Data Mining and Knowledge Discovery

Petra Kralj Novak

January 13, 2020

http://kt.ijs.si/petra_kralj/dmtm2.html

Neural networks

Neuron, perceptron
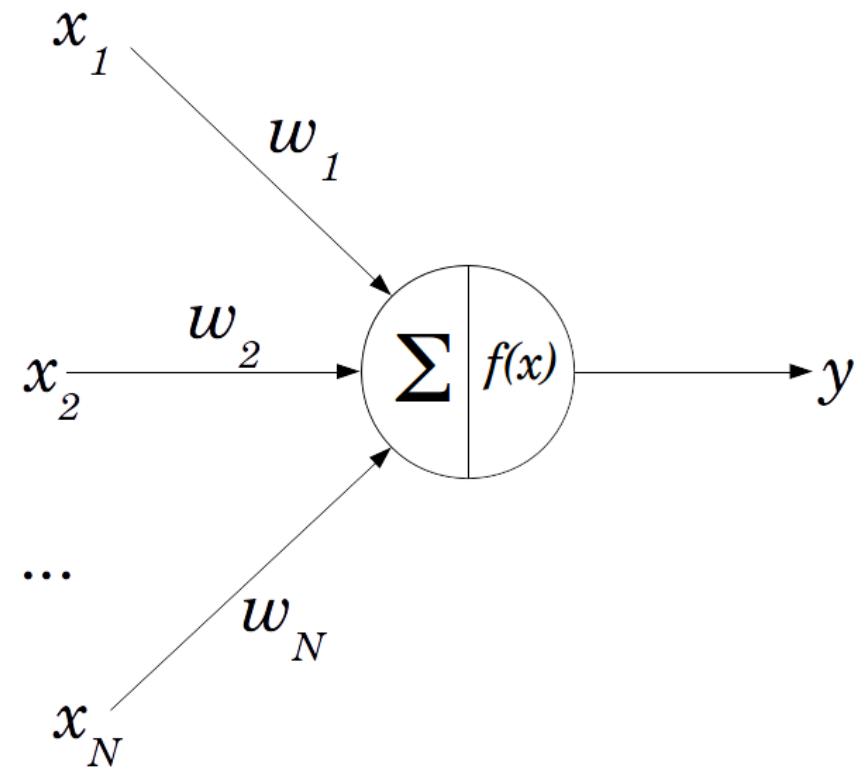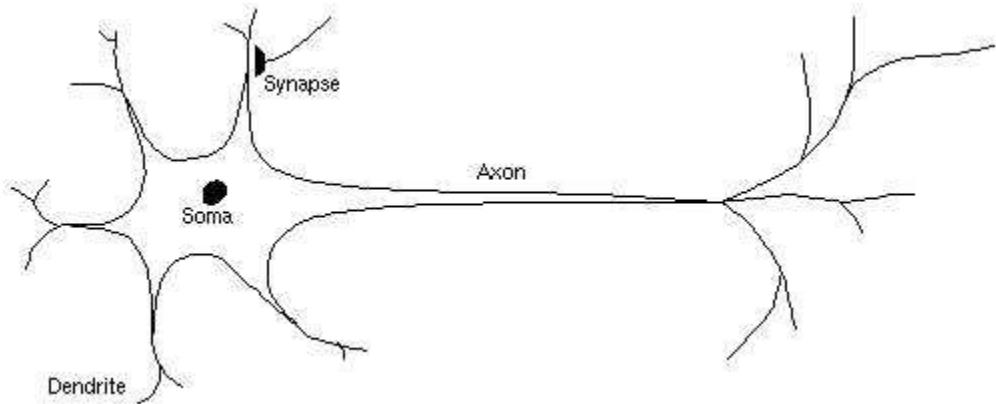
The **perceptron** is a mathematical model of a biological neuron

x

A

y

B

$\Sigma \mid f(x)$

class

C

$f(x) =$
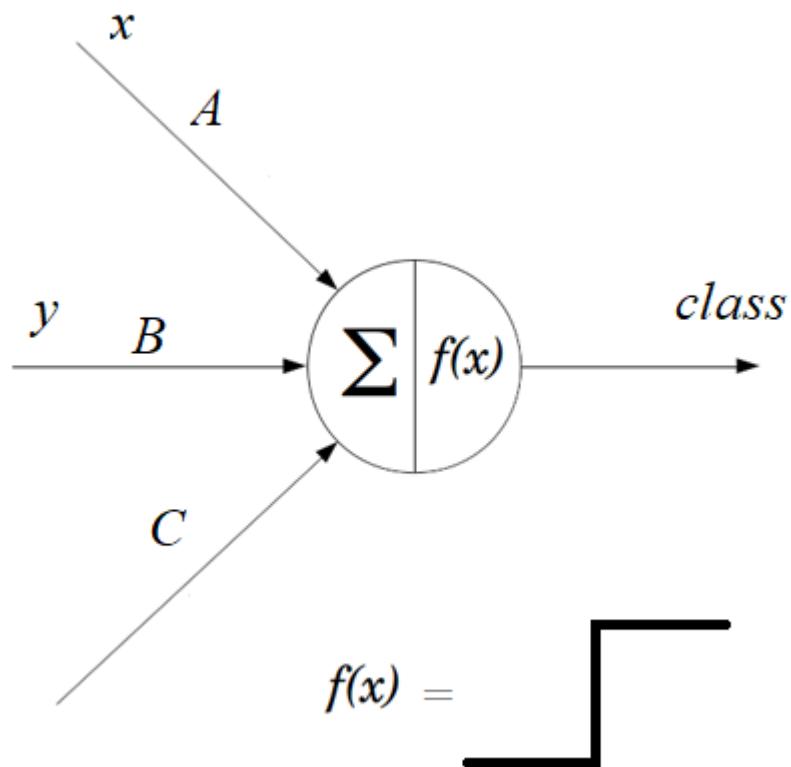
- A single perceptron can separate linearly.

Output of P = $\begin{cases} 1 \text{ if } A\,x + B\,y > C \\ 0 \text{ if } A\,x + B\,y < = C \end{cases}$

Neural network
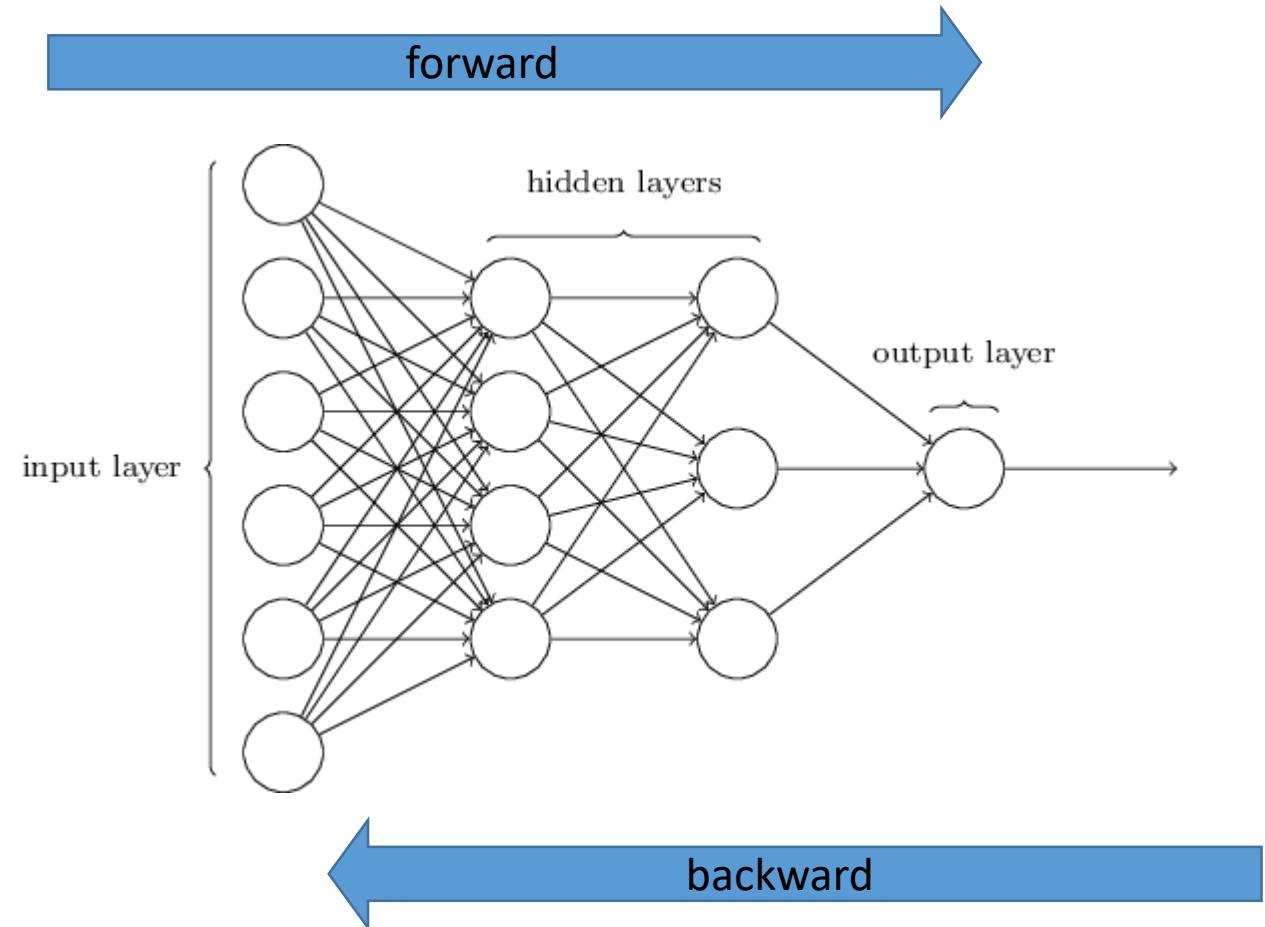
What is backpropagation really doing?

Deep learning, chapter 3
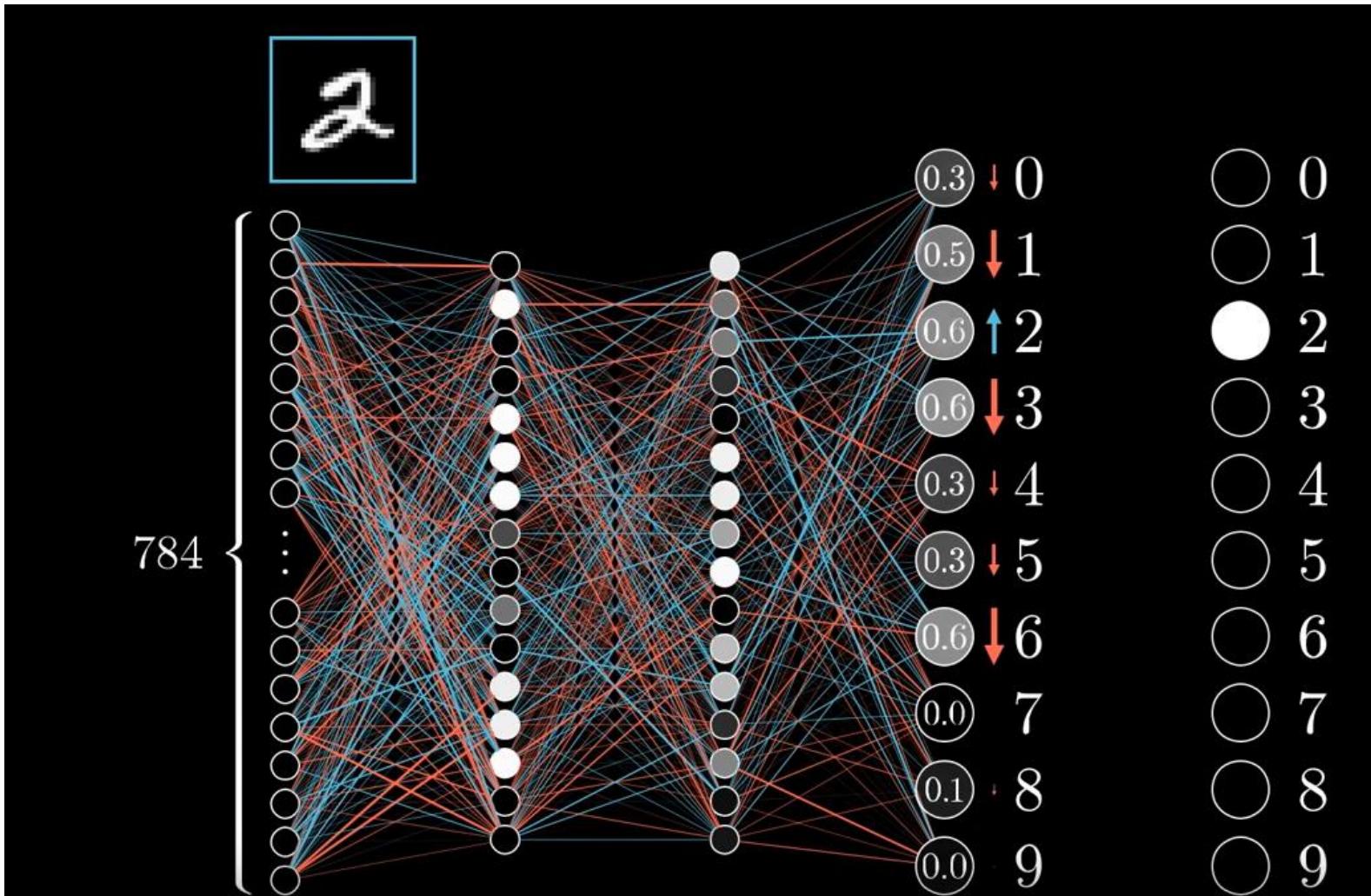
Predictive model

- Architecture
  - Define
  - Compile
- Train (fit)
  - Forward
  - Backward
  - Optimize
- Predict (evaluate)
 - Forward

Train

- **Forward propagation** (check performance)
  - **loss function** is an error metric between actual and predicted
  - absolute error, sum of squared errors
- **Backpropagation** (direction of parameter/weight change)
  - how much the total error will change if we change the internal weight of the neural network with a certain small value **Δw (gradient)**
  - backpropagate the errors using the derivatives of these functions: auto-differentiation
- **Optimization** (change weights based on learning rate, gradient descent)
  - New weight = old weight — Derivative Rate * learning rate
  - **Batch size** is a hyperparameter that controls the number of training samples to work through before the model's internal parameters are updated.
  - The number of **epochs** is a hyperparameter that controls the number of complete passes through the training dataset.

Neural networks and back-propagation explained in a simple way

Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras

https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

Keras: The Python Deep Learning library

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

- Google's Tensorflow: is a low-level framework that can be used with Python and C++.

  - Install packages: tensorflow, keras

MINST – handwritten digits

- Each image is a 28 by 28 pixel square (784 pixels total).
- Normalized in size and centered
- A standard spit of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it.

From the MINST Database of Hand-written Digits

Exercise

- Load the MNIST dataset in Keras.
- Train and evaluate a **baseline neural network** model for the MNIST problem.
- Train and evaluate a simple **Convolutional Neural Network** for MNIST.
- Implement a **close to state-of-the-art deep learning** model for MNIST.

Load the data: 9_neural_nets-0-load_data.py

```python
from keras.datasets import mnist
import matplotlib.pyplot as plt

# Plot ad hoc mnist instances

(X_train, y_train), (X_test, y_test) = mnist.load_data()   # Dataset of 60,000 28x28
grayscale images of the 10 digits, along with a test set of 10,000 images.
# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
# show the plot
plt.show()
```

Prepare data: 9_neural_nets-1-perceptron.py

```python
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

# train-validation split
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.1, random_state=42)


# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_validation = X_validation / 255
X_test = X_test / 255

# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_validation = np_utils.to_categorical(y_validation)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

One-hot Encoding
for Multi-label and multi-target prediction

```
# one-hot encoding class labels

from keras.utils import np_utils

y_train[:10]
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

```
y_train_OneHotEncoding = np_utils.to_categorical(y_train)
y_train_OneHotEncoding[:10]
```

```
array([[ 0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.],
       [ 1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   1.],
       [ 0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
       [ 0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.]])
```

```
   0    1    2    3    4    5    6    7    8    9
```

# Define + compile, fit, predict: 9_neural_nets-1-perceptron.py

```python
# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal', activation='relu
    model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model


# build the model
model = baseline_model()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=10, batch_size=200

# Final evaluation of the model
print("Final evaluation of the model")
scores = model.evaluate(X_test, y_test, verbose=1)
print("Baseline Error: %.2f%%" % (100 - scores[1] * 100))
```

Activation functions

- relu(x) = max(0, x)
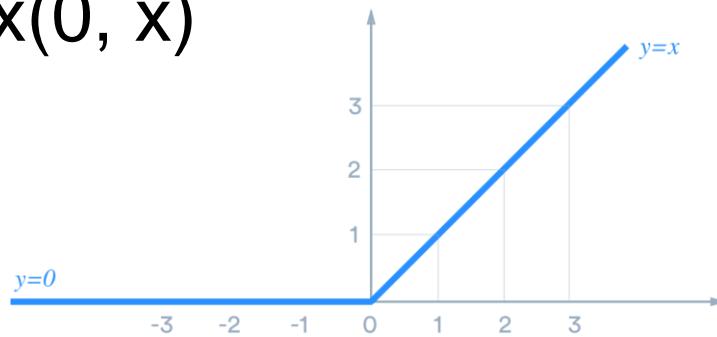
- Softmax
  - After applying softmax, each component will be in the interval [0,1], and the components will add up to 1
  - The softmax function is frequently used as the final activation function in neural networks for classification problems.
  - Maps the non-normalized output of a network to a probability distribution over predicted output classes.

Loss function: categorical_crossentropy

- Multi-class classification tasks
- Must be combined with Softmax

$$L(y, \hat{y}) = -\sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} * log(\hat{y}_{ij}))$$

- $\hat{y}_{ij}$ is the predicted value
- y_{ij} is the actual (correct) value

Architecture

- Layers: type, inicialization, regularization
    - Dense
    - Convolutional
    - Pooling
    - Dropout – for regularization
    - Recurrent
    - Embedding
- Activation functions
    - relu
    - softmax (output layer)
- Loss function
    - Classification
        - **categorical_crossentropy**, categorical_hinge, sparse_categorical_crossentropy, binary_crossentropy, …
    - Numeric prediction
        - **mean_squared_error**, mean_absolute_error, mean_absolute_percentage_error, mean_squared_logarithmic_error, cosine_proximity, …

- **Model.compile**

Types of layers (1)

Dense
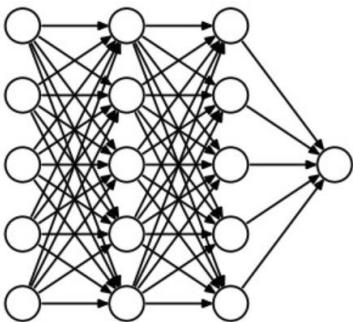
Fully connected.

Dropout
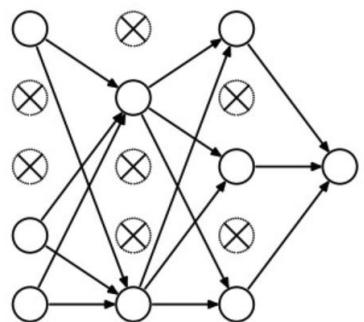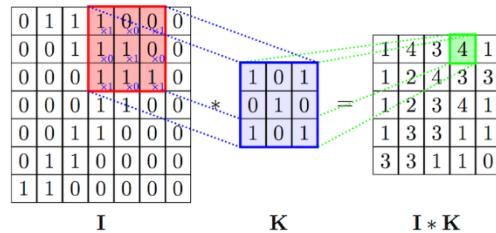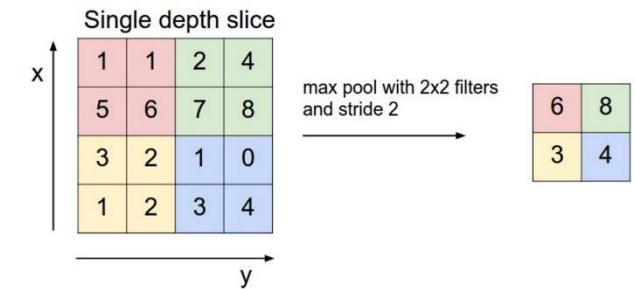
During training, some neurons on a particular layer will be deactivated. This improves generalization because it forces the layer to learn with different neurons the same "concept".

Convolutional

The convolution layer comprises of a set of independent filters. Each filter is independently convolved with the image.

Example: link

Pooling

A max-pooling layer takes the maximum of features over small blocks of a previous layer.

Edge detection example
https://youtu.be/puxHUGpuOVw

Types of layers (2)

Flatten

matrix

vector

Fully connected.

LAYERS

About Keras layers

Core Layers

Convolutional Layers

Pooling Layers

Locally-connected Layers

Recurrent Layers

Embedding Layers

Merge Layers

Advanced Activations Layers

Normalization Layers

Noise layers

Layer wrappers

Writing your own Keras layers
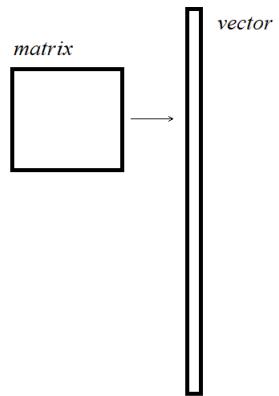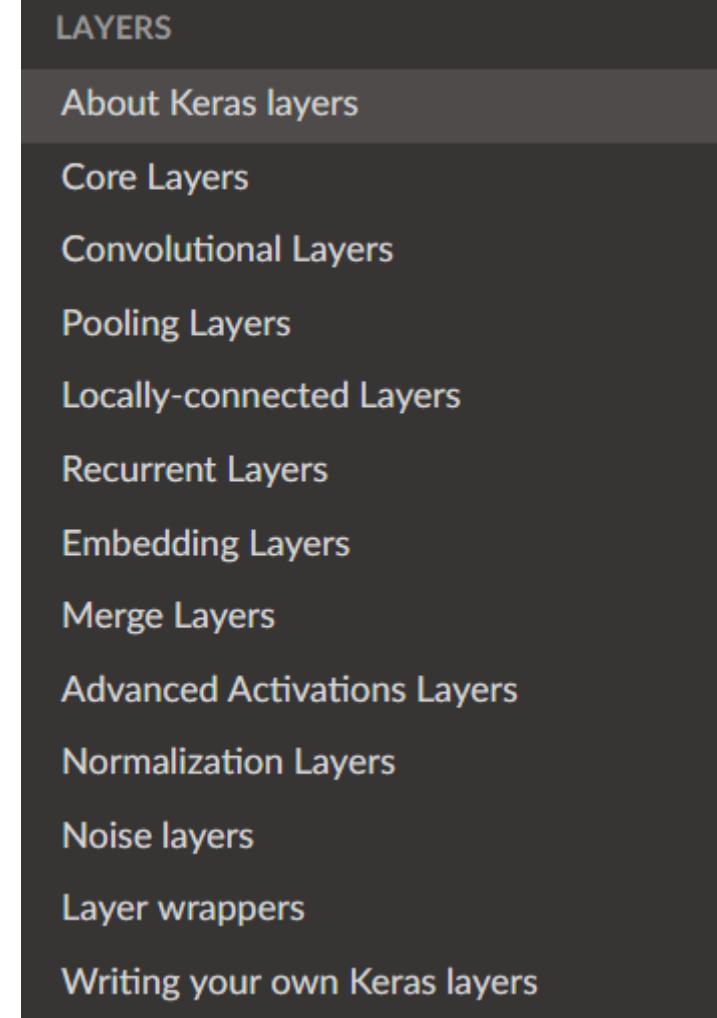
Convolutional model

```python
def baseline_model():
    # create model
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```