Focus on supervised Deep Learning to classify images of waste in the wild

From Machine Learning to Deep Learning: a concise introduction

Dr. Khatuna Kakhiani

26.03.- 28.03.2024, HLRS, Universität Stuttgart

Agenda - March 27, 2024

9:00 - 17:30 CEST

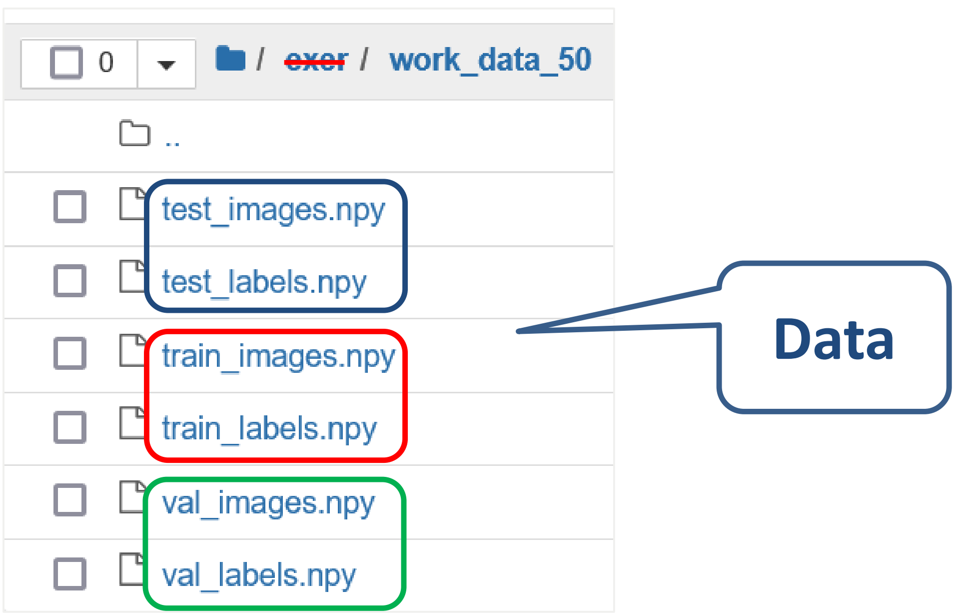
- A Brief Introduction to Deep Learning
  - Tutorial 1: Image exploration with Jupyter Notebook
- Data Processing
  - Tutorial 2: Dataset exploration and preprocessing
- Waste Image Classification - Neural Networks
  - Tutorial 3: Sigmoid Neural Model; Simple Neural Networks
  - Tutorial 4: Image Classification with Multilayer Perceptron (MLP)
- Convolutional Neural Networks (CNN)
  - Tutorial 5: Image Classification with CNN

Lunch break: 13:00 – 14: 00 PM; 15 min Breaks @ 11:00 AM & 15:45 PM

Input

Training set	Validation set	Testing set
• Model is trained • ~ 80% of the dataset	• Model is assessed • ~ 10% of the dataset	• Model is tested • ~ 10% of the dataset

The ground truth: train, validation & test **label** sets



Learning a model

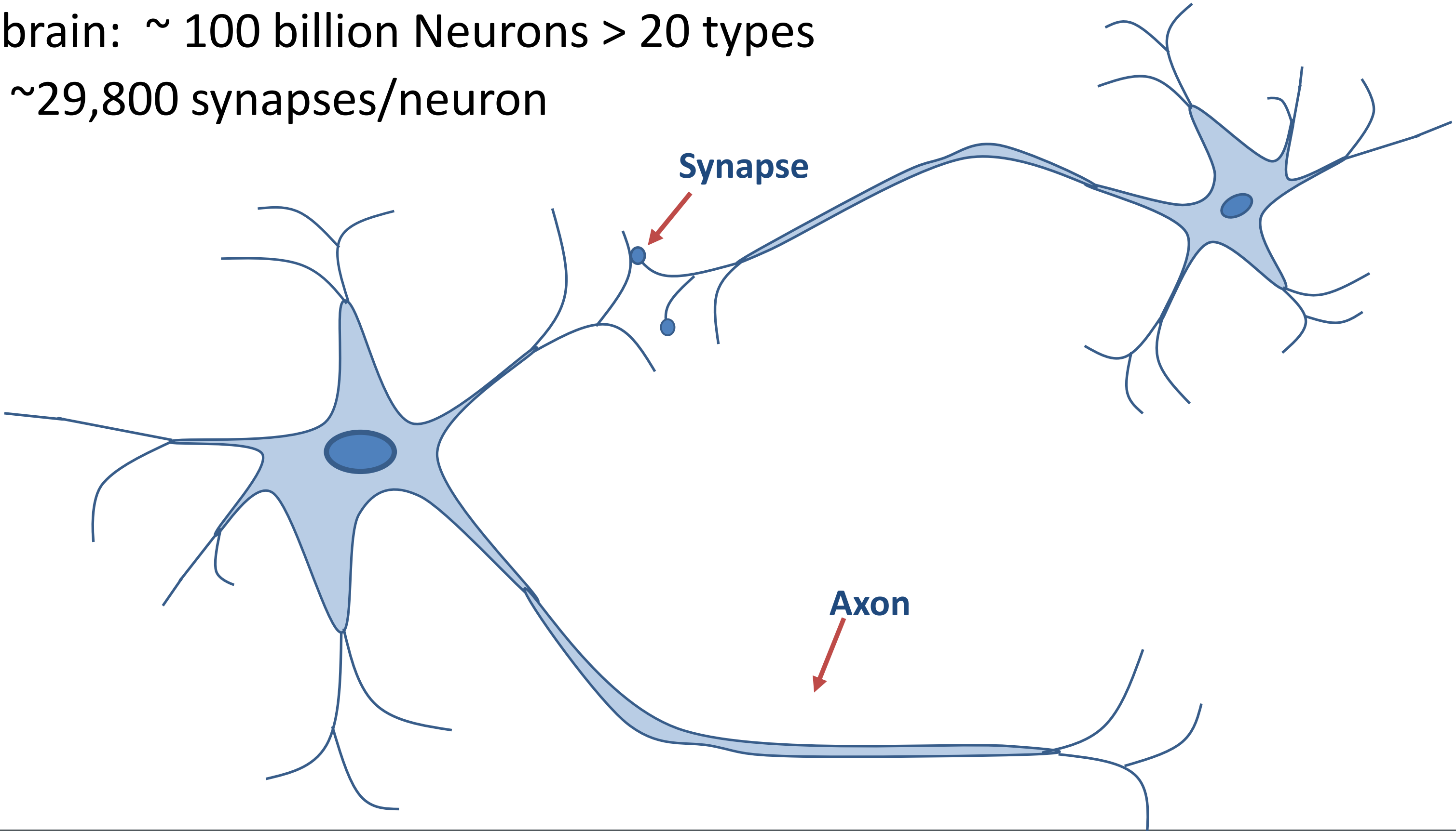
Configure the Model Architecture	Compile the model	Train the Model
• Artificial Neural Networks (ANN) • Multilayer Perceptron (MLP) • Convolutional Neural Networks (CNN)	• Loss (to measures how accurate the model is during training) • Optimizer (to minimize Loss with respect of parameters) • Metrics (to evaluate performance)	• Performance at Task improves with an Experience • Train to classify images • Track epochs, let model see every pictures many times; babysit process

Neural Networks

Brain Neurons

Neurons are excitable cells which chemically transmit electrical signals through connections called synapses.

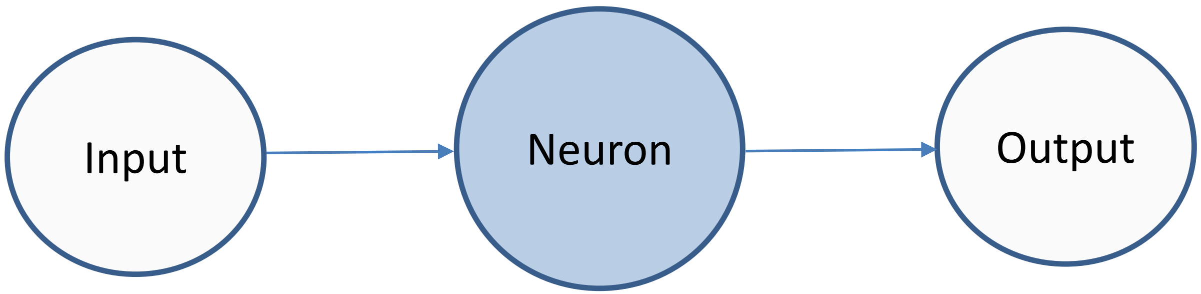
- Human brain: ~ 100 billion Neurons > 20 types
- Cortex: ~29,800 synapses/neuron



Artificial Neuron

A very coarse model of a **biological neuron**

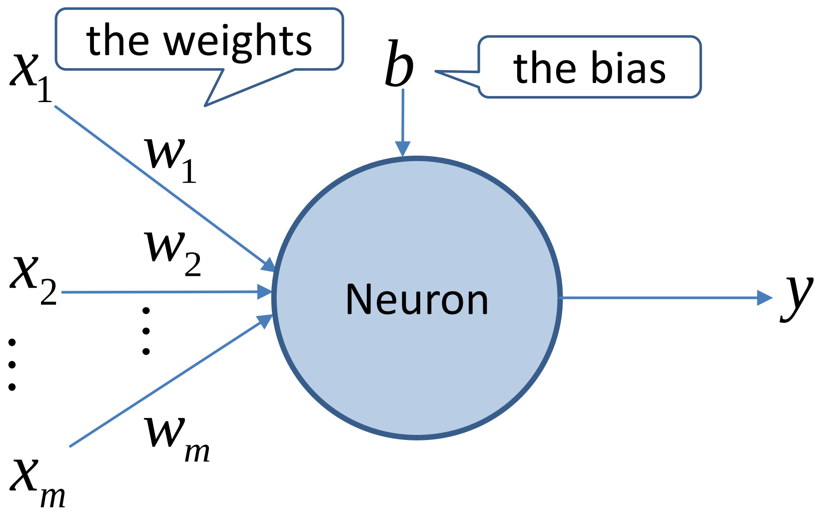
- The smallest unit of a neural network: **a single neuron**



- Such a neuron can handle input with several values, where each values can be weighted differently
- A neuron has the functionality of a **logistic regression**

Artificial Neuron

- Input of a Neuron $z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_m \cdot w_m + b$

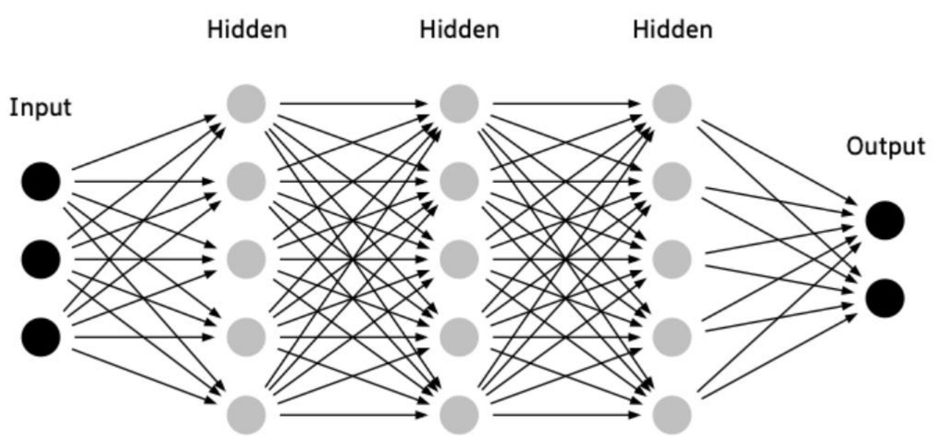


- The affine transformation, a linear transformation of input features via weighted sum, combined with a translation via the added bias.
- Neuron calculates output if we apply **activation function** f on an input function z : $y = f(z)$.
- Combining & connecting of many neurons → **Neural Network**

Feedforward Neural Networks

In Feedforward Neural Networks(FNN)/Multilayer Perceptrons (MLPs):

- **set of neurons** make one **layer;** interlayer nodes - fully connected;
- **transform an input** through a series of **hidden layers**
- every input influences every neuron in the hidden layer, and each of these → every neuron in the output layer
- output layer represents the class scores (i.e., in classification)



MLP with 3 inputs, 3 hidden layers of 5 neurons (nodes) each, and 1 output layer.

Feedforward Neural Networks

Examples of usage:

- Convolutional NNs (object recognition from photos)
- Recurrent NNs (in many natural language applications)

Feedforward Neural Networks

- The goal in FNN is to approximate some function f^*.
- For a classifier $y = f^*(x)$ maps an input x to a category y.
- A FNN defines a mapping $y = f(x;\theta)$, learns the value of the parameters θ that result in the best function approximation.
- **Networks is represented by many different functions** (i.e., 3 here) **connected in a chain** to form:

$$f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right)$$

- With $f^{(1)}$, $f^{(2)}$, $f^{(3)}$ being the first, second & third network layers respectively.
- During neural network training we drive $f(x)$ to match $f^*(x)$.

Weights & Biases

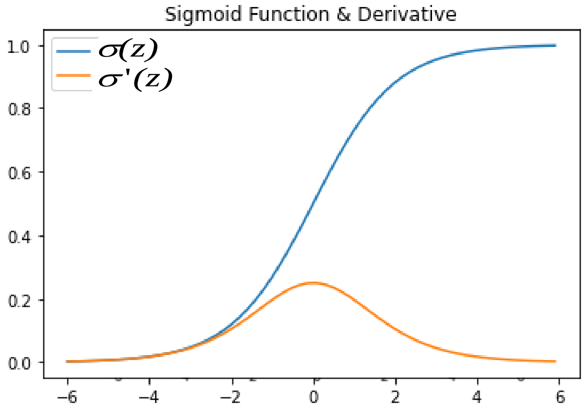
Feedforward Neural Networks

- Let start with linear models and their limitations. *
- Linear models, i.e., logistic regression and linear regression
  - can be fit efficiently either in closed form or with convex optimization;
  - are limited to linear functions, no understanding about the interaction between any two input variables;
  - If $f^{(1)}$ were linear:
    - the FNN as a whole would remain a linear function;
    - stacking of neurons in network would be useless;
    - its derivative with respect to x will be constant; constant gradient …
- To extend linear models to represent nonlinear functions of x apply the linear model not to x but to a transformed input $\sigma(x)$, where σ is a nonlinear transformation.

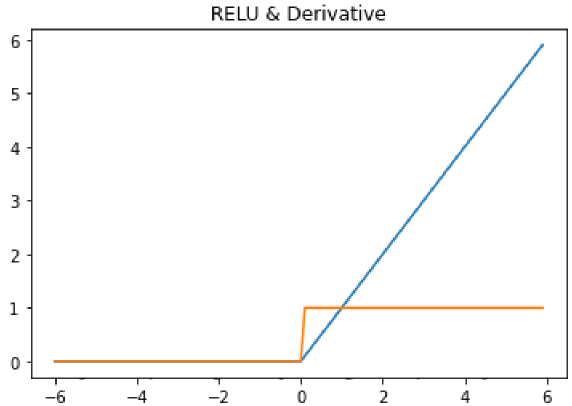
* Goodfellow et. al., Deep Learning, MIT Press, 2016.

Activation functions

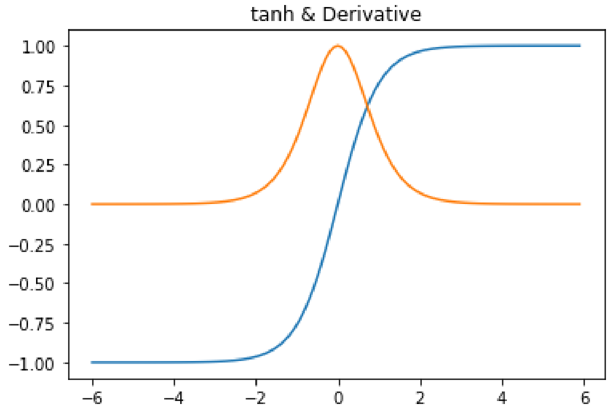
- Introduces non-linearity to Neural Network
- **Non-linear transformation of input** to allow complex tasks



Sigmoid



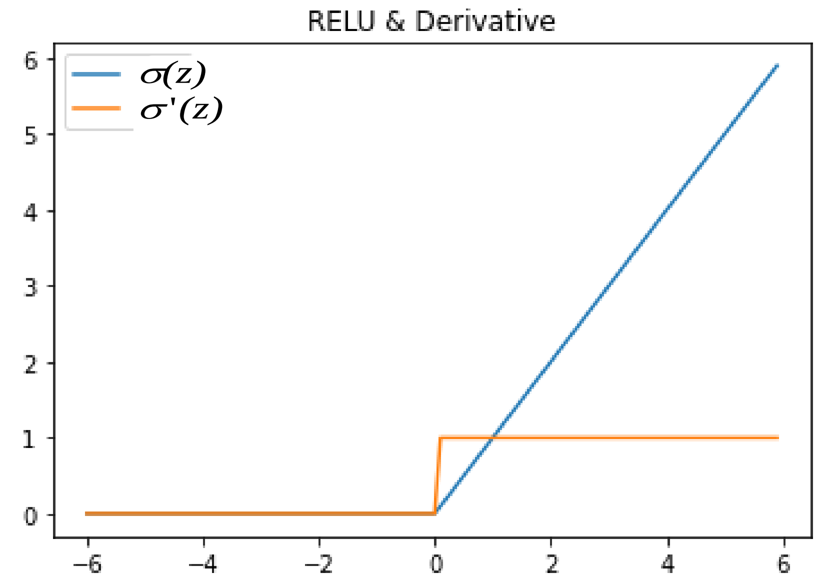
Rectified linear unit (ReLU)



Hyperbolic tangent (tanh)

Activation functions - ReLU

- Rectified linear unit activation function
- Fast convergence (sparse activations)
- Constant values
- Negative values do not get activated
- For **CNN ReLU performs faster ***



$$\sigma(z) = \max\{0, z\}$$

Problem:

- Dying ReLU: neurons get stuck at 0
- Can lead to model not learning

Solution: Leaky ReLU w/ small slope for negatives

[* 1906.01975.pdf (arxiv.org)]

Softmax

– used as the output of a classifier (last output layer)

– to represent the probability distribution over **n** different classes

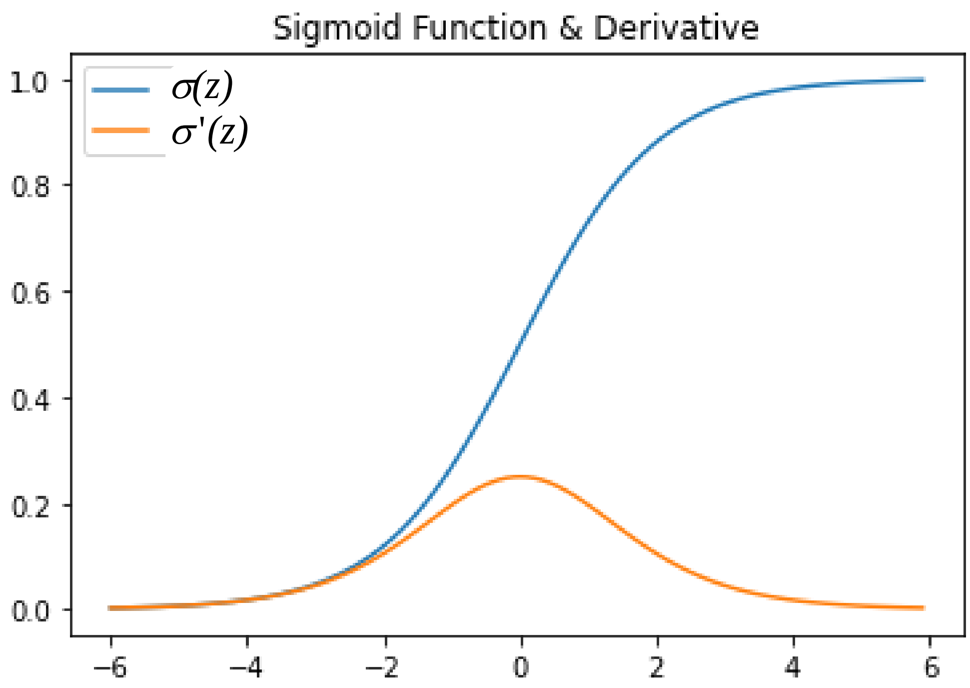
– receives vector as an input and returns a normalized probability distribution of a list of outcomes

```
In [11]: def softmax(x):      # x is vector
             return (np.exp(x))/sum(np.exp(x))

         x = np.array([1, 0.3, 3, 0.5])
         prob = softmax(x)       # converts list of numbers to a list of probabilities
         print(prob)             # output - probabilities
         print(sum(prob))        # sum of the probabilities gives 1
```

```
[0.10534997 0.05231524 0.77843681 0.06389798]
1.0
```

Activation functions - Sigmoid

- Squashes weighted sum of neurons (real numbers) into range (0,1)
- Problem: vanishing gradient (smaller) and sparsity (dense neurons)
- Solution: ReLU (constant value and sparse activations)

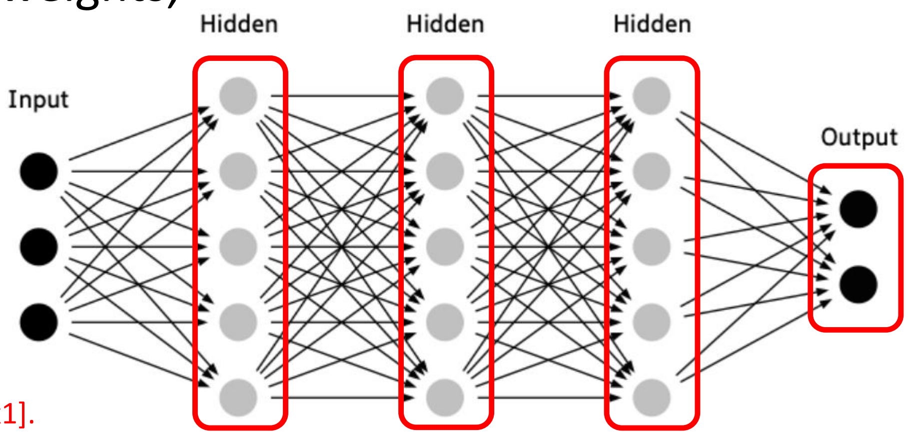


$$\sigma(z) = \frac{1}{1 + e^{(-z)}}$$

Feedforward Neural Networks

Given **MLP** with 3 inputs, 3 hidden layers of 5 neurons each, and 1 output layer.

- 5 + 5 + 5 + 2 = 17 neurons (not counting the inputs),
- [3 x 5] + [5 x 5] + [5 x 5] + [5 x 2] = 75 weights,
- 5 + 5 + 5 + 2 = 17 biases.
- A total of 92 learnable parameters:
  75 + 17 = 92



For our tutorial example:

x input column vector containing all pixel data of the image [2500x1].

Neurons most commonly do not have an activation function (or you can think of them as having a linear identity activation function).

MLP

$x_i \in \mathrm{R}^D;$

here i = 1…m

m images, each with D = 50 x 50 x 1 px

$y_i \in 1 \ldots n$; here n = 2;

$x\left[2500 \times 1\right];\ \mathbf{W}\left[2 \times 2500\right]; b\left[2 \times 1\right]$

$f : \mathrm{R}^D \mapsto \mathrm{R}^n$

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 2500)              0
_________________________________________________________________
dense (Dense)                (None, 128)               320128
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 320,386
Trainable params: 320,386
Non-trainable params: 0
```

Forward Propagation

- Given the weights W, biases B, and the activation of the input layer $x = x_0$, the **output** $f(x, \mathbf{W}, \mathbf{B})$ of the **neural network** can be computed using forward propagation, matrix multiplication followed by a bias offset and an activation function.

$$z_l = \mathbf{W}_{l-1}\ x_{l-1} + \mathrm{B}_{l-1}$$
$$x_l = \sigma(z_l)$$
$$f(x, \mathbf{W}, \mathrm{B}) = x_{L+1} = \mathbf{W}_L\ x_L + \mathrm{B}_L$$

- For MLP with *L* hidden leyers, each with h_l neurons. z_l *and* x_l denote the input and activation of all neurons in layer l.

- x_{l-1} and B_{l-1} are vectors of size h_l and $\mathbf{W}_{l-1}$ is a matrix of size $h_l \times h_l$.

Backpropagation

- The algorithm is used to effectively train a NN through a **chain rule** method**.**
- After each **forward pass** through a network, backpropagation performs a **backward pass** while **adjusting** the model's parameters (**weights and biases**) given through the **error**.
- **The gradient descent** algorithm is used to **optimize** (min/max) some **function.**
- By **moving** in the **opposite direction of the slope**, given by **derivative** of this function, we can **improve this function**.
- The **speed** of movement down the gradient is controlled by the **learning rate**, which can be adjusted.
- A higher learning rate might miss the global minimum (optimum), while a low learning rate may get stuck on a local minimum.

Baydin et al., Automatic Differentiation in Machine Learning: a Survey, 2018
Mathieu et al., Fast Training of Convolutional Networks through FFTs, 2014

Backpropagation

Part1 tutorial

Backpropagation:

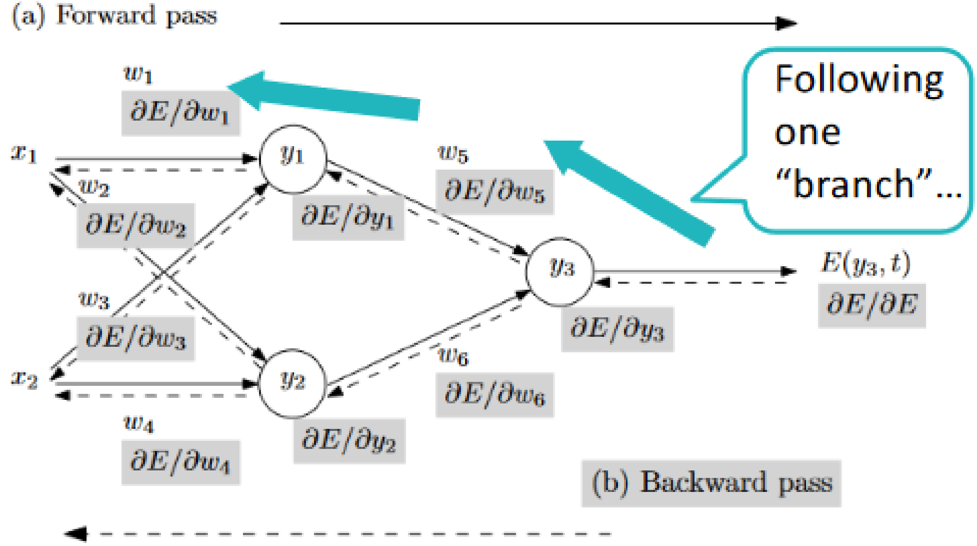
- Training input x_i are fed forward, generating corresponding activations y_i.
- E is the error between the final output (y_3) and the target ($\widehat{y_3}$, in the paper: t), same as the loss function.
- Through the chain rule:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial w_5}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_1} =$$

$$= \left(\frac{\partial E}{\partial y_3}\frac{\partial y_3}{\partial y_1}\right)\frac{\partial y_1}{\partial w_1}$$

...



Baydin et al., Automatic Differentiation in Machine Learning: a Survey, 2018
Mathieu et al., Fast Training of Convolutional Networks through FFTs, 2014

Deep Learning
Tutorial

Tutorial 3 - Simple Neural Network

- **Hands-on:** Simple Neural Networks with Sigmoid
- **Outcome:** Basic understanding of how neuron works with non-linear activation function, feedforward and backpropagation, parameters update.

Tutorial 3

Jupyter notebook:

- notebooks/NB-3-1_sigmoid-N-Model.ipynb
- notebooks/NB-3-2_simple_NN.ipynb (optional)

10 min

Tasks to complete in NB-3-1_sigmoid-neuron.ipynb:

- Define sigmoid activation function & its derivative; visualize
- Initialize parameters
- Define the input data and the ground truth; perform feedforward calculation; calculate the error, how far are we? adjust the weights accordingly to minimize error
- Get familiar with other activation functions: ReLU etc.
- Experiment with the Softmax function

Tutorial 3

NB-3-1_sigmoid-N-Model.ipynb

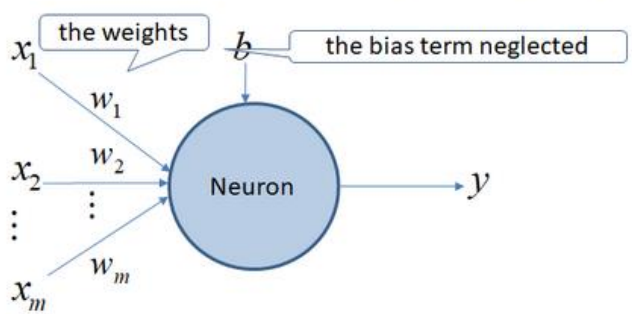
```
In [ ]: # We will use Sigmoid function for a simple demonstration of Neural learning.
        # Note code might not work with other data [2]
```

```
In [ ]: #dataset 3 x 4 Matrix
        training_data = np.array([[0,0,1],
                                  [1,1,1],
                                  [1,0,1],
                                  [0,1,1]])
```

```
In [ ]: #output dataset ground truth - i.e., classes {0, 1, 1, 0}
        # y vector
        ground_truth = np.array([[0,1,1,0]]).T
        print(ground_truth)
```

The goal is to find the combination of weights which minimizes the error function, to get training output that is close to the ground truth:

$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_m \cdot w_m$$



Neuron calculates output if we apply activation function f on an input function z:

$$y = f(z)$$

10 min

Tutorial

NB-sigmoid-N-Model.ipynb

10 min

Sigmoid

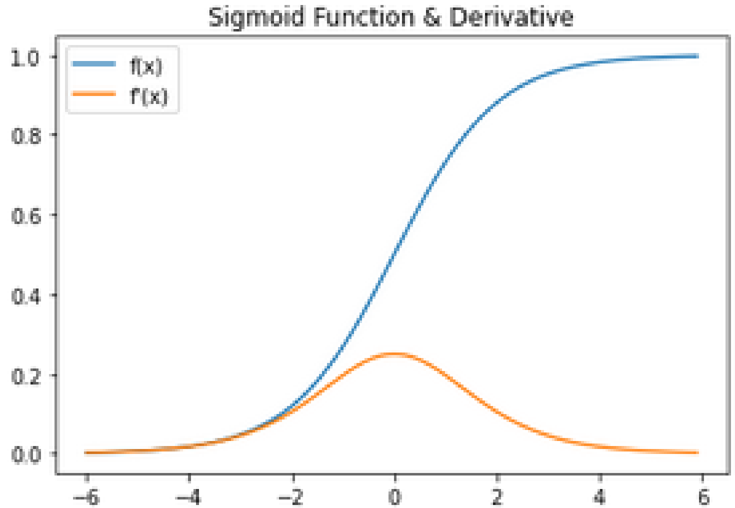
Given a number N, the sigmoid function would map that number between 0 and 1, which means we can use this as probability distribution.

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

```
The derivative of the sigmoid function with respect to x:
```

```
# lets compute

def sig(x):
    return 1/(1+np.exp(-x))

# Sigmoidal derivative
def dsig(x):
    return sig(x) * (1-sig(x))

# Generating data to plot
x = np.arange(-6., 6., 0.1)
y = sig(x)
dy = dsig(x)

# Plotting
plt.plot(x, y, x, dy)
plt.title('Sigmoid Function & Derivative')
plt.legend(['f(x)','f\'(x)'])
plt.show()
```

$$f'(x) = \sigma(x)(1 - \sigma(x))$$



Tutorial 3

NB-sigmoid-N-Model.ipynb

For a Feed forward calculations in Neuron we will need:

- input data matrix 3 x 4
- weight matrix 3 x 1 (3 input & 1 output nodes)
- activation function (here sigmoid), to apply on an input function Z (product of the input data with initial weights).

In [ ]:
```
np.random.seed(1)
```

In [ ]:
```
# weight matrix 3 x 1 (3 input & 1 output nodes)
#initialize weights randomly (-1 to 1) with mean 0
weights = 2*np.random.random((3,1))-1

print('Random starting weights', )
print(weights)
```

Training process

1. Feed forward calculation
   - Input layer: training data
   - calculate training_output in Neuron model using sigmoid activation function (bias term is neglected)
2. "Backpropagation" basics
   - Calculate the error (loss), the difference between the ground_truth and actual output
   - Calculate update term
   - Adjust the weights accordingly to minimize error
   - Repeat this 10000 times

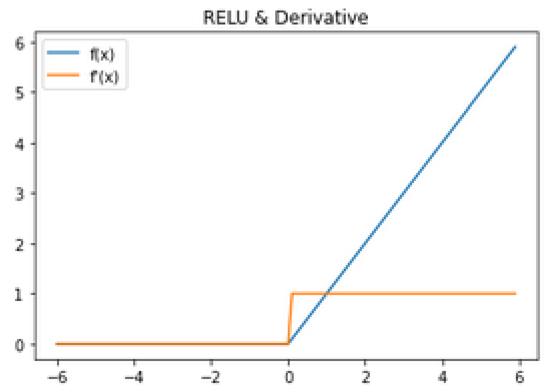
10 min

Tutorial 3

NB-3-1_sigmoid-N-Model.ipynb

ReLU

```
# Plotting
plt.plot(x, y, x, dy)
plt.title('RELU & Derivative')
plt.legend(['f(x)','f\'(x)'])
plt.show()
plt.savefig('relu.png', bbox_inches='tight')
```



Softmax

```
In [11]: def softmax(x):       # x is vector
             return (np.exp(x))/sum(np.exp(x))

         x = np.array([1, 0.3, 3, 0.5])
         prob = softmax(x)       # converts list of numbers to a list of probabilities
         print(prob)             # output - probabilities
         print(sum(prob))        # sum of the probabilities gives 1

         [0.10534997 0.05231524 0.77843681 0.06389798]
         1.0
```

10 min

Tutorial 3

Include 2 Jupyter notebook

- notebooks/NB-3_1_sigmoid-N-Model.ipynb
- notebooks/NB-3-2_simple_NN.ipynb (optional)

10 min

Hands-on: Implement simple straight-forward neural network in TF

Outcome: Basic understanding of Neural Network architecture and building blocks.

Tutorial 3

Include 2 Jupyter notebook

- notebooks/NB-3-1_sigmoid-N-Model.ipynb
- notebooks/NB-3-2_simple_NN.ipynb

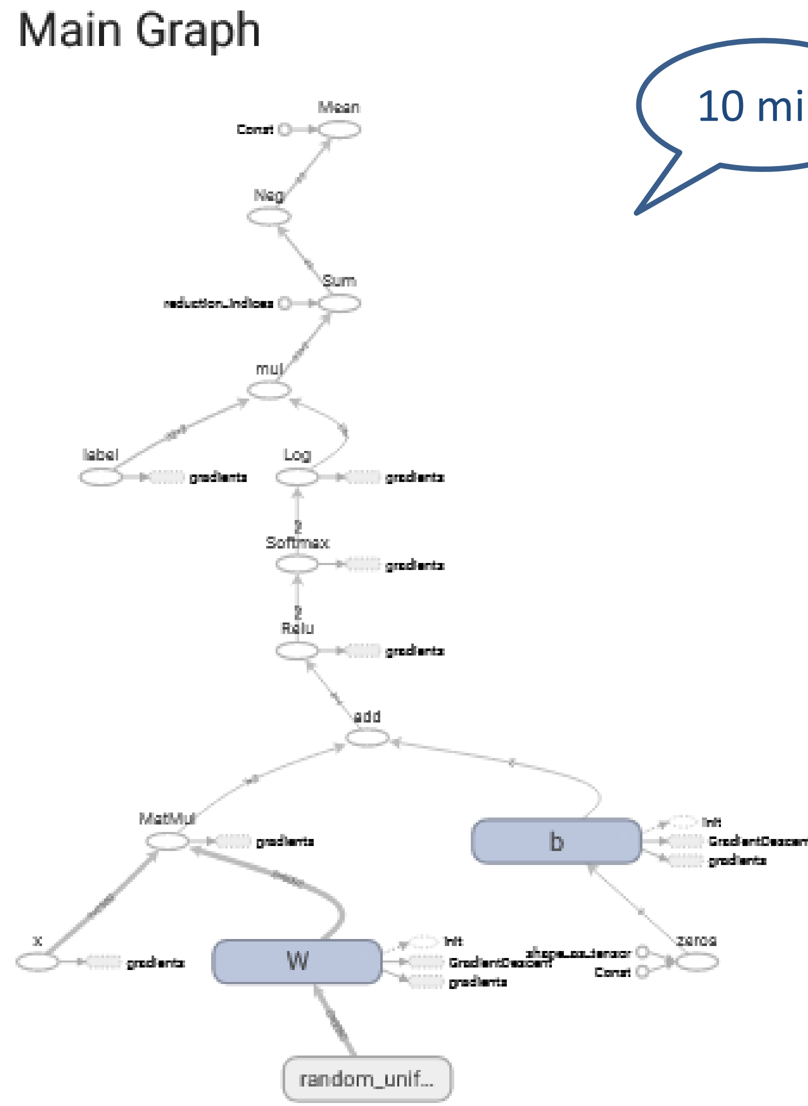
10 min

- TensorFlow (TF) origin: Google Brain Team
- Open source software library for numerical computation using data flow graphs
- It deploys computation to one ore more CPUs / GPUs and TPUs in a desktop, server, or mobile device with a single API.

- Tensorboard (visualization tool)

Tutorial 3

NB-3-2_simple_NN.ipynb



10 min

$x_i \in \mathrm{R}^D;$

here $i = 1 \ldots m$

m images, each with $D = 50 \text{ x } 50 \text{ x } 1$ px

$y_i \in 1 \ldots n;$ here $n = 2;$

$x\left[2500 \text{ x } 1\right];\ \mathbf{W}\left[2 \text{ x } 2500\right]; b\left[2 \text{ x } 1\right]$

$f : \mathrm{R}^D \mapsto \mathrm{R}^n$

Tutorial 3

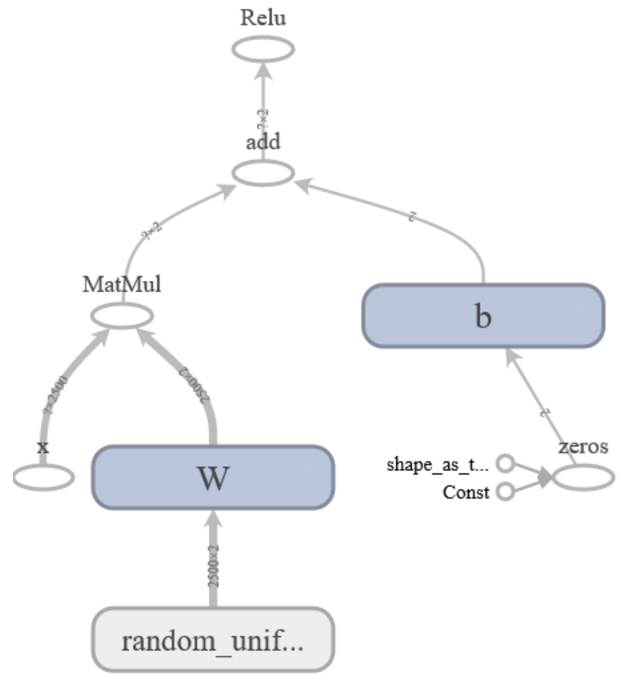
NB-3-2_simple_NN.ipynb

10 min

Data Flow Graphs

Representations of the data dependencies between a number of operations

- Graph Nodes - math. Operation
- Edge – multi-dimensional data set



TensorFlow (TF) does have its own data structure

- Tensors - an n-dimensional (n-d) array or list
  - core of TensorFlow
  - only tensors are passed between operations

Order/n	0	1	2	3
	Scalar	Vector	Matrix	
	100	[4, 5, 7, ..., 10]	$\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$	

Details skipped

Tutorial 3

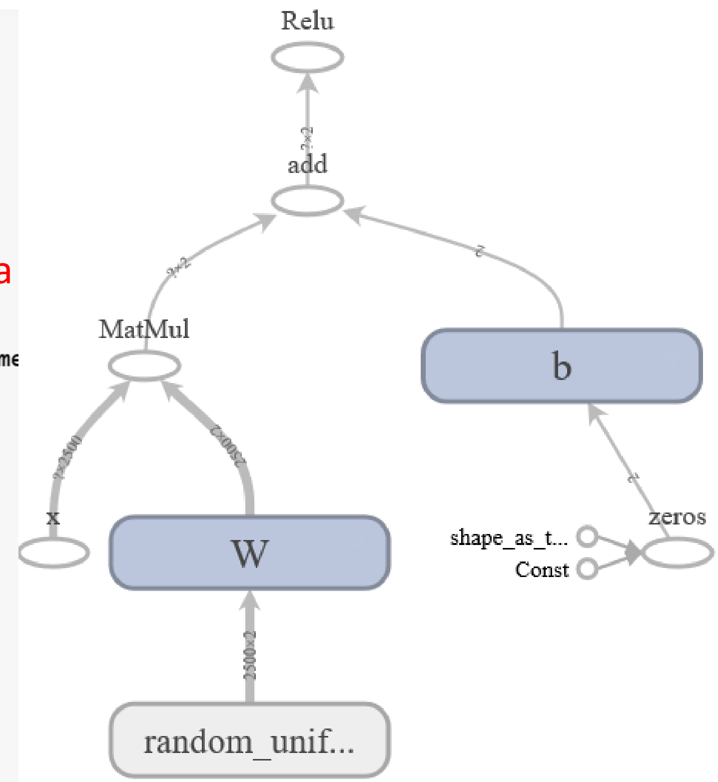
NB-3-2_simple_NN.ipynb

10 min

To do calculations:
- Build the graph … construct all the operation dependencies
- Run the graph … feed with data and compute result

```
#Create a graph
# Input parameters
max_height, max_width = 50, 50
num_classes = 2
learning_rate = 0.001
batch_size = 32

#The bias term b
b = tf.Variable(tf.zeros((num_classes)), name='b')

# Shape and type of data W(weight) - random uniform distribution between -1 and 1
W = tf.Variable(tf.random.uniform((max_height*max_width, num_classes), -1, 1), name

# x(image input)(m*max_height*max_width) placeholder
x = tf.placeholder(tf.float32, [None, max_height*max_width], name='x')

# apply nonlinearity on (Wx+b) via ReLU activation function
Z = tf.nn.relu(tf.matmul(x, W) + b)

#print(b, W, x)
print("Bias shape:", b.shape)
print("Weight shape:", W.shape)
print("Data shape:", x.shape)

#create the writer out of the sesion
writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())
```

Shape and type of data



Tutorial 3

NB-3-2_simple_NN.ipynb

10 min

In [6]:
```
#softmax function returns an array of 2 probability scores that sum to 1.
prediction = tf.nn.softmax(Z)
label = tf.placeholder(tf.float32, [batch_size, num_classes], name ='label
# init. variables
init_op = tf.variables_initializer(tf.global_variables())
# loss function
cross_entropy = tf.reduce_mean(-tf.reduce_sum(label * tf.log(prediction),
# training with Gradient Descent (GD) optimizer
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cro

#create the writer out of the session
writer = tf.summary.FileWriter('./graphs', tf.get_default_graph())

#Convert training data fr
train_data = np.zeros((len
for i in range (len(train_
    train_data[i] = train_
```

In [7]:
```
#To compute, launch the graph in a session
with tf.Session() as sess:
    #sess.run(tf.initialize_all_variables())
    sess.run(init_op)
    max_epochs = 1   # We will run just for 2 epochs
    for epoch in range(max_epochs):
        # Compute gradient and update parameters per batch
        for batch_num in range(int(len(train_data)/batch_size)):
            batch_data = train_data[batch_num*batch_size:min((batch_num+1)
            batch_label = np.eye(num_classes)[np.int_(train_labels[batch_n
            sess.run(train_step, feed_dict={x: batch_data, label: batch_la
            #You can calculate and report how the batch loss here changes
            loss = sess.run([cross_entropy], feed_dict={x: batch_data, lab
```

- Define a session, which graph to run
- Run the operation
- feed data to the graph

Tutorial 3

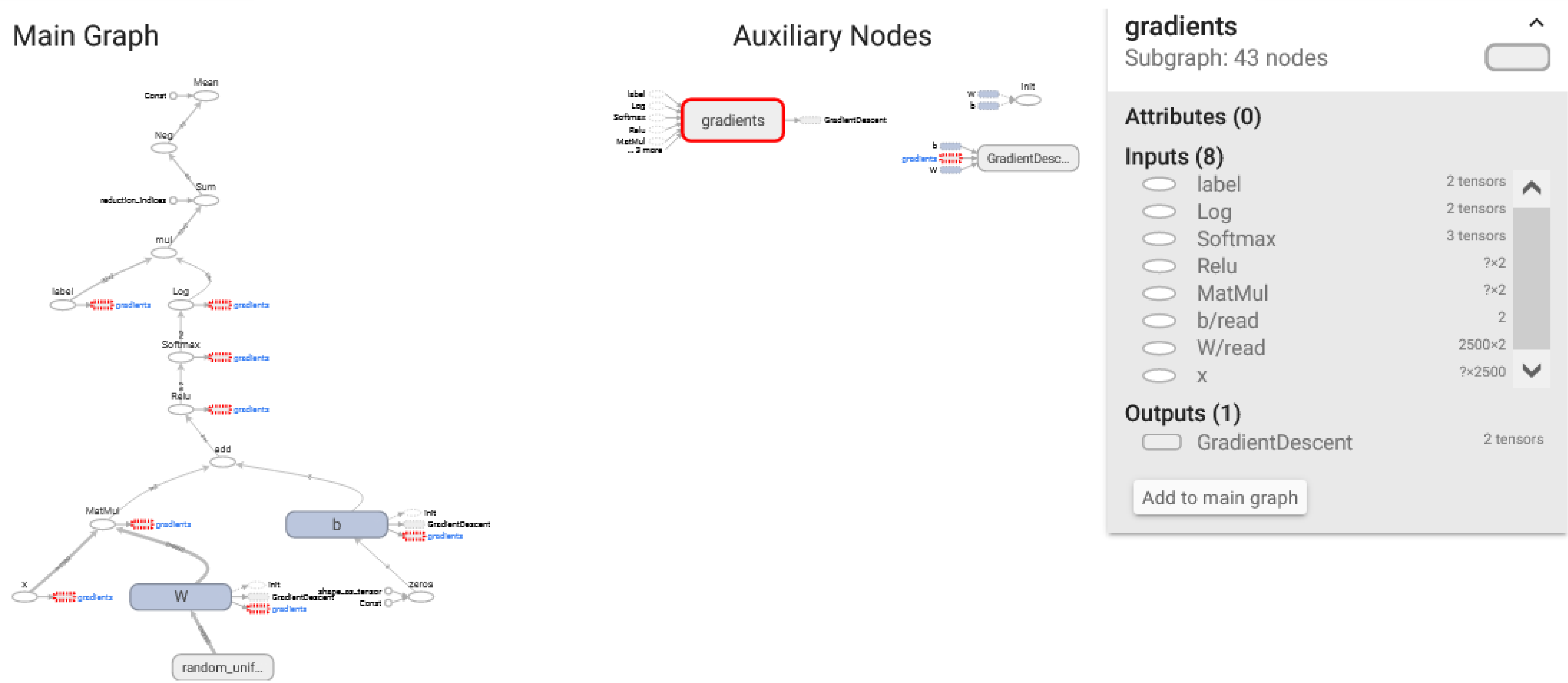
NB-3-2_simple_NN.ipynb

TensorBoard Visualization

Next open TensorBoard at http://n08xxxx:6006/ to visualize main graph.

10 min

```
In [ ]: !tensorboard --logdir="./graphs" --bind_all
```



Learning a model

Configure the Model Architecture	Compile the model	Train the Model
• Artificial Neural Networks (ANN) • Multilayer Perceptron (MLP) • Convolutional Neural Networks (CNN)	• Loss (to measures how accurate the model is during training) • Optimizer (to minimize Loss with respect of parameters) • Metrics (to evaluate performance)	• Performance at Task improves with an Experience • Train to classify images • Track epochs, let model see every pictures many times; babysit process

Compile the model

- **Loss function:**
  measures how accurate the model is during the training.

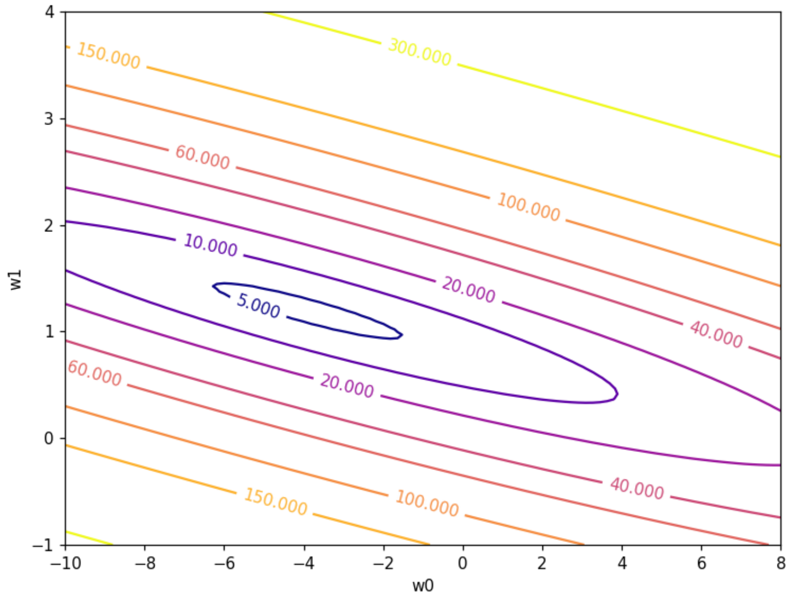
Minimize the loss function

- **Optimizer:**
  the model update based on the data it sees and its loss function.

- **Metrics:**
  used to monitor the training and evaluation steps.

Compile the model

- **Loss function:** measures how accurate the model is during the training.
- This can be as simple as MSE (mean squared error) or more complex like cross-entropy.
-



Optimization

- Based on the **gradient descent** algorithm
- Minimization of error through optimization of function
- Moving in the opposite direction of the gradient
- Backpropagation adjusts parameters (backward) given the error
- Training Data may be iterated multiple times
- Complete pass over the data - „epoch“

- Optimizers: GD, SGD, Adam, PMSProp …

Classification metrics

- We need a Performance **measure P**
  - to assess the performance of the model
  - to monitor the training and evaluation steps

- Default metric for classification is **accuracy,** the fraction of the images that are correctly classified

- This **metric is not useful** when there is a **data imbalance**
  - the distribution of examples in the training dataset across the supercategories is not equal
  - e.g. proportion in supercategory < 50%

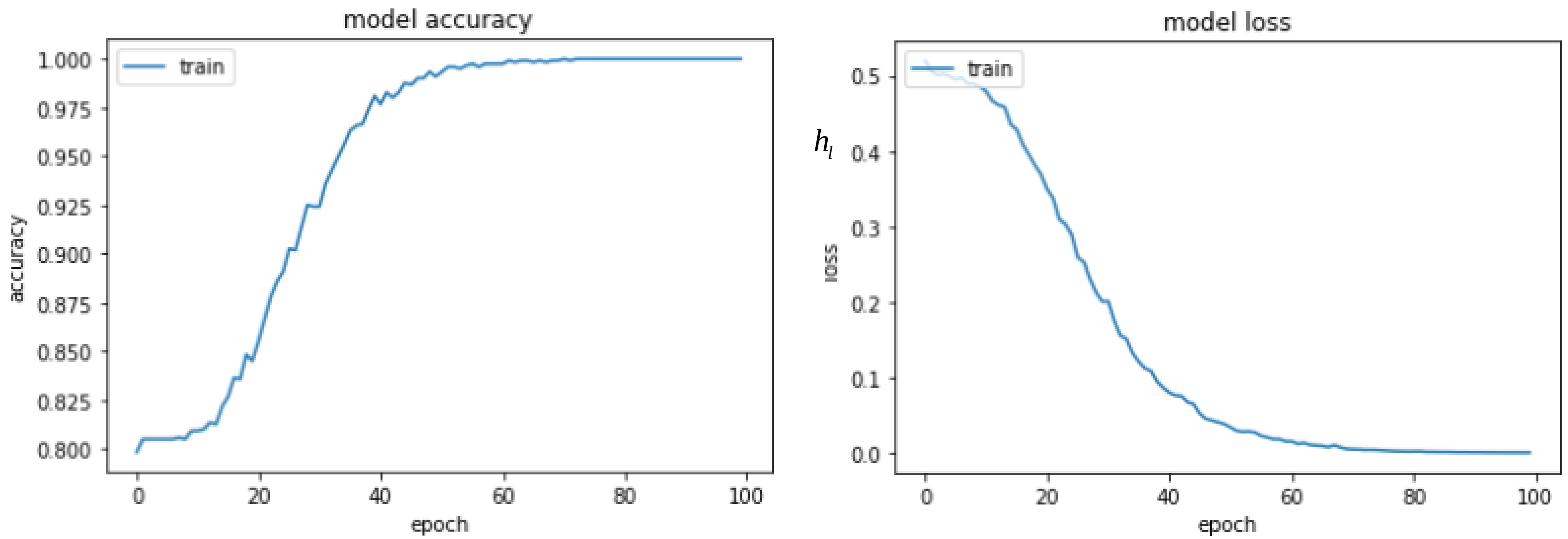
Classification metrics

Problem:

- Only 37% of the validation set and 21% test set with bottles .
- Predicting every image **as not containing** a bottle would give ~63% and ~79 % accuracy, which is not representative of how well the model is doing on predicting bottles.

Classification metrics

The **low performance** on the minority class (supercategory) **is** not **captured** in the accuracy metric.



Almost perfect accuracy according to the model training history.

Classification metrics

More complete picture according to the **confusion matrix**

- how many classes were correctly classified vs misclassified?
- The simplest confusion matrix for a 2-class classification problem, with negative (0 - no bottle) and positive (1 bottle) classes
- Precision - percentage of relevant results
- While recall is characterized as the percentage relevant results that are correctly classified

Predictions

		+	-
Actual class	+	**TP** True Positive	**FN** False Negative Type II error
	-	**FP** False Positive Type I error	**TN** True Negative

what if FN represents one with COVID-19 ?

Classification metrics

Main metrics: Breakdown the accuracy formula even further

Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

Learning a model

Configure the Model Architecture	Compile the model	Train the Model
• Artificial Neural Networks (ANN) • Multilayer Perceptron (MLP) • Convolutional Neural Networks (CNN)	• Loss (to measures how accurate the model is during training) • Optimizer (to minimize Loss with respect of parameters) • Metrics (to evaluate performance)	• Performance at Task improves with an Experience • Train to classify images • Track epochs, let model see every pictures many times; babysit process

Training of a NN

- From the training example Inputs get the neurons output (feedforward)
- Calculate the error (loss), the difference between the output we got after calculation and actual output (the ground truth - input labels)
- Adjust the weights accordingly to minimize error (Backpropagation)
- Backpropagation computes the derivative of the loss with respect of weights (different optimizer: GD, SGD, Adam, PMSProp …)
- Repeat this many times (i.e., Epoche = 20 or more)
- A small loss leads to a good prediction

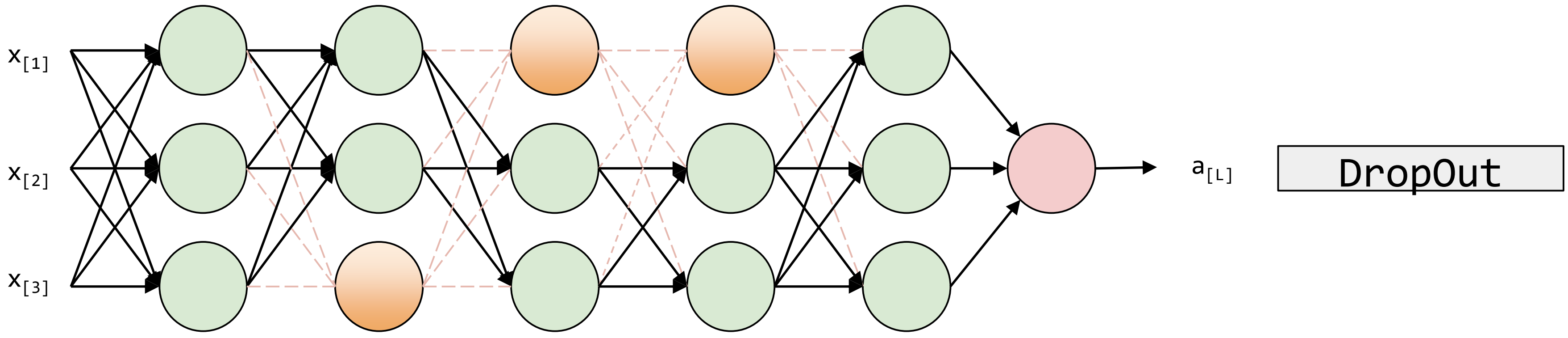
Evaluation

Evaluate

- Model performance is evaluated on validation set
- Trained Model gives predictions on unseen data
- Chosen Metrics suffice

Dropout

- Regularization technique
- Prevents overfitting making memorization difficult
- Method: randomly throw activations away (e.g. p=0.5),
- Early dropout coupled with RELU – preventive



Source: https://github.com/dair-ai/ml-visuals

Deep Learning

Tutorial 4

Tutorial 4 - Image Classification with MLP

- **Hands-on:** Image Classification with Multilayer Perceptron (MLP)
- **Outcome:** Basic understanding of Neural Network architecture and building blocks of an image classification pipeline. Ability to modify the model architecture, compile, train NN and visualize.

Tutorial 4

Jupyter notebook

- Open a **notebooks/** NB-4_train_NN_50.ipynb

Tasks to complete:

- Load saved numpy arrays (3 image sets, 3 label sets)
- Summarize training, validation, and test data.
- Normalize, scale, experiment
- Configure model for MLP
- Experiment with hyperparamenters (learning rate etc.)
- During experiment use number of epochs = 30
- Visualize training history

Tutorial 4

NB-4_train_NN_50.ipynb

$x_i \in \mathrm{R}^D$;

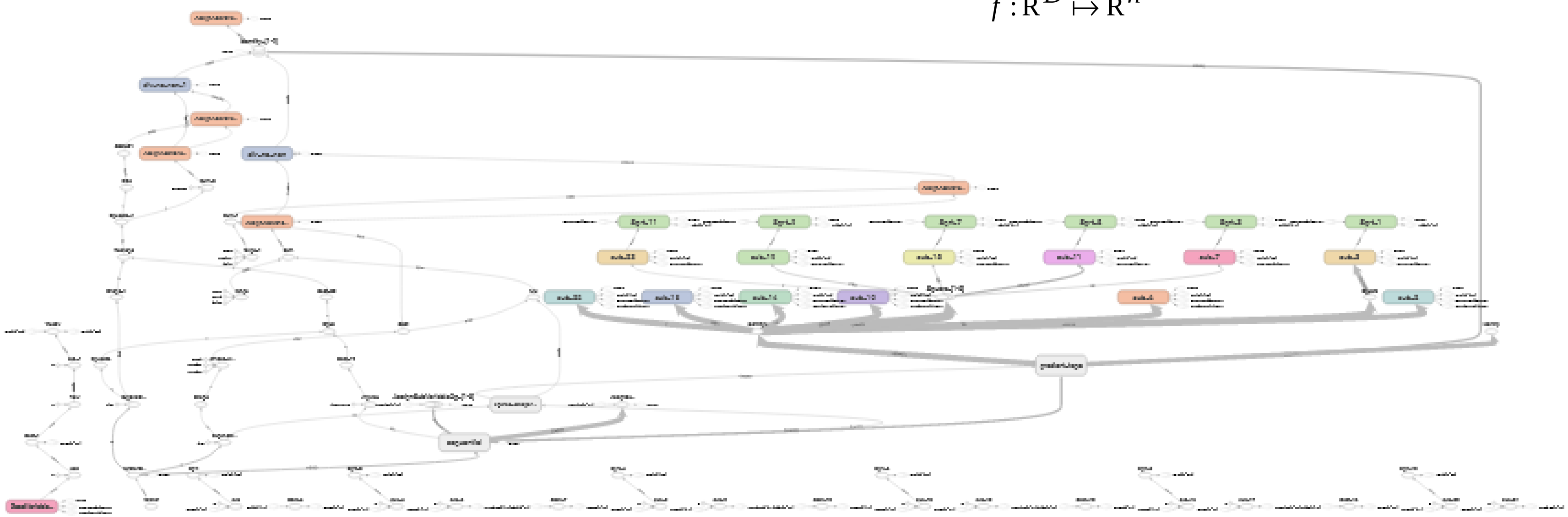
here i = 1…m

m images, each with D = 50 x 50 x 1 px

$y_i \in 1 \ldots n$; here n = 2;

$x\left[2500 \times 1\right]$; $\mathbf{W}\left[2 \times 2500\right]$; $b\left[2 \times 1\right]$

$f : \mathrm{R}^D \mapsto \mathrm{R}^n$



Learning a model

Configure the Model Architecture	Compile the model	Train the Model
• Artificial Neural Networks (ANN) • Multilayer Perceptron (MLP) • Convolutional Neural Networks (CNN)	• Loss (to measures how accurate the model is during training) • Optimizer (to minimize Loss with respect of parameters) • Metrics (to evaluate performance)	• Performance at Task improves with an Experience • Train to classify images • Track epochs, let model see every pictures many times; babysit process

Instead of MLP

Image Classification task

- **Dataset** → annotated, RGB images of different size
- **Image Classification task** → predict a single label (or a distribution over labels to indicate confidence) for a given image.

- Resize: 1000 pixels wide, 1000 pixels tall.
- RGB → gray scale images
- Results → 1000 x 1000 x 1, or a total of 1 Million numbers
- Pixel range: from 0 (black) to 255 (white)
- The task: to turn numbers into a single label, such as *"bottle"*

Image Classification challenges

- **Viewpoint variation** of a single instance of an object (bottle)
- **Scale variation** - size in the real world vs in the image
- **Deformation** - i.e., deformed plastic bottle
- **Occlusion** - only a small portion of an object visible
- **Illumination conditions** - direct effects on the pixel level
- **Background clutter** - making hard to identify object
- **Intra-class variation** - many different types of these objects

➢ CNNs systematize this idea of **spatial invariance**, exploiting it to learn useful representations with fewer parameters.

Why Convolutions?

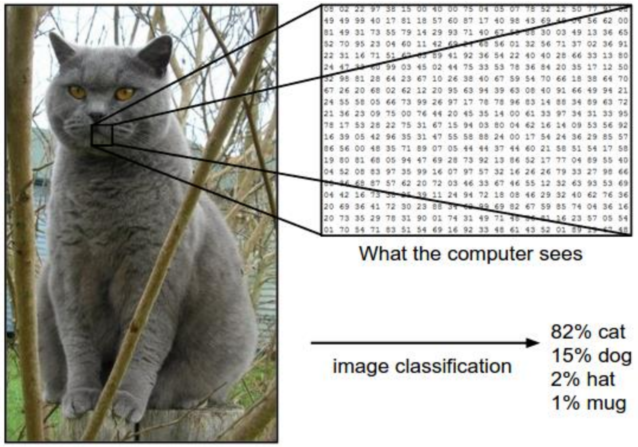
Problems:

- **Impractical** to use ANNs for real-world image classification
  - a 2D image - 1 Million numbers per image
  - If the first hidden layer has 1000 nodes
  - the matrix of input weighs → 1000 x 1000 x 1000
  - increasing the number of layers **increase numbers rapidly**
- **Vectorising** an image **ignores** the complex **2D spatial structure**

- How to build a system that overcomes both these disadvantages?
  - **Convolutional neural networks (CNNs)** are one creative way

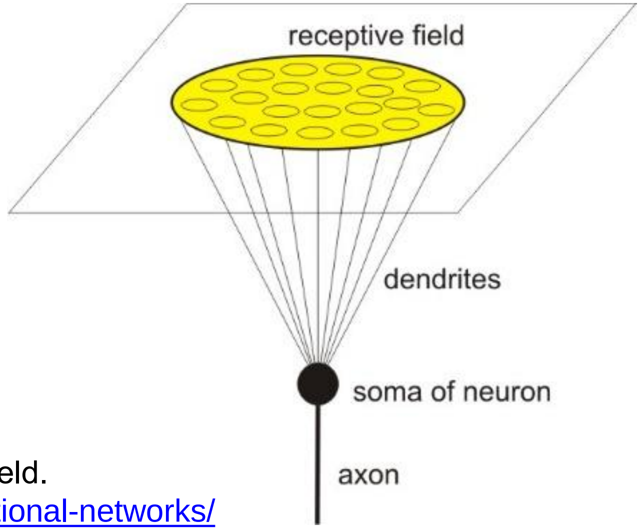
CNN

- Human perception is very accurate
- Computers see images as 2D arrays of pixels
- Algorithms need to be trained on lots of images



Source: http://cs231n.github.io/classification/

- CNN mimics human eye



A a single sensory neuron’s receptive field.
Source: http://cs231n.github.io/convolutional-networks/

CNN - advantages and disadvantages

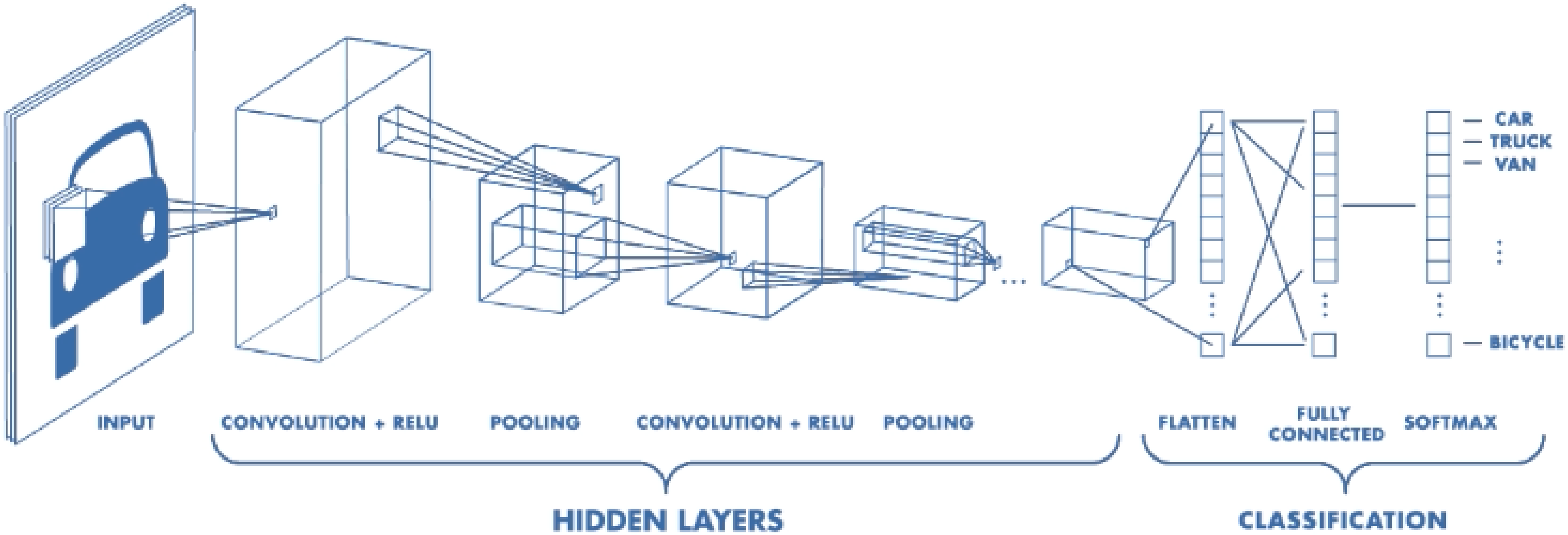
Pros

- CNN - is a class of deep feedforward ANN, that **contain convolutional layers**
- Improves the performance by using **spatial information** of pixels of an image
- Convolutional layers **require fewer parameters** than fully-connected layers
- **Larger the data, greater the accuracy** - the first fully connected layer with thousands of weights
- **Translation invariance** in images **automatically obtained**
  - all patches of an image are treated in the same manner
  - the same weights across the whole space
- **Locality** - from a small neighborhood of pixels to the corresponding hidden representations

Cons

- Downside of deep CNN: a bad learning performance could be improved with hyperparameter tuning

Components of CNN

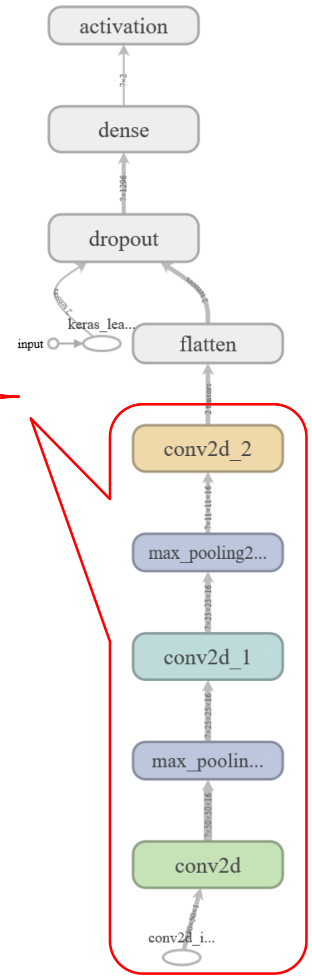


Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

CNN MODEL

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 50, 50, 16)	160
max_pooling2d (MaxPooling2D)	(None, 25, 25, 16)	
conv2d_1 (Conv2D)	(None, 23, 23, 16)	2320
max_pooling2d_1 (MaxPooling2	(None, 11, 11, 16)	
conv2d_2 (Conv2D)	(None, 9, 9, 16)	2320
flatten (Flatten)	(None, 1296)	0
dense (Dense)	(None, 2)	2594
activation (Activation)	(None, 2)	0

Total params: 7,394
Trainable params: 7,394
Non-trainable params: 0



→ Results of the labels; a class {1, 0}

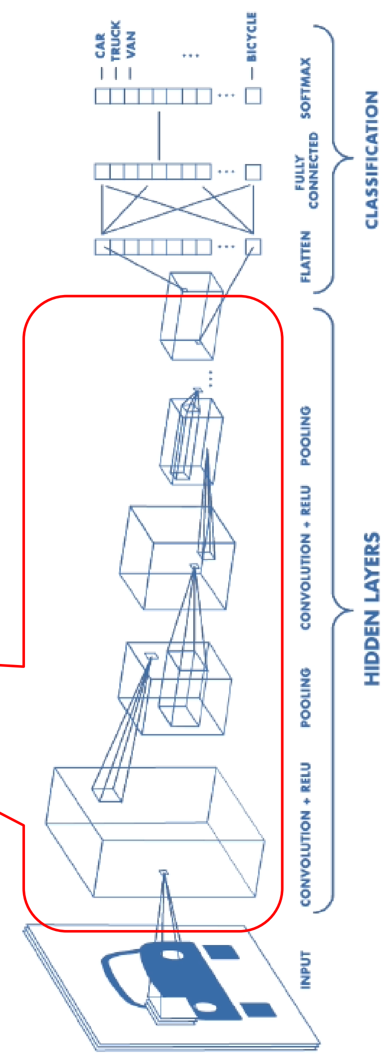
→ Computation

→ Regularization

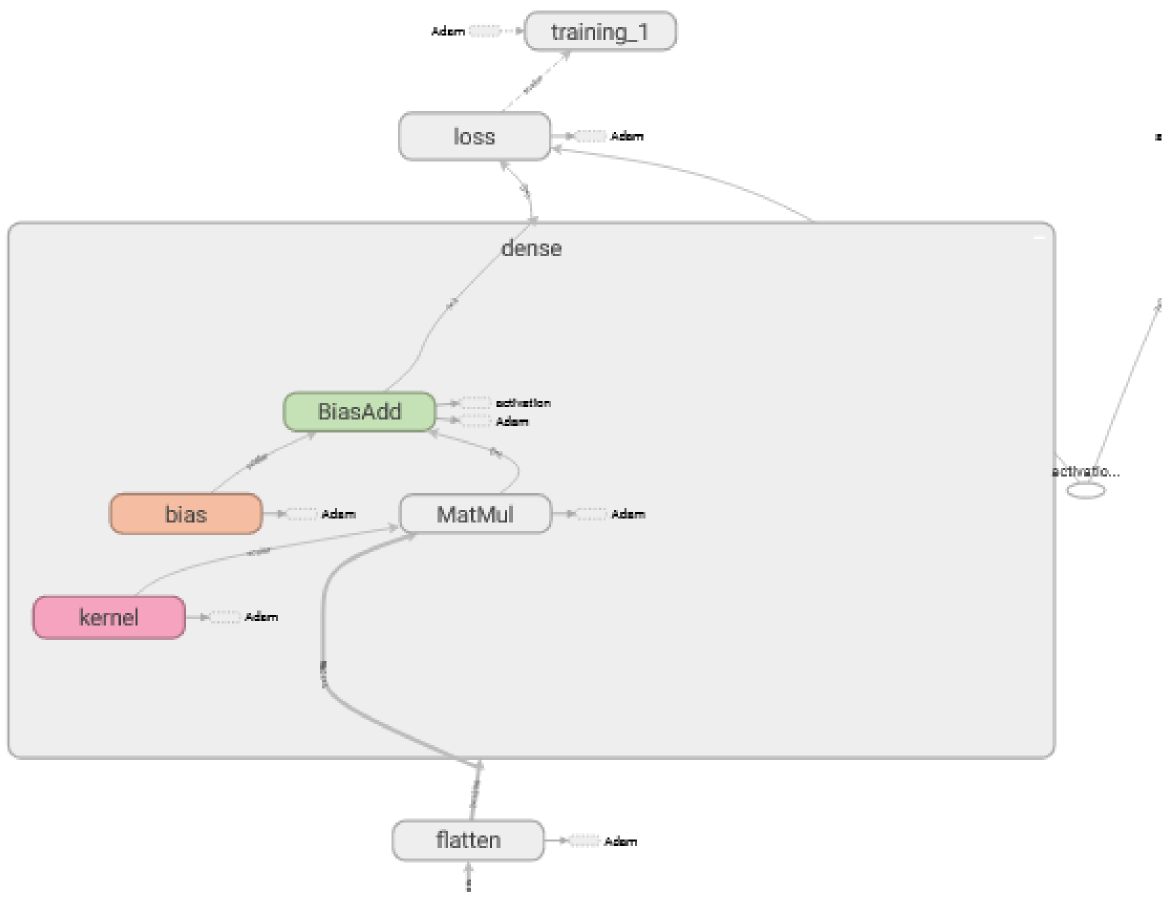
→ a single vector

→ Feature mapping in Convolution layer

→ down-sampling of the data



In Fully Connected Dense layer



MLP

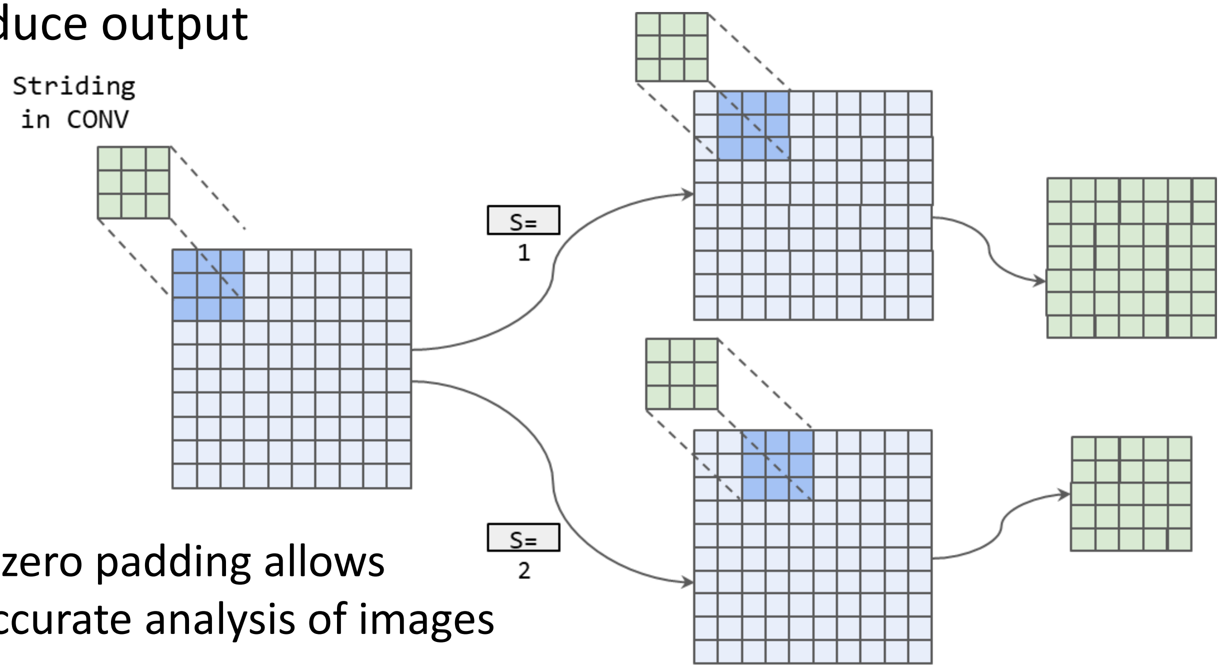
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 2500)              0
_________________________________________________________________
dense (Dense)                (None, 128)               320128
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 320,386
Trainable params: 320,386
Non-trainable params: 0
```

CNN

- Convolution
- Pooling operation
- Activation functions
- Dropout
- Backpropagation

Convolution Operation

- Combination of 2 functions to produce a third function
- Input, kernel (e.g. 3x3), feature map (output)
- Stride kernel across the input and compute matrix multiplication to produce output

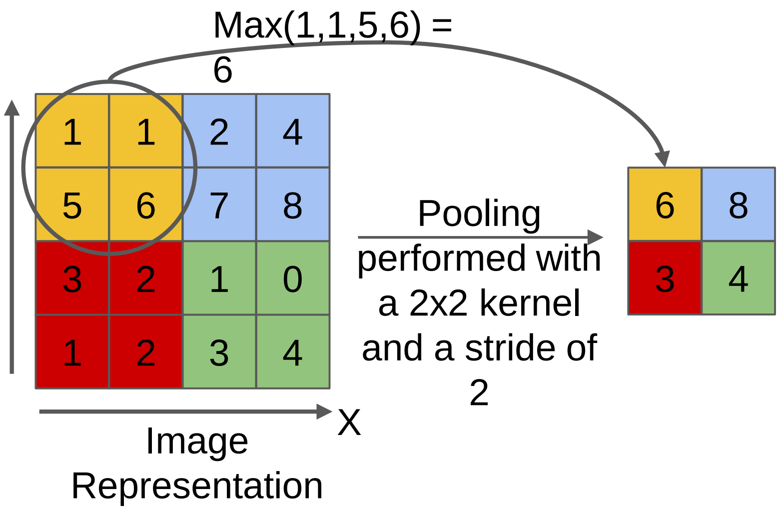


- Adding zero padding allows more accurate analysis of images

Source: https://github.com/dair-ai/ml-visuals

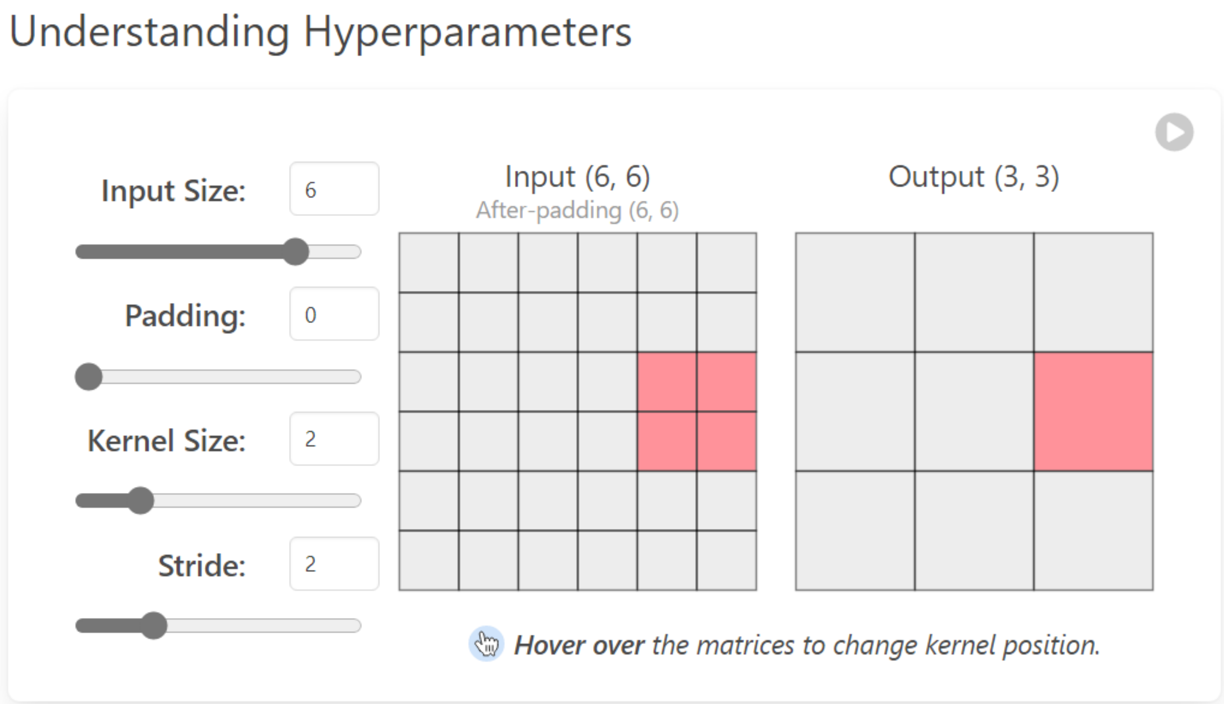
Pooling Operation

- Summarizes the output of a region
- Helps reduce the effect of invariants (small changes to the input)
- Max vs mean-pooling



CNN Explainer

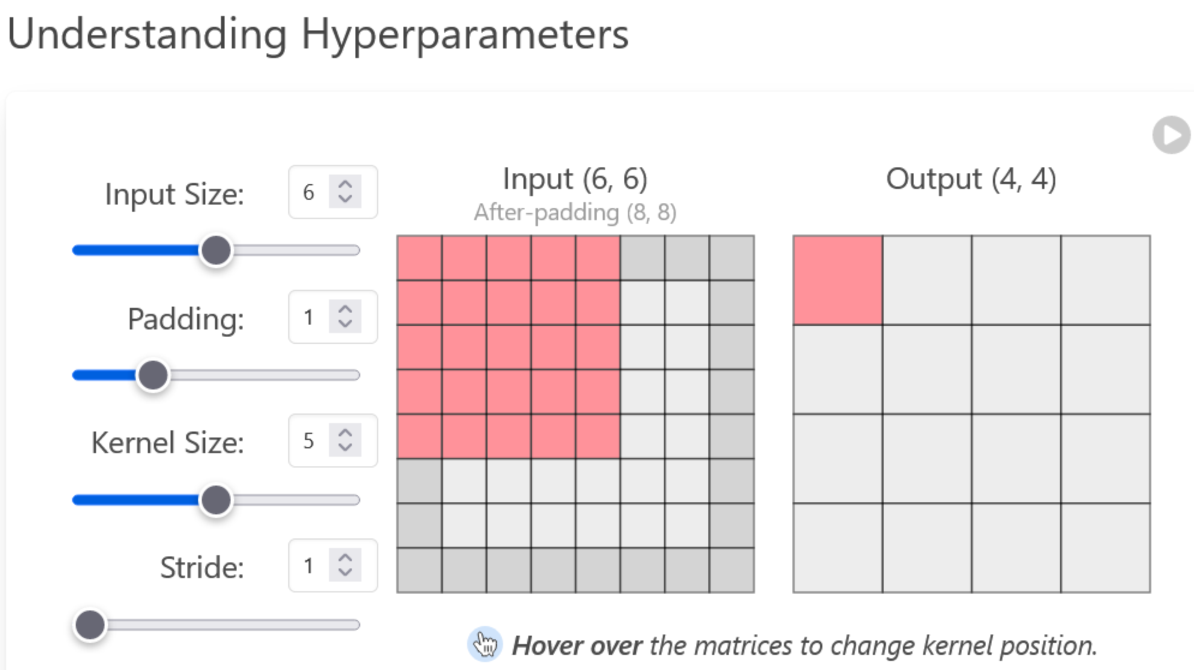
An interactive *visualization* system designed to help non-experts learn about Convolutional Neural Networks (CNNs).



https://poloclub.github.io/cnn-explainer/

CNN Explainer

An interactive *visualization* system designed to help non-experts learn about Convolutional Neural Networks (CNNs).



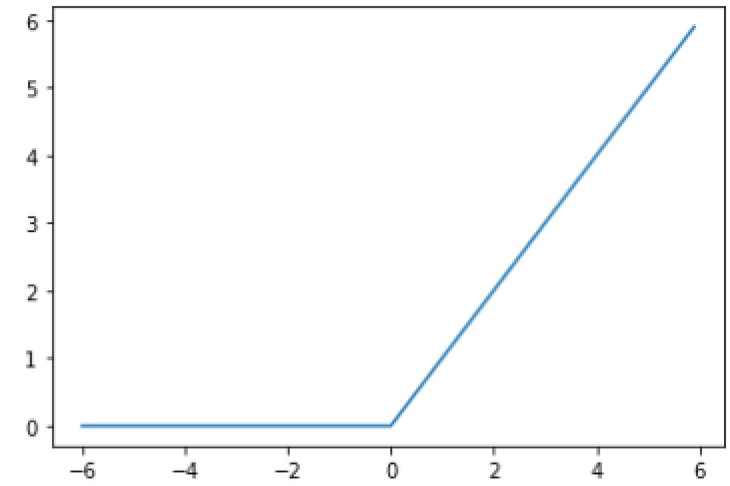
https://poloclub.github.io/cnn-explainer/

Activation functions - ReLU

- Rectified linear unit activation function
- Fast convergence (sparse activations)
- Constant values
- Negative values do not get activated
- For CNN ReLU performs faster *

Problem:

- Dying ReLU: neurons get stuck at 0
- Can lead to model not learning



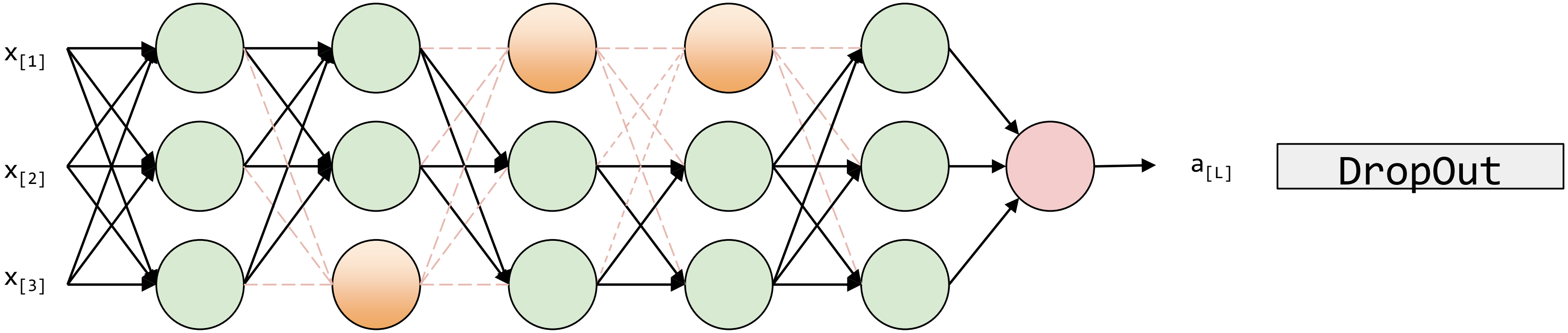
$$\sigma(z) = \max\{0, z\}$$

Solution: Leaky ReLU w/ small slope for negatives

[* 1906.01975.pdf (arxiv.org)]

Dropout

- Regularization technique
- Prevents overfitting making memorization difficult
- Method: randomly throw activations away (e.g. p=0.5),
- Early dropout coupled with RELU – preventive



Source: https://github.com/dair-ai/ml-visuals

Deep Learning

Tutorial 5

Tutorial 5 - Image Classification with CNN

- **Hands-on:** Image Classification with CNN
- **Outcome:** Basic understanding of CNN architecture and building blocks. Ability to explain difference between CNN and FNN; advantages of CNN; modify the model architecture, compile, train CNN and evaluate. Experiment with model hyperparemeters and proper metrics for the unbalanced dataset.

Tutorial 5

20 min

Jupyter notebook

> Open a **notebooks/**NB-5-train_CNN_50.ipynb

Tasks to complete:

- Load saved numpy arrays (3 image sets, 3 label sets)
- Summarize training, validation, and test data.
- Normalize, scale, experiment
- **Configure model for CNN**
- Experiment with the hyperparamenter (learning rate etc.)
- During experiment use number of epochs = 30
- Visualize training history
- Observe how changes affecting results in confusion matrix

Classification metrics

Problem:

- Only 37% of the validation set and 21% test set with bottles .
- Predicting every image **as not containing** a bottle would give ~63% and ~79 % accuracy, which is not representative of how well the model is doing on predicting bottles.

Classification metrics

Detailed report with desired evaluation metrics

Precision:

- Precision measures the proportion of **true positive predictions** relative to all positive predictions.
- It answers the question: **"Of all predicted positive cases, how many are actually positive?"**

Recall (Sensitivity):

- Recall measures the proportion of **true positive predictions** relative to all actual positive cases.
- It answers the question: **"Of all actual positive cases, how many did we correctly predict?"**

	precision	recall	f1-score	support
0.0	0.61	0.84	0.71	94
1.0	**0.25**	**0.09**	0.13	56
accuracy			0.56	150
macro avg	0.43	0.46	0.42	150
weighted avg	0.47	0.56	0.49	150

Validation set

	precision	recall	f1-score	support
0.0	0.78	0.86	0.81	118
1.0	**0.15**	**0.09**	0.12	32
accuracy			0.69	150
macro avg	0.46	0.46	0.46	150
weighted avg	0.64	0.69	0.67	150

Test set

Outlook

- **Data Augmentation:**
  - Check the influence of data augmentation on the model performance
  - From unbalanced data → balanced data

- Work with RGB channel images in CNN

Outlook

- **Image data augmentation**
  - Optional data augmenting using
    - **solution/preprocess.py**
    - **solution/job_preprocessing.pbs**
  - Used to improve the performance and ability of the model to generalize

- **Transfer Learning and Fine-tuning**

  important methods to make big-scale model with a small amount of data.

Transfer Learning

- **Transfer learning** leverages knowledge gained while solving one problem and applies it to a different but related problem.

- It allows to use pre-trained models and adapt them for specific tasks with less data and training time

Fine-Tuning

- **Fine-tuning** involves taking an already trained neural network (such as **VGG-16**) and **retraining part of it using a new dataset**.
- VGG-16 is a **convolutional neural network (CNN)** architecture known for its effectiveness in image classification.
- Stack of multiple, smaller 3x3 convolution kernels, resulting in fewer parameters and more non-linear transformations, enhancing feature learning.

- **A preprocessing** of a new dataset could be necessary.

Thank you!

Acknowledgements:

- SiVeGCS
- HLRS, Universität Stuttgart
- You for your attention

Karlsruher Institut für Technologie (KIT)
138.537 Follower:innen
2 Wochen

Zum Tod von Horst Hippler – der Gründungspräsident des KIT und frühere Rektor der Universität Karlsruhe starb mit 77 Jahren

Das Karlsruher Institut für Technologie trauert um seinen Gründungspräsidenten Professor Horst Hippler, der am 6. März 2024 im Alter von 77 Jahren verstarb. Hippler war seit 2002 Rektor der damaligen Universität Karlsruhe, die er mit der Idee zur Gründung des #KIT im Jahr 2006 zu ihrem ersten Erfolg in der Exzellenzinitiative führte. Von 2009 bis 2012 stand er gemeinsam mit Professor Eberhard Umbach an der Spitze des KIT. Anschließend war er bis 2018 Präsident der Hochschulrektorenkonferenz.

https://lnkd.in/ez9kN78K

Foto: Thomas Klink

Email: khatuna.kakhiani@hlrs.de