

# WBL Deep Learning:: Lecture 1

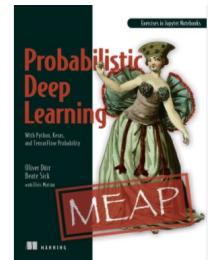
*Beate Sick, Oliver Dürr*

Lecture 1: Introduction and technicalities

Zürich, 9/12/2022

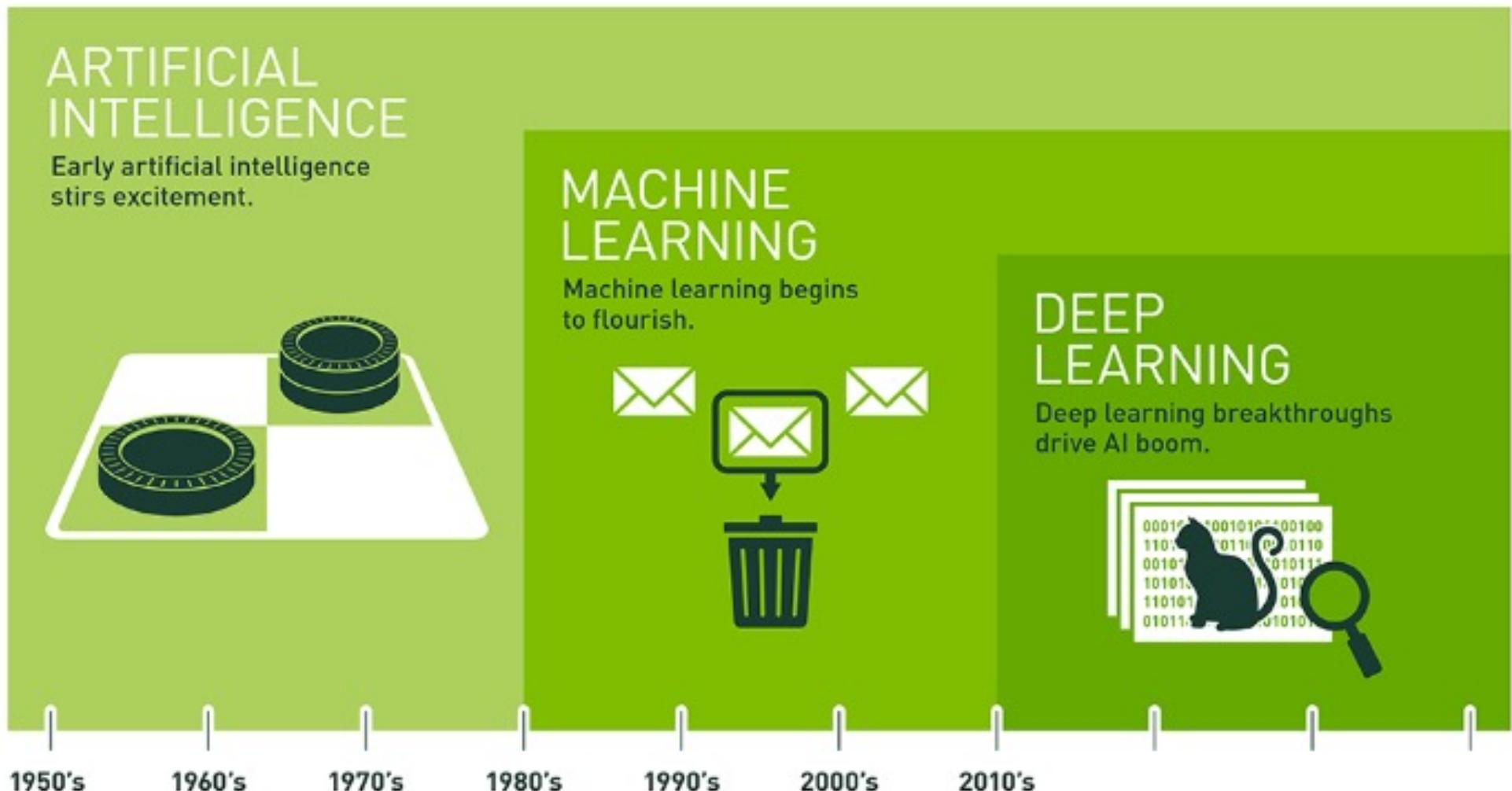
# Literature

- Additional Course website
  - [https://tensorchiefs.github.io/dl\\_rcourse\\_2022/](https://tensorchiefs.github.io/dl_rcourse_2022/)
- Probabilistic Deep Learning (Manning in production)
  - Our probabilistic take
  - [https://www.manning.com/books/probabilistic-deep-learning?a\\_aid=probabilistic\\_deep\\_learning&a\\_bid=78e55885](https://www.manning.com/books/probabilistic-deep-learning?a_aid=probabilistic_deep_learning&a_bid=78e55885)
- Deep Learning Book (DL-Book) <http://www.deeplearningbook.org/>. This is a quite comprehensive book which goes far beyond the scope of this course.
- Courses
  - Convolutional Neural Networks for Visual Recognition <http://cs231n.stanford.edu>
  - Martin Görner (very practical)
    - <https://cloud.google.com/blog/products/gcp/learn-tensorflow-and-deep-learning-without-a-phd>



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

# AI, Machine Learning, Deep Learning



Slide credit: <https://www.datasciencecentral.com/profiles/blogs/artificial-intelligence-vs-machine-learning-vs-deep-learning>

# Machine Perception

- Computers have been quite bad in perceptual tasks which are easy for humans.
  - Images
  - Text
  - Sound
- A Kaggle contest 2012

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?

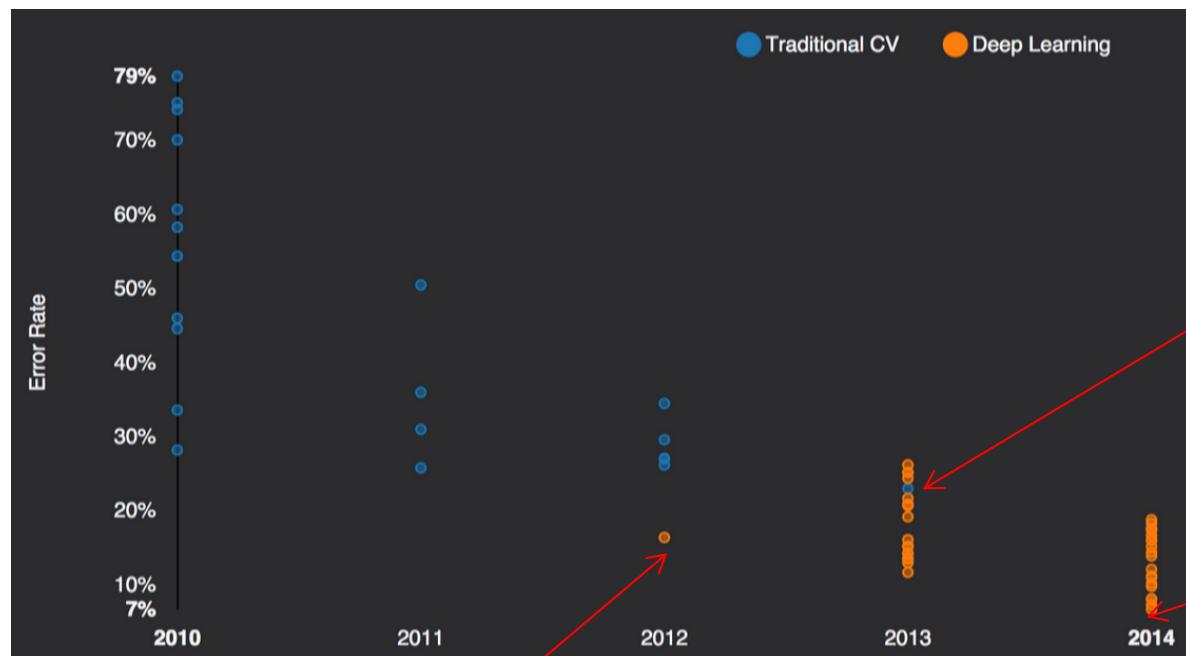
What happened, to solve the problem?

# Deep Learning Success Story: ImageNet 2012, 2013, 2014, 2015

1000 classes  
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN  
approach in 2013

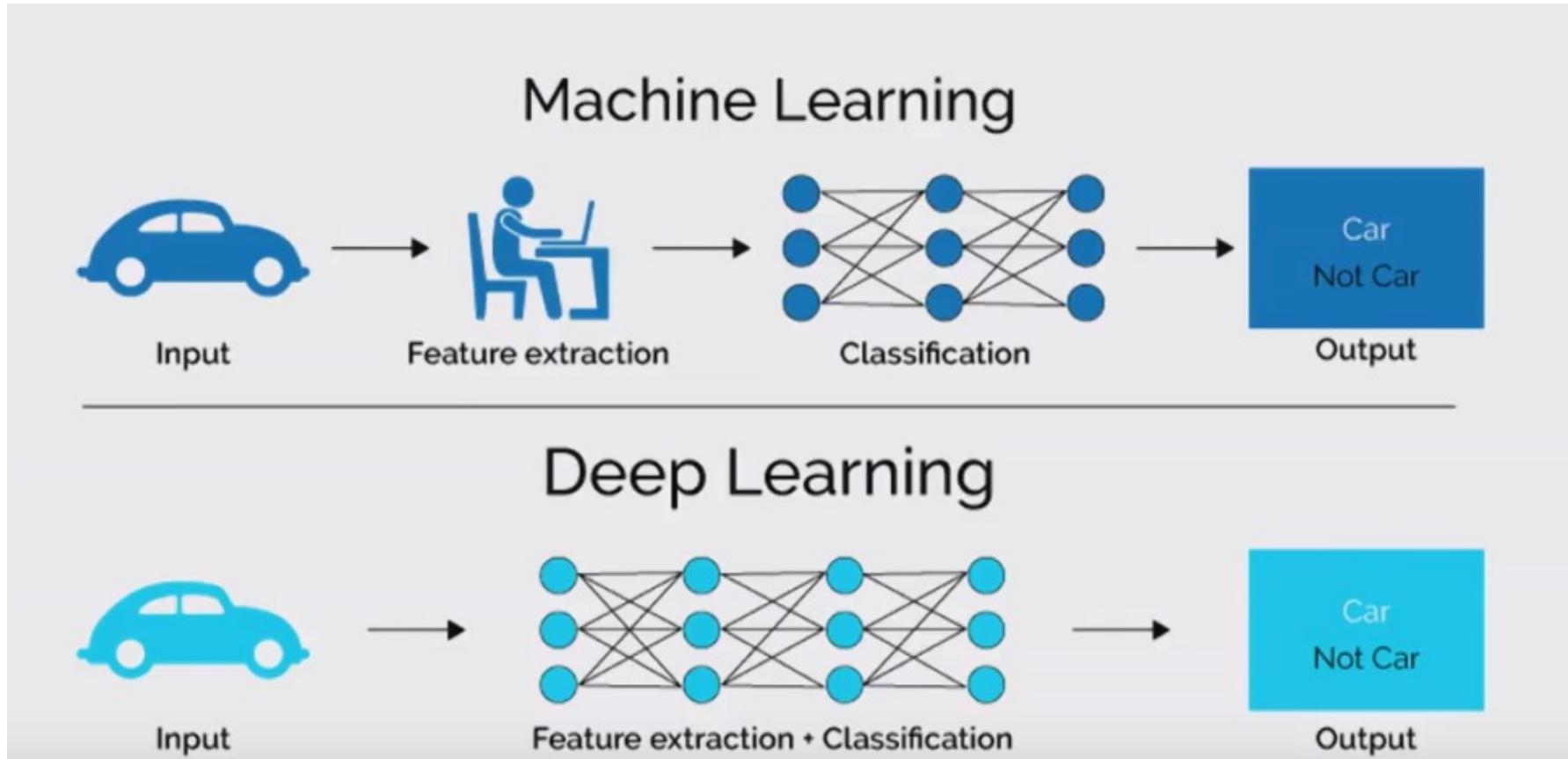
GoogLeNet 6.7%

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

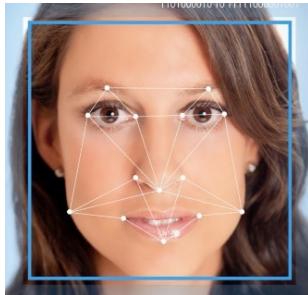
2015: It gets tougher

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)

# Deep Learning vs. Machine Learning



# What is new in the deep learning approach?

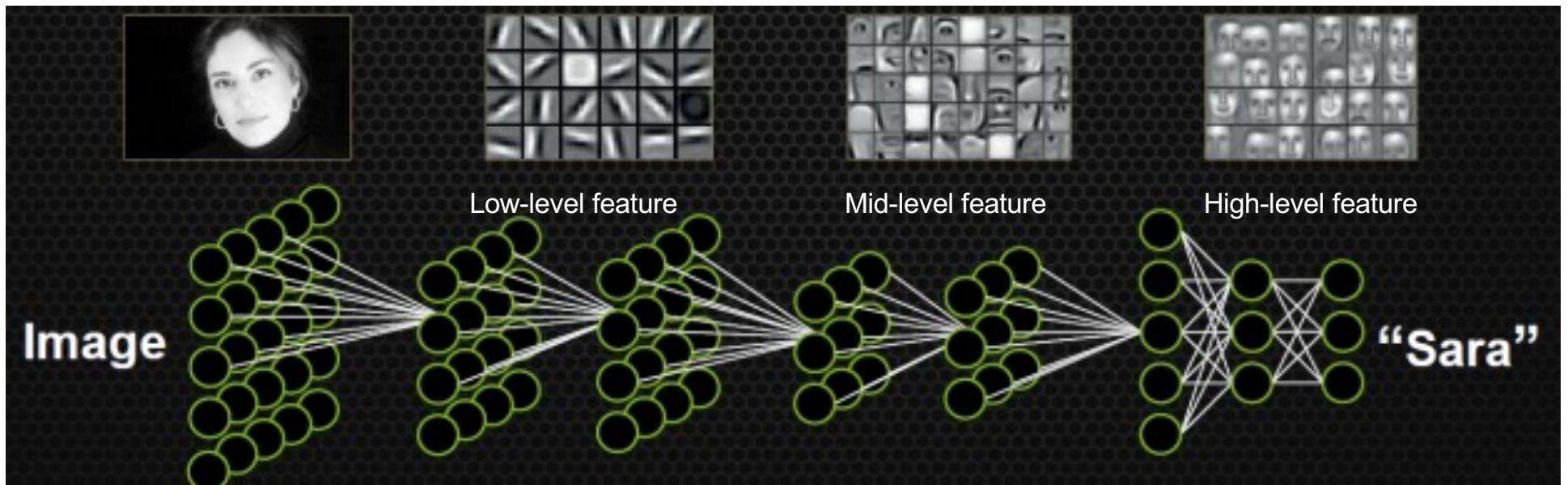


## Traditional ML:

Extract **handcrafted features** & use these features to **train / fit a model** (e.g. SVM, RF) and use fitted model to perform classification/prediction.

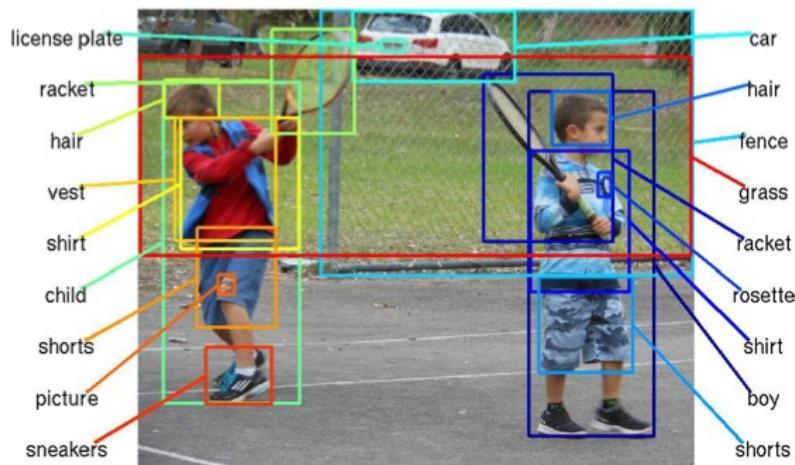
## Deep learning (end-to-end approach)

**Deep neural networks** start with raw data and **learn** during training/fitting to extract appropriate **hierarchical features** and to use them for classification/prediction.



# Deep Learning successes

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



"man in black shirt is playing guitar."



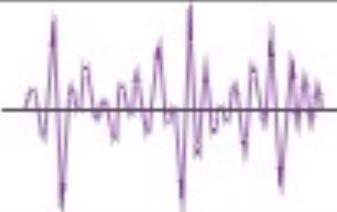
- Further examples



...

Images from cs229n

# Use cases of deep learning

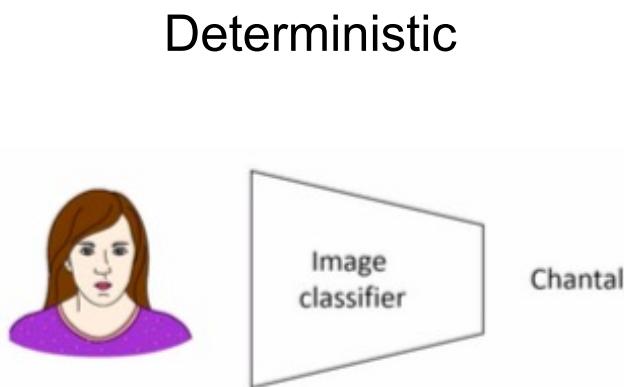
Input x to DL model	Output y of DL model	Application
Images 	Label "Tiger"	Image classification
Audio 	Sequence / Text "see you tomorrow"	Voice recognition
ASCII-Sequences "Hallo, wie gehts?"	Unicode-Sequences "你好，你好吗？"	Translation
ASCII-Sequence This movie was rather good	Label (Sentiment) positive	Sentiment analysis

Deep Learning öffnet Tür zu hören, sehen und Texten. (kein Verstehen aber statistische Zusammenhänge).

**Focus in this lectures:  
Probabilistic Viewpoint**

# Probabilistic vs deterministic models

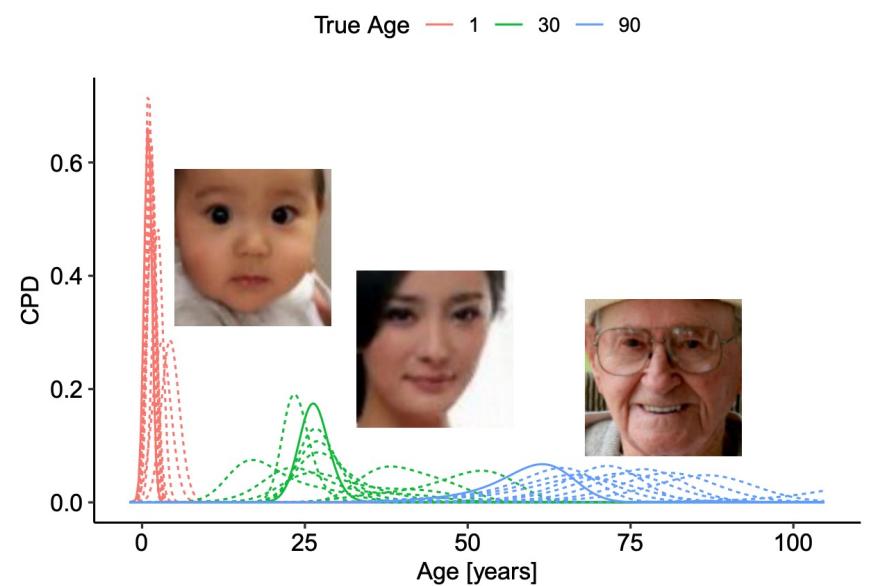
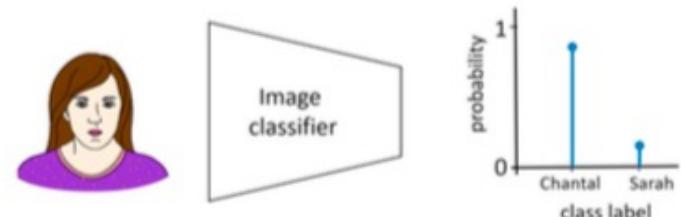
“Classification”



“Regression”



Probabilistic



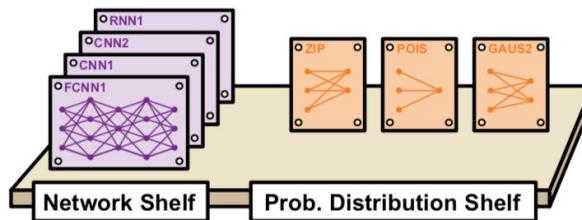
Conditional probability distribution (CPD)  
 $p(y|x)$

# Guiding Theme of the course

- We treat DL as *probabilistic models*, as a continuation of GLMs (logistic regression, ...) for CPD  $p(y|x)$
- The models are fitted to training data with maximum likelihood (or Bayes)

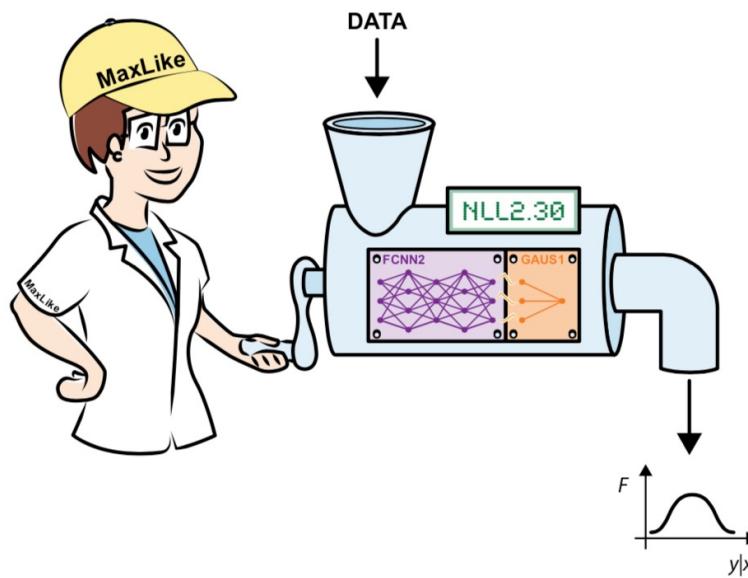
Special networks for x

- Vector FCNN
- Image CNN
- Text CNN/RNN



Special heads for y

- Classes
- Regression



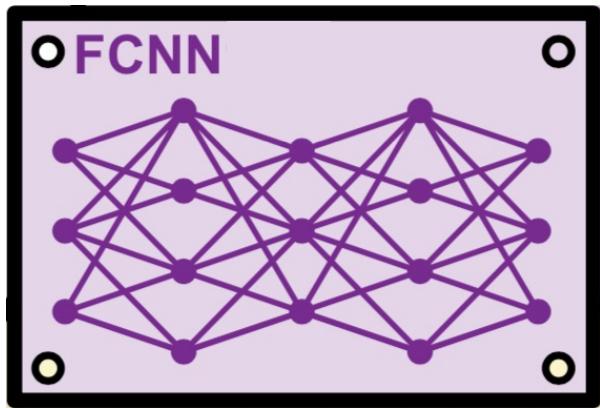
## Dates

Date	Lectures
12.09.2022	L1 (10:15-12:00), L2 (14:15-16:00) and Exercises (16:15-18:00)
19.09.2022	L3 (14:15-16:00) and Exercises (16:15-18:00)
26.09.2022	L4 (14:15-16:00) and Exercises (16:15-18:00)
03.10.2022	This day is reserved to work on your projects
10.10.2022	L5 and presentation of the projects

There will be also small exercises during the lectures

# Topics

- Lecture 1
  - Introduction to DL
  - Fully connected neural Networks (fcNN)
  - Introduction to TensorFlow and Keras
- Lecture 2
  - Convolutional Neural Networks (CNN) for image data
  - Classification and Regression with fcNN and CNNs
- Lecture 3
  - Probabilistic DL
  - Extending the GLM with DL for scalar features and image data
- Lecture 4,5 (Not 100% sure yet)
  - Extending deep GLMs by deep transformation models
  - Deep interpretable ordinal regression models



# Fully Connected Neural Networks

## FCNN

# The Single Cell: Biological Motivation

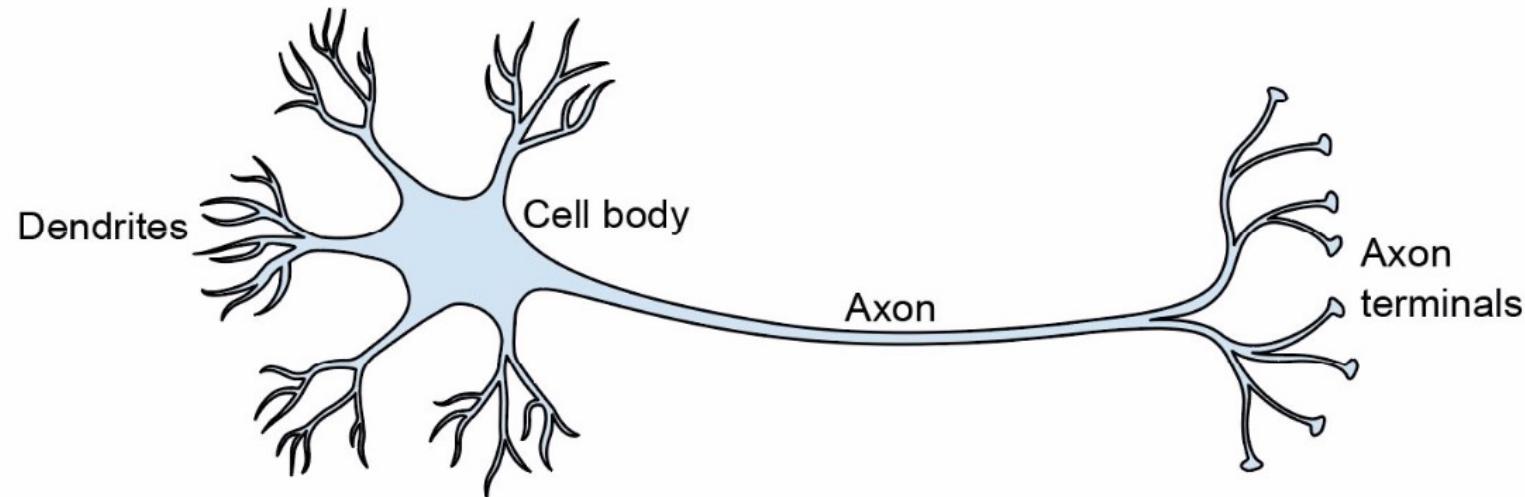
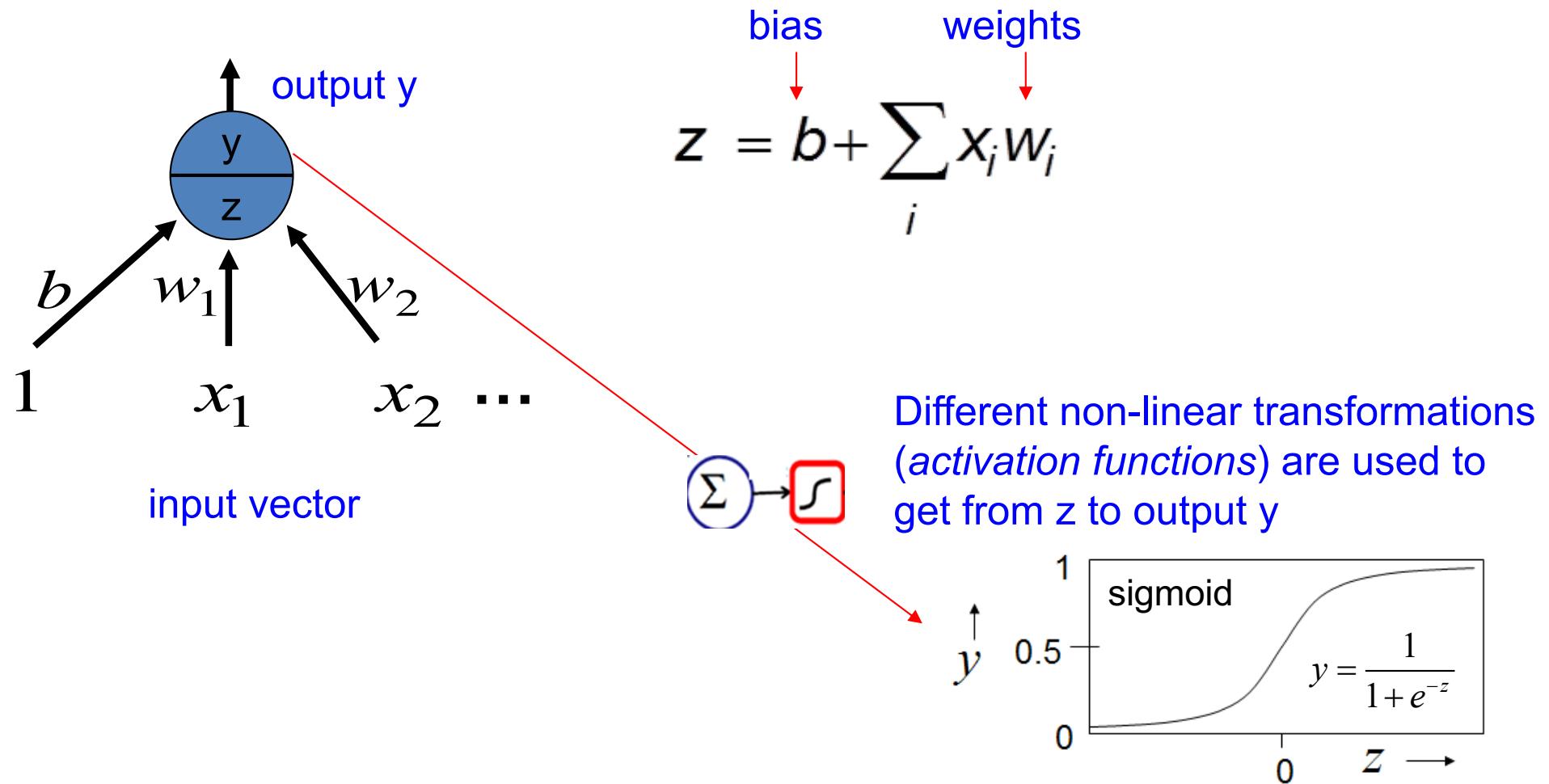


Figure 2.2 A single biological brain cell. The neuron receives the signal from other neurons via its dendrites shown on the left. If the cumulated signal exceeds a certain value, an impulse is sent via the axon to the axon terminals, which, in turn, couples to other neurons.

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

# An artificial neuron

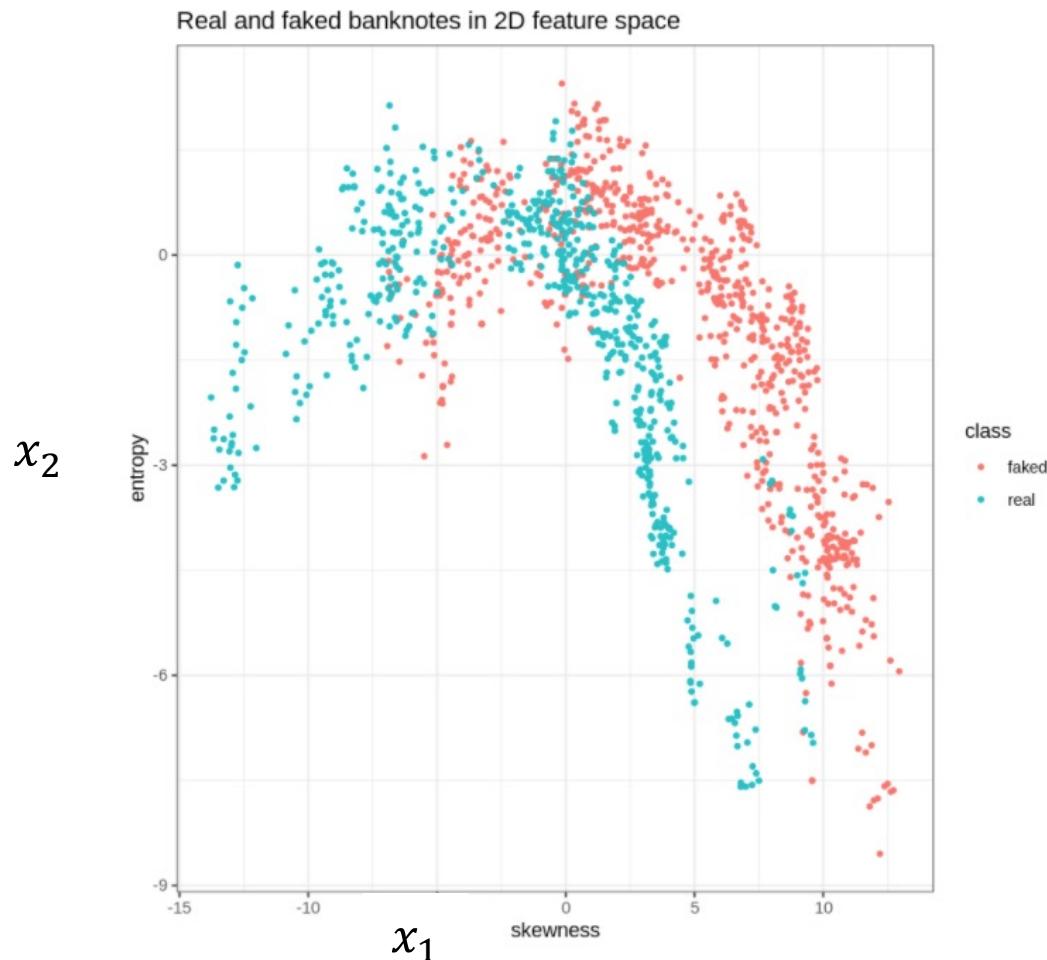


The  $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$  ensures number from 0 to 1, which can be interpreted as probability.

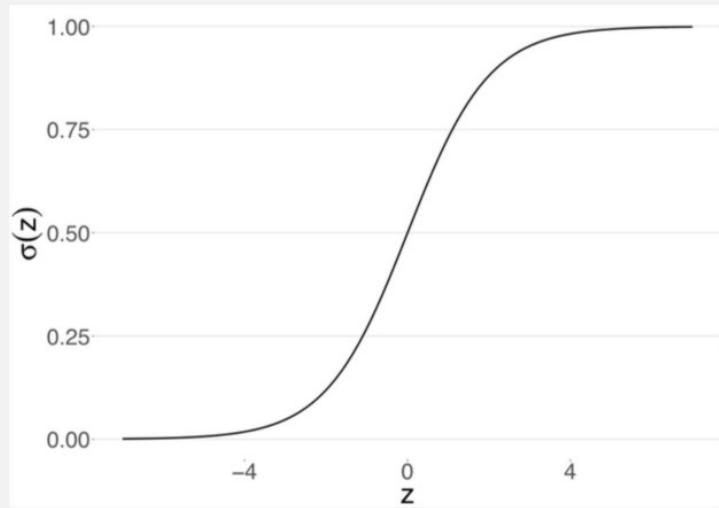
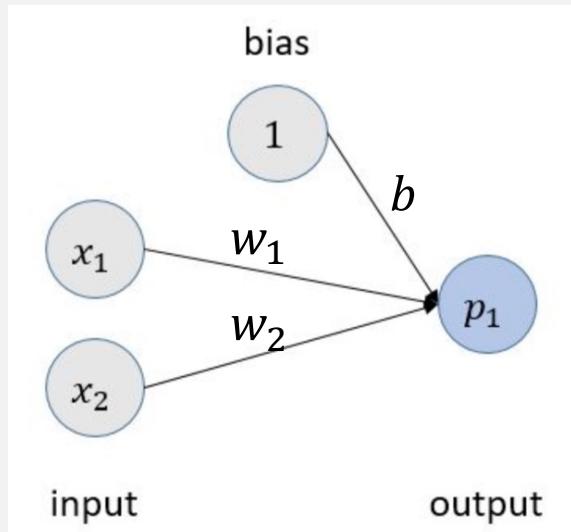
**Question: How is this model called in statistic?**

## Toy Task

- Task tell fake from real banknotes (see exercise later [01\\_nb\\_ch02\\_01](#))
- Banknotes described by two features ( $x_1, x_2$ ) extracted from image



# Exercise: Part 1



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

## TASK

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

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

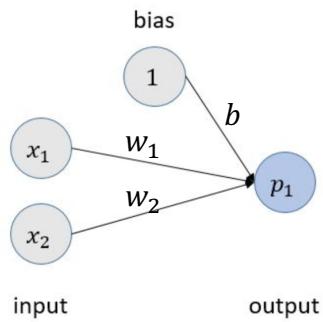
The probability can be calculated from  $z$  using the function  $\text{sigmoid}(z)$

1. What is the probability (rough estimate) that a banknote characterized by  $x_1=1$  and  $x_2 = 2.2$  is fake?

# GPUs love Vectors



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

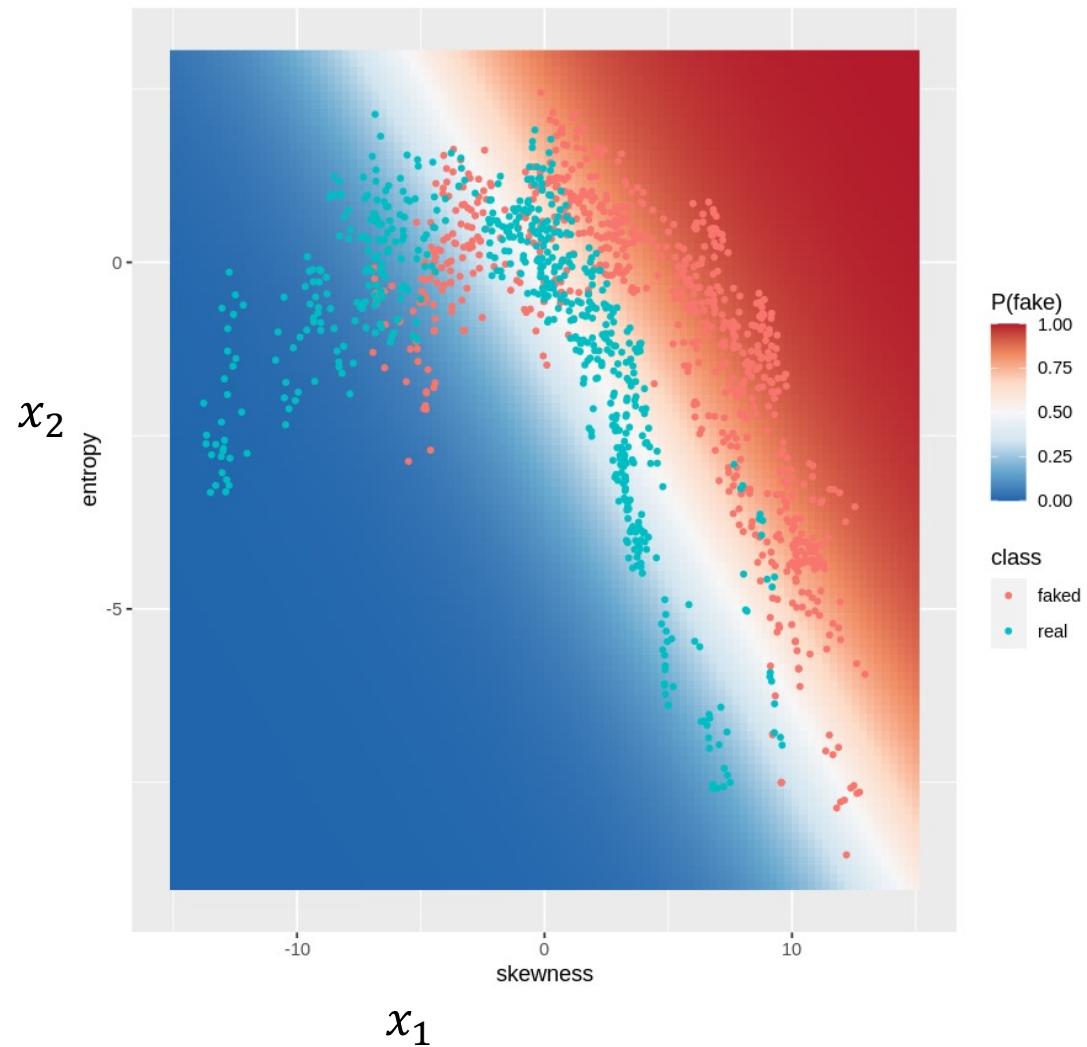


$$p_1 = \text{sigmoid} \left( (x_1 \quad x_2) \cdot \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} + b \right)$$

## DL: better to have column vectors

# Result\*

fcnn separation without hidden layer



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

\*Details of training later

A close-up shot from the movie Inception. Two men in dark suits are looking intensely at each other. The man on the left has blonde hair and is wearing a white shirt and a dark tie. The man on the right has dark hair and is also wearing a white shirt and a dark tie. They are both looking directly at each other with serious expressions.

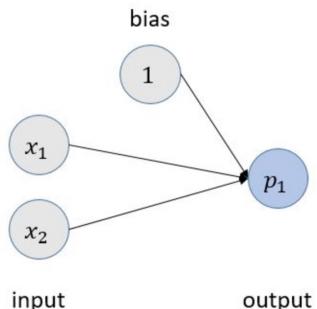
**WE NEED TO GO**

**DEEPER**

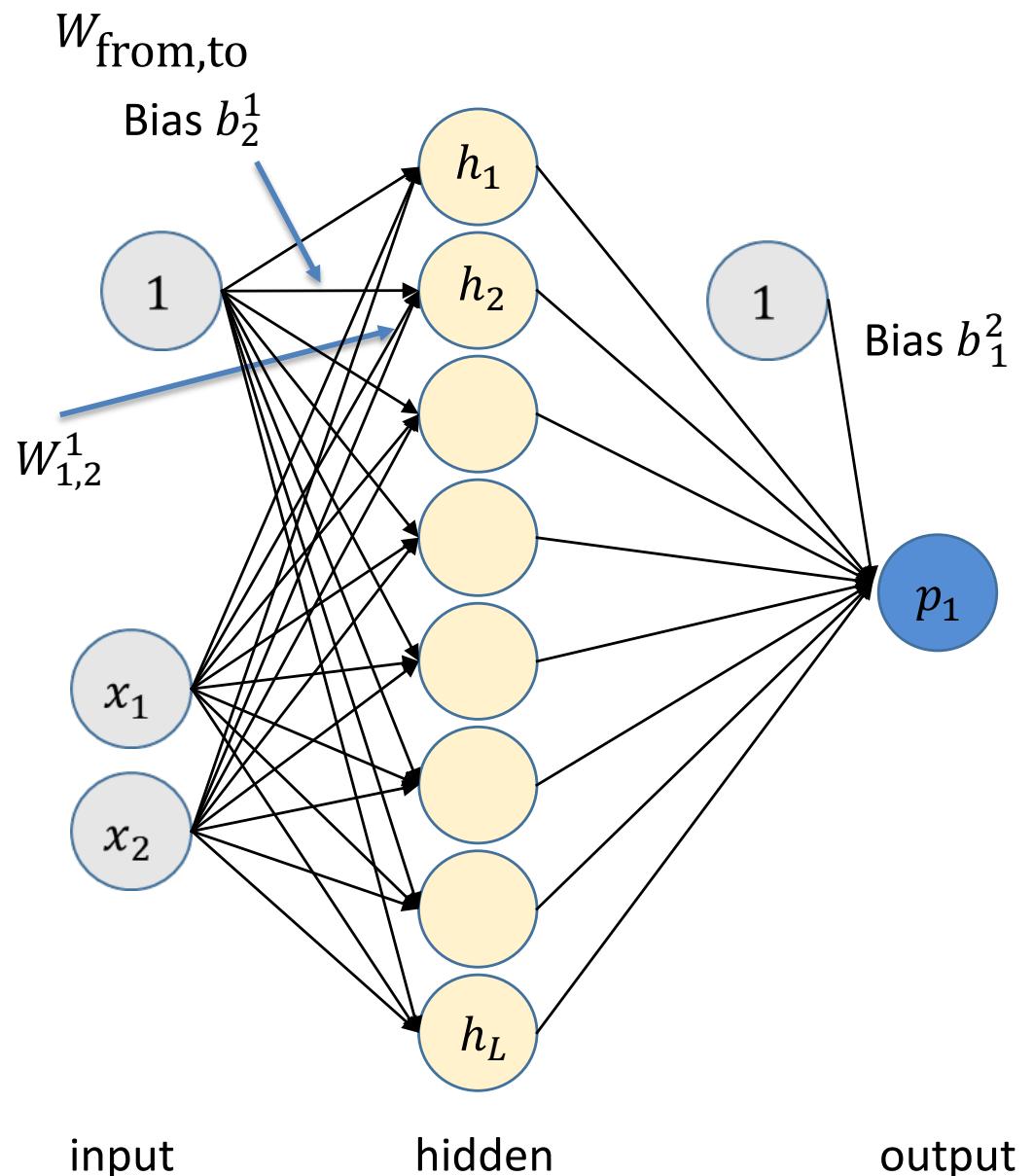
memegene

# Introducing hidden layer

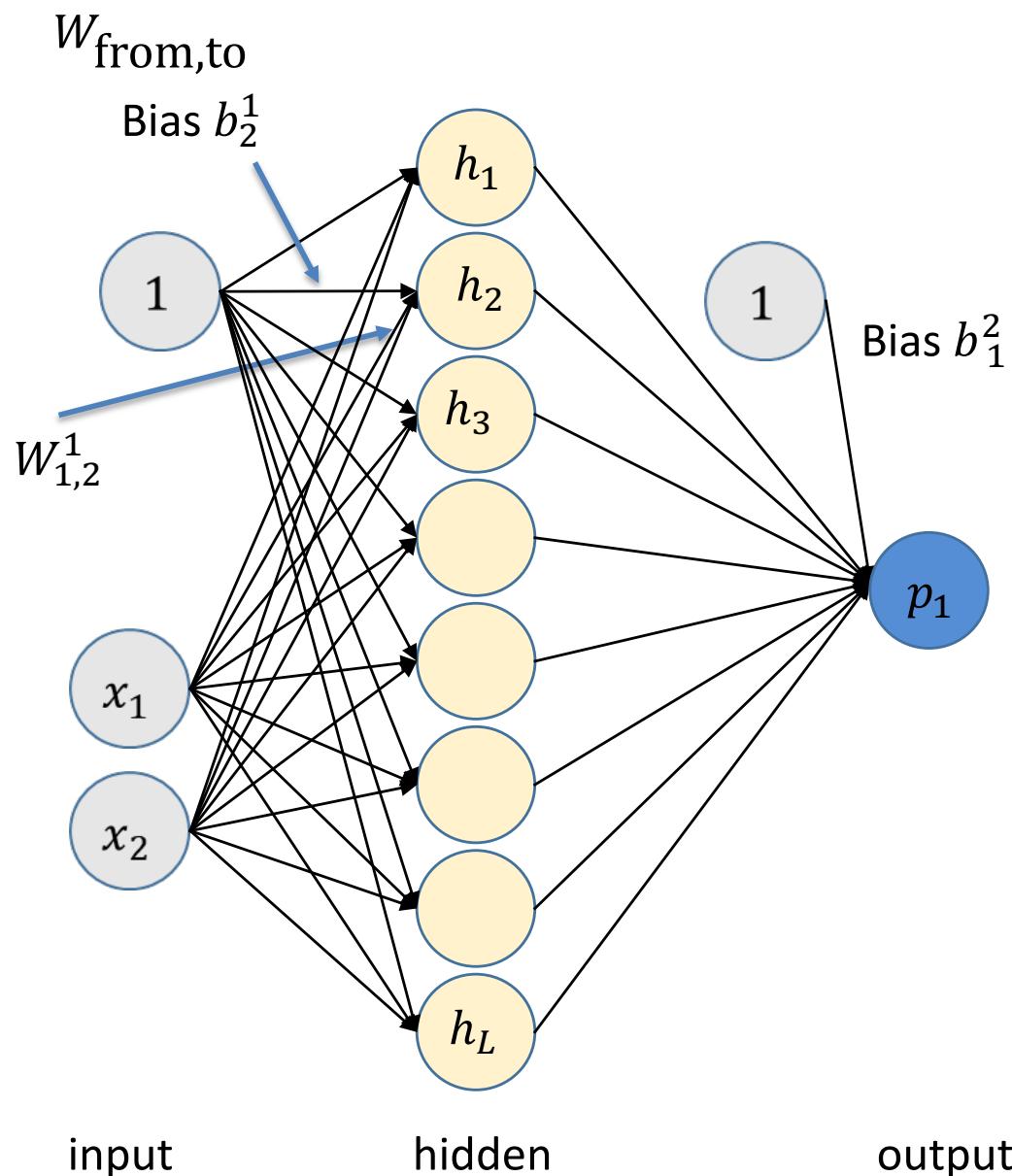
We stack single neurons in layers.  
We use the output of neuron as input to the new neuron.



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



# Introducing hidden layer



$$h_3 = \text{sigmoid}\left((x_1 \ x_2) \cdot \begin{pmatrix} W_{1,3}^1 \\ W_{2,3}^1 \end{pmatrix} + b_3^1\right)$$

General

$$h_i = \text{sigmoid}\left(\sum_l x_l W_{l,i}^1 + b_i\right)$$

$$h_i = \text{sigmoid}\left((x_1 \ x_2) \cdot \begin{pmatrix} W_{1,i}^1 \\ W_{2,i}^1 \end{pmatrix} + b_i^1\right)$$

Matrix Notation (later we drop  $\vec{\cdot}$ )  
Note column vectors!

$$\vec{h} = \text{sigmoid}(\vec{x} \cdot \vec{W}^1 + \vec{b}^1)$$

Complete Network

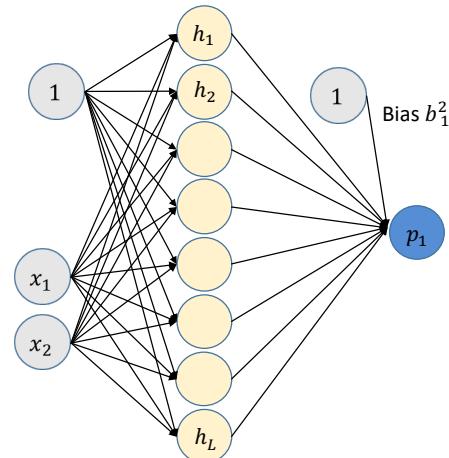
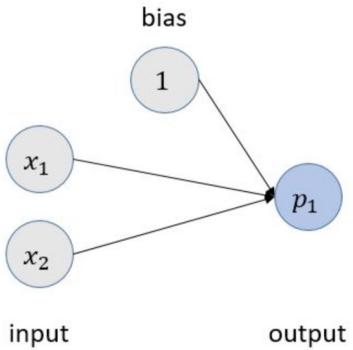
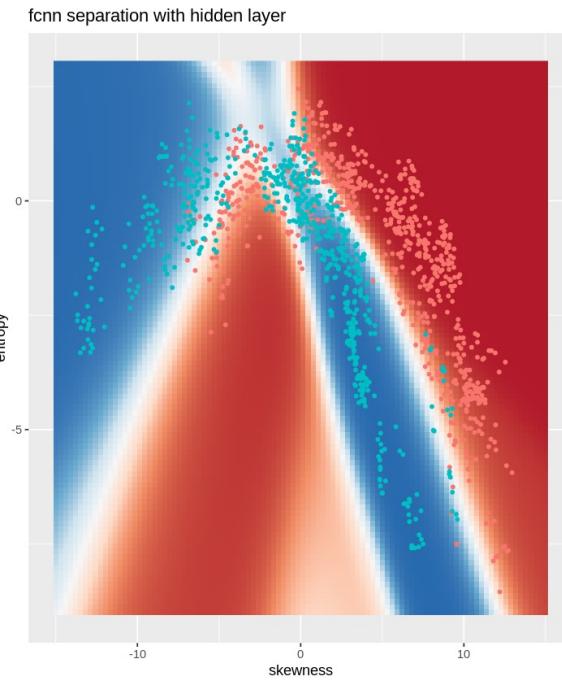
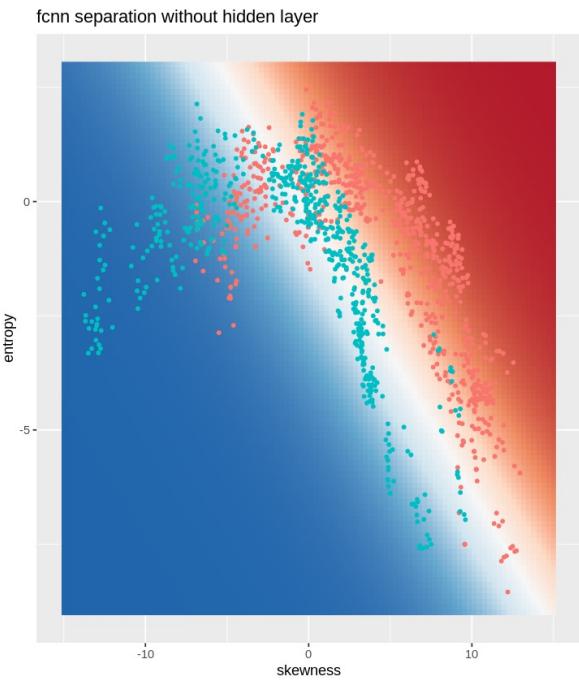
$$p_1 = \text{sigmoid}(\vec{h} \cdot \vec{W}^2 + b_1^2)$$

Code:

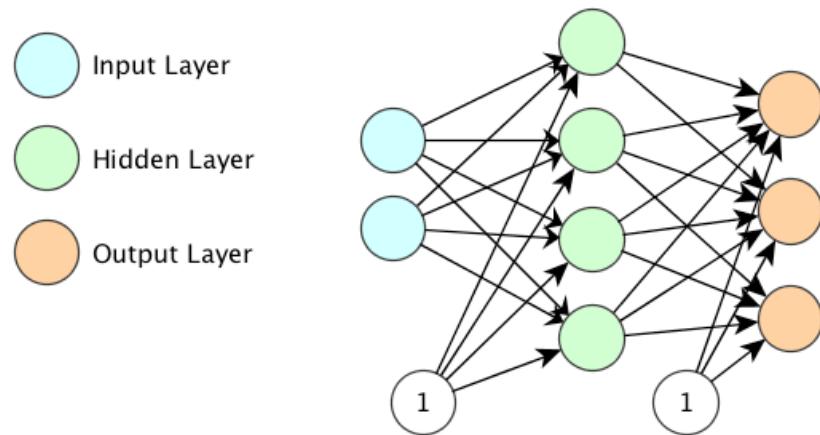
```
h = sigmoid(x %*% W1 + b1)
```

```
p1=sigmoid(h %*% W2 + b2)
```

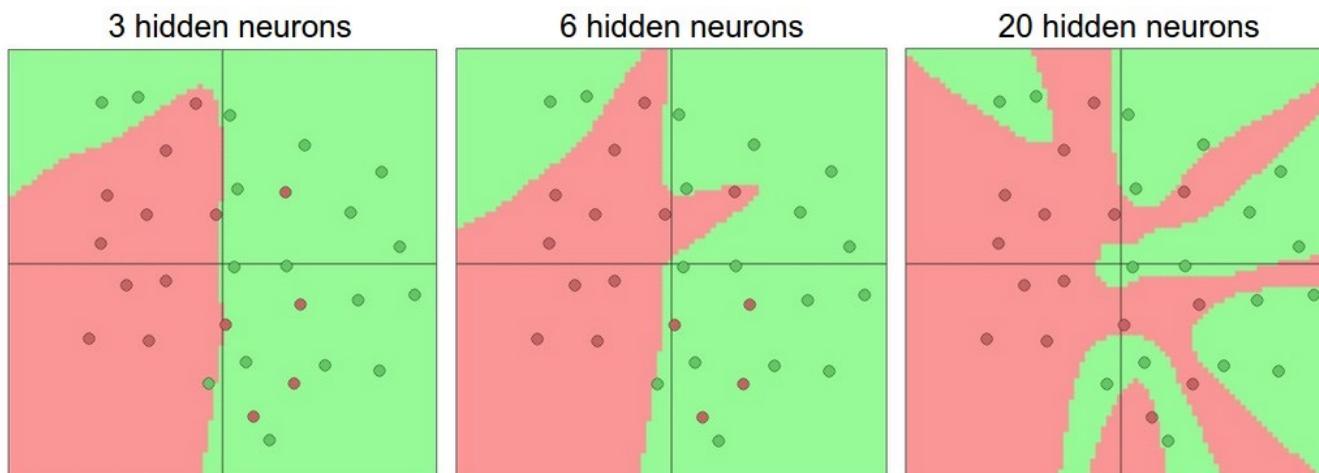
# The benefit of hidden layers



# Increasing number of neurons in the hidden layer

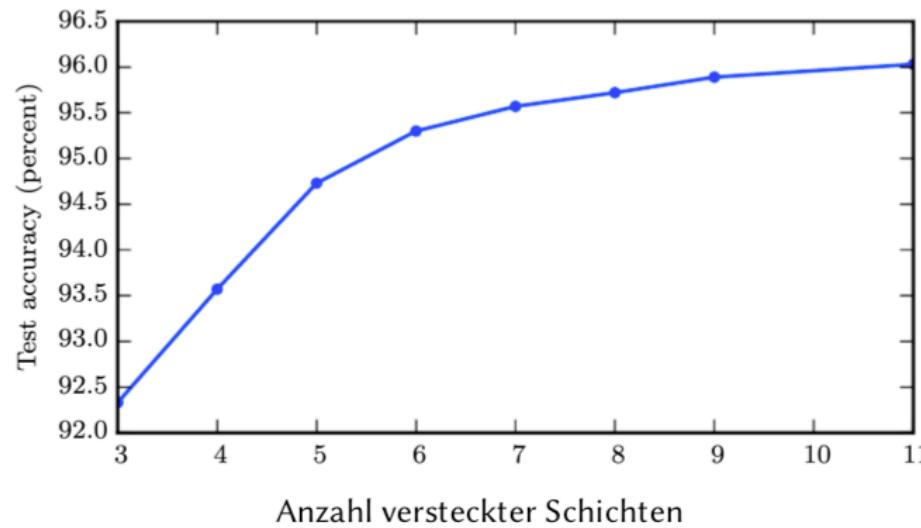


**A network with one hidden layer is a universal function approximator!**



# DL use many hidden layers

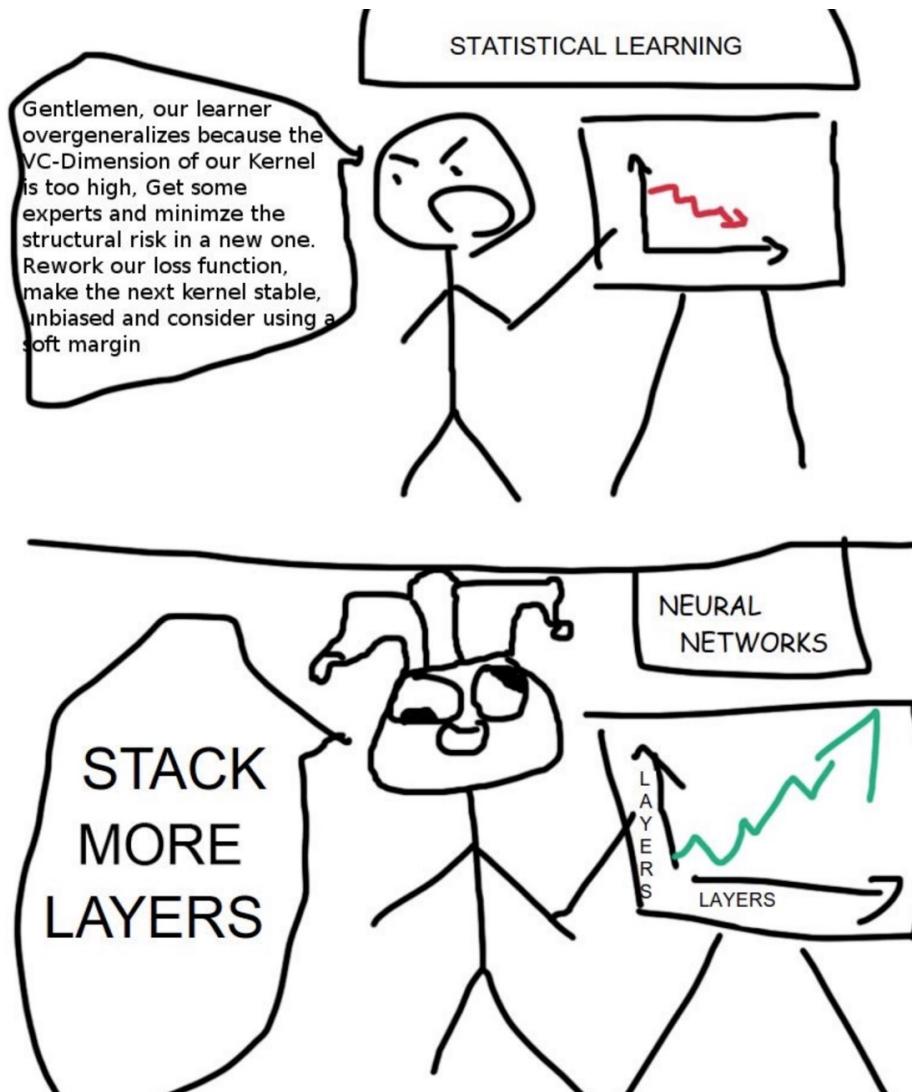
- Empirical observation that having more than one layer improves generalization (no overfitting)



[ Goodfellow et al. 2014]

- Not completely understood why. Some intuition:
  - Multiple layers allow hierarchical features
  - With same number of weights yield more flexible networks
  - Observed in brains

# DL vs Machine Learning Meme

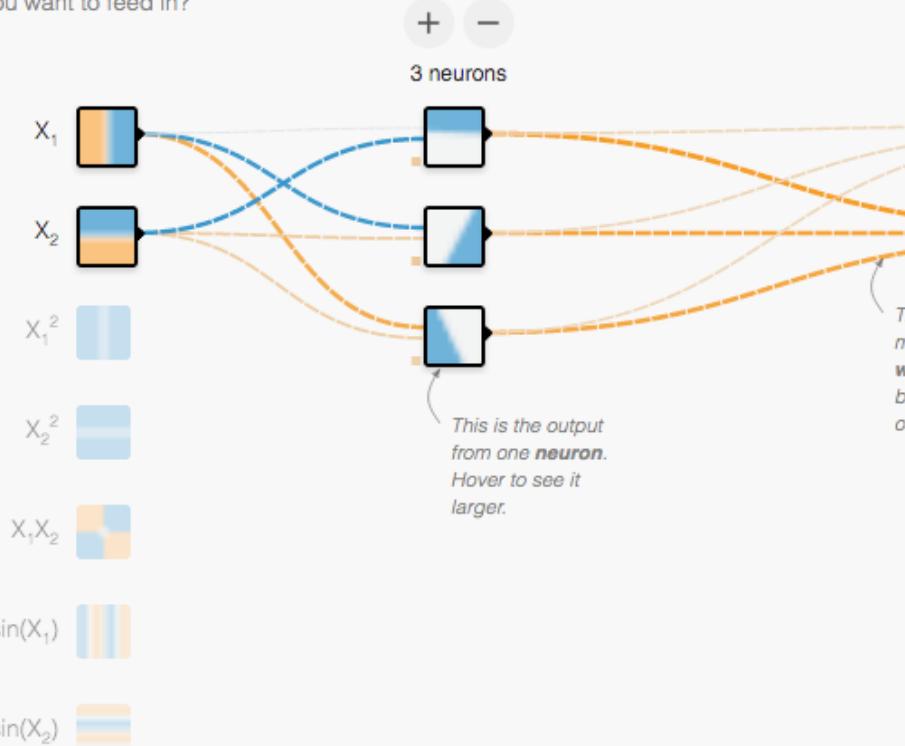


# Experiment yourself, play at home

## FEATURES

Which properties do you want to feed in?

+ - 2 HIDDEN LAYERS

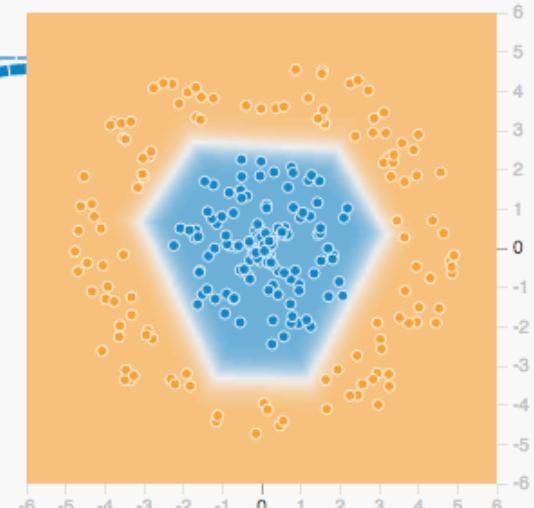


+

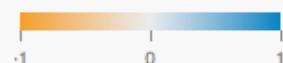
-

## OUTPUT

Test loss  
Training loss



Colors shows  
data, neuron and  
weight values.



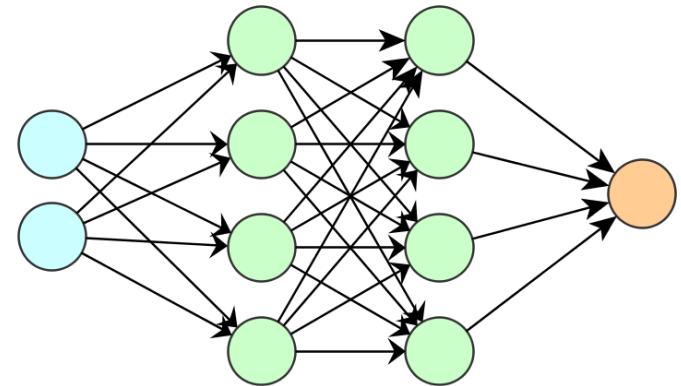
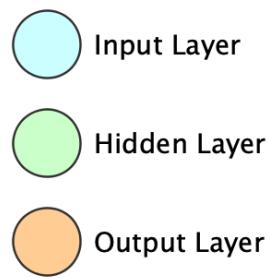
Show test data

Discretize output

<http://playground.tensorflow.org>

Let's you explore the effect of hidden layers

# Structure of the network



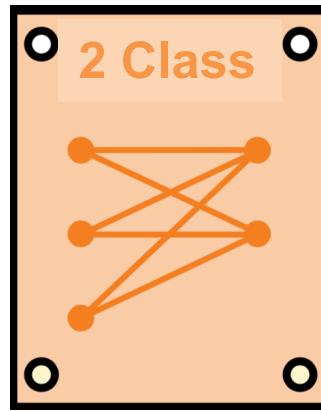
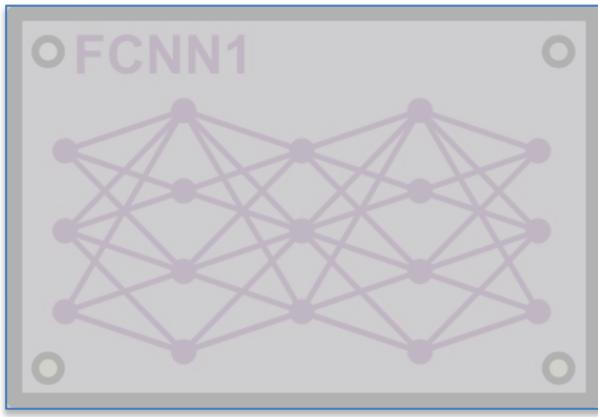
In code:

```
## Solution 2 hidden layers
hidden_1=sigmoid(X %*% W1 + b1)
hidden_2=sigmoid(hidden_1 %*% W2 + b2)
res = sigmoid(hidden_2 %*% W3 + b3)
```

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

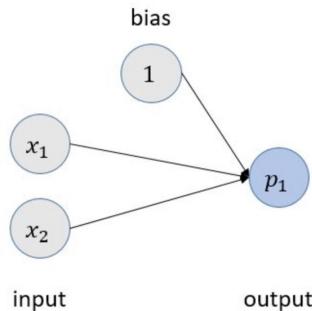
$$p = f(f(f(x W^1)W^2)W^3)$$

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

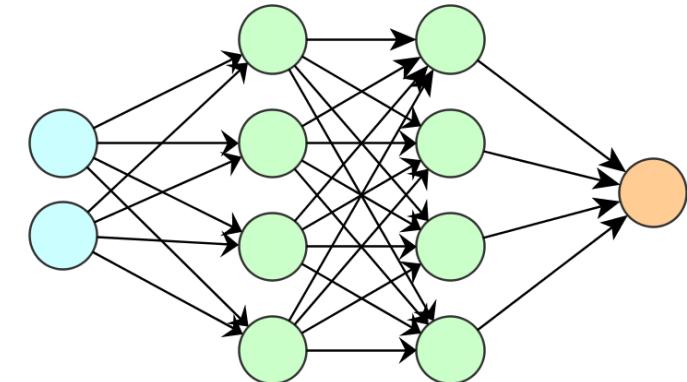
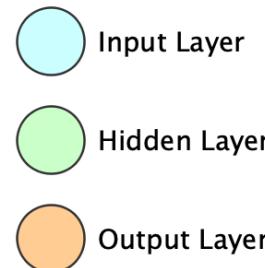


# Using Networks for Classification

# So far: Logistic Regression / Binary Classification



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



- Networks output the probability for one class (logistic regression / logistic regression with hidden layers).
  - In probabilistic framework: parameter of a Bernoulli ( $Y|x) \sim \text{Bern}(p_1(x))$ )
- What to do with more than one class?

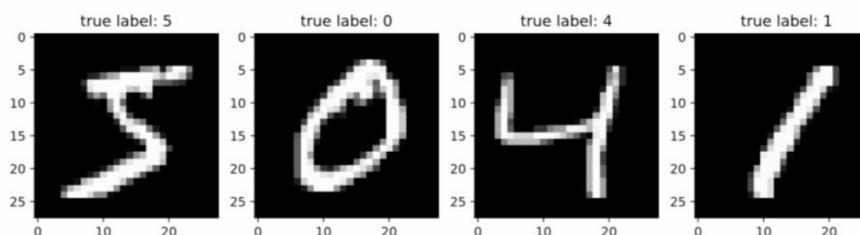
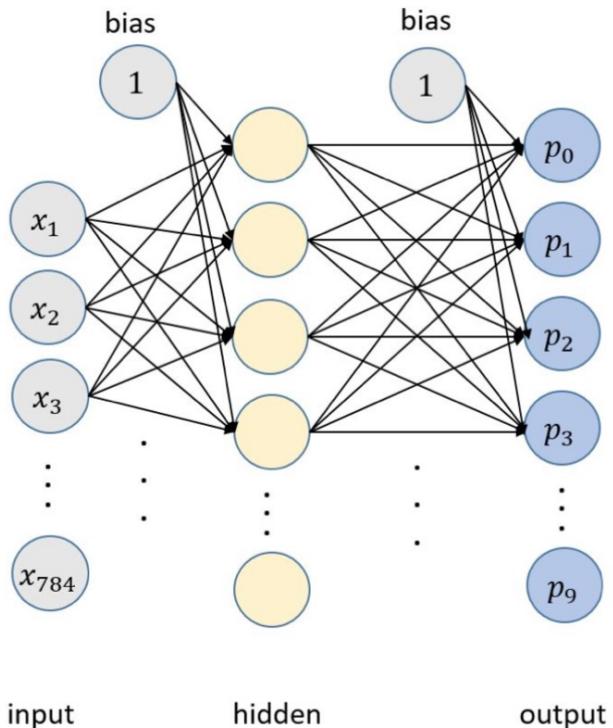


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

# Classification: Softmax Activation



$p_0, p_1 \dots p_9$  are probabilities for the classes 0 to 9.

Incoming to last layer  $z_i \ i = 0 \dots 9$

$$p_i = \frac{e^{z_i}}{\sum_{j=0}^9 e^{z_j}}$$

Makes outcome positive

Ensures that  $p_i$ 's sum up to one

This activation is called *softmax*

Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

Networks out the probabilities for the classes

In probabilistic framework (parameter vector  $\vec{p}$  of a multinomial  $\overrightarrow{Y|x} \sim mn(\overrightarrow{p(x)})$ )

# Training NN

# Training

Input  $x^{(i)}$

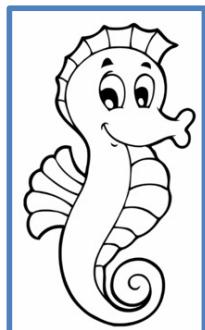


True class  $y^{(i)}$

Tiger



Tiger



Seeh  
orse

...

Typical 1 Mio. Trainingsdaten

Suggested class  
(Show is most likely class)

→ Seal 🤢

Neural network with many weights  $W$



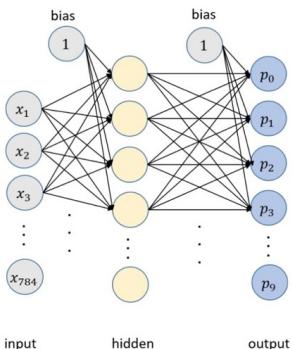
→ Tiger 🤗

→ Seehorse  
👍

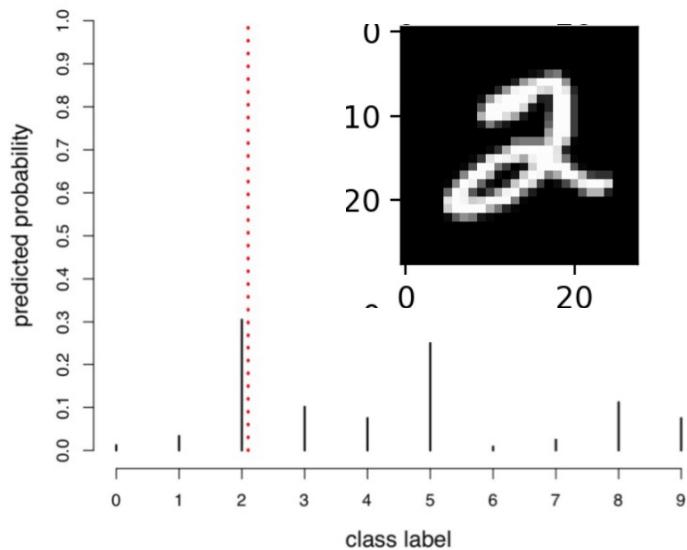
Trainingsprinciple:  
Weights are tuned so a loss function gets minimized.

$$\text{loss} = \text{loss}(\{y^{(i)}, x^{(i)}\}, W)$$

# Loss for classification ('categorical cross-entropy')



$p_0, p_1 \dots p_9$  are probabilities for the classes 0 to 9.



Definition (Negative Log-likelihood NLL / cat. crossentropy):

The loss  $l_i$  of a single training example  $x^{(i)}$  with true label  $y^{(i)}$  is

$$l_i = -\log(p_{model}(y^{(i)}|x^{(i)}))$$

Notation: if true label is 2 then  $p_{model}(y^{(i)}|x^{(i)}) = p_2$

- Perfect, i.e. predicts class of training example  $y^{(i)}$  with probability 1  $\Rightarrow l_i = 0$
- Worst, i.e. predicts class  $y^{(i)}$  with probability 0  $\Rightarrow l_i = \infty$

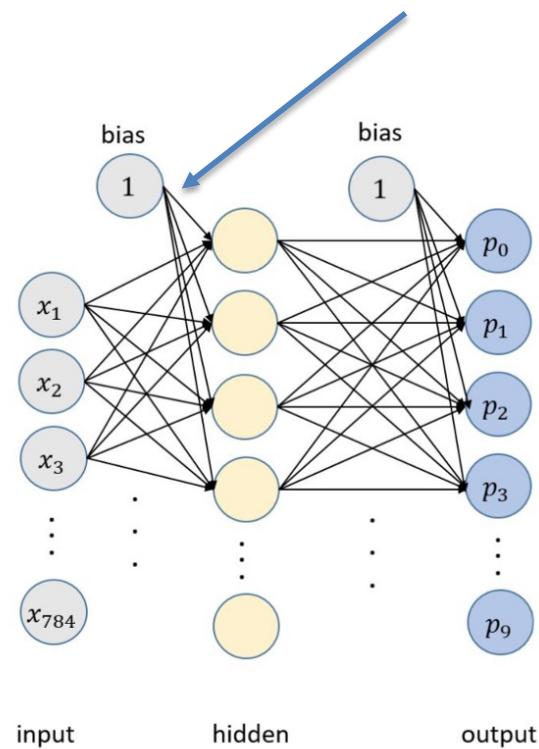
For more all examples, just average loss =  $\frac{1}{N} \sum l_i$

# Training / Gradient Descent

# Optimization in DL

- DL many parameters
  - Optimizing the loss by simple gradient descent
- Algorithm: Stochastic Gradient Descent (SGD)\*
  - Take a random batch of training examples  $\{y^{(i)}, x^{(i)}\}_{i=1,\dots,B}$
  - Calculate the loss of that batch  $\text{loss}(\{y^{(i)}, x^{(i)}\}, W)$
  - Tune the weights so that loss gets minimized a bit (gradient descent)
  - Repeat

Parameters of the network are the weights.



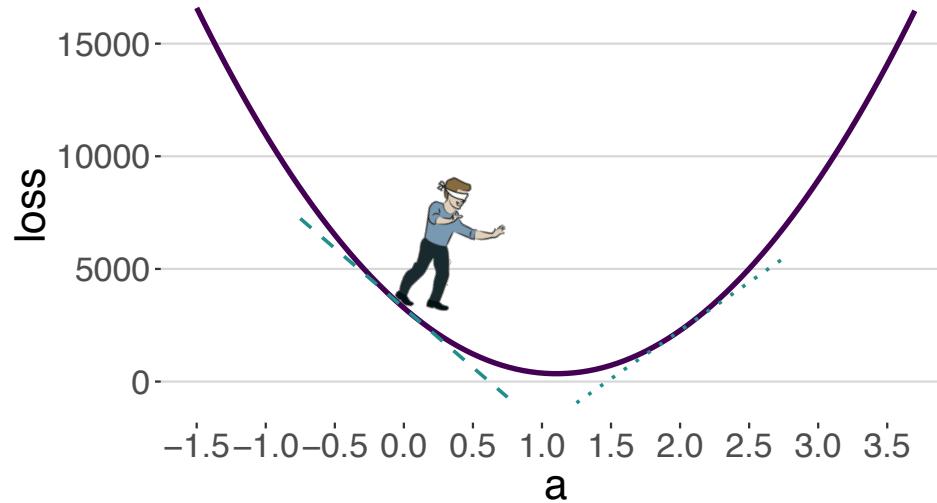
Modern Networks have Billions ( $10^9$ ) of weights.

Record 2020:  $175 \cdot 10^9$  (GPT-3)

\*aka minibatch gradient descent.

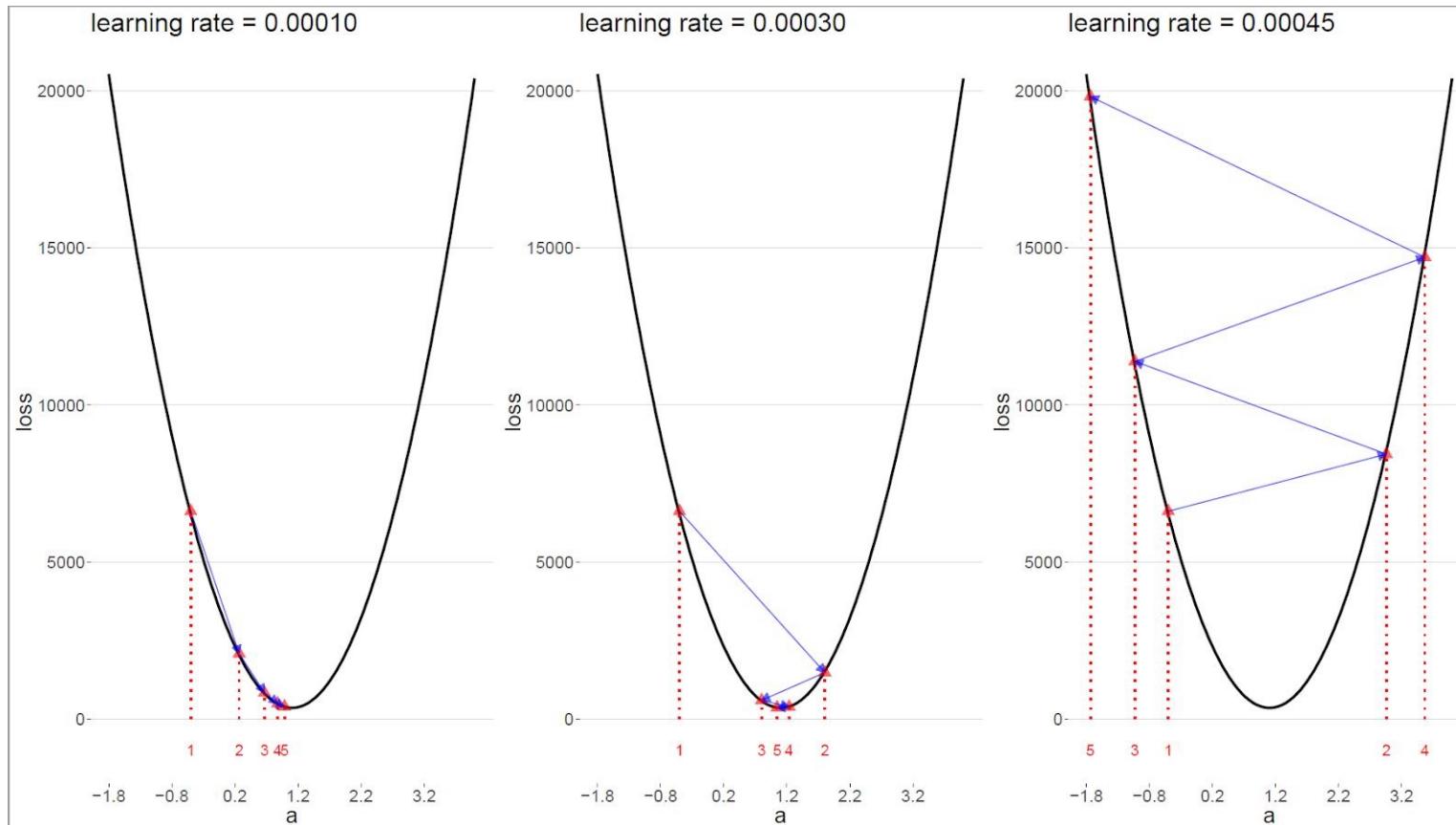
# Idea of gradient descent

- Shown loss function for a single parameter  $a$



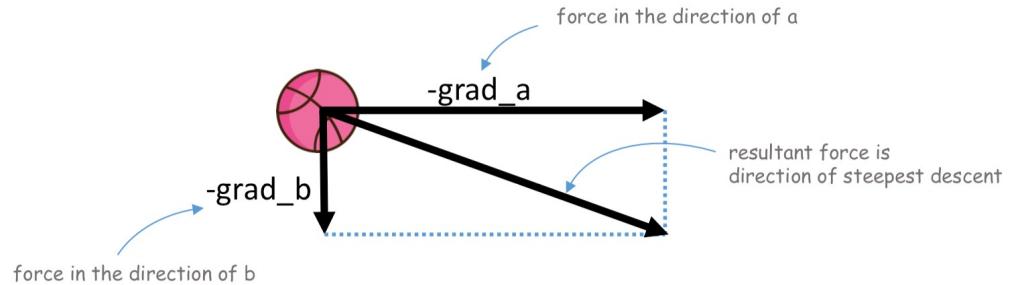
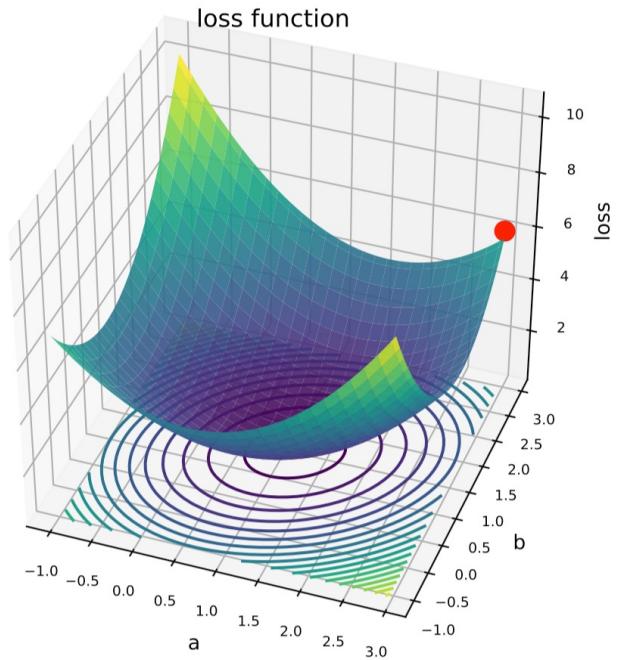
- Imagine you are the blinded wanderer and just know the loss and the slope at a position. How to reach the minimum?
  - Take a large step if slope is steep (you are away from minimum)
- Slope of loss function is given by gradient (this is a local quantity)
- Iterative update of the parameters
  - $a_{t+1} = a_t - \epsilon \text{ grad}_a(\text{loss})$

# Proper learning rate (Important parameter for DL)

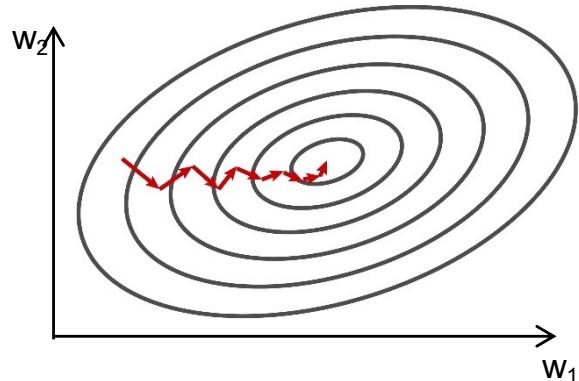


See: <https://developers.google.com/machine-learning/crash-course/fitter/graph>

## In two dimensions



Gradient is perpendicular to contour lines



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

# The miracle of gradient descent in DL

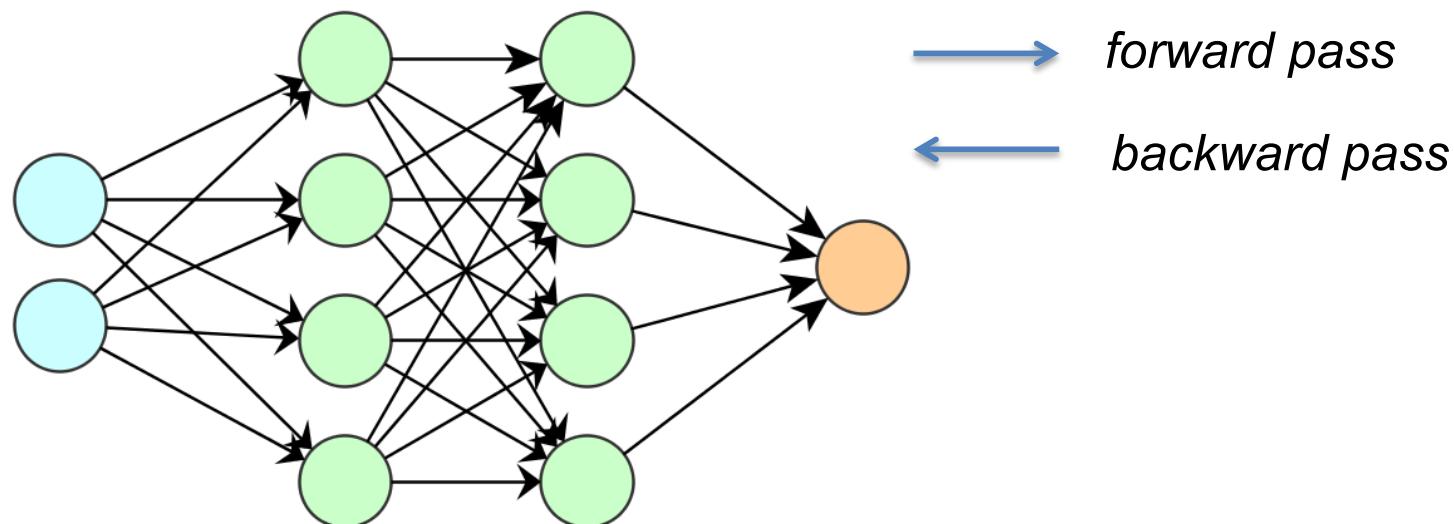


Loss surface in DL (is not convex) but SGD magically also works for non-convex problems.

Modern deep learning: No distinction between network (model) and training (SGD)

# Backpropagation

- We need to calculate the derivative of the loss function  $\text{loss}(\{X_i, Y_i\}, W)$  w.r.t. all weights  $W$
- Efficient way backpropagation (chain rule)
  - Forward Pass propagate training example through network
    - Gives output for current configuration of network
  - Backward pass propagate training example through network
    - Chain rule all gradients can be calculated in a single flow from the “end”



For more see e.g. chapter 3 in Probabilistic deep learning

# Typical Training Curve / ReLU

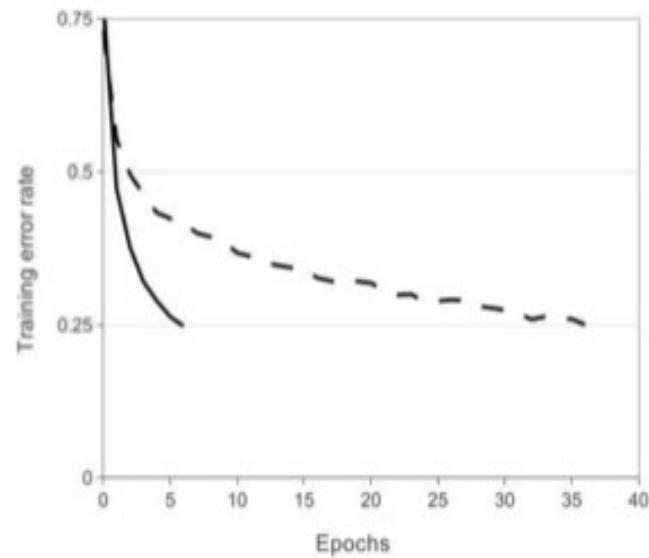
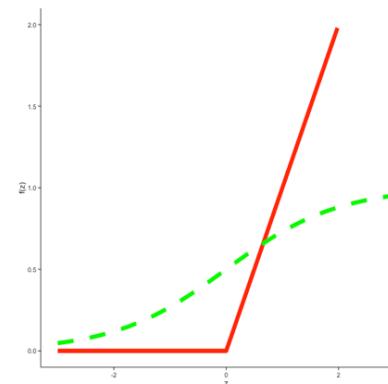
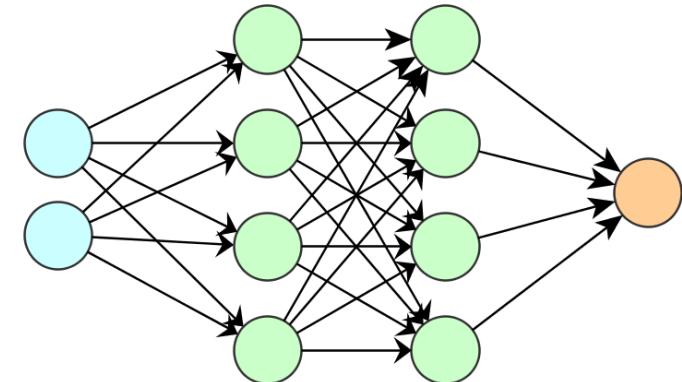
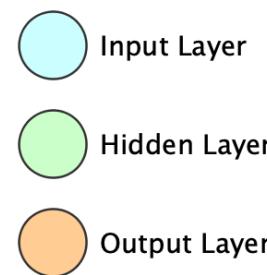


Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

Source:  
Alexnet  
Krizhevsky et al 2012

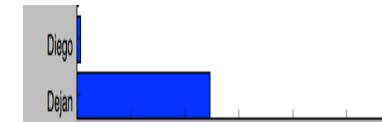
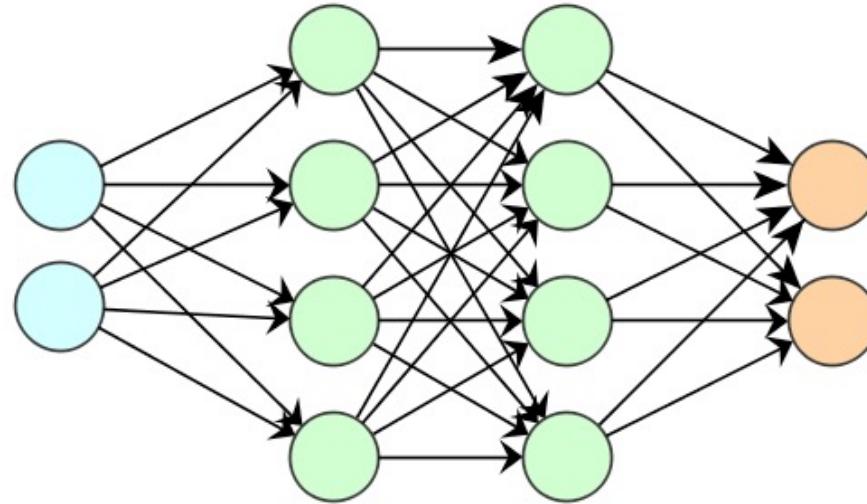
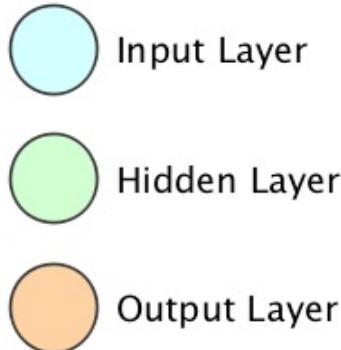


Motivation:  
**Green:**  
sigmoid.  
**Red:**  
ReLU faster  
convergence

Epochs: "each training examples is used once"

# Deep Learning Frameworks

# Recap: The first network



- The input: e.g. intensity values of pixels of an image
  - (Almost) no pre-processing
- Output: probability that image belongs to certain class
- Information is processed layer by layer from building blocks
- Arrows are weights (these need to be learned / training)
- Training requires gradients of loss w.r.t weights

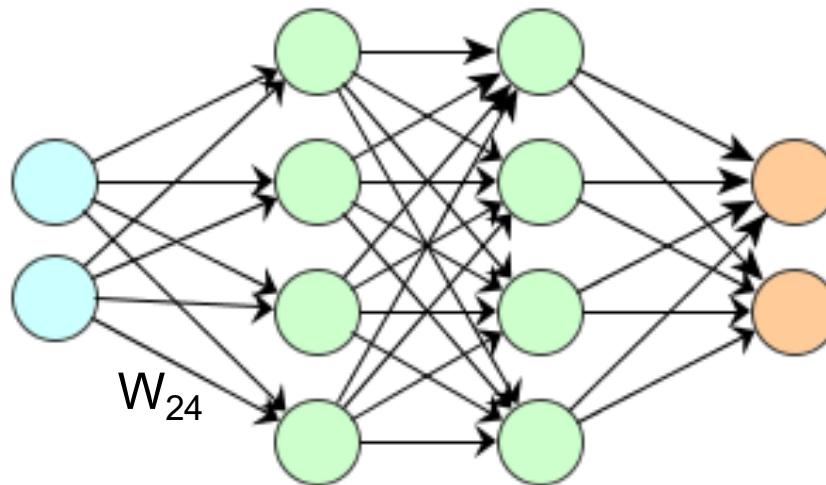


# Deep Learning Frameworks (common)

- Computation needs to be done on GPU or specialized hardware (compute performance)
- Data Structure are multidimensional arrays (*tensors*) which are manipulated
- Automatic calculation of the gradients
  - Static: computational graph (see chapter 3 in probabilistic deep learning)
  - Dynamic: reverse mode auto diff
- In this course: TensorFlow with Keras (and TensorFlow Probability)



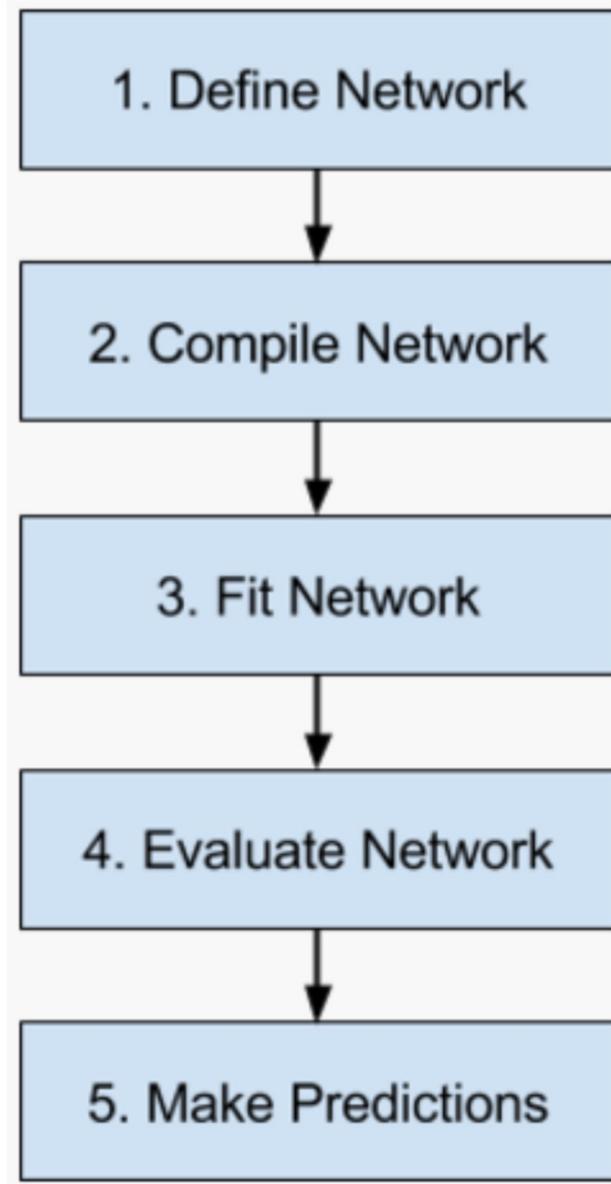
# Typical Tensors in Deep Learning



- 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.

# Introduction to Keras

# Keras Workflow



Define the network (layerwise)

Add loss and optimization method

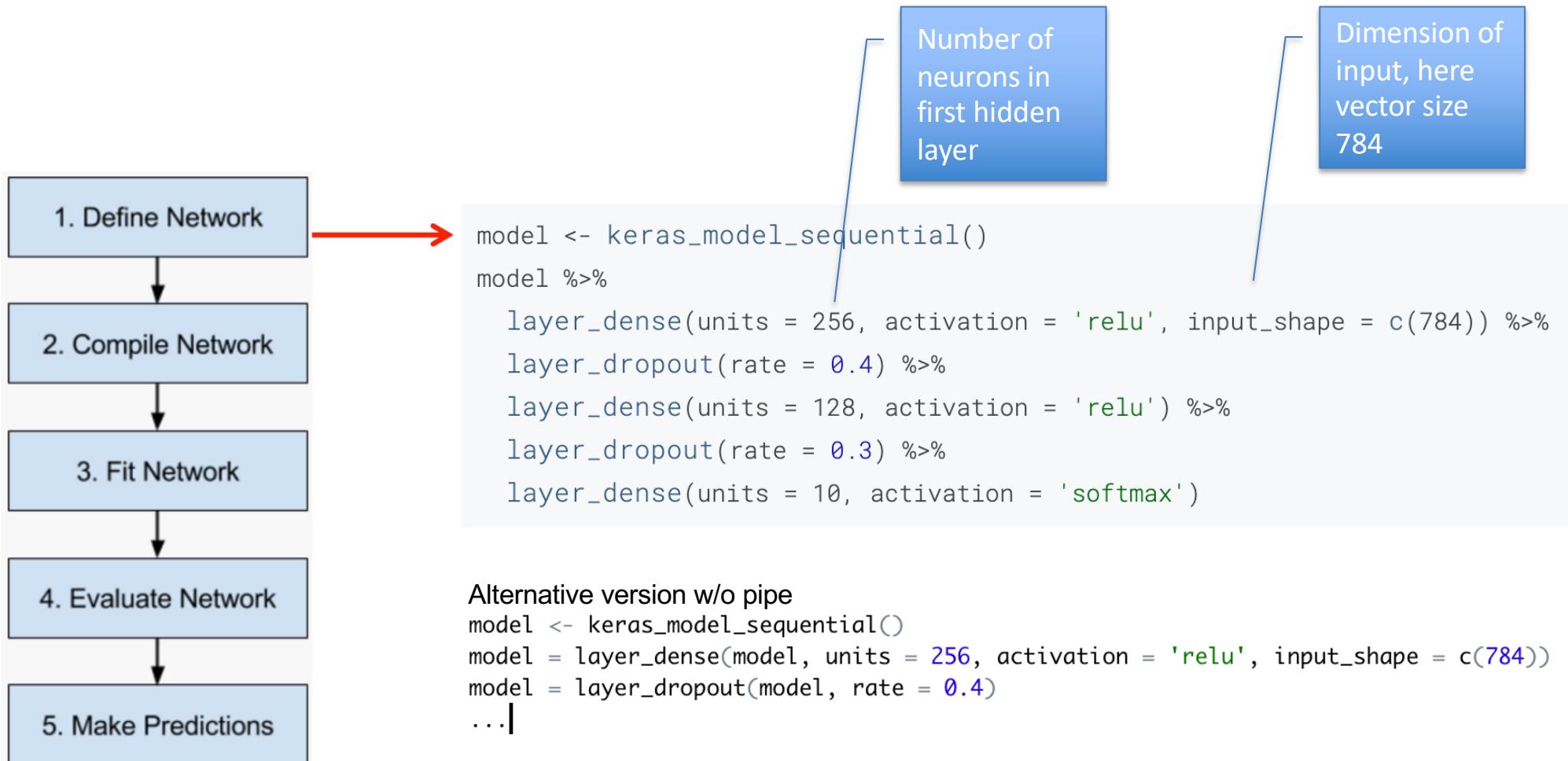
Fit network to training data

Evaluate network on test data

Use in production

# A first run through

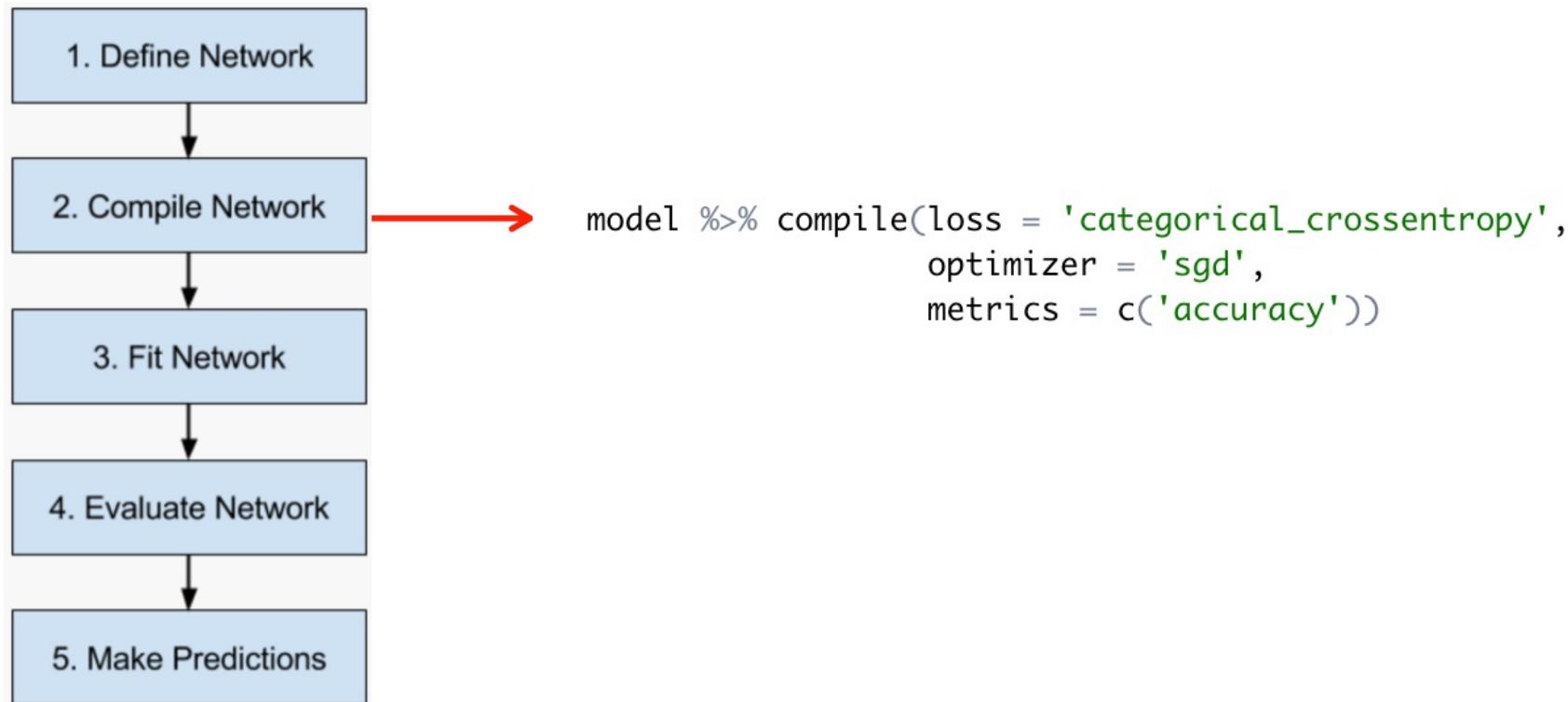
# Define the network



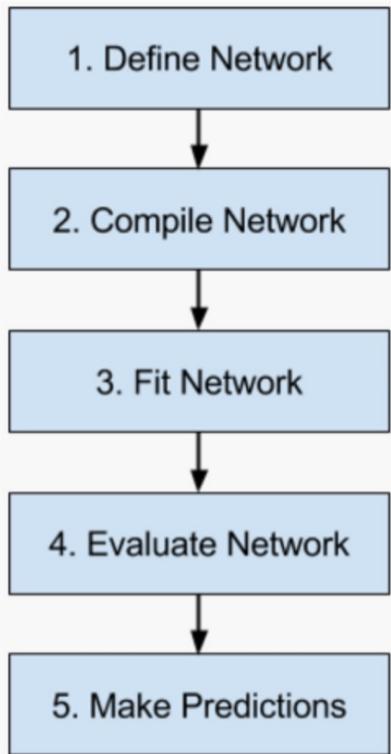
Input shape needs to be defined only at the beginning.

Alternative: `input_dim=784`

# Compile the network

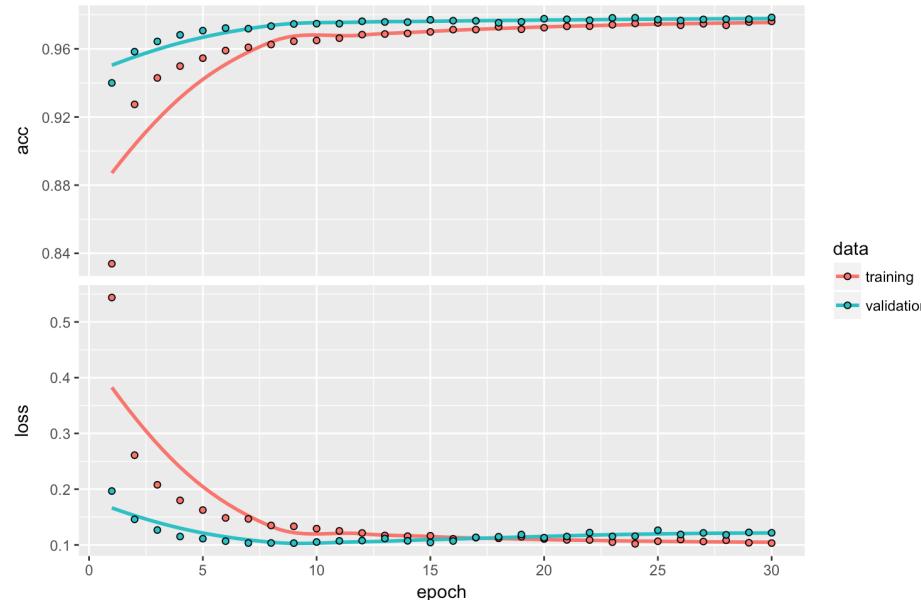


# Fit the network



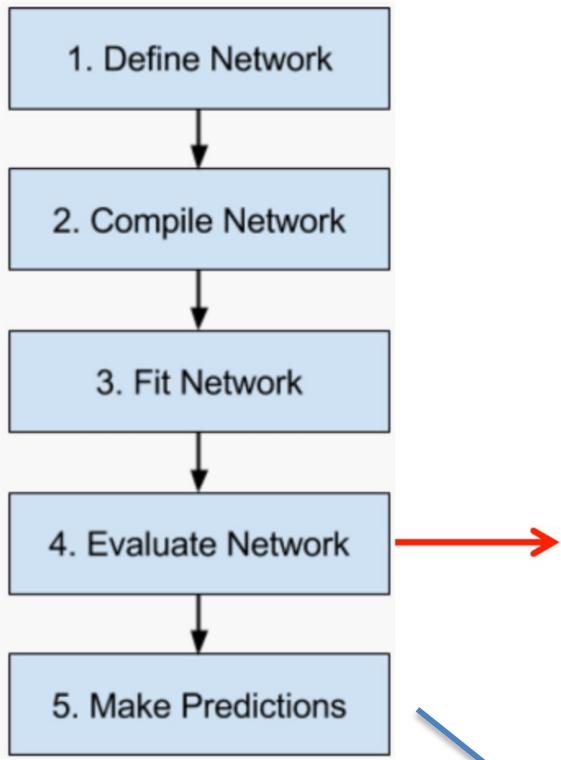
```
history <- model %>% fit(  
    x_train, y_train,  
    epochs = 30, batch_size = 128,  
    validation_split = 0.2  
)
```

```
plot(history)
```



20% of the data is not used for fitting the weights.

# Evaluate the network



Evaluate the model's performance on the test data:

```
model %>% evaluate(x_test, y_test)
```

```
$loss  
[1] 0.1149
```

```
$acc  
[1] 0.9807
```

```
#starting tf version 2.6  
model %>% predict(x) %>% `>` (0.5) #for binary  
predict(model,x) %>% k_argmax()#for multiclass
```

```
model %>% predict_classes(x_test)
```

```
[1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1
```

Deprecated

## More layers

- Dropout
  - `keras.layers.Dropout`
- Convolutional (see lecture on CNN)
  - `keras.layers.Conv2D`
  - `keras.layers.Conv1D`
- Pooling (see lecture on CNN)
  - `keras.layers.MaxPooling2D`
- Recurrent (not in course)
  - `keras.layers.SimpleRNNCell`
  - `keras.layers.GRU`
  - `keras.layers.LSTM`



# How to use TF and Keras in the course

- Use google colab
  - Free resource with preinstalled (kind of) jupyter notebooks
  - Usually for python
  - To start R notebook (not possible over GUI)
    - <https://colab.research.google.com/notebook#create=true&language=r>
- Use RStudio
  - Installation a bit tedious, especially for tfprobability
  - You might by lucky though

# Excercise



For this lecture:

Play around with code, answer questions ask questions if you have any. See also  
[https://tensorchiefs.github.io/dl\\_rcourse\\_2022/](https://tensorchiefs.github.io/dl_rcourse_2022/)

1. Check installation colab 00\_R\_Keras\_TF\_TFP.ipynb

[https://colab.research.google.com/github/tensorchiefs/dl\\_rcourse\\_2022/blob/main/notebooks/00\\_R\\_Keras\\_TF\\_TFP.ipynb](https://colab.research.google.com/github/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/00_R_Keras_TF_TFP.ipynb)

2. Banknote example (01\_nb\_ch02\_01.ipynb):

[https://colab.research.google.com/github/tensorchiefs/dl\\_rcourse\\_2022/blob/main/notebooks/01\\_nb\\_ch02\\_01.ipynb](https://colab.research.google.com/github/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/01_nb_ch02_01.ipynb)

Optional / If time do Solution with Tensorflow

3. MNIST with simple FCNN (02\_nb\_ch02\_02a.ipynb)

[https://colab.research.google.com/github/tensorchiefs/dl\\_rcourse\\_2022/blob/main/notebooks/02\\_nb\\_ch02\\_02a.ipynb](https://colab.research.google.com/github/tensorchiefs/dl_rcourse_2022/blob/main/notebooks/02_nb_ch02_02a.ipynb)