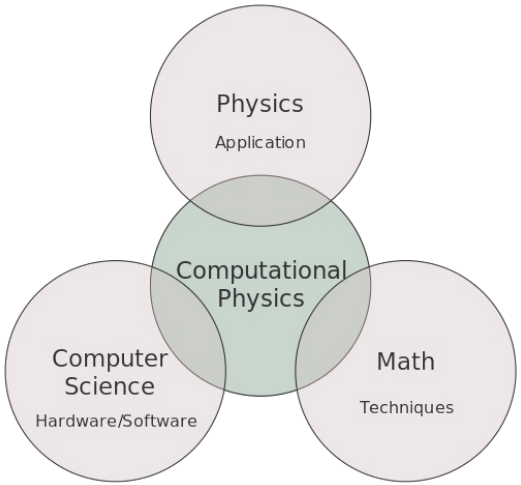
Machine Intelligence:: Deep Learning
Week 1

Oliver Dürr, Beate Sick, Lilach Goren, Pascal Bühler

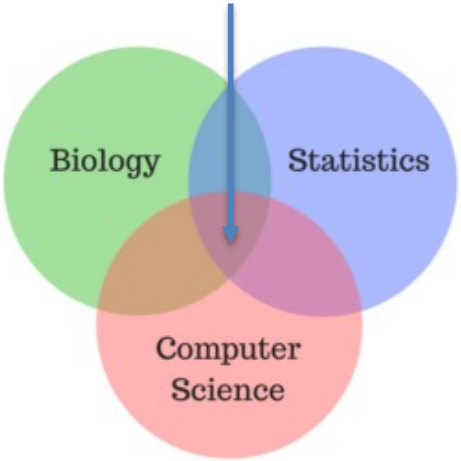
Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Oliver's Background

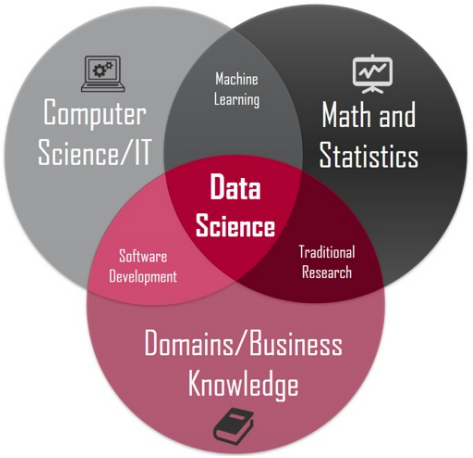
Computational Physics



Bioinformatics



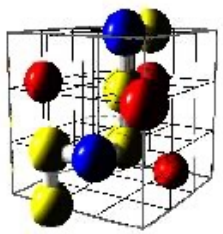
Data Science

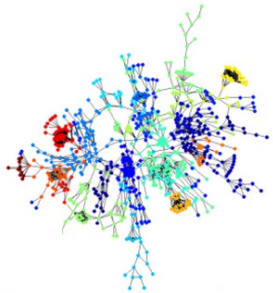


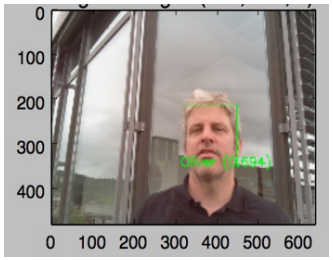
1990's
Uni-Konstanz

2000's
Genedata Basel

2010's
ZHAW Winterthur HTWG Konstanz







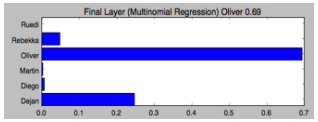


Image credits: Wikipedia, http://omgenomics.com/what-is-bioinformatics/ https://www.fox.temple.edu/institutes-and-centers/data-science/

Tell us something about you

- Computer Science Background
  - Fluent in python?
- Statistics / Math
  - Who visited CAS StMo (statistisches Modellieren)?
  - What is a distribution?
  - Vector times Matrix?
    - Please make sure to check https://tensorchiefs.github.io/dl_course_2025/prerequistites.html
- Any contacts with deep learning yet?

Technical details for this course

- Course website
  - https://tensorchiefs.github.io/dl_course_2025/
- Deep Learning with Python, Second Edition
  - https://www.manning.com/books/deep-learning-with-python-second-edition
- Keras Documentation:
  - https://keras.io/
- Probabilistic Deep Learning: Our probabilistic take
  - https://www.manning.com/books/probabilistic-deep-learning
- Other Courses
  - Convolutional Neural Networks for Visual Recognition http://cs231n.stanford.edu

Organizational Issues: ~~Test~~ Projects

- Projects (2-3 People)

- Presented on the last day
  - Spotlight talk (5 Minutes)
  - Poster
- Topics
  - You can / should choose a topic of your own (please discuss your topic with us by week4 latest)

  - Possible Topics (see website)
    - Take part in a Kaggle Competition (e.g. Leaf Classification / Dogs vs. Cats)
    - Music classification
    - Polar bear detection
    - ...

- Computing: colab, laptop (or cloud computing)

Outline of the DL Module (tentative)

The course is split in 8 sessions, each 4 lectures long. Topics might be adapted during the course

Day	Date	Time	Topic
1	15.04.2025	09:00-12:30	Introduction to Deep Learning & Keras, first NNs
-	21.04.2025	-	FRÜHLINGS-FERIEN
-	28.04.2025	-	FRÜHLINGS-FERIEN
2	06.05.2025	09:00-12:30	Loss, Optimization, Regression, Classification
3	13.05.2025	09:00-12:30	Computer vision, CNN-archictecture
4	20.05.2025	09:00-12:30	DL in practice, pretrained (foundation) models
5	27.05.2025	09:00-12:30	Model evaluation, baselines, xAI, troubleshooting
6	03.06.2025	09:00-12:30	Generative Models, Transformer-architecture
7	10.06.2025	09:00-12:30	Vision Transformer
8	17.06.2025	09:00-12:30	Projects, deep Ensembling

Learning Objectives for today

- Get a rough idea what the DL is about
- Framework
  - Introduction to Keras

Introduction to Deep Learning --what's the hype about?

Machine Perception

- Computers have been quite bad in things which are easy for humans (images, text, sound)
- A Kaggle contest 2012
- In the following we explain why

Kaggle dog vs cat competition

Deep Blue beat Kasparov at chess in 1997.
Watson beat the brightest trivia minds at Jeopardy in 2011.
Can you tell Fido from Mittens in 2013?

Deep Learning vs. Machine Learning

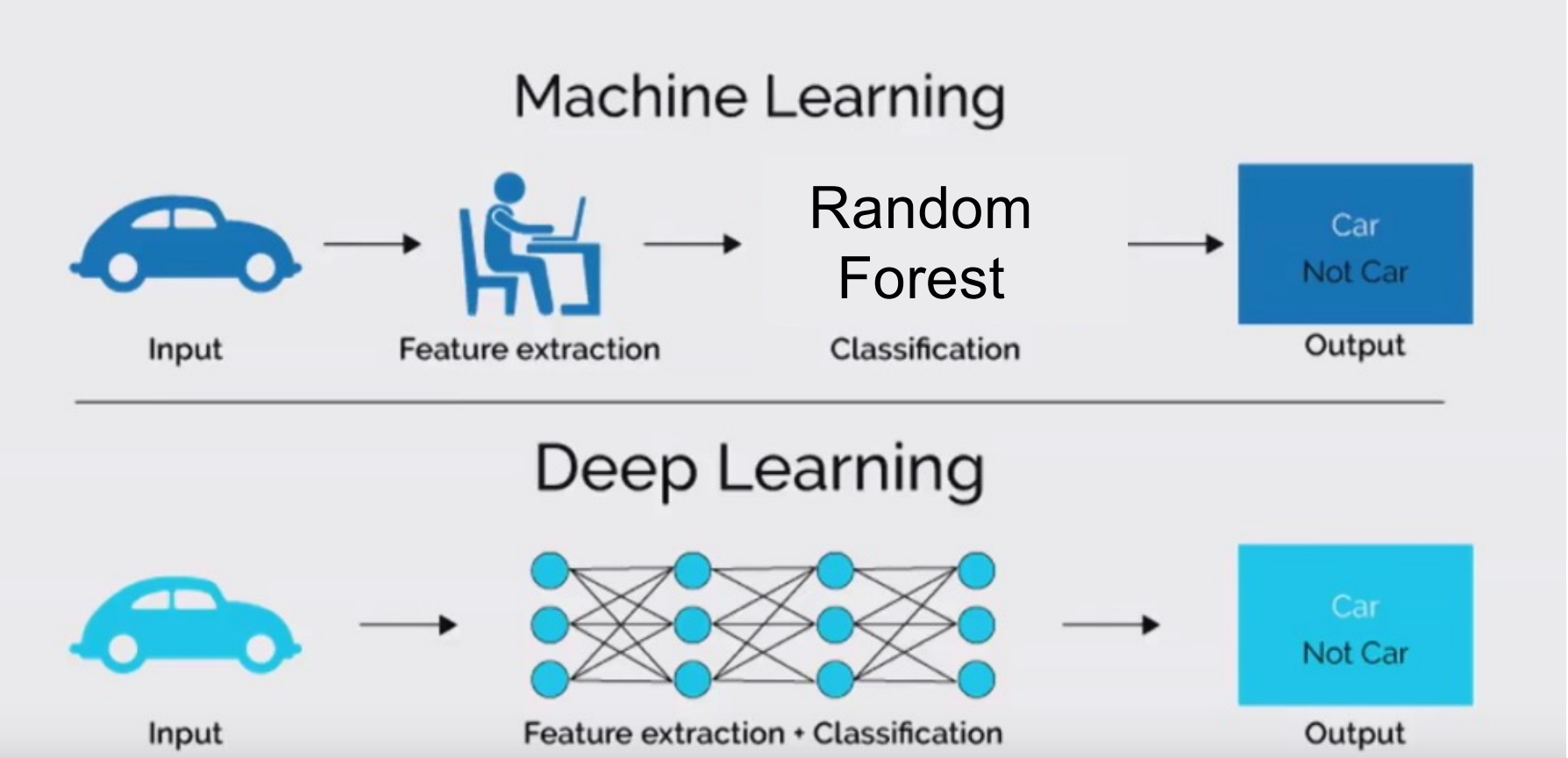
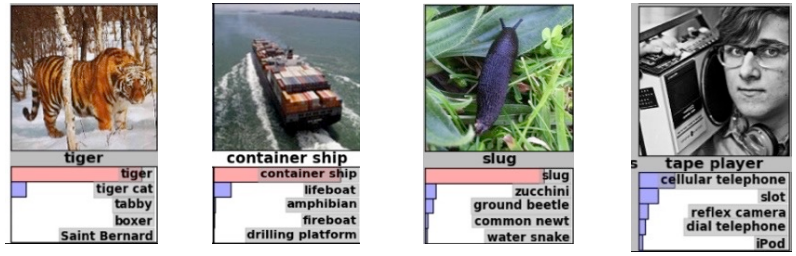


image credit: adapted from Siraj Raval

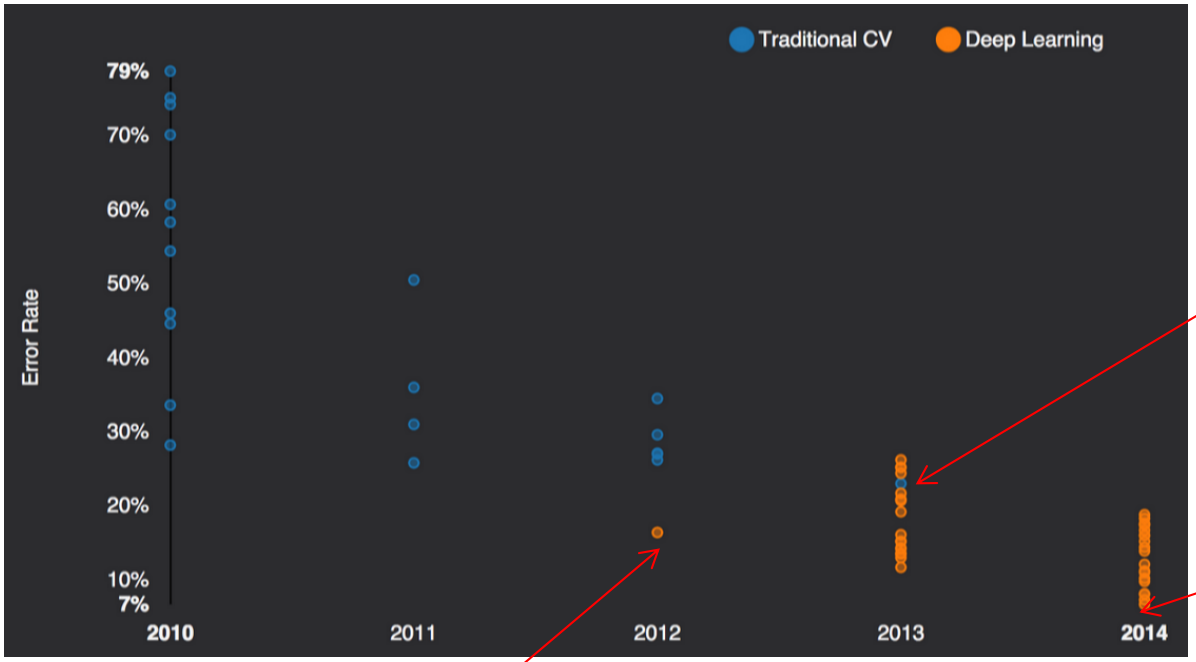
The most convincing case for DL (subjective view)

Why DL: Imagenet 2012, 2013, 2014, 2015

1000 classes
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN approach in 2013

GoogLeNet 6.7%

A. Krizhevsky
first CNN in 2012
Und es hat zoom gemacht

2015: It gets tougher

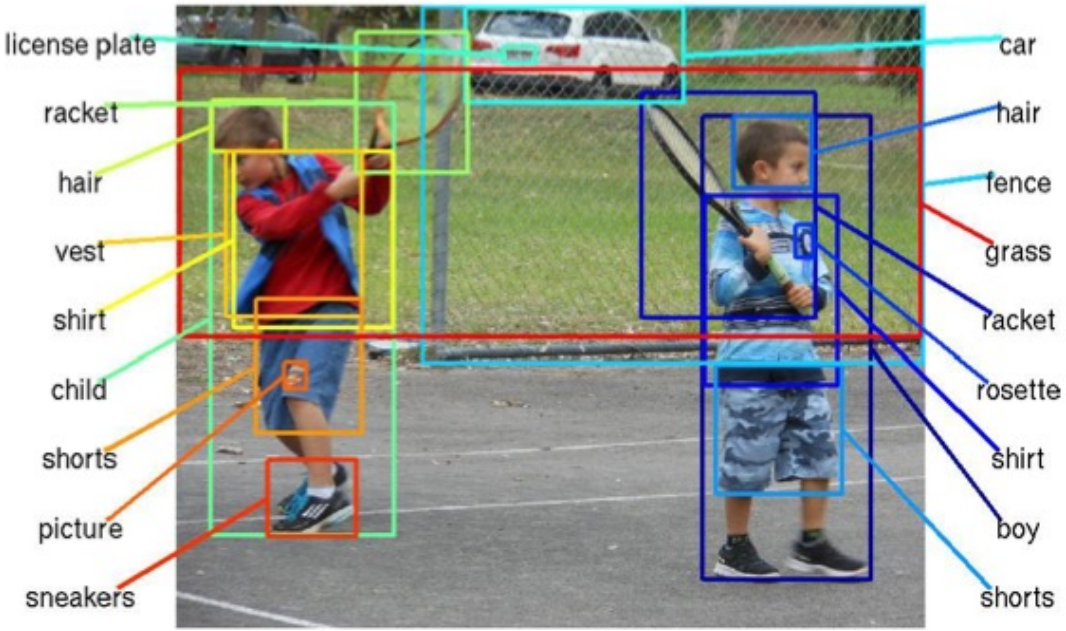
- 4.95% Microsoft (Feb 6 surpassing human performance 5.1%)
- 4.8% Google (Feb 11) -> further improved to 3.6 (Dec)?
- 4.58% Baidu (May 11 banned due too many submissions)
- 3.57% Microsoft (Resnet winner 2015) → task solved!

Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

The computer vision success story

- With DL it took approx. 3 years to solve object detection and other computer vision task

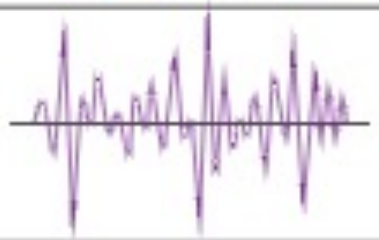
Deep Blue beat Kasparov at chess in 1997.
Watson beat the brightest trivia minds at Jeopardy in 2011.
Can you tell Fido from Mittens in 2013?



"man in black shirt is playing guitar."

Images form cs229n

Use cases of deep learning

Input x to DL model	Output y of DL model	Application
Images 	Label "Tiger"	Image classification
Audio 	Sequence / Text "see you tomorrow"	Voice recognition
Sequence of tokens representing e.g. "Hallo, wie gehts?"	Next token	Translation text generation, QA
Sequence of tokens representing e.g. This movie was rather good	Label (Sentiment) positive	Sentiment analysis

Deep learning models can „see“, „hear“, „read“, „write“ .

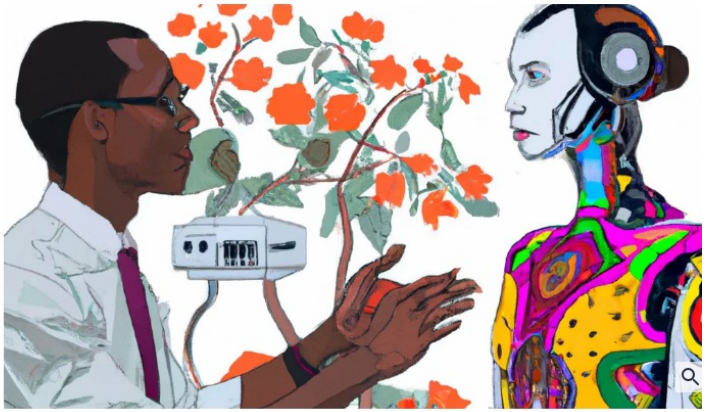
Last breakthrough 2022: Large Language Models (LLMs) like ChatGPT

This is the new shit: LLM/ChatGPT

Marilyn Manson - This Is The New Shit (Official Music Video)

Die gefühlte Revolution

4. Dezember 2022, 18:51 Uhr | Lesezeit: 3 min



Das kommt heraus, wenn man der künstlichen Intelligenz Dall-E die Anweisung gibt: "Ein Roboter lässt beim Turing-Test einen Menschen glauben, dass sie ein Mensch ist, im Stil von Kehinde Wiley." (Foto: Dall-E-Bild: SZ)

GPT (short for "**Generative Pre-training Transformer**")

is a type of language processing AI model developed by OpenAI. It is a large, deep learning model that has been trained on a diverse range of texts and can generate human-like text when given a prompt.

https://www.slideshare.net/SilentShikari/what-is-chat-gpt

First Neural Network

The Single Neuron: Biological Motivation

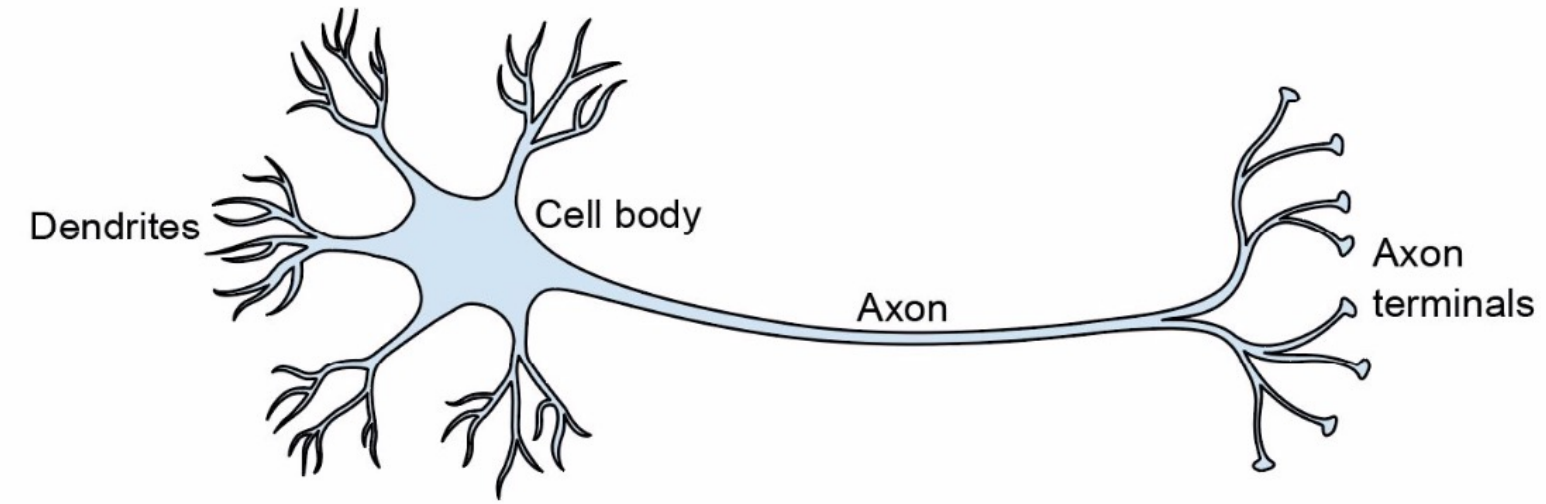
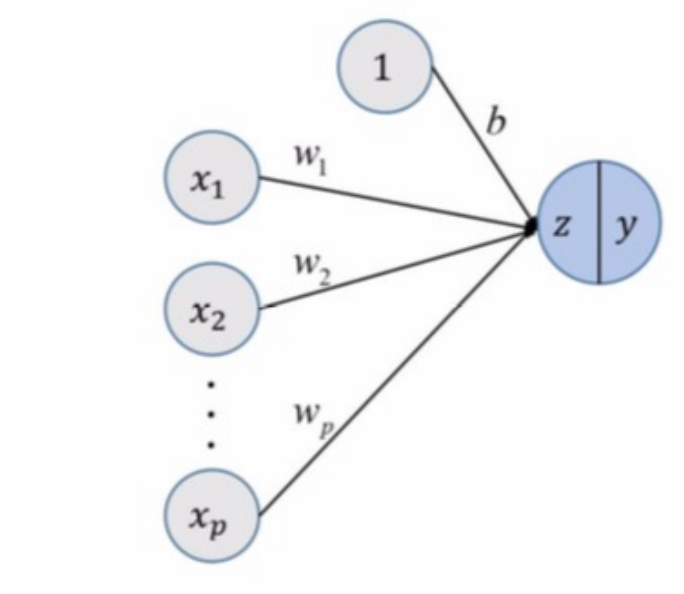


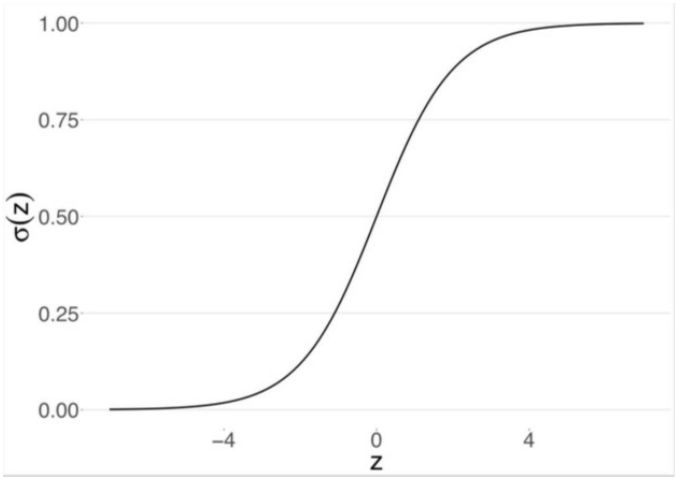
Figure 2.2 A single biological brain cell. The neuron receives the signal from other neurons via its dendrites shown on the left. If the cumulated signal exceeds a certain value, an impulse is sent via the axon to the axon terminals, which, in turn, couples to other neurons.

Neural networks are **loosely** inspired by how the brain works

The Single Neuron: Mathematical Abstraction



Sigmoid Activation

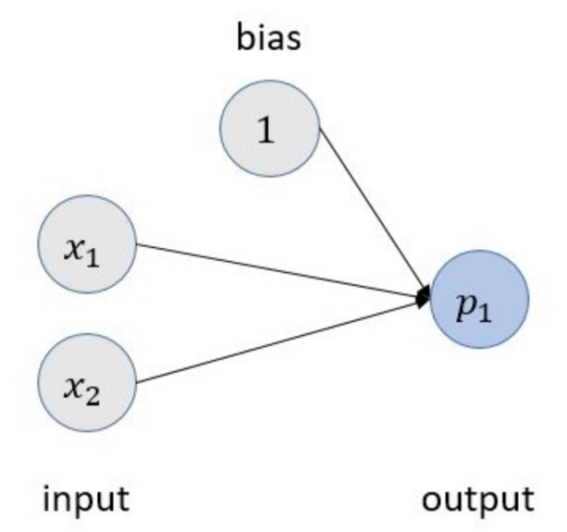


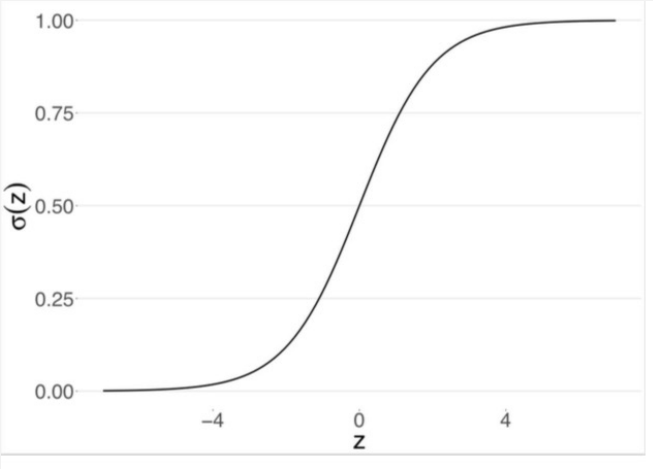
$$z = b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots x_p \cdot w_p$$

$$y = \sigma(z) = \sigma\left(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{ip} x_{ip}\right) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{ip} x_{ip})}}$$

The output after the sigmoid activation can be interpreted as probability for y=1

Exercise: Part 1





Model: The above network models the **probability** p_1 that a given banknote is false.

TASK (with pen and paper)

The weights (determined by a training procedure later) are given by

$$w_1 = 0.3, w_2 = 0.1, \text{ and } b = 1.0$$

What is the probability that a banknote, that is characterized by x_1=1 and $x_2 = 2.2$, is a faked banknote?

GPUs love Vectors

$$F^{\mu\nu}$$

In Math:

$$p_1 = \text{sigmoid}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + b \right)$$

In code:

```
## function to return the probability output after the matrix multiplication
def predict_no_hidden(X):
    return sigmoid(np.matmul(X,W)+b)
```

Toy Task

- Task tell fake from real banknotes
- Banknotes described by two features

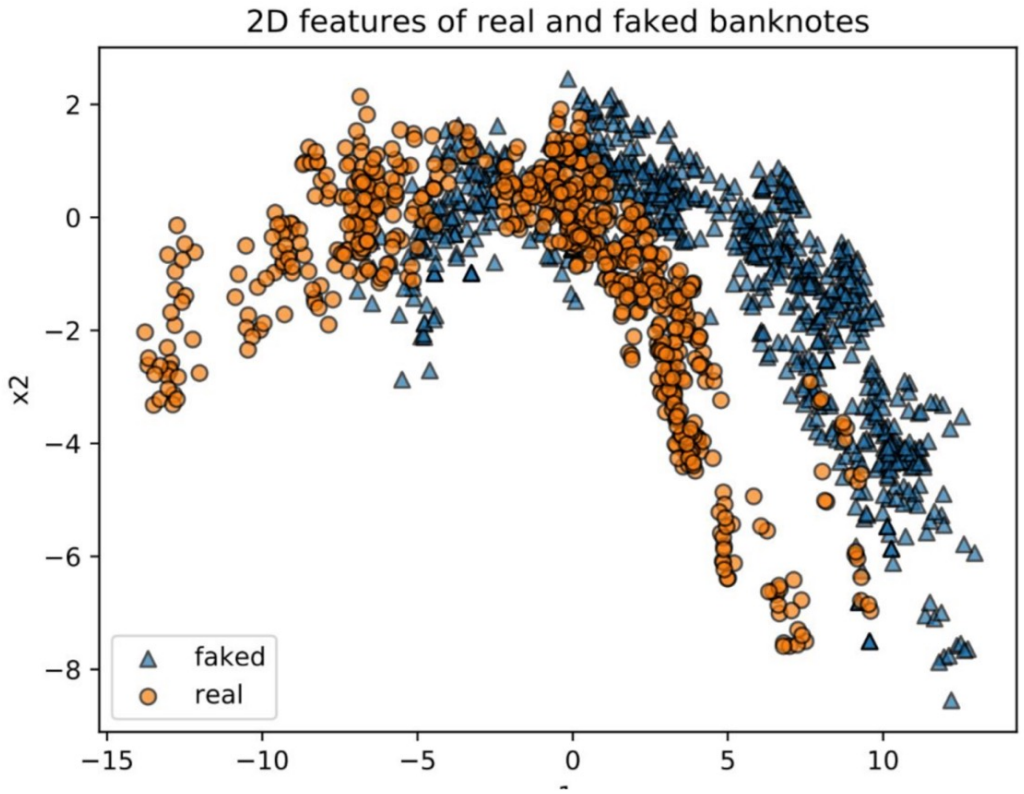
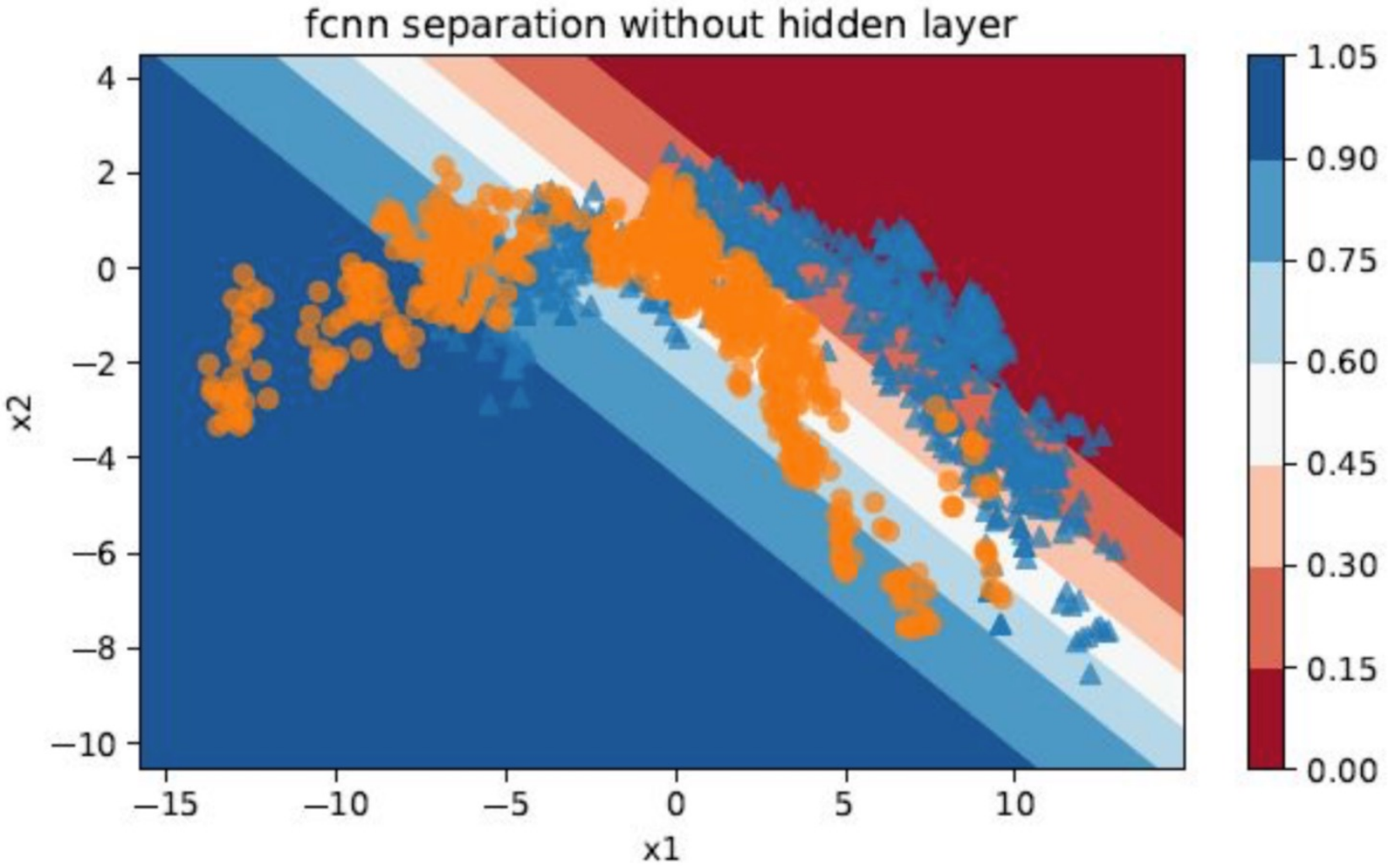


Figure 2.5 The (training) data points for the real and faked banknotes

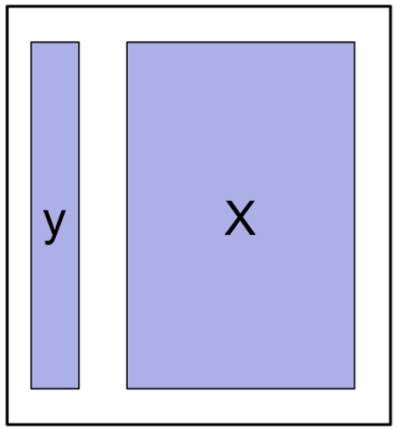
Result (see later in the notebook)



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

Our take on Deep Learning: Probabilistic Viewpoint

Tasks in supervised DL



Supervised Learning

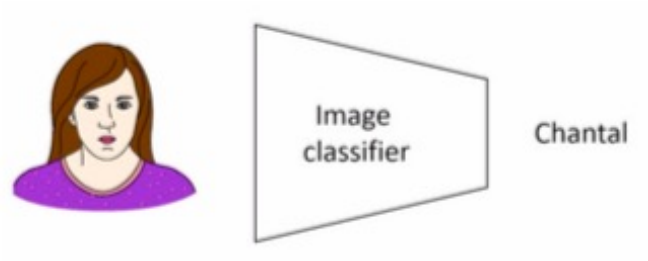
- 2 Main tasks in DL predict y given x
  - **Classification**
    - Point prediction: Predict a class label
    - Probabilistic prediction:
      predict a discrete probability distribution over all possible class labels
  - Regression
    - Point prediction: Predict a number
    - Probabilistic prediction:
      predict a continuous probability distributions over the possible Y value range
- The loss function depends on the task

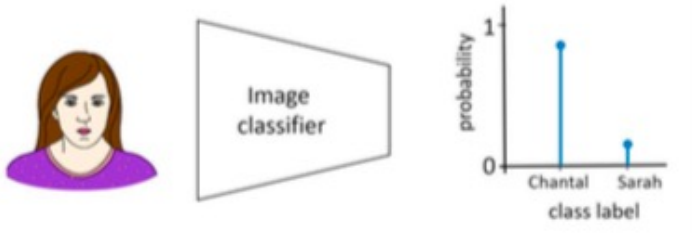
Probabilistic vs deterministic models

Deterministic

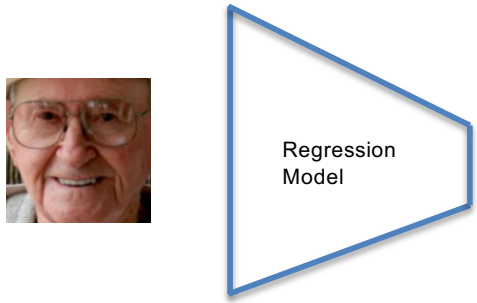
Probabilistic

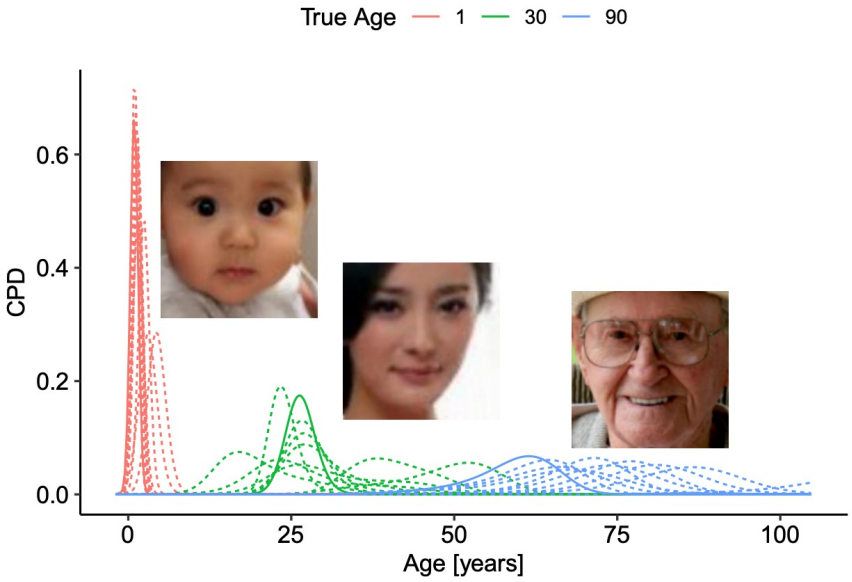
"Classification"





"Regression"





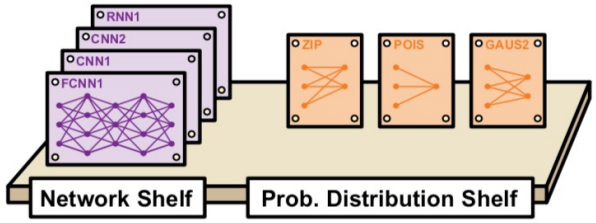
Conditional probability distribution (CPD)

$p(y|x)$

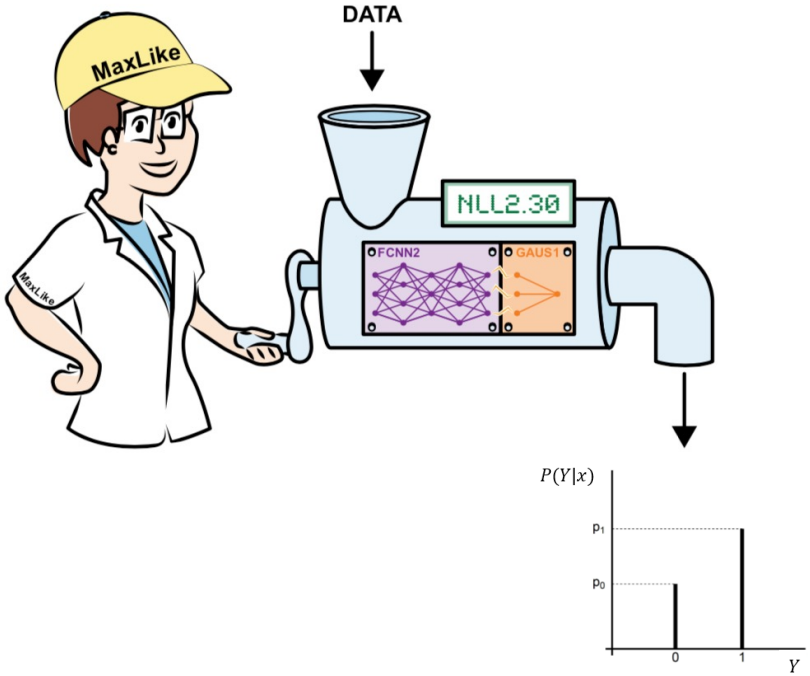
Guiding Theme of the course

- We treat DL as *probabilistic models*, as statistical model (logistic regression, ...) to predict the conditional probability distribution $P(Y|x)$ for the outcome
- The models are fitted to training data with maximum likelihood (or Bayes)

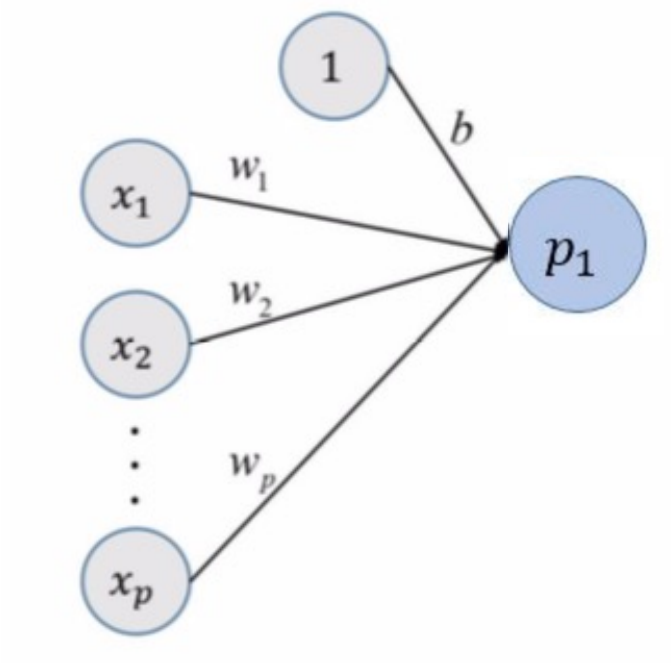
Special networks for x
- Vector FCNN
- Image CNN
- Text CNN/RNN



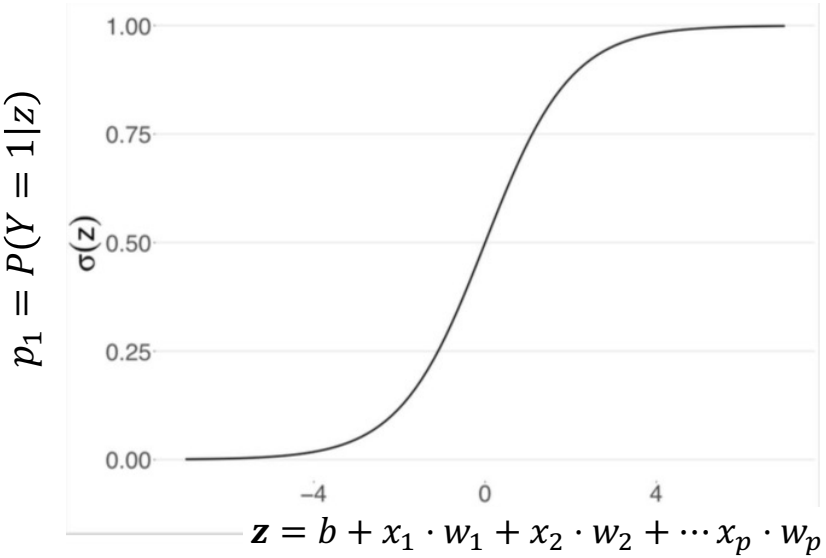
NN heads tailored for Y
- Probabilistic classification
- Probabilistic regression



Logistic regression in DL view



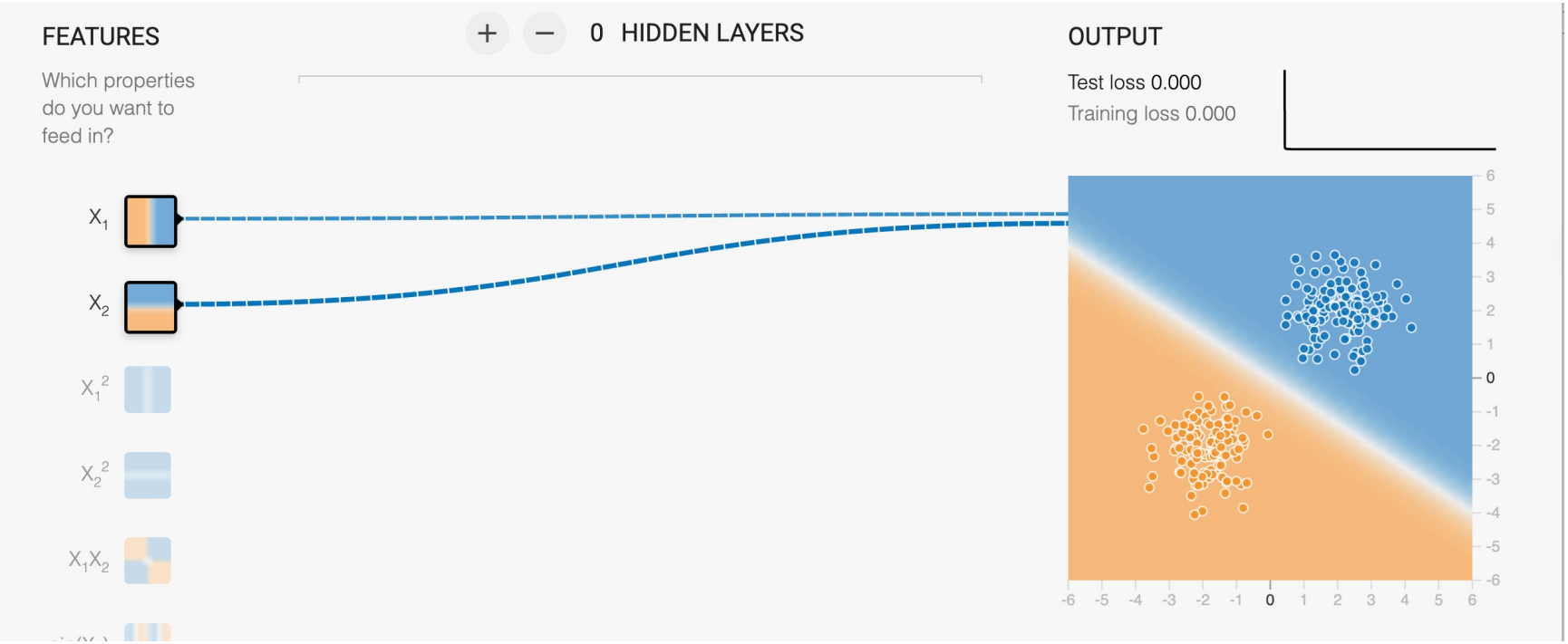
Sigmoid Activation



$$p_1 = P(Y = 1|z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

A NN for a binary outcome with only 1 neuron with sigmoid-activation (and no hidden layer) is nothing else than logistic regression!

Experiment



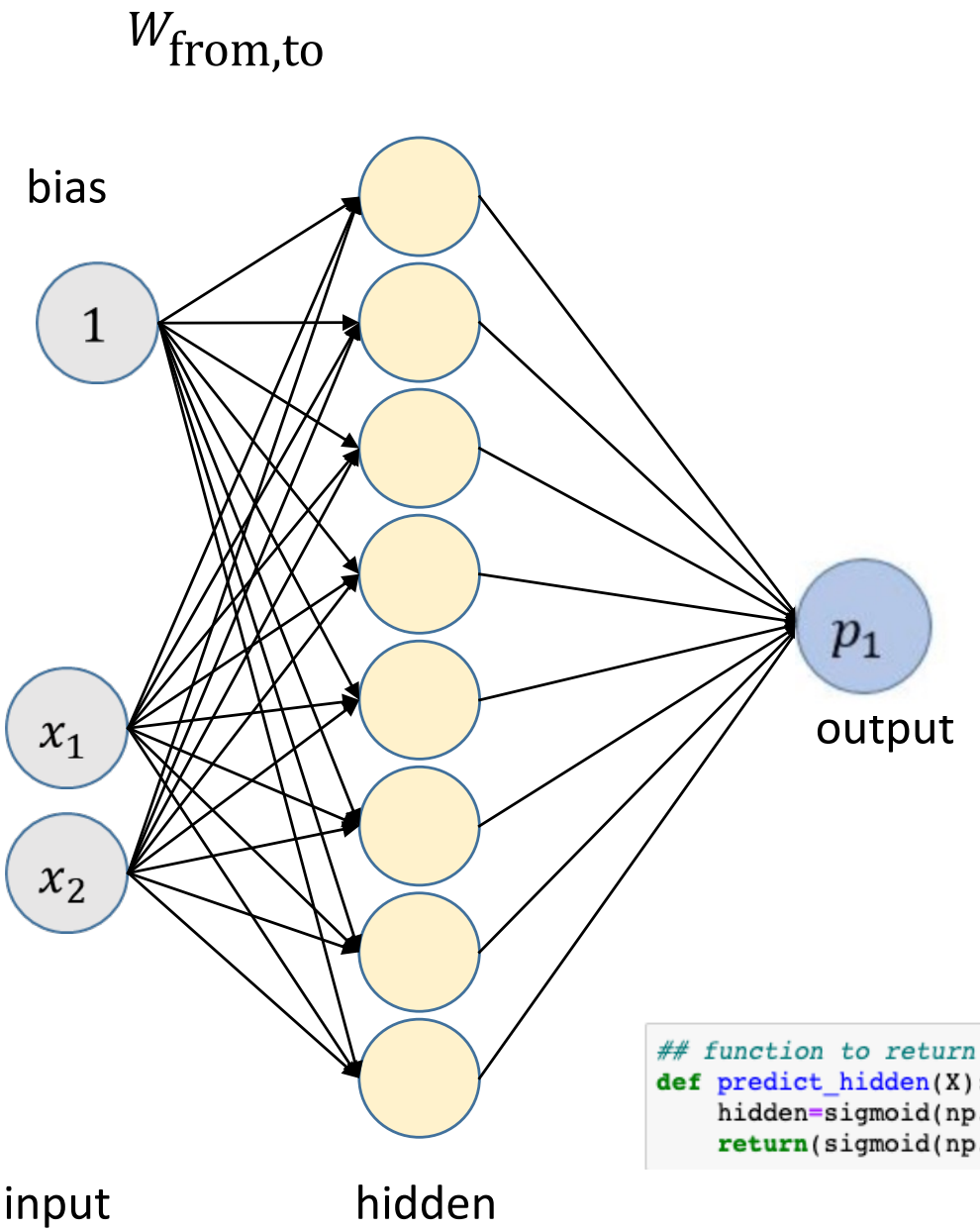
Demo show Hidden Layers

http://playground.tensorflow.org

Let's you explore the effect of hidden layers


WE NEED TO GO
DEEPER
memegene

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))
```

To go deep non-linear activation functions are needed

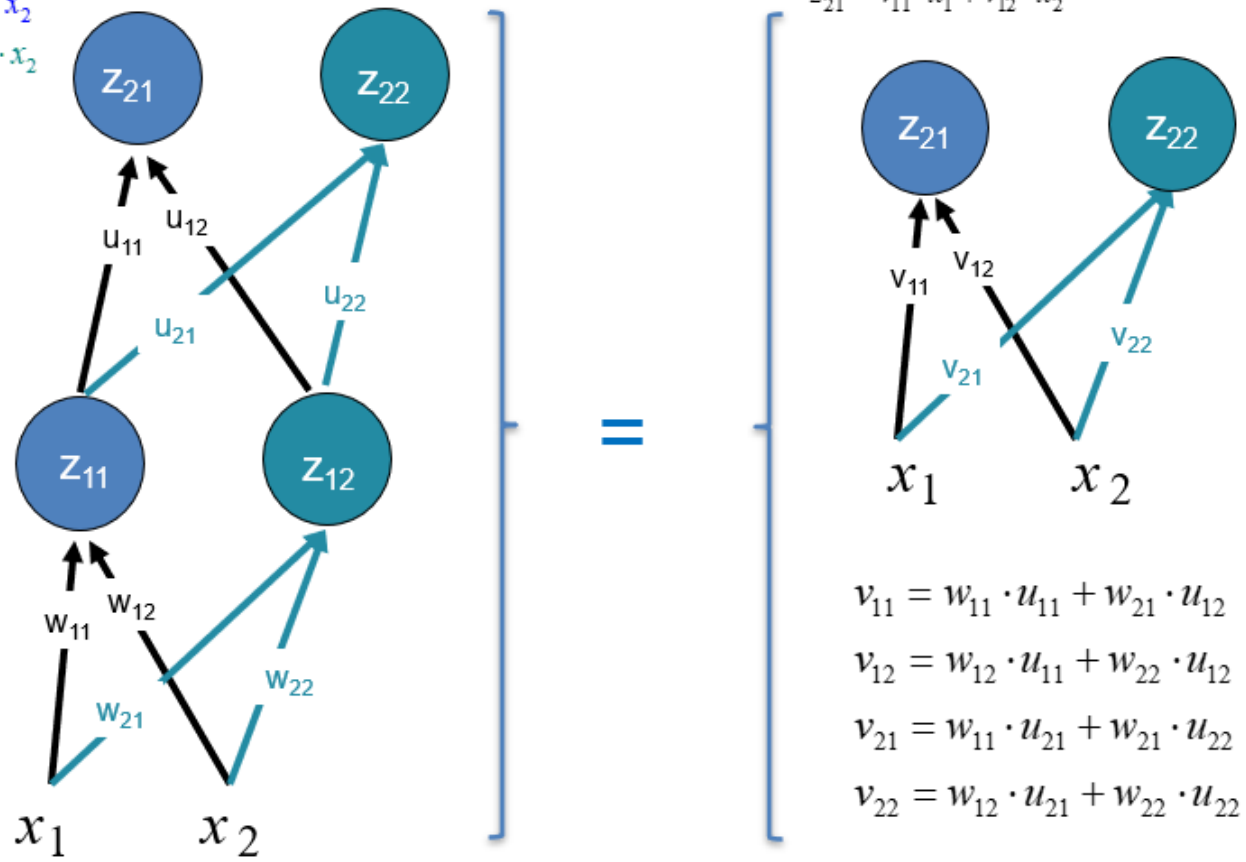
2 linear layers can be replaced by 1 linear layer -> can't go deep with linear layers!

$$z = (x \cdot W) \cdot U = x \cdot (W \cdot U) = x \cdot V$$

$$z_{21} = z_{11} \cdot u_{11} + z_{12} \cdot u_{12} = (w_{11} \cdot x_1 + w_{12} \cdot x_2) \cdot u_{11} + (w_{21} \cdot x_1 + w_{22} \cdot x_2) \cdot u_{12}$$
$$= x_1 \cdot (w_{11} \cdot u_{11} + w_{21} \cdot u_{12}) + x_2 \cdot (w_{12} \cdot u_{11} + w_{22} \cdot u_{12})$$

$z_{11} = w_{11} \cdot x_1 + w_{12} \cdot x_2$

$z_{12} = w_{21} \cdot x_1 + w_{22} \cdot x_2$



$$v_{11} = w_{11} \cdot u_{11} + w_{21} \cdot u_{12}$$
$$v_{12} = w_{12} \cdot u_{11} + w_{22} \cdot u_{12}$$
$$v_{21} = w_{11} \cdot u_{21} + w_{21} \cdot u_{22}$$
$$v_{22} = w_{12} \cdot u_{21} + w_{22} \cdot u_{22}$$

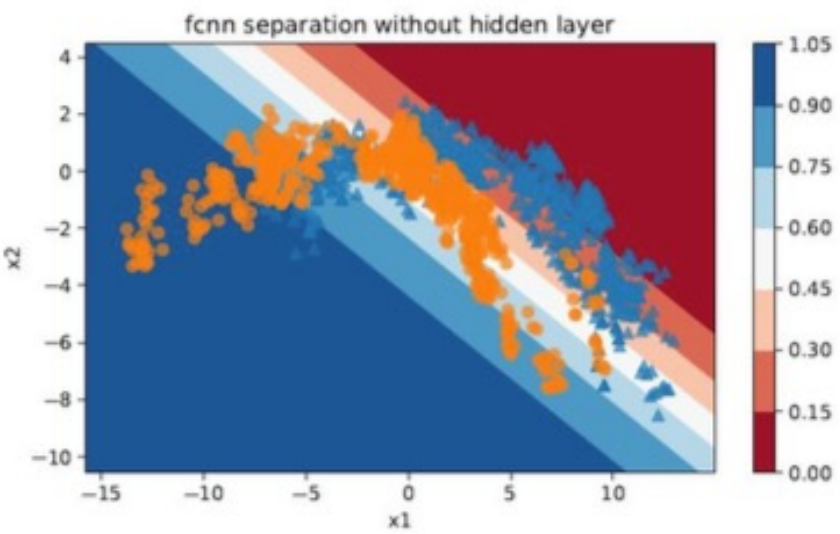
Remark: biases are ignored here, but do not change fact

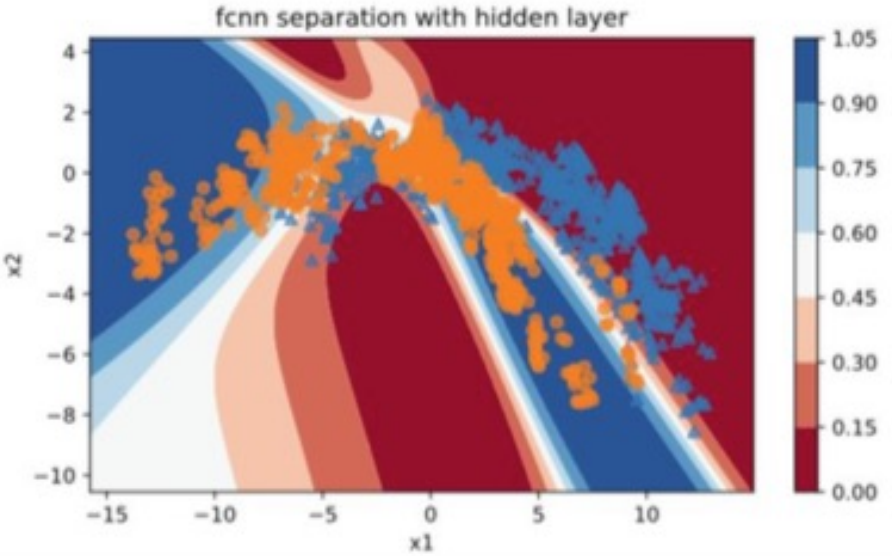
Exercise run your first network

Open Notebook: **01_simple_forward_pass_keras_torch.ipynb**

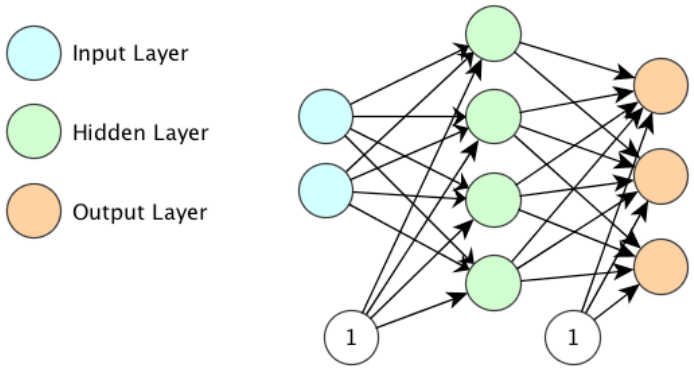
Follow the instructions in the notebook until the introduction of Keras:

Observations from NB: The benefit of hidden layers



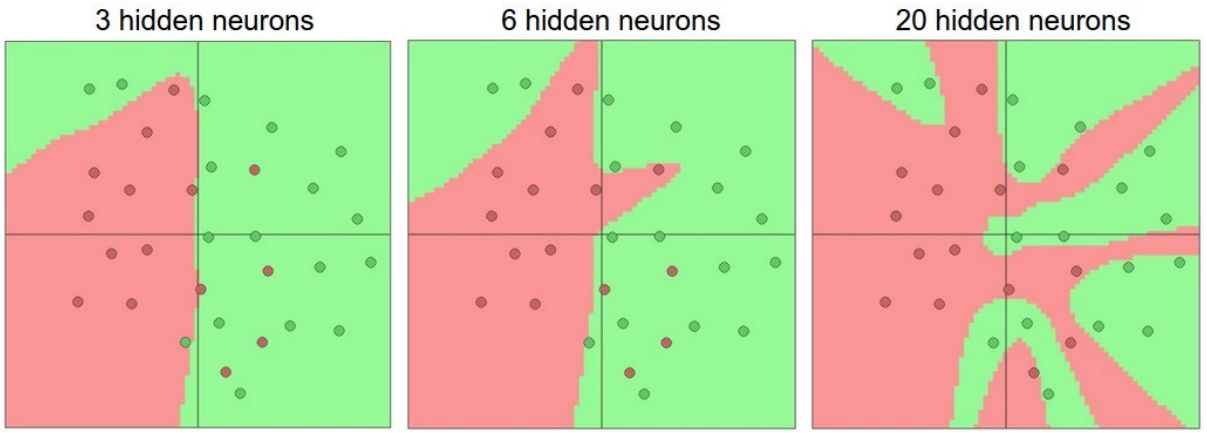


One hidden Layer is "in theory" enough



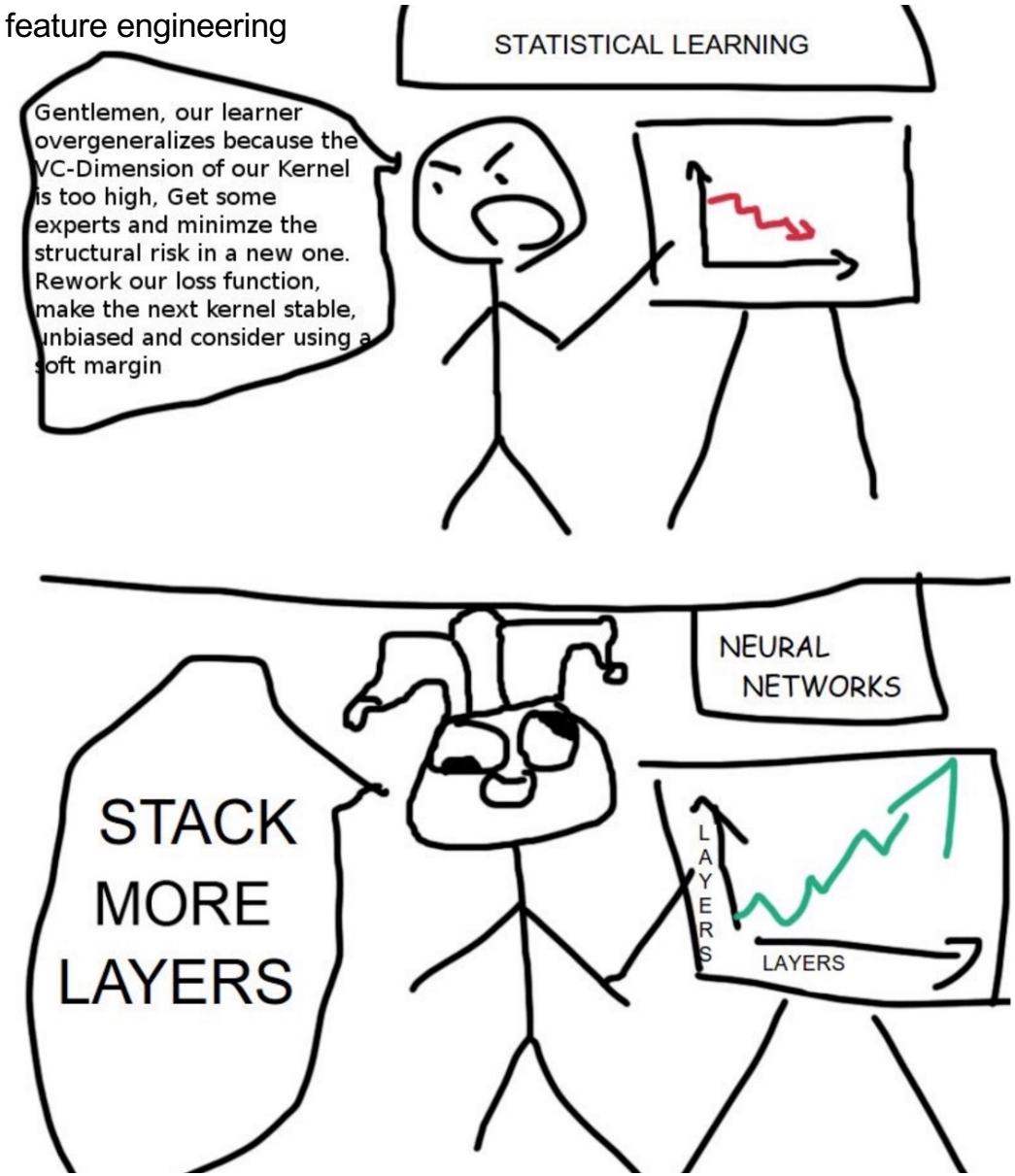
A network with one hidden layer is a universal function approximator!

→ Each decision curve can be fitted with a NN with large enough hidden layer



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

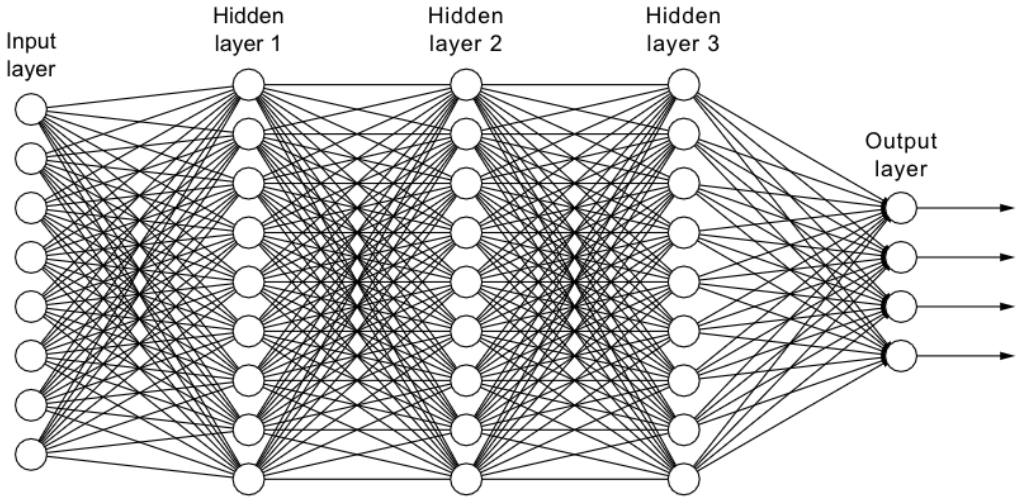
Game time – Recall learning DL vs Statistical Learning



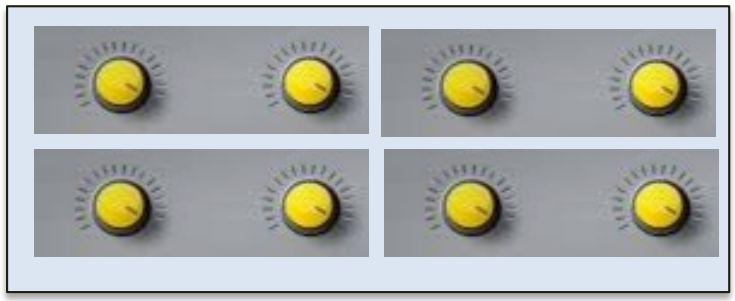
https://www.reddit.com/r/ProgrammerHumor/comments/8c1i45/stack_more_layers/

Training NN:
Minimize the loss function

How to determine the weights



Given the input, the output of a NN is defined by the weights

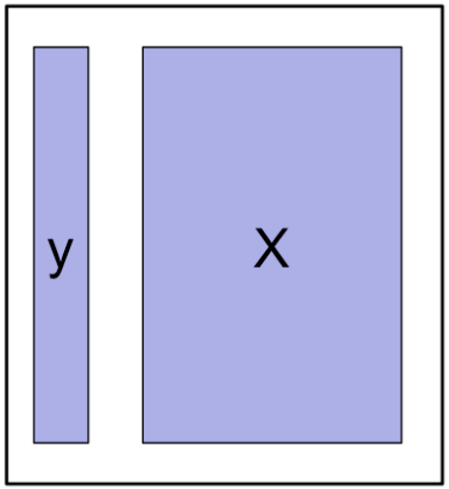


Typical >1 Mio. weights

During training the weights (including biases) are tuned so that the NN bests fulfils its specific tasks by minimizing a loss function.

Tasks in DL

- The loss function depends on the task
- 2 Main tasks in DL predict y given x

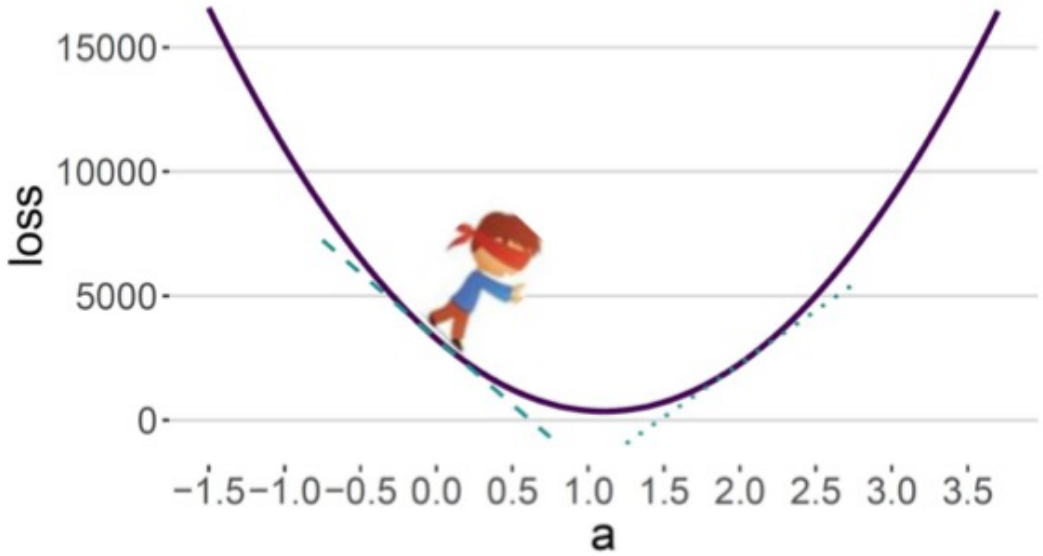


Supervised Learning

  - Regression
    - Point prediction: Predict a number
    - Probabilistic prediction: predict a continuous distributions
    - **Loss Function Mean Squared Error (MSE)**
  - **Classification**
    - Point prediction: Predict a class label
    - Probabilistic prediction: predict a discrete probability distribution over the class labels
    - **First we focus on probabilistic binary classification where $Y \in \{0, 1\}$**
    - **Binary Cross Entropy (MSE not suited)**
      - **Log of Probabilities**

Idea of gradient descent

- Shown loss function for a single parameter a

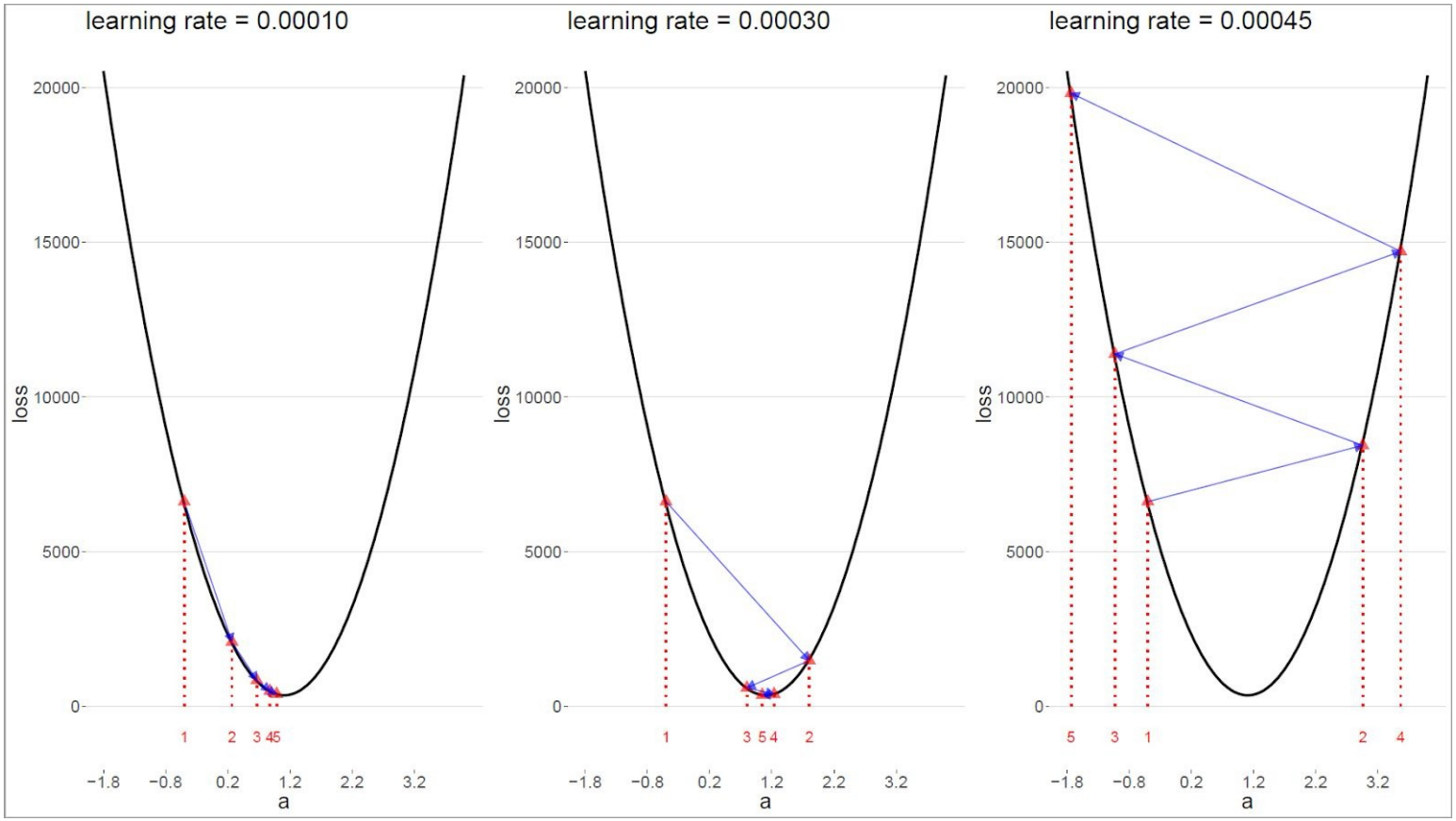


- Take a large step if slope is steep (you are away from minimum)
- Slope of loss function is given by gradient of the loss w.r.t. a
- Iterative update of the parameter a

$$a^{(t)} = a^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial L(a)}{\partial a} \right|_{a=a^{(t-1)}}$$

learning rate

The learning rate is a very important parameter for DL



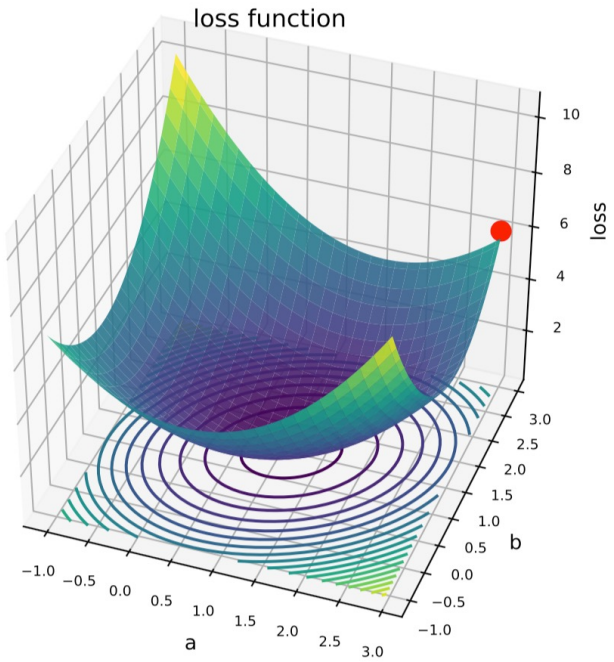
good – little slow

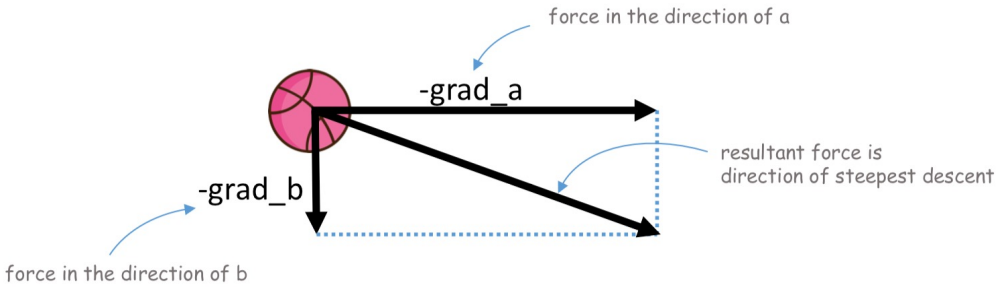
good

learning rate too high

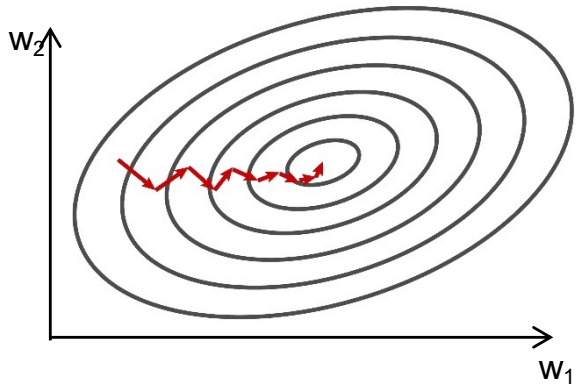
If the loss diverges to infinity: Don't panic, lower the learning rate!

In two dimensions





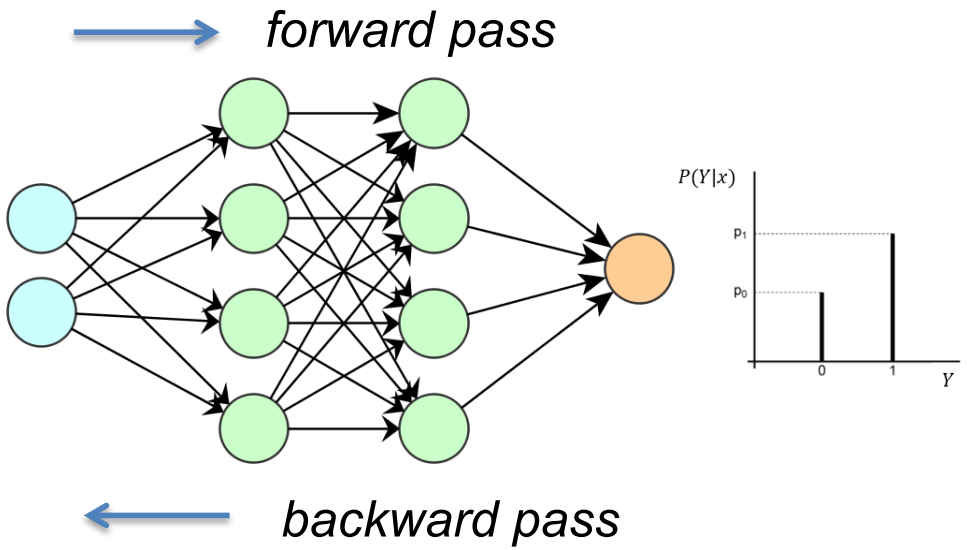
Gradient is perpendicular to contour lines



$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial L(\mathbf{w})}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$

Backpropagation

- We efficiently train the weights in a NN via forward and backward pass
  - Forward Pass propagate training example through network
    - Predicts as output $P(Y|x_i)$ for each input x_i in the batch given the NN weights w
      → With $P(Y|x_i)$ and the observed y_i we compute the loss $L = \left(\text{NLL} = -\frac{1}{N}\sum log(L_i)\right)$
  - Backward pass propagate gradients through network
    - Via chain rule all gradients $\frac{\partial L(\mathbf{w})}{\partial w_k}$ are determined
      → update the weights $w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \frac{\partial L(\mathbf{w})}{\partial w_i}\Big|_{w_i = w_i^{(t-1)}}$



For more see lecture 4 (or chapter 3 in our DL book)

The miracle of gradient descent in DL

Loss surface in DL (is not convex) but SGD magically also works for non-convex problems.

Source: chapter 3 in our DL Book

Typical Training Curve / ReLU

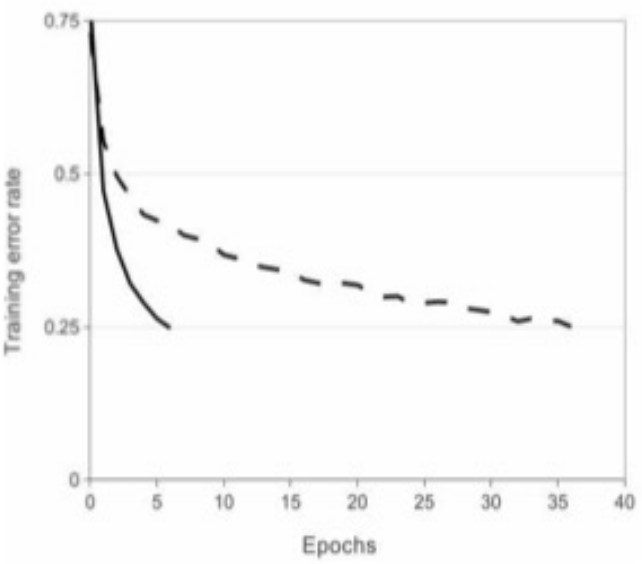
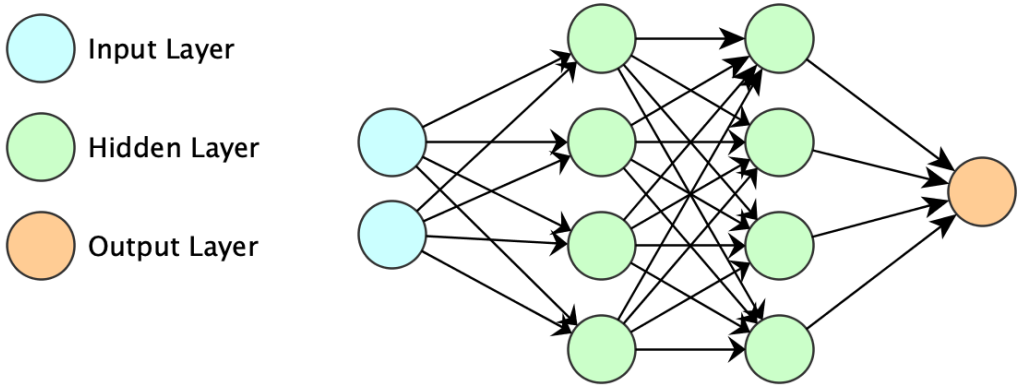
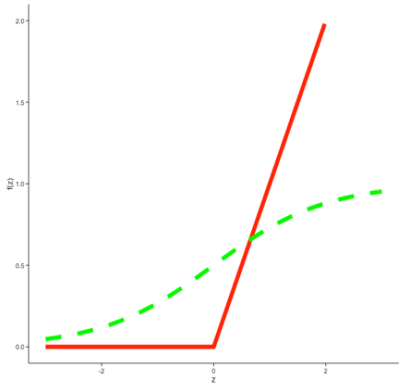


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

Source:
Alexnet
Krizhevsky et al 2012





Motivation:
Green:
sigmoid.
Red:
ReLU faster
convergence

Epochs: ”each training examples is used once”

Deep Learning Libraries

Deep-learners talk about tensors - What is a tensor?

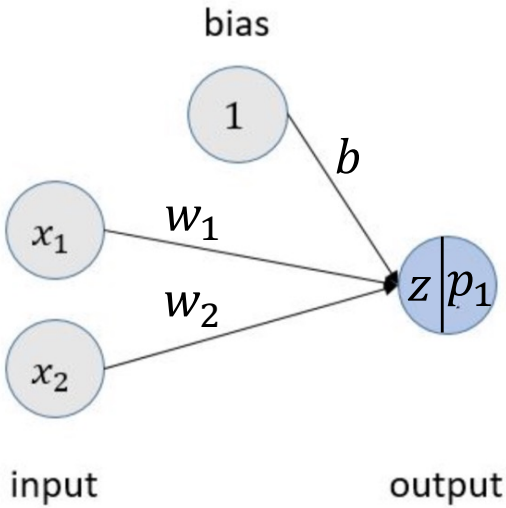
For Deep Learning: A tensor is an array with several indices (like in numpy). Order (a.k.a rank) is the number of indices and shape is the range.

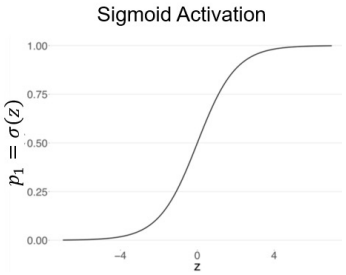
```
In [1]: import numpy as np

In [2]: T1 = np.asarray([1,2,3]) #Tensor of order 1 aka Vector
        T1

Out[2]: array([1, 2, 3])

In [3]: T2 = np.asarray([[1,2,3],[4,5,6]]) #Tensor of order 2 aka Matrix
        T2

Out[3]: array([[1, 2, 3],
               [4, 5, 6]])

In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)

In [6]: print(T1.shape)
        print(T2.shape)
        print(T3.shape)

        (3,)
        (2, 3)
        (10, 2, 3)
```

Tensors (as a generalization of matrix and vectors) are the basic quantities in DL.

GPUs love Vectors

$F^{\mu\nu}$

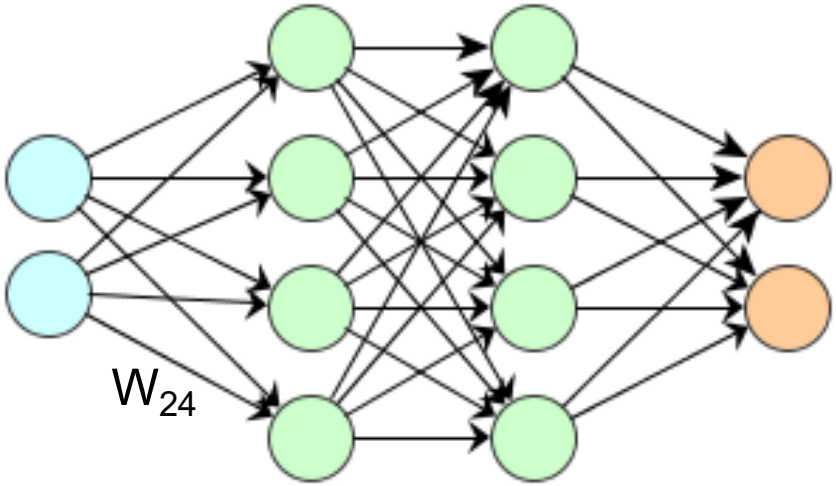




$$p_1 = \text{sigmoid}\left(\begin{pmatrix} x_1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + b \right)$$

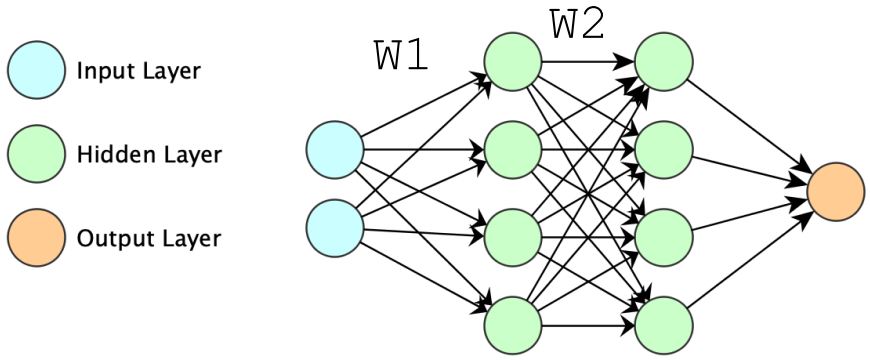
DL: better to have column vectors

Typical Tensors in Deep Learning



- The input can be understood as a vector
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch, x_j)
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

Structure of the network



In code:

```
## Solution 2 hidden layers
def predict_hidden_2(X):
    hidden_1 = F.sigmoid(X @ W1 + b1) #@ is Matrix Mult.
    hidden_2 = F.sigmoid(hidden_1 @ W2 + b2)
    return F.sigmoid(hidden_2 @ W3 + b3)
```

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

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

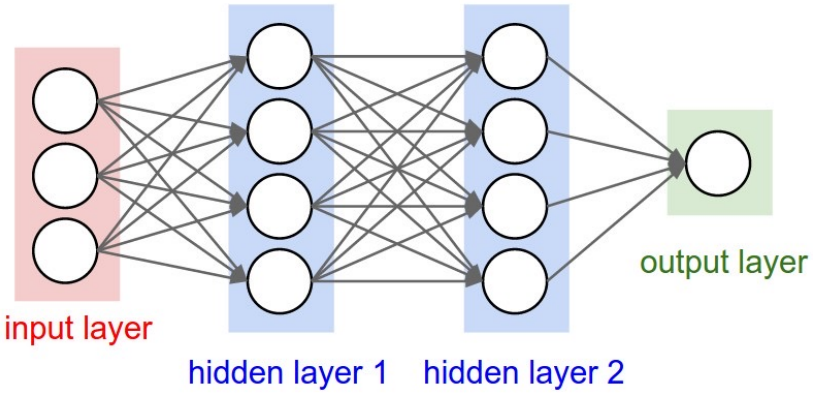
Defining the NN with code is done with so-called low-level libraries (here pytorch)
Looks a bit like onions, matryoshka (Russian Dolls) or lego bricks…

Question: Dimensions of W1, W2, and W2

Keras as High-Level library

Keras

- We use Keras as high-level library
- Libraries make use of the Lego like block structure of networks



High Level Libraries

Keras

Under the “hood” of Keras runs the engine;

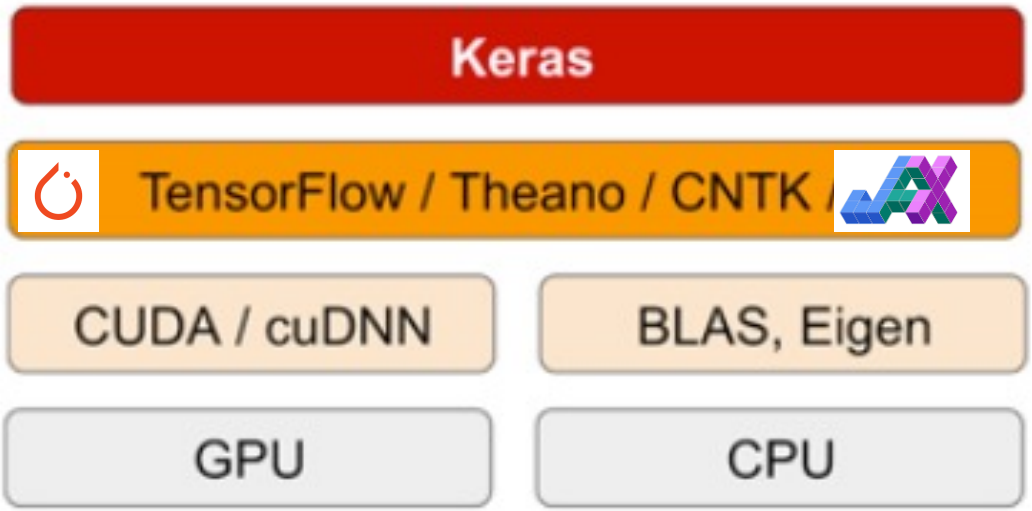


Figure: From Deep Learning with Python, Francois Chollett

Keras Workflow

Keras

Load packages
- Select “Engine” (before `import keras`)

PyTorch JAX TensorFlow

```
import os
os.environ["KERAS_BACKEND"] = "torch"    # or 'jax' or 'tensorflow'
import keras
```

Load building blocks
- Layers, functions etc.

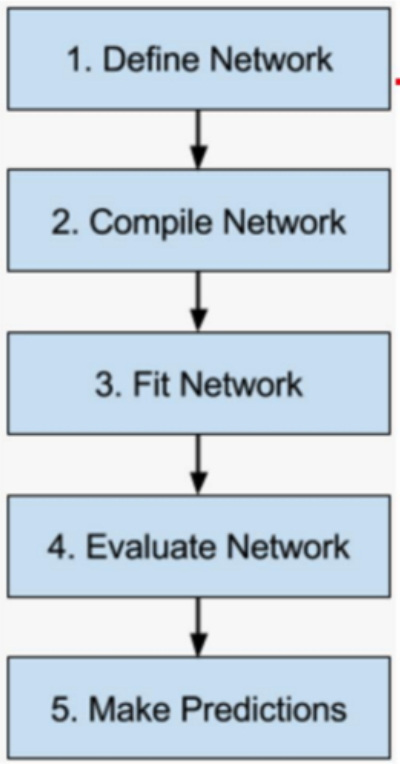
```
from keras.models import Sequential
from keras.layers import Dense,
from keras.optimizers import SGD
```

Define the network

Sequential API, layers output are the input for the next layer, Alternative Functional API

Keras

Hidden layer with 8 Neurons, Activation function: sigmoid




```
# Define fcNN
model = Sequential()
model.add(Dense(8,
                input_shape=(2,),
                activation='sigmoid'))
model.add(Dense(1,
                activation='sigmoid'))
```

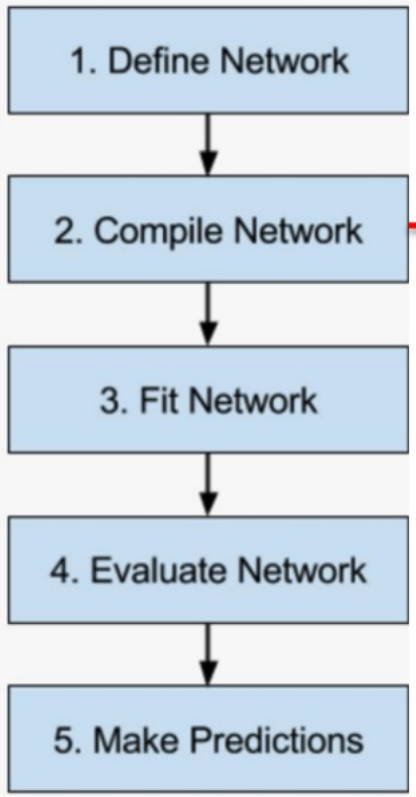
Last layer with one output neuron for binary classification

Input shape needs to be defined only at the beginning.

Alternative: `input_dim=2, Functional API or Sequential API`

Compile the network

Keras




```
model.compile(optimizer=SGD(learning_rate=0.01)
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

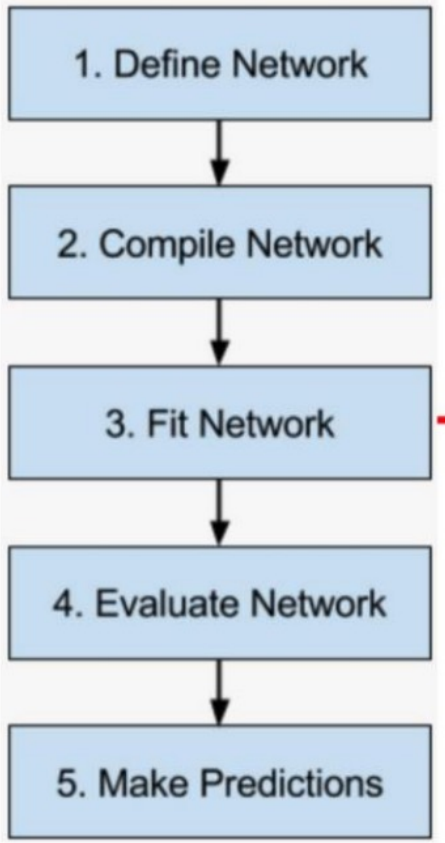
Which optimizer should be used?
Here Stochastic Gradient Descent

Loss Function to optimize
Here: Binary CE

Which metrics do we want to track,
Here: Accuracy

Fit the network

Keras



Data Tensor X for training

Label Tensor Y

Validation at
End of each epoch

```
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=1000,
                    batch_size=10,
                    verbose=1)
```

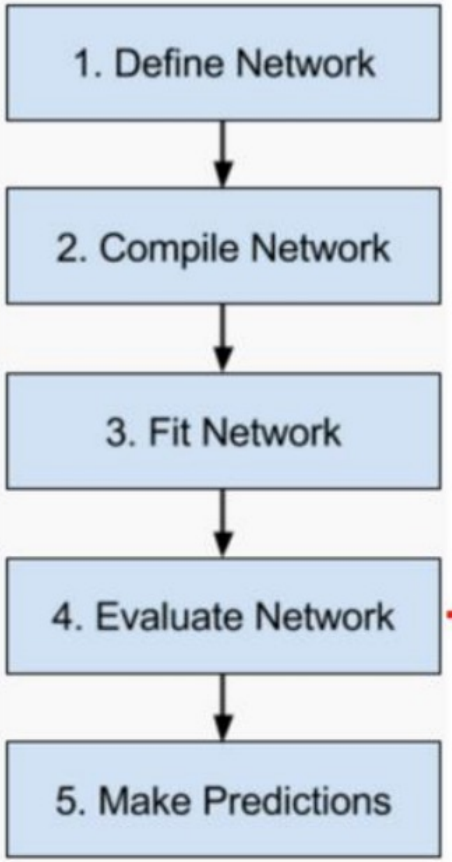
How many time do we feed the whole dataset to the model

How many samples per batch, 1 batch = 1 iteration of weight updates

Should we see the whole output? If no verbose = 0

Evaluate the network

Keras



Unseen (to the model) Test Data X with labels Y

```
model.evaluate(X_test, y_test, verbose=0)
```

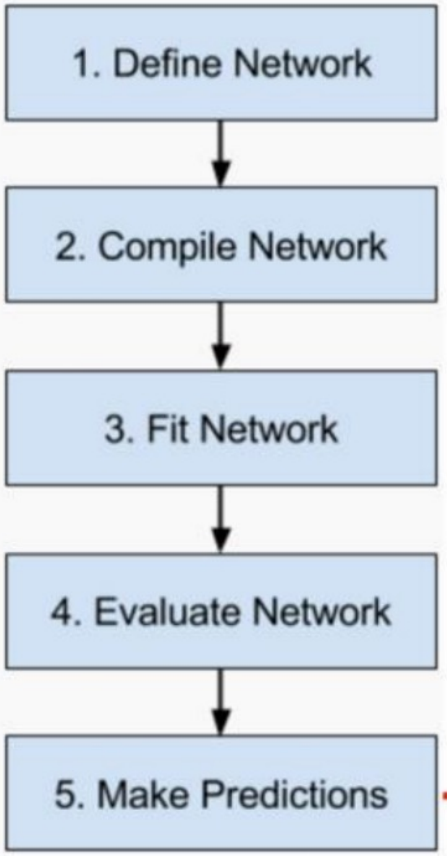
```
Test Loss: 0.33591118454933167, Test Accuracy: 0.8581818342208862
```

Remember from the .compile

```
loss='binary_crossentropy',
metrics=['accuracy'])
```

Make Predictions

Keras



predict with the model,
Weights are fixed now

First 10 rows
(observations) from
Testdata

All features, here: 2

```
model.predict(X_test[0:10,:])
```

```
1/1 [==============================] - 0s 148ms/step
array([[5.9251189e-01],
       [8.8978779e-01],
       [7.8563684e-01],
       [9.1934395e-01],
       [6.1078304e-01],
       [8.2838058e-01],
       [2.5862646e-01],
       [3.1314052e-05],
       [2.6938611e-01],
       [3.7005499e-02]], dtype=float32)
```

Each of the observations
Gets a probability to
Belong to class 1

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

Slides from Chollet (founder of Keras)

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)
```

Slides from Chollet (founder of Keras)

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)
```

Slides from Chollet (founder of Keras)

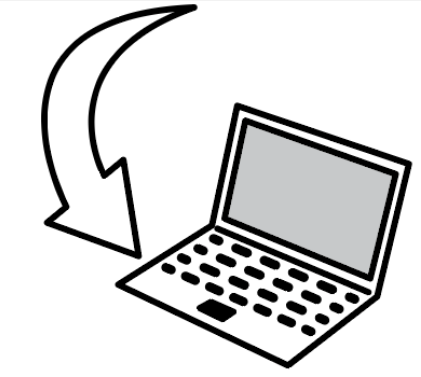
Functional (do you spot the error?)

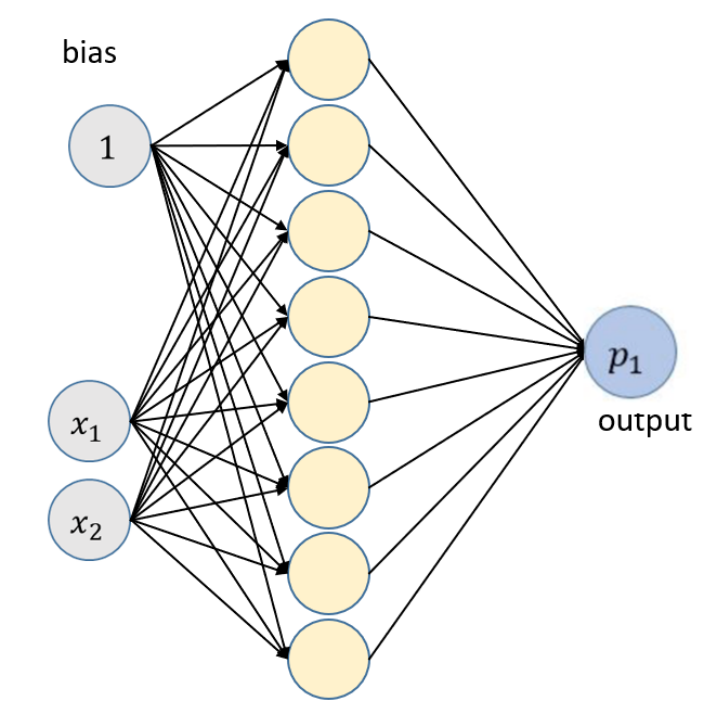
The functional API

```
import keras
from keras import layers

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

Slides from Chollet (founder of Keras)

Exercise:





input hidden

Open NB 01_simple_forward_pass_keras_torch.ipynb and do the Keras part

Summary

- A neuron in a NN is loosely inspired by a neuron in the brain.
- The value of a neuron is computed by the weighted sum of the values in connected neurons of the previous layer, which is then passed through an activation function such as sigmoid or relu
- For a binary classification task we can use the sigmoid activation function for a single neuron in the last layer.
  As loss we use in Keras 'binary_crossentropy' which is the NLL
- To achieve a non-linear (non-planar) decision boundary between the classes in binary classification, we need NNs with hidden layers.
- The weights of a NN are learned during the training via backprobagation minimizing the loss using SGD (Stochastic Gradient Descent)
- For efficient training use the relu activation function for hidden layers.
- If the loss diverges to infinity: Don't panic, lower the learning rate!