Machine Intelligence:: Deep Learning
Week 4

Beate Sick, Oliver Dürr, Pascal Bühler

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Status: Slight changes possible before lecture

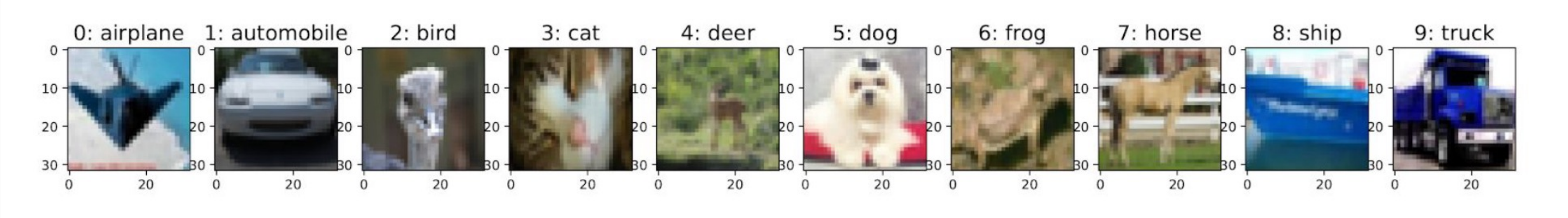
Outline of the DL Module (tentative)

- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II
  - Tips and Tricks
  - Modern Architectures
  - Biological Inspiration
- Day 4: Looking at details
  - CNNs for Sequence Data (for projects)
  - Linear Regression ()
  - Backpropagation
  - 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

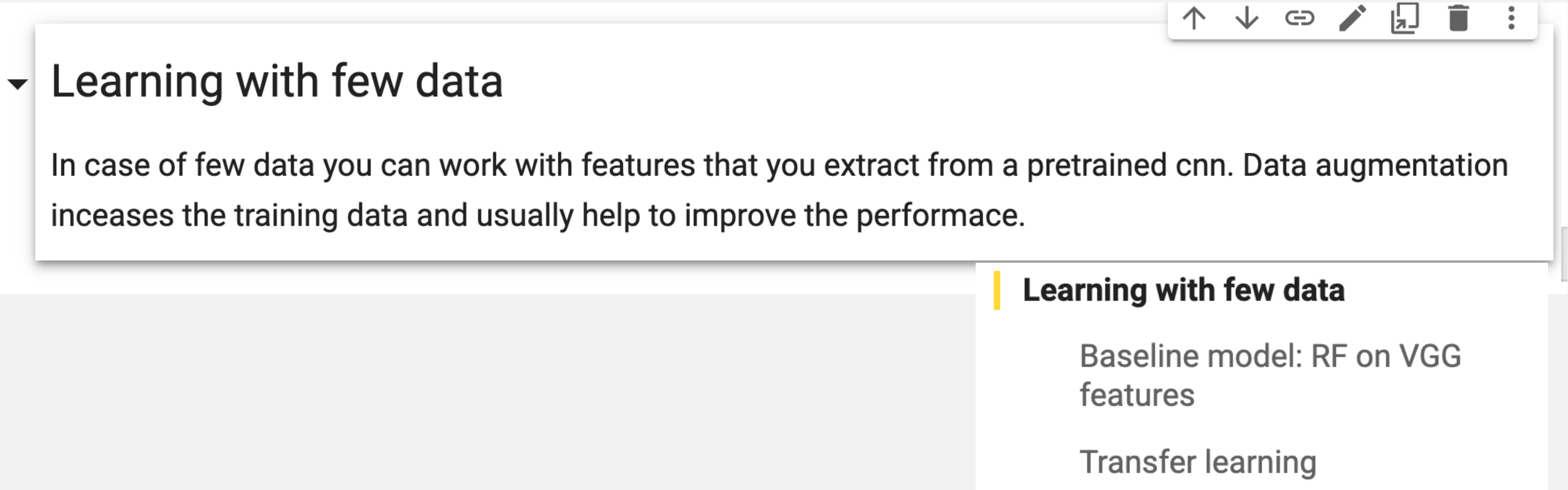
Projects: Please register until next week for projects

Besprechung

08_classification_transfer_learning_few_labels



Notebook:



Learning with few data

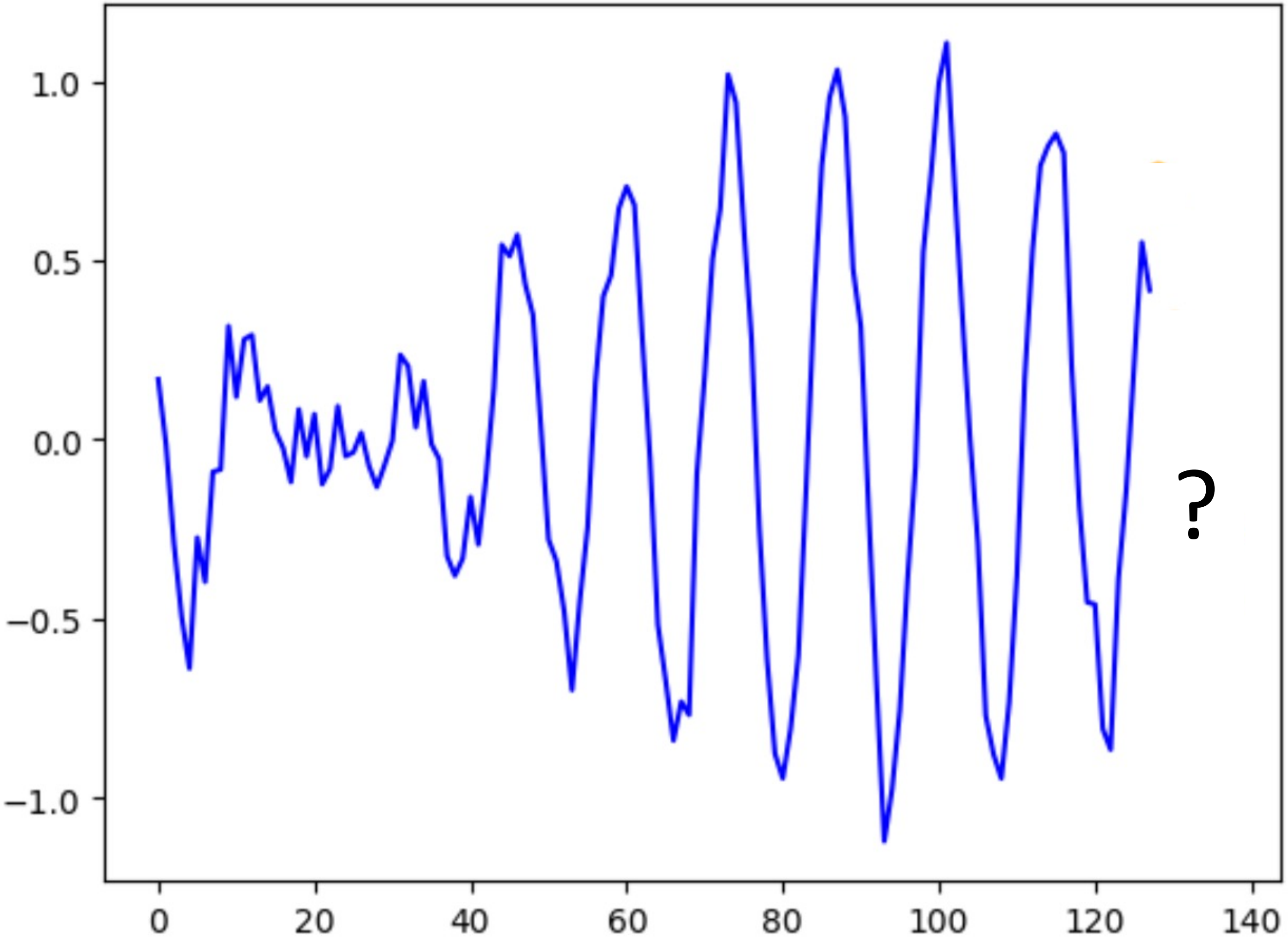
In case of few data you can work with features that you extract from a pretrained cnn. Data augmentation inceases the training data and usually help to improve the performace.

Learning with few data

- Baseline model: RF on VGG features
- Transfer learning

1D CNNs for sequence data

How to make predictions based on a given time series?

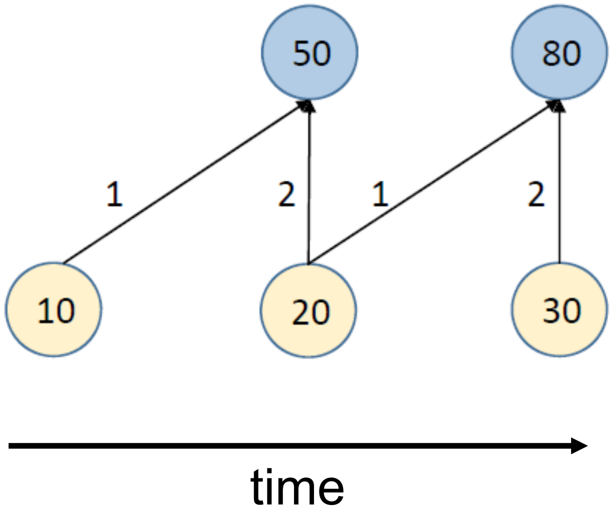


1D "causal" convolution for time-ordered data

Toy example:

Input X: 10,20,30

1D kernel of size 2: 1,2



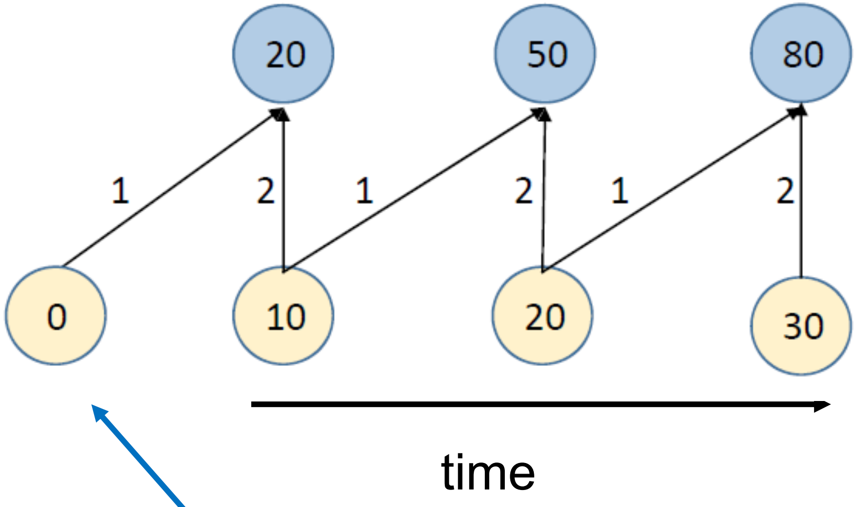
It's called "causal" networks, because the architecture ensured that only information from the past has an influence on the present and future.

Zero-padding in 1D "causal" CNNs

Toy example:

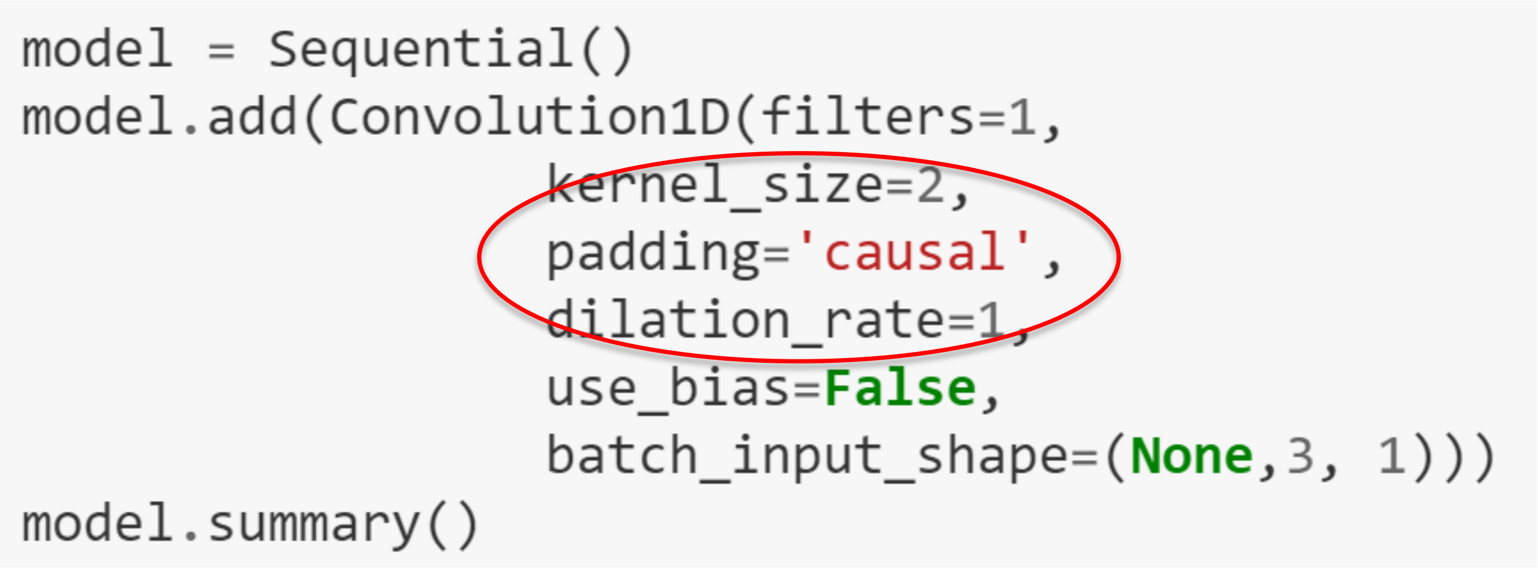
Input X: 10,20,30

1D kernel of size 2: 1,2



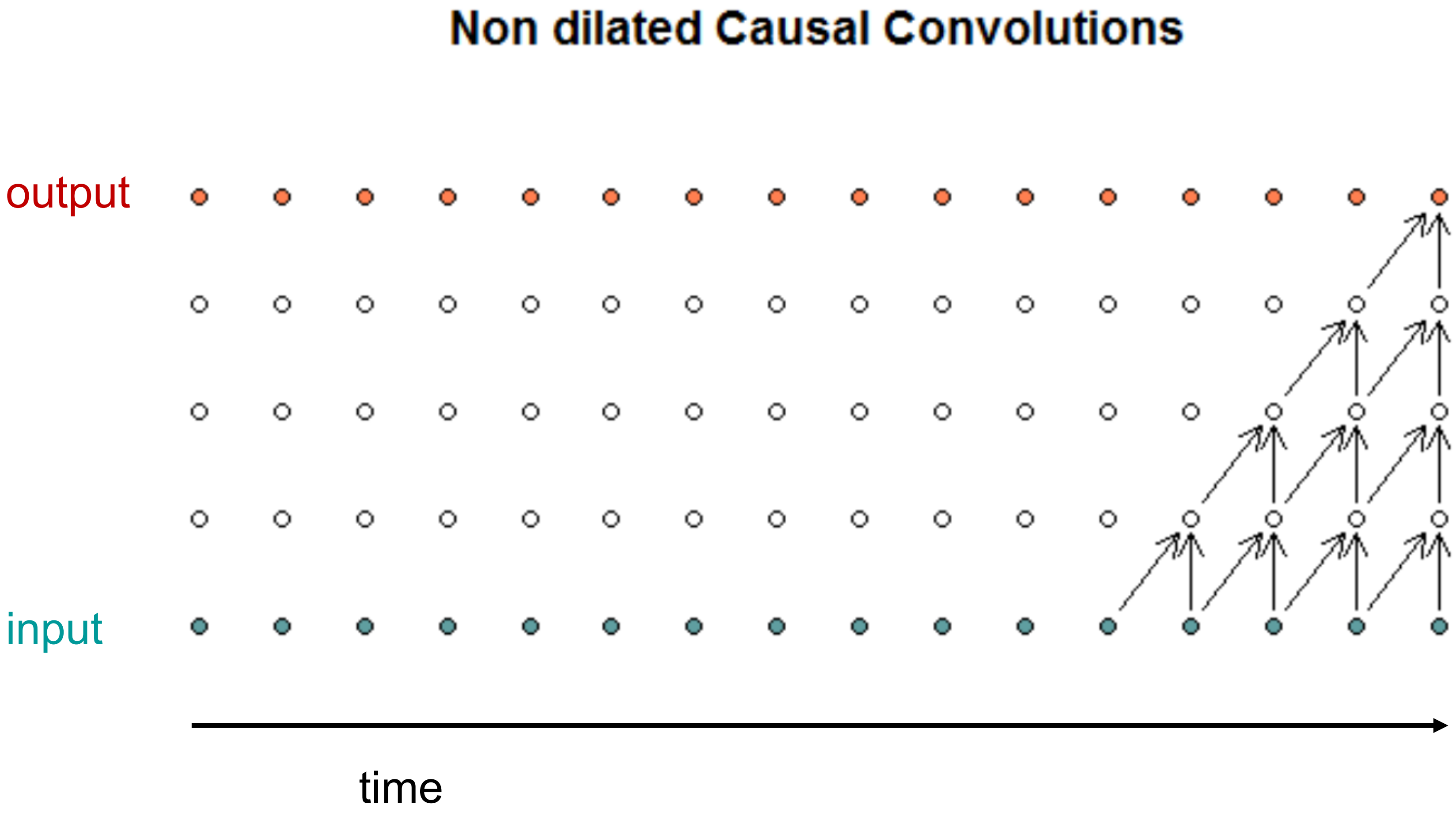
To make all layers the same size, a zero padding is added to the beginning of the input layers

1D "causal" convolution in Keras



```
model = Sequential()
model.add(Convolution1D(filters=1,
                        kernel_size=2,
                        padding='causal',
                        dilation_rate=1,
                        use_bias=False,
                        batch_input_shape=(None,3, 1)))
model.summary()
```

Stacking 1D “causal” convolutions without dilation



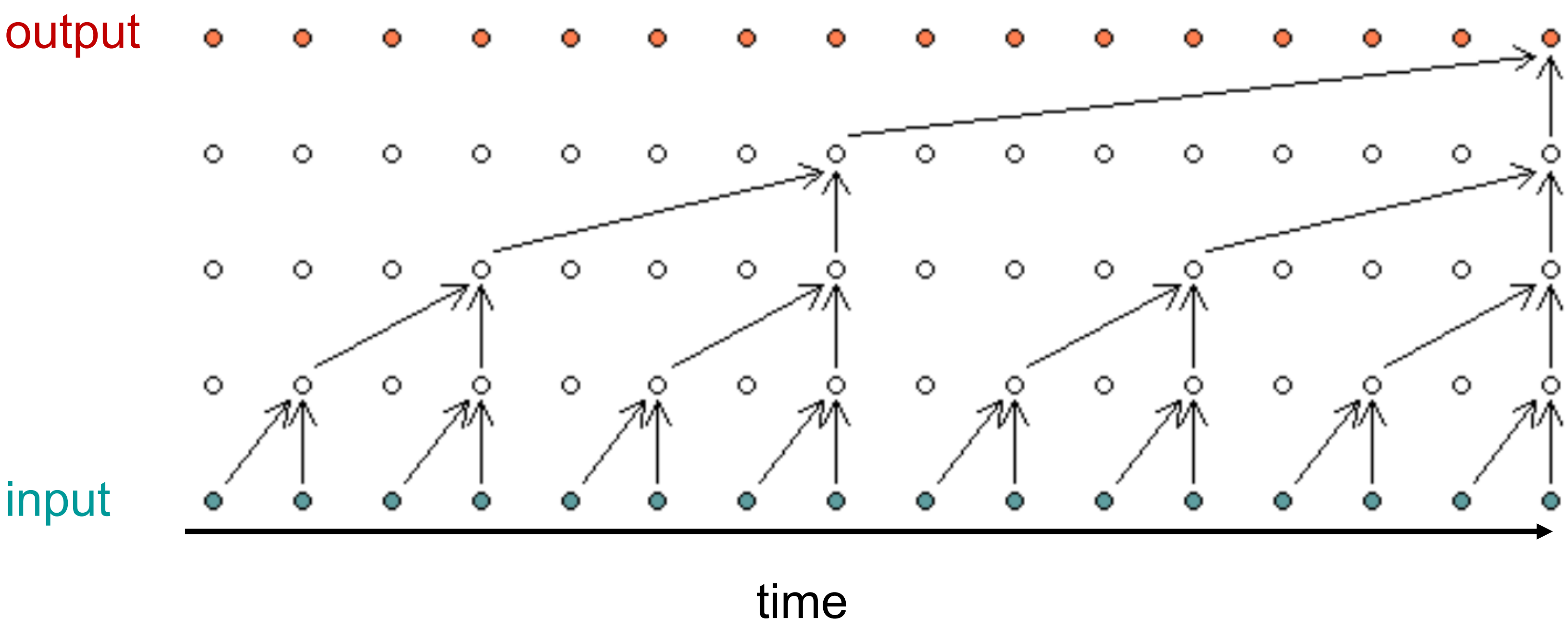
Stacking k causal 1D convolutions with kernel size 2 allows to look back k time-steps.

After 4 layers each neuron has a “memory” of 4 time-steps back in the past.

https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Dilation allows to increase receptive field

To increase the memory of neurons in the output layer, you can use "dilated" convolutions:

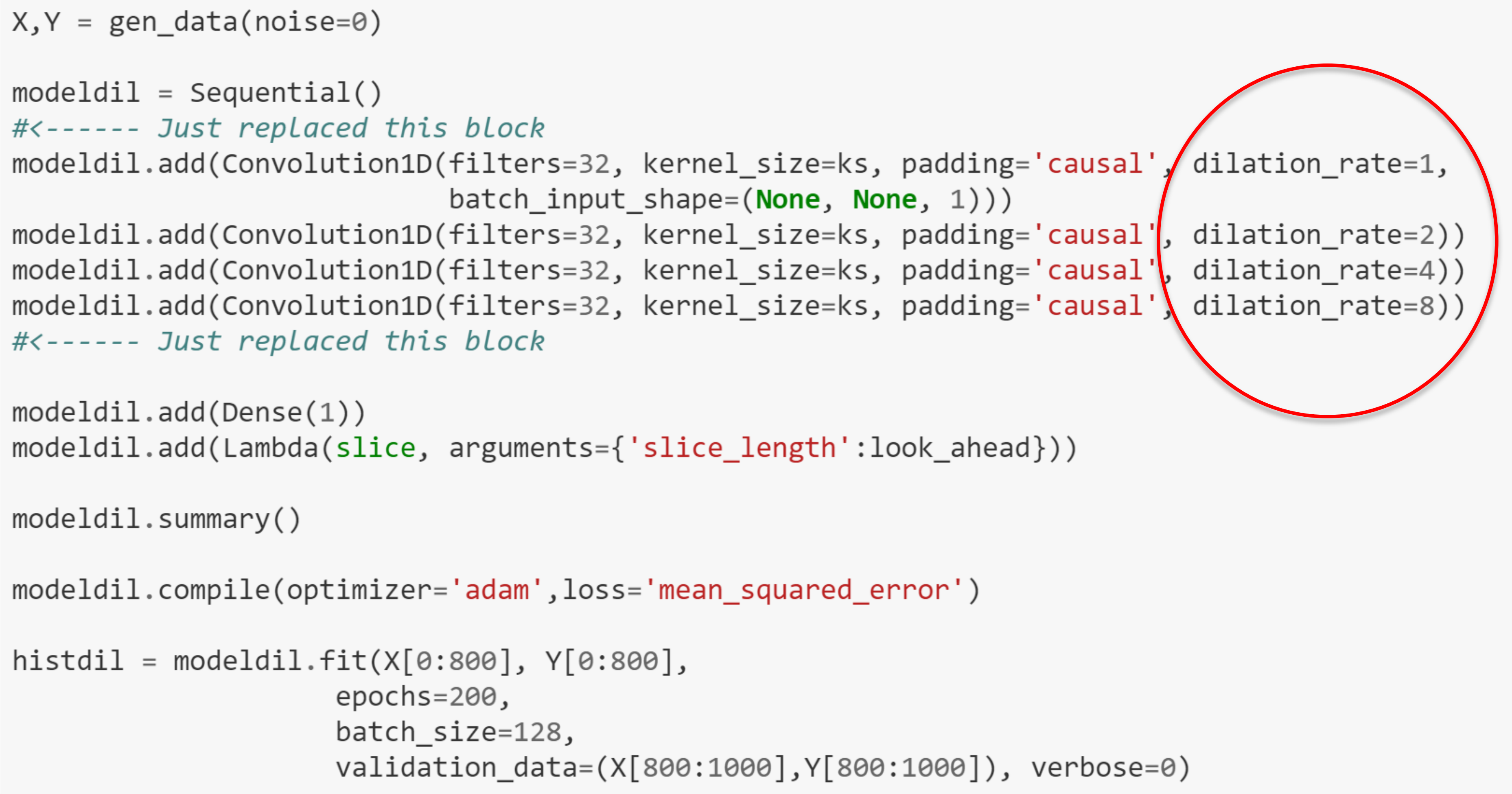


After 4 layers each neuron has a "memory" of 15 time-steps back in the past.

https://deepmind.com/blog/wavenet-generative-model-raw-audio/

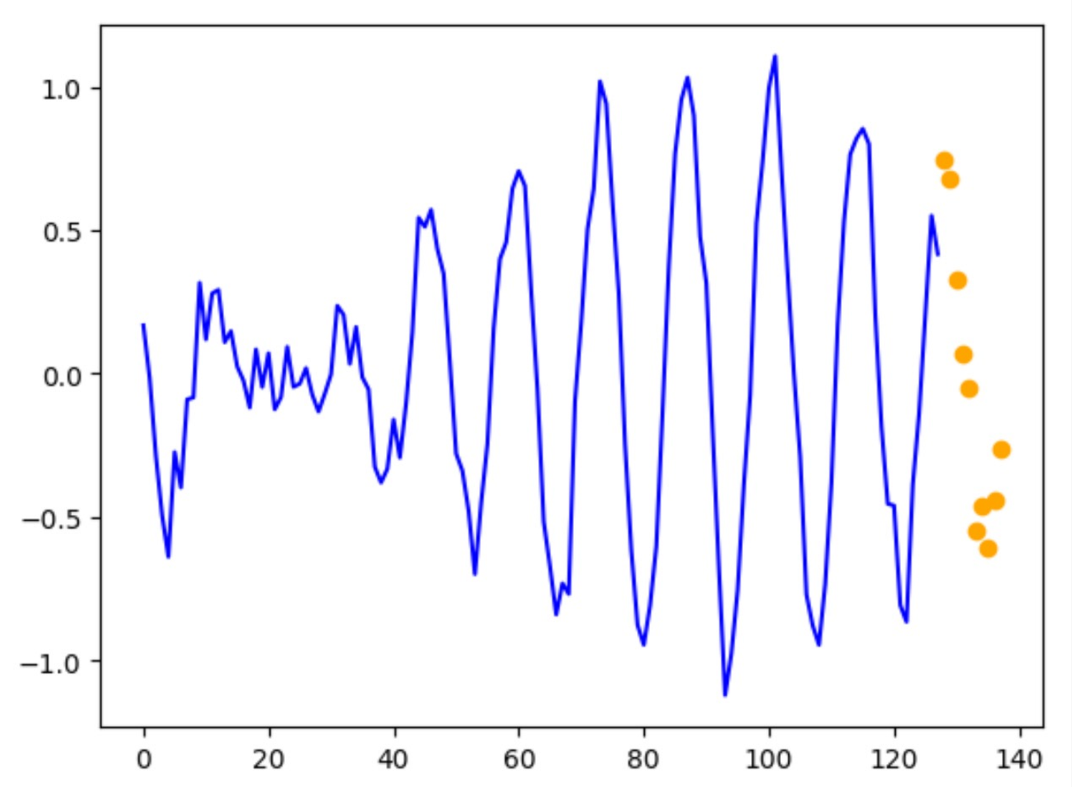
Dilated 1D causal convolution in Keras

To use time-dilated convolutions, simply use the argument rate dilation_rate=... in the Convolution1D layer.



```
X,Y = gen_data(noise=0)

modeldil = Sequential()
#<------ Just replaced this block
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=1,
                           batch_input_shape=(None, None, 1)))
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=2))
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=4))
modeldil.add(Convolution1D(filters=32, kernel_size=ks, padding='causal', dilation_rate=8))
#<------ Just replaced this block

modeldil.add(Dense(1))
modeldil.add(Lambda(slice, arguments={'slice_length':look_ahead}))

modeldil.summary()

modeldil.compile(optimizer='adam',loss='mean_squared_error')

histdil = modeldil.fit(X[0:800], Y[0:800],
                    epochs=200,
                    batch_size=128,
                    validation_data=(X[800:1000],Y[800:1000]), verbose=0)
```

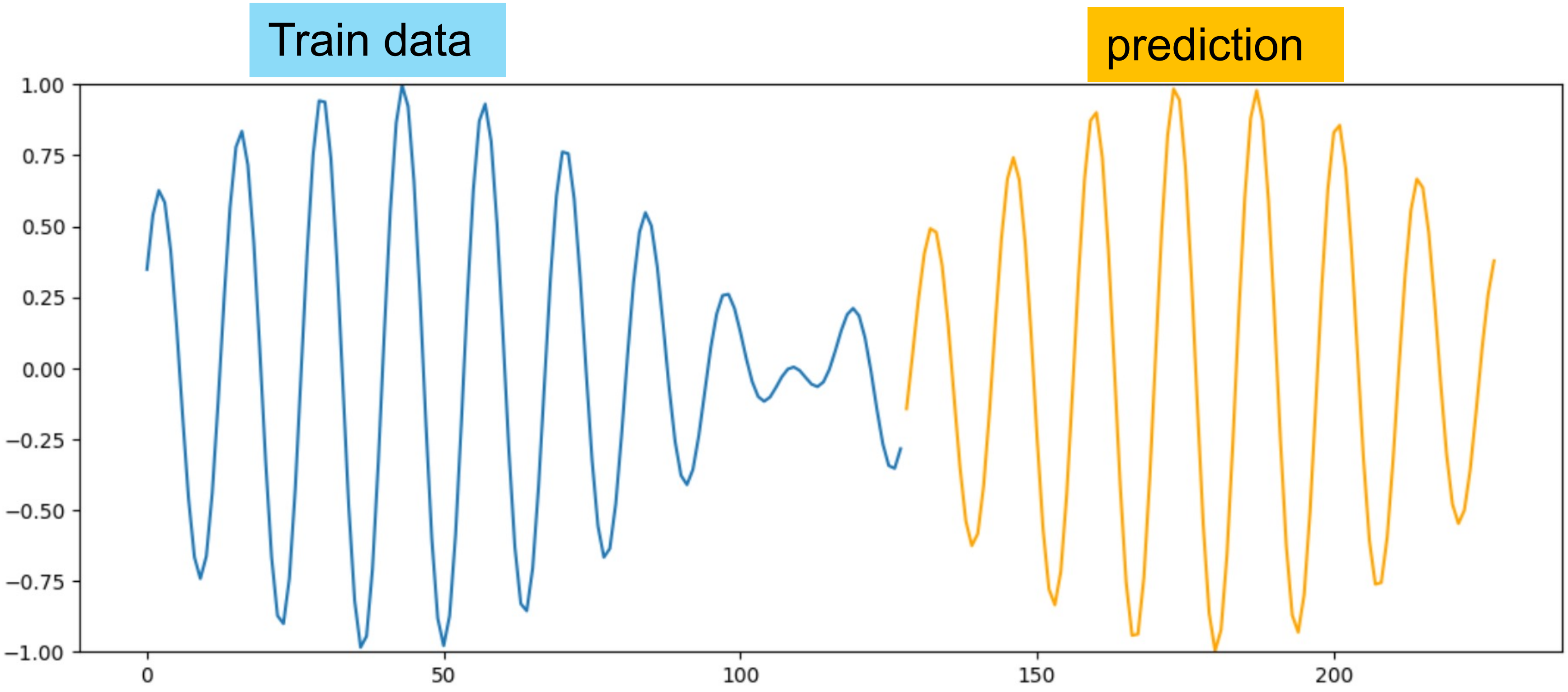
1D-Convolution (Notebook just for Reference)



Work through the notebook (optional)
https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/09_1DConv.ipynb.

Dilated 1D causal CNNs help if long memory is needed

Dilated 1D CNNs can picked up the long-range time dependencies.



Train data

prediction

If you want to get a better understanding how 1D convolution work, you can go through the notebook at (optional in case you want to work with 1D CNN) https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/09_1DConv_sol.ipynb.

Learning Objectives for today: looking under the hood

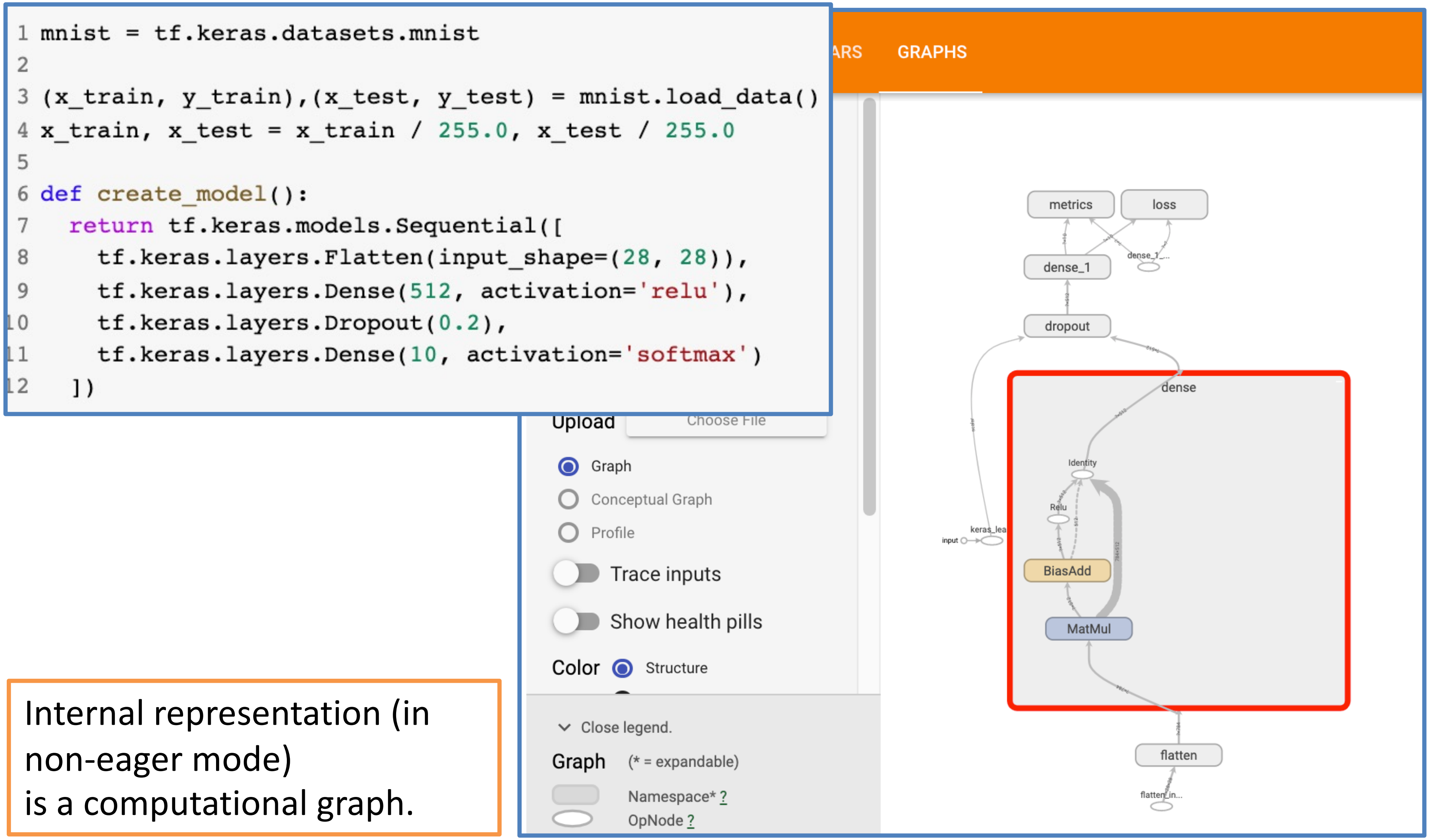
- Get an understanding of
  - Computational Graph
  - Backpropagation in Computational Graph
  - Maximum Likelihood principle for neural networks

Computational Graph

Looking under the hood of tf / Keras

Keras

TensorFlow



```
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

def create_model():
  return tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
  ])
```

Internal representation (in non-eager mode)
is a computational graph.

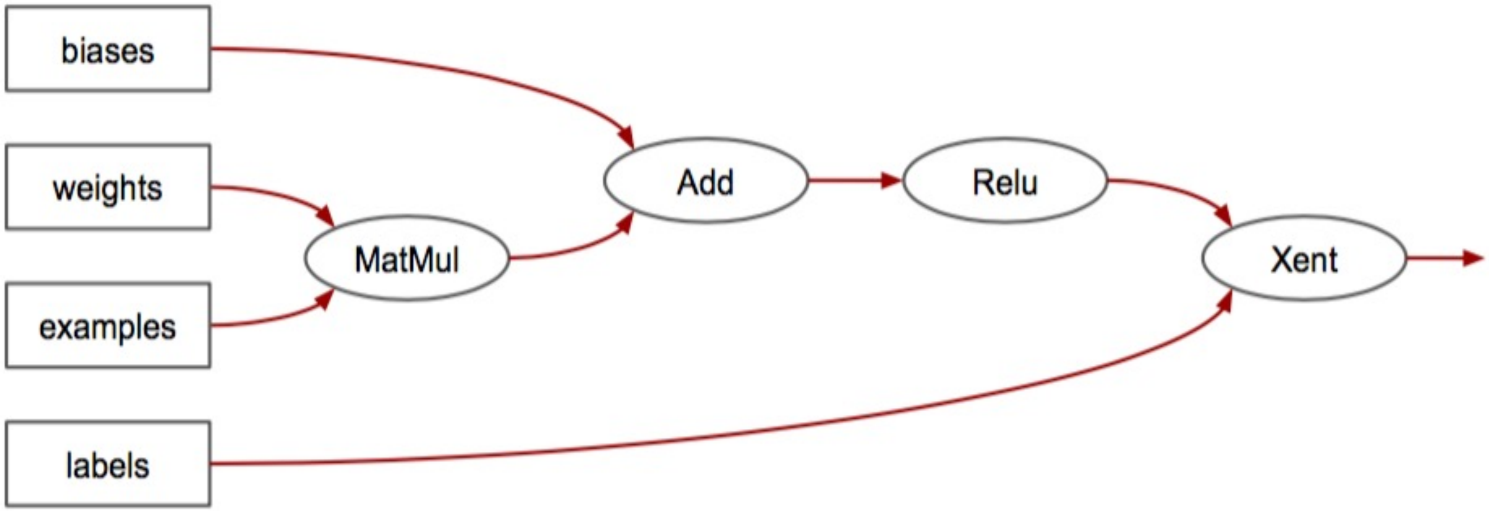
https://github.com/tensorflow/tensorboard/blob/master/docs/get_started.ipynb

Next steps

- Understand the computational graph (theoretical)
- Understand backpropagation in a graph (theoretical)
- Example Linear Regression (the mother of all networks)

Recap

- The computation in TF is done via a computational graph

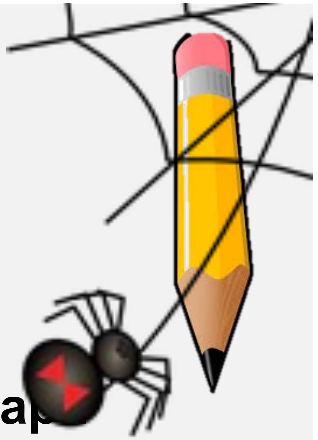


- The nodes are ops
- The edges are the flowing tensors

Recap Matrix Multiplication (scalar and with vector)

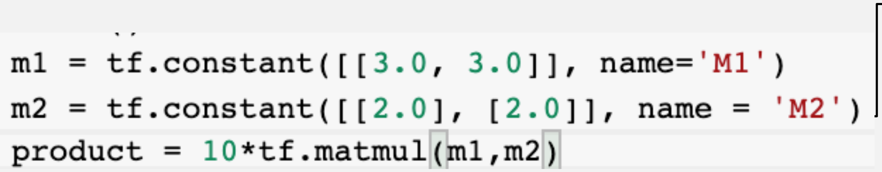
$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

Be the spider who knits a computational graph



Translate the following TF code in a graph

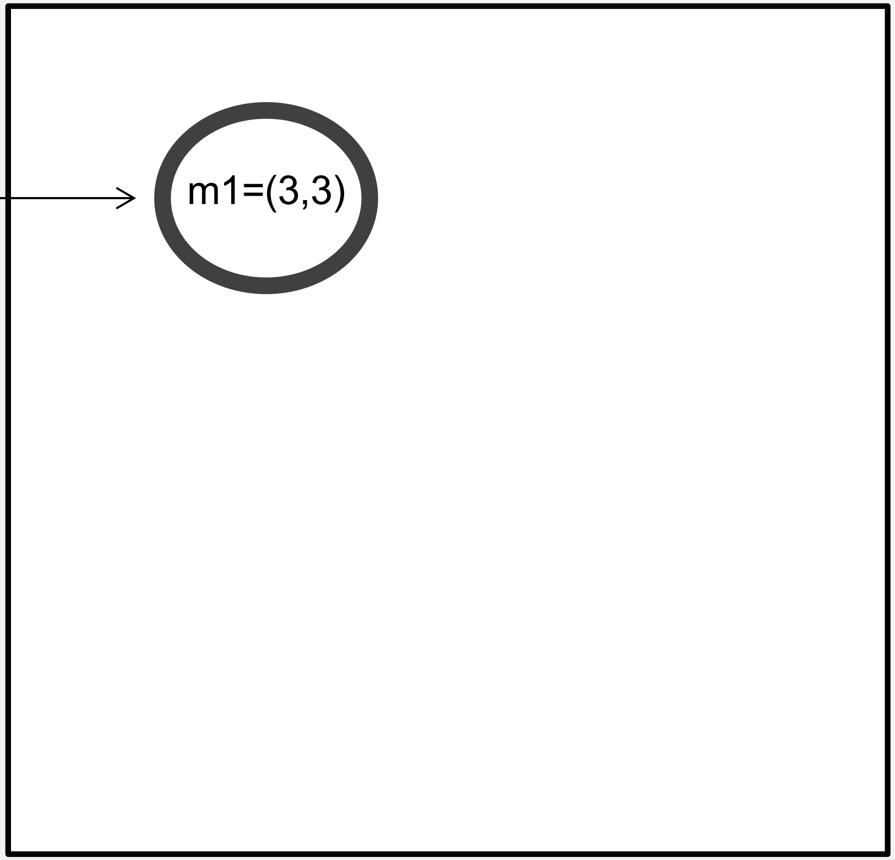
TensorFlow: Building the graph



```
m1 = tf.constant([[3.0, 3.0]], name='M1')
m2 = tf.constant([[2.0], [2.0]], name = 'M2')
product = 10*tf.matmul(m1,m2)
```

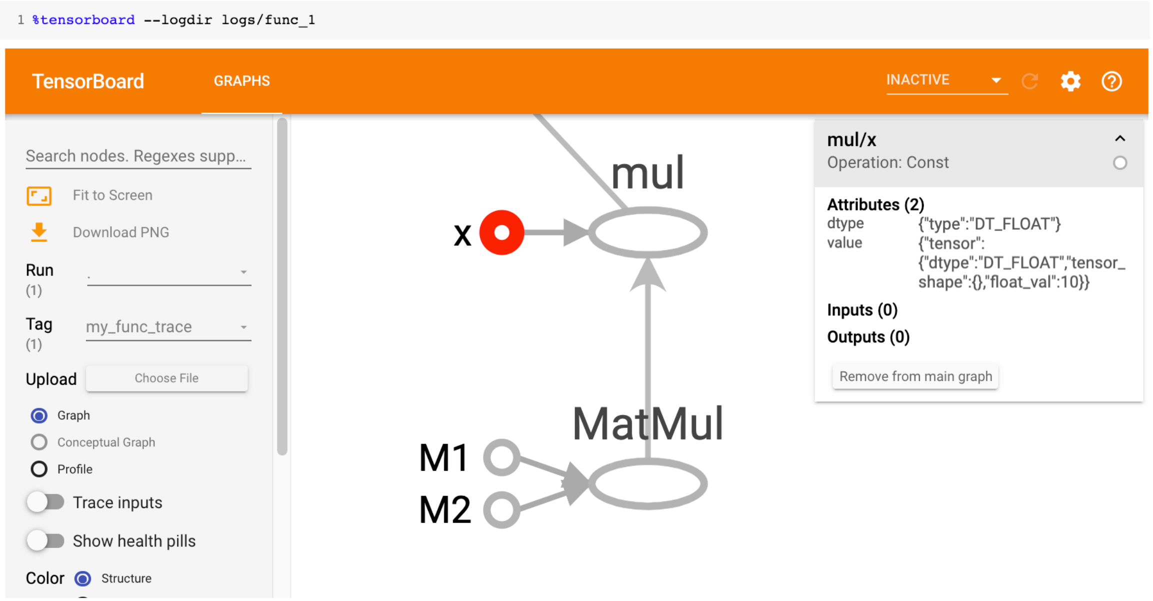
Quite much happen in here!

Finish the computation grap



TensorFlows internal representation

- For fast computation a graph is build
  - Technical detail in tf 2.0 you need to decorate a function with @tf.function to build a graph. Otherwise eager execution happens.



```
%tensorboard --logdir logs/func_1
```

Run (1) .

Tag (1) my_func_trace

○ Graph

○ Conceptual Graph

○ Profile

Color ○ Structure

mul/x
Operation: Const

Attributes (2)

dtype	{"type":"DT_FLOAT"}
value	{"tensor": {"dtype":"DT_FLOAT","tensor_shape":{},"float_val":10}}

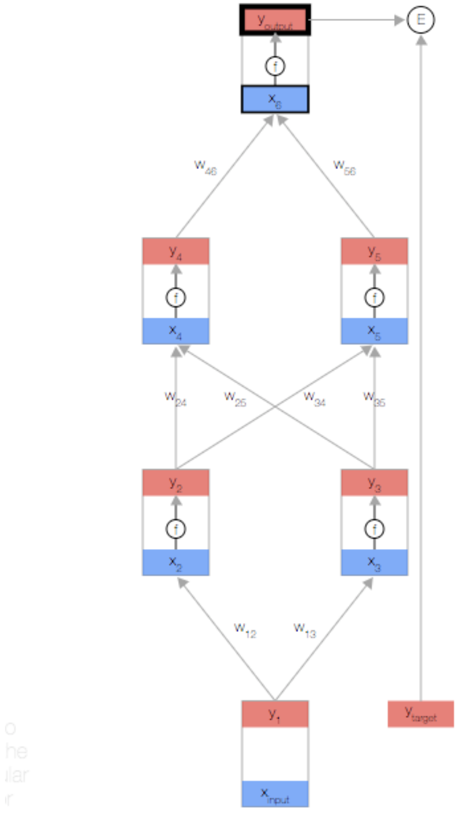
Inputs (0)

Outputs (0)

The most important benefit of computational graphs is back propagation…

Motivation: The forward and the backward pass

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



Scroll until the forward pass and swiftly go over the backward pass.

(The backward pass is described in more details in the next following slides).

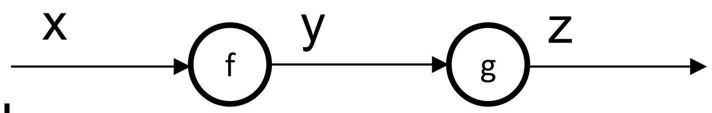
Chain rule recap

- If we have two functions f,g

  $y = f(x)\ and$

  $z = g(y)$

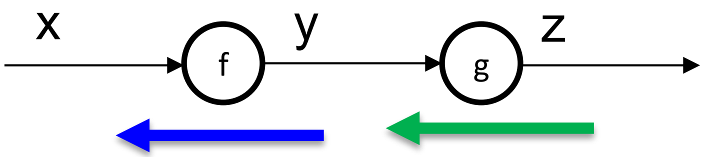
  then y and z are dependent variables.



- The chain rule:

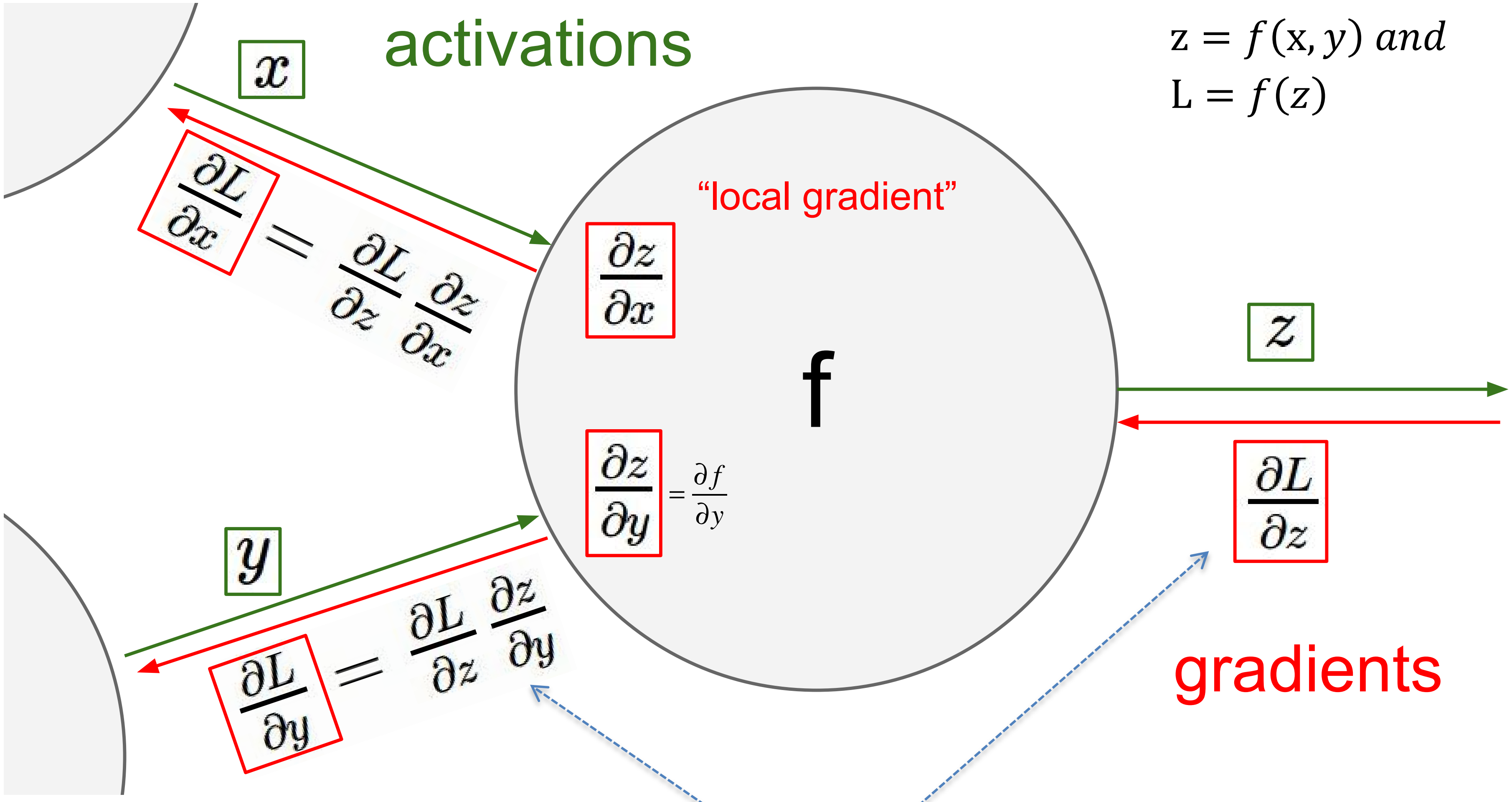
$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x} \cdot \frac{\partial z}{\partial y}$$

- Backpropagation (flow of the gradient)



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

Gradient flow in a computational graph: local junction

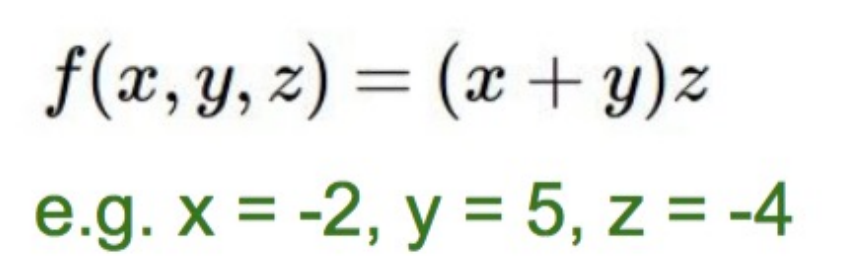

activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$
“local gradient”
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y} = \frac{\partial f}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}$
gradients

$\mathrm{z} = f(\mathrm{x}, y)$ *and*
$\mathrm{L} = f(z)$

is modified by local gradient

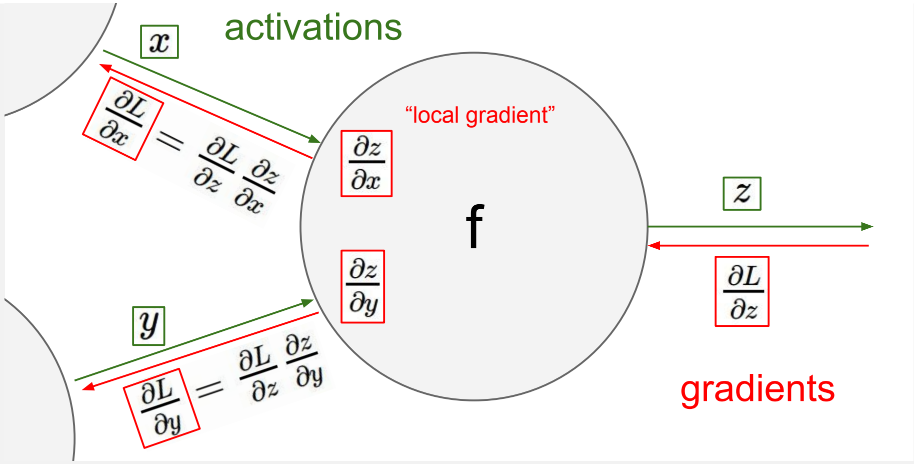
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

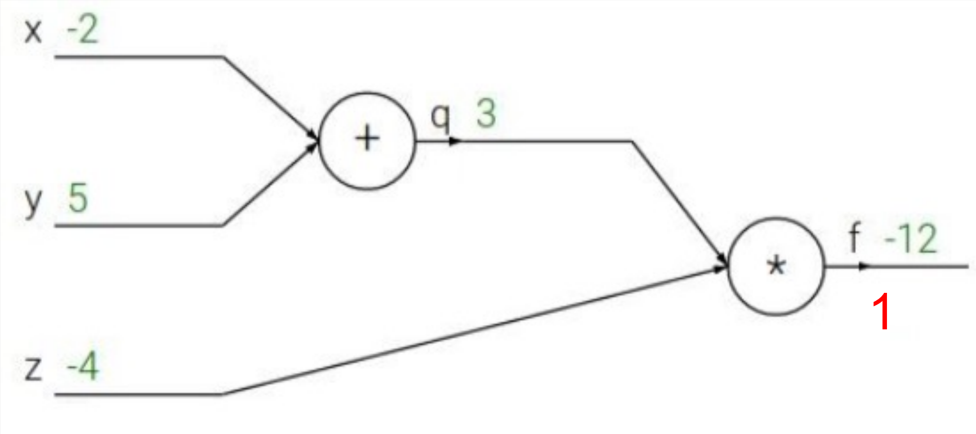
Example



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

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


x
activations
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients



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

Task (10min): Calculate the derivatives. Once by hand, once with backpropagation (follow the graph)

➔Multiplication do a switch

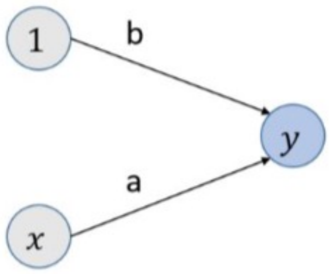
Interlude: Derivations in TF using Tape Mechanism Demonstration if time possible

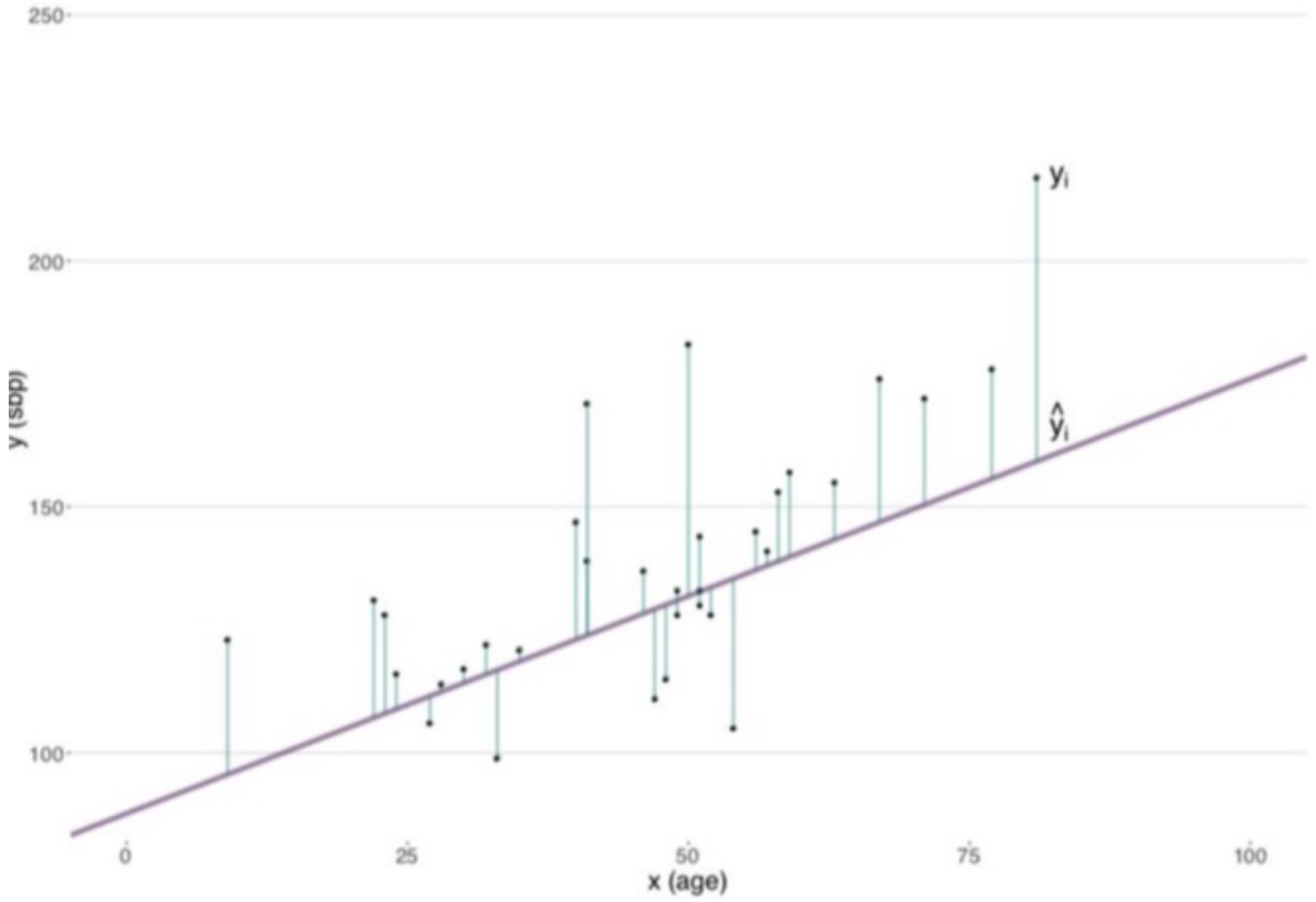
```
x = tf.Variable(-2.)
y = tf.Variable(5.)
z = tf.Variable(-4.)

with tf.GradientTape() as tape: #We need to store
    res = (x+y)*z
    print(tape.gradient(res, [z]))
```

10_a_gradient_tape.ipynb

In depth example: Linear Regression

Example Linear Regression





$$Loss = MSE = 1/n \sum_{i=1}^{n}(y_i - \widehat{y}_i)^{2} = 1/n \sum_{i=1}^{n}(y_i - (a \cdot x_i + b))^{2}$$

Forward and Backwardpass for linear regression 10/11

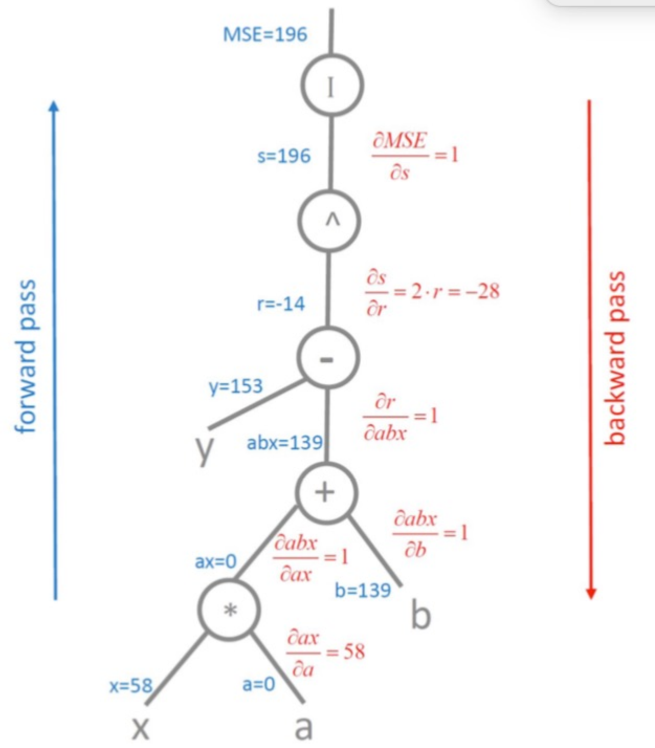

MSE=196
I
s=196
$\frac{\partial MSE}{\partial s}=1$
^
r=-14
$\frac{\partial s}{\partial r}=2\cdot r=-28$
forward pass
backward pass
-
y=153
$\frac{\partial r}{\partial abx}=1$
y
abx=139
+
$\frac{\partial abx}{\partial b}=1$
ax=0
$\frac{\partial abx}{\partial ax}=1$
b=139
b
*
$\frac{\partial ax}{\partial a}=58$
x=58
a=0
x
a

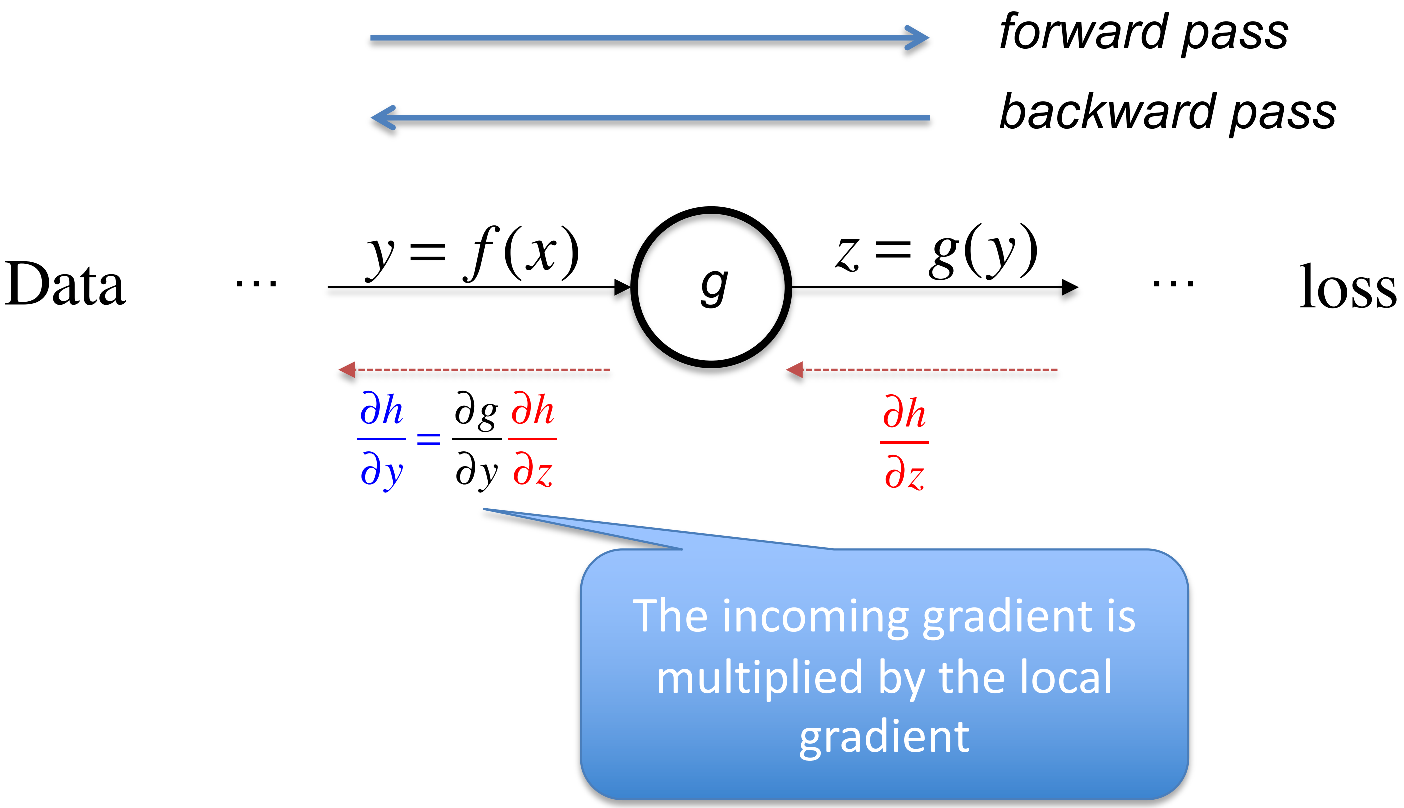
Figure 3.12: The forward and backward pass for the concrete example with one data point (*x* = 58, *y* = 153) and the initial parameter values *a* = 0 and *b* = 139. The flowing values in the forward pass are shown on the left side the graph; the flowing values of the gradients during the backward pass are shown on right side the graph.

Do exercise 10 (works with Tape Gradient)
If you have time you can skim over 11 (Works with the static graph)

Further References / Summary

- For a more in depth treatment have a look at
  - Lecture 4 of http://cs231n.stanford.edu/
  - Slides http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

- Gradient flow is important for learning: remember!

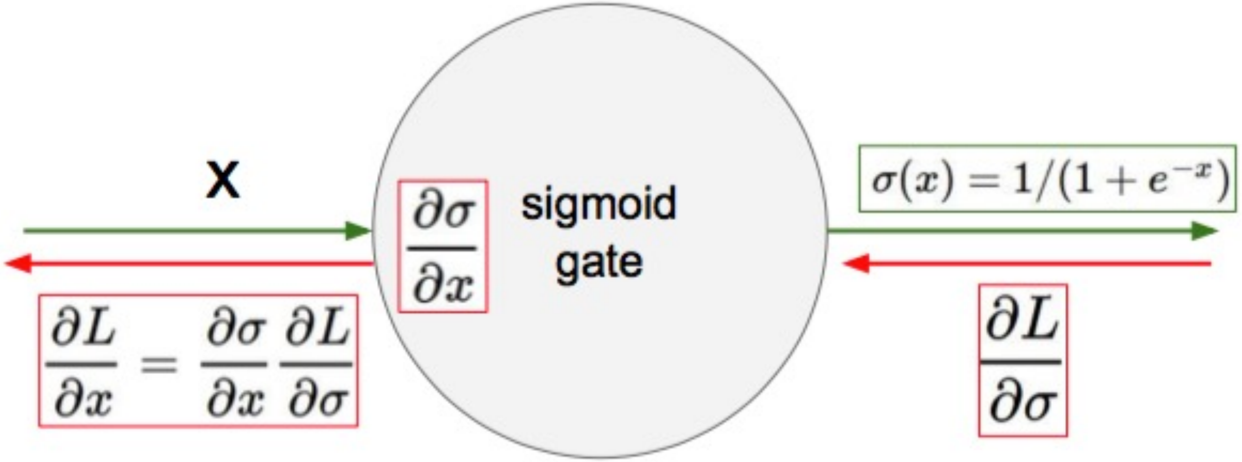


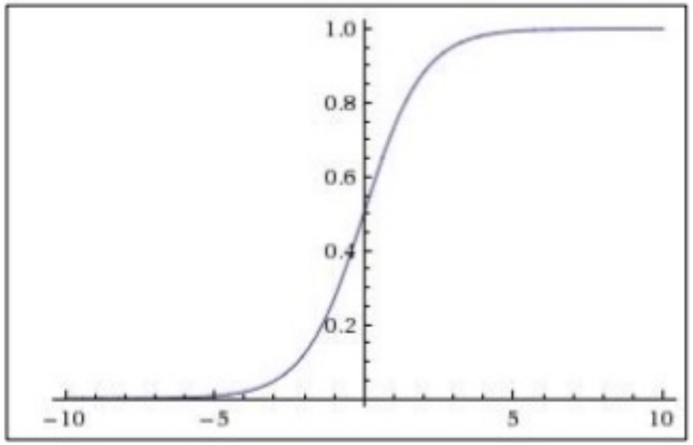
forward pass
backward pass
Data
…
$y = f(x)$
g
$z = g(y)$
…
loss
$\frac{\partial h}{\partial y} = \frac{\partial g}{\partial y}\frac{\partial h}{\partial z}$
$\frac{\partial h}{\partial z}$

The incoming gradient is multiplied by the local gradient

Consequences of Backprop

Backpropagation through sigmoid


x
$\frac{\partial \sigma}{\partial x}$
sigmoid gate
$\sigma(x) = 1/(1+e^{-x})$
$\frac{\partial L}{\partial \sigma}$
$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x}\frac{\partial L}{\partial \sigma}$



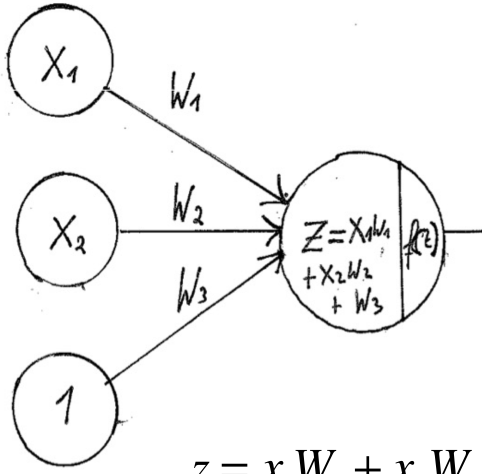
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Gradients are killed, when not in active region! Slow learning!

Slide from: CS231

Different activations in inner layers

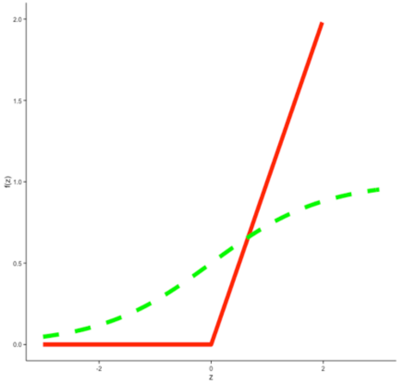
N-D log regression



$$f(z)=\begin{cases}\dfrac{\exp(z)}{1+\exp(z)}\\ \max(0,z)\end{cases}$$

$$z=x_1W_1+x_2W_2+b=Wx+b$$

Activation function a.k.a.
Nonlinearity f(z)



Motivation:
Green:
logistic regression.
Red:
ReLU faster
convergence

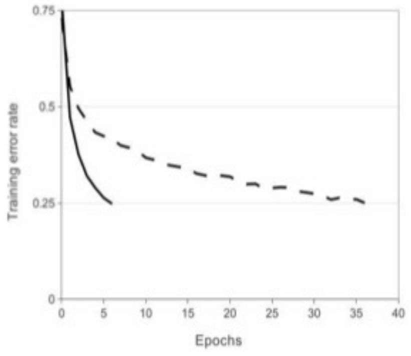


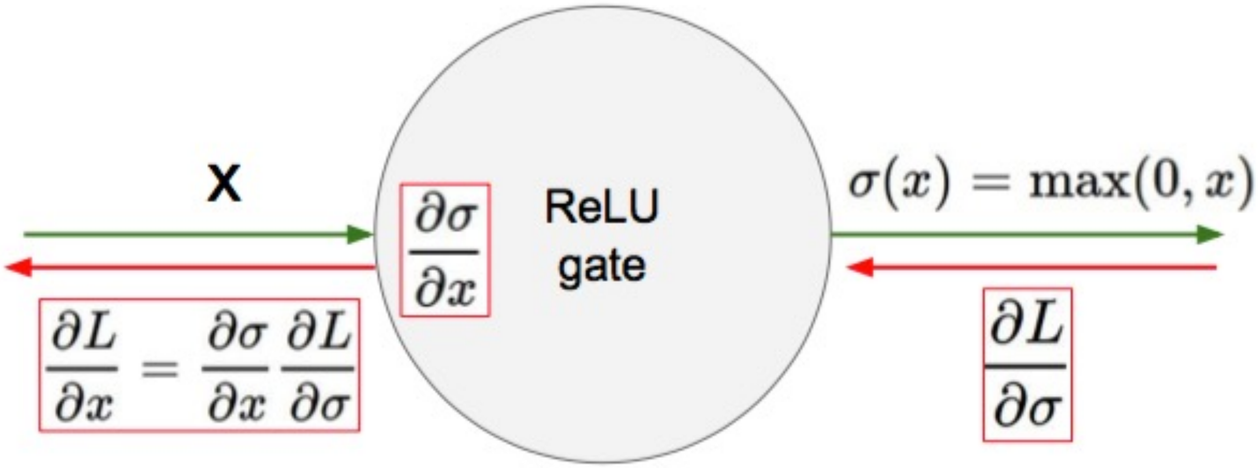
Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

Source:
Alexnet
Krizhevsky et al 2012

There are other alternatives besides sigmoid and ReLU.

Currently ReLU is standard

Backpropagation through ReLU



x

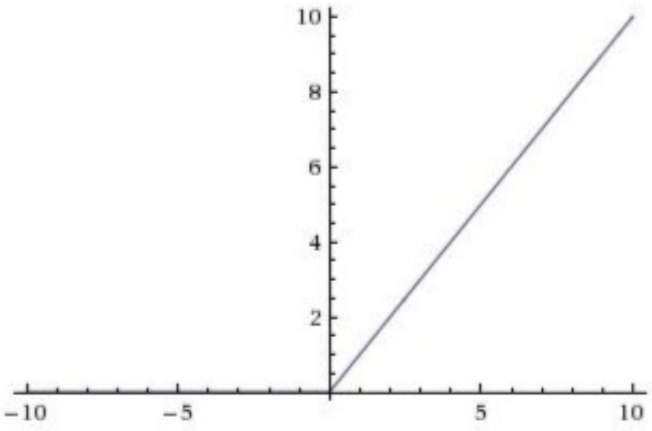
$\frac{\partial \sigma}{\partial x}$

ReLU gate

$\sigma(x) = \max(0, x)$

$\frac{\partial L}{\partial \sigma}$

$\frac{\partial L}{\partial x} = \frac{\partial \sigma}{\partial x} \frac{\partial L}{\partial \sigma}$



What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Gradients are killed, only when x < 0

Slide from: CS231

Recap: Batch Normalization is beneficial in many NN

After BN the input to the activation function is in the sweet spot

Observed distributions of signal after BN before going into the activation layer.

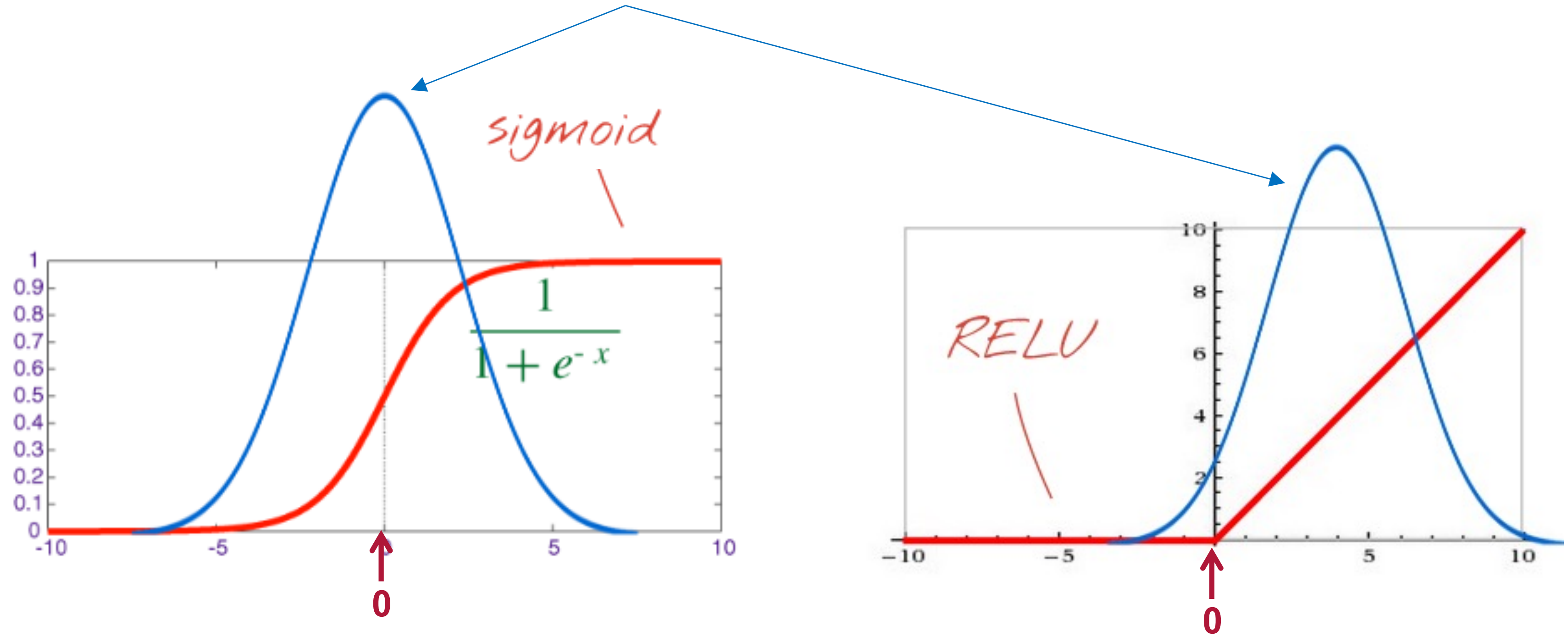


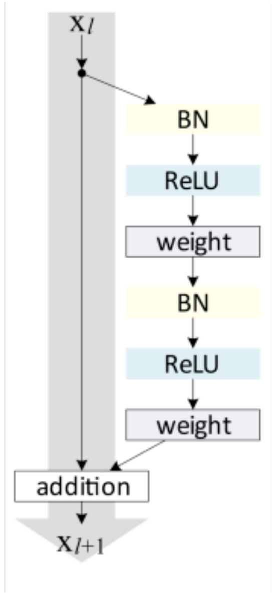
Image credits: Martin Gorner: https://docs.google.com/presentation/d/e/2PACX-1vRouwj_3cYsmLrNNI3Uq5gv5-hYp_QFdeoan2GlxKgIZRSejozruAbVV0IMXBoPsINB7Jw92vJo2EAM/pub?slide=id.g187d73109b_1_2921

"ResNet" from Microsoft 2015 winner of imageNet

152 layers

ResNet basic design (VGG-style)

- add shortcut connections every two
- all 3x3 conv (almost)



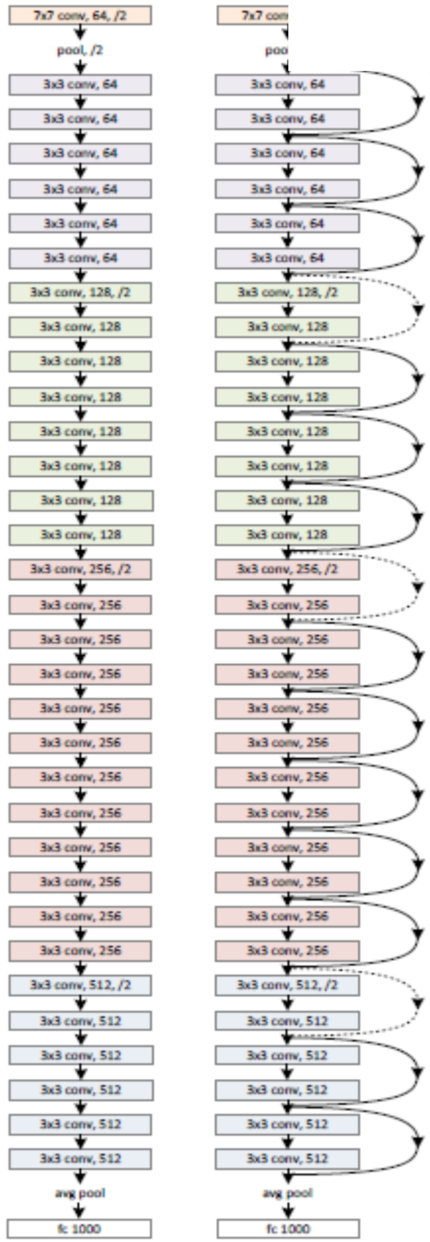
$F(x_l)$

$H(x_l)=x_{l+1} = x_l + F(x_l)$

152 layers:
Why does this train at all?

This deep architecture could still be trained, since the gradients can skip layers which diminish the gradient!

F(x) is called "residual" since it only learns the "delta" which is needed to add to x to get H(x)

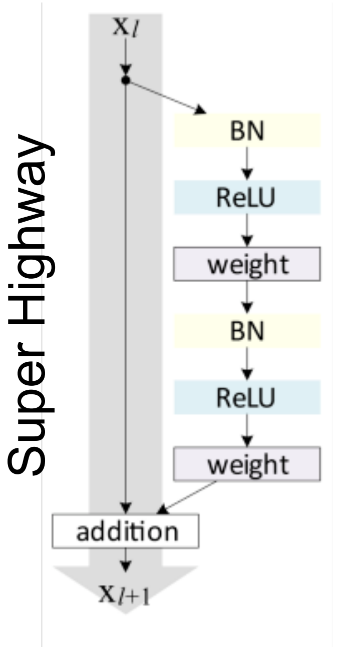


plain VGG

ResNet

credits: https://chatbotslife.com/resnets-highwaynets-and-densenets-oh-my-9bb15918ee32

Closer Look



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

➔'Gradient Super Highways'

What comes in (on the right) does go out (on the left)

Similar to LTMS (just in case you know)

Building Loss Functions with Maximum Likelihood

Simple regression via a NN: no probabilistic model in mind

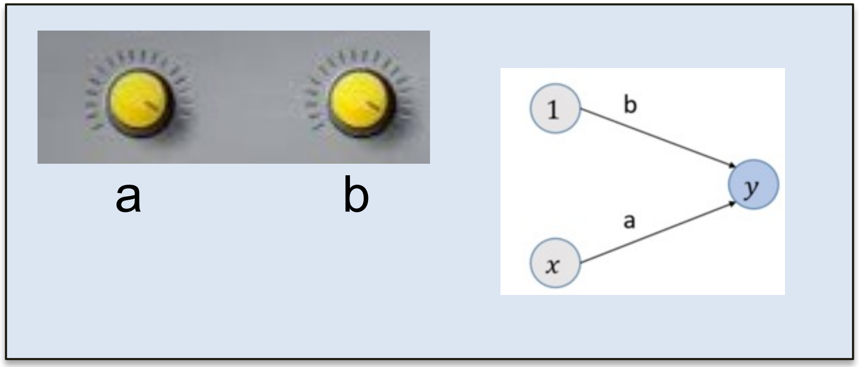
age

Input x

Systolic blood pressure

Output y

$x = 35$ →

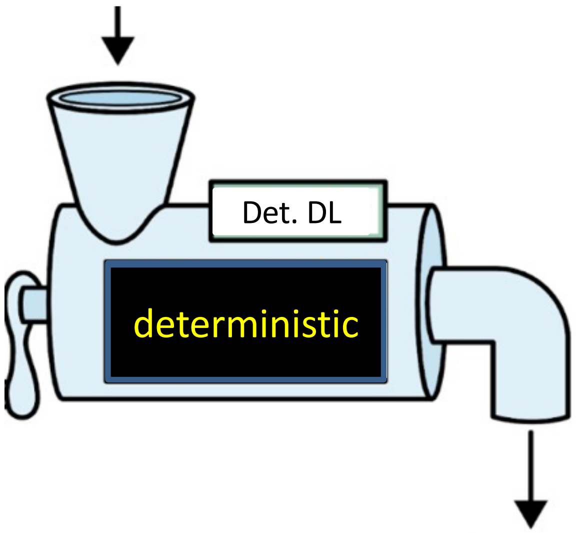


→ $s\hat{b}p = 117$

One input x (age) → one predicted outcome (sbp)

Traditional versus probabilistic regression DL models

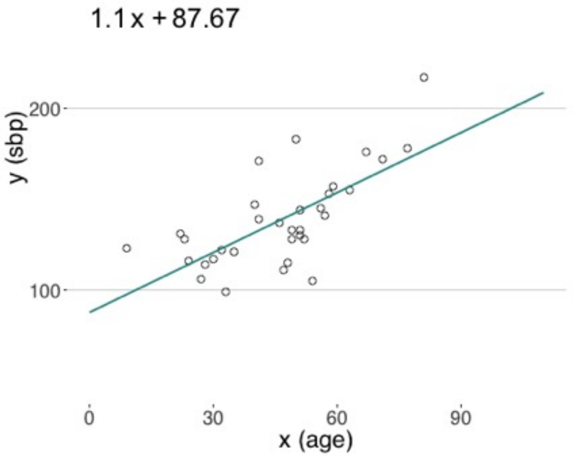
X=35



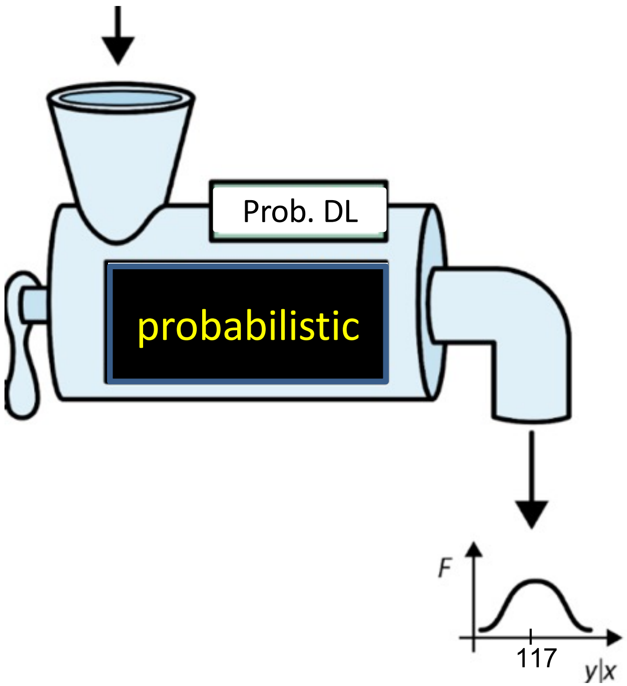
Det. DL

deterministic

Y=117

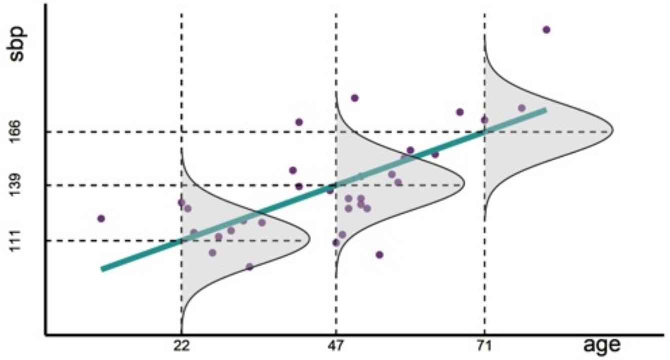


X=35



Prob. DL

probabilistic



Describes the spread of the data

Recap Classification: Softmax Activation

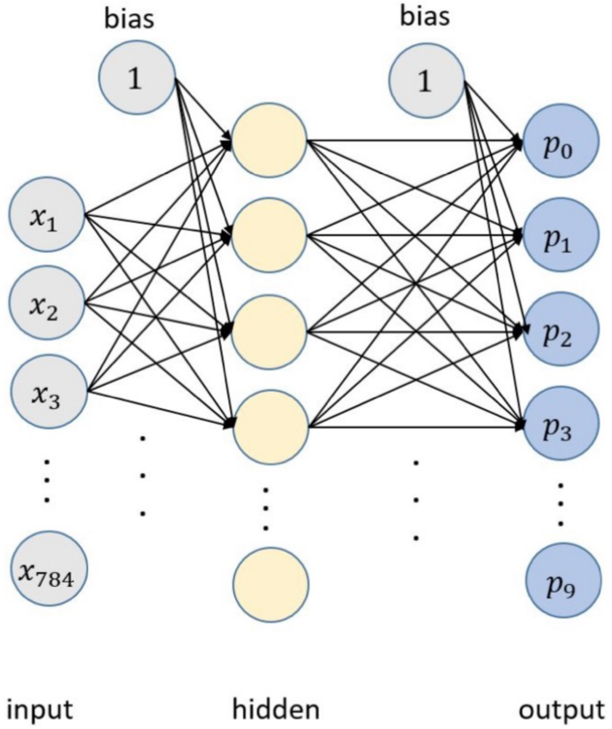


Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

$p_o, p_1 \ldots p_9$ are probabilities for the classes 0 to 9.

Activation of last layer z_i incomming

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

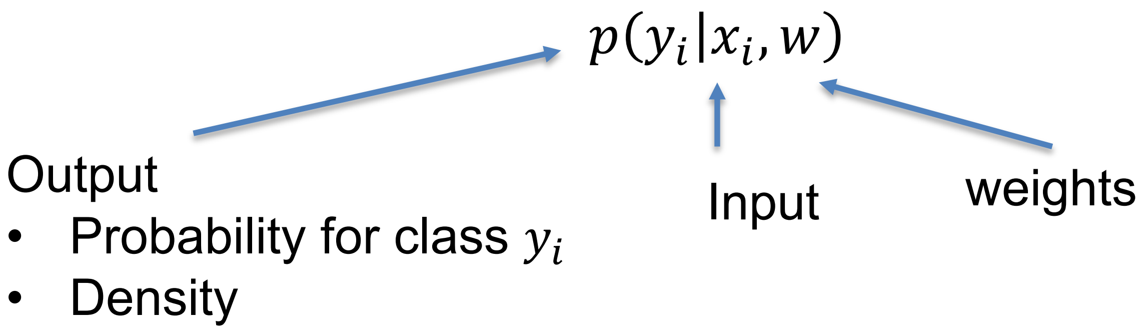
Makes outcome positive

Ensures that pi's sum up to one

This activation is called softmax

Neural networks are probabilistic models

- The output of a neural network, can be understood as a probability
  - Classification
    - Probability of class 1…,K
  - Regression
    - Probability density
- Output of a neural network for training example i



$$p(y_i|x_i,w)$$

Output

- Probability for class y_i
- Density

Input

weights

Maximum Likelihood (a universal loss for DL)

Tune the parameters weights of the network, so that observed data (training data) is most likely.

Practically: Minimize Negative Log-Likelihood of the CPD

$$\hat{w} = argmin \sum_{i=i}^{N} -\log(p(y_i|x_i, w))$$

Maximum Likelihood (one of the most beautiful ideas in statistics)

Ronald Fisher in 1913
Also used before by
Gauss, Laplace

$$M(\theta) \xrightarrow[\text{(often known)}]{\text{Likelihood / “probability“}} \text{Data}$$

Tune the parameter(s) θ of the model M so that (observed) data is most likely

What‘s the likelihood of the data for lin. regression...

Motivating Example of MaxLike

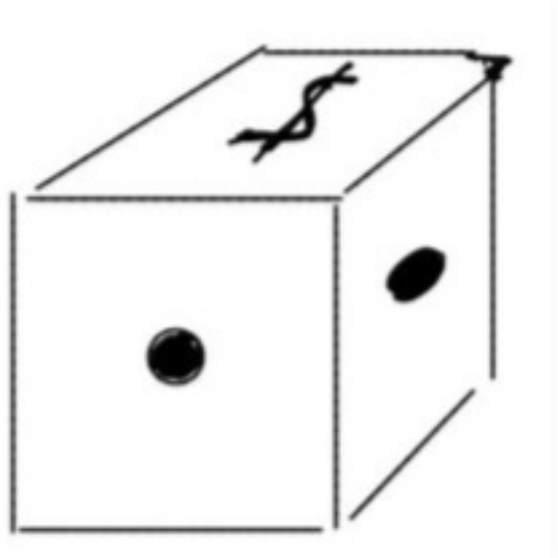


Figure 4.2 A die with one side showing a dollar sign and the others a dot.

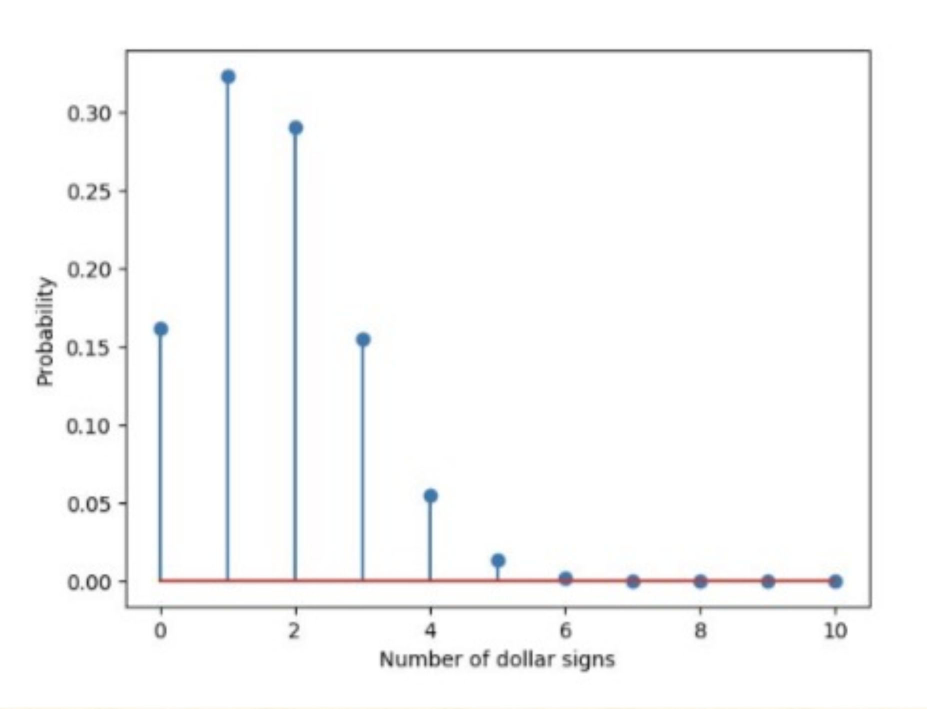
Question: What is probability to see one $-signs in 10 throws?

A see Blackboard:

k ~ binom(n=10, p = 1/6)

Solution

```
from scipy.stats import binom
ndollar = np.asarray(np.linspace(0,10,11), dtype='int')
pdollar_sign = binom.pmf(k=ndollar, n=10, p=1/6)
plt.stem(ndollar, pdollar_sign)
plt.xlabel('Number of dollar signs’)
plt.ylabel('Probability')
```



Max-likelihood

Now you don’t know how many dollar signs are on the die.

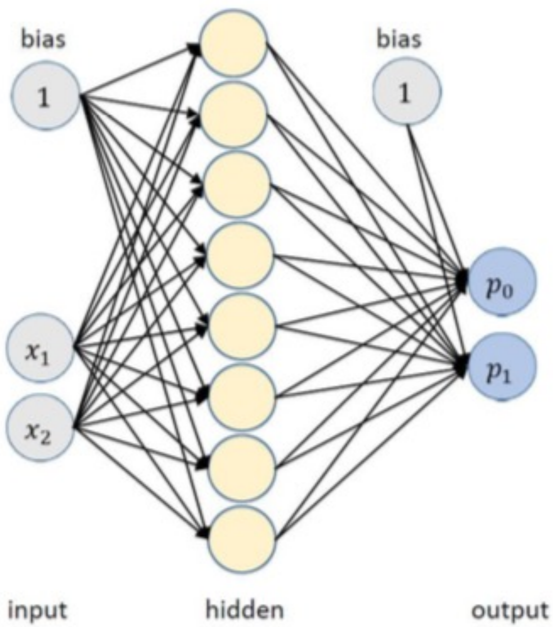
You throw the die 10 times and get k=2 dollar signs.

What is you best guess?

Work Through Exercise

https://github.com/tensorchiefs/dl_book/blob/master/chapter_04/nb_ch04_01.ipynb

ML principle for binary classification



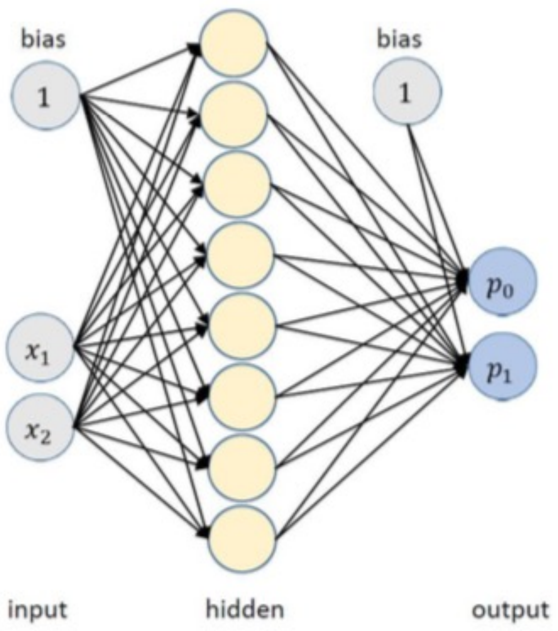
x_i, y_i Training data $i = 1, \dots N$

$p_0(x_i)$ is probability for $y_i = 0$

$p_1(x_i)$ is probability for $y_i = 1$

Question:
What is probability for the training set of say 5 examples? The first 3 are of class 0, last two 2 of class 1?

ML principle for binary classification



x_i, y_i Training data $i = 1, \dots N$

$p_0(x_i)$ is probability for $y_i = 0$

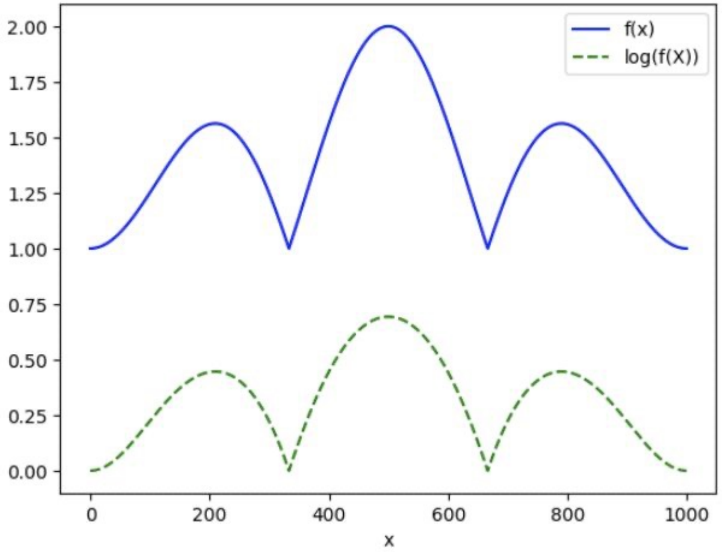
$p_0(x_i)$ is probability for $y_i = 1$

Answer:
What is probability for the training set of say 5 examples? The first 3 are of class 0, last two 2 of class 1?

$$Pr(Training) = p_0(x_1) \cdot p_0(x_2) \cdot p_0(x_3) \cdot p_1(x_4) \cdot p_1(x_5) = \prod_{j=1}^{3} p_0(x_j) \cdot \prod_{j=4}^{5} p_1(x_j)$$

Likelihood → Log Likelihood

- Many Data Points
  - Many data points $i = 1, \ldots N$ → product of the individual contributions
  - Taking log
    - Likelihood $\prod p_i$ -> Log-Likelihood $\log(\prod p_i) \rightarrow \sum \log(\mathrm{p_i})$
    - Does not take change position of maximum



  - Often taken negative Log-Likelihood → Minimizing..

NLL=crossentropy = $-\frac{1}{n}\sum \log(p(y_i|x_i)$

Negative Log-Likelihood (NLL)

- Likelihood of training data

$$Pr(Training) = \prod_{j\ for\ with\ y=0} p_0(x_j) \cdot \prod_{j\ for\ with\ y=1} p_1(x_j)$$

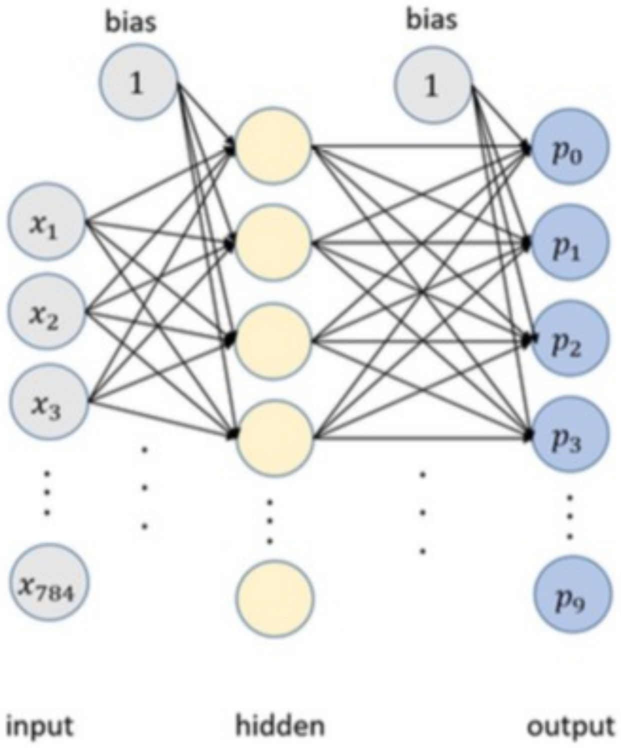
- LogLike

$$\log(Pr(Training)) = \sum_{j\ for\ y=0} \log(p_0(x_j)) + \sum_{j\ for\ y=1} \log(p_1(x_j))$$

- Crossentropy / NNL negative log likelihood (per example divided by n)

$$crossentropy = -\tfrac{1}{n}\left(\sum_{j\ for\ y=0} \log(p_0(x_j)) + \sum_{j\ for\ y=1} \log(p_1(x_j))\right)$$

More than 2 classes



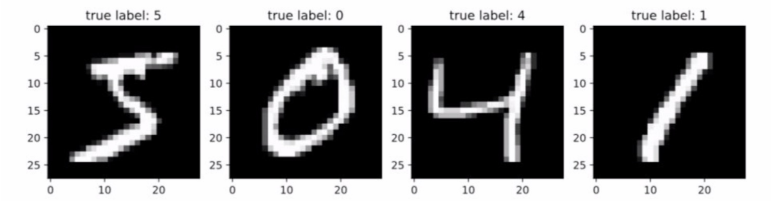


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

$$crossentropy = -\frac{1}{n}\Big(\sum_{j\,for\,y=0}\log(p_0(x_j)) + \sum_{j\,for\,y=1}\log(p_1(x_j)) + \ldots + \sum_{j\,for\,y=K-1}\log(p_{k-1}(x_j))\Big))$$

NLL=crossentropy = $-\frac{1}{n}\sum\log(p(y_i|x_i)$

Excercise

12b_mnist_loglike

Tasks in the NB:

- Load MNIST and make a small CNN **without** training and calculate the loss.
- Calculate (with calculator or numpy) the excepted cross-entropy for the MNIST example if you just guess, each class with p=1/10.