

Machine Intelligence:: Deep Learning

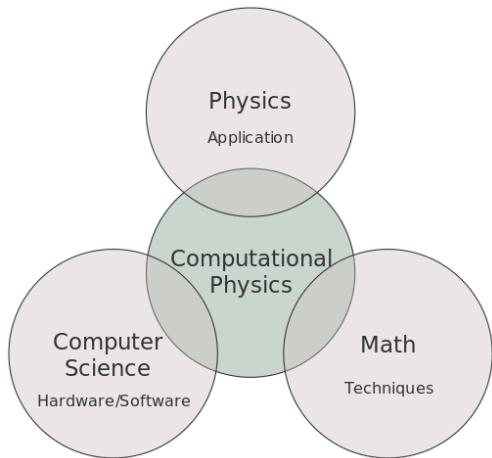
Week 1

Oliver Dürr, Beate Sick, Lilach Goren, Pascal Bühler

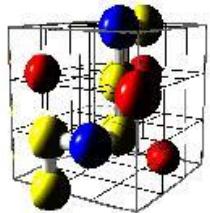
Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Oliver's Background

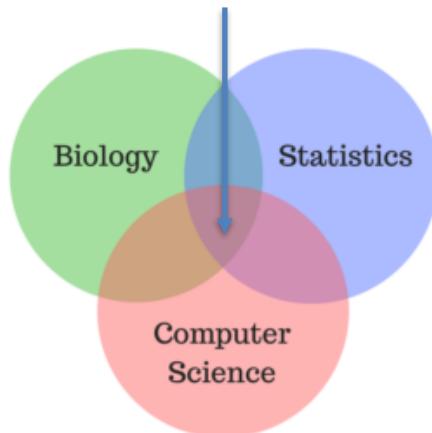
Computational Physics



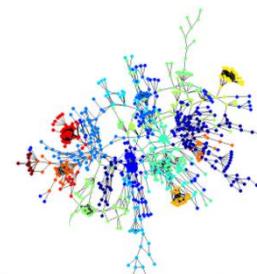
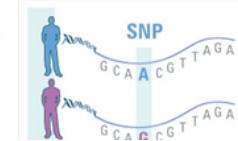
1990's
Uni-Konstanz



Bioinformatics

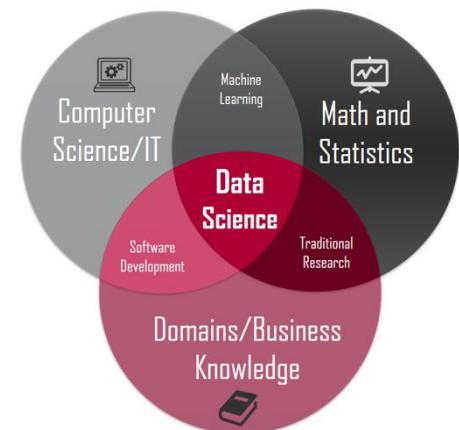


2000's
Genedata Basel

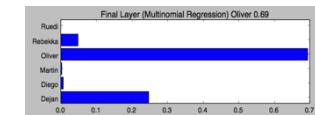
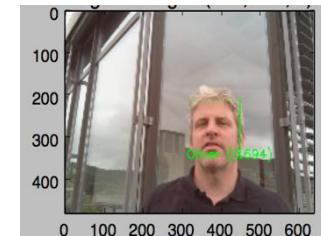


eclipse

Data Science



2010's
ZHAW Winterthur HTWG Konstanz



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_2024/prerequisites.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)

Dates & Topics

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

Day	Date	Time	Topic
1	20.02.2024	13:30-17:00	Introduction to Deep Learning & Keras, first NNs
2	27.02.2024	13:30-17:00	Loss, Optimization, Regression, Classification
3	05.03.2024	13:30-17:00	Computer vision, CNN-architecture
4	12.03.2024	13:30-17:00	DL in practice, pretrained (foundation) models
5	19.03.2024	13:30-17:00	Model evaluation, baselines, xAI, troubleshooting
6	26.03.2024	13:30-17:00	Generative Models, Transformer-architecture
7	02.04.2024	13:30-17:00	Vision Transformer
8	09.04.2024	13:30-17:00	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

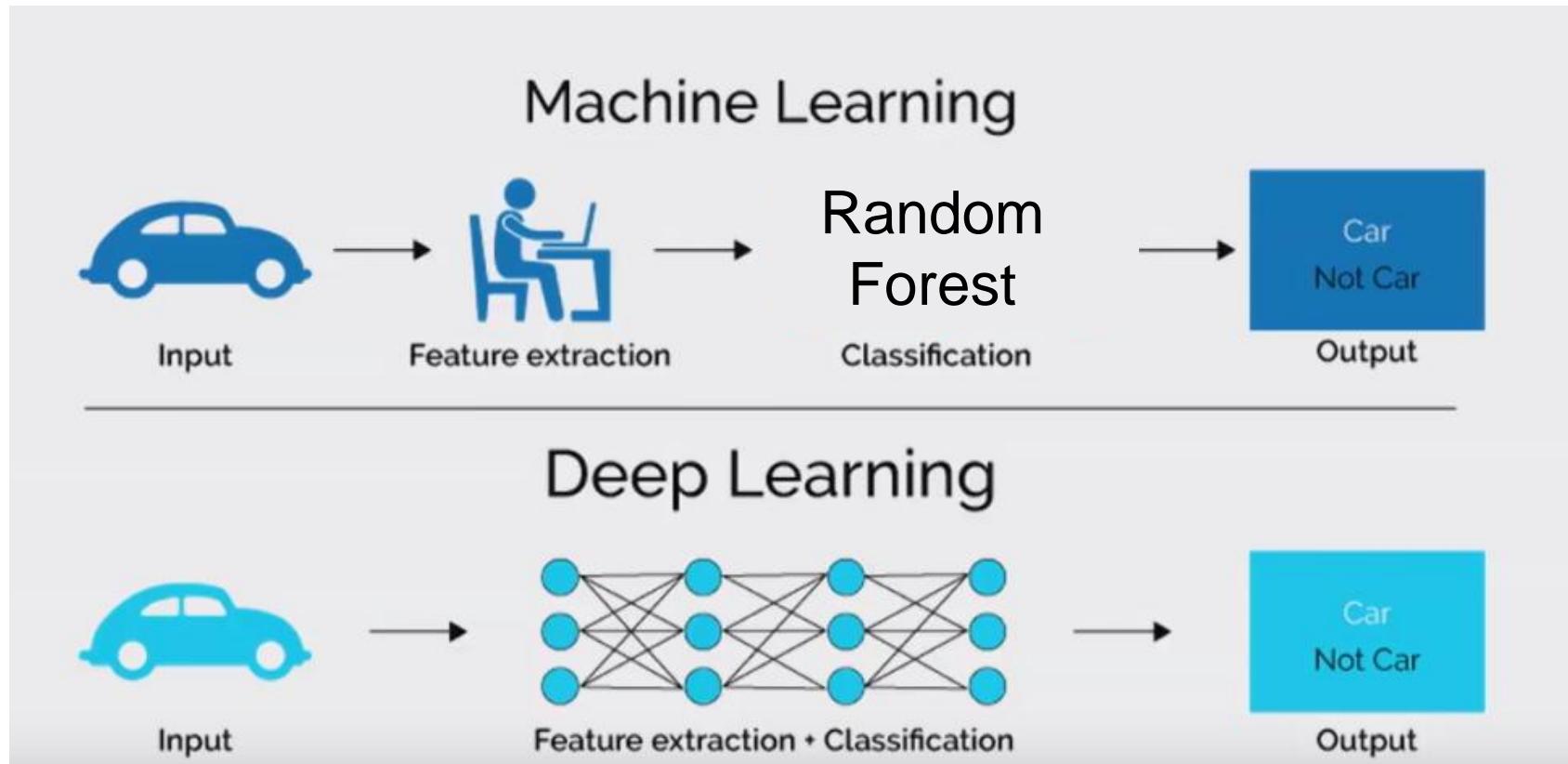
- Computers have been quite bad in things which are easy for humans (images, text, sound)
- A Kaggle contest 2012
- In the following we explain why

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?

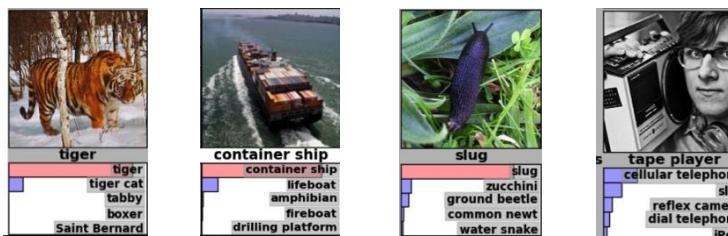
Deep Learning vs. Machine Learning



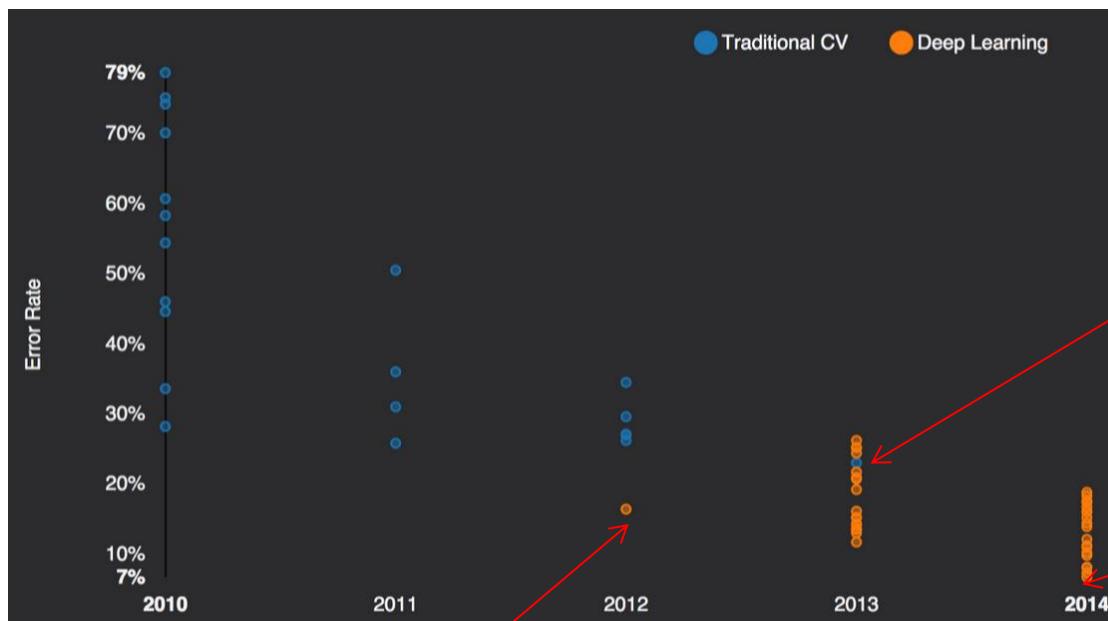
The most convincing case for
DL (subjective view)

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

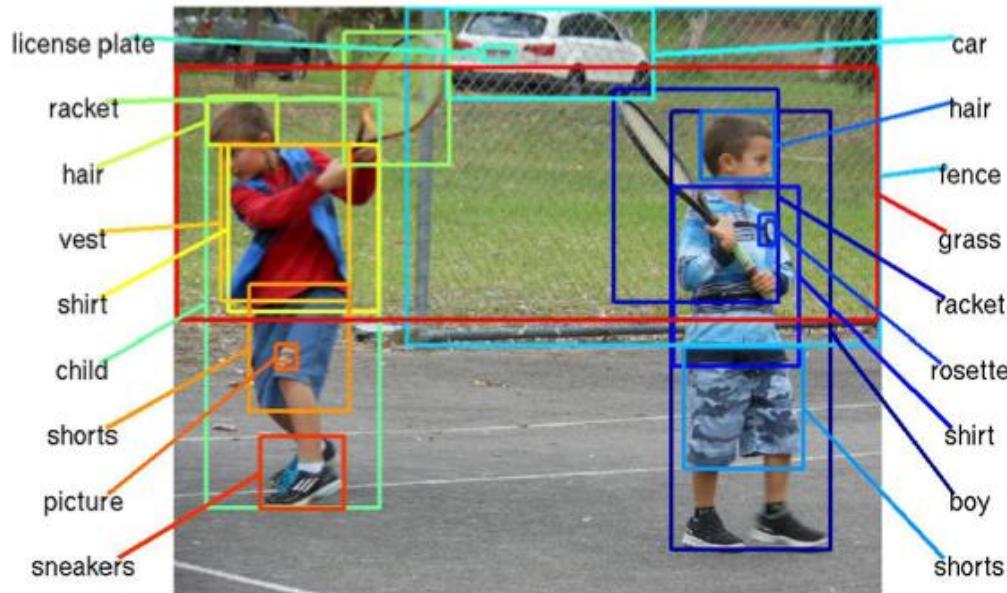
Und es hat zoom gemacht

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 to many submissions](#))
- 3.57% Microsoft (Resnet winner 2015) → task solved!

The computer vision success story

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



Images from cs229n

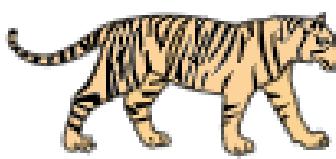
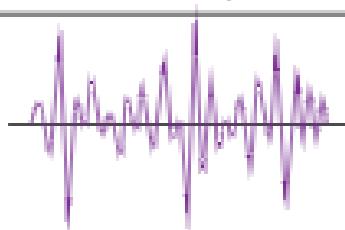


Deep Blue beat Kasparov at chess in 1997.
Watson beat the brightest trivia minds at Jeopardy in 2011.
Can you tell Fido from Mittens in 2013?



"man in black shirt is playing guitar."

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



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.

First Neural Network

The Single Neuron: Biological Motivation

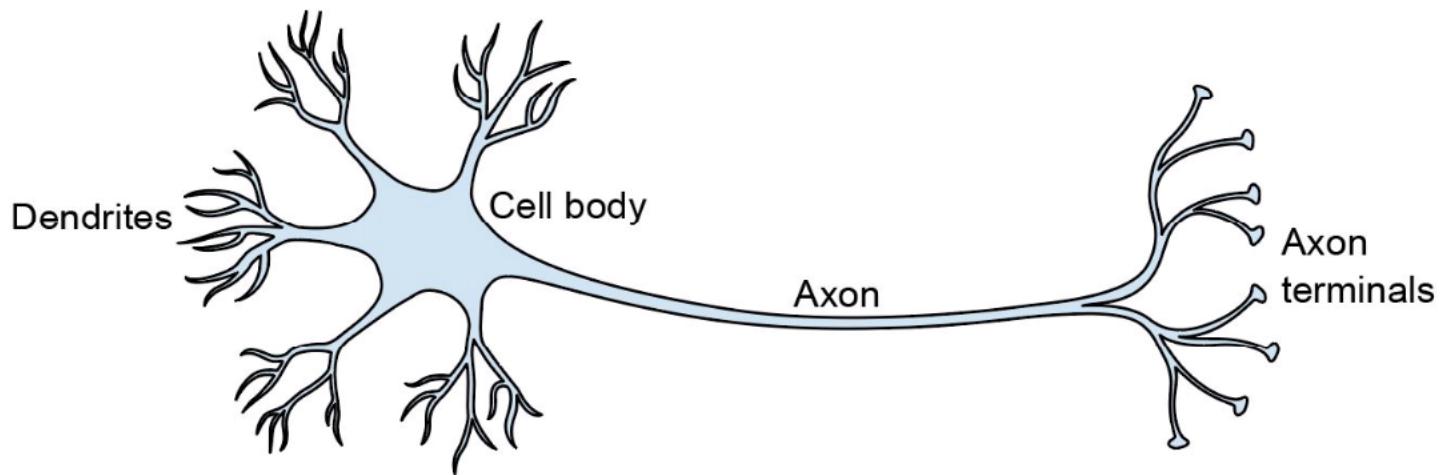
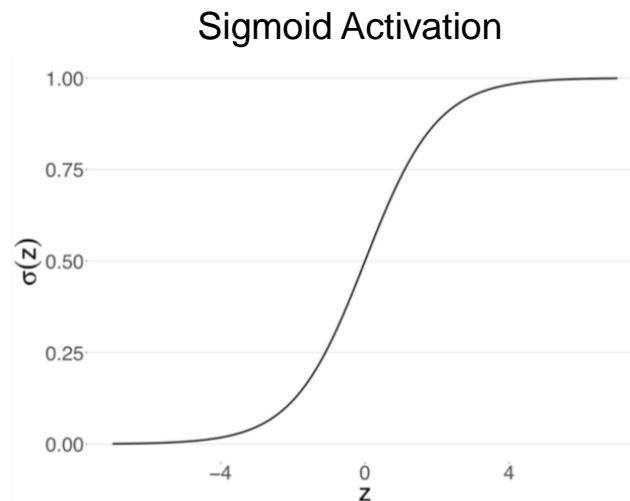
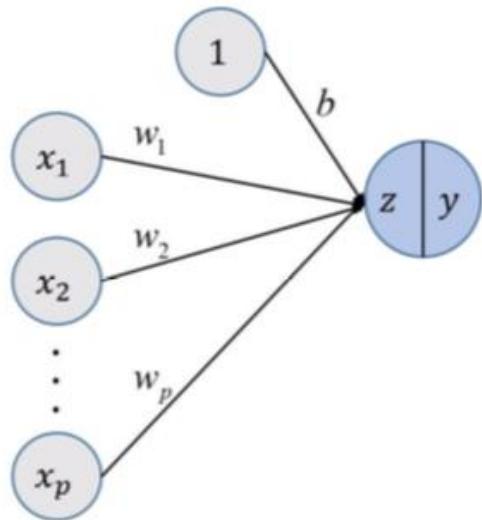


Figure 2.2 A single biological brain cell. The neuron receives the signal from other neurons via its dendrites shown on the left. If the cumulated signal exceeds a certain value, an impulse is sent via the axon to the axon terminals, which, in turn, couples to other neurons.

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

The Single Neuron: Mathematical Abstraction

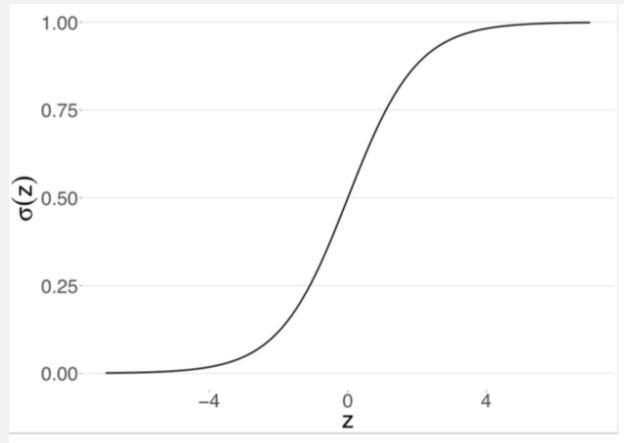
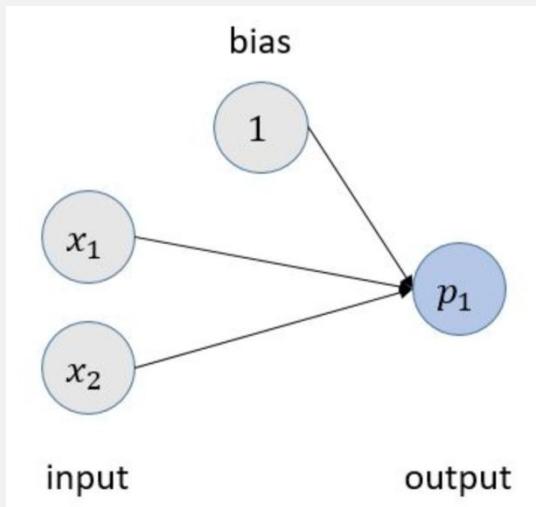


$$z = b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_p \cdot w_p$$

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

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:

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

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

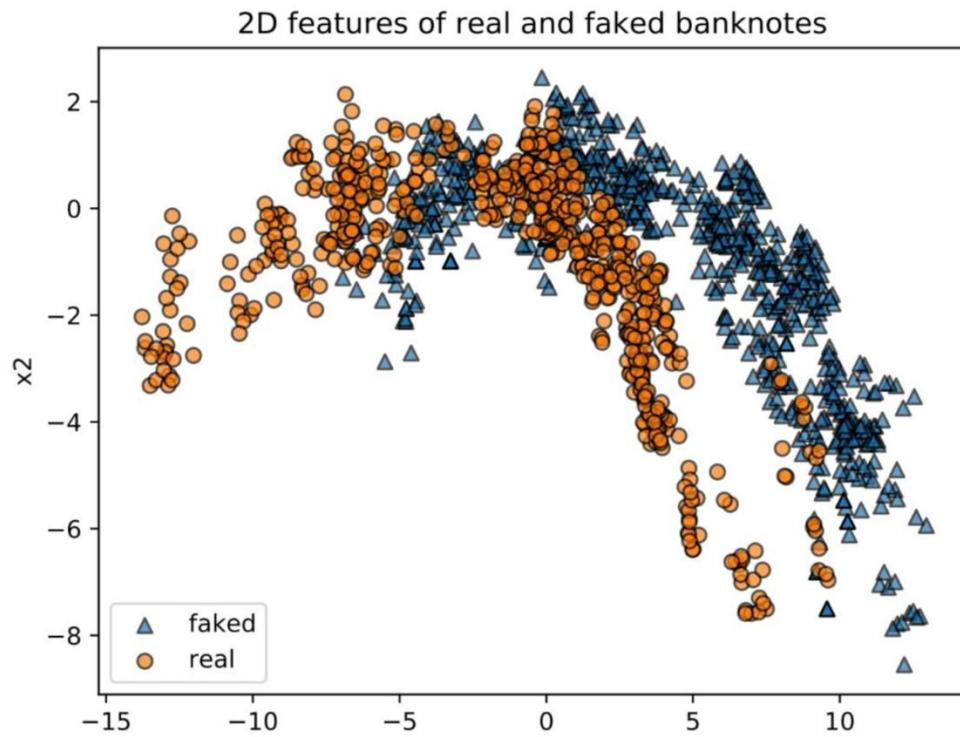
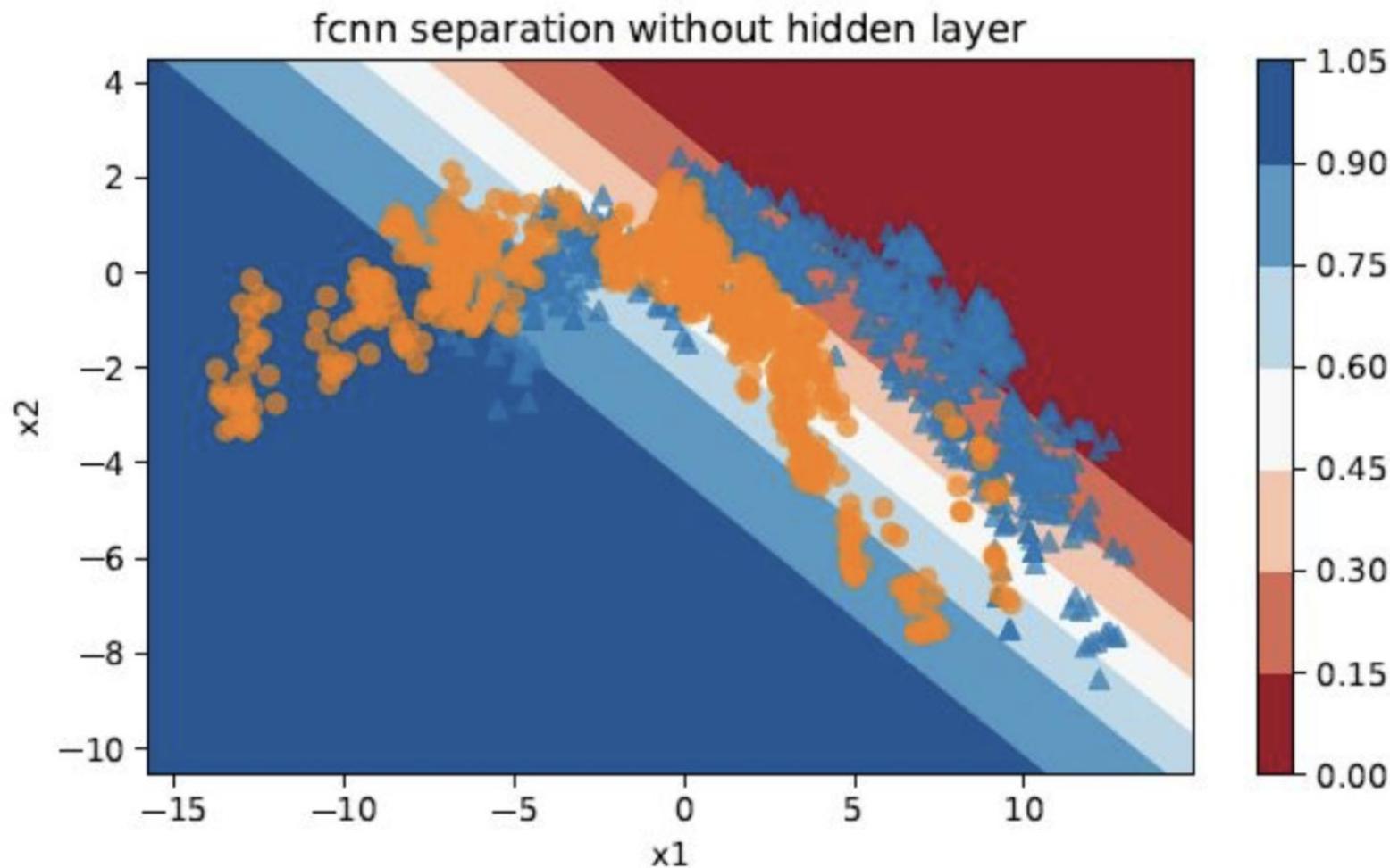


Figure 2.5 The (training) data points for the real and faked banknotes

https://github.com/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_01.ipynb

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

- 2 Main tasks in DL predict y given x

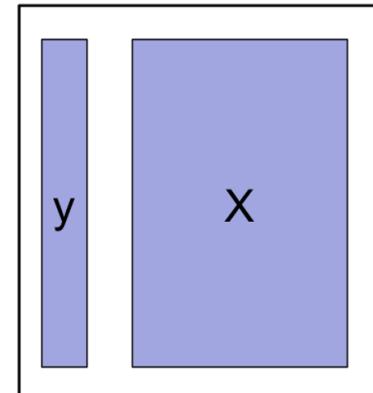
- **Classification**

- Point prediction: Predict a class label
 - Probabilistic prediction:
predict a discrete probability distribution over all possible class labels

- **Regression**

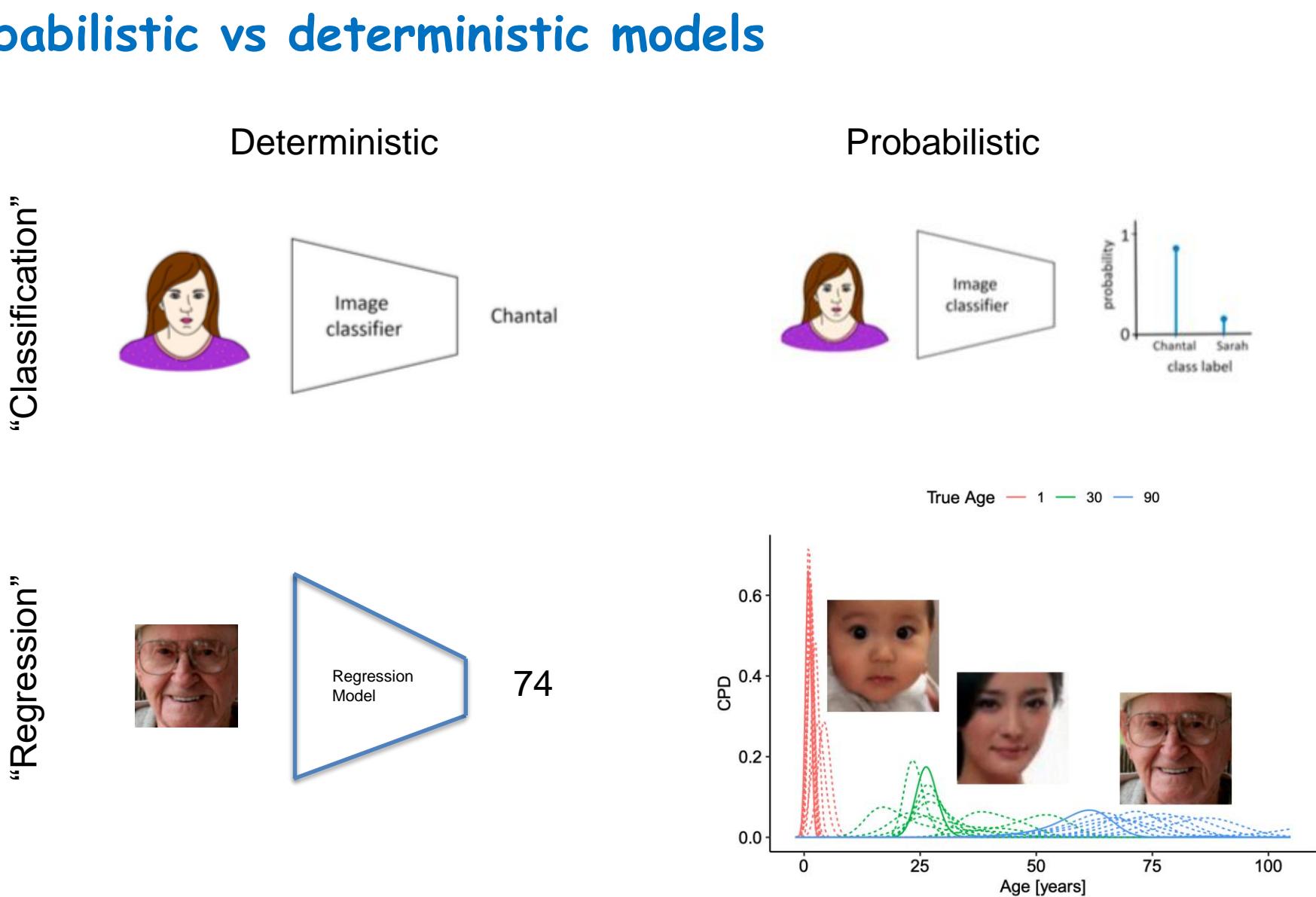
- Point prediction: Predict a number
 - Probabilistic prediction:
predict a continuous probability distributions over the possible Y value range

- The loss function depends on the task



Supervised Learning

Probabilistic vs deterministic models



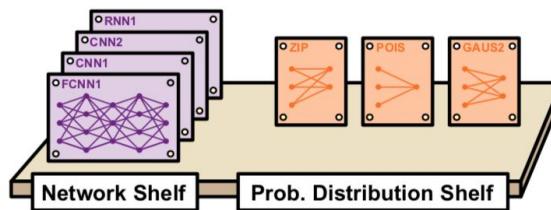
Conditional probability distribution (CPD)
 $p(y|x)$

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)

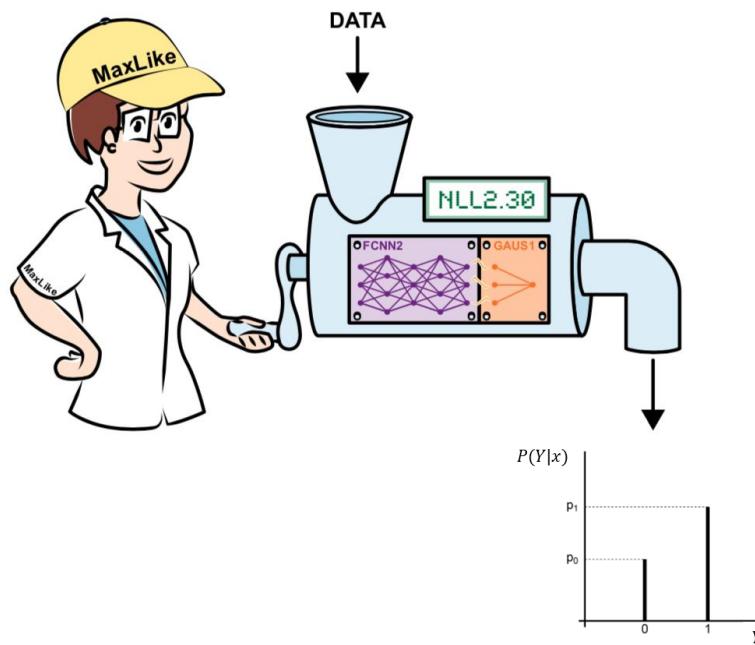
Special networks for x

- Vector FCNN
- Image CNN
- Text CNN/RNN



NN heads tailored for Y

- Probabilistic classification
- Probabilistic regression



Recall linear regression from statistics

$$(Y|X = x_i) \sim N(\mu(x_i), \sigma^2)$$

$$Y \in \mathbb{R}, \quad \mu_x \in \mathbb{R}$$

point prediction
probabilistic prediction

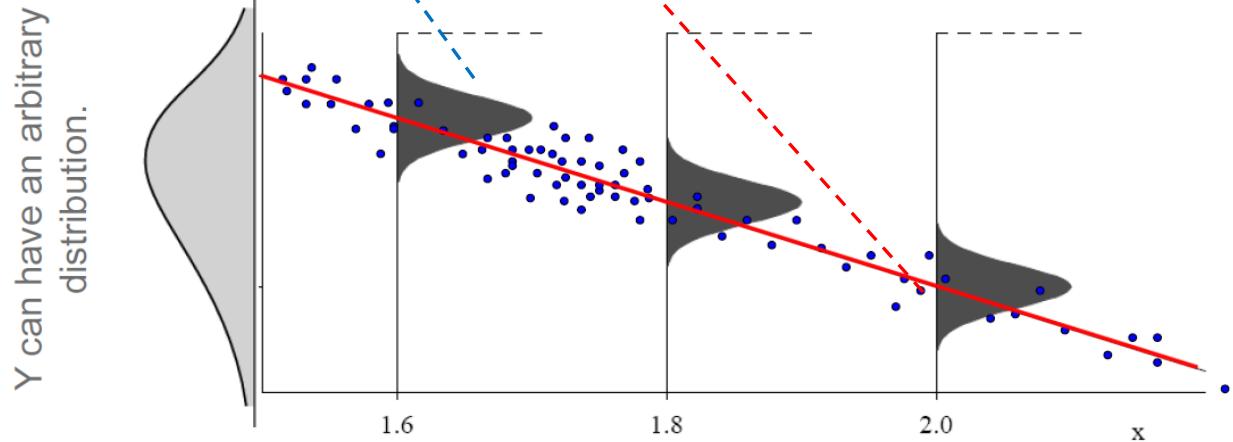
$$y_i = \beta_0 + \beta_1 \cdot x_{i1} + \varepsilon_i$$

$$E(Y_{x_i}) = \mu_{x_i} = \hat{y}_{x_i} = \beta_0 + \beta_1 \cdot x_{i1}$$

$$\text{Var}(Y_{x_i}) = \text{Var}(\varepsilon_i) = \sigma^2$$

$$\varepsilon_i \sim N(0, \sigma^2)$$

Probabilistic linear regression predicts for each input x_i a Gaussian conditional probability distribution for the output $P(Y|x_i)$ that assigns each possible value of Y a likelihood.



Recall logistic regression from statistics

$$(Y|X_i) \sim \text{Ber}(p_{x_i})$$

$$Y \in \{0,1\}$$

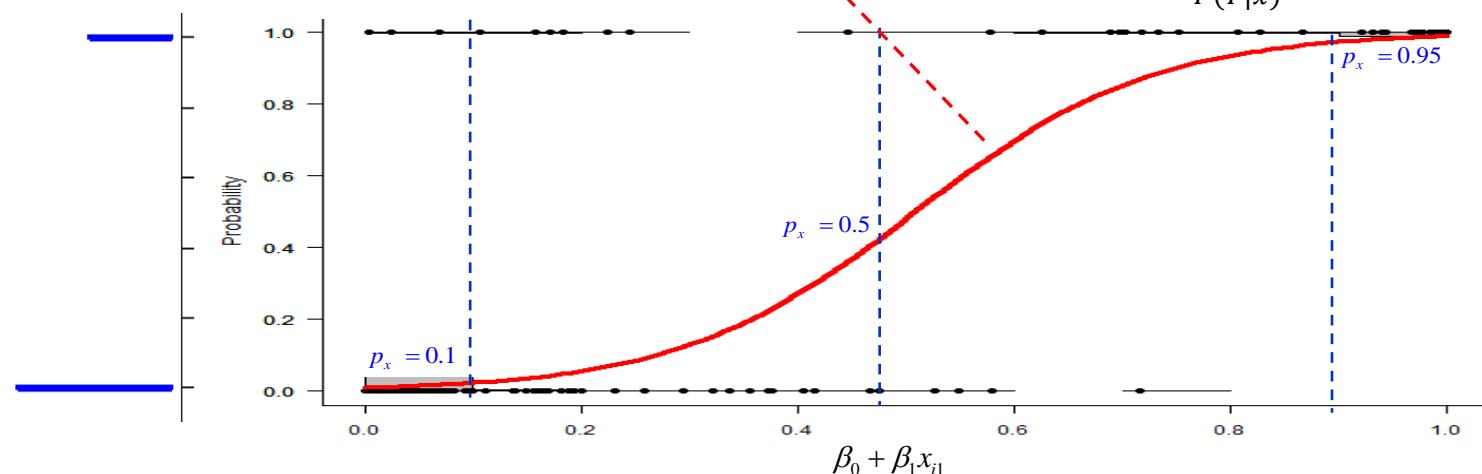
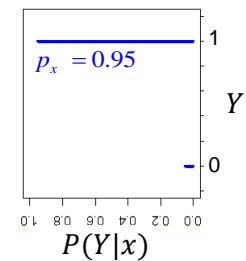
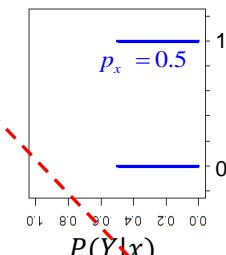
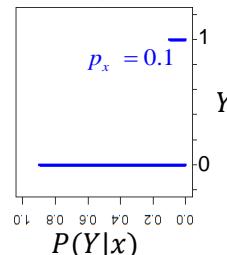
$$p_x = P(Y = 1|x) \in \mathbb{R}$$

$$\log\left(\frac{p_{x_i}}{1-p_{x_i}}\right) = \beta_0 + \beta_1 x_{i1}$$

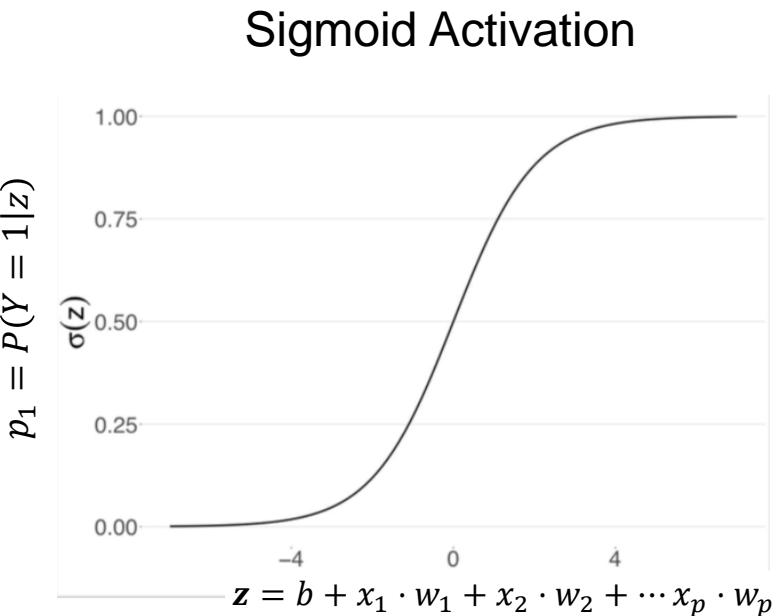
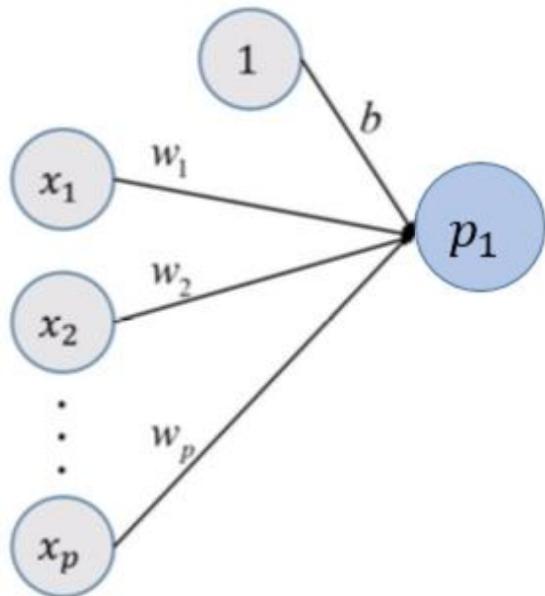
$$P(Y = 1|x_i) = p_{x_i} = \sigma(\beta_0 + \beta_1 x_{i1}) = \frac{1}{1+e^{-(\beta_0 + \beta_1 x_{i1})}}$$

Logistic regression predicts a conditional Bernoulli $P(Y|x_i)$

$$P(Y | X = x) = \begin{cases} p_x & , y=1 \\ 1-p_x & , y=0 \end{cases}$$



Logistic regression in DL view



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

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

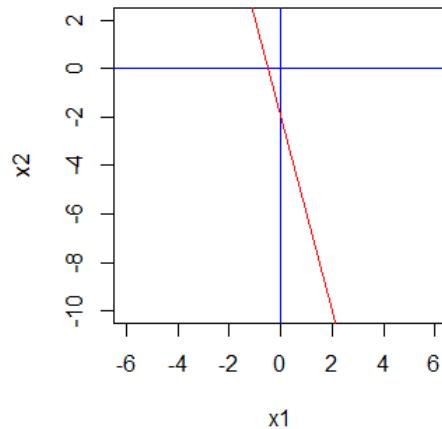
Logistic regression yield linear/planar decision curves

Logistic regression model:

$$\ln\left(\frac{p}{1-p}\right) = 1 + 2x_1 + 0.5x_2$$

Determine the **separation curve between $Y=1$ and $Y=0$** in the feature room which is spanned by x_1 and x_2 and draw it in the following plot x_2 and x_1 .

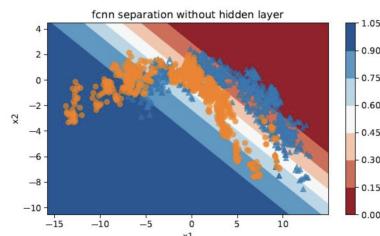
Hint: on the separation curve should hold: $P(Y = 1|x) = 0.5$
-> plug in 0.5 for p and solve for x_2 .



$$\ln\left(\frac{0.5}{1-0.5}\right) = 0 = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$x_2 = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2} \cdot x_1$$

$$x_2 = -2 - 4 \cdot x_1$$



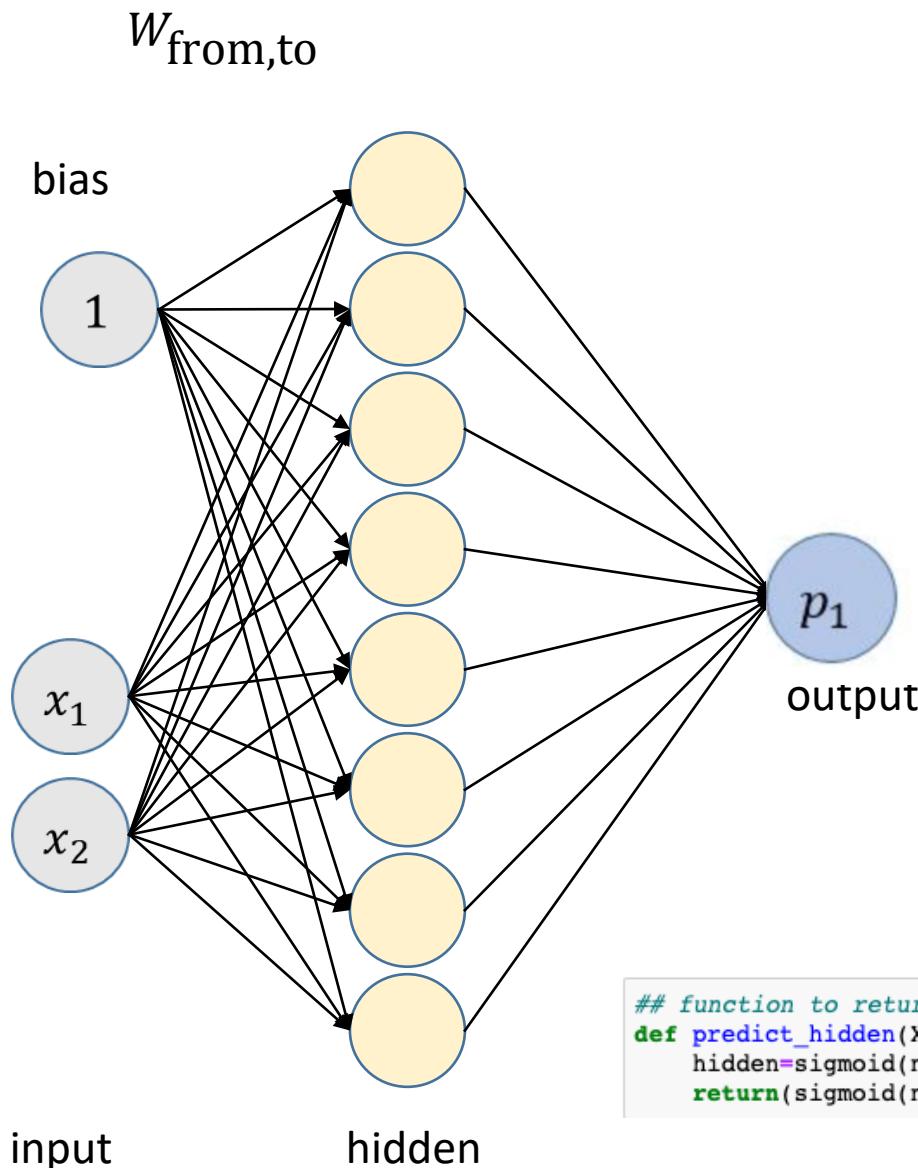
Hence a NN with 1 output neuron with sigmoid-activation w/o hidden layer have linear decision boundary

A close-up shot from the movie Inception. Two men in dark suits are facing each other. The man on the left has his eyes closed and a slight smile. The man on the right has his eyes open and is looking directly at the other man. The lighting is warm and dramatic.

WE NEED TO GO

DEEPER

A first deep network



In math ($f = \text{sigmoid}$)
 $p = f(f(X \cdot W_1 + b_1) \cdot W_2 + b_2)$

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

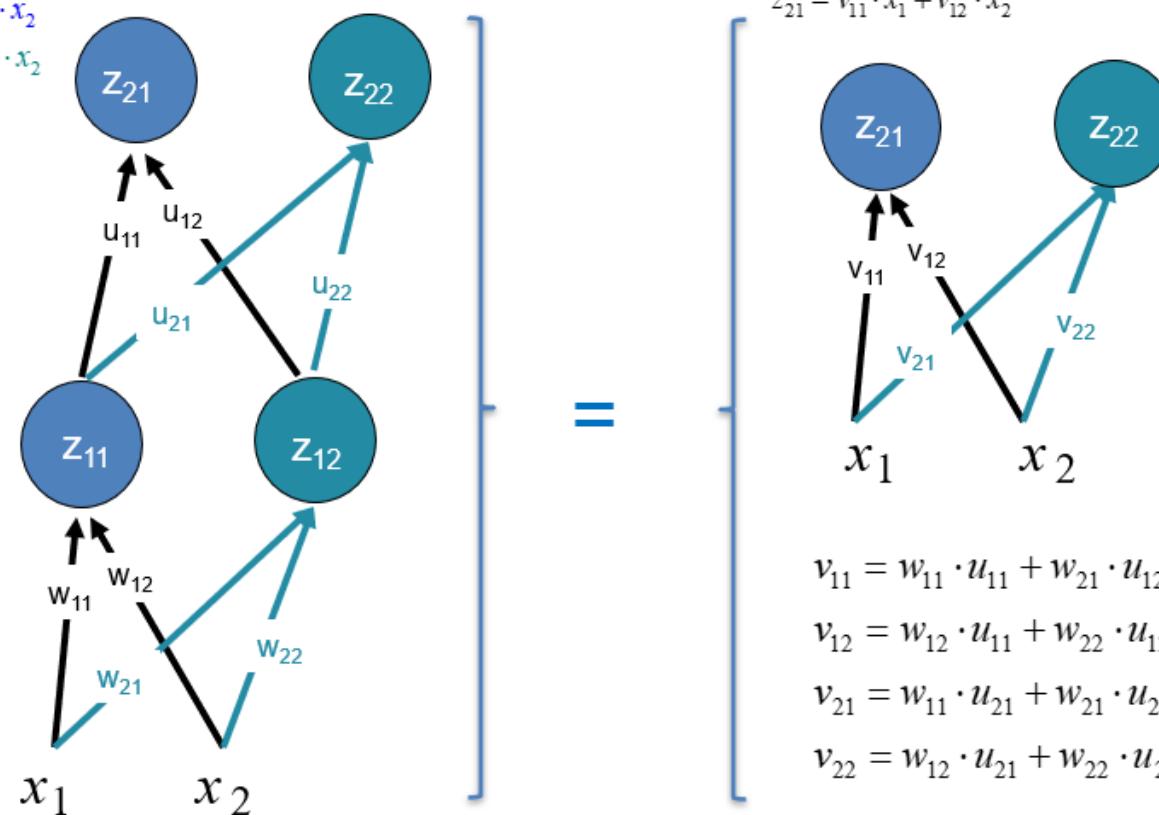
2 linear layers can be replaced by 1 linear layer -> can't go deep with linear layers!

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

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

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

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



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

Recap: Matrix Multiplication aka dot-product of matrices

We can only multiply matrices if their dimensions are compatible.

$$\mathbf{A} \times \mathbf{B} = \mathbf{C}$$

$$(m \times n) \times (n \times p) = (m \times p)$$

$$\begin{array}{c} \mathbf{A}_{3 \times 3} \quad \times \quad \mathbf{B}_{3 \times 2} \quad = \quad \mathbf{C}_{3 \times 2} \\ \left[\begin{array}{ccc} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{array} \right] \times \left[\begin{array}{cc} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{array} \right] = \left[\begin{array}{cc} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{array} \right] \end{array}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}$$

$$c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}$$

$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}$$

$$c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}$$

$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}$$

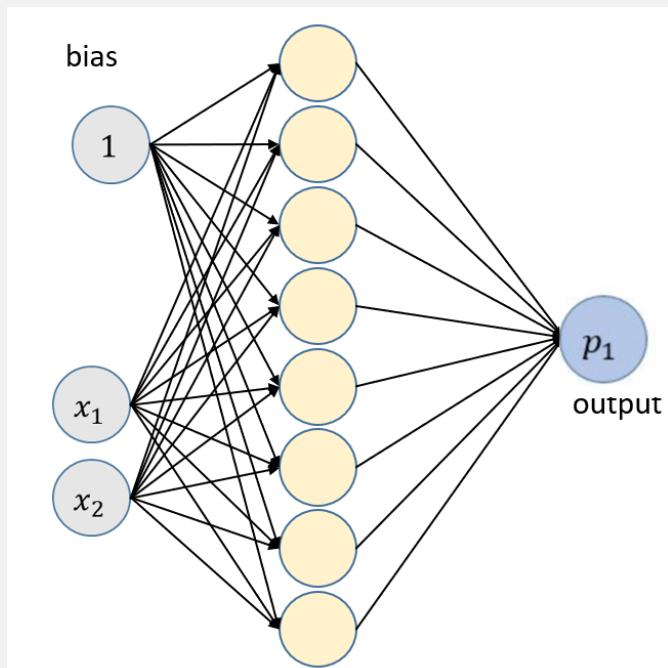
Example:

$$\mathbf{A}_{2 \times 2} = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$$

$$\mathbf{B}_{2 \times 3} = \begin{pmatrix} 3 & 1 & 7 \\ 8 & 2 & 4 \end{pmatrix}$$

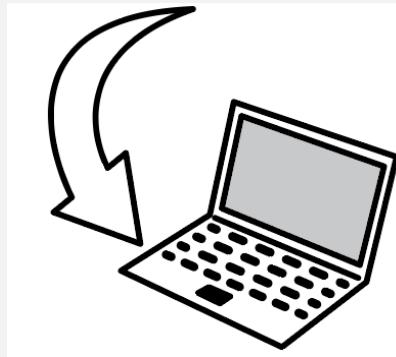
$$\mathbf{C}_{2 \times 3} = \mathbf{A}_{2 \times 2} \cdot \mathbf{B}_{2 \times 3} = \begin{pmatrix} 11 & 4 & 18 \\ 24 & 6 & 12 \end{pmatrix}$$

Exercise:



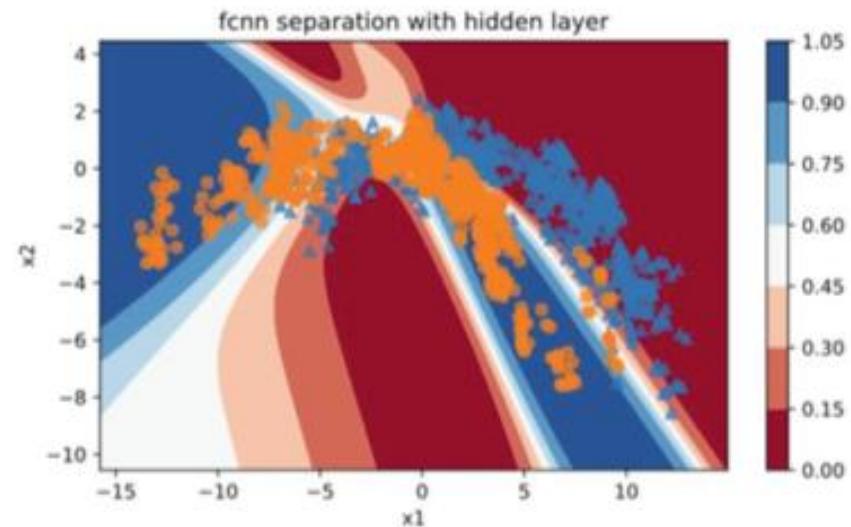
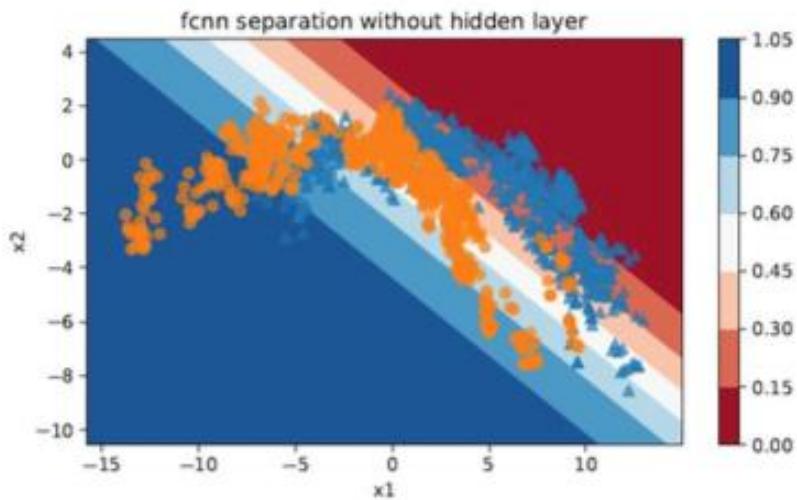
input

hidden

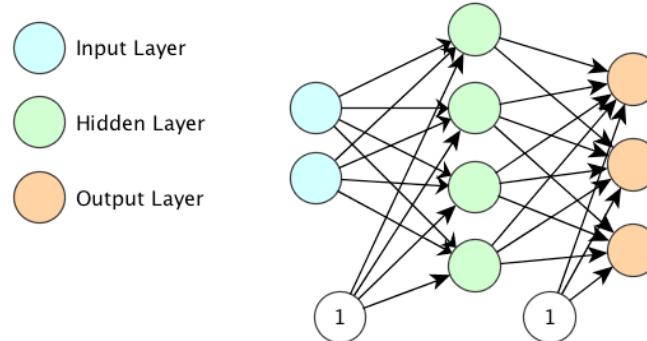


Open NB [01_simple_forward_pass.ipynb](#) and do exercise **stop** before Keras

Observations from NB: The benefit of hidden layers

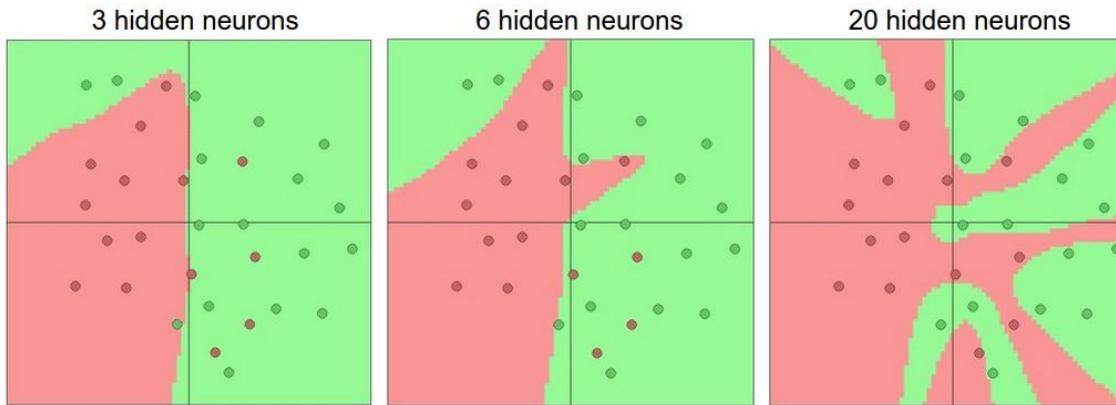


One hidden Layer is “in theory” enough



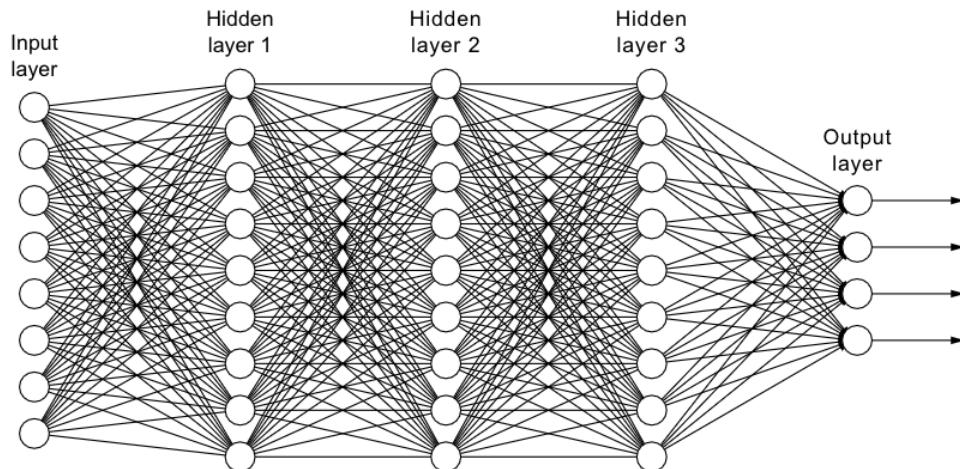
A network with one hidden layer is a universal function approximator!

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

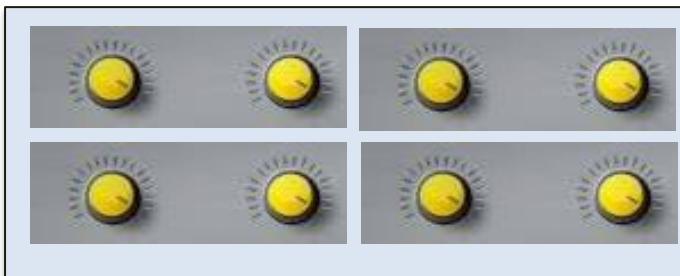


**Training NN:
Minimize the loss function**

How to determine the weights



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

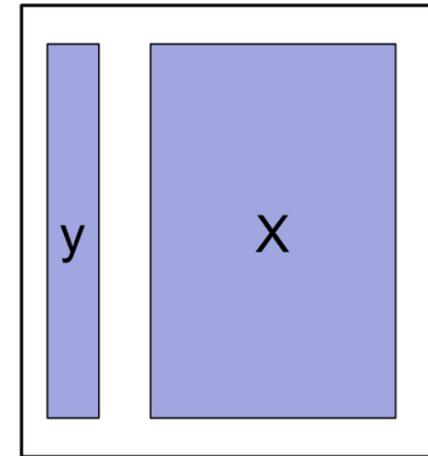


Typical >1 Mio. weights

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

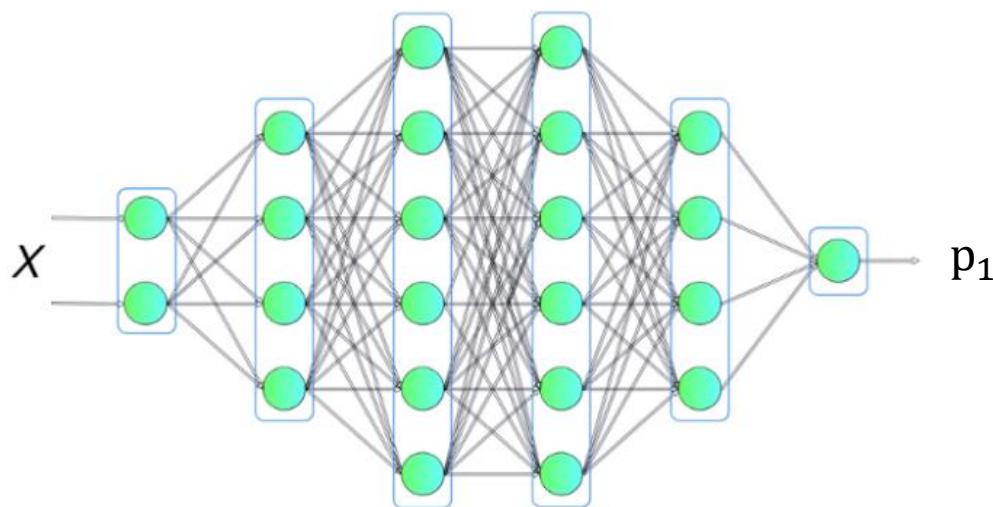
– Classification

- 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 $Y \in \{0, 1\}$**

– Regression

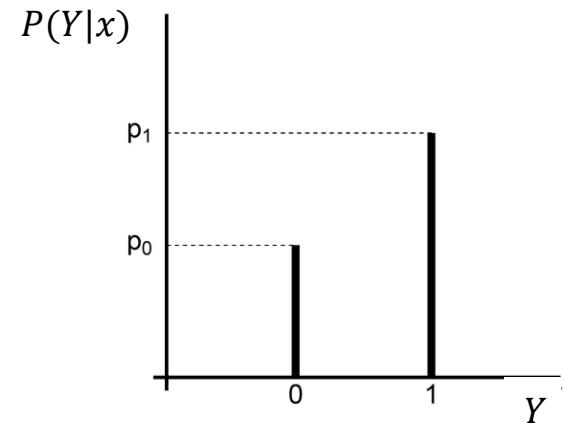
- Point prediction: Predict a number
- Probabilistic prediction: predict a continuous distributions

Example of a NN for binary classification



For binary classification we can use one neuron in the last layer with sigmoid activation yielding a conditional probability p_1 for $Y = 1$

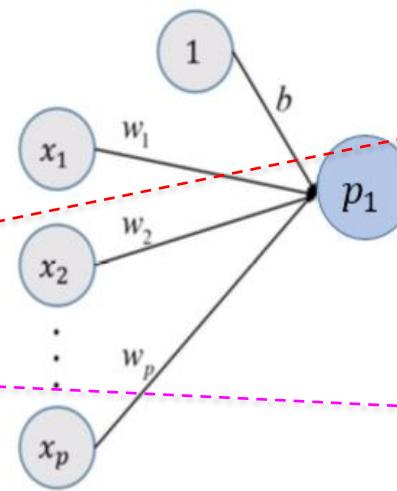
Given (=conditioned on) the input features x of an observation, the NN predicts as output the probability that this observation corresponds to class $Y = 1$ by $p_1 = P(Y = 1|x)$, and hence the probability for the $Y = 0$ is $p_0 = P(Y = 0|x) = 1 - p_1$.



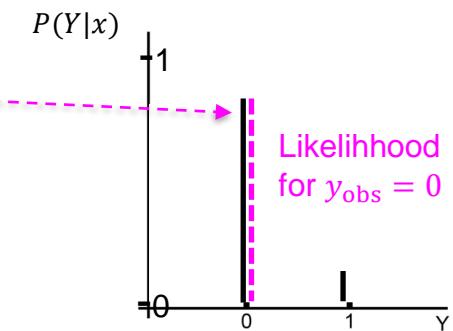
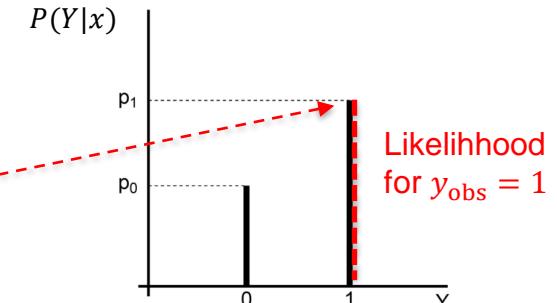
Probabilistic binary classification aka logistic regression

Training data set

x₁	...	x_p	y
0.4	...	-1.3	0
1.1	...	0.2	1
:	:	:	:
0.6	...	-0.9	0



probabilistic prediction



Given (=conditioned on) the input features x of an observation i , a well trained NN

- should predict large $p_1 = P(Y = 1|x)$ if the observed class is $y_i = 1$
 - should predict small p_1 hence large $p_0 = P(Y = 0|x)$ if observed is $y_i = 0$
- The likelihood (for the observed outcome) or LogLikelihood should be large

$$\text{LogLikelihood} = \sum_{i=1}^n [y_i \log(p_{1i}) + (1 - y_i) \log(1 - p_{1i})]$$

Notation trick in statistics – it selects correct log-probability since $y_i \in \{0,1\}$

Fitting a probabilistic binary classification

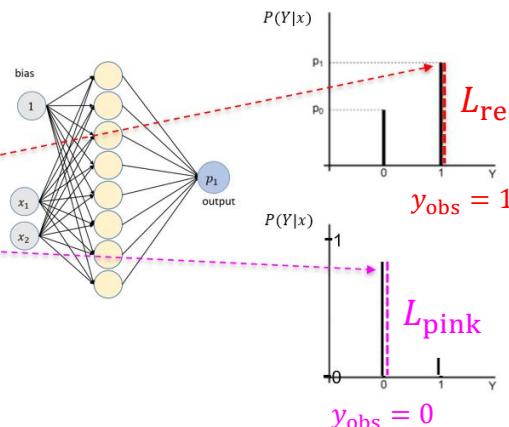
The Likelihood of an observation i is the likelihood (probability), that the predicted probability distribution $P(Y|x_i)$ assigns to the observed outcome y_i .

Note: the predicted $P(Y|x_i)$ and the corresponding likelihood for the observed y_i depends on the data-point (x_i, y_i) and the model parameter values

→ The higher the likelihood, the better is the model prediction $P(Y|x)$

Training data set

x1	...	xp	y
0.4	...	-1.3	0
1.1	...	0.2	1
:	:	:	:
0.6	...	-0.9	0



L_{red} L_{red} is likelihood of red observation with $y_{\text{obs}} = 1$

L_{pink} L_{pink} is likelihood of pink observation with $y_{\text{obs}} = 0$

Maximum likelihood principle:

Statistical models are fit to maximize the average LogLikelihood

$$L = \frac{1}{N} \sum L_i = \frac{1}{N} \sum [y_i \log(p_{1i}) + (1 - y_i) \log(1 - p_{1i})]$$

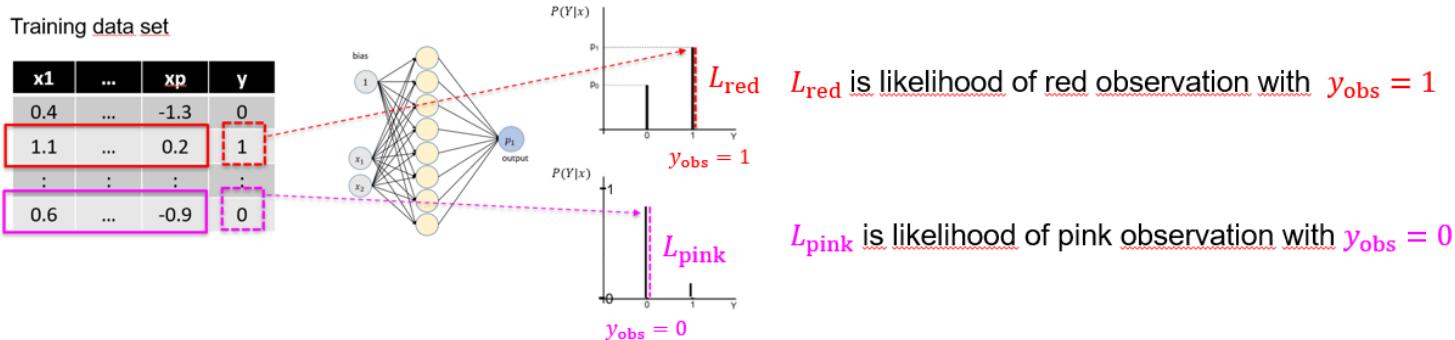
we often use the simplified notation average logLik : $L = \frac{1}{N} \sum \log(p_i)$

Predicted probability for
the observed outcome y_{obs}

NLL Loss for probabilistic binary classification

In DL we aim to minimize a loss function $L(\text{data}, w)$ which depends on the weights
→ Instead of maximizing the average LogLikelihood
we minimize the averaged Negative LogLikelihood (NLL):

$$\text{loss} = \text{NLL} = -\frac{1}{N} \sum \log(L_i)$$



The best possible value of the NLL contribution of an observation i is $\log(L_i) = -\log(1) = 0$

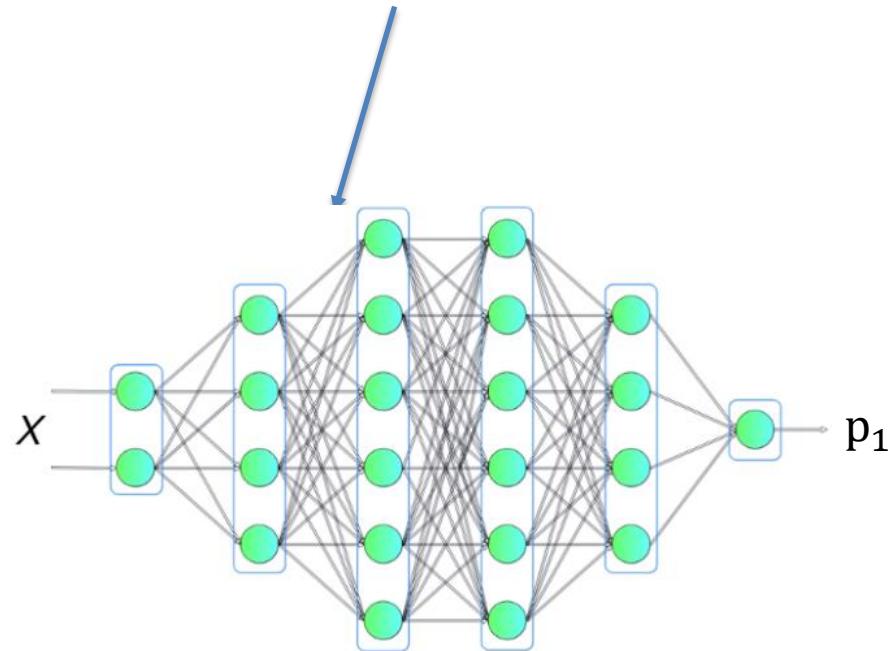
The worst possible value of the NLL contribution of an observation i $\log(L_i) = -\log(0) = \infty$

Note: In Keras we use the loss 'binary_crossentropy', if we do probabilistic binary classification with one output node with sigmoid activation.

Optimization in DL

- DL 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

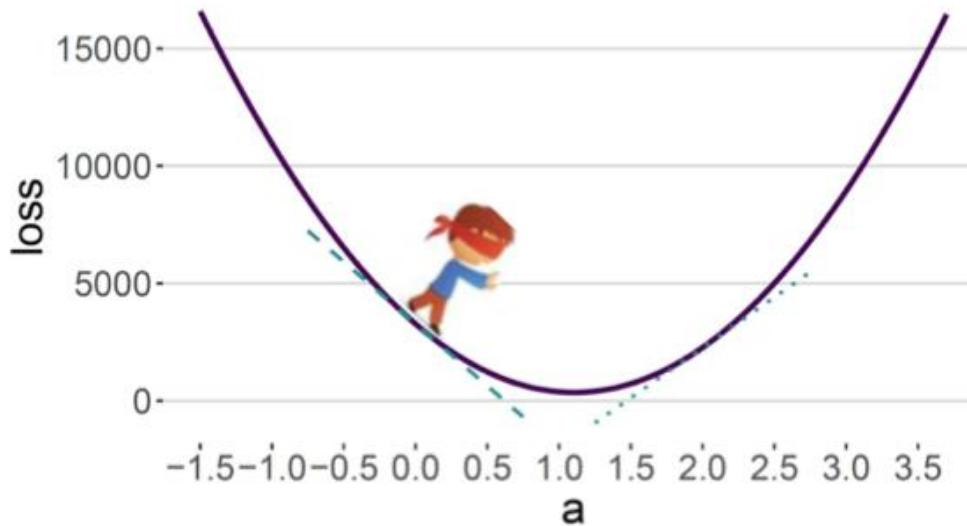
Parameters of the NN are the weights.



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

Idea of gradient descent

- Shown loss function for a single parameter a

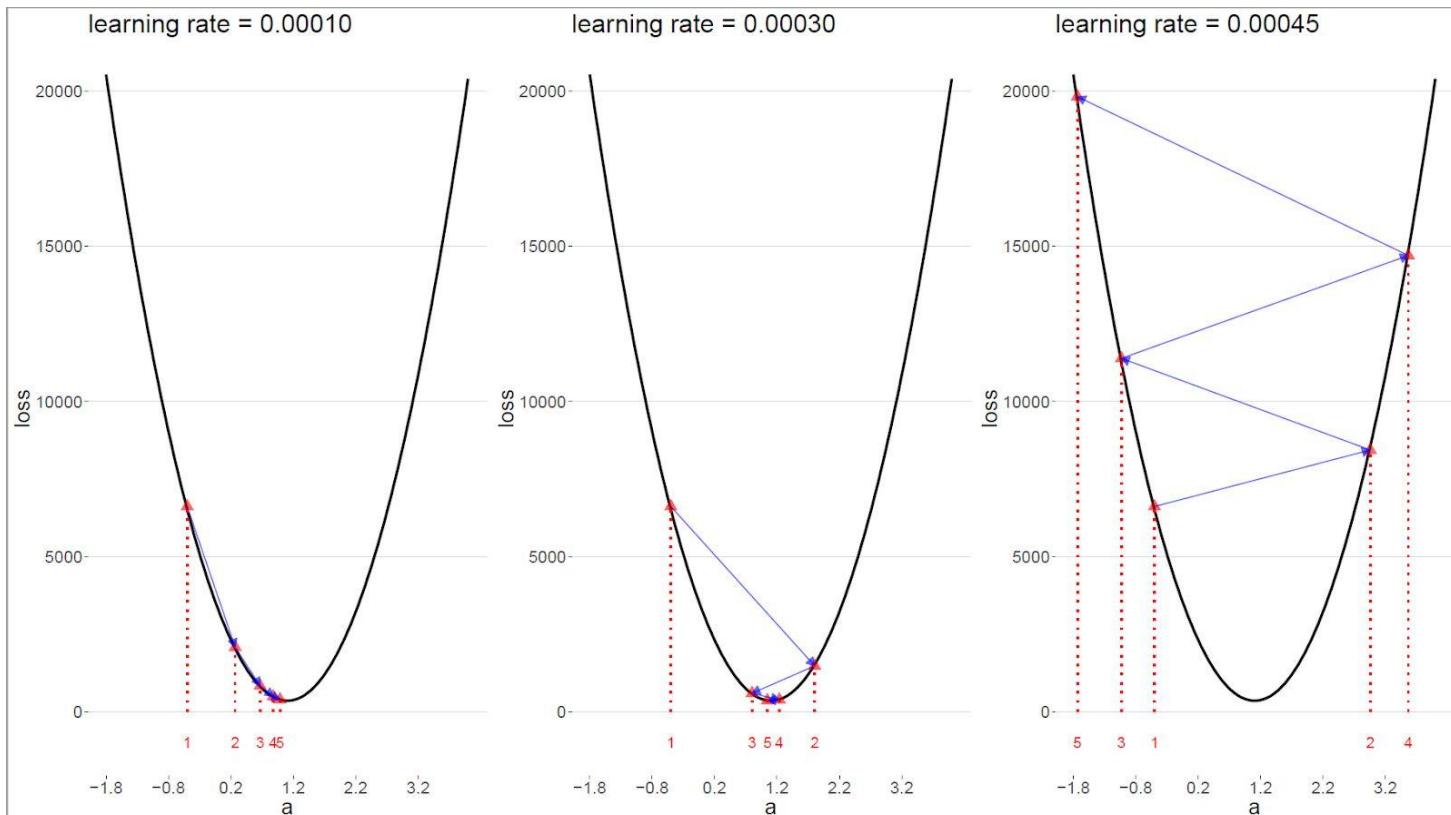


- Take a large step if slope is steep (you are away from minimum)
- Slope of loss function is given by gradient of the loss w.r.t. a
- Iterative update of the parameter a

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

learning rate

The learning rate is a very important parameter for DL



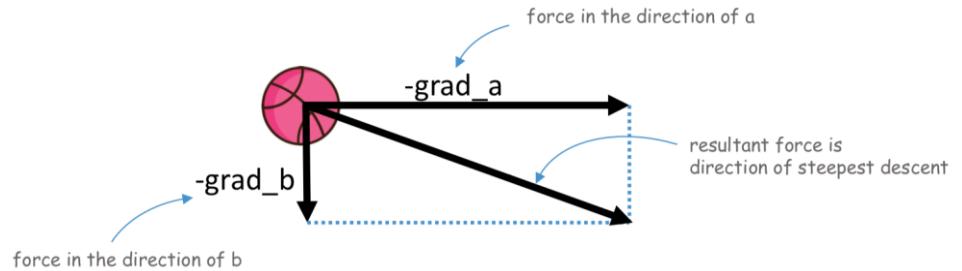
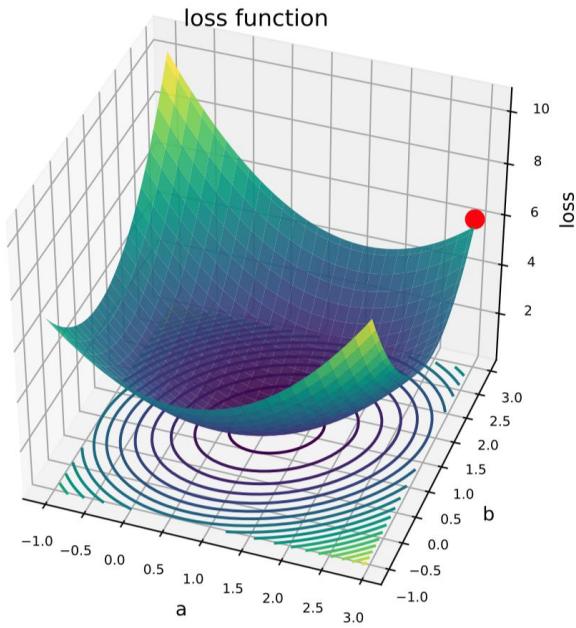
good – little slow

good

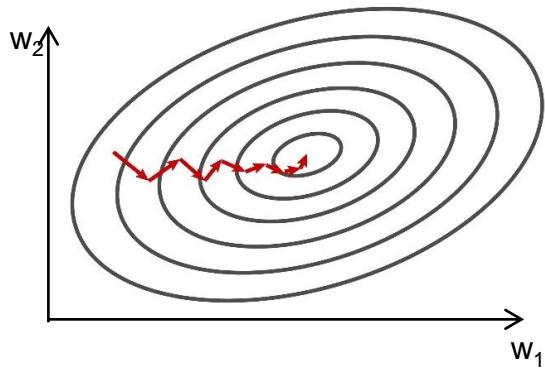
learning rate too high

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

In two dimensions



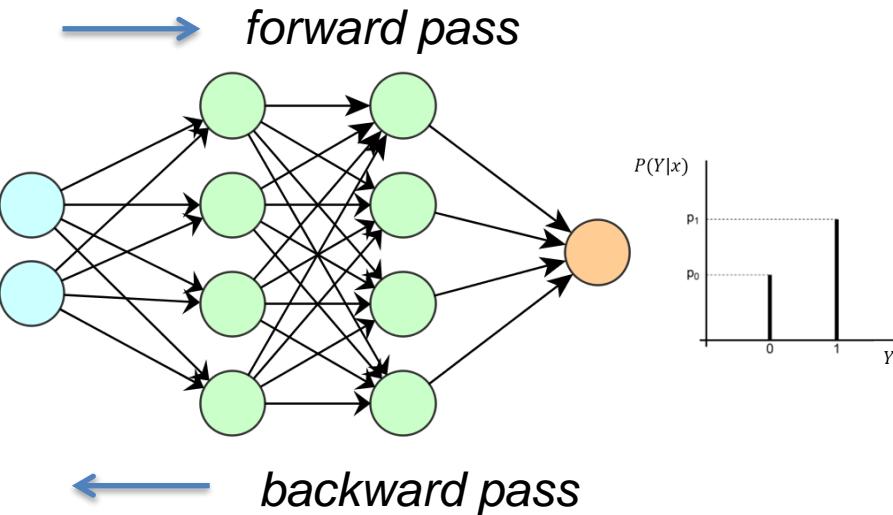
Gradient is perpendicular to contour lines



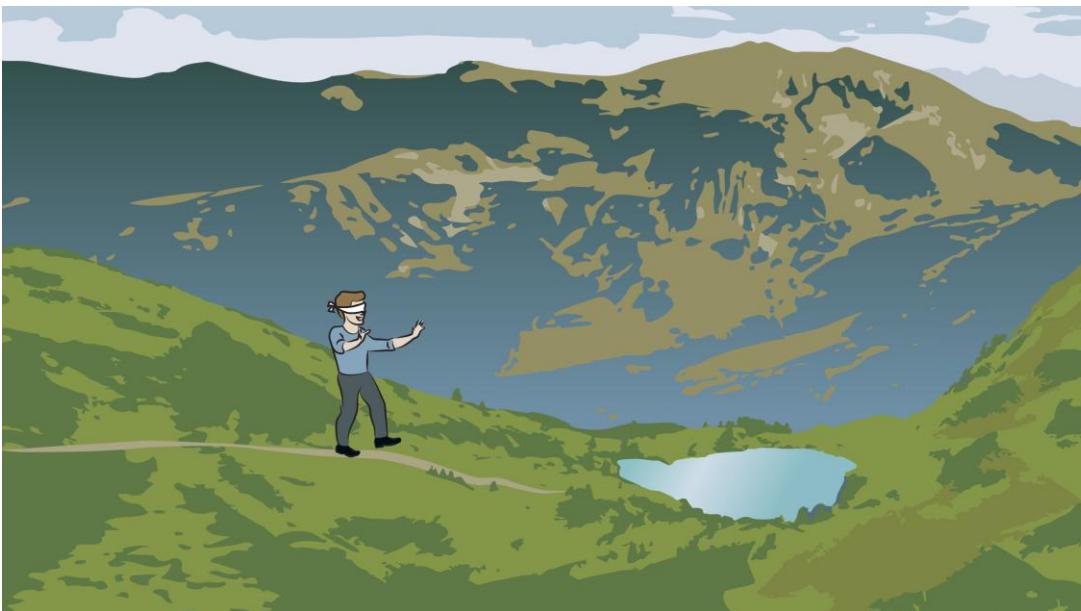
$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial L(\mathbf{w})}{\partial w_i} \right|_{w_i=w_i^{(t-1)}}$$

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



The miracle of gradient descent in DL



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

Typical Training Curve / ReLU

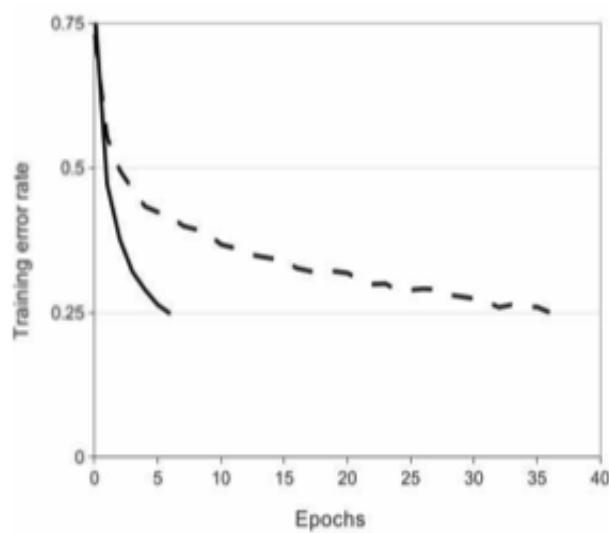
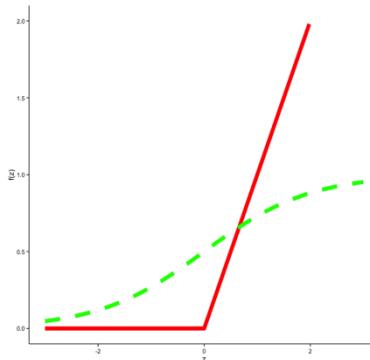
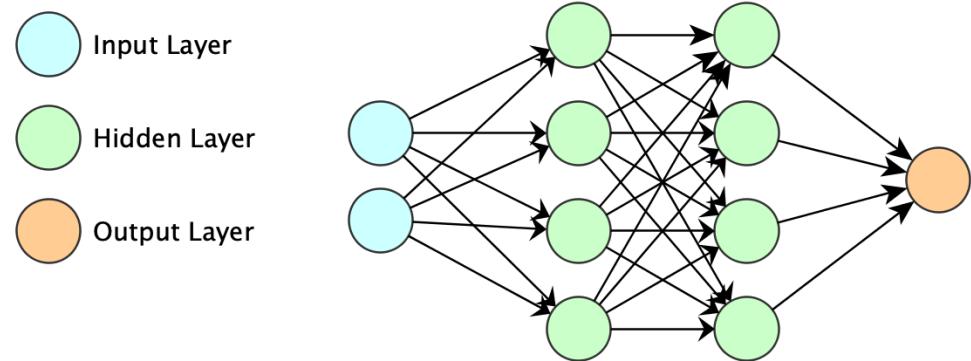


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

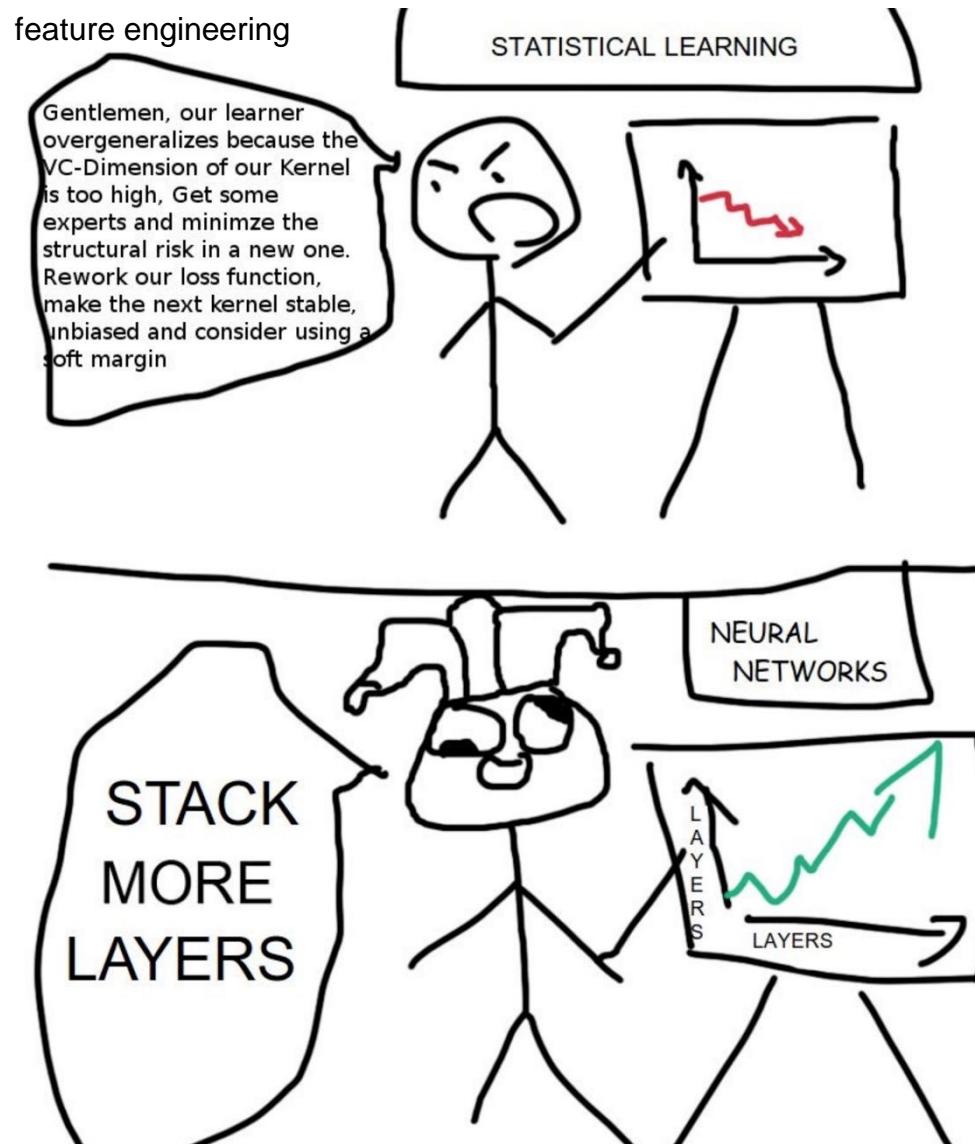
Source:
Alexnet
Krizhevsky et al 2012



Motivation:
Green:
sigmoid.
Red:
ReLU faster
convergence

Epochs: "each training examples is used once"

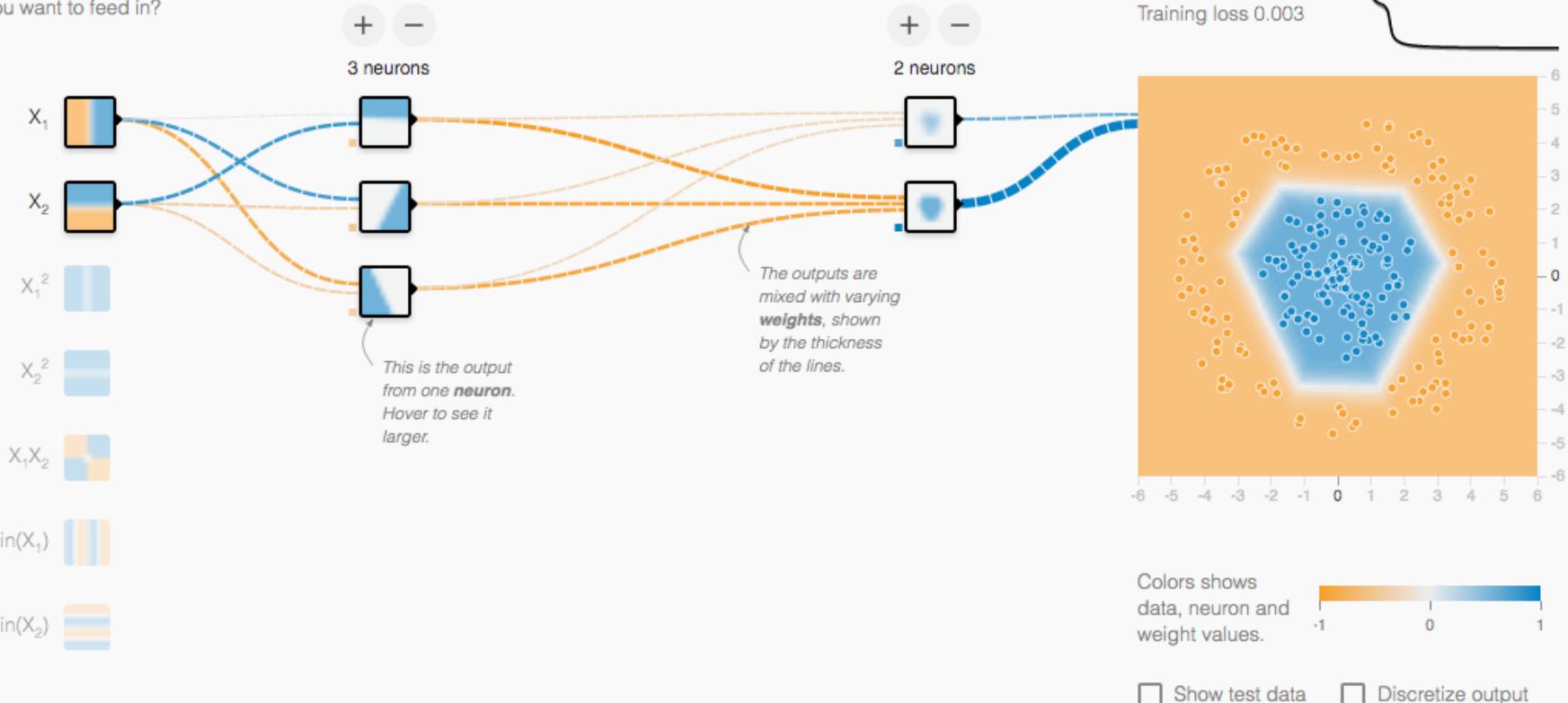
Game time - Recall learning DL vs Statistical Learning



Experiment yourself (homework)

FEATURES

Which properties do you want to feed in?



<http://playground.tensorflow.org>

Let's you explore the effect of hidden layers

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]])

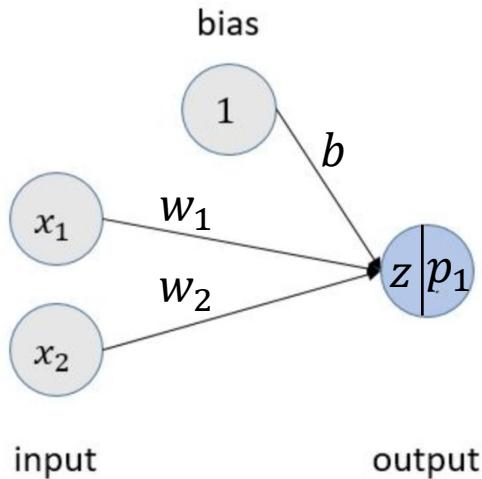
In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)

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

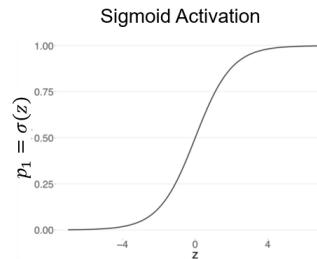
(3,)
(2, 3)
(10, 2, 3)
```

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

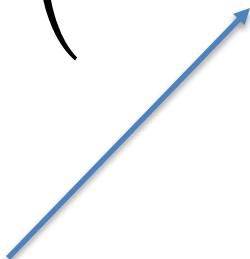
GPUs love Vectors



$F^{\mu\nu}$

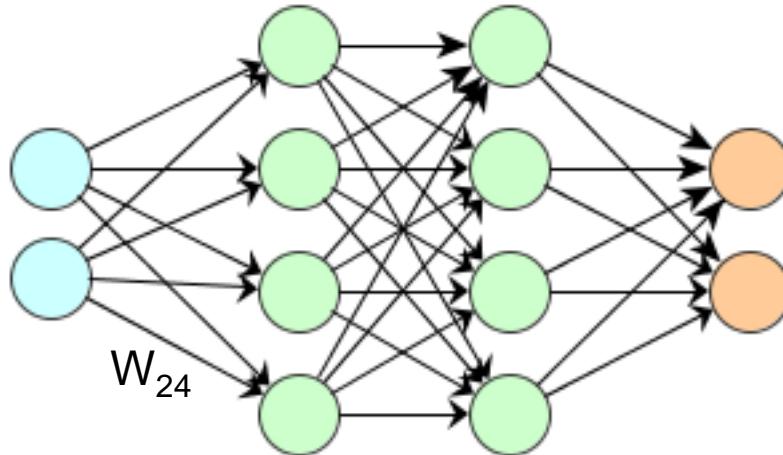


$$p_1 = \text{sigmoid} \left((x_1 \quad x_2) \cdot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + b \right)$$



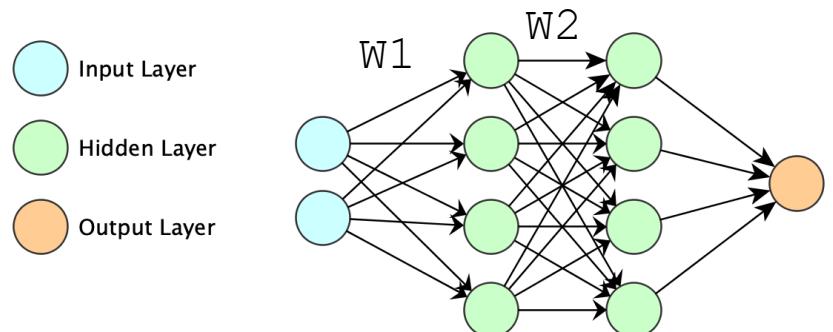
DL: better to have column vectors

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 ($f = \text{sigmoid}$) and $b1=b2=b3=0$

$$p = f(f(f(x W^1)W^2))$$

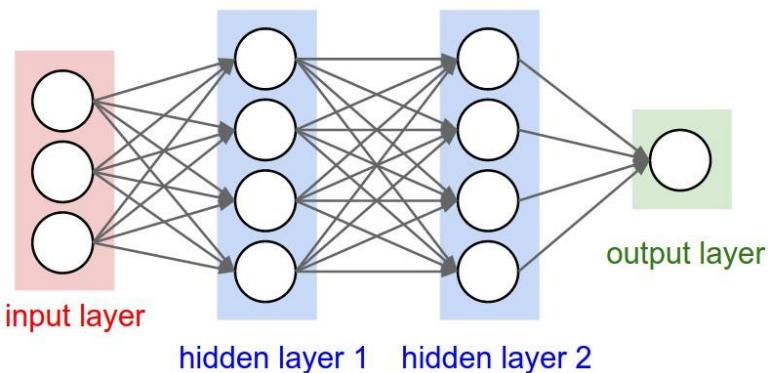
Defining the NN with code is done with so-called low-level libraries (here pytorch)
Looks a bit like onions, matryoshka (Russian Dolls) or lego bricks...

Question: Dimensions of $W1$, $W2$, and $W2$

Keras as High-Level library



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



High Level Libraries



Under the “hood” of Keras runs the engine;

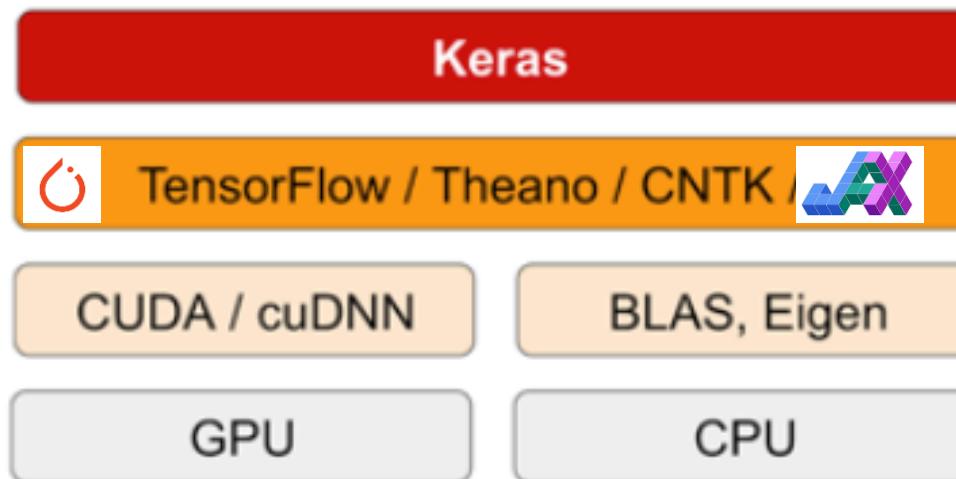


Figure: From Deep Learning with Python, Francois Chollett

Load packages

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



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

Load building blocks

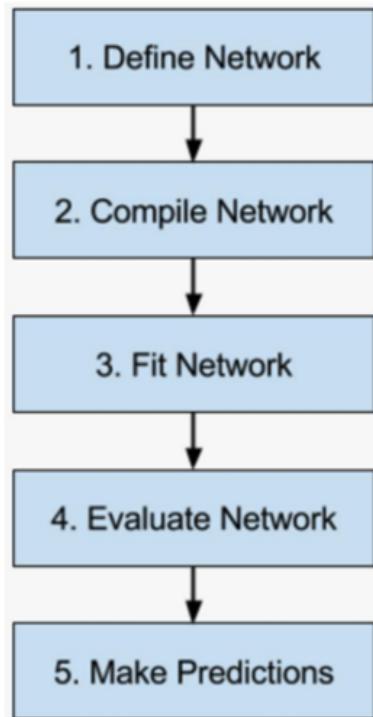
- Layers, functions etc.

```
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



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

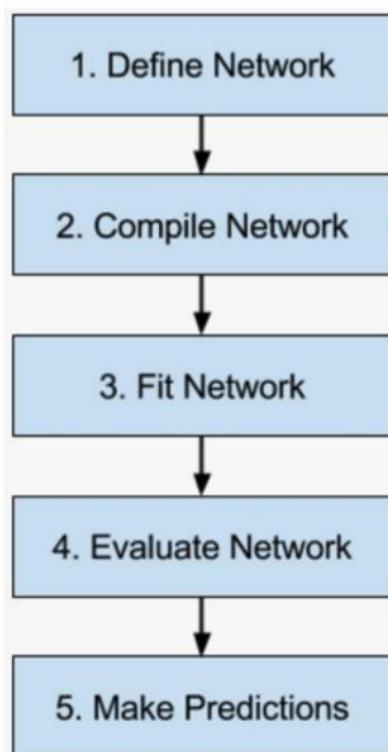
Hidden layer with 8 Neurons,
Activation function: sigmoid

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



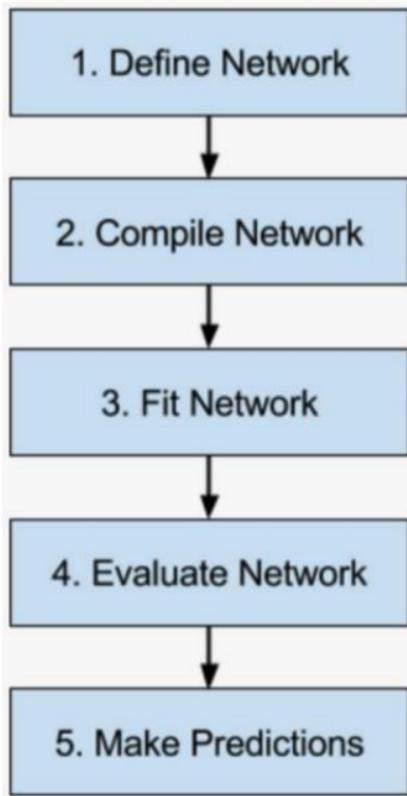
```
model.compile(optimizer=SGD(learning_rate=0.01),  
              loss='binary_crossentropy',  
              metrics=['accuracy'])
```

Loss Function to optimize
Here: Binary CE

Which optimizer should be used?
Here Stochastic Gradient Descent

Which metrics do we want to track,
Here: Accuracy

Fit the network



Data Tensor X for training

Label Tensor Y

Validation at
End of each epoch

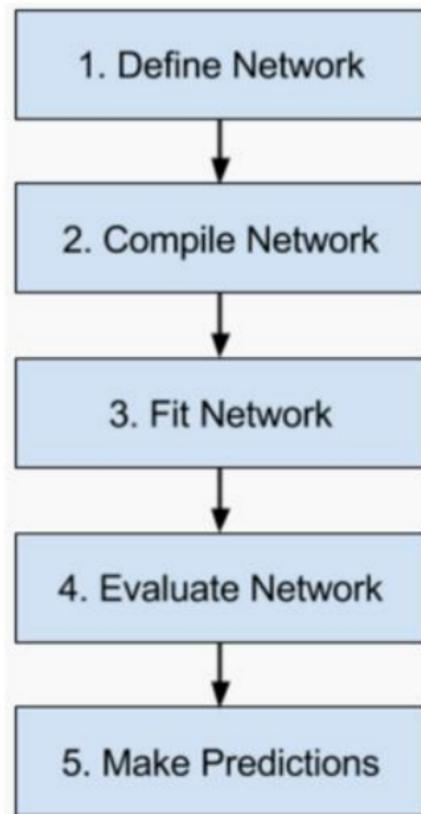
```
history = model.fit(x_train,  
                     y_train,  
                     validation_data=(x_val, y_val),  
                     epochs=1000,  
                     batch_size=10,  
                     verbose=1)
```

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

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

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

Evaluate the network



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

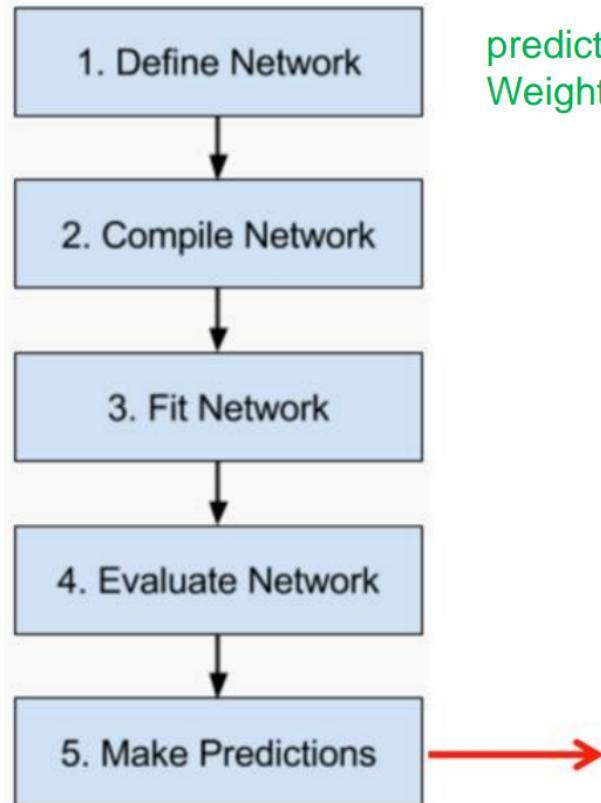
```
model.evaluate(x_test, y_test, verbose=0)
```

Test Loss: 0.33591118454933167, Test Accuracy: 0.8581818342208862

Remember from the
.compile

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

Make Predictions



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

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)
```

Functional (do you spot the error?)

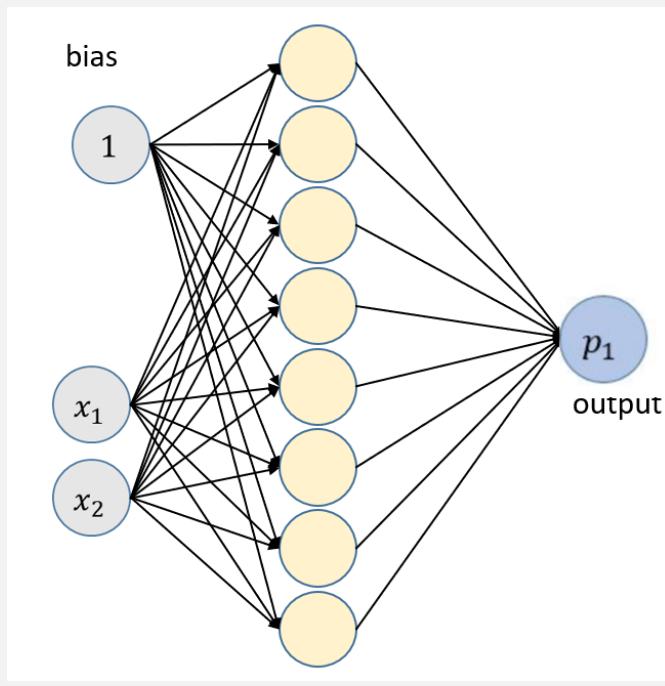
The functional API

```
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)

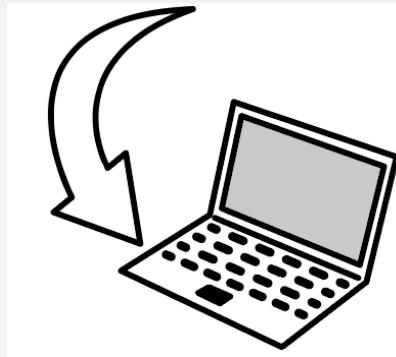
model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Exercise:



input

hidden



Open NB [01_fcnn_with_banknotes.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 backpropagation 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!