Machine Intelligence:: Deep Learning
Week 1

Beate Sick, Oliver Dürr, Pascal Bühler

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

Beate's Background

Head of bioinformatics

Focus: Gene expression

Researcher and Professor for applied statistics

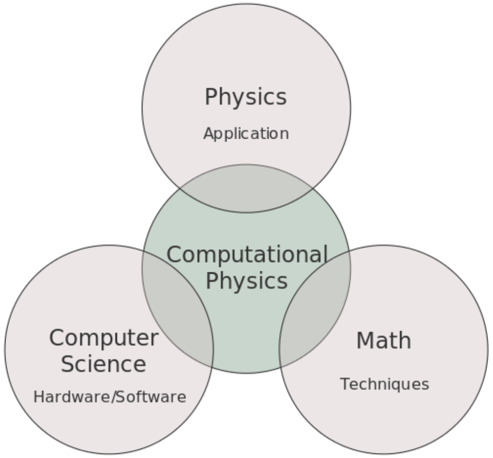
Focus: deep learning

Researcher and lecturer

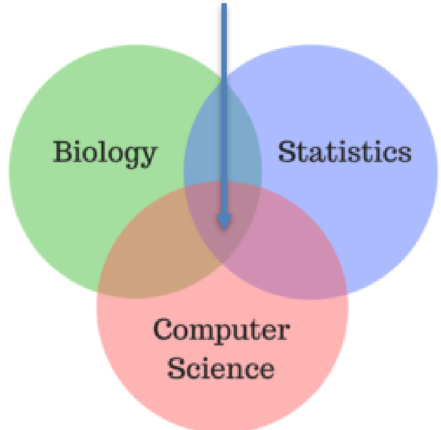
Focus: Biostatistics, DL

Oliver's Background

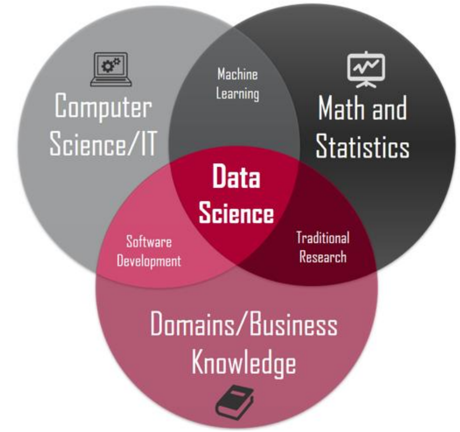
Computational Physics



Bioinformatics



Data Science



1990's

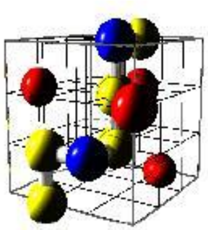
Uni-Konstanz

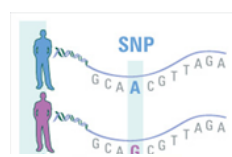
2000's

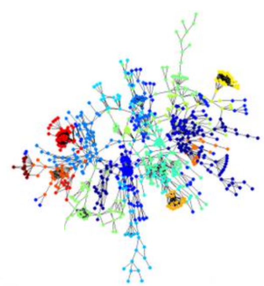
Genedata Basel

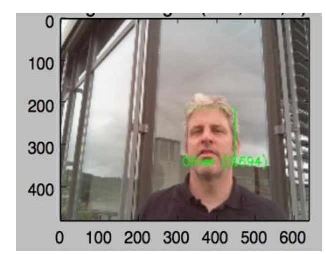
2010's

ZHAW Winterthur HTWG Konstanz









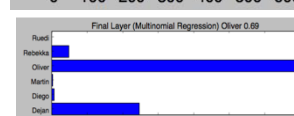


Image credits: Wikipedia, http://omgenomics.com/what-is-bioinformatics/ https://www.fox.temple.edu/institutes-and-centers/data-science/

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_2024/prerequistites.html

- Any contacts with deep learning yet?

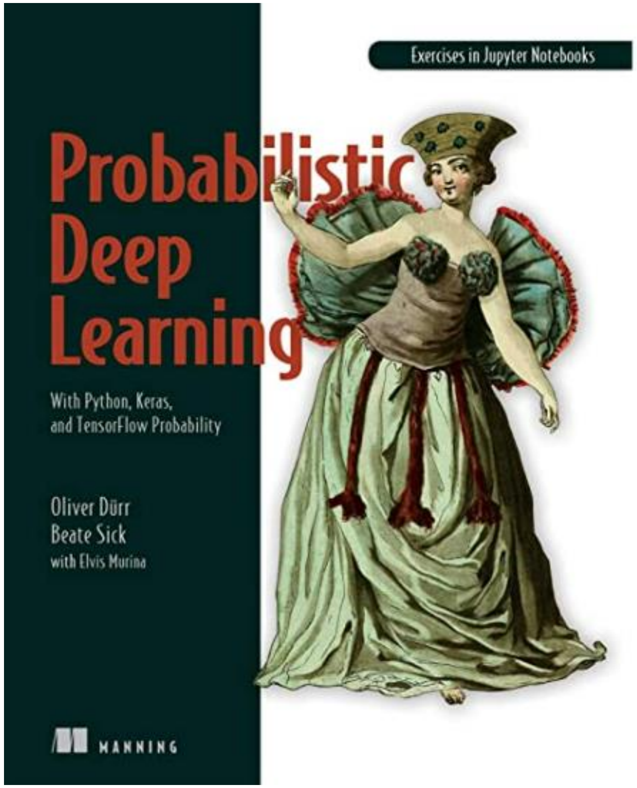
Technical details for this course

- Running the code:
  - Colab Notebooks
    - needs no installation, only internet and google account
  - Anaconda
    - Installation by your own (no support)

Material for the course

- Website and Github repository
  - The CAS Deep Learning Course
    - https://tensorchiefs.github.io/dl_course_2024/

- Our Book "Probabilistic Deep Learning"
  - Can be used in addition to the course
  - https://github.com/tensorchiefs/dl_book



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)

Organizational Issues: Times

- Dates and times: see our webpage [CAS machine intelligence]
- Afternoon sessions
  - 13:30-17:00
- Theory and exercises will be mixed
  - Could be 50 minutes theory 30 minutes exercises
  - Could be vice versa
- **Please interrupt us if something is unclear! The less we talk the better!**

Outline of the DL Module (tentative)

- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II and RNN
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Day 4: Looking at details
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle
- Day 5: Probabilistic Aspects
  - TensorFlow Probability (TFP)
  - Negative Loss Likelihood NLL
  - Count Data
- Day 6: Probabilistic models in the wild
  - Complex Distributions
  - Generative modes with normalizing flows
- Day 7: Uncertainty in DL
  - Bayesian Modeling
- Day 8: Uncertainty cont'd
  - Bayesian Neural Networks
  - Projects

Day 1-4 should get you ready for your project.

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

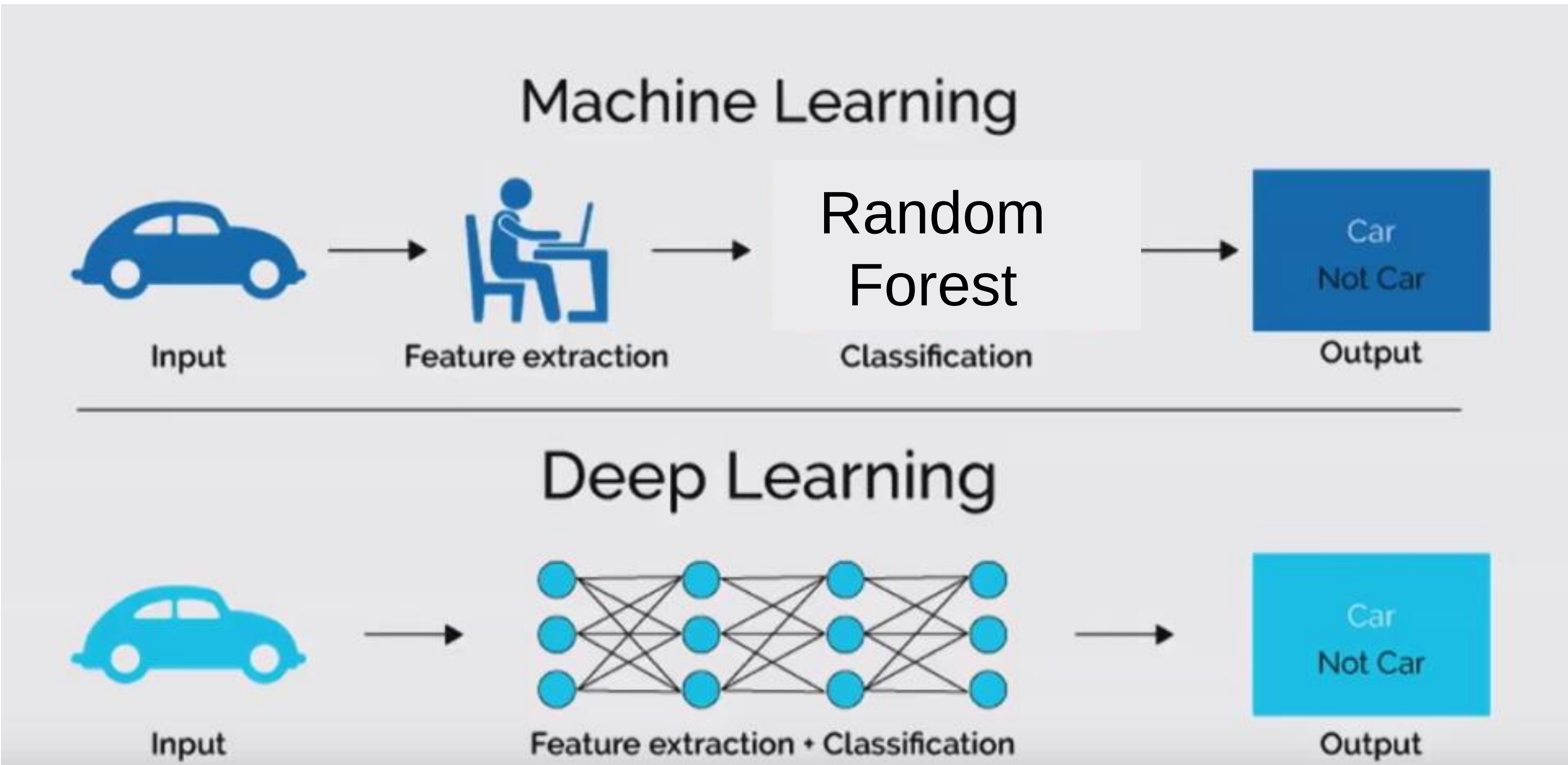
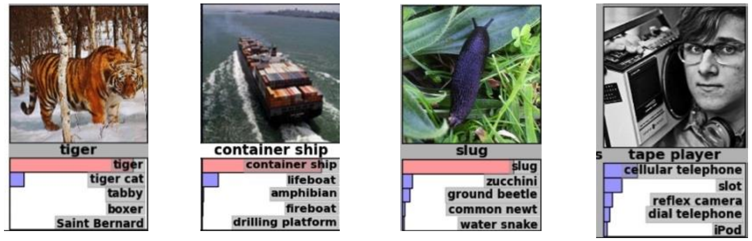


image credit: adapted from [Siraj Raval]

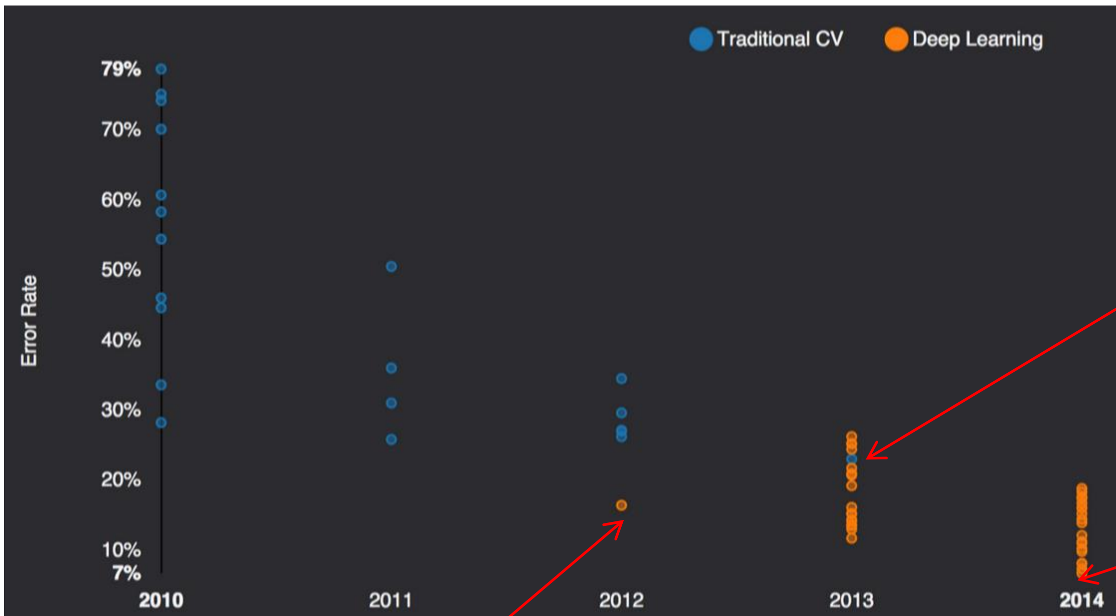
The most convincing case for DL (subjective view)

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!

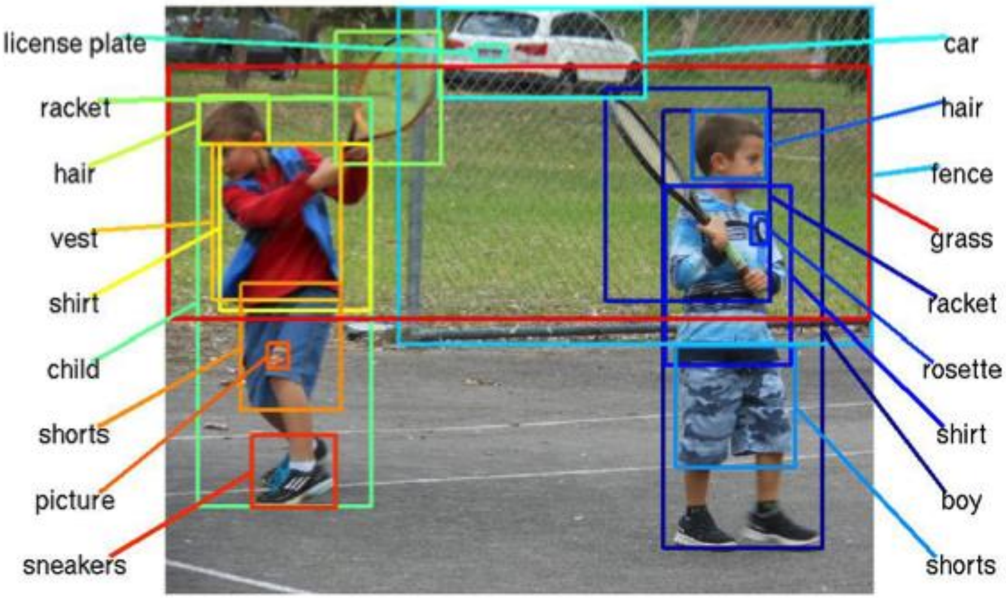
Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

15

The computer vision success story

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

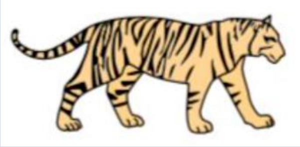
Deep Blue beat Kasparov at chess in 1997.
Watson beat the brightest trivia minds at Jeopardy in 2011.
Can you tell Fido from Mittens in 2013?



"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 (prompt) An astronaut riding a horse in a photorealistic style		Image Generation
Sequences (prompt) "Hallo, wie?"	Next word "geht"	Language Models
Simple number (age) age=52	Simple number (SPB) sbp = 152	Simple Regression Educational

Deep Learning öffnet Tür zu hören, sehen und Texten.
Status Quo: kein Verstehen aber Erfassung statistische Zusammenhänge.

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)

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

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

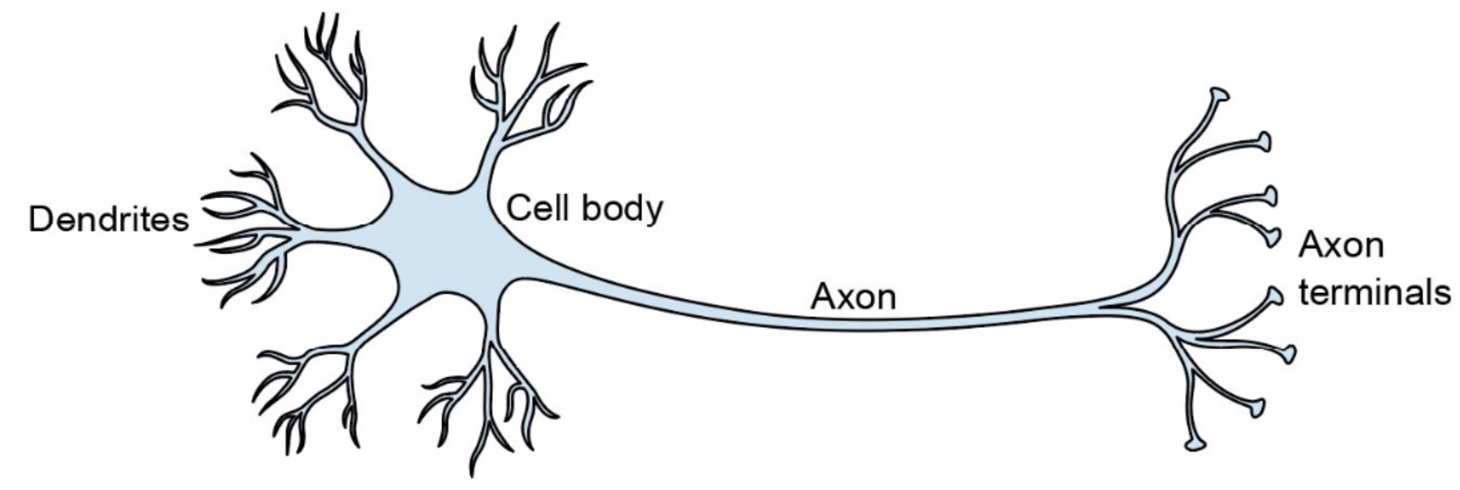
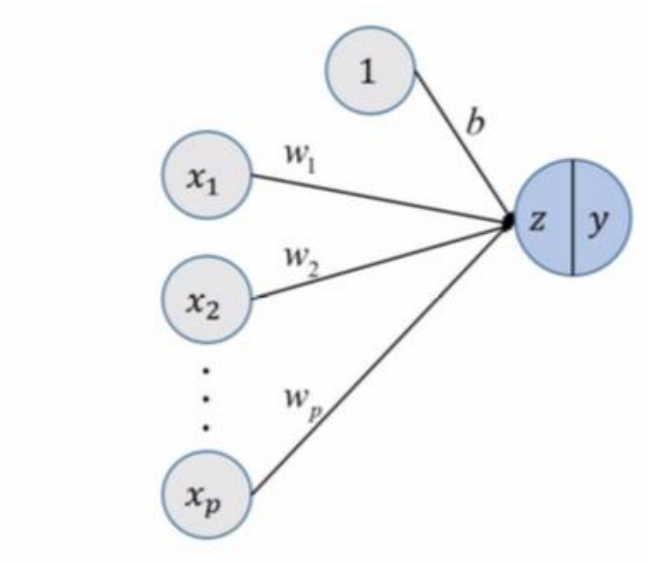
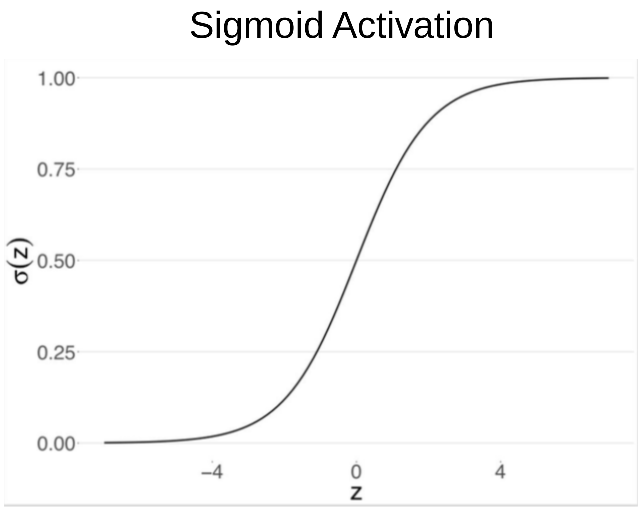


Figure 2.2 A single biological brain cell. The neuron receives the signal from other neurons via its dendrites shown on the left. If the cumulated signal exceeds a certain value, an impulse is sent via the axon to the axon terminals, which, in turn, couples to other neurons.

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

The Single Neuron: Mathematical Abstraction



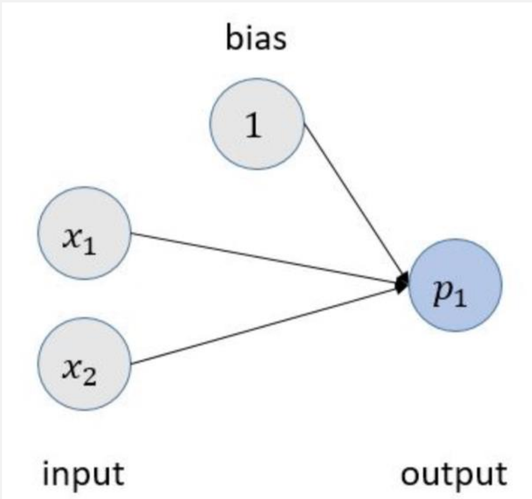


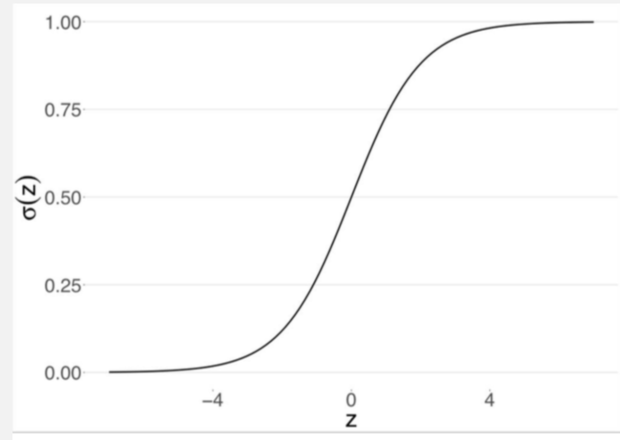
$$z = b + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots x_p \cdot w_p$$

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

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:

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

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

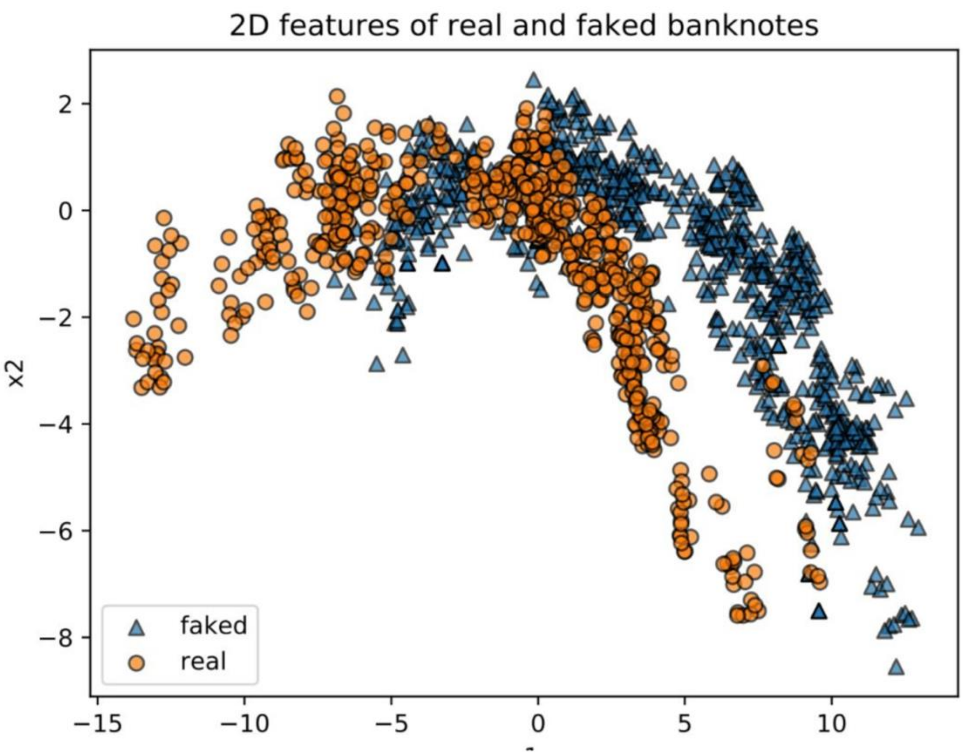
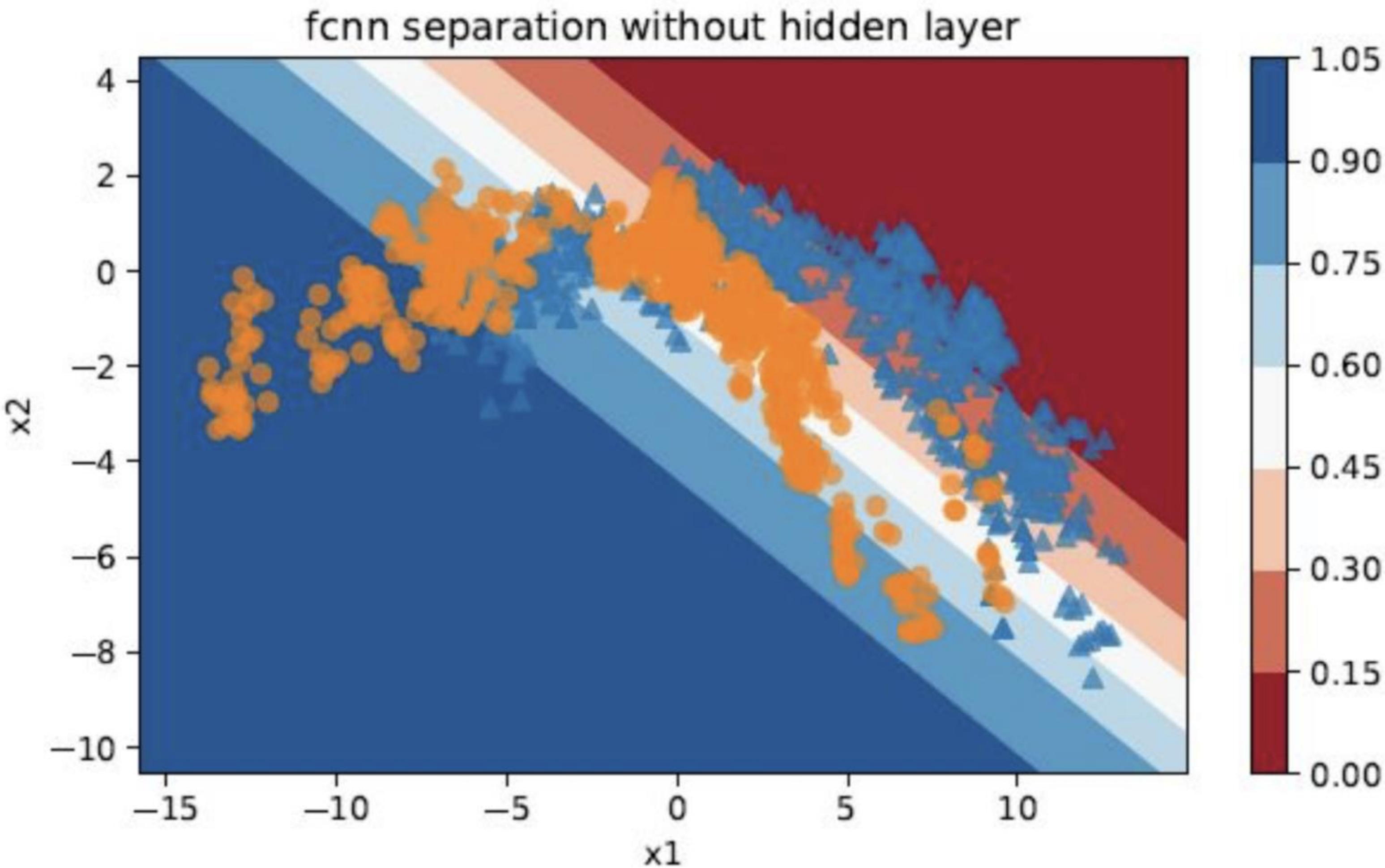


Figure 2.5 The (training) data points for the real and faked banknotes

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

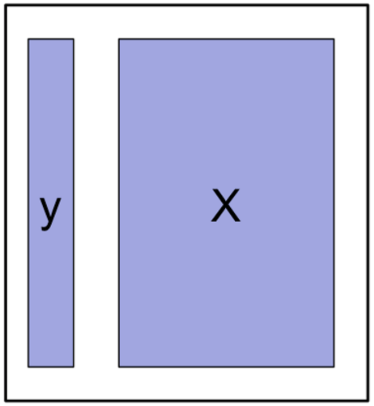
Result (see later in the notebook)



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

Our take on Deep Learning: Probabilistic Viewpoint

Tasks in supervised DL



Supervised Learning

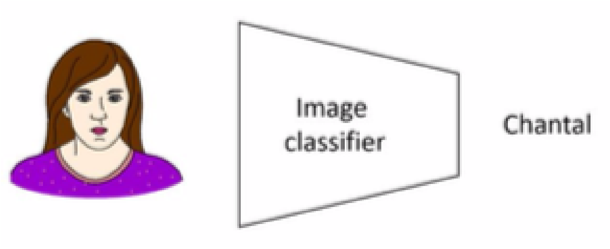
- 2 Main tasks in DL predict y given x
  - **Classification**
    - Point prediction: Predict a class label
    - Probabilistic prediction:
      predict a discrete probability distribution over all possible class labels
  - Regression
    - Point prediction: Predict a number
    - Probabilistic prediction: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

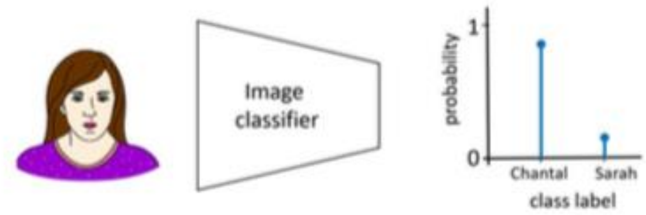
Probabilistic vs deterministic models

Deterministic

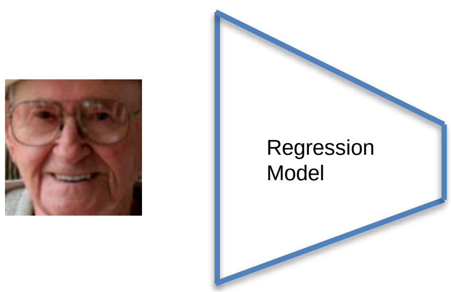
Probabilistic

"Classification"

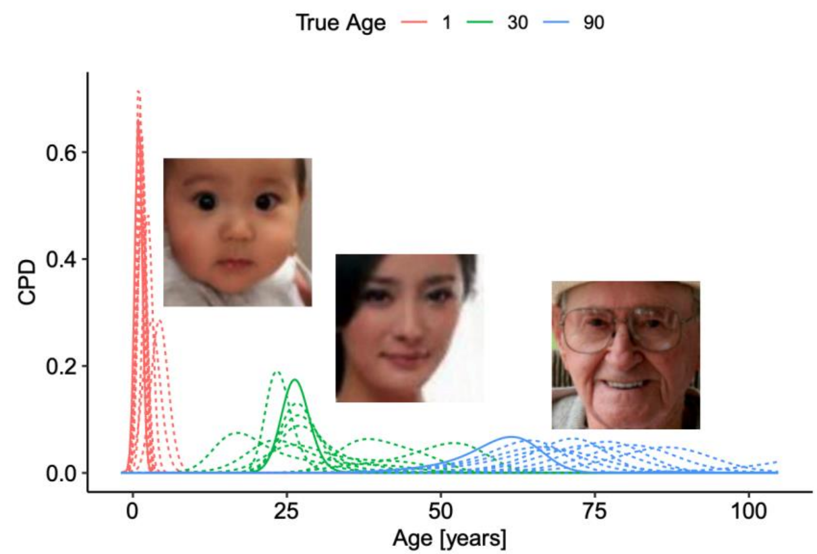




"Regression"



74



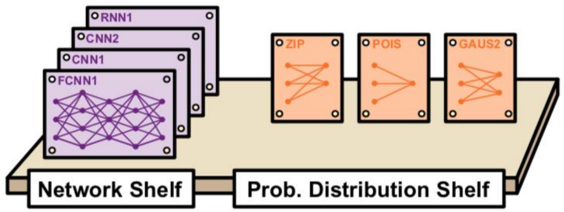
Conditional probability distribution (CPD)

$p(y|x)$

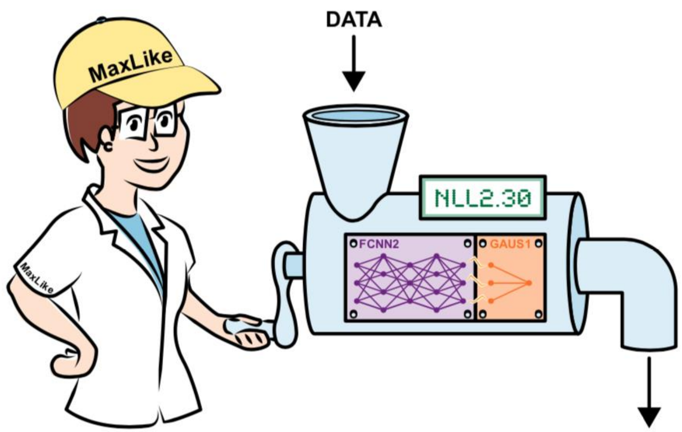
Guiding Theme of the course

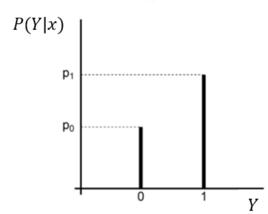
- We treat DL as *probabilistic models*, as statistical model (logistic regression, ...) to predict the conditional probability distribution $P(Y|x)$ for the outcome
- The models are fitted to training data with maximum likelihood (or Bayes)

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

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

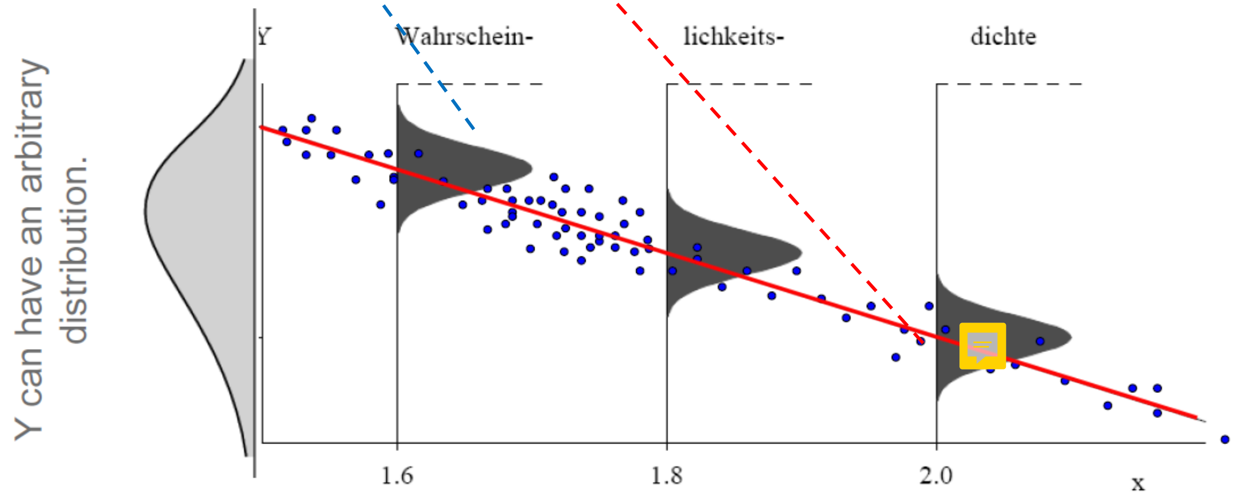
$Y \in \mathbb{R}$,
$\mu_x \in \mathbb{R}$

$$y_i = \beta_0 + \beta_1 \cdot x_{i1} + \varepsilon_i$$
$$E(Y_{X_i}) = \mu_{x_i} = \hat{y}_{x_i} = \beta_0 + \beta_1 \cdot x_{i1}$$
$$\mathrm{Var}(Y_{X_i}) = \mathrm{Var}(\varepsilon_i) = \sigma^2$$
$$\varepsilon_i \sim N(0, \sigma^2)$$

point prediction

probabilistic prediction

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

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

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

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

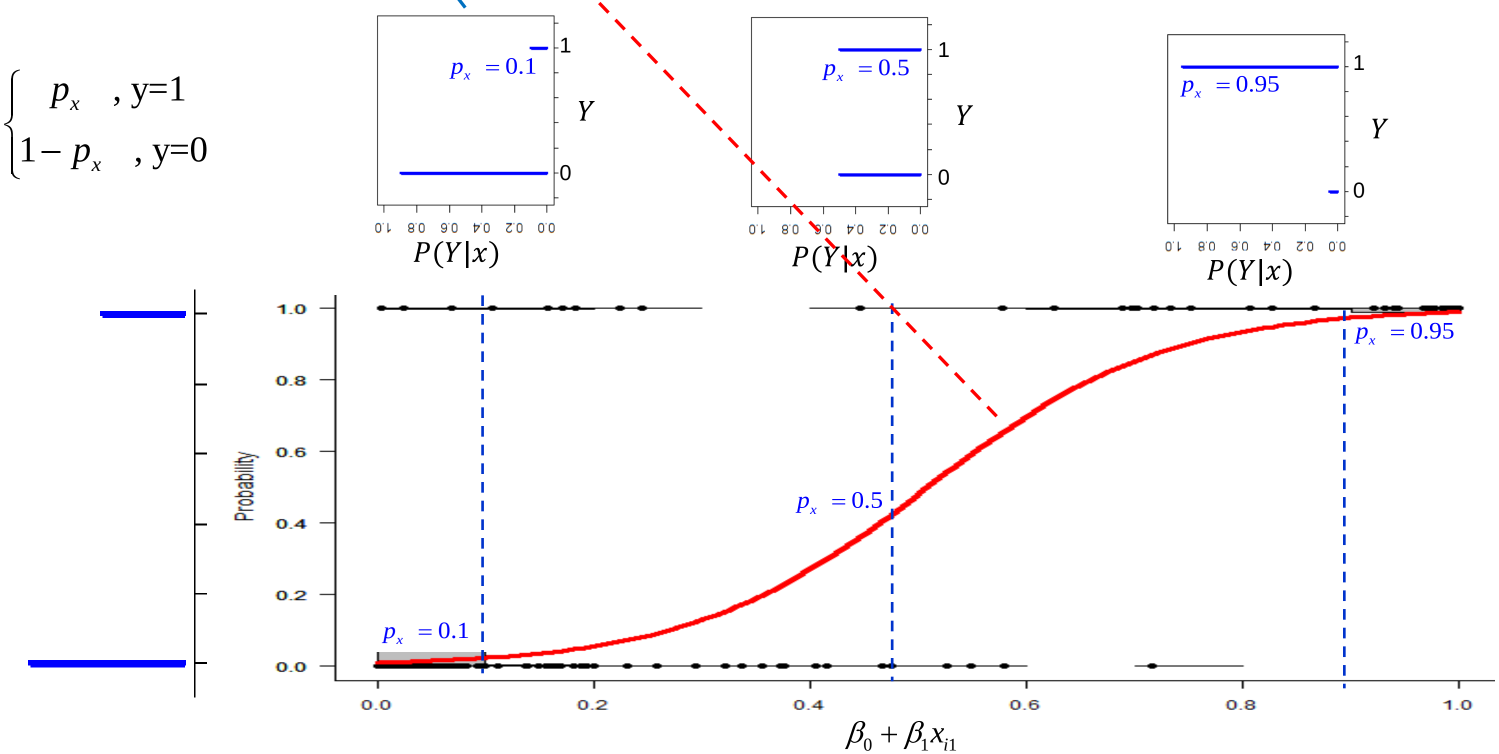
$$\log\left(\frac{p_{x_i}}{1-p_{x_i}}\right) = \beta_0 + \beta_1 x_{i1}$$

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

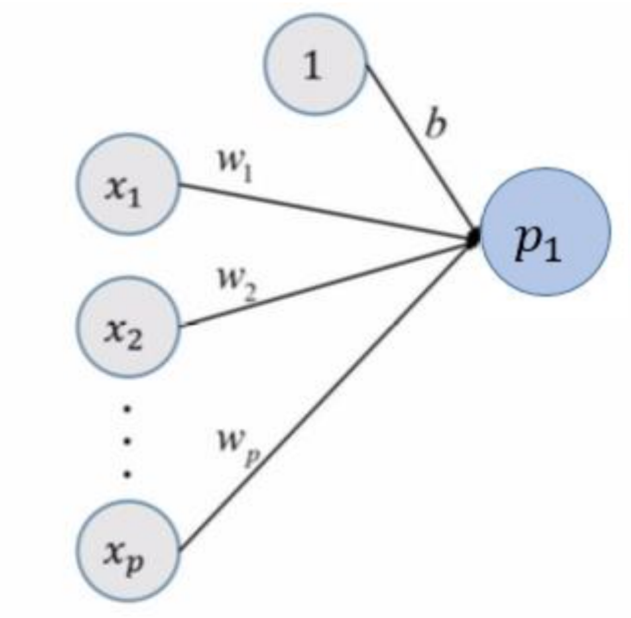
probabilistic prediction

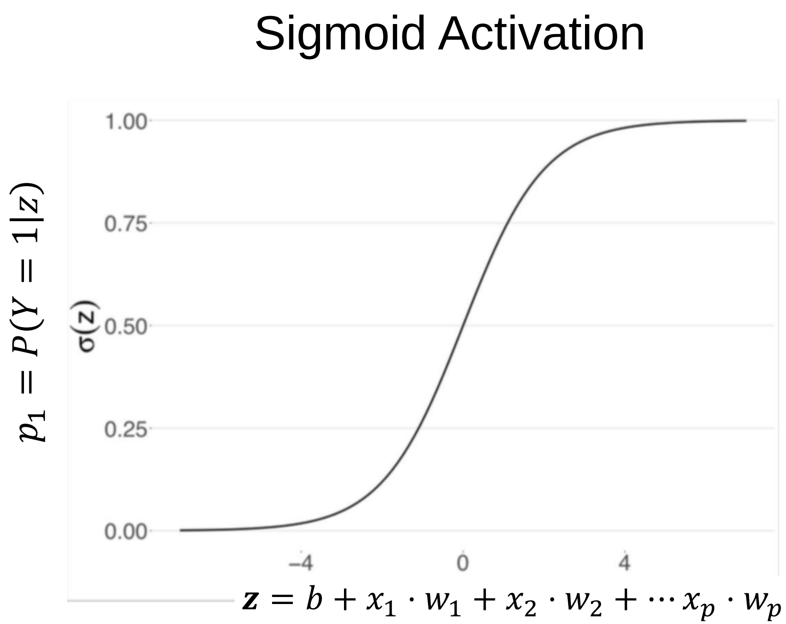
Logisic regression predicts a conditional Bernoulli $P(Y|x_i)$

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



Logistic regression in DL view





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

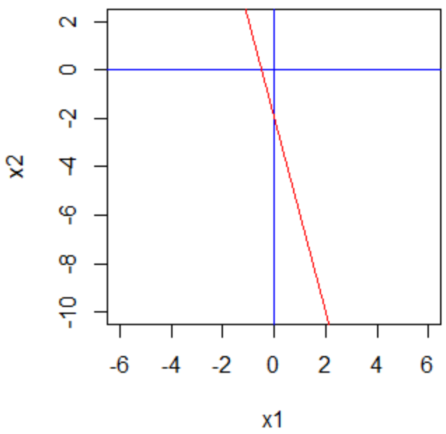
A NN for a binary outcome with only 1 neuron with sigmoid-activation (and no hidden layer) is nothing else than logistic regression!

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_2.

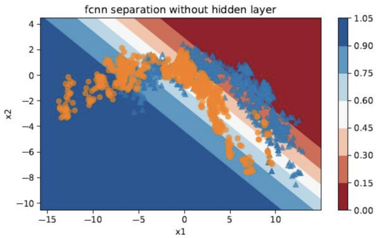
Hint: on the separation curve should hold: $P(Y=1|x) = 0.5$
-> plug in 0.5 for p and solve for x_2.



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

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

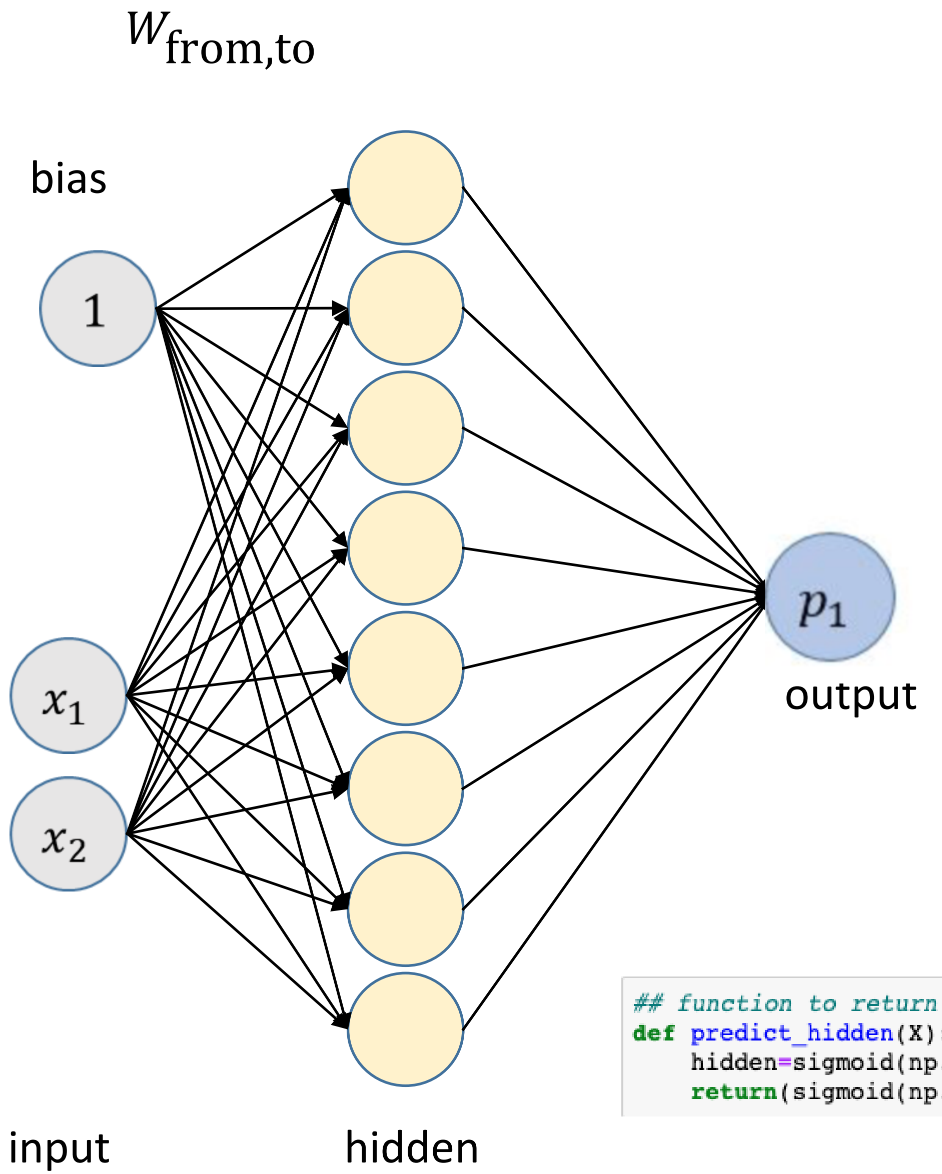
$$x_2 = -2 - 4 \cdot x_1$$



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


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

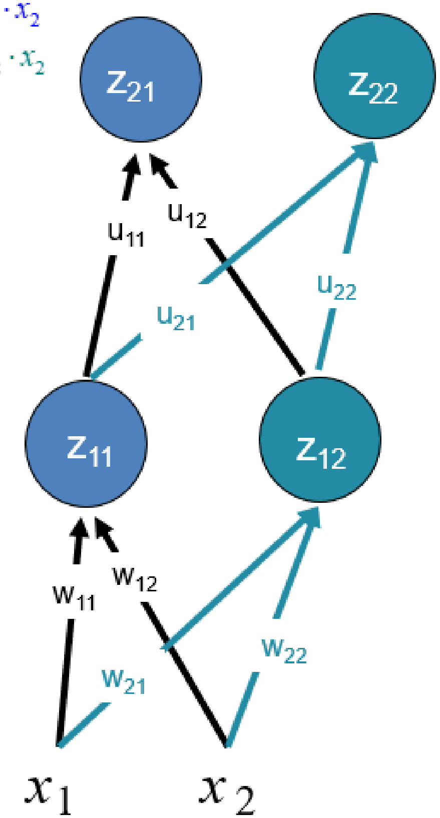
2 linear layers can be replaced by 1 linear layer -> can't go deep with linear layers!

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

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

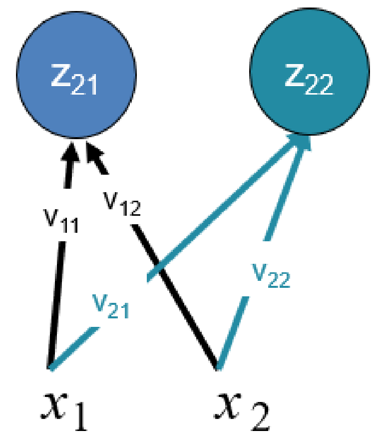
$z_{11} = w_{11} \cdot x_1 + w_{12} \cdot x_2$

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



=

$z_{21} = v_{11} \cdot x_1 + v_{12} \cdot x_2$



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

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

Recap: Matrix Multiplication aka dot-product of matrices

We can only multiply matrices if their dimensions are compatible.

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

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

$$\mathbf{A}_{3x3} \times \mathbf{B}_{3x2} = \mathbf{C}_{3x2}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} x \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ \boxed{c_{31}} & c_{32} \end{bmatrix}$$

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

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

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

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

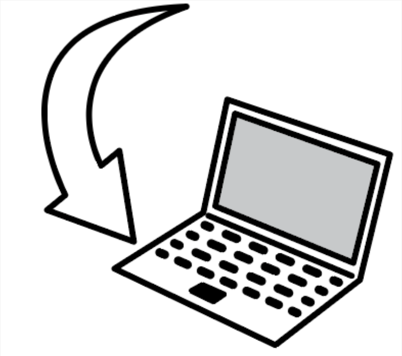
$$\boxed{c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}}$$

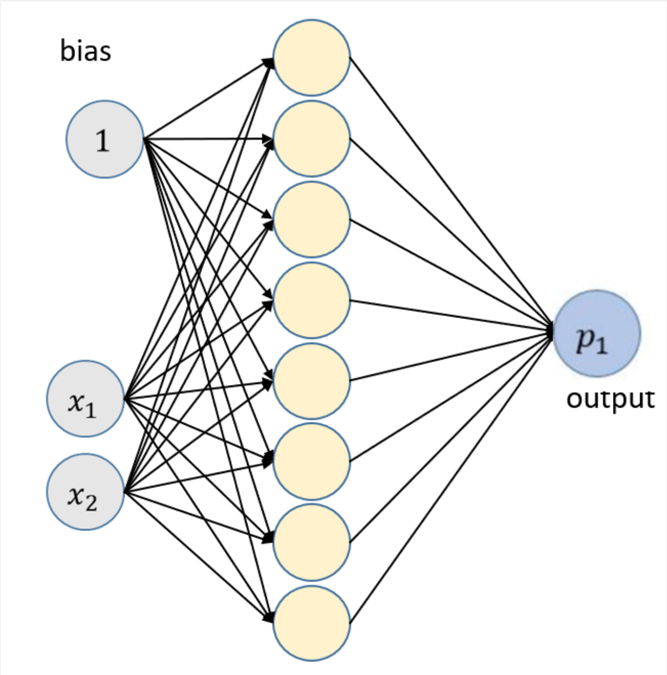
$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}$$

Example:

$$\mathbf{A}_{2x2} = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \qquad \mathbf{B}_{2x3} = \begin{pmatrix} 3 & 1 & 7 \\ 8 & 2 & 4 \end{pmatrix} \qquad \mathbf{C}_{2x3} = \mathbf{A}_{2x2} \cdot \mathbf{B}_{2x3} = \begin{pmatrix} 11 & 4 & 18 \\ 24 & 6 & 12 \end{pmatrix}$$

Exercise:

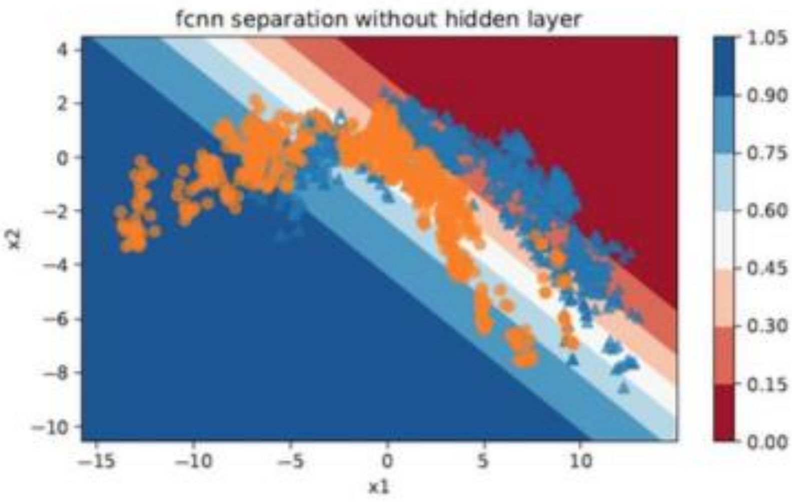


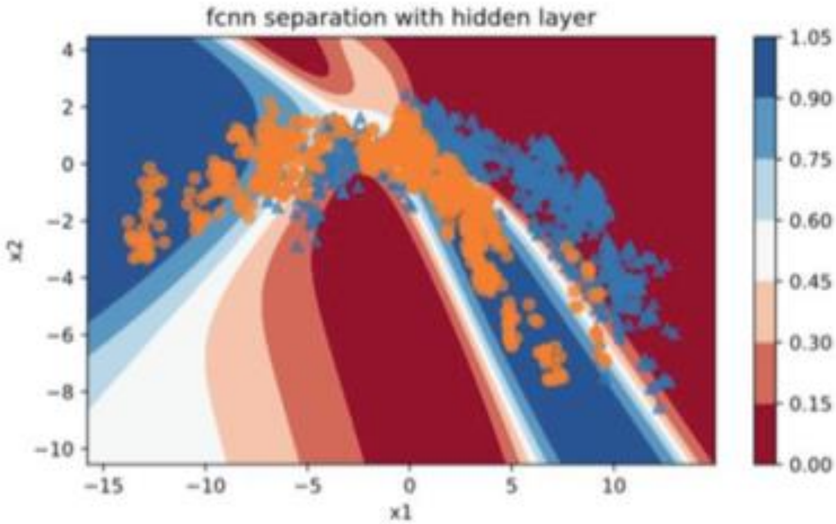


input hidden

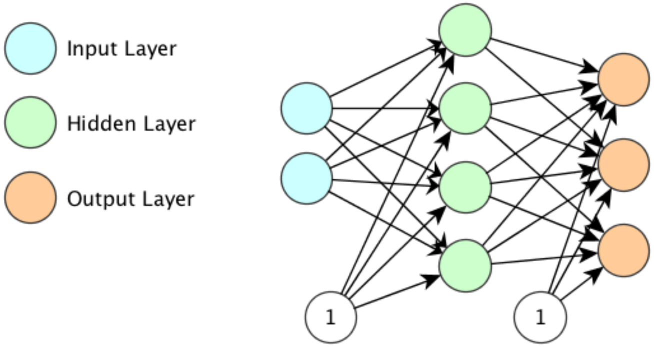
Open NB [01_simple_forward_pass.ipynb] and do exercise <u>stop</u> before Keras

Observations from NB: The benefit of hidden layers



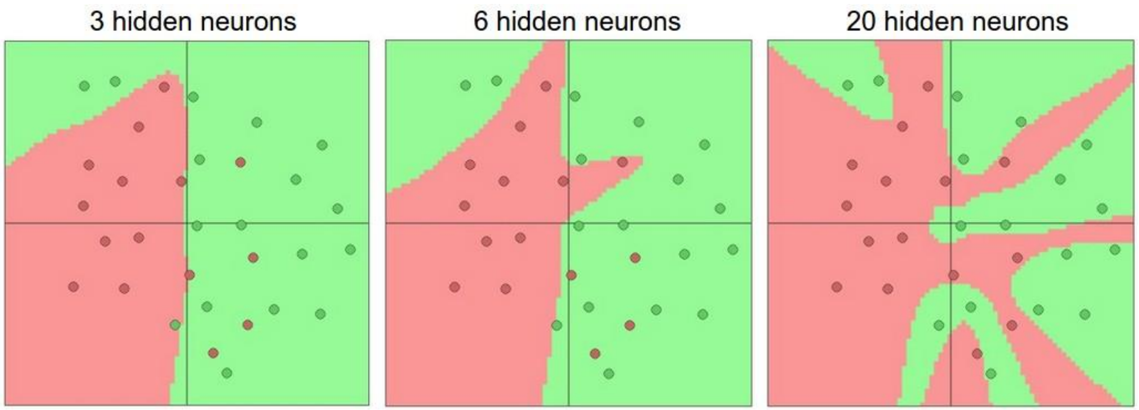


One hidden Layer is “in theory” enough



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

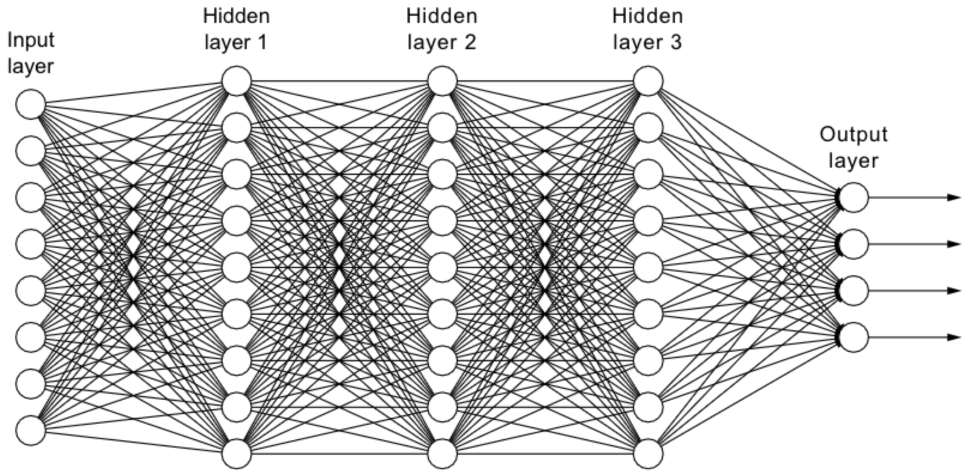
→ Each decision curve can be fitted with a NN with large enough hidden layer



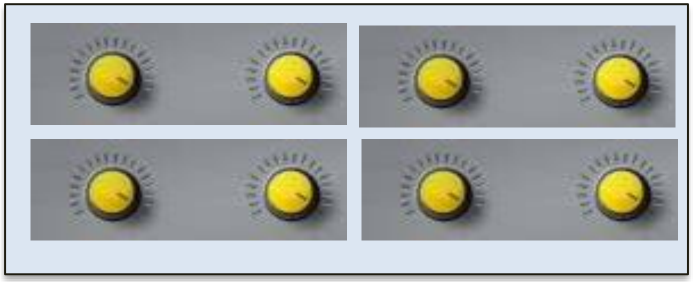
http://cs231n.github.io/neural-networks-1/

Training NN:
Minimize the loss function

How to determine the weights



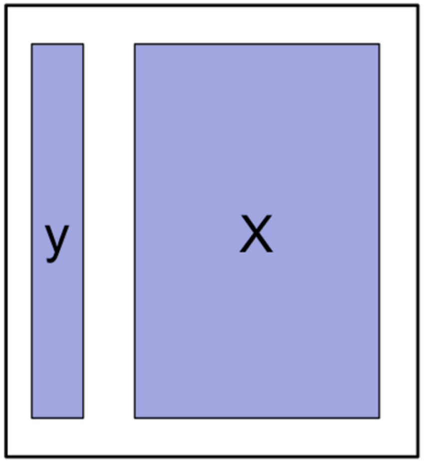
Given the input, the output of a NN is defined by the weights



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



Supervised Learning

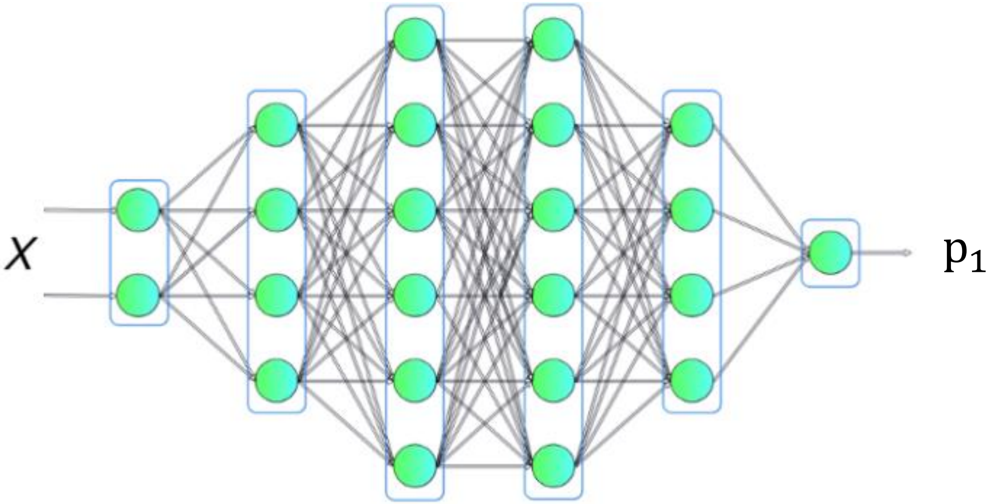
- The loss function depends on the task

- 2 Main tasks in DL predict y given x

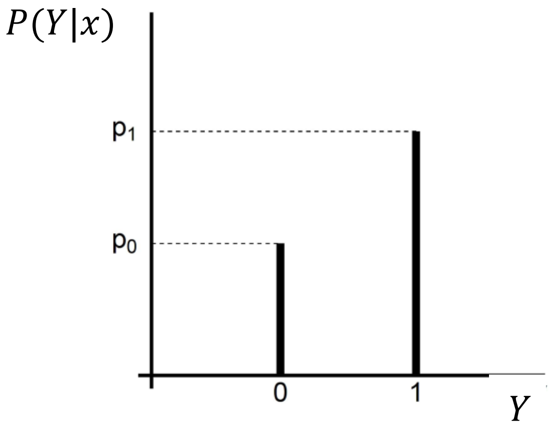
  - **Classification**
    - Point prediction: Predict a class label
    - Probabilistic prediction: predict a discrete probability distribution over the class labels
    - **First we focus on probabilistic binary classification where $Y \in \{0, 1\}$**

  - 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 $Y = 1$



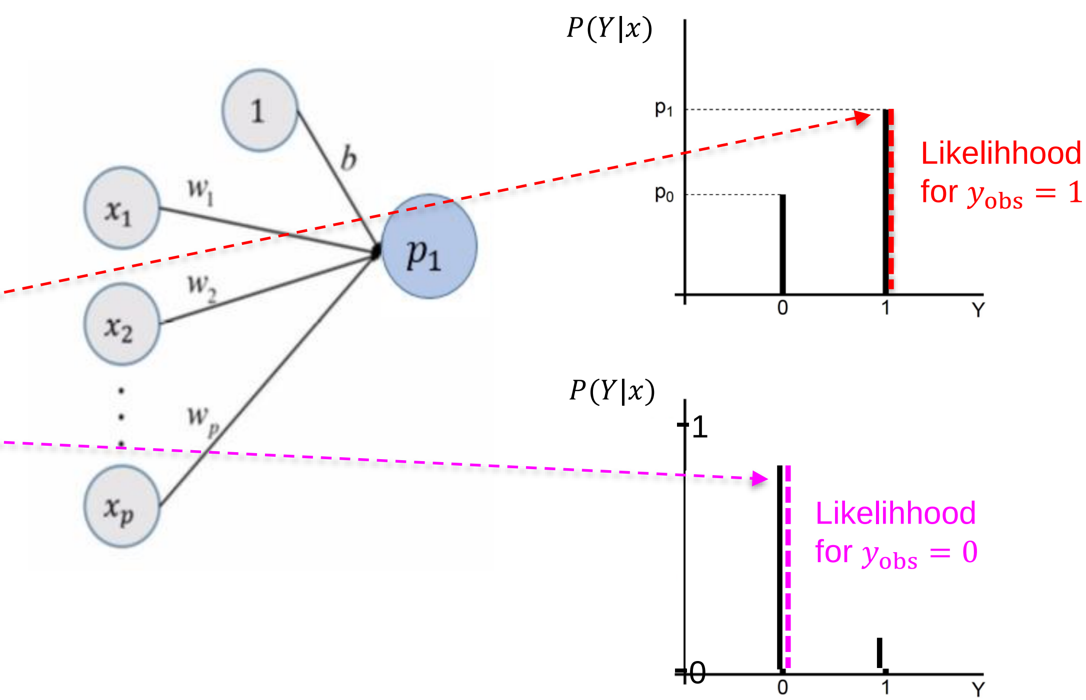
Given (=conditioned on) the input featurs x of an observation, the NN predicts as output the probability that this observation corresponds to class $Y = 1$ by $p_1 = \mathrm{P}(Y = 1|x)$, and hence the probability for the $Y = 0$ is $p_0 = \mathrm{P}(Y = 0|x) = 1 - p_1$.

Probabilistic binary classification aka logistic regression

probabilistic prediction

Training data set

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



Given (=conditioned on) the input featurs 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

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

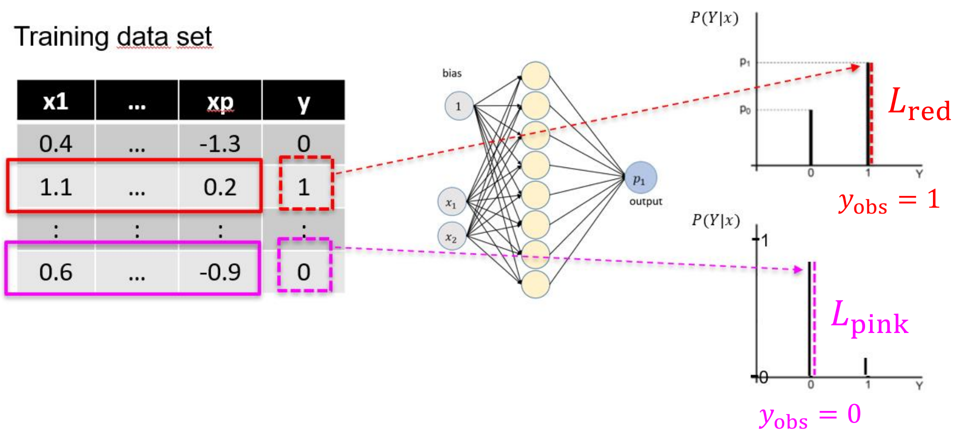
Notation trick in statistics – it selects correct log-probability since $y_i \in \{0,1\}$

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 $P(Y \mid x_i)$ 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 $P(Y|x)$



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

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 : $\mathbf{L} = \frac{1}{N}\sum \boldsymbol{log(p_i)}$

Predicted probability for the observed outcome y_{obs}

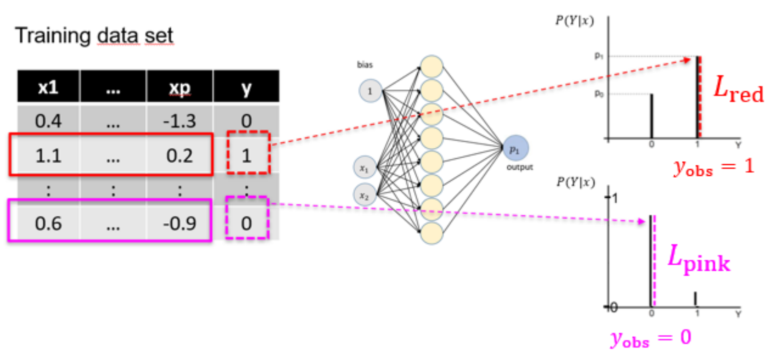
NLL Loss for probabilistic binary classification

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):

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



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

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

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

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

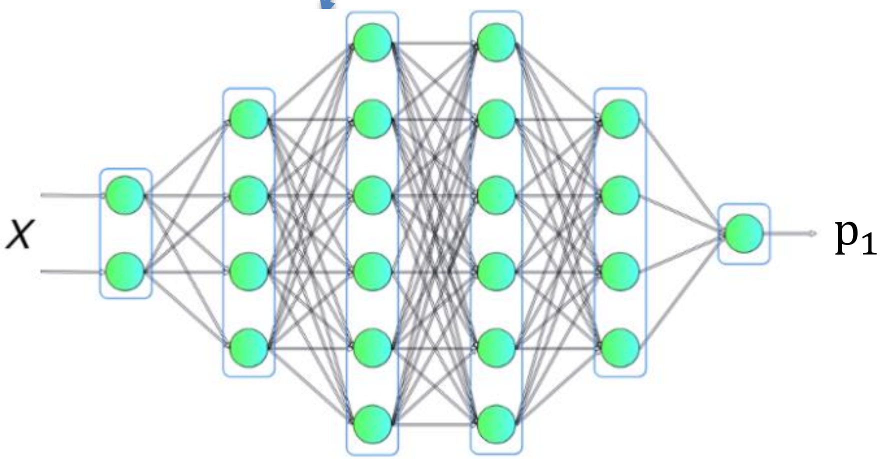
Note: In Keras we use the loss 'binary_crossentropy', if we do probabilistc binary classification with one output node with sigmoid activation.

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

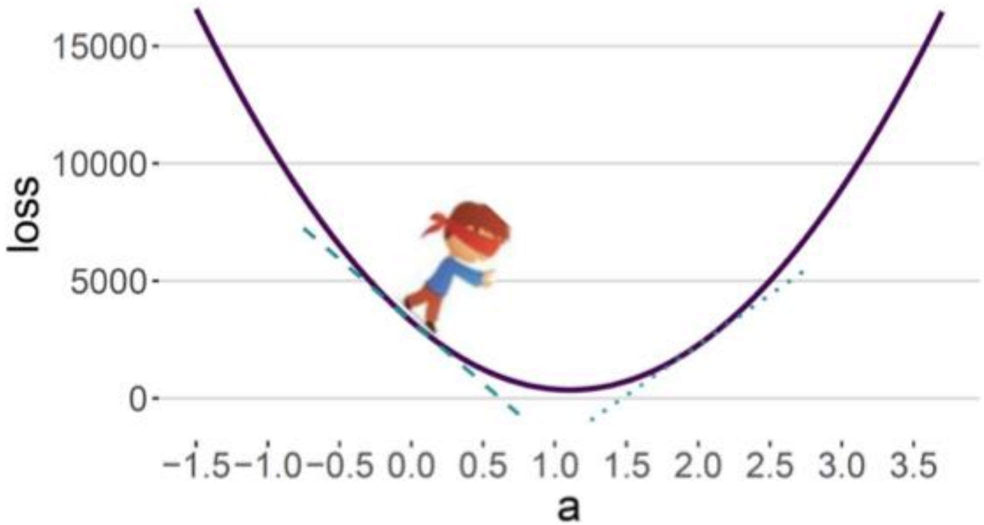
Parameters of the NN are the weights.



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

Idea of gradient descent

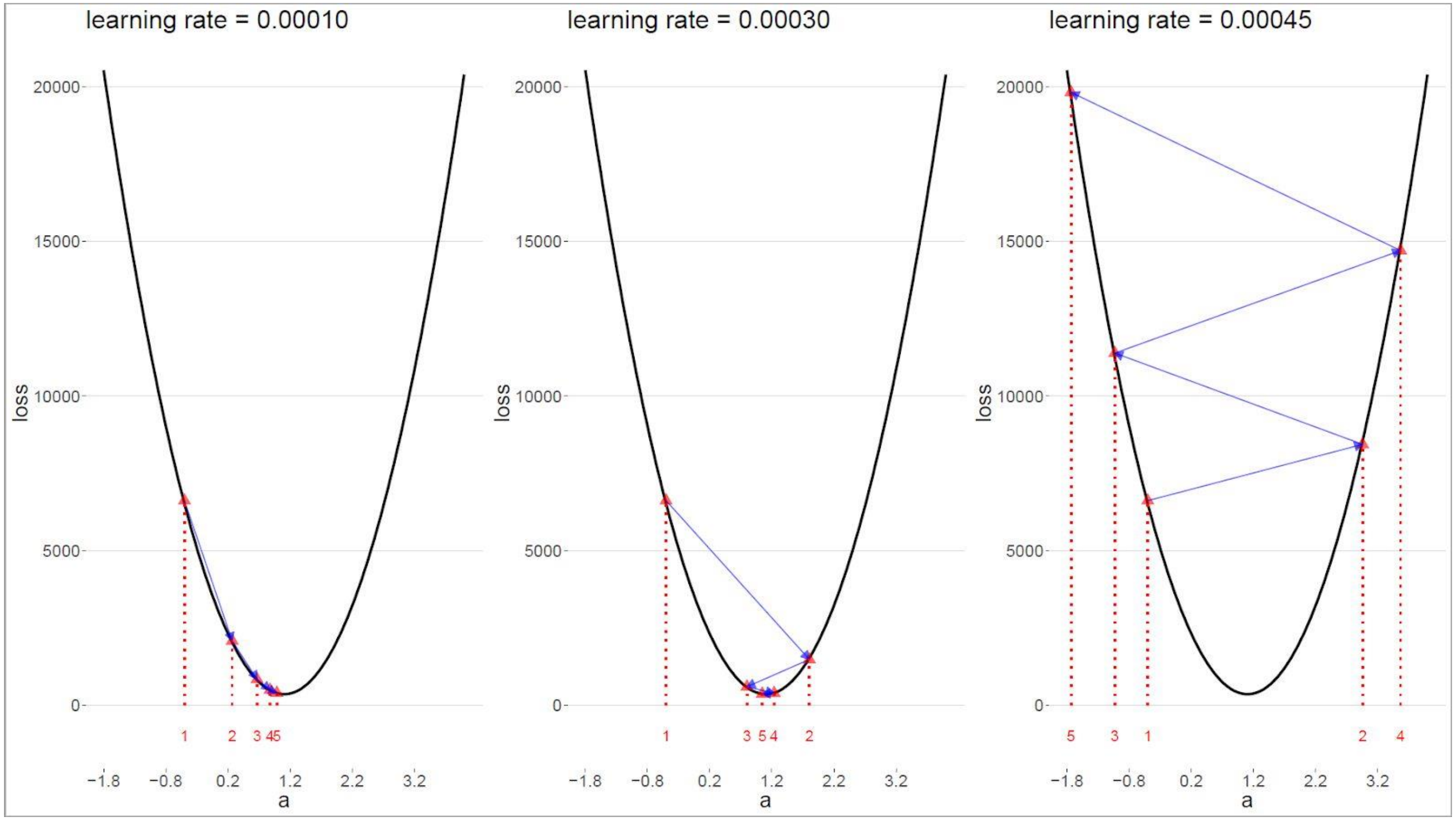
- Shown loss function for a single parameter a



- 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

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

The learning rate is a very important parameter for DL



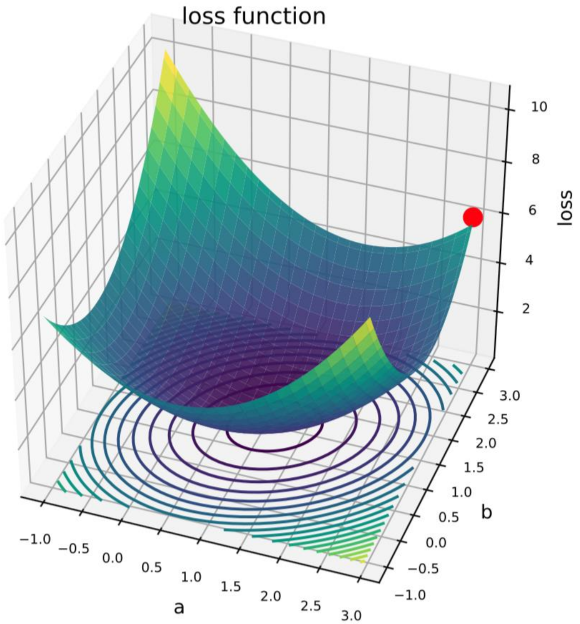
good – little slow

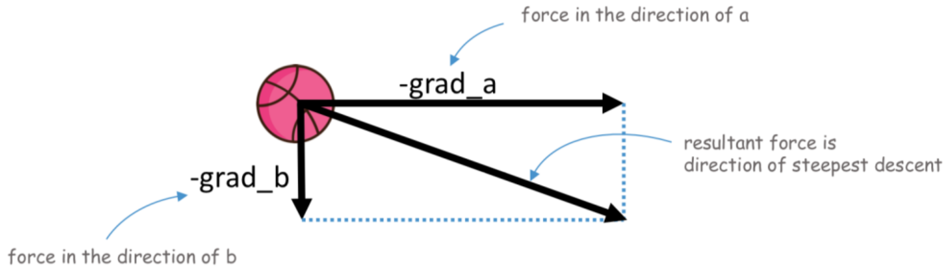
good

learning rate too high

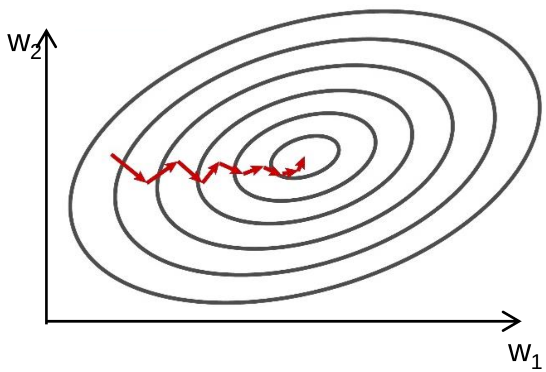
If the loss diverges to infinity: Don't panic, lower the learning rate!

In two dimensions





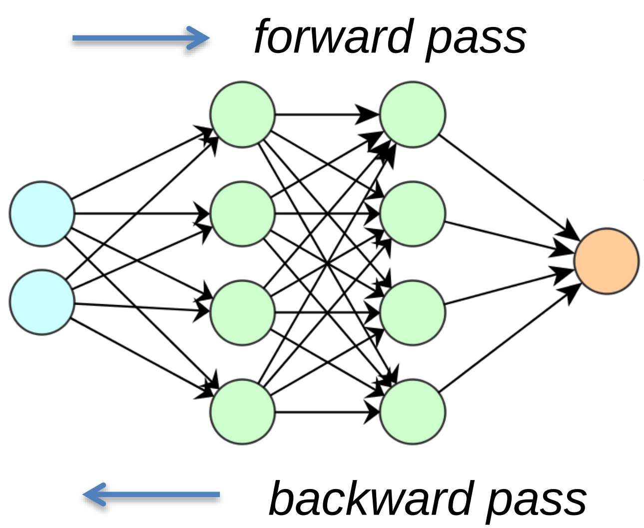
Gradient is perpendicular to contour lines

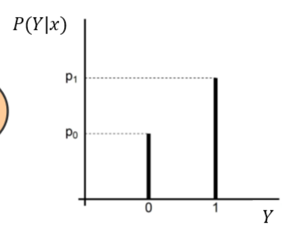


$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left.\frac{\partial L(\mathbf{w})}{\partial w_i}\right|_{w_i = w_i^{(t-1)}}$$

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





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

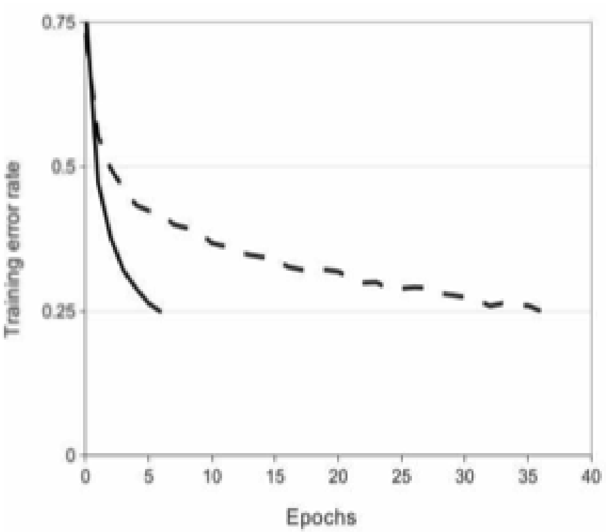
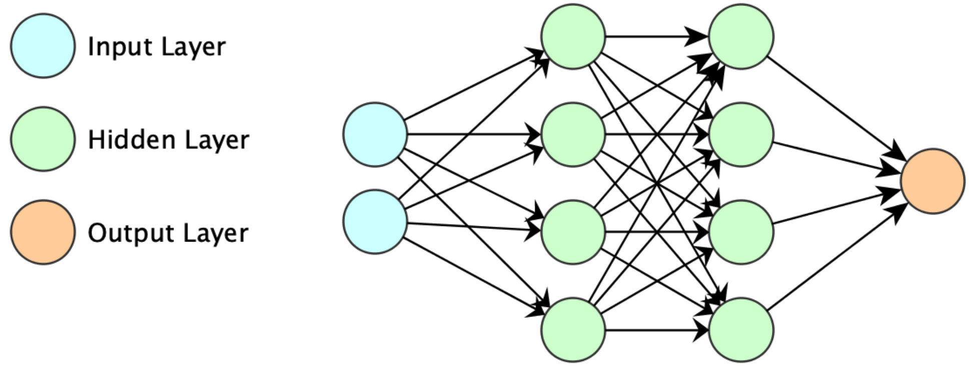
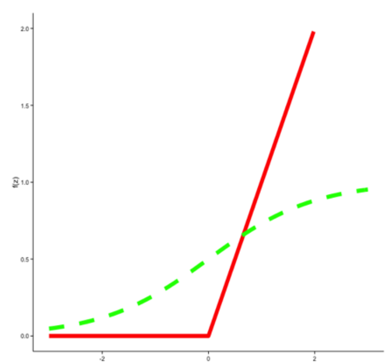


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

Source:
Alexnet
Krizhevsky et al 2012

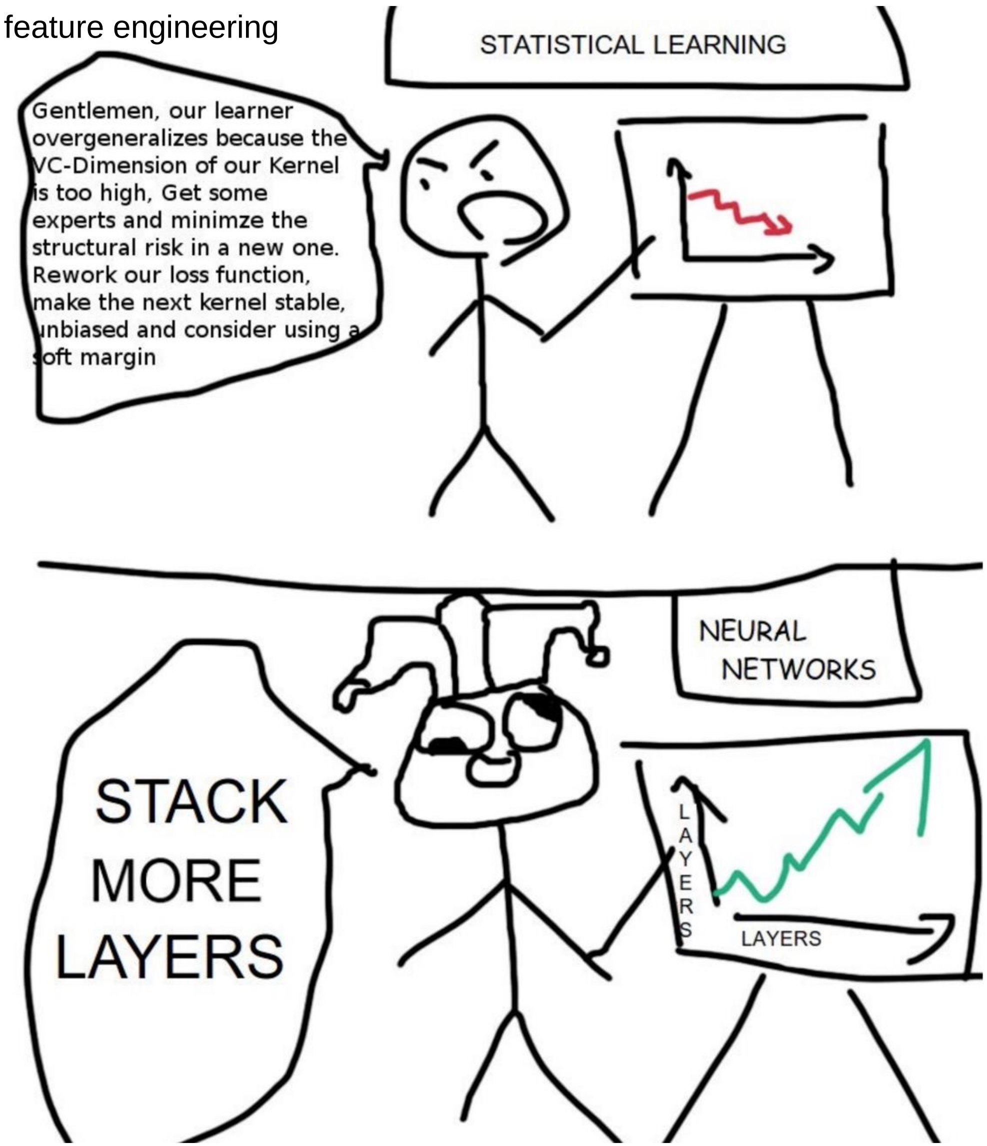




Motivation:
Green: sigmoid.
Red: ReLU faster convergence

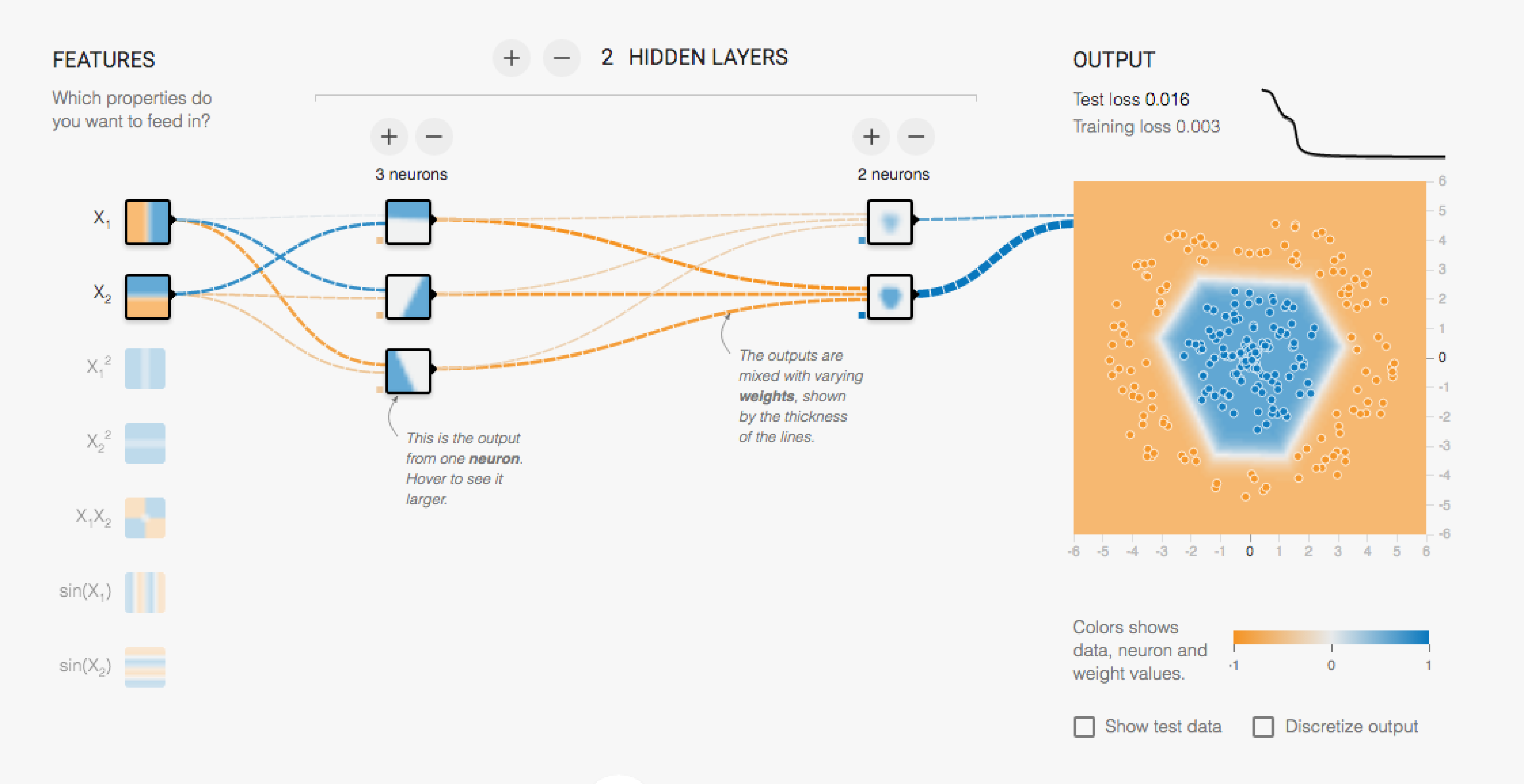
Epochs: ”each training examples is used once”

Game time – Recall learing DL vs Statistical Learning



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

Experiment yourself (homework)



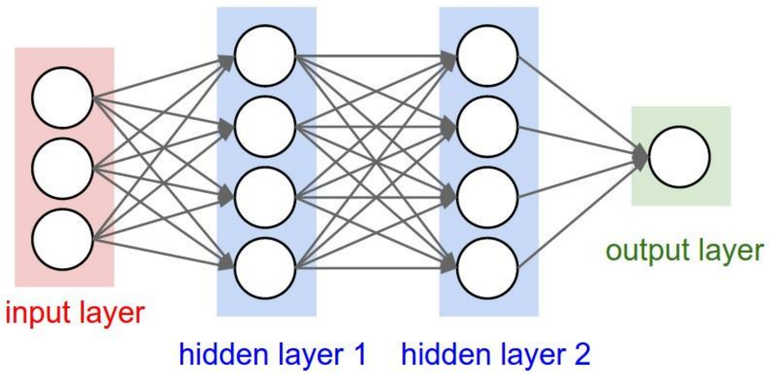
http://playground.tensorflow.org

Let's you explore the effect of hidden layers

Introduction to Keras

Keras as High-Level library to TensorFlow

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



High Level Libraries

- Keras
  - Keras is now part of TF core
  - https://keras.io/

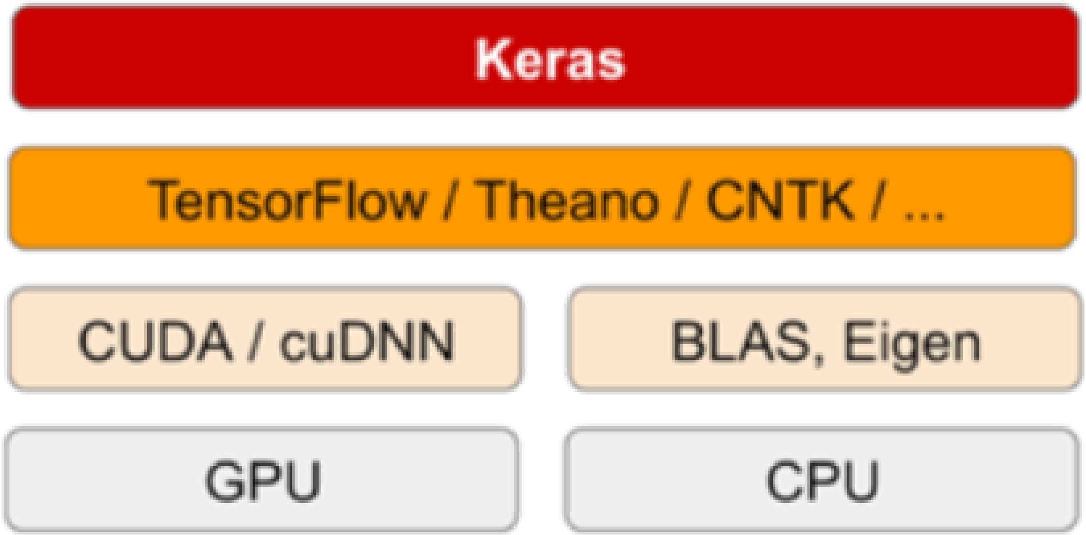
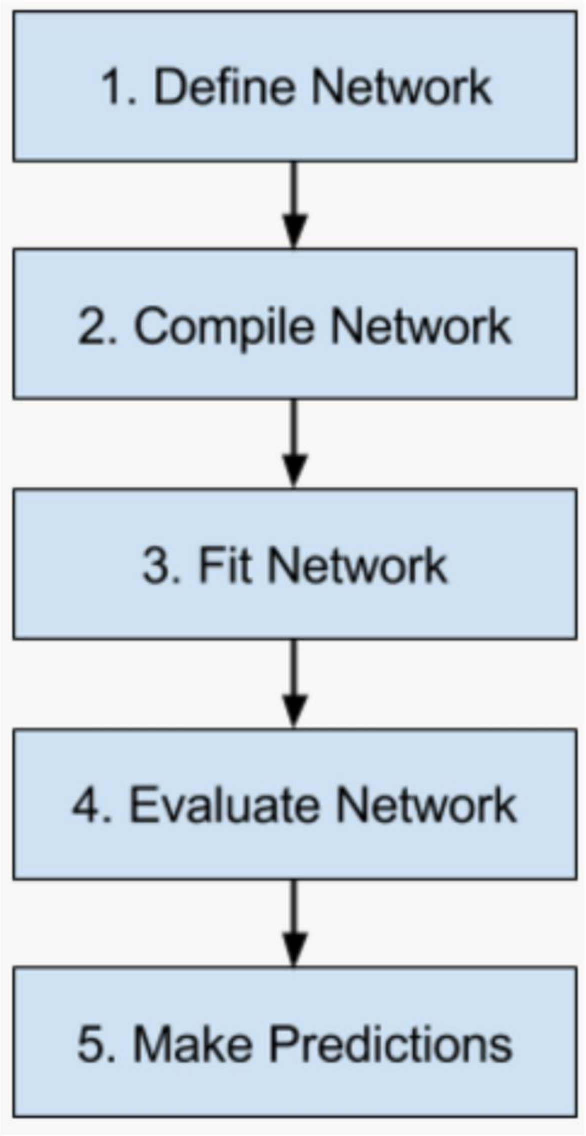


Figure: From Deep Learning with Python, Francois Chollett

Keras Workflow



Define the network (layerwise)

Add loss and optimization method

Fit network to training data

Evaluate network on test data

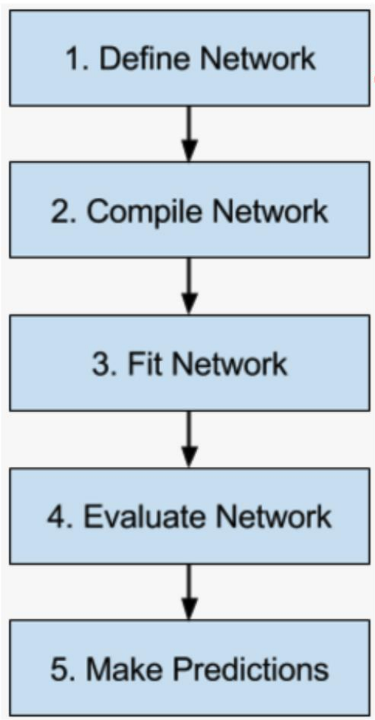
Use in production

A first run through

Define the network

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

Hidden layer with 8 Neurons, Activation function: sigmoid




```
# Define fcNN
model = Sequential()
model.add(Dense(8,
                batch_input_shape=(None, 2),
                activation='sigmoid'))
model.add(Dense(1,
                activation='sigmoid'))
```

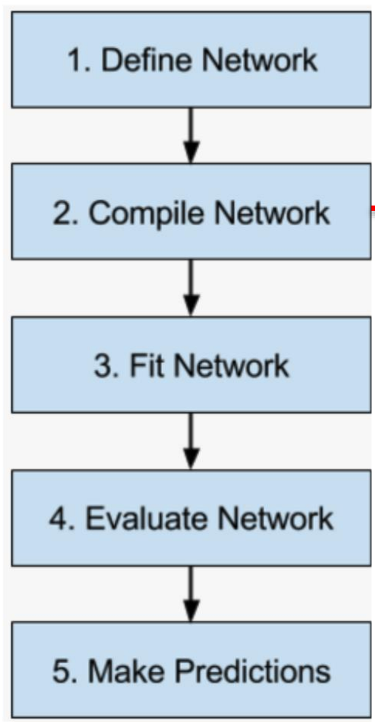
Last layer with one output neuron for binary classification

Input shape needs to be defined only at the beginning.

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

Compile the network

Which optimizer should be used?
Here Stochastic Gradient Descent

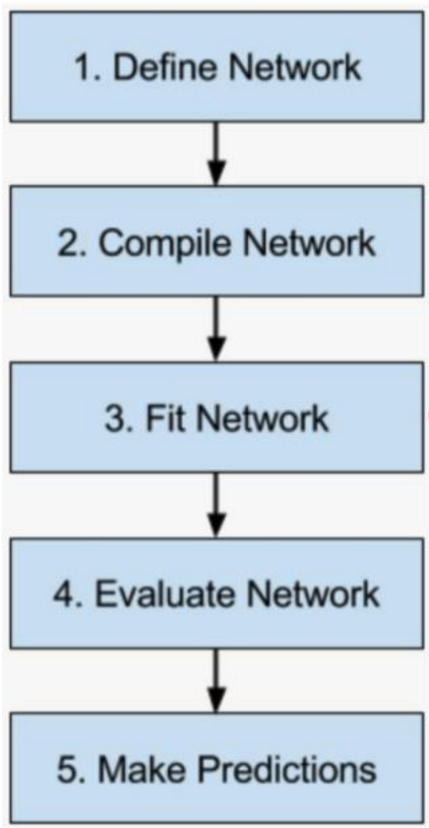



```
model.compile(optimizer=SGD(learning_rate=0.01),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Loss Function to optimize
Here: Binary CE

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

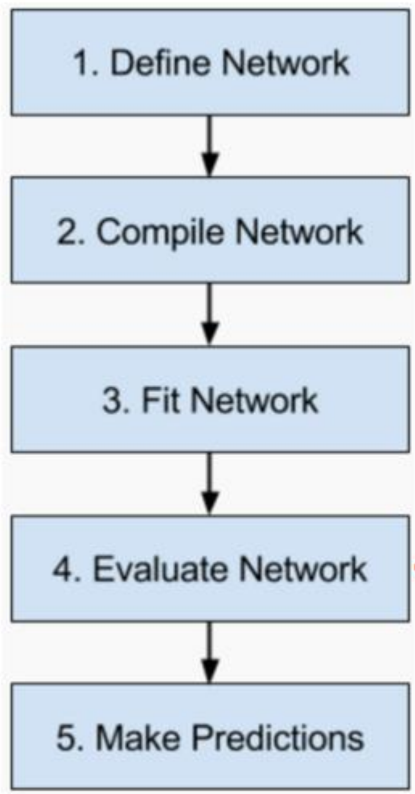
```
history = model.fit(X_train,
                    y_train,
                    validation_data=(X_val, y_val),
                    epochs=1000,
                    batch_size=10,
                    verbose=1)
```

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

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



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

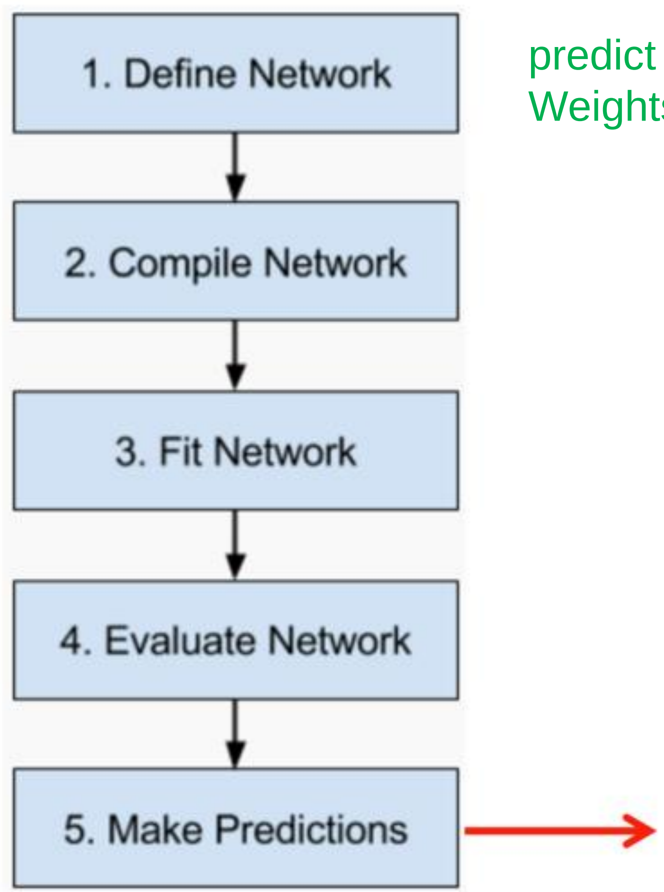
```
model.evaluate(X_test, y_test, verbose=0)
```

```
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

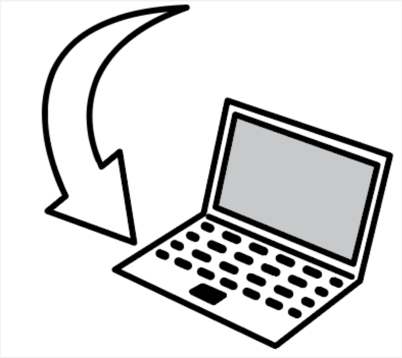
All features, here: 2

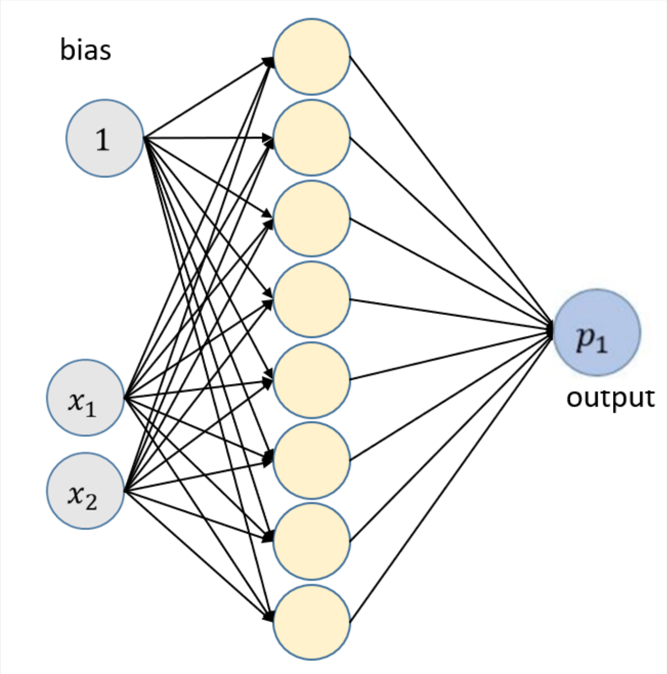
```
model.predict(X_test[0:10,:])
```

```
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

Exercise:





input hidden

Open NB 01_simple_forward_pass.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!