Hochschule Konstanz
Fakultät Informatik

Prof. Dr. Oliver Dürr

KI Vorlesung: Machine Learning(III)

Achtung Aufnehmen nicht vergessen

Remark: Much of the material have been developed together with Beate Sick (ZHAW/Uni Zurich).

Learning Objective

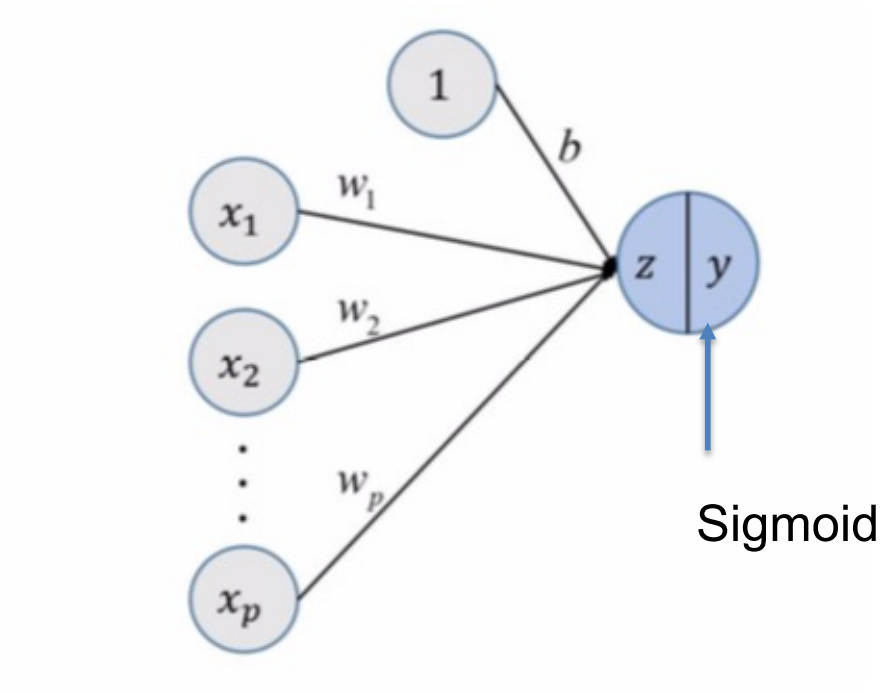
- Know why you need a hidden layer
- Have a rough idea about computational graph and backpropagation
- Is able to construct and train a simple neural network with Keras

Recap:: The Single Cell (logistic regression)

$$z = b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots x_p \cdot w_p$$

$$z = b + \sum x_i \cdot w_i = b + \boldsymbol{x} \cdot \boldsymbol{w}$$

Activation (many possibilities)



Sigmoid

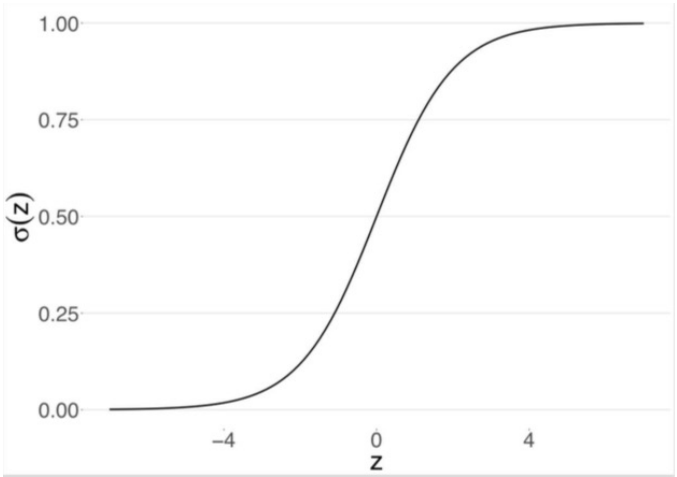
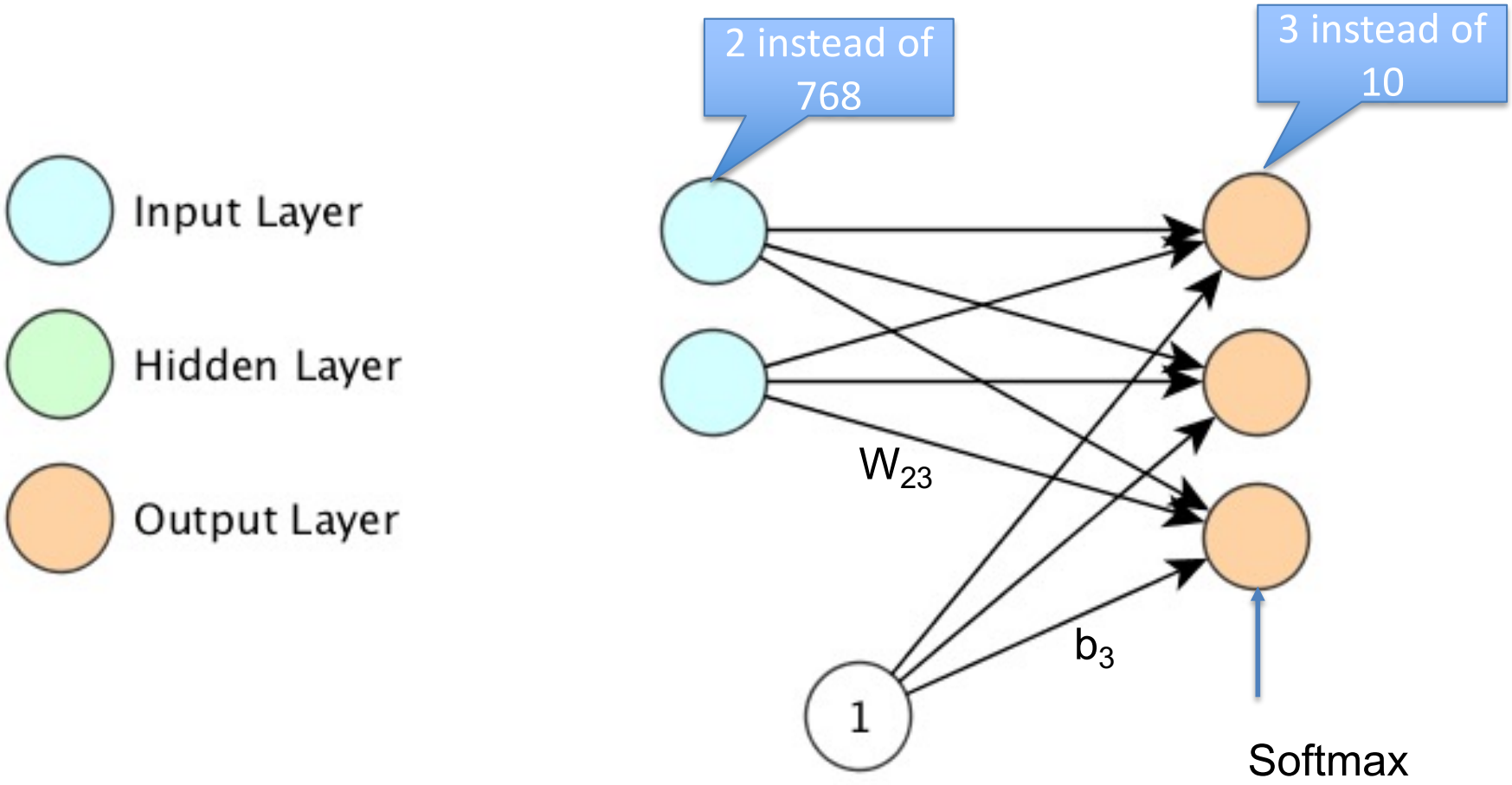


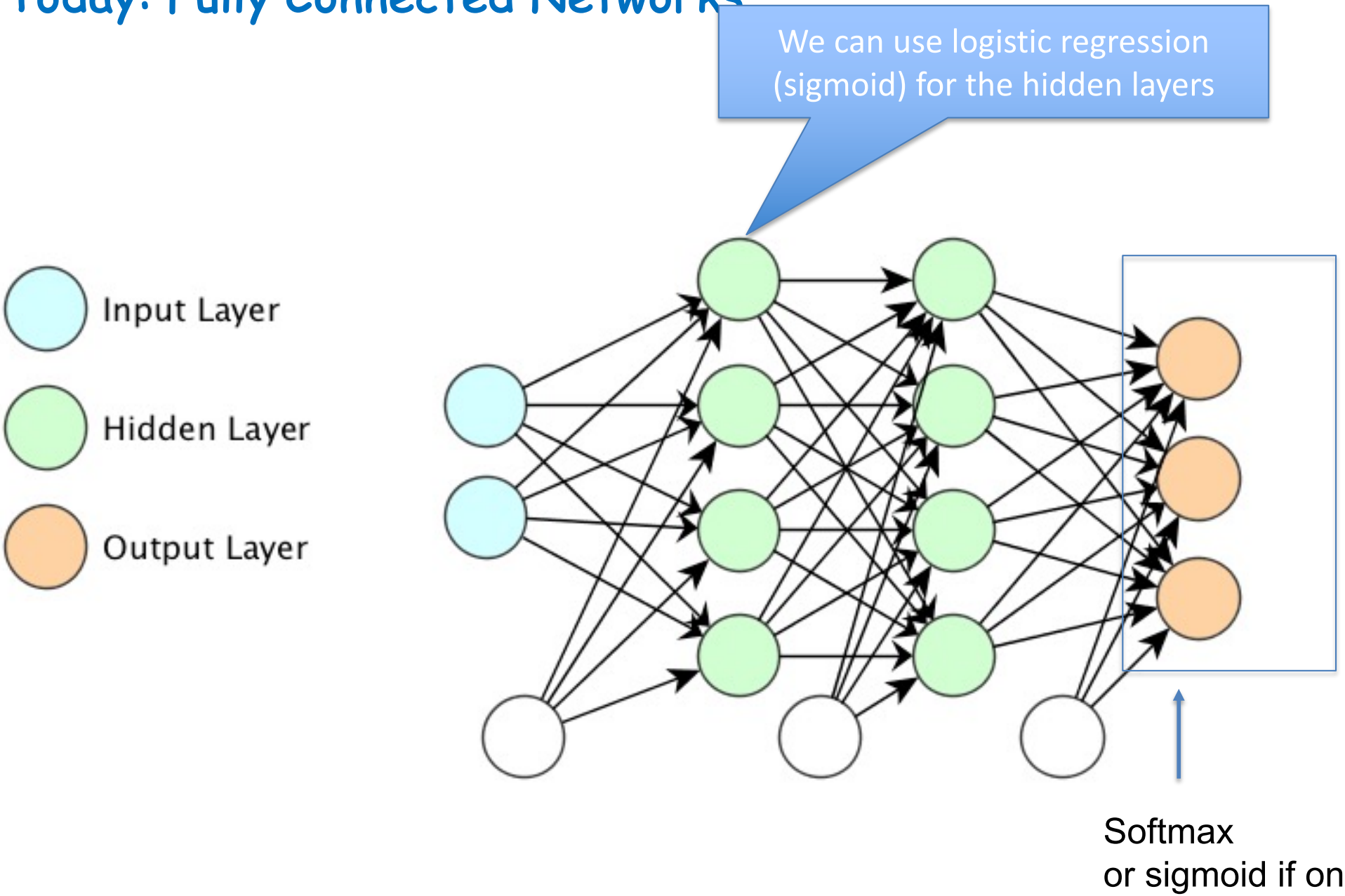
Figure 2.3 The mathematical abstraction of a brain cell (an artificial neuron). The value *z* is computed as the weighted sum of the *p* input values, *x*1 to *xp*, and a bias term *b* that shifts up or down the resulting weighted sum of the inputs. The value *y* is computed from *z* by applying an activation function.

```
# definition of the sigmoid function
def sigmoid(z):
    return (1 / (1 + np.exp(-z)))
```

Recap:: Classification for more than two classes

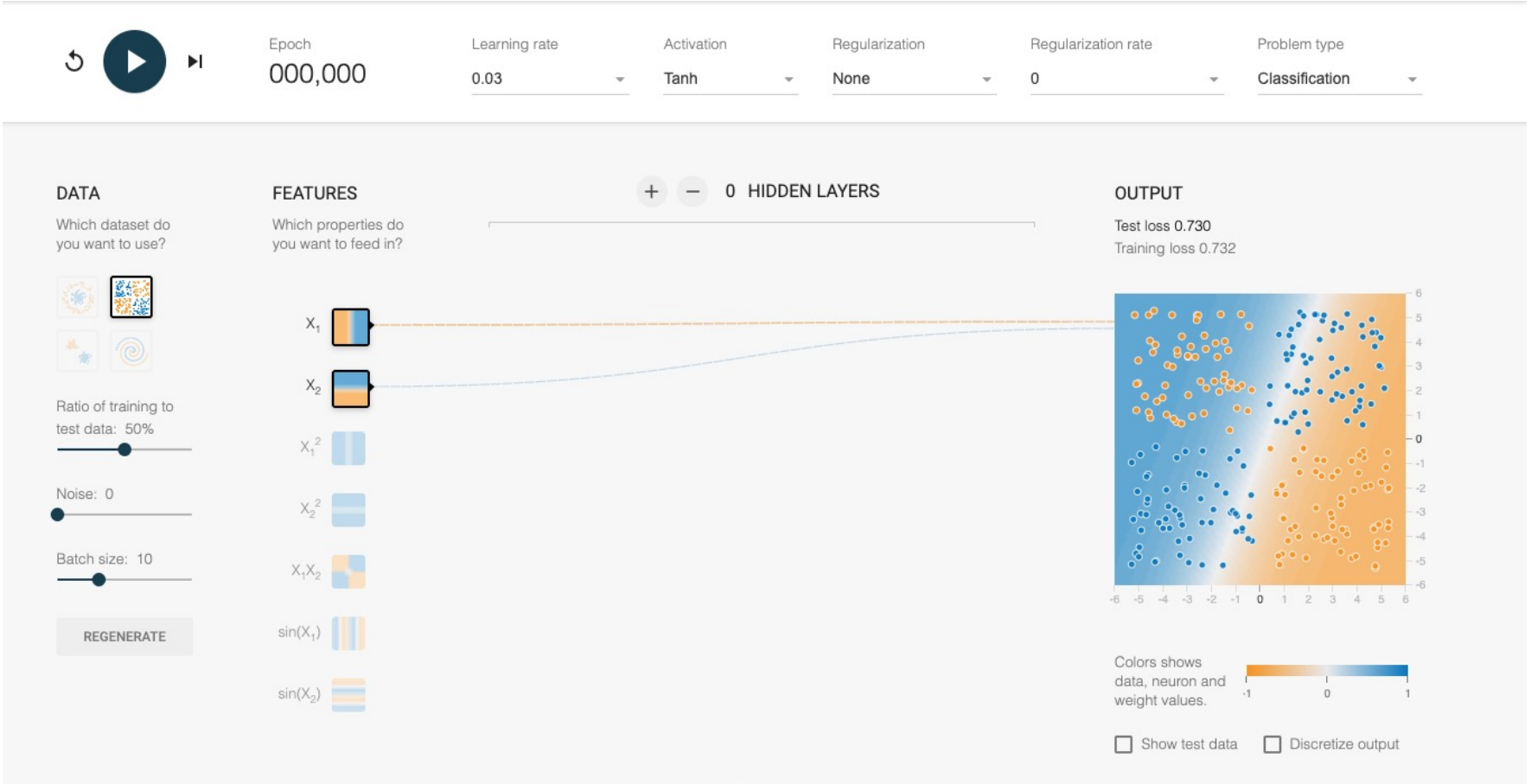


Today: Fully Connected Networks



The need for hidden layers

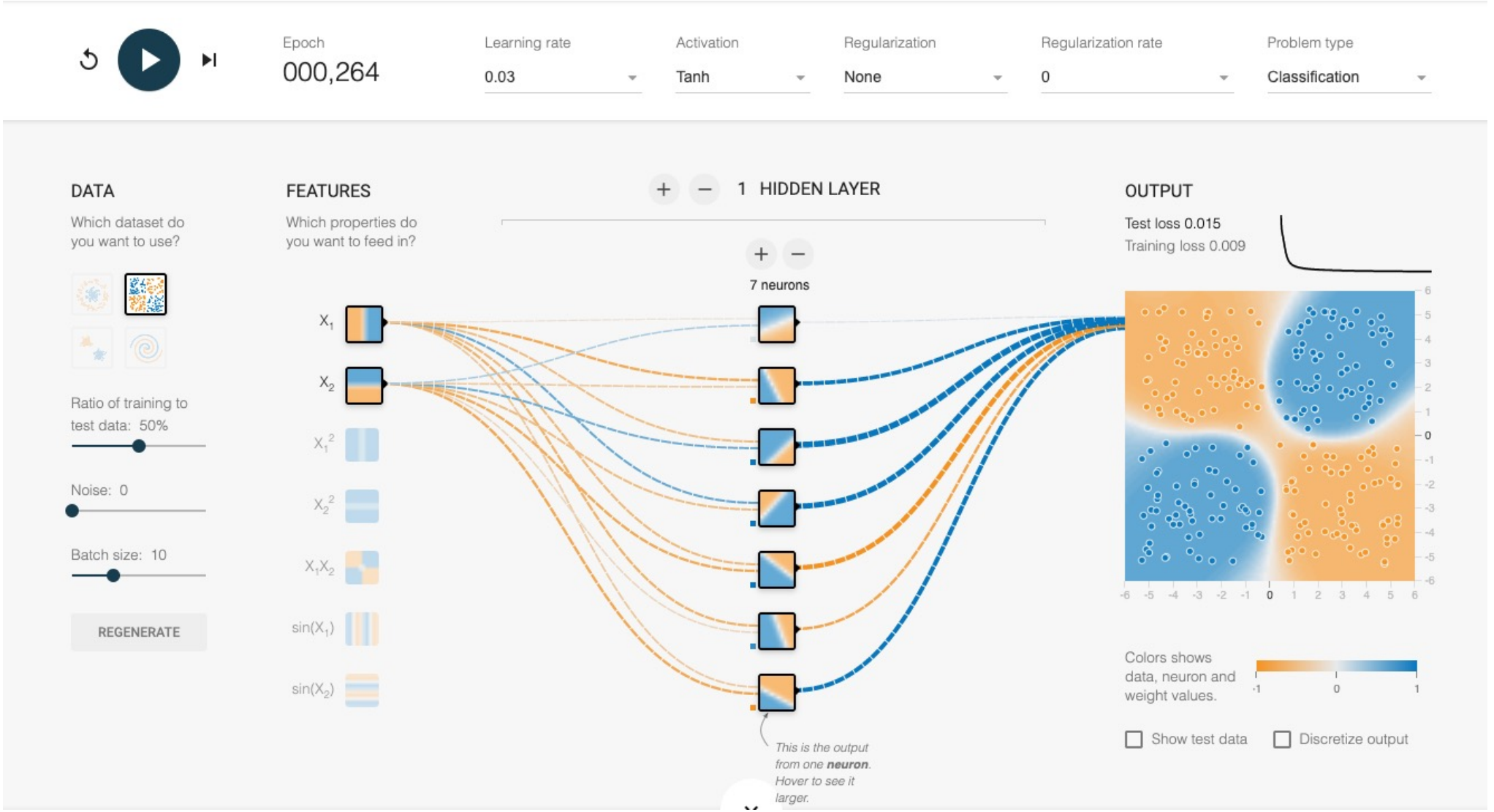
No Hidden / Linear Decision Boundary



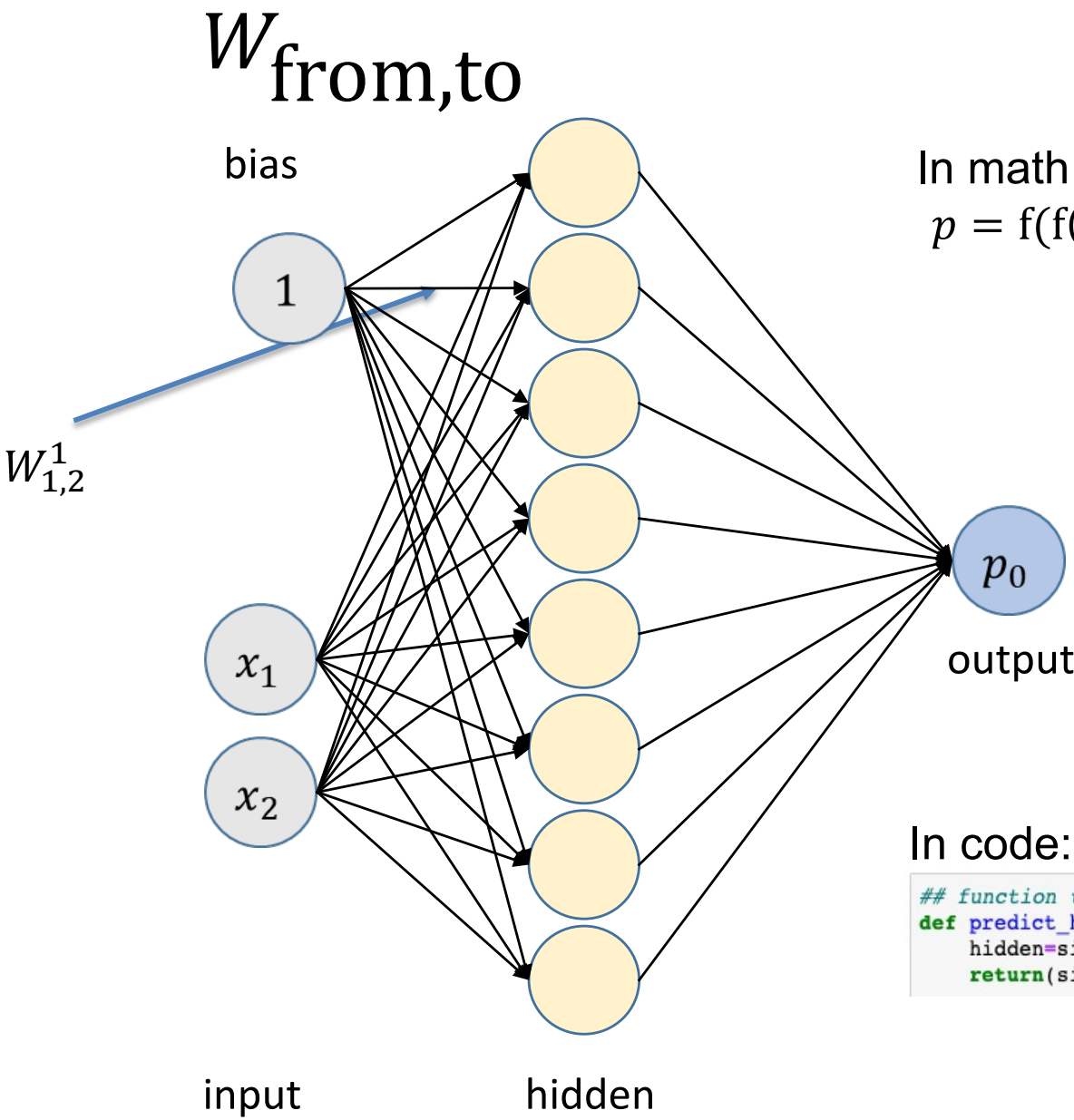
General rule: Networks without hidden layer have linear decision boundary.

http://playground.tensorflow.org/

Hidden Layer



A first deep network



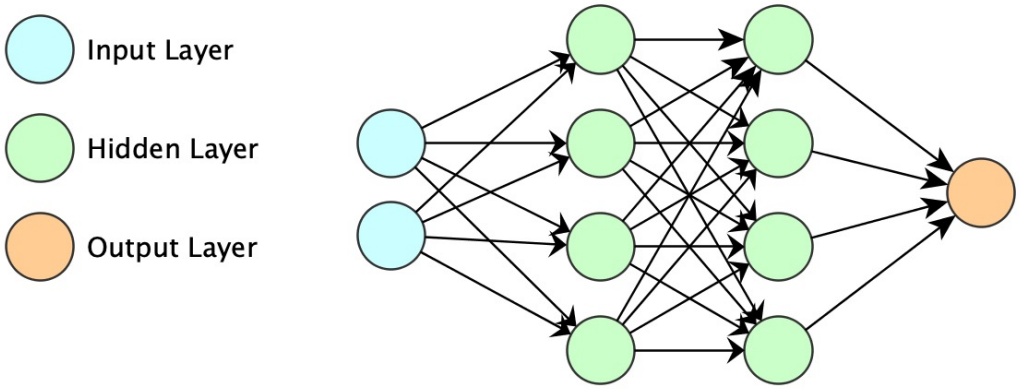
In math (f = sigmoid)

$$p = \mathrm{f}(\mathrm{f}(X \cdot W_1 + b_1) \cdot W_2 + b_2)$$

In code:

```
## function to return the probability output after the hidden layer
def predict_hidden(X):
    hidden=sigmoid(np.matmul(X,W1)+b1)
    return(sigmoid(np.matmul(hidden,W2)+b2))
```

Structure of the network



In code:

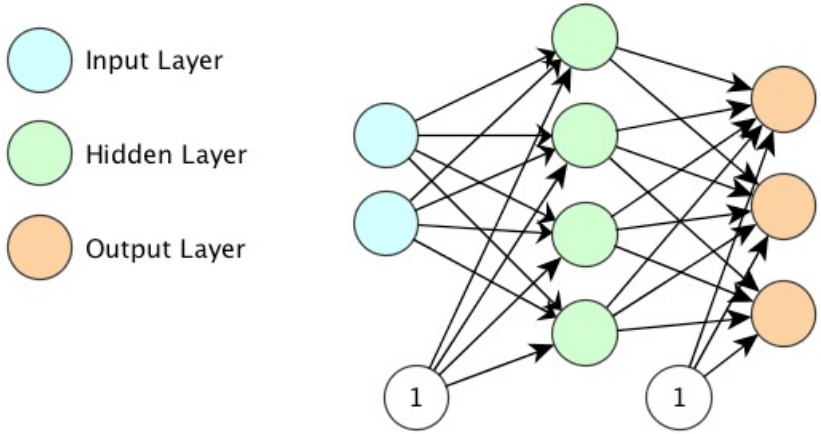
```
## Solution 2 hidden layers
def predict_hidden_2(X):
    hidden_1=sigmoid(np.matmul(X,W1)+b1)
    hidden_2=sigmoid(np.matmul(hidden_1,W2)+b2)
    return(sigmoid(np.matmul(hidden_2,W3)+b3))
```

In math (f = sigmoid) and b1=b2=b3=0

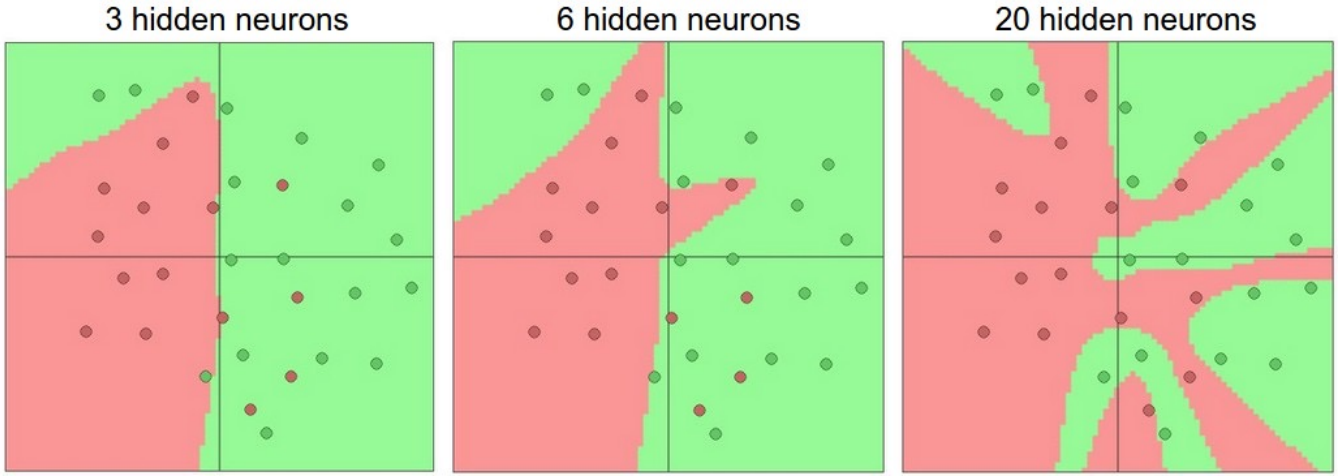
$$p = f(f(f(\mathrm{x}\,\mathrm{W}^1)\mathrm{W}^2))$$

Looks a bit like onions, matryoshka (Russian Dolls) or lego bricks

One hidden Layer

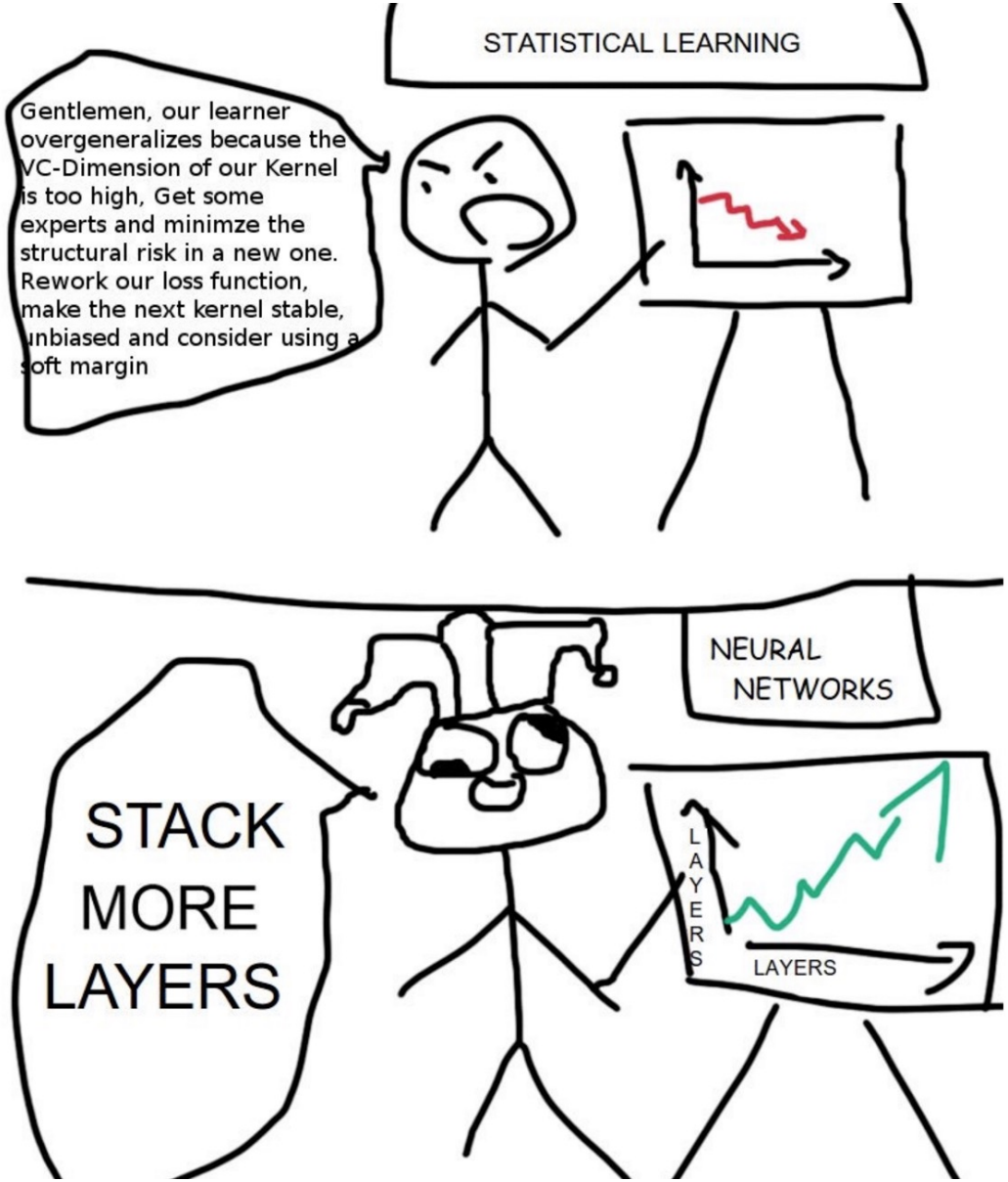


A network with one hidden layer is a universal function approximator!



http://cs231n.github.io/neural-networks-1/

DL vs Statistical Learning Meme



https://www.reddit.com/r/ProgrammerHumor/comments/8c1i45/stack_more_layers/

Training of Neural Networks

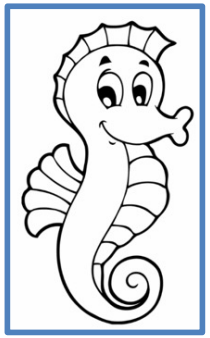
Prinzipielle Funktionsweise: Training Bild Klassifikation

Wahre Klasse

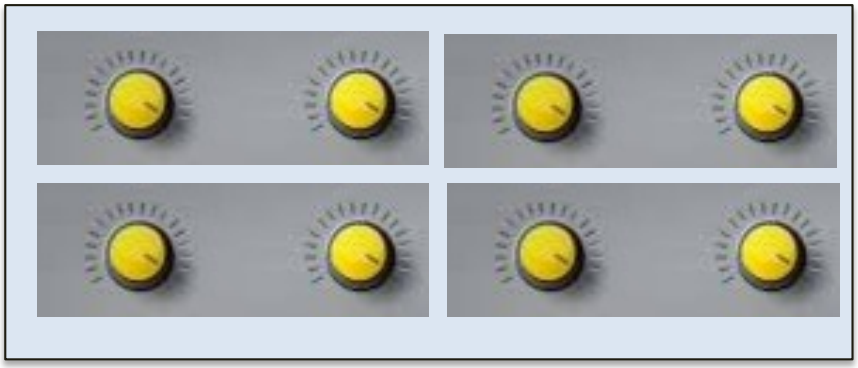
Vorhergesagtes Label

Tiger → Seehund 👎

Tiger → Tiger 👍



Seepferd → Seepferd 👍



...

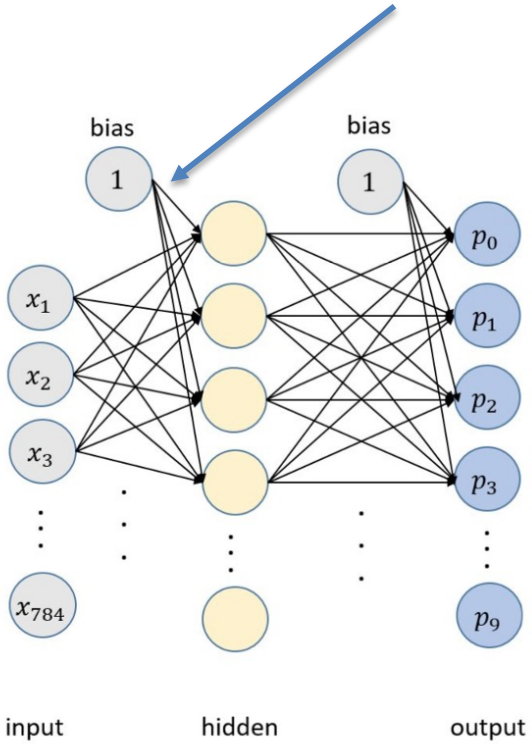
Typisch 1 Mio. Trainingsdaten

Trainingsprinzip:
Parameter werden so eingestellt, dass möglichst wenige Fehler in den Trainingsdaten gemacht werden.

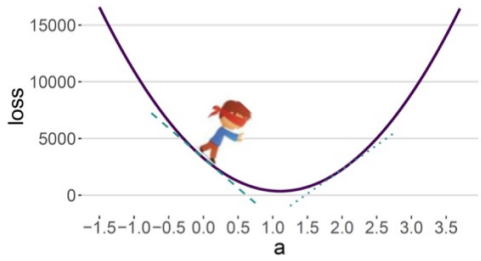
Optimization in ML

- ML many parameters
  - Optimization by gradient descent
- Algorithm
  - Take a batch of training examples
  - Calculate the loss of that batch
  - Tune the parameters so that loss gets minimized

Parameters of the network are the weights.



Again, with Gradient Descent



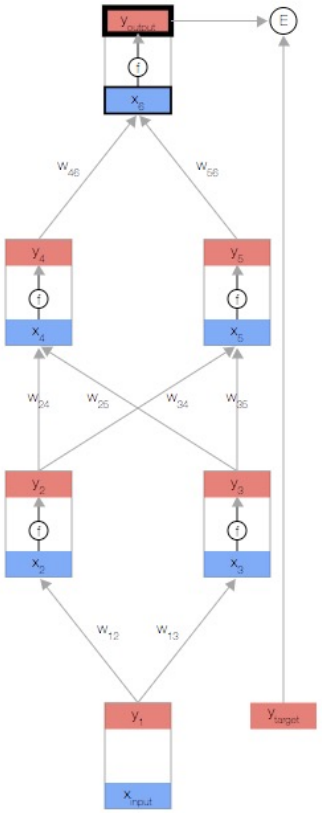
Modern Networks have Billions (10^9) of weights. Record 2020 1.5E9 https://openai.com/blog/better-language-models/

Backpropagation

- There is an efficient way to update all parameters of the network
- This is called Backpropagation
- We need to calculate the derivative of the loss function w.r.t. all weights
- Doing this efficiently (on graphic cards GPU) by hand is tedious
- Enter: Deep Learning Frameworks
  - Make a computational graph from networks
  - Automagical differentiation in that graph (Backprob)

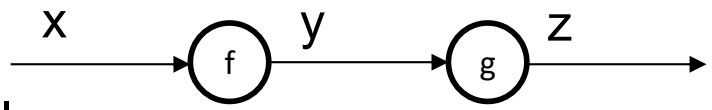
Motivation: The forward and the backward pass

- https://developers-dot-devsite-v2-prod.appspot.com/machine-learning/crash-course/backprop-scroll



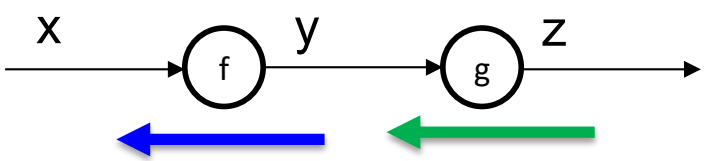
Chain rule recap

- If we have two functions f,g
  $y = f(x)\ and$
  $z = g(y)$
  then y and z are dependent variables.



- The chain rule:
  $$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} \cdot \frac{\partial z}{\partial y}$$

- Backpropagation (flow of the gradient)



$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y}$$

Gradient flow in a computational graph: local junction

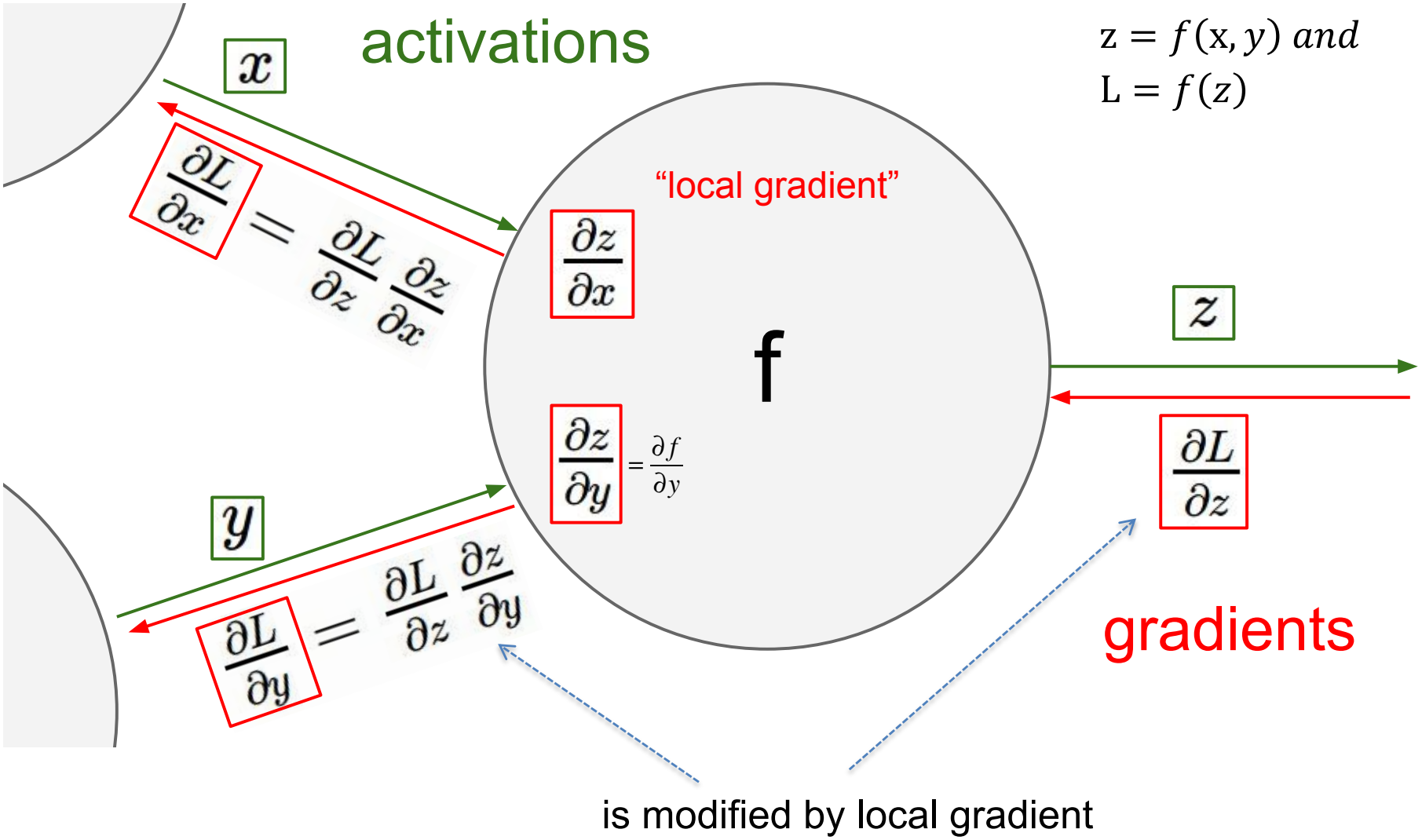
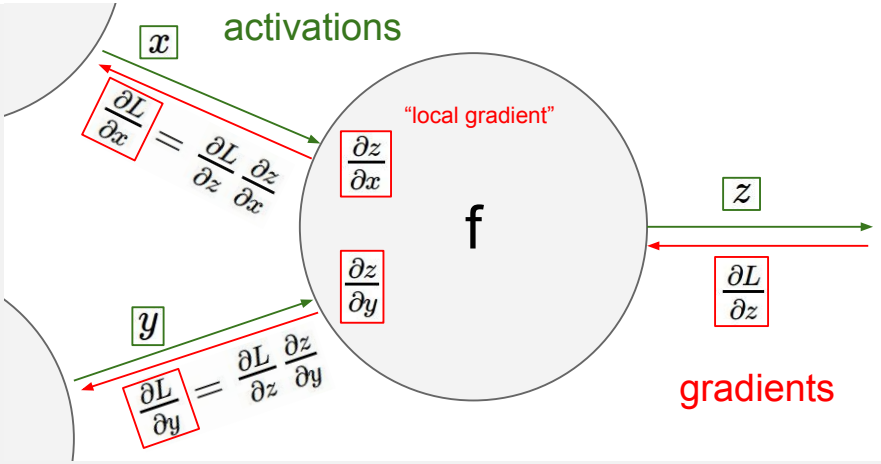


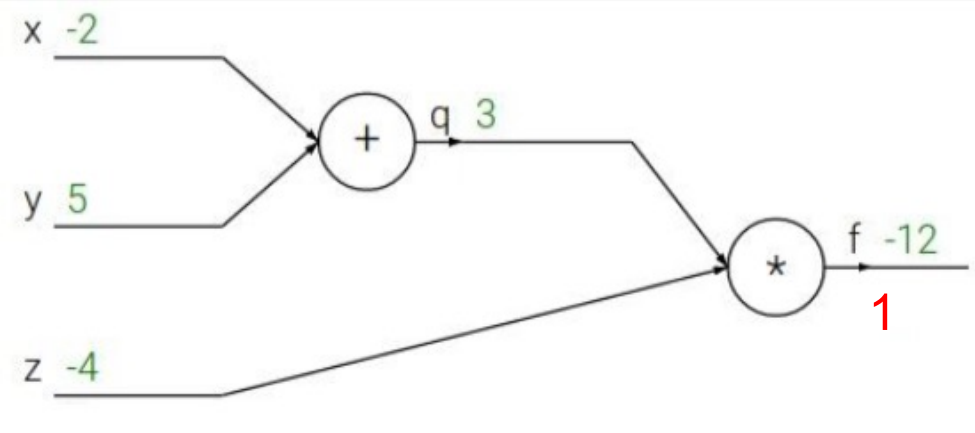
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4





$$\frac{\partial(\alpha+\beta)}{\partial\alpha} = 1 \qquad \frac{\partial(\alpha * \beta)}{\partial\alpha} = \beta$$

➔Multiplication do a switch

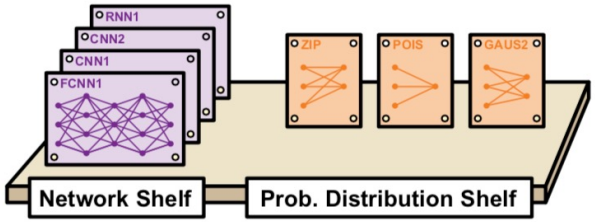
Task: Calculate the derivatives.
Once by hand, once with backpropagation (follow the graph)

Introduction to Keras

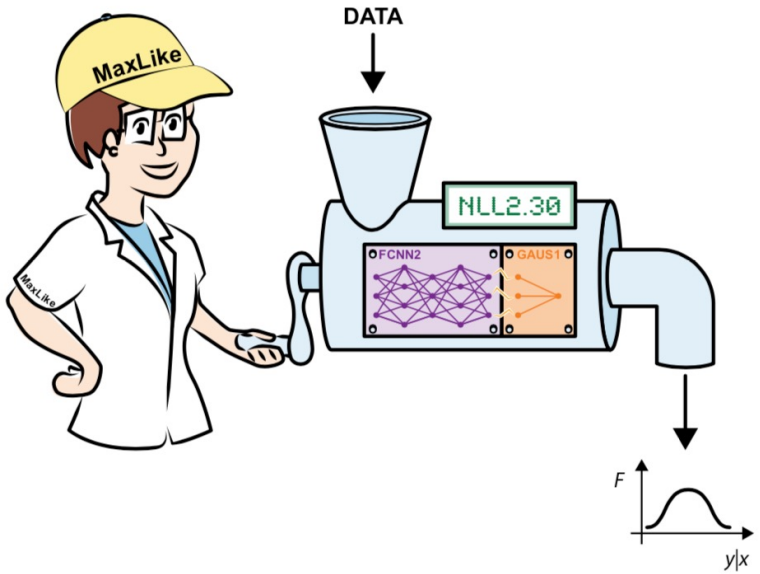
Overview of NN

- The models are fitted to training data with maximum likelihood (or Bayes)

Special networks for x
- Vector FCNN
- Image CNN
- Text CNN/RNN

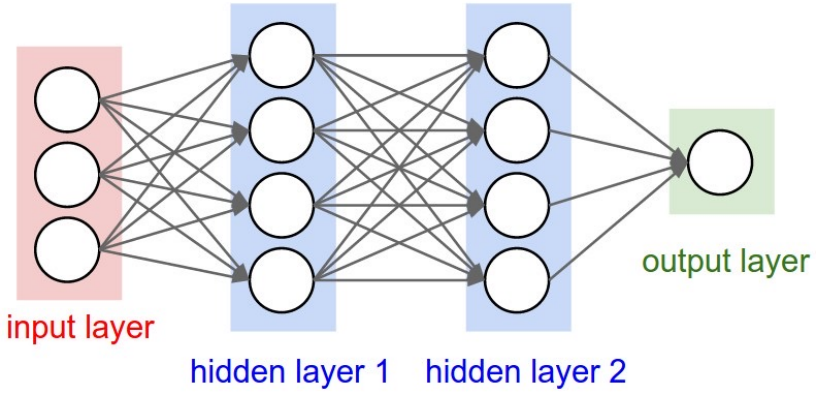


Special heads for y
- Classes
- Regression



Keras as High-Level library to TensorFlow

- There is a multitude of libraries (currently too many!) which help you with training and setting up the networks.
- Main competitors on low-level pytorch and TensorFlow
- We use Keras as high-level library
- Libraries make use of the Lego like block structure of networks



The Keras user experience [marketing]

The Keras user experience

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf

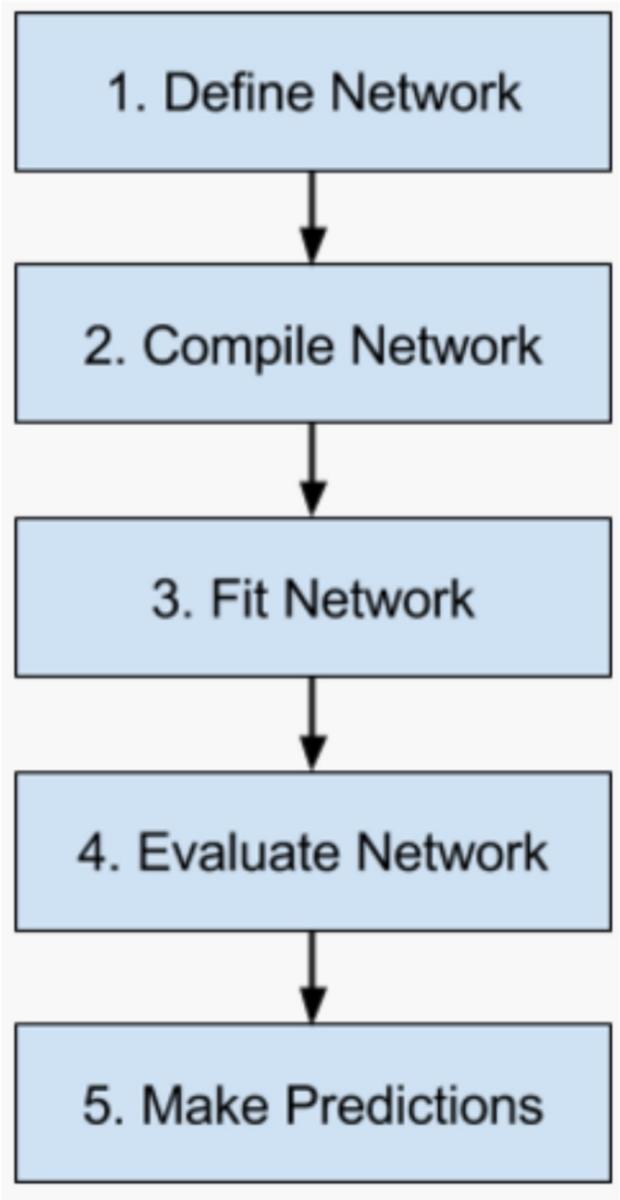
[marketing]

Keras is multi-backend, multi-platform

- Develop in Python, R
  - On Unix, Windows, OSX
- Run the same code with...
  - TensorFlow
  - CNTK
  - Theano
  - MXNet
  - PlaidML
  - ??
- CPU, NVIDIA GPU, AMD GPU, TPU...

https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf

Keras Workflow



Define the network (layerwise)

Add loss and optimization method

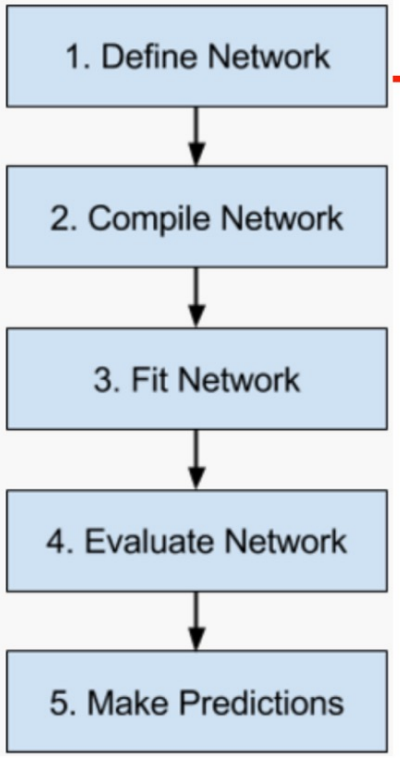
Fit network to training data

Evaluate network on test data

Use in production

A first run through

Define the network



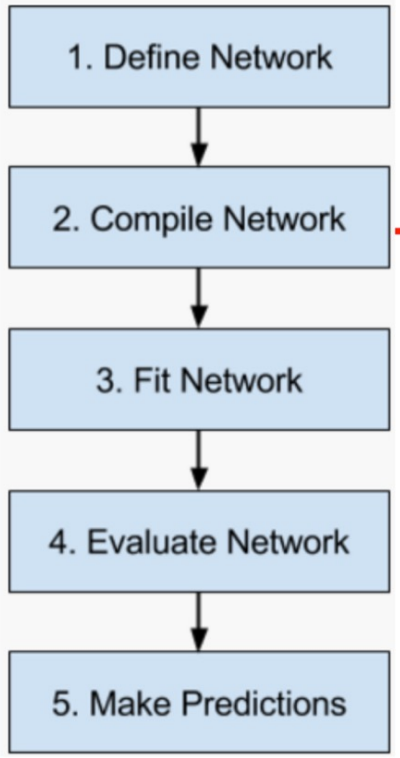
Number of neurons in (first) hidden dense layers (will be input to next layer)

```
model = Sequential()

model.add(Dense(500, batch_input_shape=(None, 784)))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(50))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))
```

Input shape needs to be defined only at the beginning.

Alternative: `input_dim=784`

Compile the network



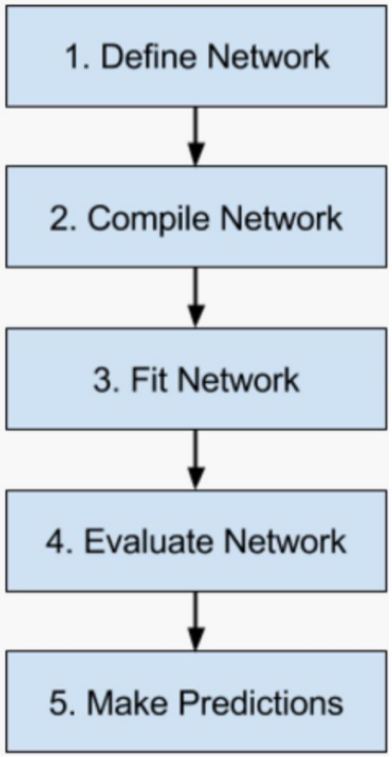
loss function that will be minimized

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

easiest optimizer is SGD
(stochastic gradient descent)

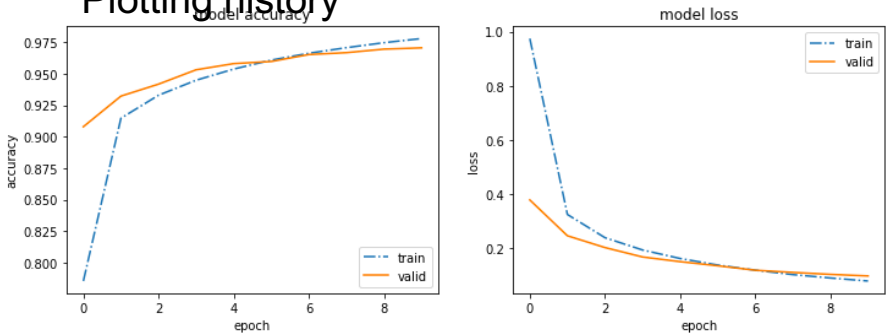
Which metrics besides 'loss' do we want to collect during training

Fit the network

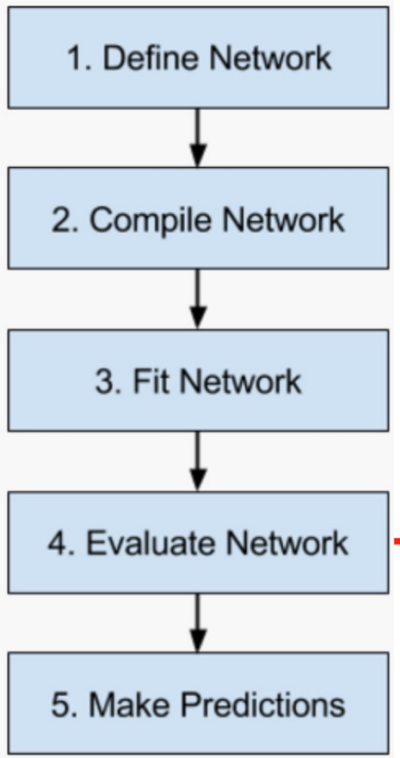



```
# train the model
history=model.fit(X_train_flat, Y_train,
                  batch_size=128,
                  epochs=10,
                  verbose=2,
                  validation_data=(X_val_flat, Y_val)
                 )
```

Plotting history



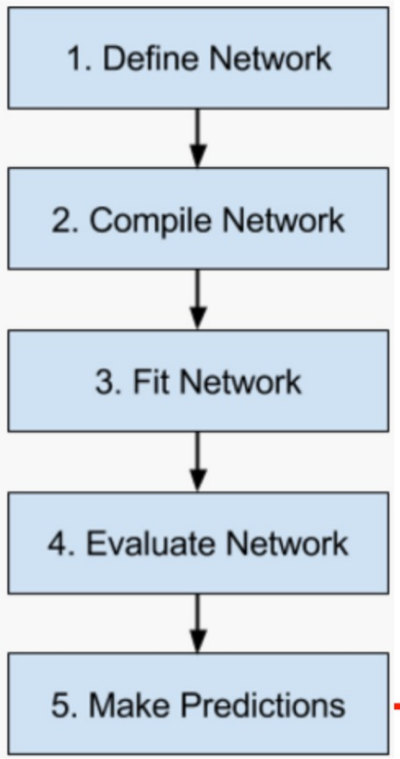
Evaluate the network




```
1 model.evaluate(X_train_sub, Y_train_sub,verbose=2)
```

```
12665/1 - 5s - loss: 0.6775 - accuracy: 0.4677
[0.6865228767974829, 0.4676668]
```

Make Predictions




```
model.predict_classes(X[0:10])
```

```
array([0, 9, 9, 1, 6, 9, 8, 9, 0, 0])
```

Building a network (API Styles)

Three API styles

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases
- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases
- Model subclassing
  - Maximum flexibility
  - Larger potential error surface

Sequential API

The Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional (do you spot the error?)

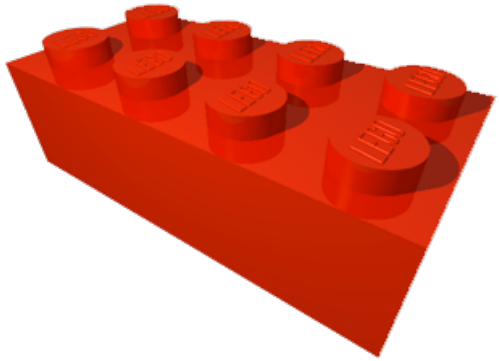
The functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

layers

Dense fully connected

```
keras.layers.core.Dense(
        output_dim,
        init='glorot_uniform',
        activation='linear',
        weights=None,
        W_regularizer=None,
        b_regularizer=None,
        activity_regularizer=None,
        W_constraint=None,
        b_constraint=None,
        input_dim=None)
```



https://keras.io/layers/

More layers

- Dropout
  - keras.layers.Dropout
- Convolutional (see lecture on CNN)
  - keras.layers.Conv2D
  - keras.layers.Conv1D
- Pooling (see lecture on CNN)
  - keras.layers.MaxPooling2D
- Recurrent (See Lecture on RNN)
  - keras.layers.SimpleRNNCell
  - keras.layers.GRU
  - keras.layers.LSTM

- Roll your own:
  - Implement keras.layers.Layer class
  - https://keras.io/layers/writing-your-own-keras-layers/

Activation

```
keras.layers.Activation(activation)
```

Applies an activation function to an output.

Arguments

e.g. 'sigmoid', 'softmax', 'tanh', 'relu',...

- **activation**: name of activation function to use (see: activations), or alternatively, a Theano or TensorFlow operation.

```
from keras.layers import Activation, Dense

model.add(Dense(64))
model.add(Activation('tanh'))
```

This is equivalent to:

```
model.add(Dense(64, activation='tanh'))
```

Note: Activations are also layers

Output Layer

The last layer of the network is a bit special

- Classification
  - # of nodes = # of classes
    - For binary classification sometime only probability of class 1 is reported
  - Usually output is probability for class
    - Use softmax in that case

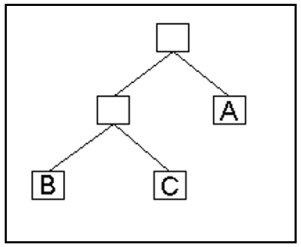
- Regression (simple distributions)
  - In the interpretation the output of a NN is a probability
  - #nodes = 1
  - Gaussian with fixed variance (the usual regression)
  - Poisson (count data) see later

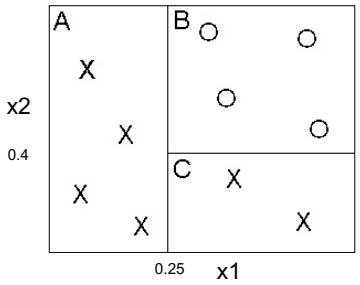
- Regression (more complicated distributions)
  - #nodes = #number of parameters for distribution

End of Machine Learning

What did we not cover

- Here only supervised learning
- Strong Focus on “neural network like” classifiers
  - Linear Regression → Logistic Regression → Fully Connected NN
- Other classifiers
  - Trees





  - Ensemble (of Trees)
    - Random Forest
    - Boosting
  - Support Vector Machines