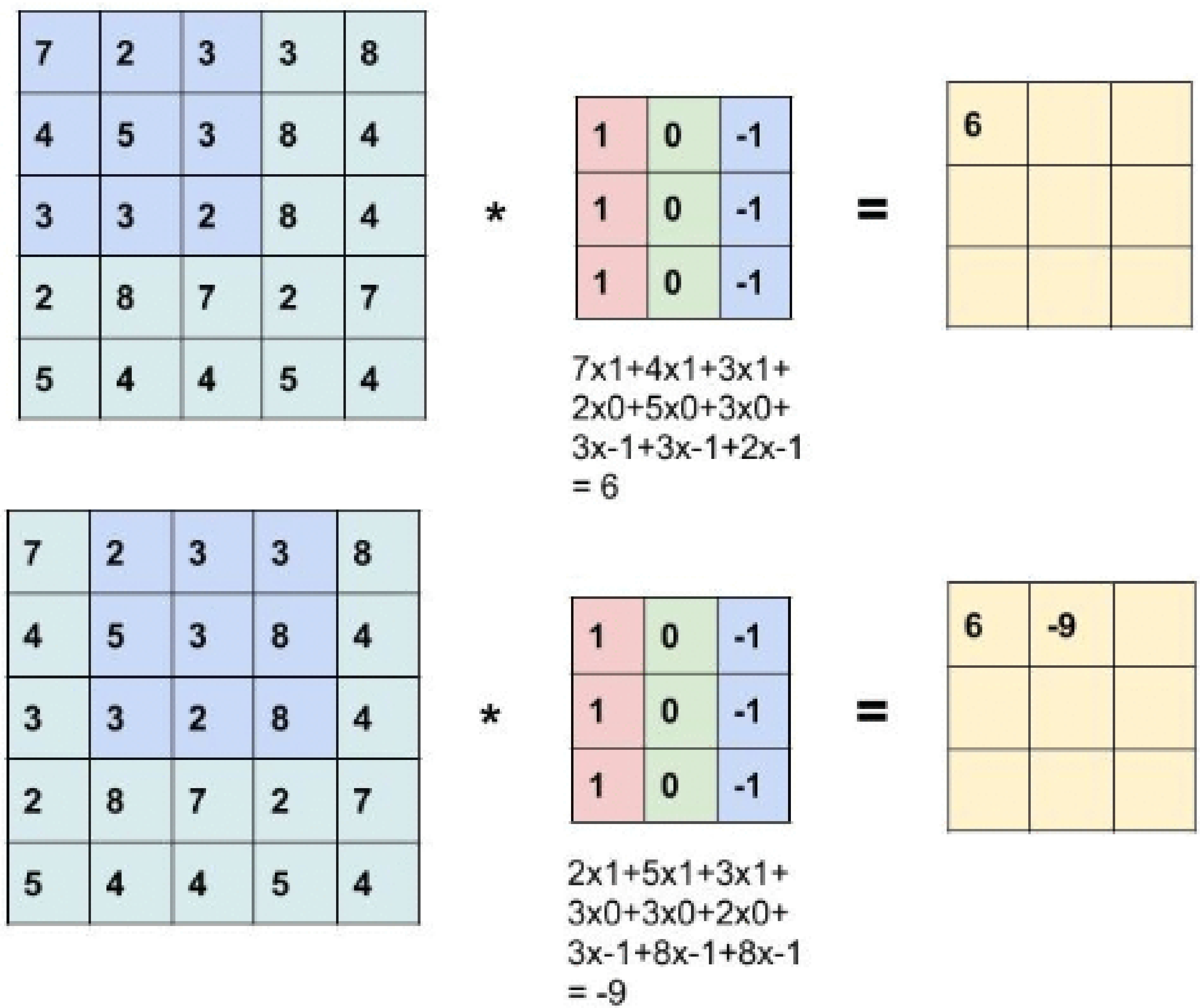
$$g_{x,y} = \omega * f_{x,y} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega_{i,j} f_{x-i,y-j},$$

where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image, ω is the filter kernel. Every element of the filter kernel is considered by $-a \le i \le a$ and $-b \le j \le b$.

an element-wise matrix multiplication followed by summing it up

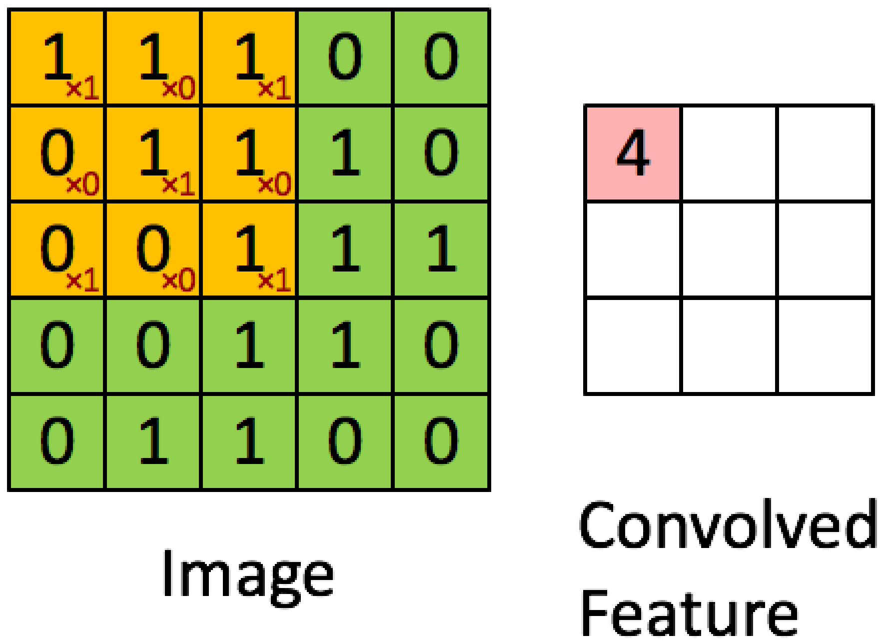
Deep Learning

Convolution1d + MaxPooling + LSTM + dense



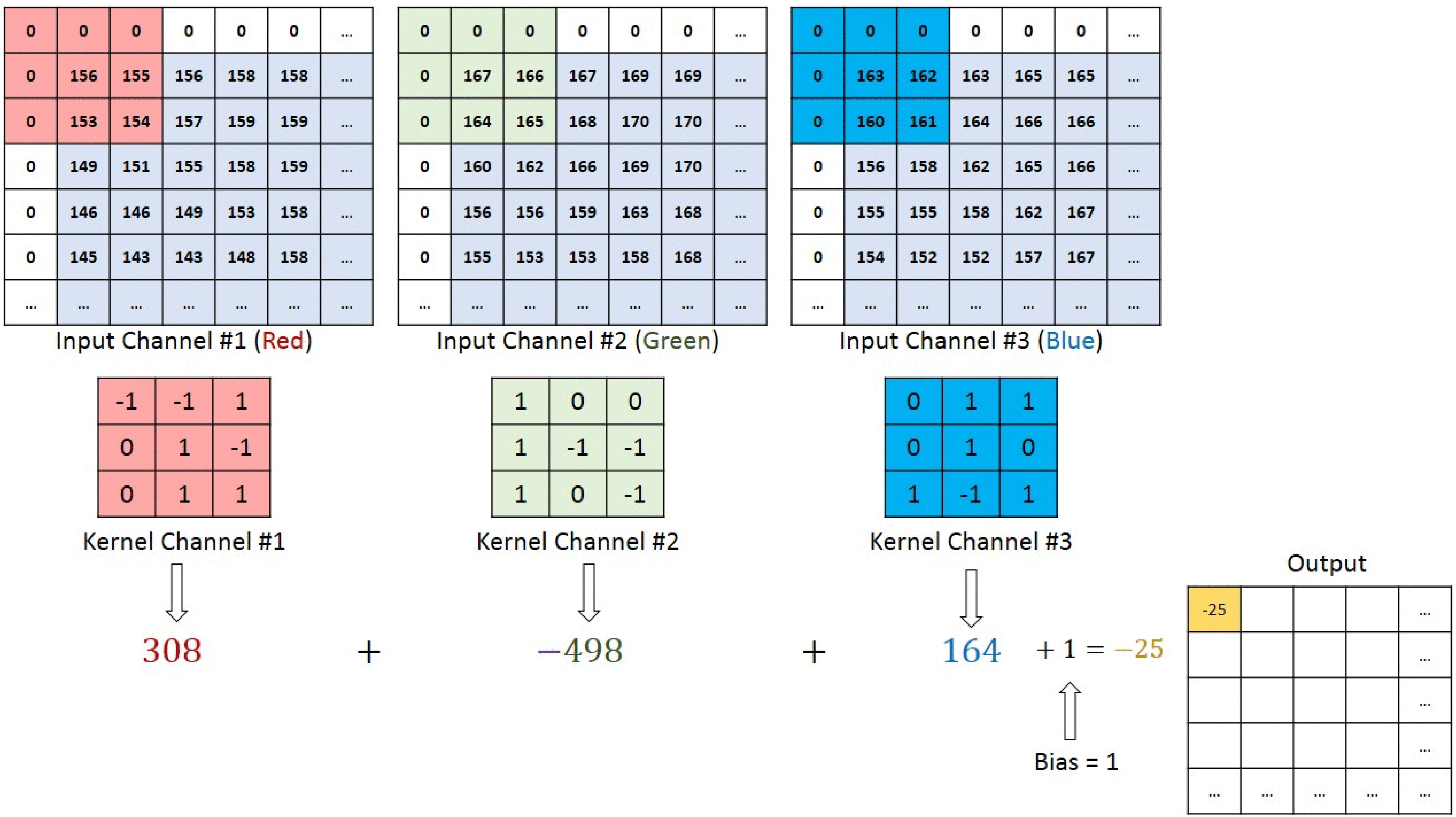
an element-wise matrix multiplication followed by summing it up

Convolution1d + MaxPooling + LSTM + dense



Convolution1d + MaxPooling + LSTM + dense

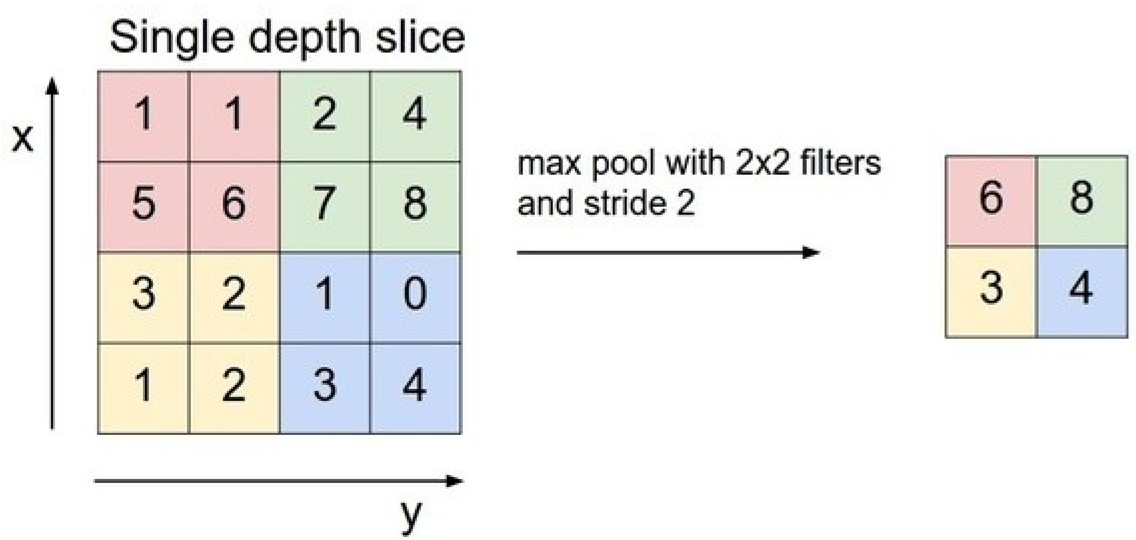
Convolution of a 3 channel image with a 3x3x3 kernel



https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

More sophisticated model

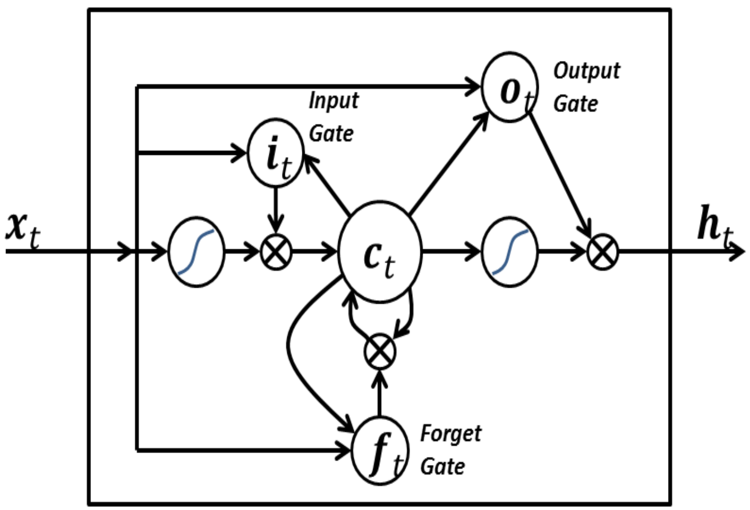
Convolution1d + **MaxPooling** + LSTM + dense



LSTM – long-short term memory

recurrent neural network

Convolution1d + MaxPooling + **LSTM** + dense



Traditional LSTM with forget gates.[2][7] $c_0 = 0$ and $h_0 = 0$. $\circ$ denotes the Hadamard product.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Variables

- x_t: input vector
- h_t: output vector
- c_t: cell state vector
- W, U and b: parameter matrices and vector
- f_t, i_t and o_t: gate vectors
  - f_t: Forget gate vector. Weight of remembering old information.
  - i_t: Input gate vector. Weight of acquiring new information.
  - o_t: Output gate vector. Output candidate.

Activation functions

- σ_g: The original is a sigmoid function.
- σ_c: The original is a hyperbolic tangent.
- σ_h: The original is a hyperbolic tangent, but the peephole LSTM paper suggests $\sigma_h(x) = x$.[8

https://en.wikipedia.org/wiki/Long_short-term_memory
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Optimizers

- SGD (stochastic gradient descent)
- Momentum
- Nesterov
- Adagrad
- Adadelta
- Rmsprop
- ADAM
- ADAMAX

...

Optimizers

- **SGD (stochastic gradient descent)**
- **Momentum**
- **Nesterov**
- **Adagrad**
- **Adadelta**
- **Rmsprop**
- **ADAM**
- **ADAMAX**

...

Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameter (**TensorFlow default**)

ADAM adaptive learning rates for each parameter with storing an exponentially decaying average of past squared gradients (like Adadelta and RMSprop). Additionally keeps an exponentially decaying average of past gradients, similar to momentum **(perform the best)**

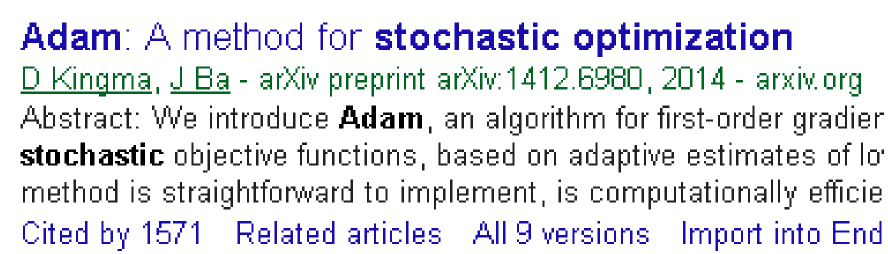
Optimizers

- **SGD (stochastic gradient descent)**
- **Momentum**
- **Nesterov**
- **Adagrad**
- **Adadelta**
- **Rmsprop**
- **ADAM**
- **ADAMAX**

...

Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameter (**TensorFlow default**)

ADAM adaptive learning rates for each parameter with storing an exponentially decaying average of past squared gradients (like Adadelta and RMSprop). Additionally keeps an exponentially decaying average of past gradients, similar to momentum **(perform the best)**

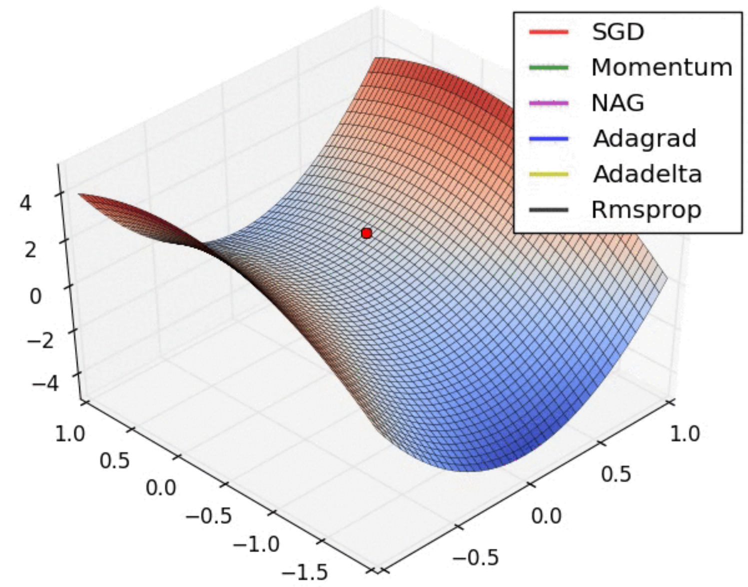

Adam: A method for **stochastic optimization**
D Kingma, J Ba - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org
Abstract: We introduce **Adam**, an algorithm for first-order gradier
stochastic objective functions, based on adaptive estimates of lo
method is straightforward to implement, is computationally efficie
Cited by 1571 Related articles All 9 versions Import into End

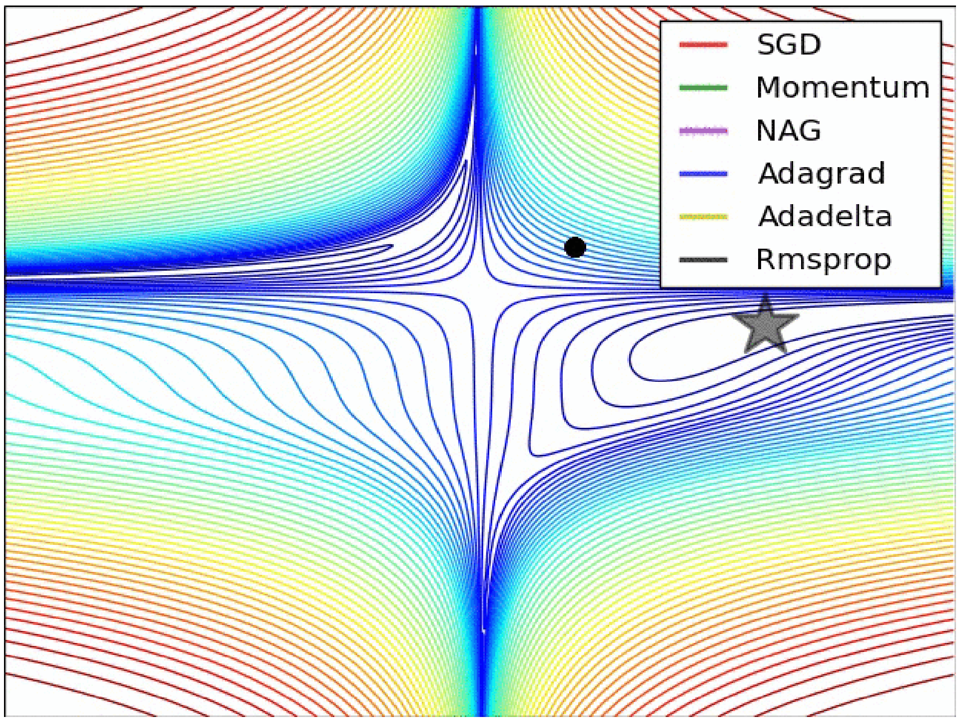
Must read: http://sebastianruder.com/optimizing-gradient-descent/

Optimizers

- SGD (stochastic gradient descent)
- Momentum
- Nesterov
- Adagrad
- Adadelta
- Rmsprop
- ADAM
- ADAMAX

...



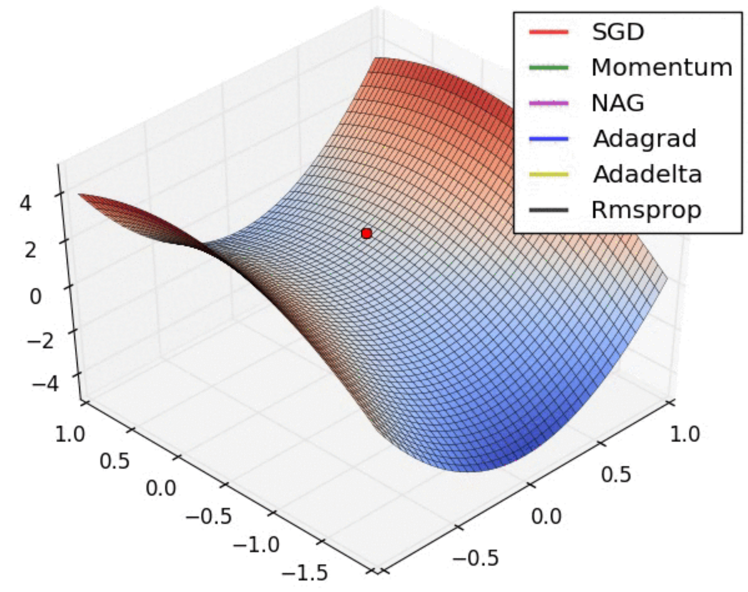


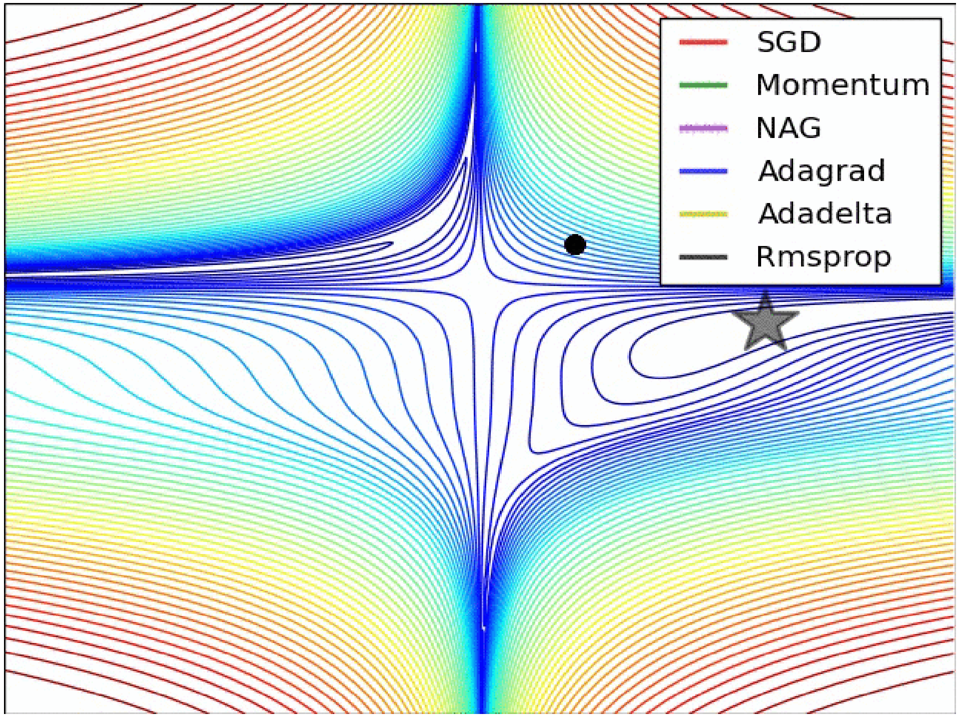
Deep Learning

Optimizers

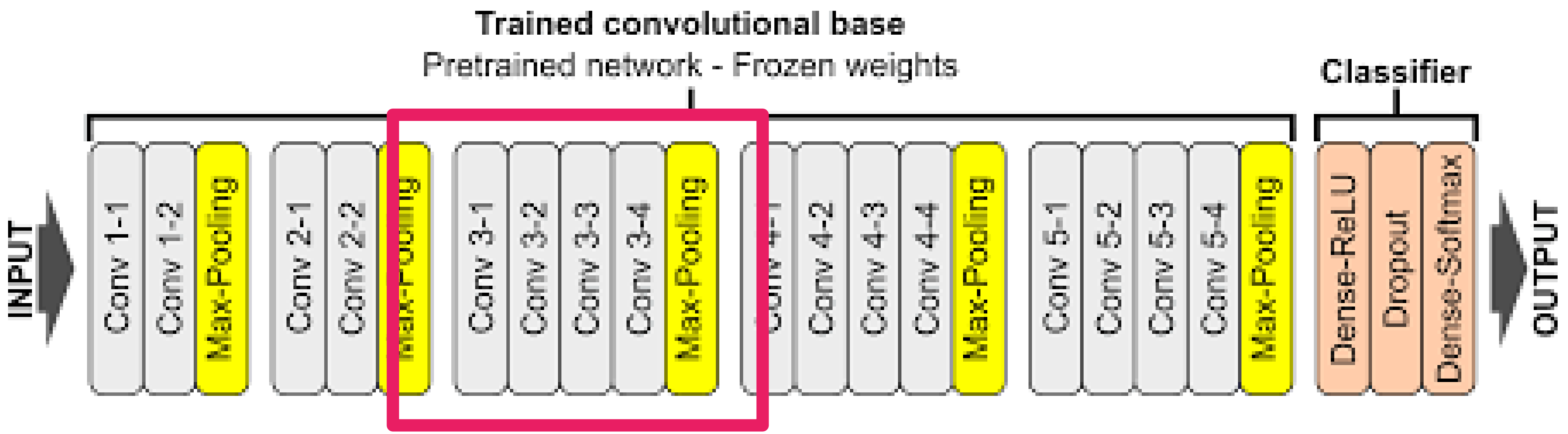
- SGD (stochastic gradient descent)
- Momentum
- Nesterov
- Adagrad
- Adadelta
- Rmsprop
- ADAM
- ADAMAX

...





Most known architectures

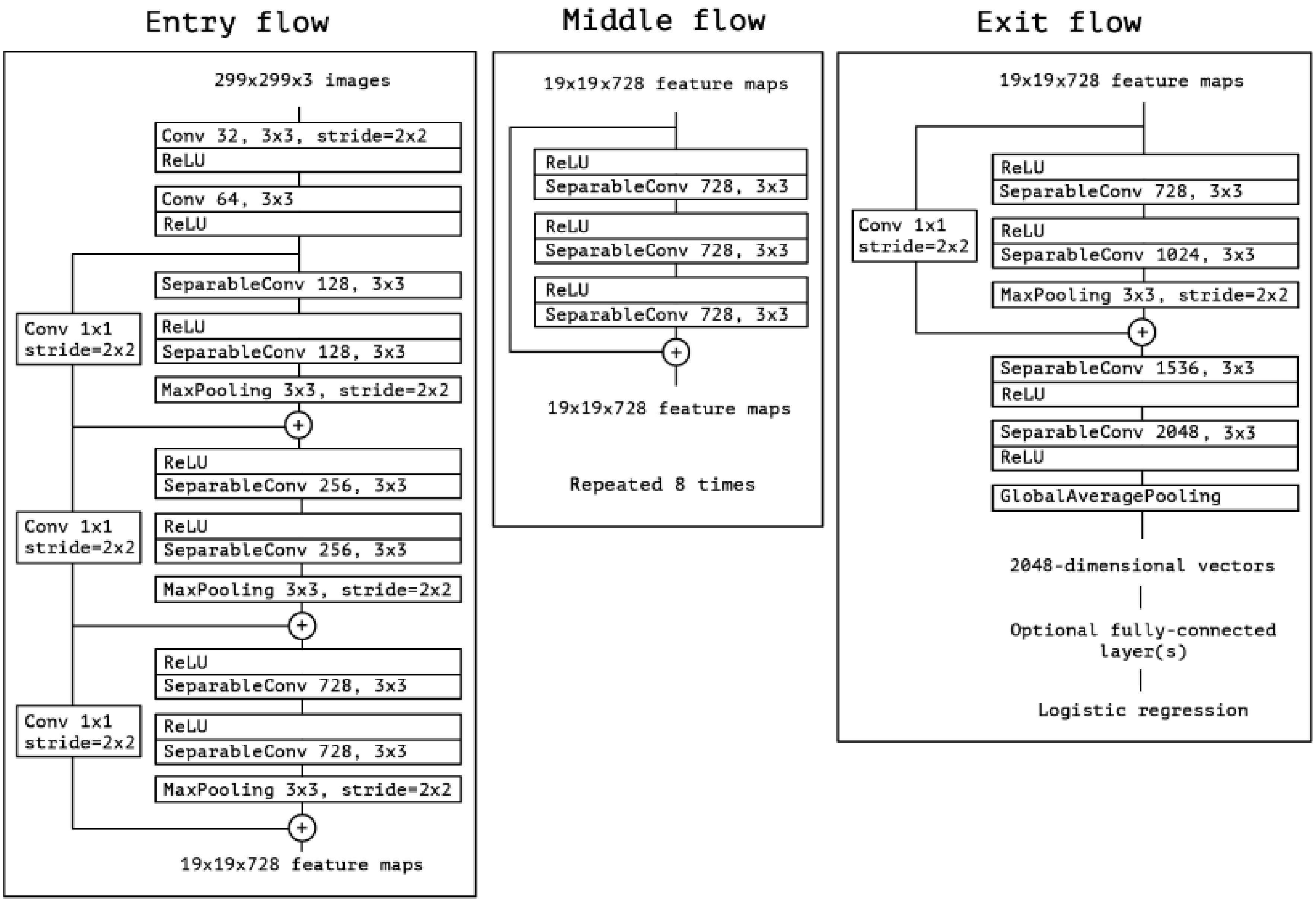


block

Deep Learning

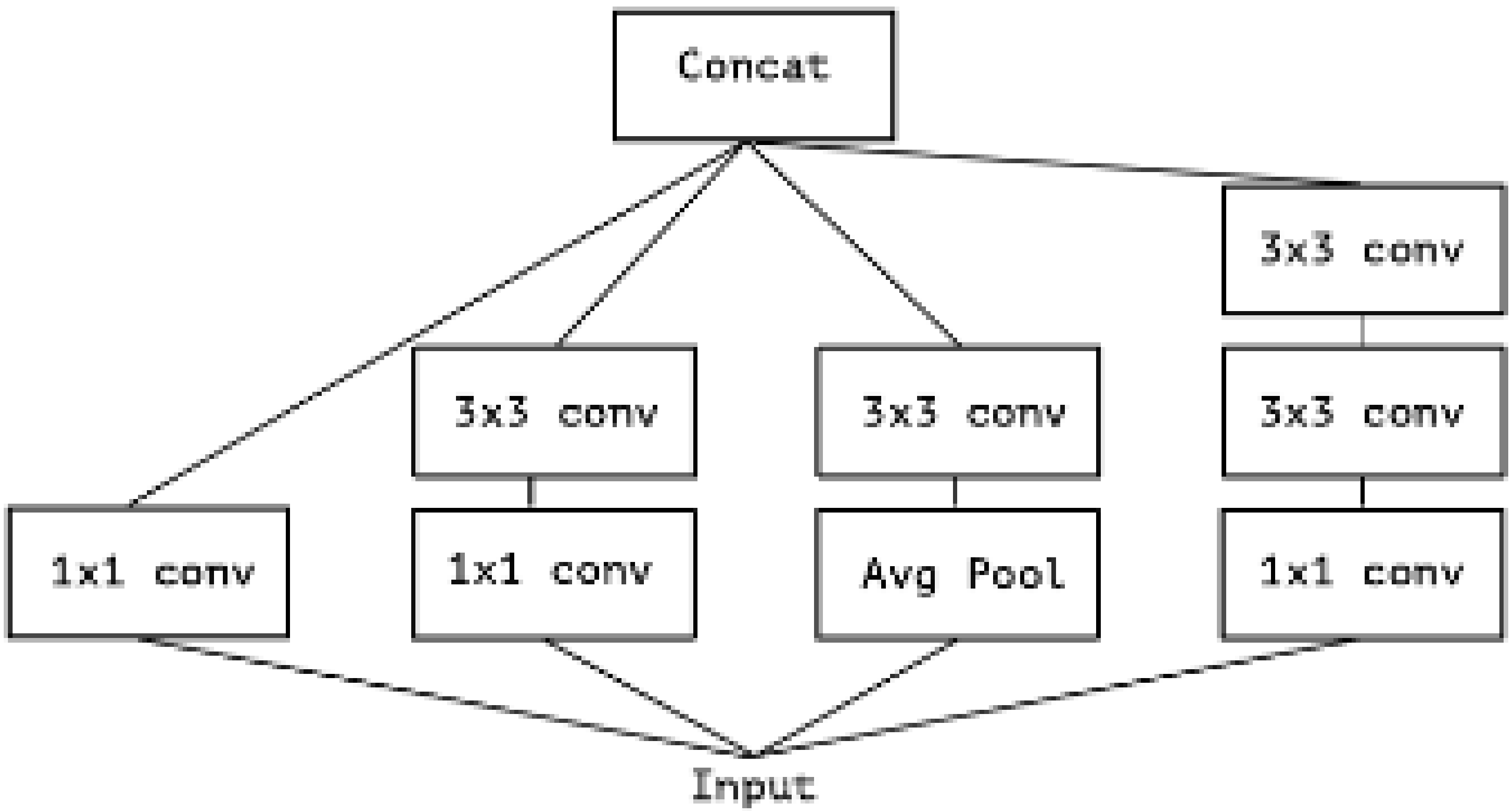
Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-

Most known architectures

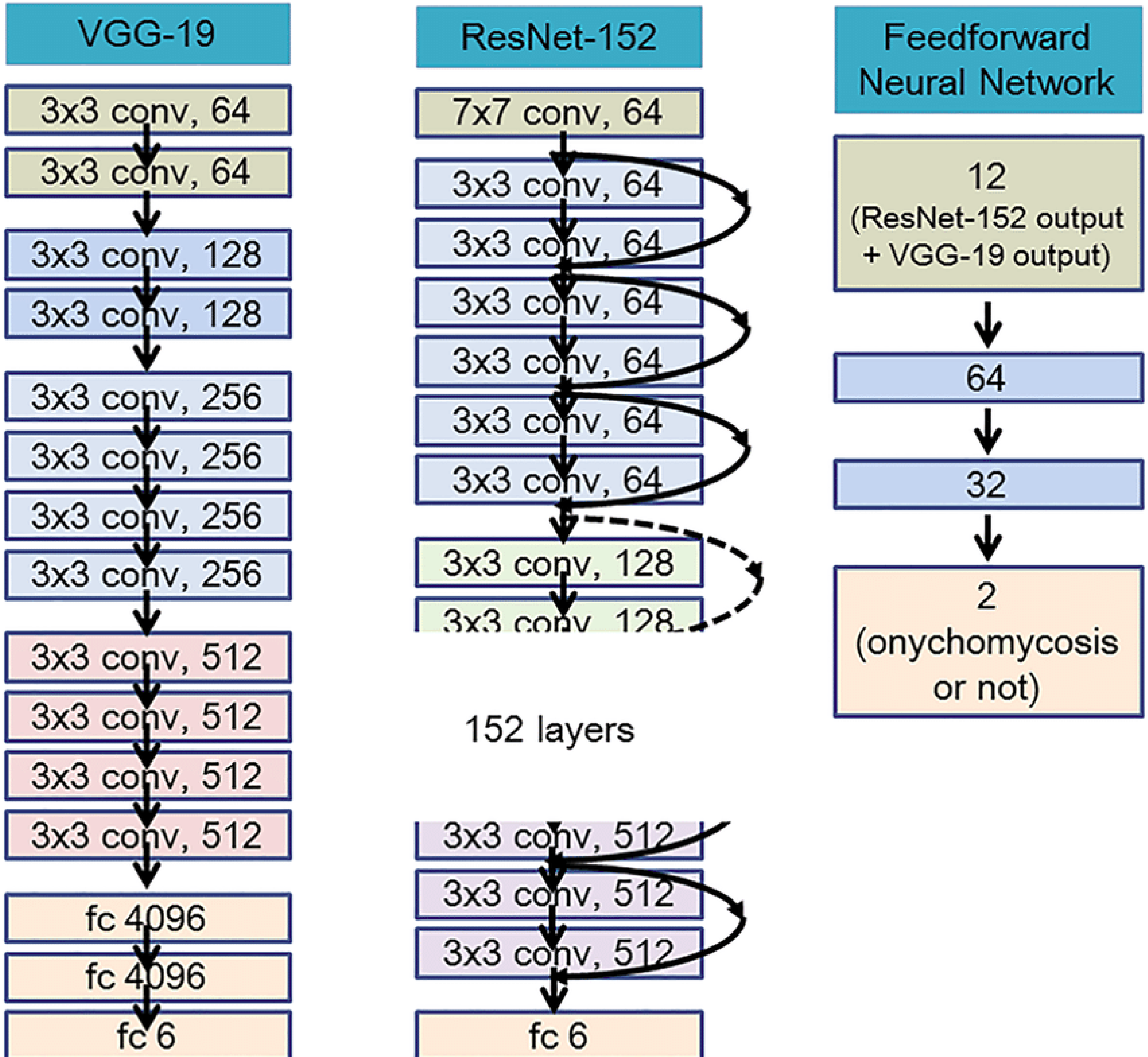


Most known architectures

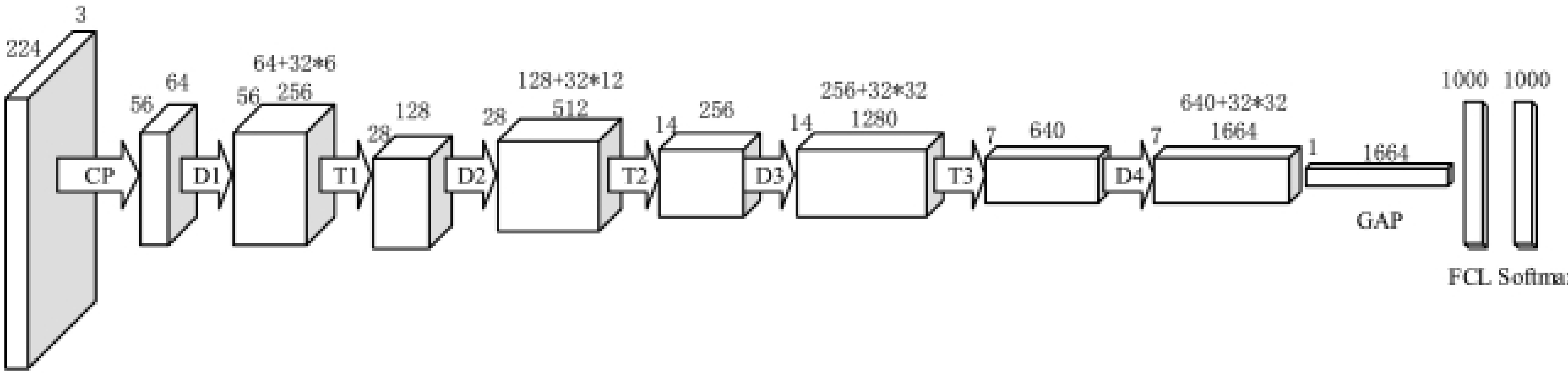
Figure 1. A canonical Inception module (Inception V3).



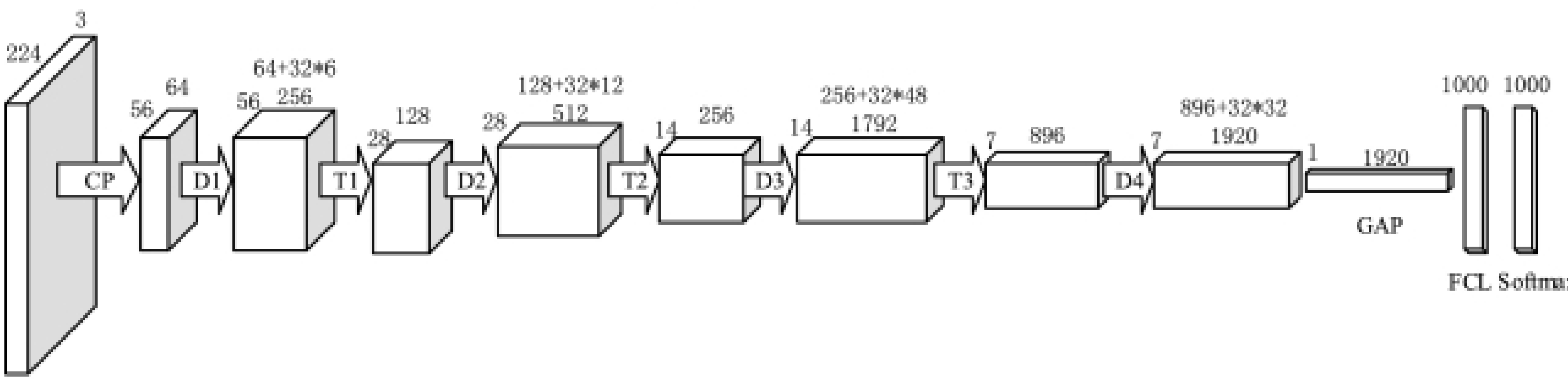
Deep Learning



Deep Learning



(a) DenseNet-169



(b) DenseNet-201

]

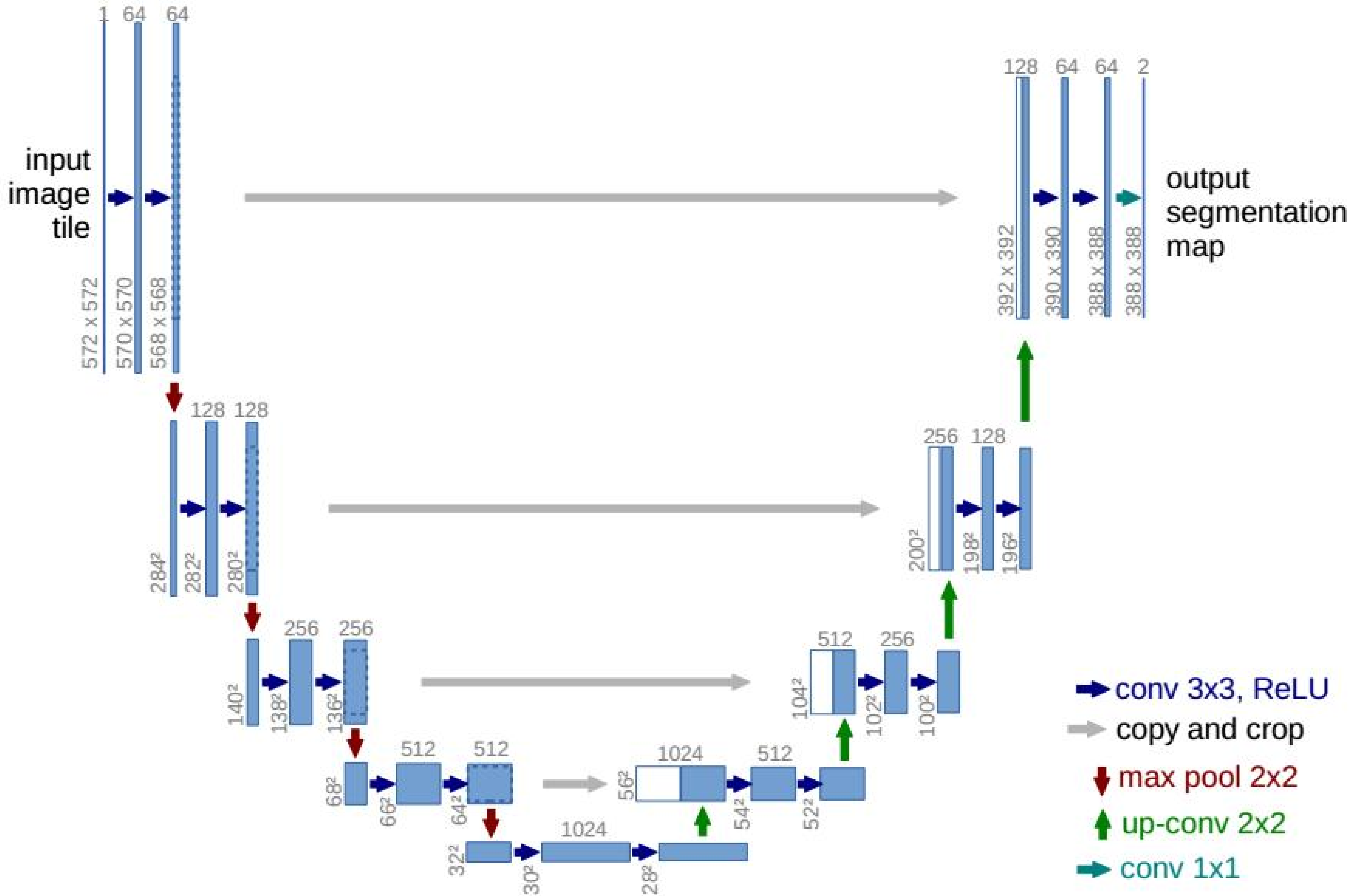
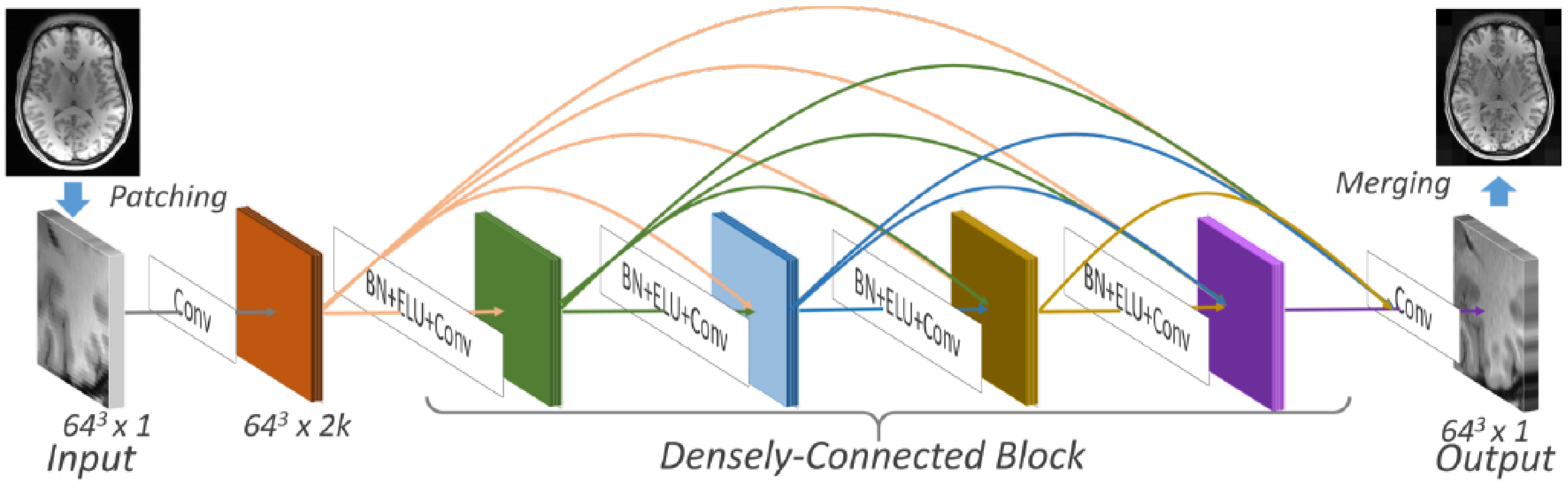
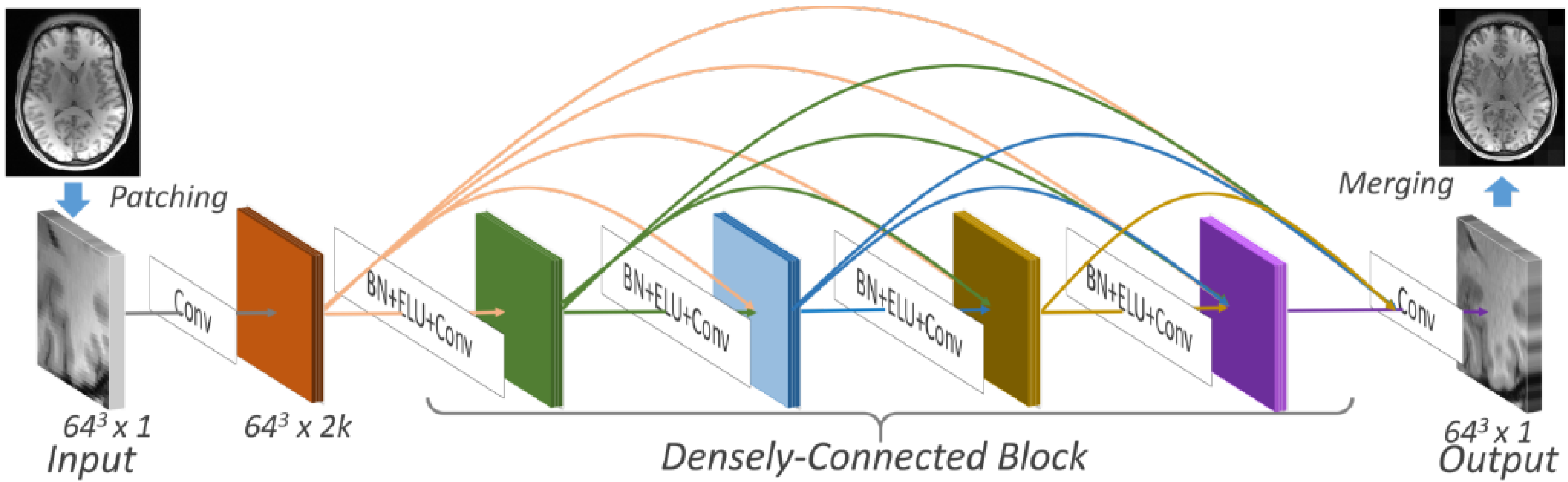


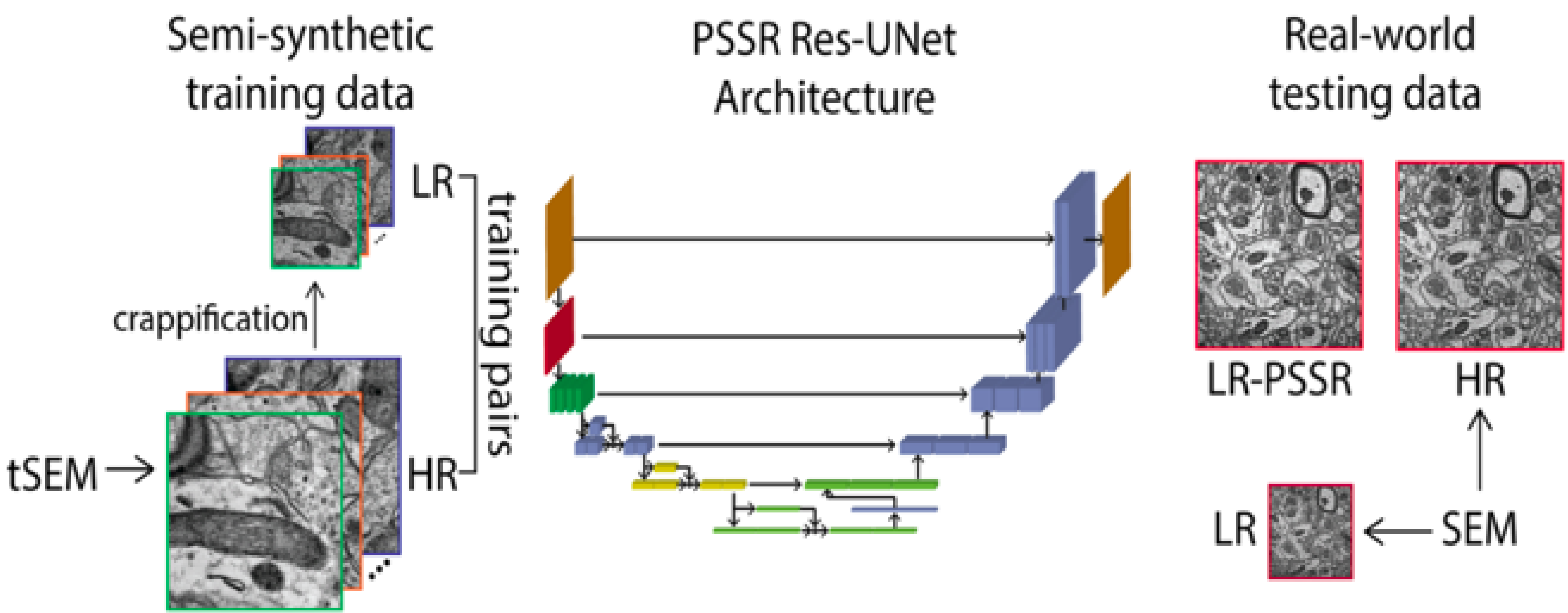
Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Deep Learning

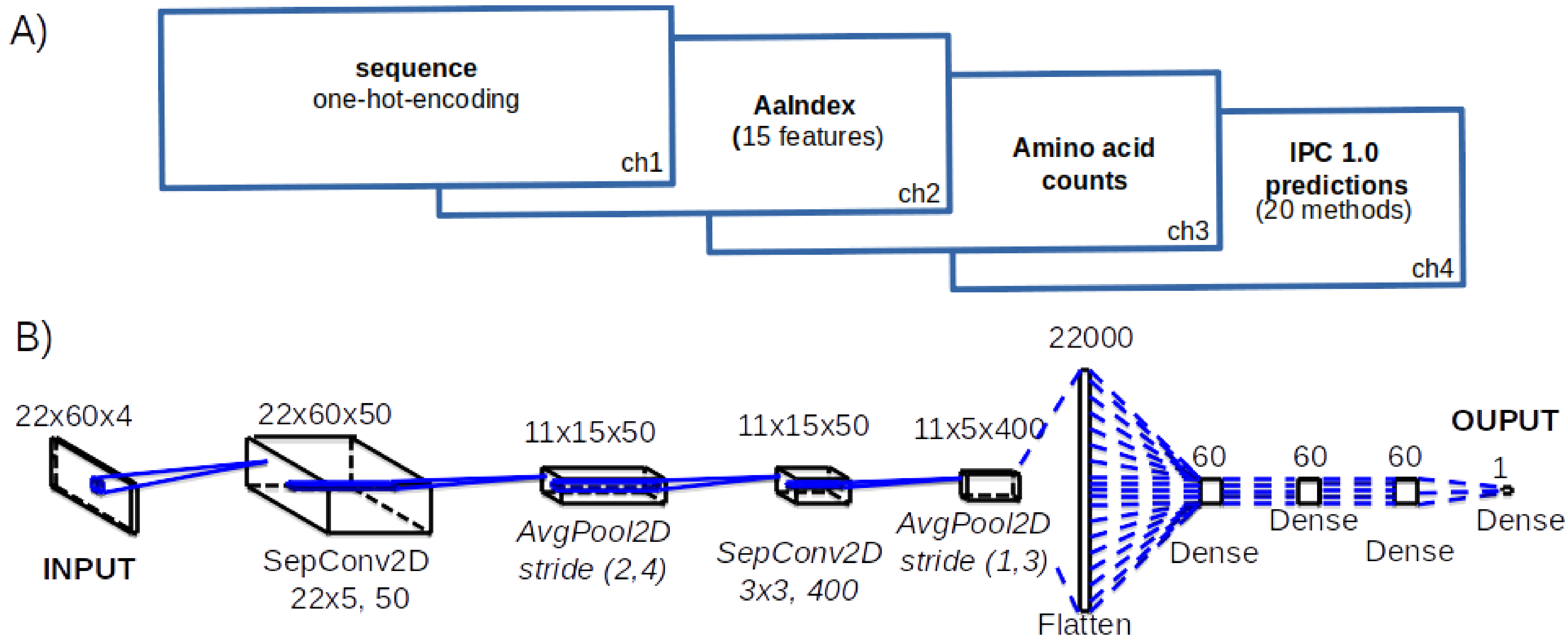


Deep Learning





Deep Learning

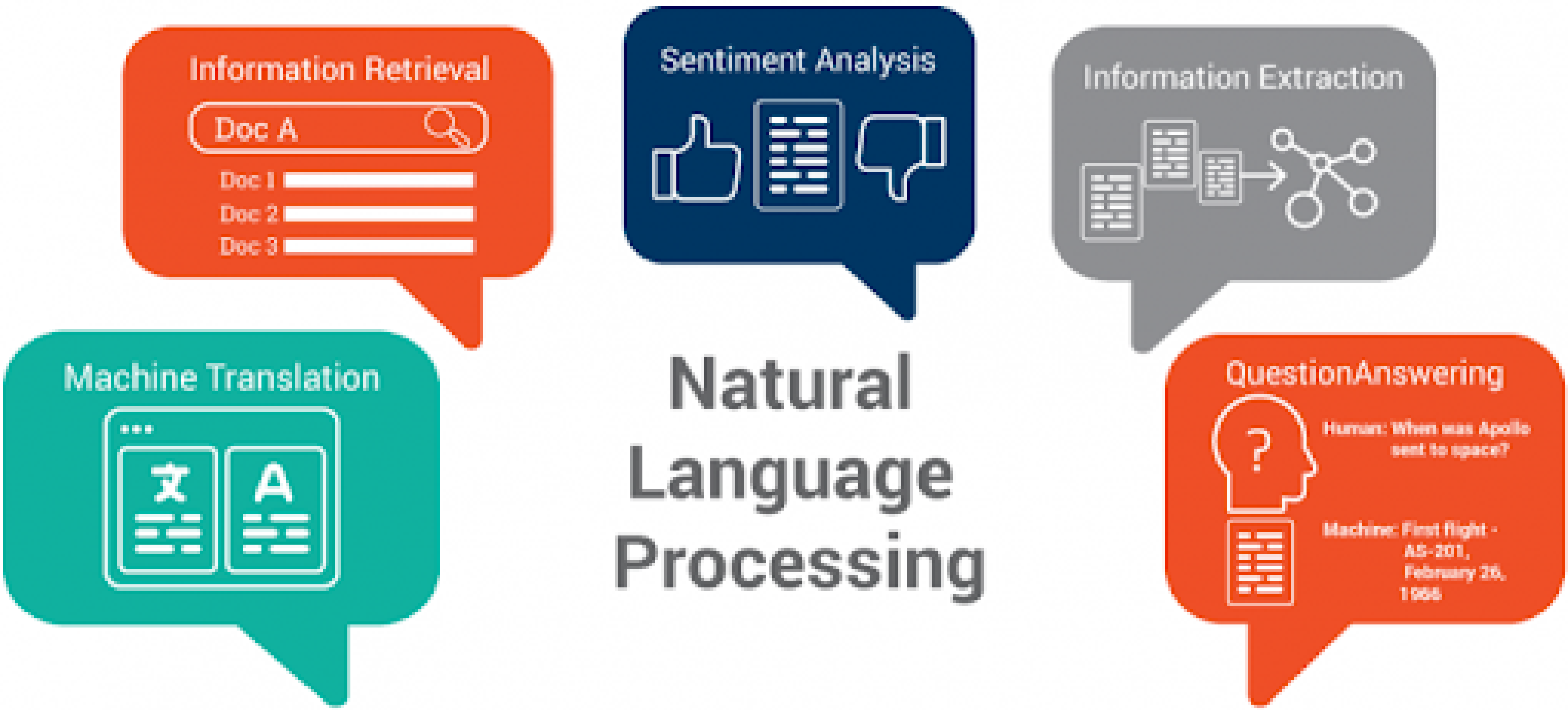



```
model = Sequential()
model.add(SeparableConv2D(f1, (fs1_1, fs1_2), padding='same',
input_shape=shape_tab[1:], activation=a1))
model.add(AveragePooling2D(pool_size=(2, 4)))
model.add(SeparableConv2D(f2, (fs2_1, fs2_2), padding='same', activation=a2))
model.add(AveragePooling2D(pool_size=(1, 3)))
model.add(Flatten())
model.add(Dense(units = 60, activation = a3))
model.add(Dense(units = 60, activation = a3))
model.add(Dense(units = 60, activation = a3))
model.add(Dense(units = 1, kernel_initializer='normal', activation="linear"))
```

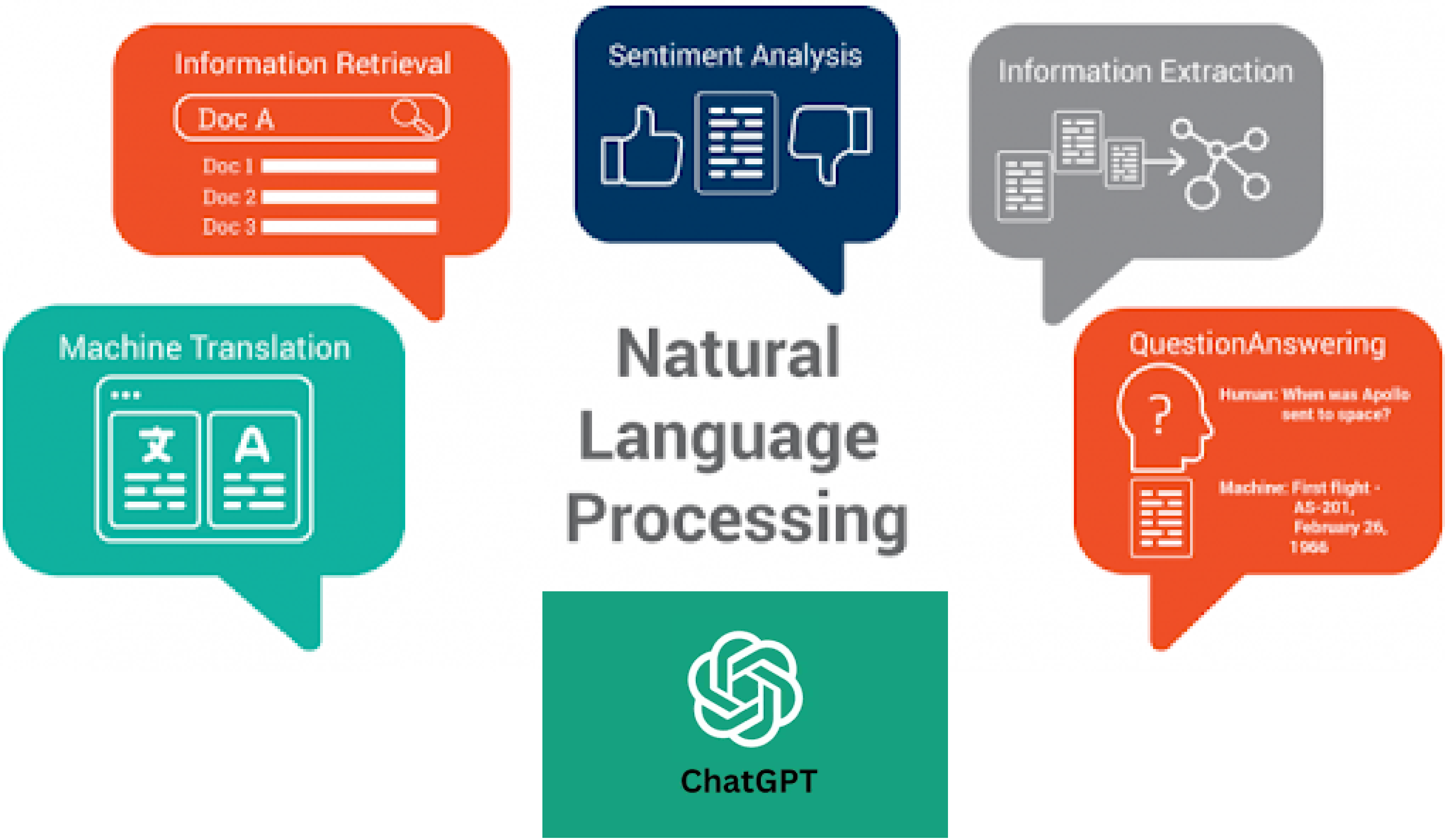
http://dx.doi.org/10.1093/nar/gkab295

Natural language Processing with deep learning

Natural language Processing with deep learning



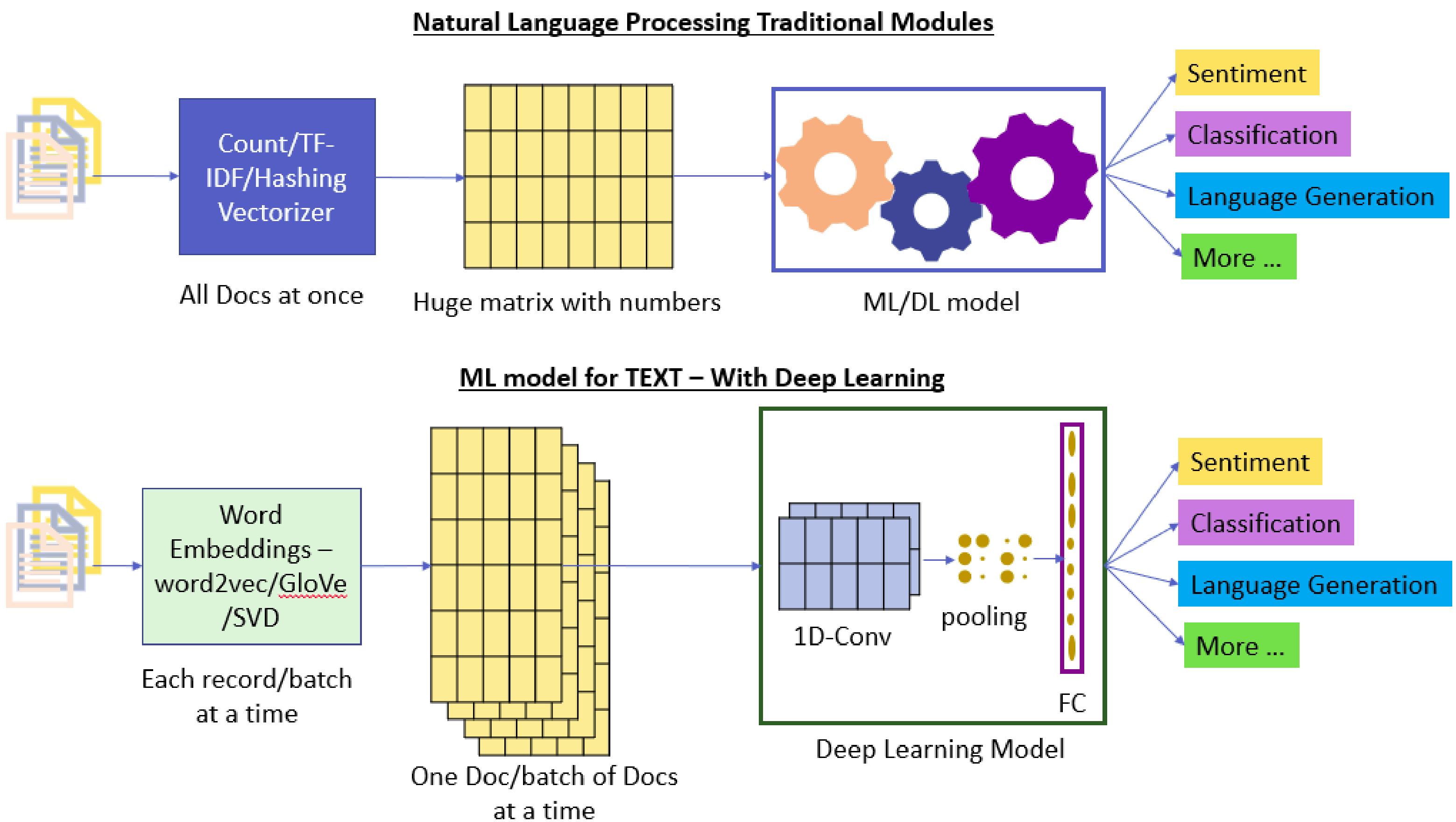
Natural language Processing with deep learning



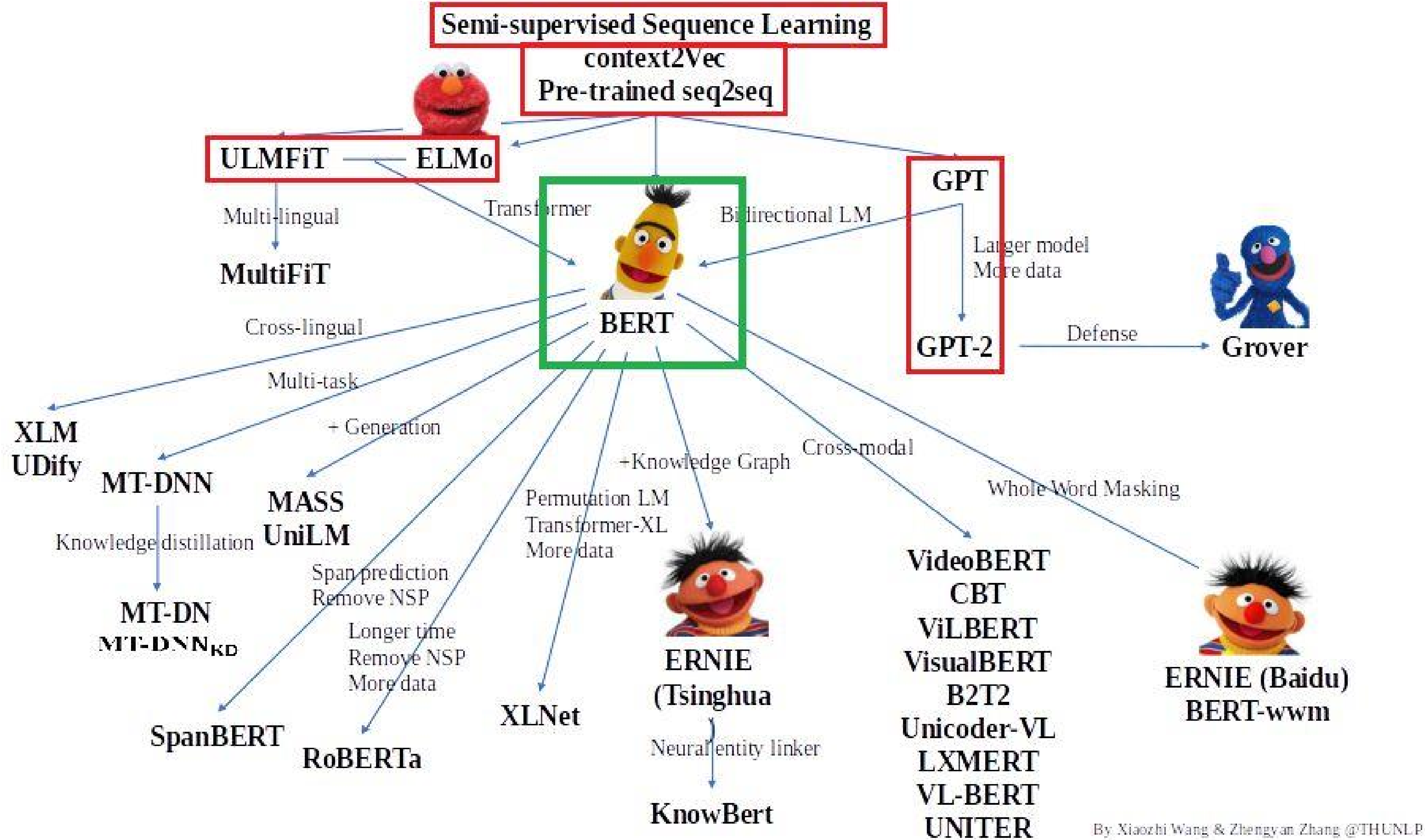
Deep Learning

Natural language Processing with deep learning

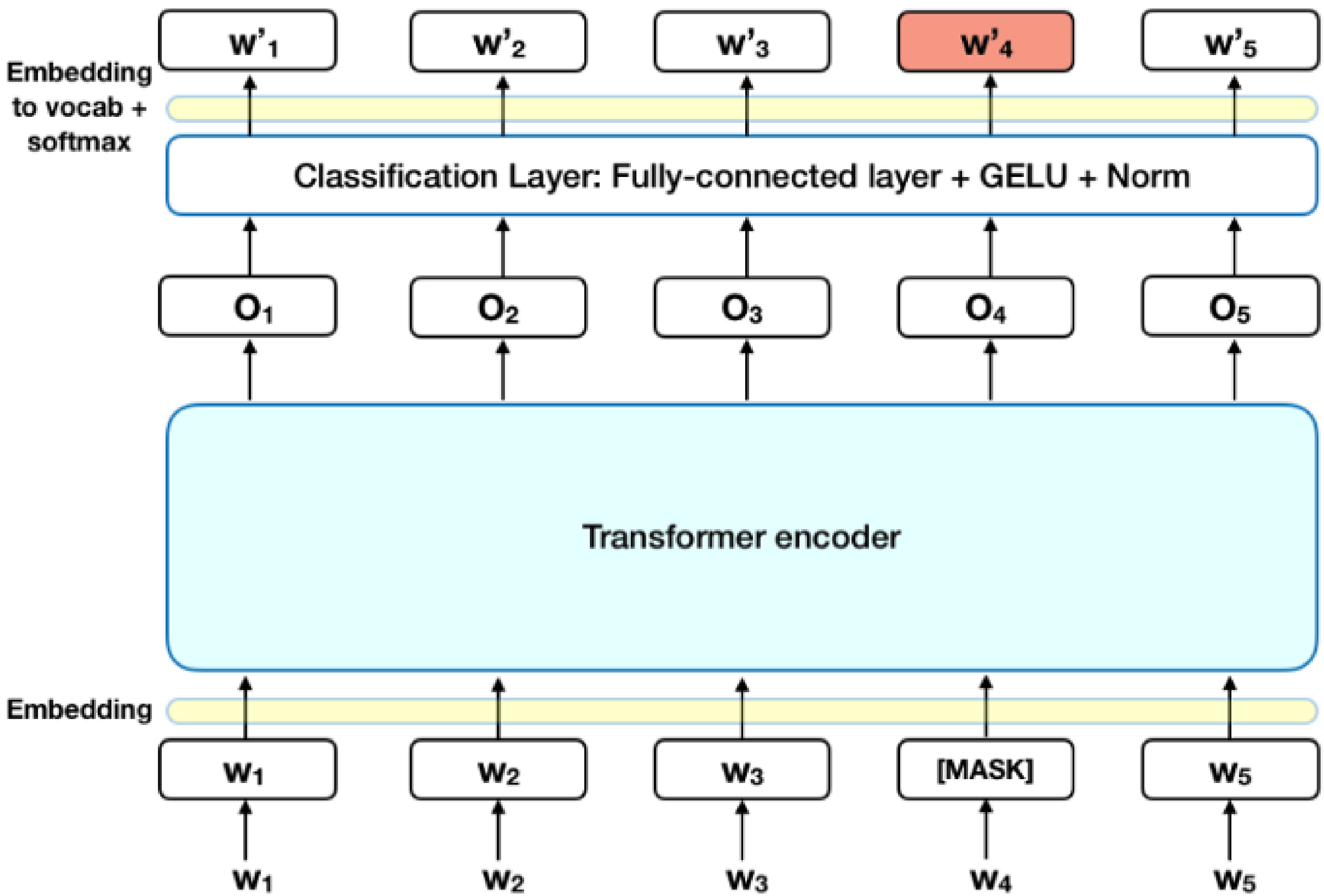
Natural language processing (NLP) deals with building computational algorithms to automatically analyze and represent human language.



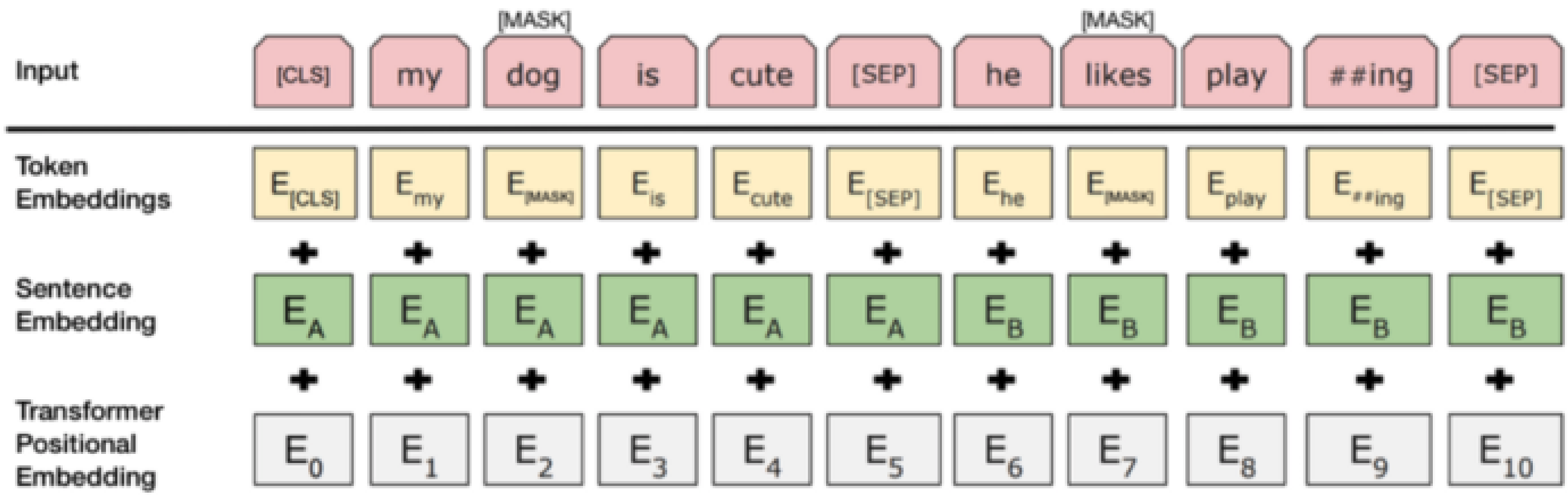
Natural language Processing with deep learning



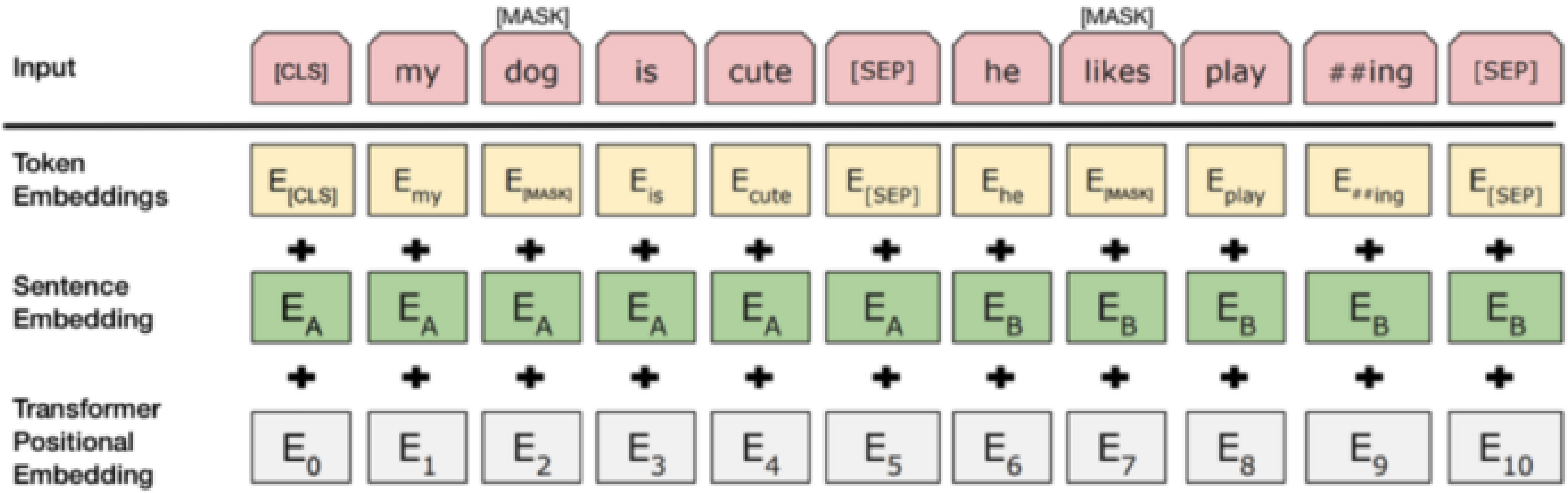
BERT (Bidirectional Encoder Representations from Transformers)

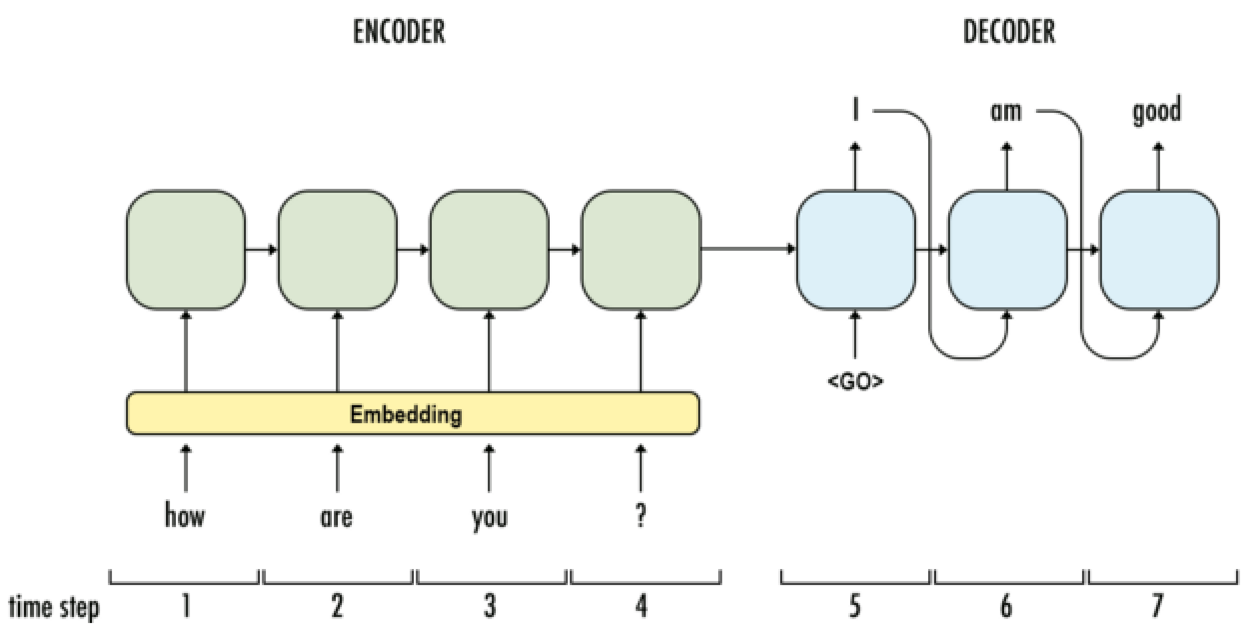


BERT (Bidirectional Encoder Representations from Transformers)



BERT (Bidirectional Encoder Representations from Transformers)





Deep Learning

BERT (Bidirectional Encoder Representations from Transformers)

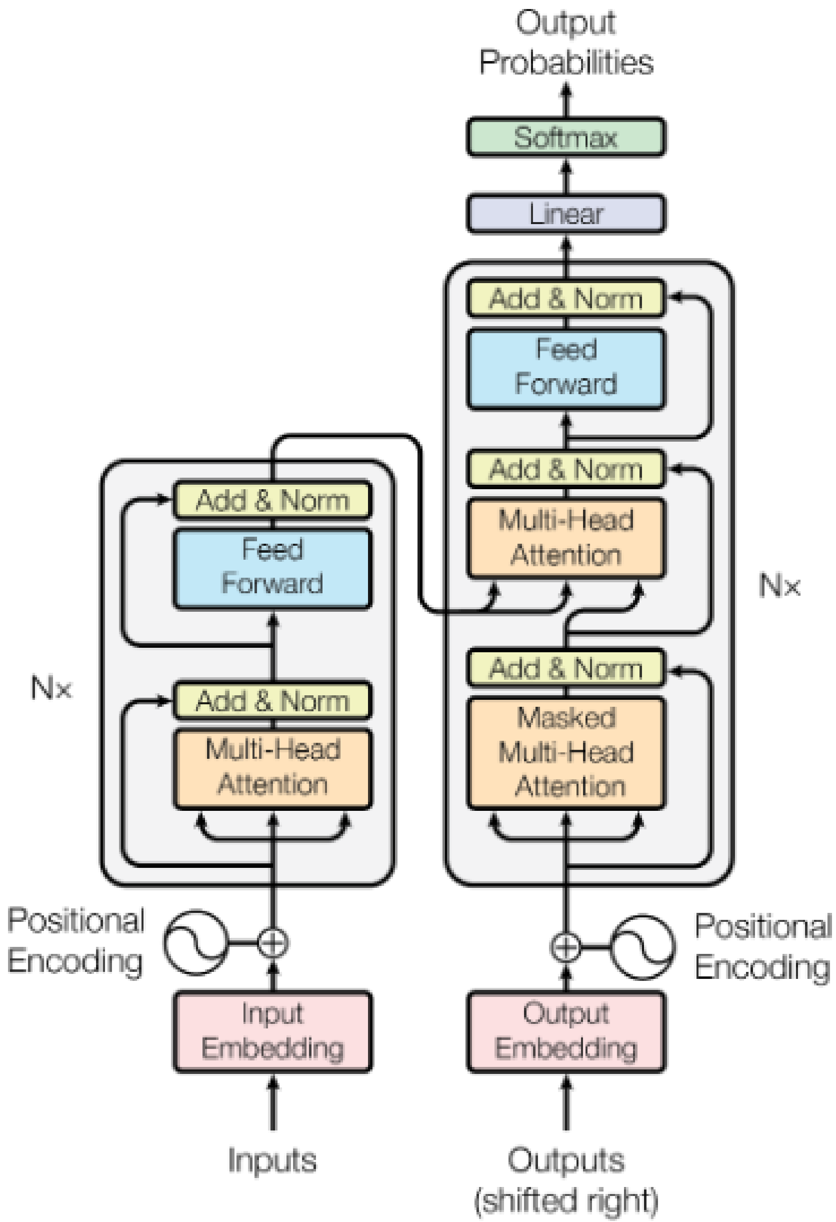


Figure 1: The Transformer - model architecture.

Deep Learning

BERT (Bidirectional Encoder Representations from Transformers)

Attention is All you Need

Part of Advances in Neural Information Processing Systems 30 (NIPS 2017)

Bibtex Metadata Paper Reviews

Authors

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

Abstract

The dominant sequence transduction models are based on complex recurrent orconvolutional neural networks in an encoder and decoder configuration. The best performing such models also connect the encoder and decoder through an attentionm echanisms. We propose a novel, simple network architecture based solely onan attention mechanism, dispensing with recurrence and convolutions entirely.Experiments on two machine translation tasks show these models to be superiorin quality while being more parallelizable and requiring significantly less timeto train. Our single model with 165 million parameters, achieves 27.5 BLEU onEnglish-to-German translation, improving over the existing best ensemble result by over 1 BLEU. On English-to-French translation, we outperform the previoussingle state-of-the-art with model by 0.7 BLEU, achieving a BLEU score of 41.1.

Attention is **all you need**

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to −∞) …

☆ Zapisz Cytuj Cytowane przez 74866 Powiązane artykuły Wszystkie wersje 46

Practical part

Splitting dataset

Step 1: Making the model *examine* data. — **Training Set**

Step 2: Making the model *learn* from its mistakes. — **Validation Set**

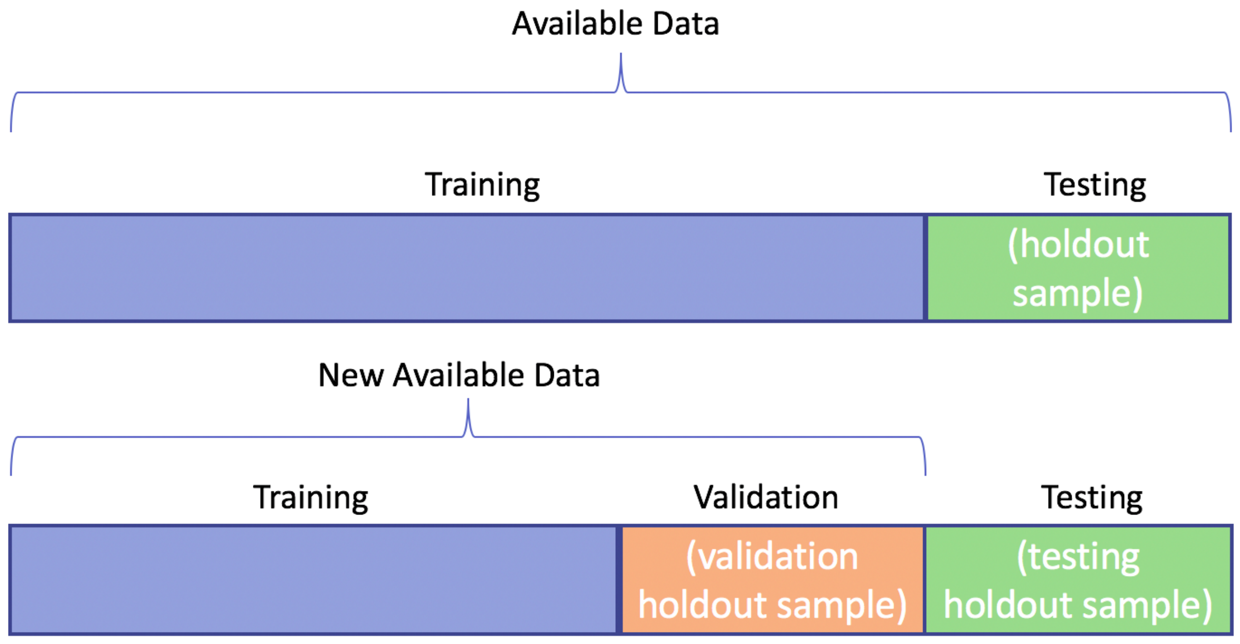
Step 3: Making a conclusion on *how well* the model performs. — **Test Set**

Splitting dataset

Step 1: Making the model *examine* data. — **Training Set**

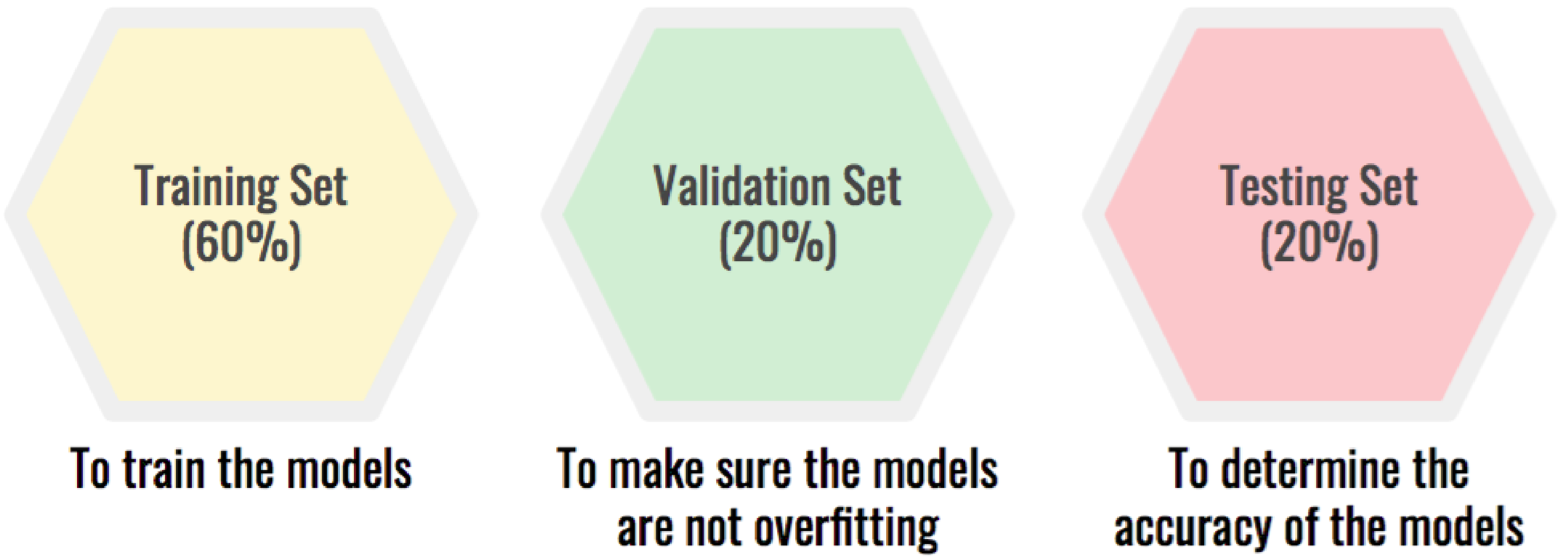
Step 2: Making the model *learn* from its mistakes. — **Validation Set**

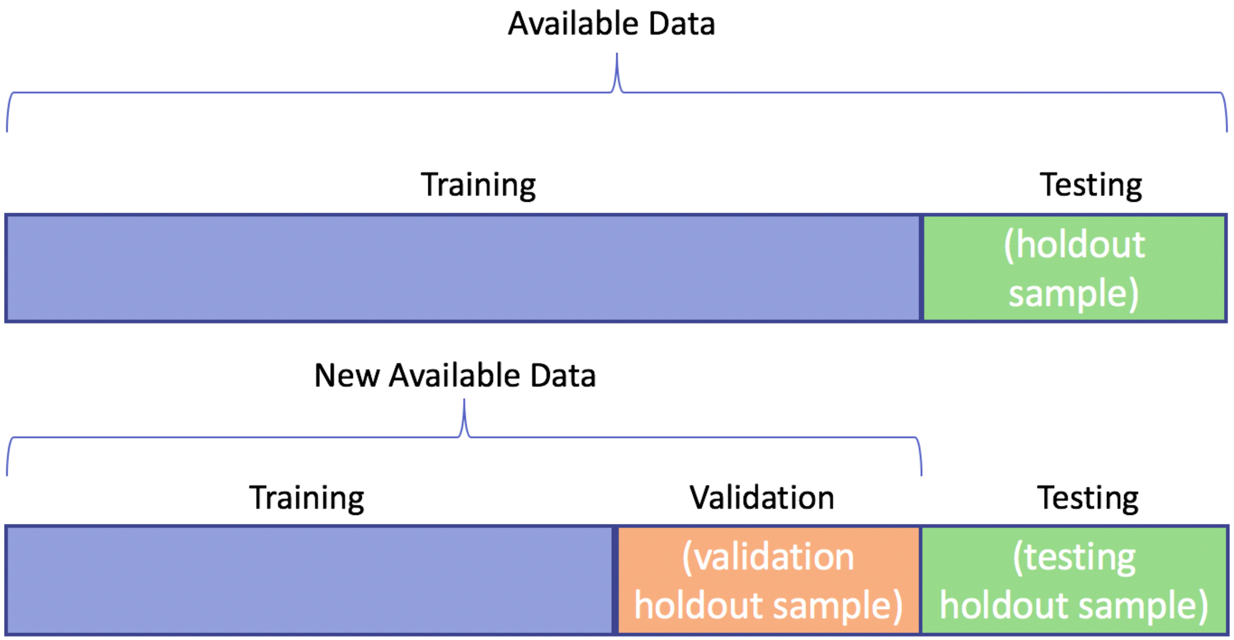
Step 3: Making a conclusion on *how well* the model performs. — **Test Set**



Neural networks (deep learning)

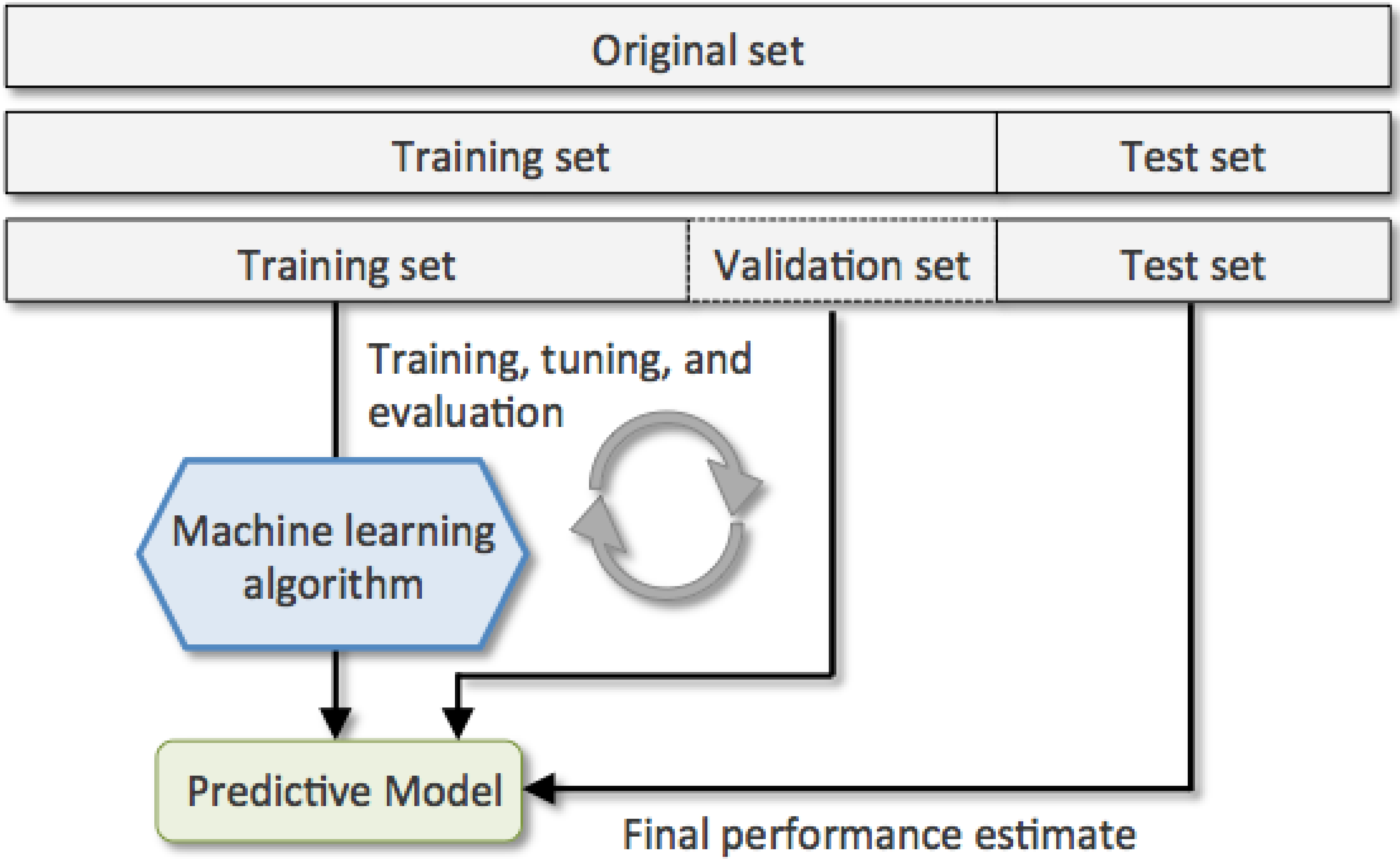
Splitting dataset





Neural networks (deep learning)

Splitting dataset



Keras

One-hot-encoding

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

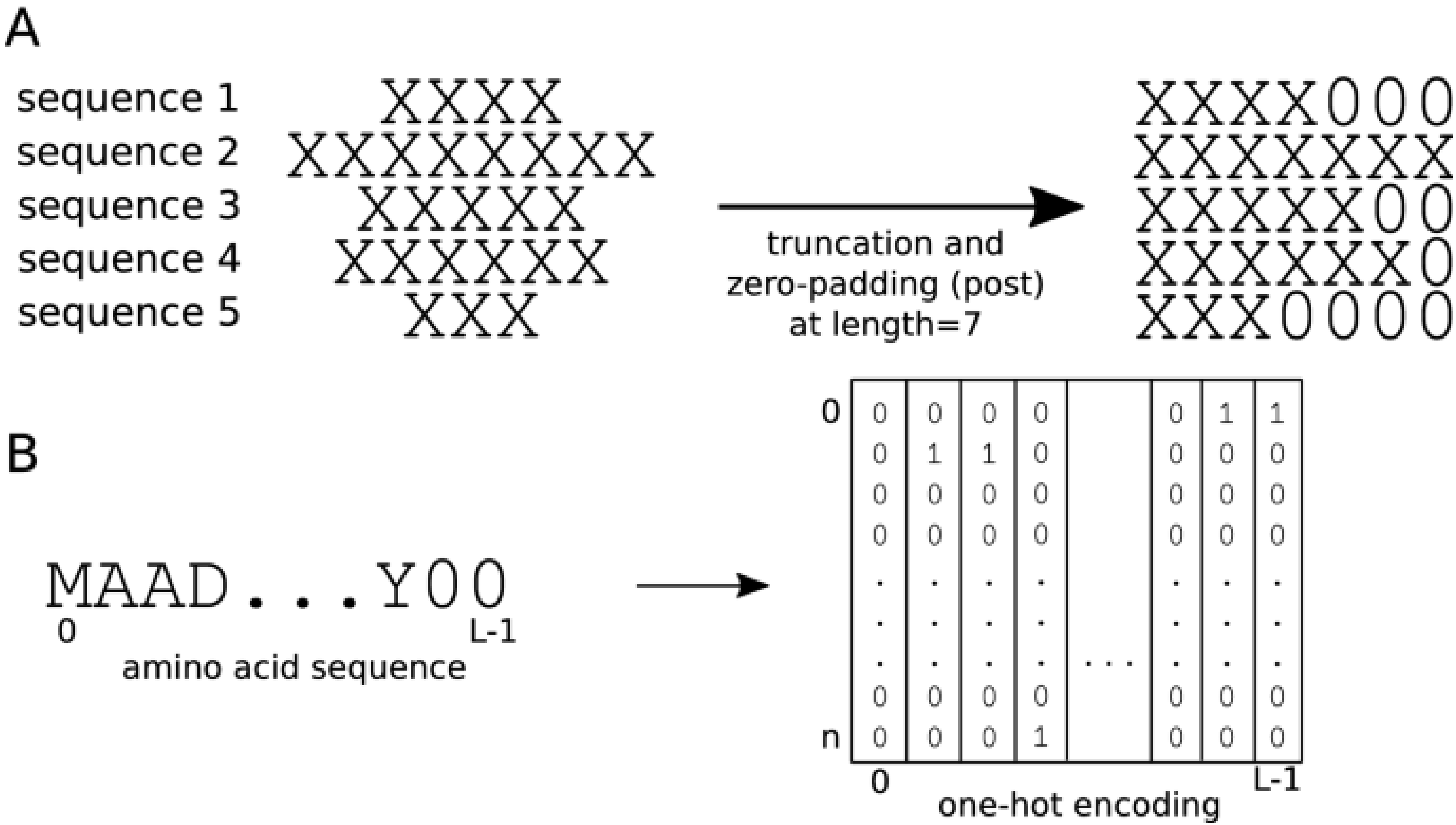
Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Keras



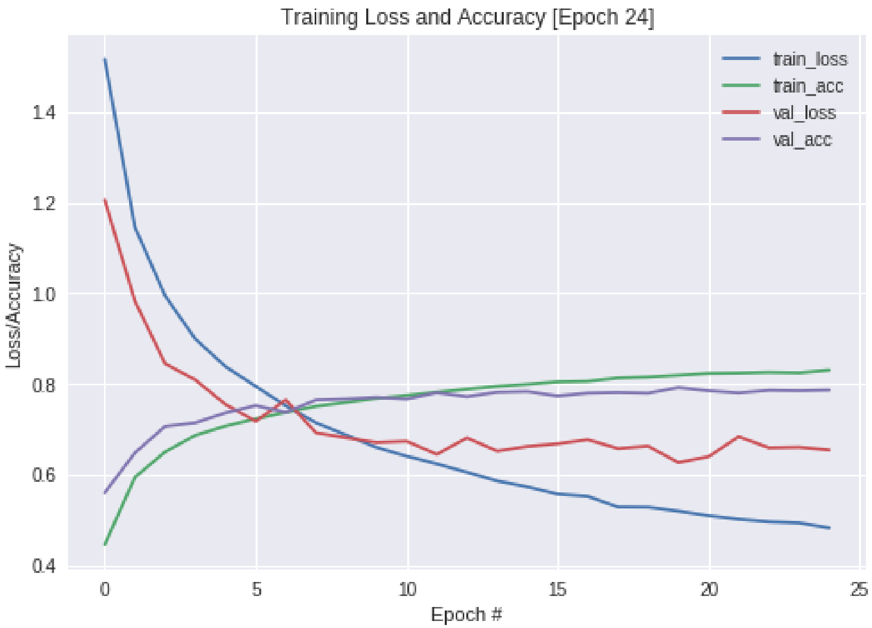
A
sequence 1
sequence 2
sequence 3
sequence 4
sequence 5
XXXX
XXXXXXXX
XXXXX
XXXXXX
XXX
truncation and
zero-padding (post)
at length=7
XXXX000
XXXXXXX
XXXXX00
XXXXXX0
XXX0000
B
MAAD...Y00
0
L-1
amino acid sequence
0
n
0
L-1
one-hot encoding

Keras

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_14 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_15 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

model.summary()

Keras




```
Using gpu device 1: GeForce GTX TITAN X (CNMeM is disabled, cuDNN 4007)
[INFO] downloading MNIST...
[INFO] compiling model...
[INFO] training...
Epoch 1/20
46900/46900 [==============================] - 3s - loss: 0.5510 - acc: 0.8474
Epoch 2/20
46900/46900 [==============================] - 3s - loss: 0.2177 - acc: 0.9361
Epoch 3/20
46900/46900 [==============================] - 3s - loss: 0.1615 - acc: 0.9527
Epoch 4/20
46900/46900 [==============================] - 3s - loss: 0.1299 - acc: 0.9620
Epoch 5/20
46900/46900 [==============================] - 3s - loss: 0.1100 - acc: 0.9681
Epoch 6/20
46900/46900 [==============================] - 3s - loss: 0.0964 - acc: 0.9721
Epoch 7/20
46900/46900 [==============================] - 3s - loss: 0.0860 - acc: 0.9741
Epoch 8/20
46900/46900 [==============================] - 3s - loss: 0.0773 - acc: 0.9765
Epoch 9/20
46900/46900 [==============================] - 3s - loss: 0.0711 - acc: 0.9786
Epoch 10/20
46900/46900 [==============================] - 3s - loss: 0.0654 - acc: 0.9797
Epoch 11/20
46900/46900 [==============================] - 3s - loss: 0.0605 - acc: 0.9819
Epoch 12/20
46900/46900 [==============================] - 3s - loss: 0.0568 - acc: 0.9830
Epoch 13/20
46900/46900 [==============================] - 3s - loss: 0.0535 - acc: 0.9839
Epoch 14/20
46900/46900 [==============================] - 3s - loss: 0.0508 - acc: 0.9844
Epoch 15/20
46900/46900 [==============================] - 3s - loss: 0.0481 - acc: 0.9857
Epoch 16/20
46900/46900 [==============================] - 3s - loss: 0.0449 - acc: 0.9864
Epoch 17/20
46900/46900 [==============================] - 3s - loss: 0.0424 - acc: 0.9875
Epoch 18/20
46900/46900 [==============================] - 3s - loss: 0.0409 - acc: 0.9878
Epoch 19/20
46900/46900 [==============================] - 3s - loss: 0.0386 - acc: 0.9887
Epoch 20/20
46900/46900 [==============================] - 3s - loss: 0.0369 - acc: 0.9890
[INFO] evaluating...
23100/23100 [==============================] - 0s
[INFO] accuracy: 98.49%
[INFO] dumping weights to file...
[INFO] Predicted: 6, Actual: 6
```

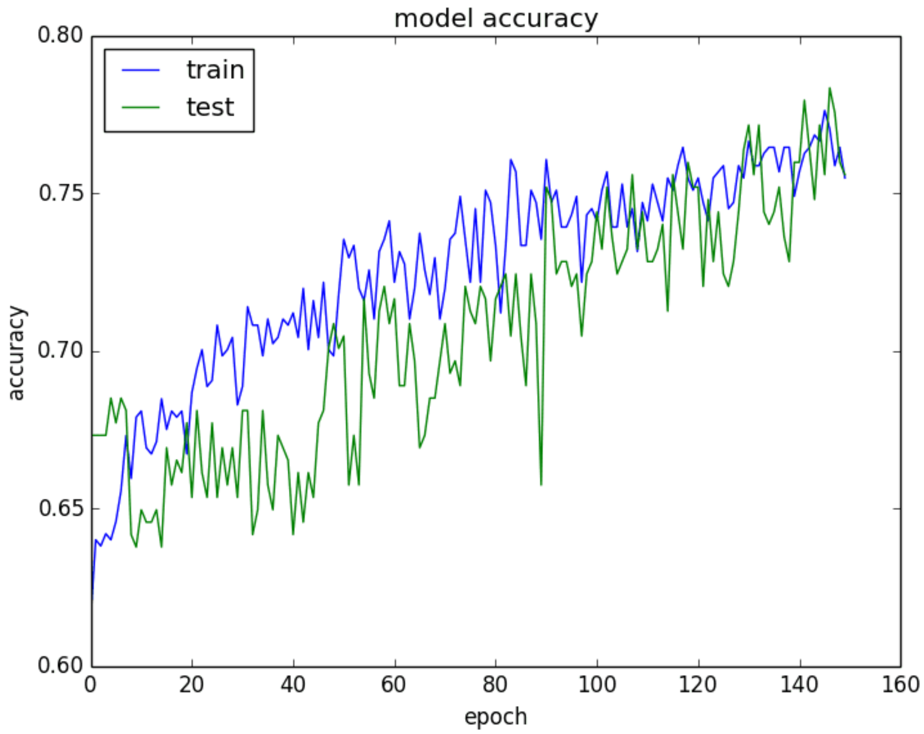
Keras

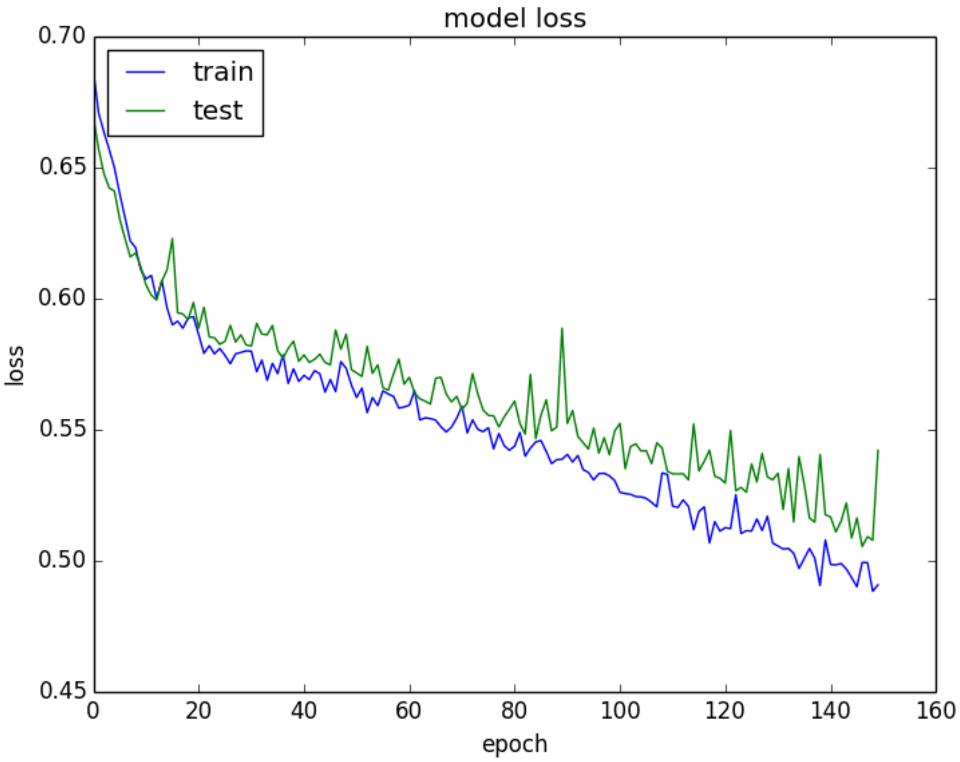
```
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accura
# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10,
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Keras

```
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import numpy
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accura
# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10,
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

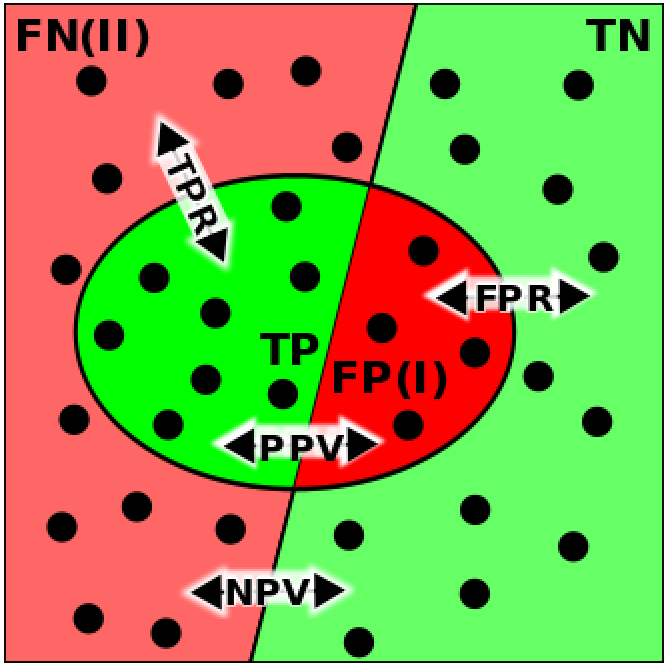
Keras






```
# Fit the model
history = model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10,
# list all data in history
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

Metrics



Actual \ Assigned	Test outcome *positive*	Test outcome *negative*
Condition positive	True *positive*	False *negative*
Condition negative	False *positive*	True *negative*

These can be arranged into a 2×2 contingency table

Metrics

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**Positive (P)**	**True positive (TP), hit**	**False negative (FN), type II error, miss, underestimation**
	Negative (N)	**False positive (FP), type I error, false alarm, overestimation**	**True negative (TN), correct rejection**

Metrics

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}} = 1 - \mathrm{FPR}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = 1 - \mathrm{FDR}$$

negative predictive value (NPV)

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FN}} = 1 - \mathrm{FOR}$$

miss rate or false negative rate (FNR)

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN}+\mathrm{TP}} = 1 - \mathrm{TPR}$$

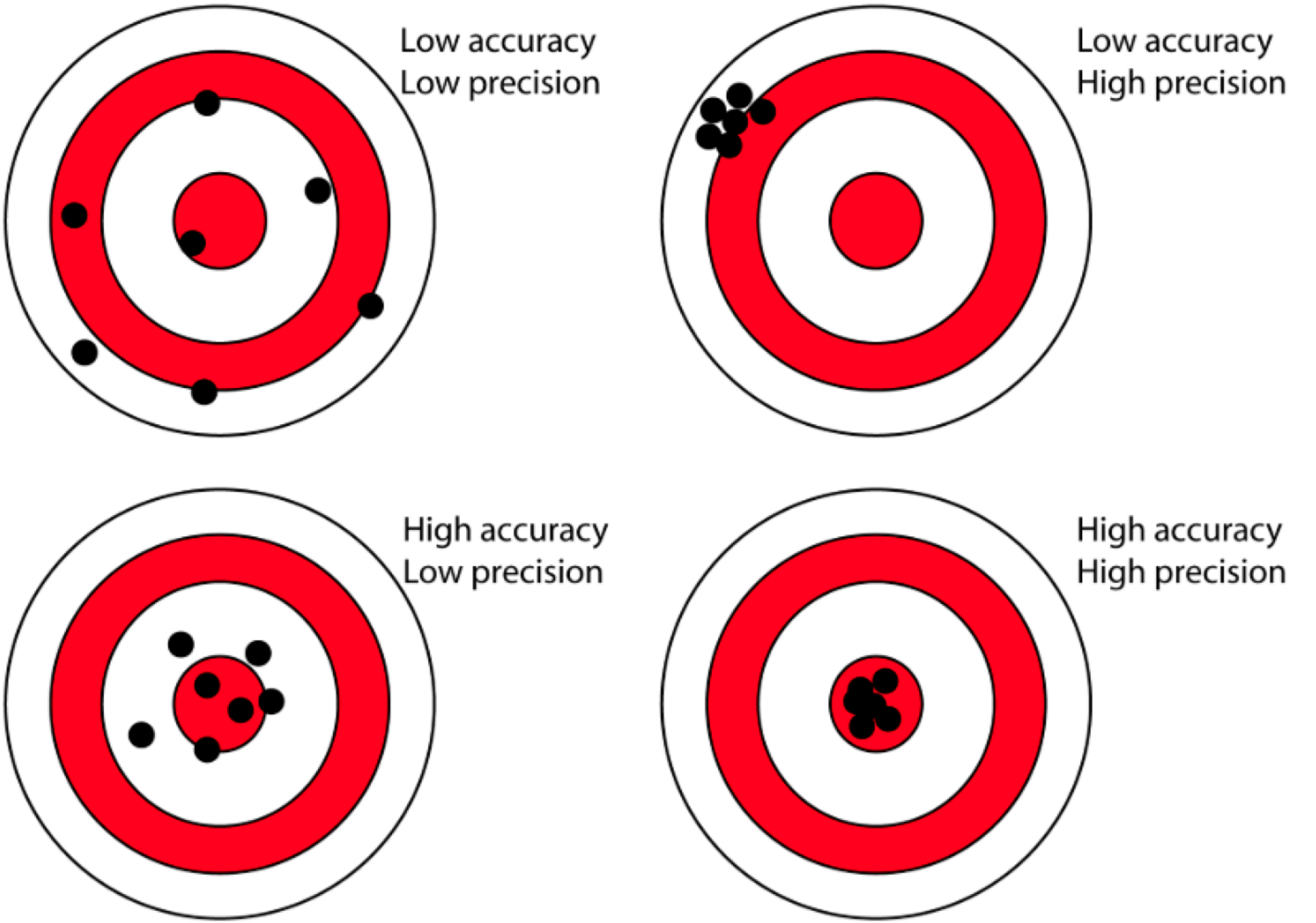
fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}$$

false discovery rate (FDR)

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TP}} = 1 - \mathrm{PPV}$$

Metrics


Low accuracy
Low precision
Low accuracy
High precision
High accuracy
Low precision
High accuracy
High precision

Metrics

accuracy (ACC)

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

balanced accuracy (BA)

$$\mathrm{BA} = \frac{TPR + TNR}{2}$$

F1 score

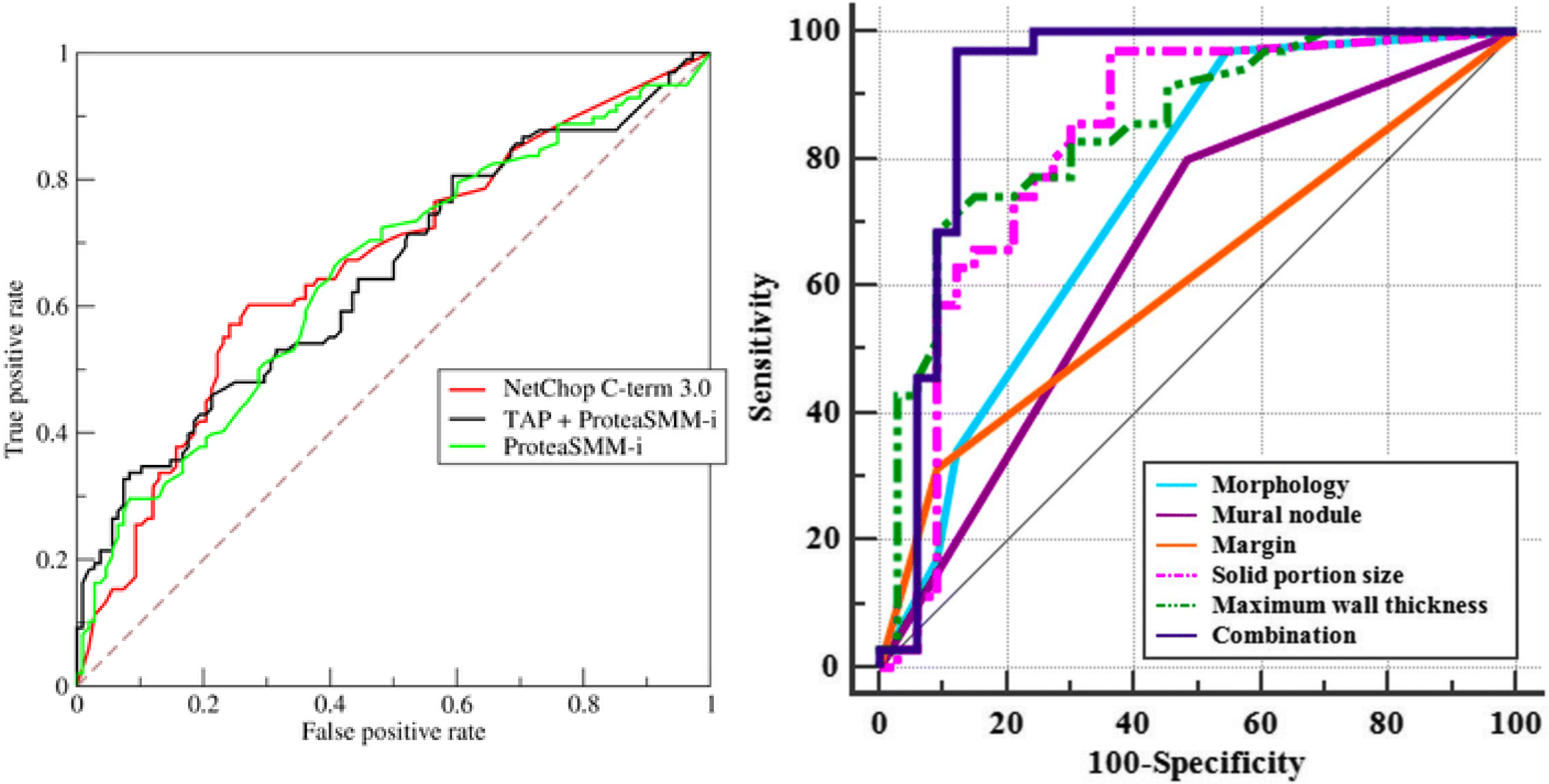
is the harmonic mean of precision and sensitivity:

$$\mathrm{F}_1 = 2 \times \frac{\mathrm{PPV} \times \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

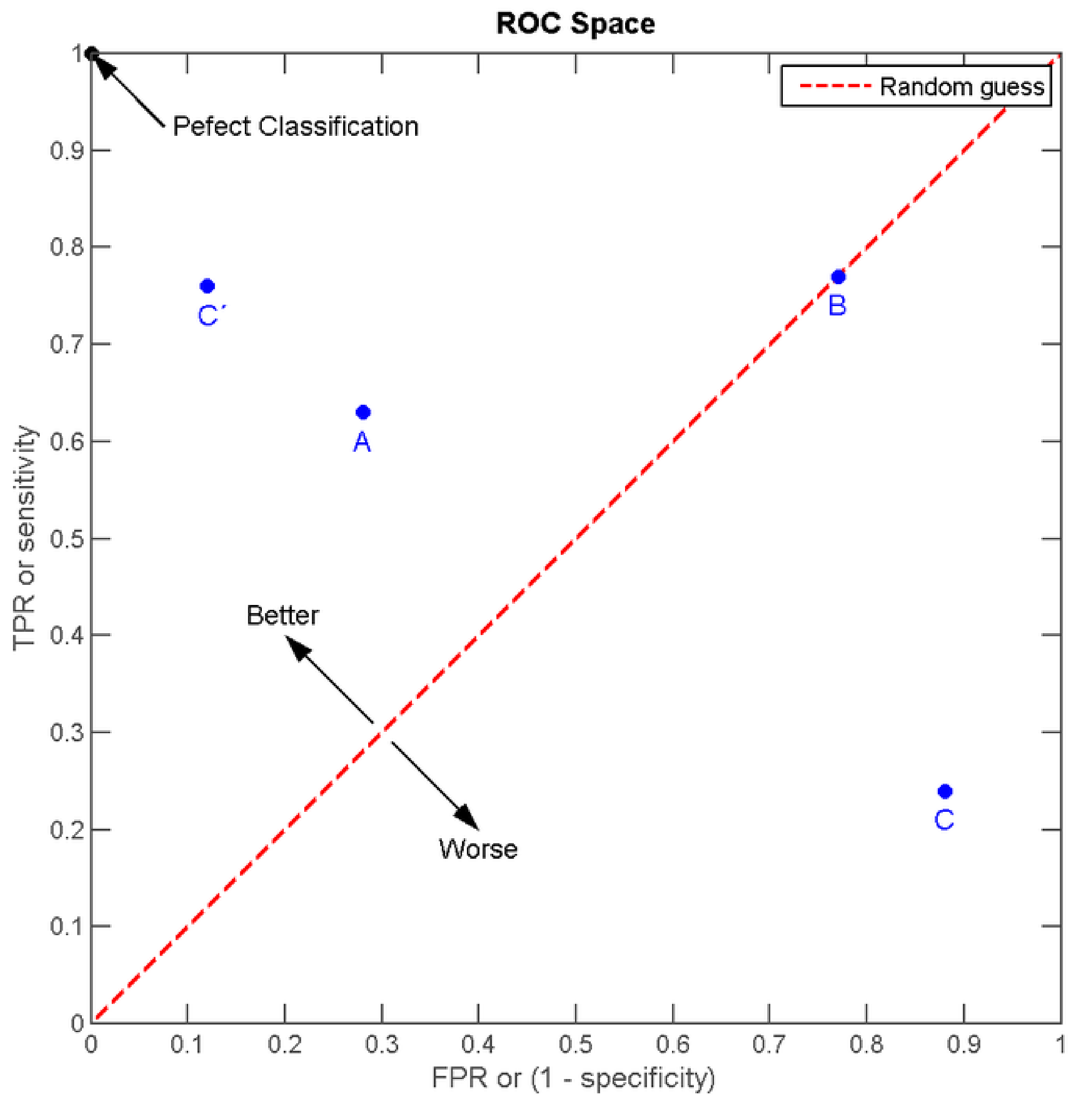
phi coefficient (φ or r_{φ}) or Matthews correlation coefficient (MCC)

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}$$

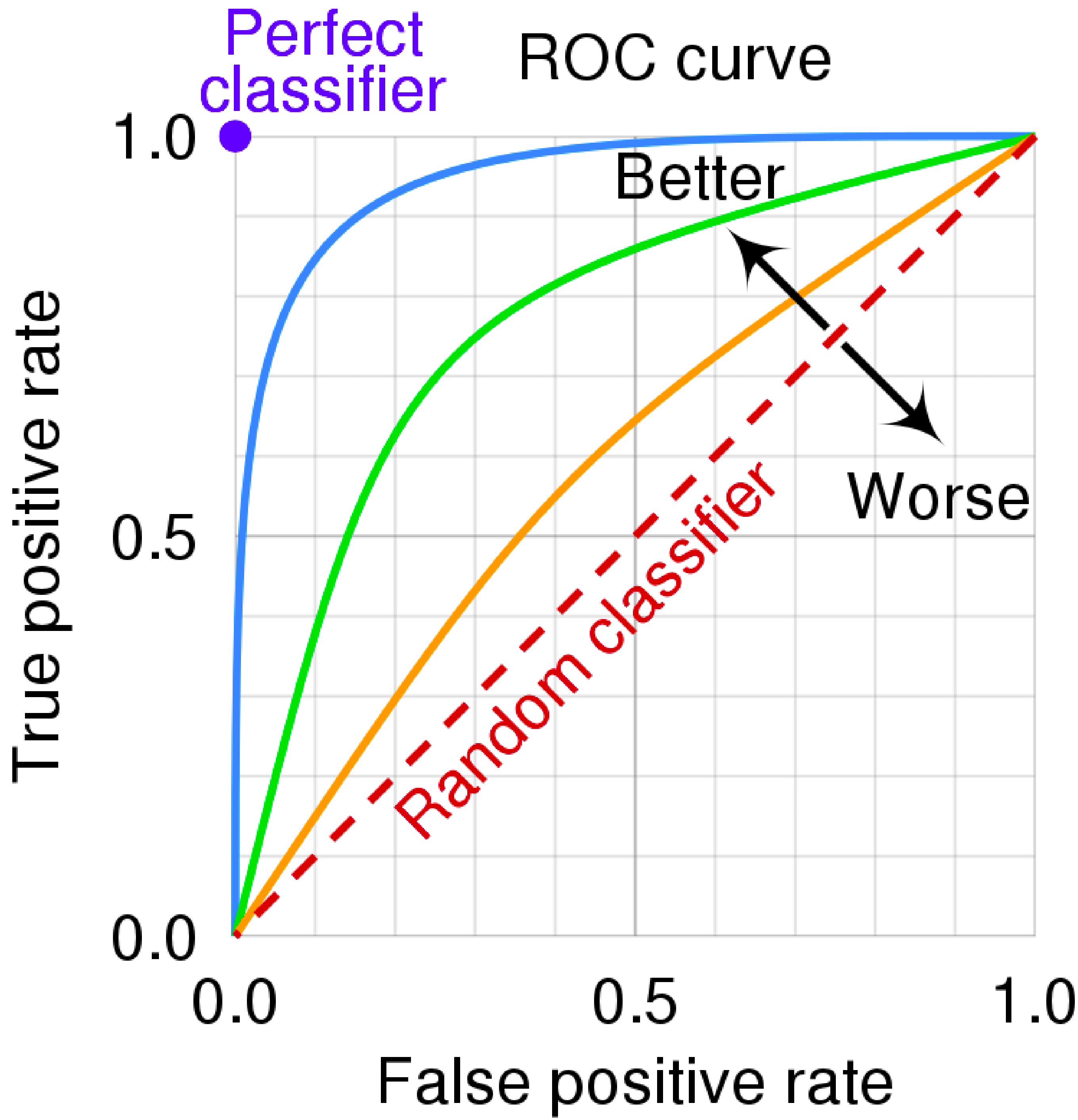
Receiver operating characteristic



Receiver operating characteristic



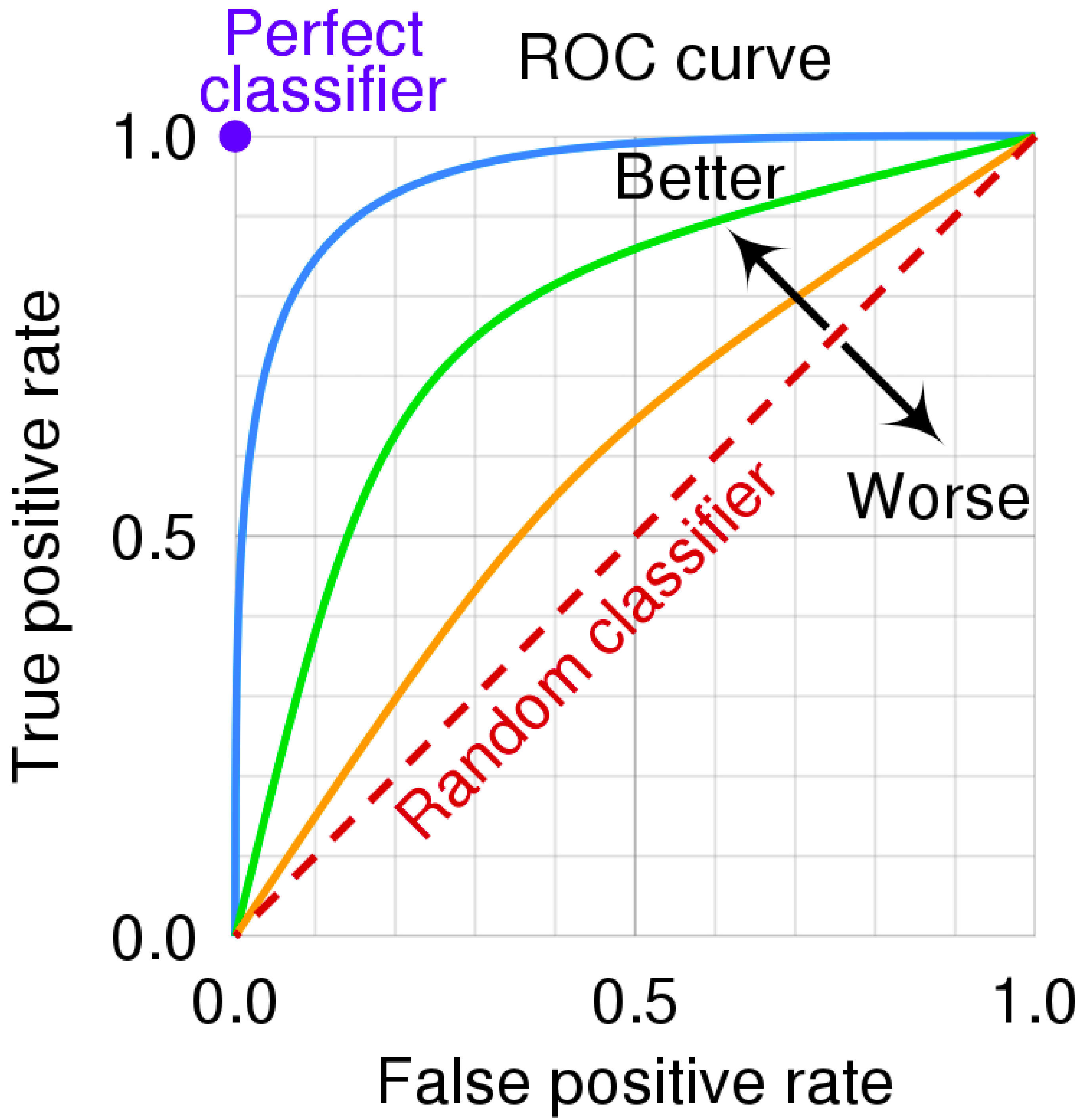
Receiver operating characteristic



Receiver operating characteristic

```
import sklearn.metrics

fpr, tpr, thresholds = sklearn.metrics.roc_curve(y_true = true_labels,
y_score = pred_probs, pos_label = 1) #positive class is 1; negative
class is 0
auroc = sklearn.metrics.auc(fpr, tpr)
```

Receiver operating characteristic



Thank you for your time
and
See you at the next lecture

Any other
questions & comments

l.kozlowski@mimuw.edu.pl

www: bioinformatic.netmark.pl