Hochschule Konstanz
Fakultät Informatik

Oliver Dürr

Short course on deep learning
Block 4

Southern Taiwan University
of Science and Technology

This course is a short version of dl_course_2022 I created with Beate Sick.

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

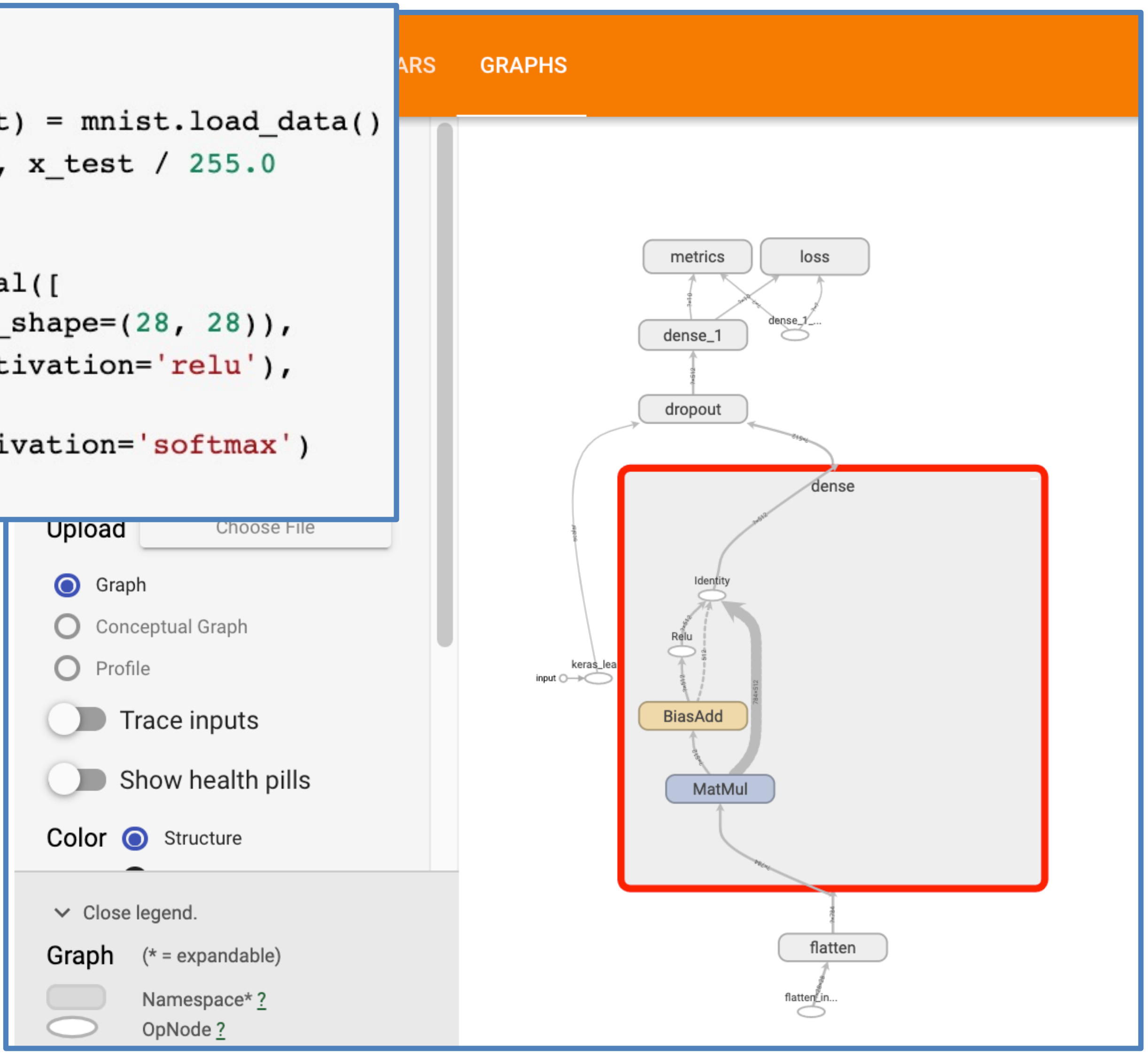
Keras

```python
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')
  ])
```

TensorFlow



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

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

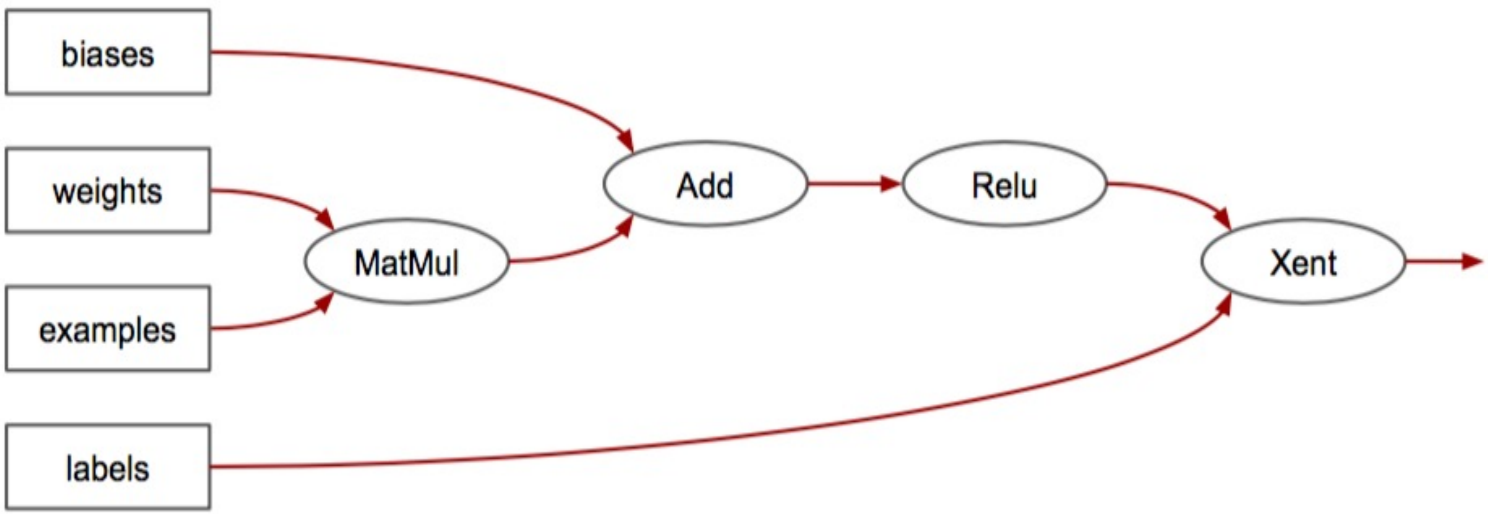
Next steps

- Understand the computational graph (theoretical)

- Understand backpropagation in a graph (theoretical)

Recap

- The computation in TF is done via a computational graph

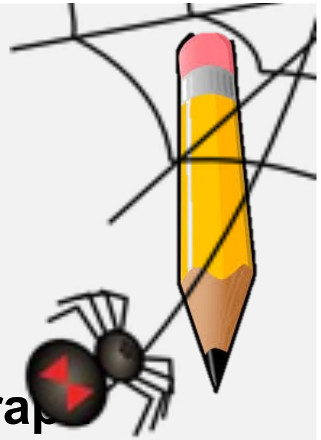


- The nodes are ops
- The edges are the flowing tensors

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



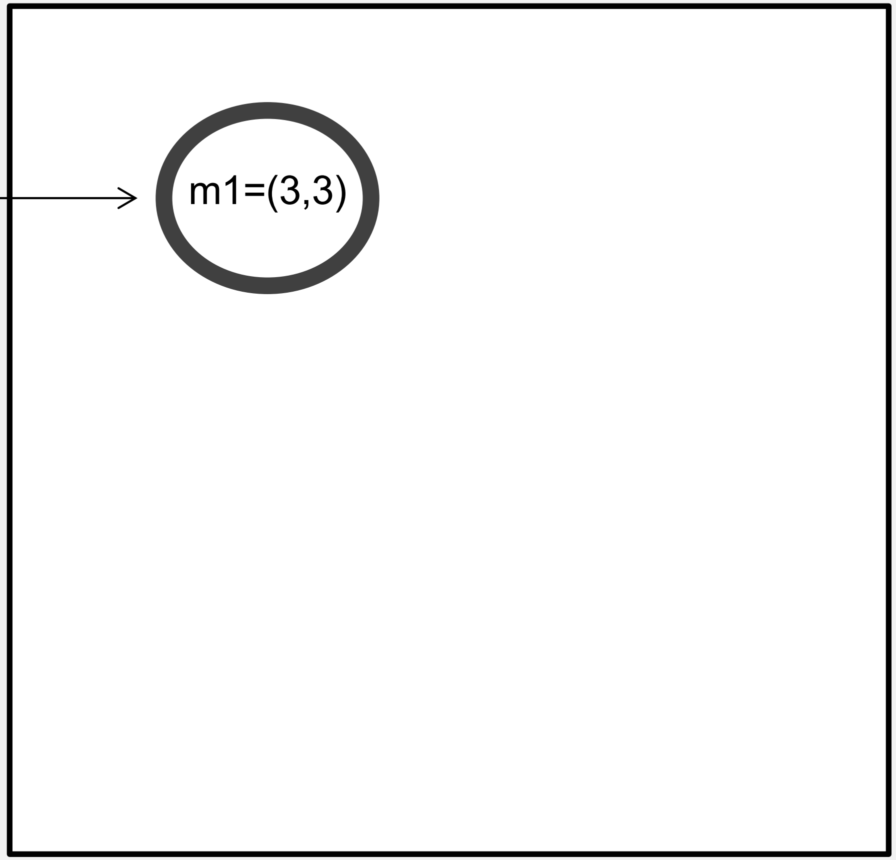
Translate the following TF code in a graph

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)
```

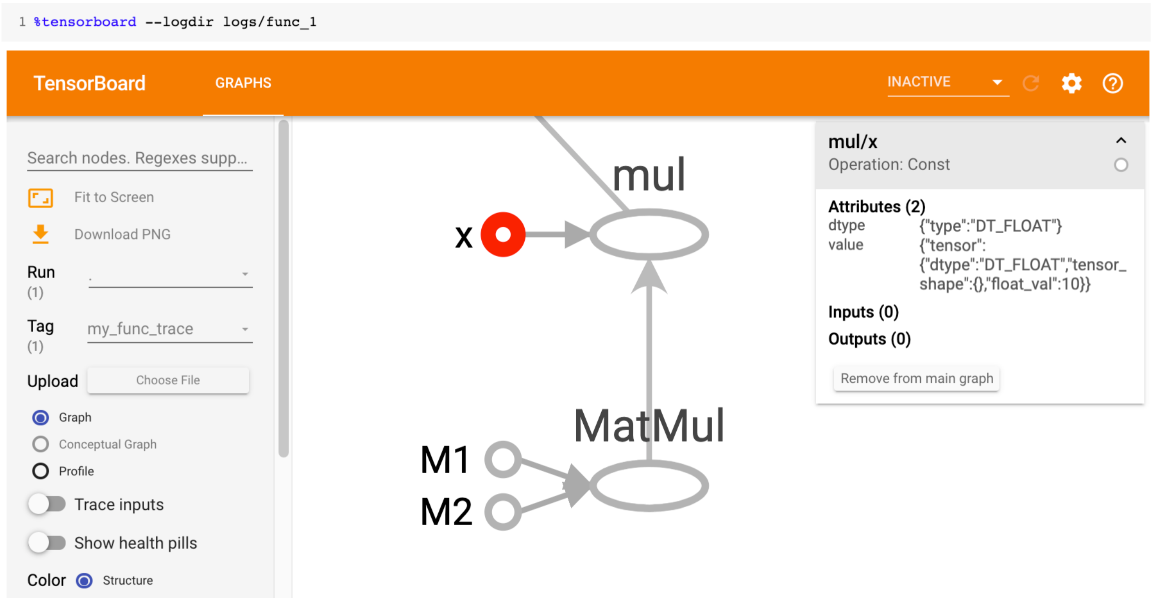
Quite much happen in here!

Finish the computation graph



TensorFlows internal representation

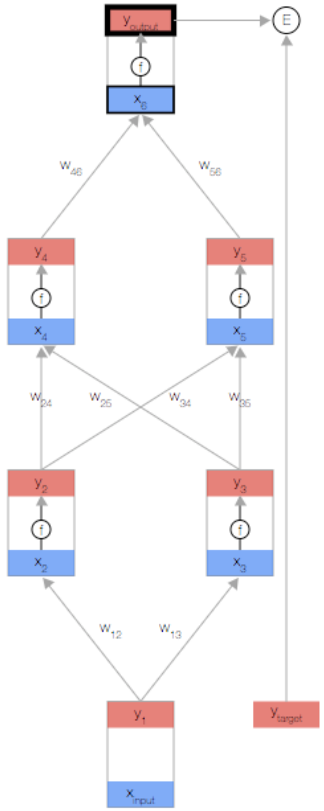
- For fast computation a graph is build
  - Technical detail in tf 2.0 you need to decorate a function with @tf.function to build a graph. Otherwise eager execution happens.



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.

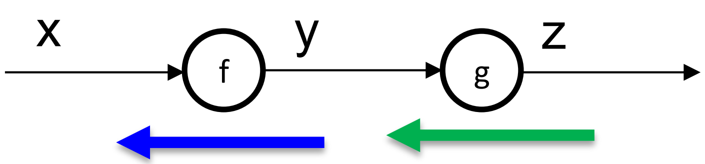
(The backward pass is described in more details in the next following slides).

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

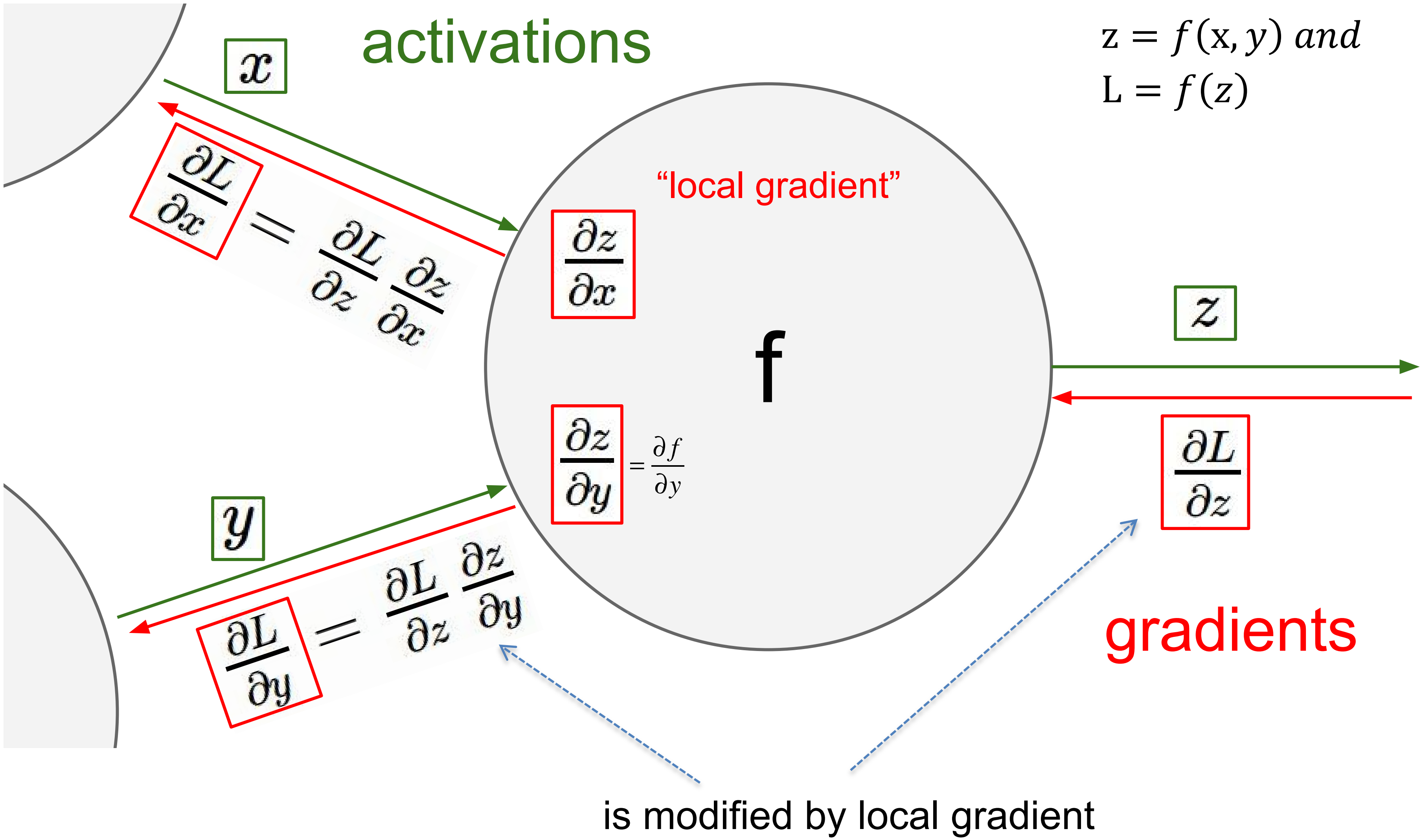
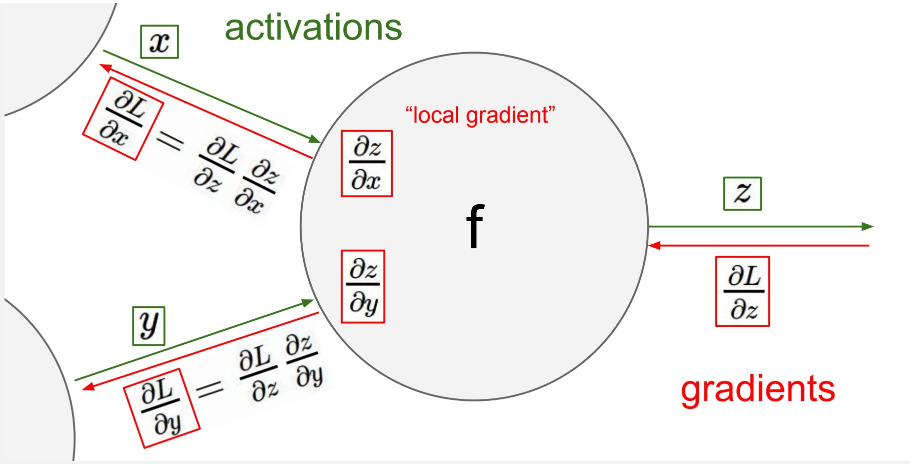


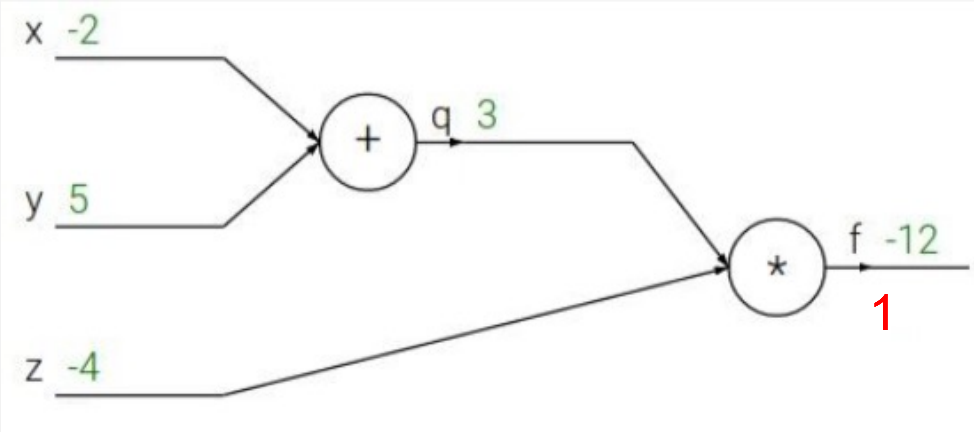
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Example



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

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



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

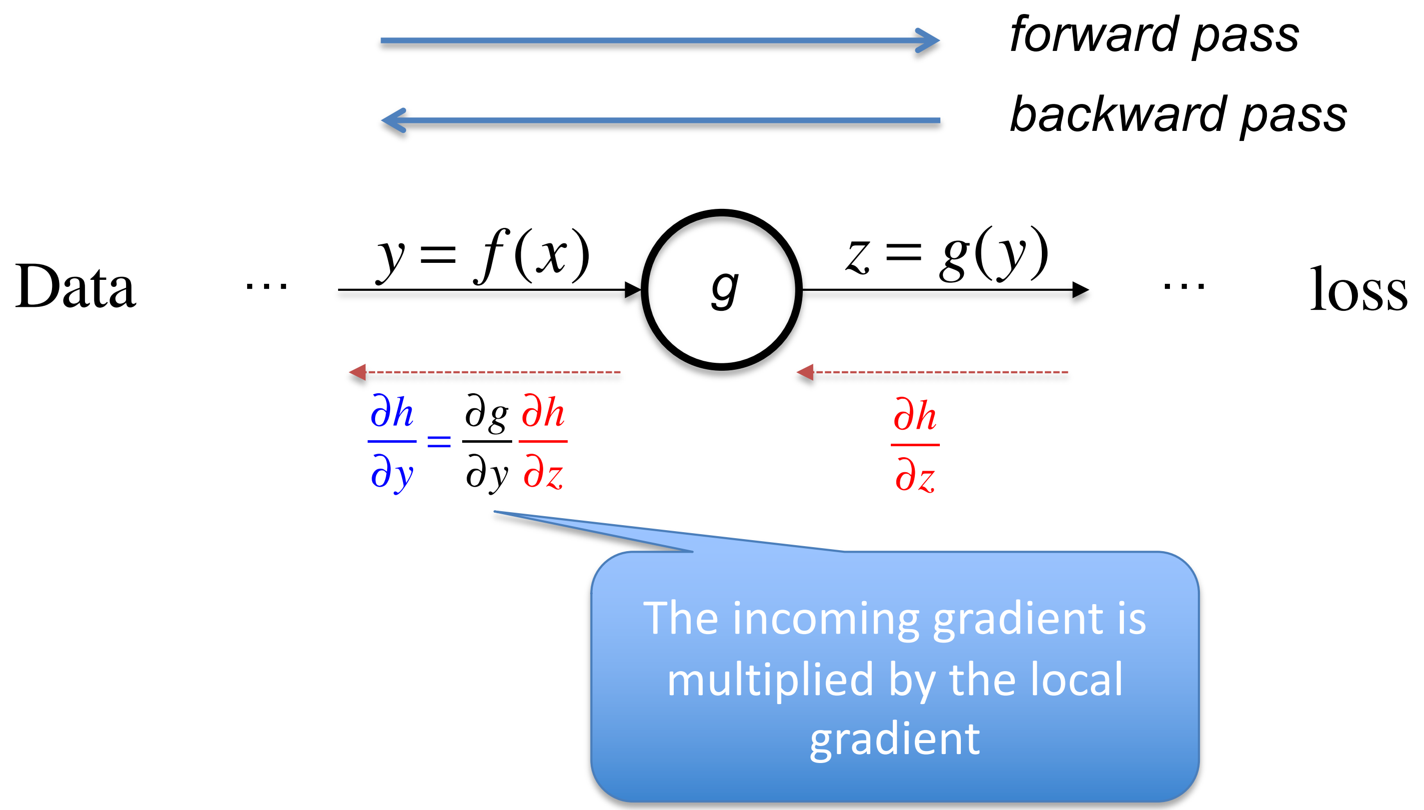
Task (10min): Calculate the derivatives. Once by hand, once with backpropagation (follow the graph)

→Multiplication do a switch

Further References / Summary

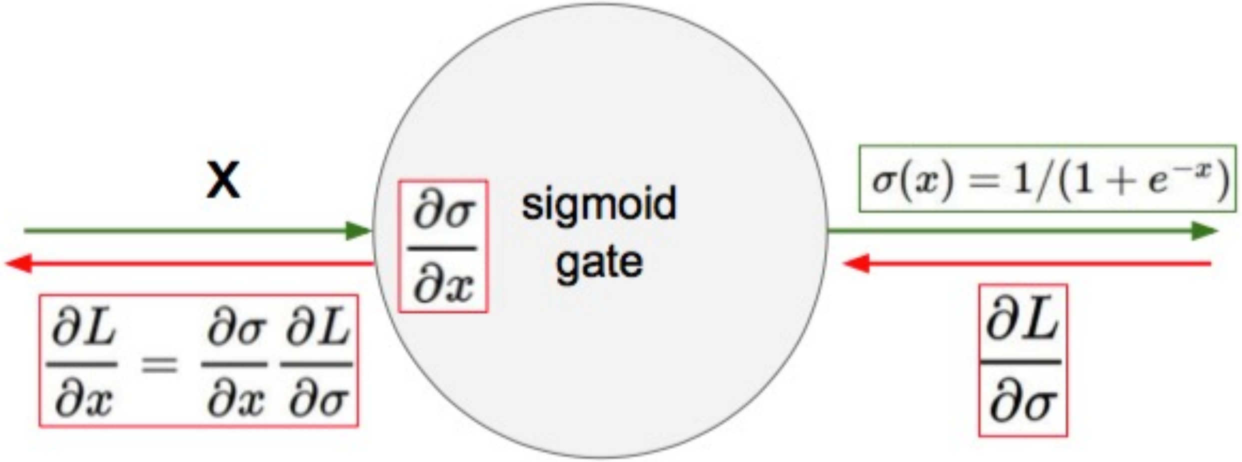
- For a more in depth treatment have a look at
  - Lecture 4 of http://cs231n.stanford.edu/
  - Slides http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

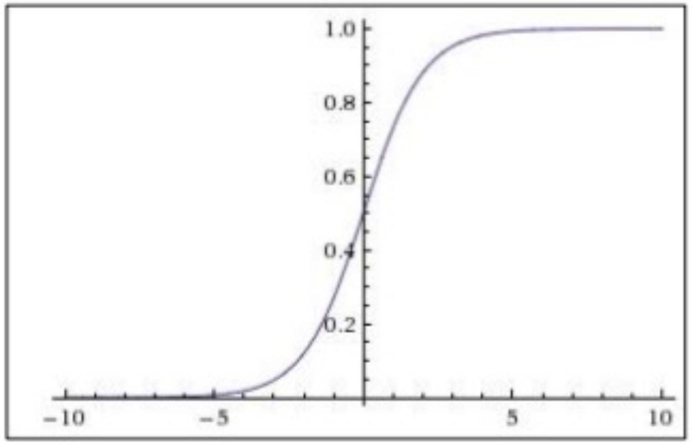
- Gradient flow is important for learning: remember!



Consequences of Backprop

Backpropagation through sigmoid





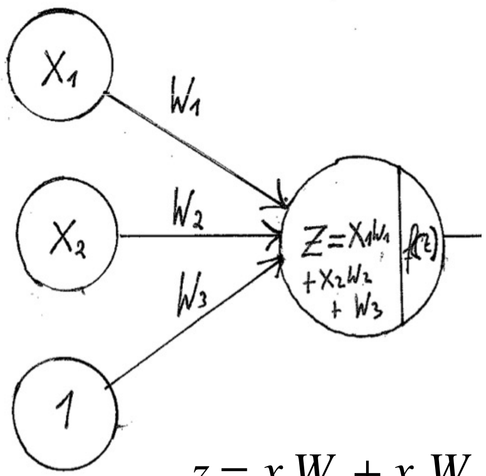
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

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

Slide from: CS231

Different activations in inner layers

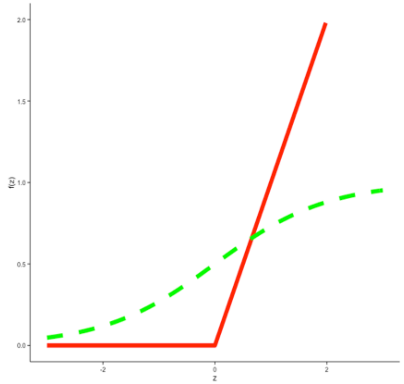
N-D log regression



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

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

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



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

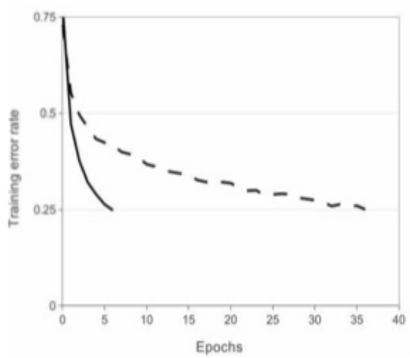


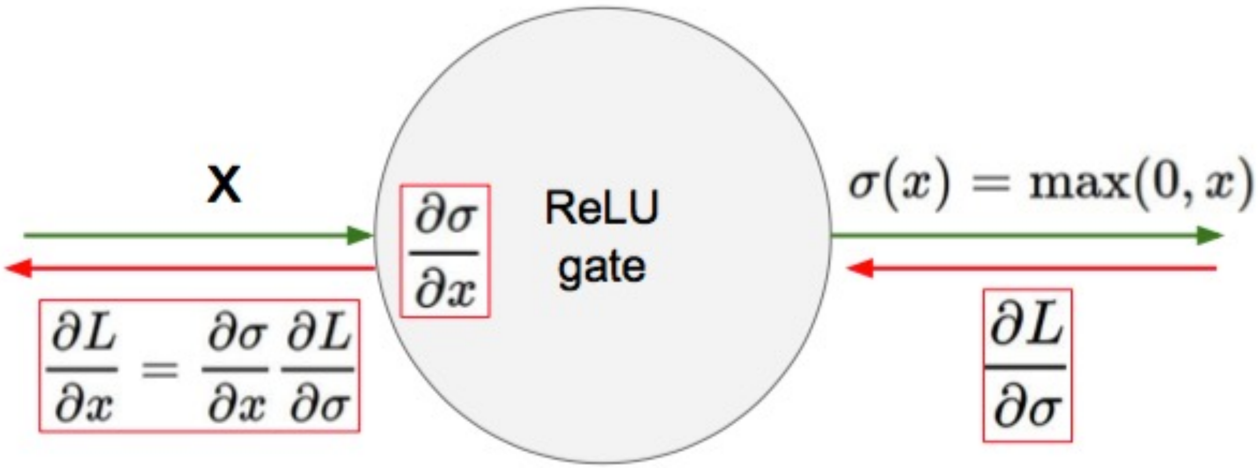
Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

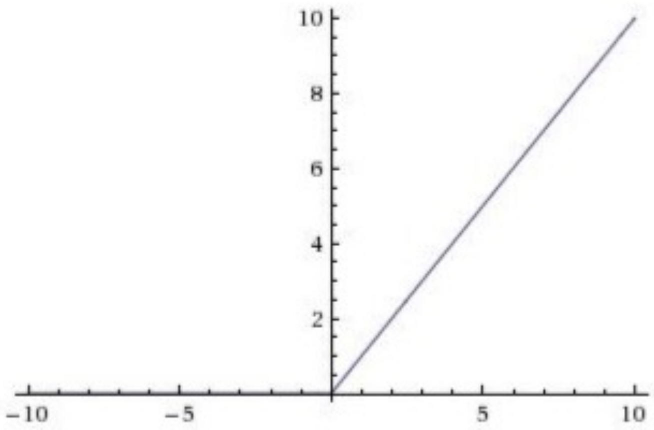
Source:
Alexnet
Krizhevsky et al 2012

There are other alternatives besides sigmoid and ReLU.

Currently ReLU is standard

Backpropagation through ReLU





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

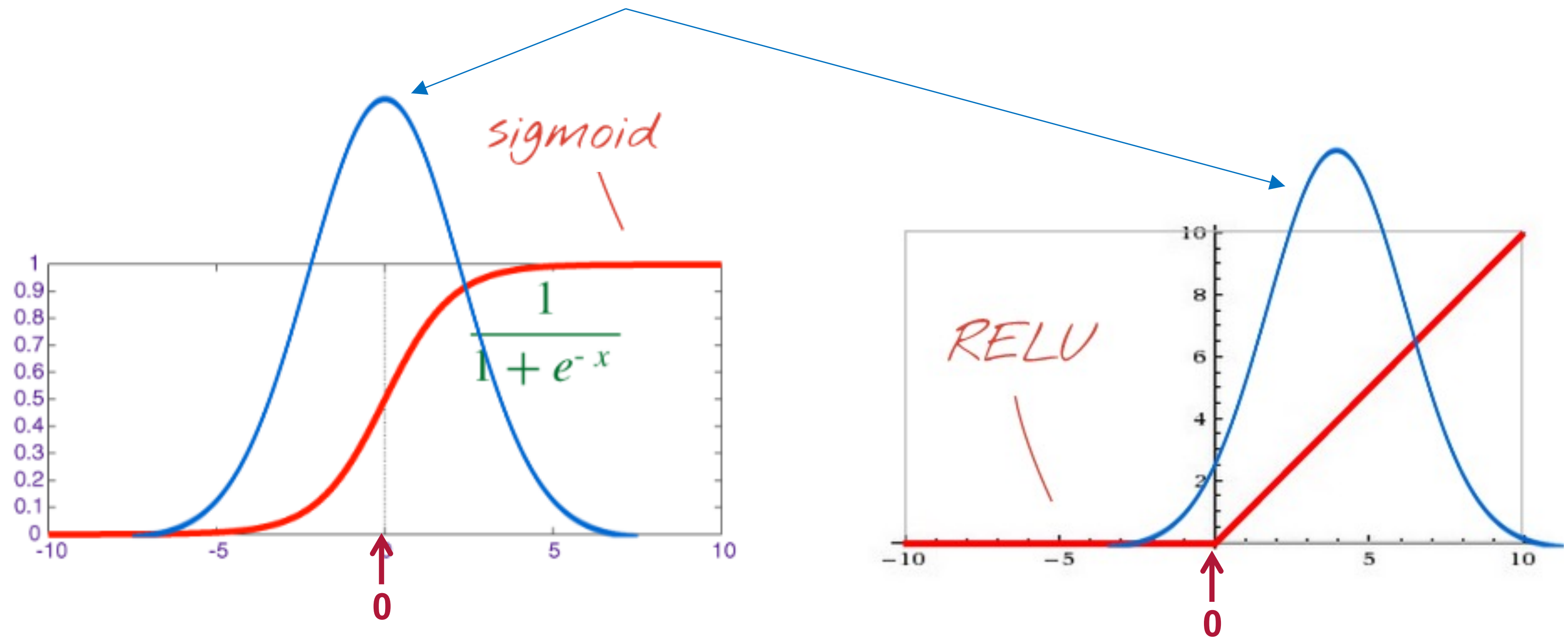
Gradients are killed, only when x < 0

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.



When using BN consider the following:

- Using a higher learning rate might work better
- Use less regularization, e.g. reduce dropout probability
- In the linear transformation the biases can be dropped (step 2 takes care of the shift)
- In case of ReLu only the shift β in steps 2 need to be learned (α can be dropped)

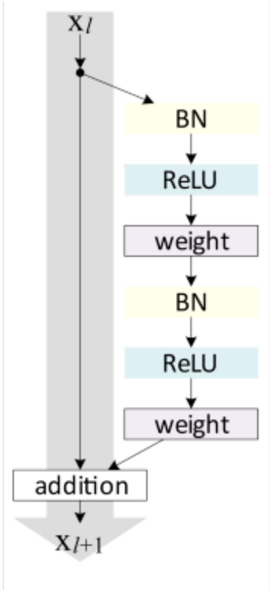
Image credits: Martin Gorner: https://docs.google.com/presentation/d/e/2PACX-1vRouwj_3cYsmLrNNI3Uq5gv5-hYp_QFdeoan2GlxKgIZRSejozruAbVV0IMXBoPsINB7Jw92vJo2EAM/pub?slide=id.g187d73109b_1_2921

"ResNet" from Microsoft 2015 winner of imageNet

152
layers

ResNet basic design (VGG-style)

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



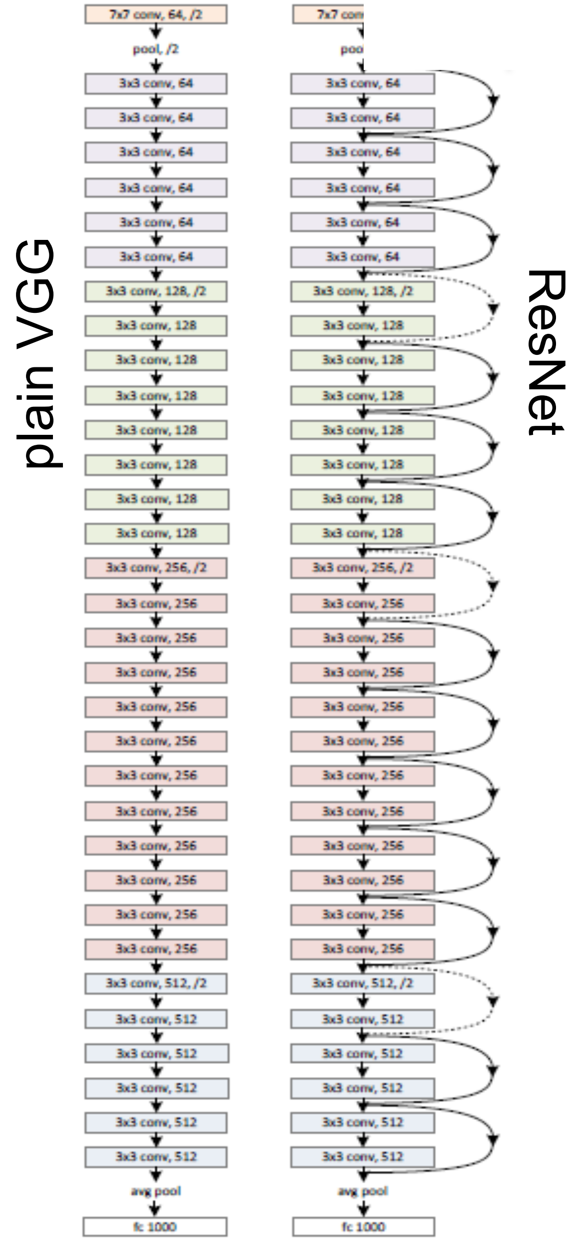
$F(x_l)$

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

152 layers:
Why does this train at all?

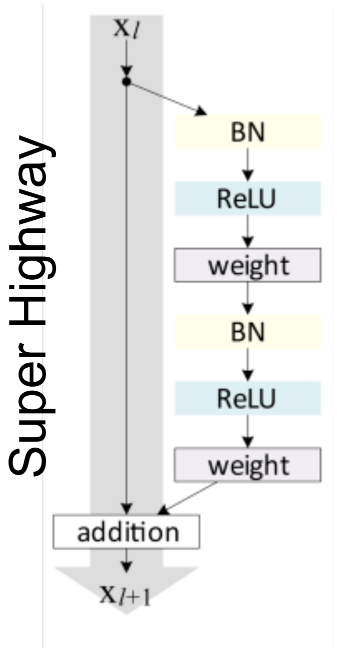
This deep architecture could still be trained, since the gradients can skip layers which diminish the gradient!

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



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

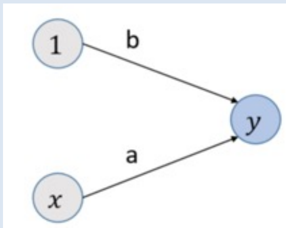
Input x

Systolic blood pressure

Output y

$$x = 35 \rightarrow$$

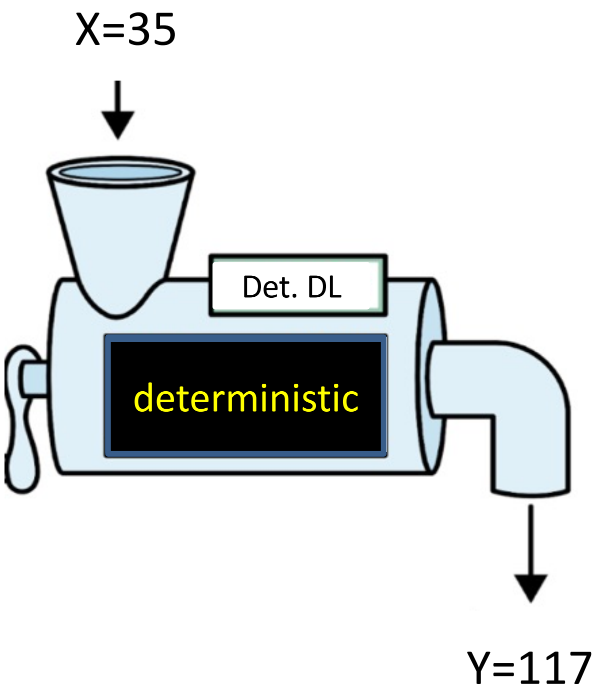
a b

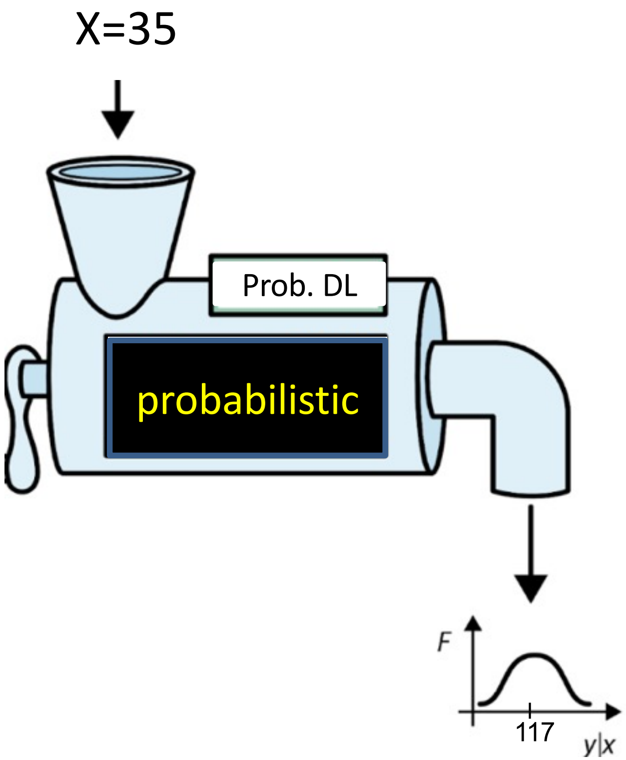


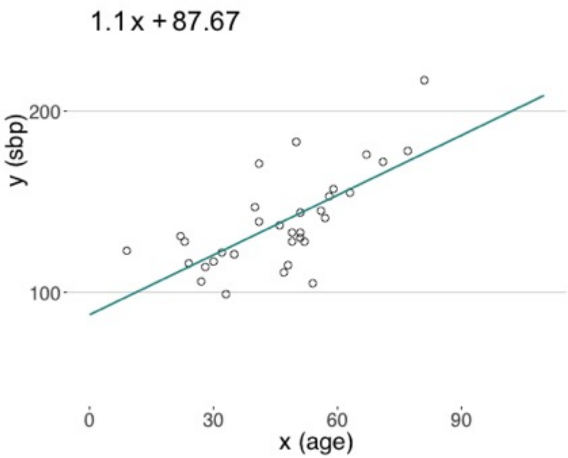
$$\rightarrow \hat{\text{sbp}} = 117$$

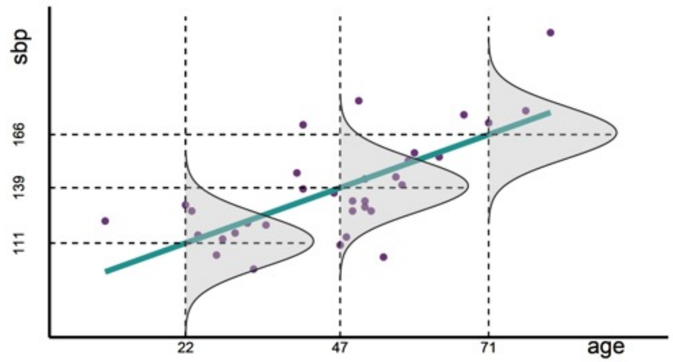
One input x (age) → one predicted outcome (sbp)

Traditional versus probabilistic regression DL models









Describes the spread of the data

Recap Classification: Softmax Activation

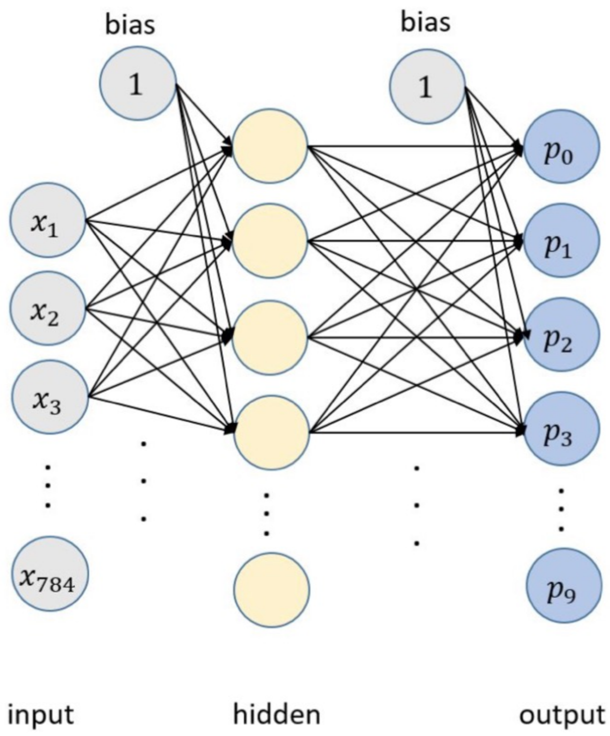


Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

$p_o, p_1 \ldots p_9$ 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 Distribution
- Output of a neural network for training example i

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

Output
- Probability for class y_i
- Distribution

Input

weights

*More on probabilistic interpretation next lecture

Maximum Likelihood

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

Likelihood / “probability“
(often known)

$$M(\theta) \longrightarrow \text{Data}$$

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

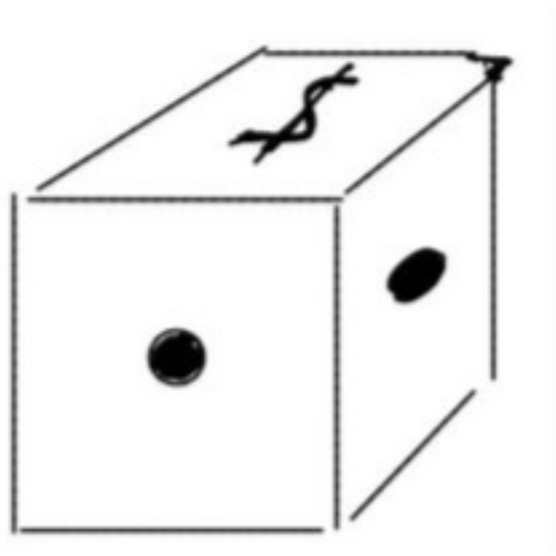


Figure 4.2 A die with one side showing a dollar sign and the others a dot.

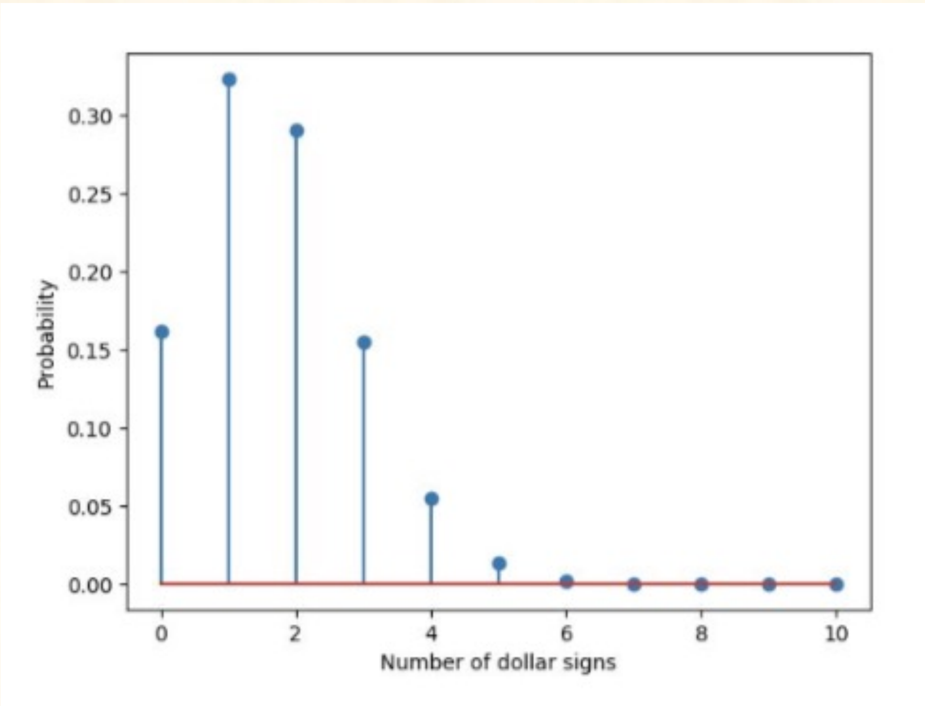
Question: What is probability to see one $-signs in 10 throws?

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')
```



Exercise 12_maxlike

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.

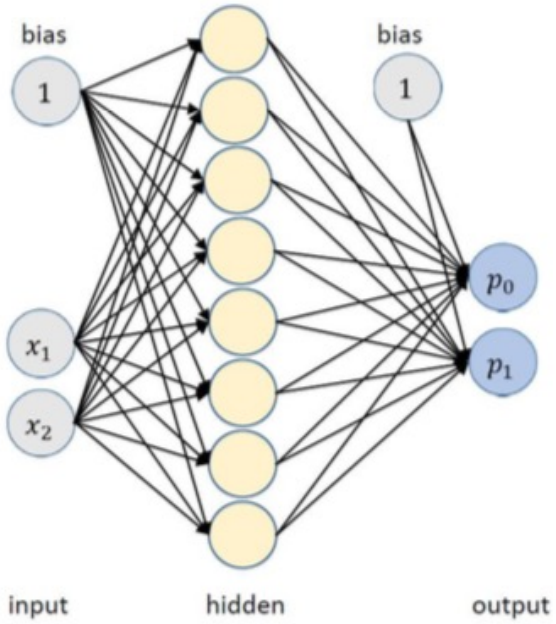
What is you best guess?

Work Through Exercise:

Work through the code until you reach the first exercise. In the exercise it is your task to determine the probability to observe two-times a dollar sign in ten dice throws, if you consider a die that has dollar signs on 0, 1, 2, 3, 4, 5, or all 6 faces.

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

ML principle for binary classification



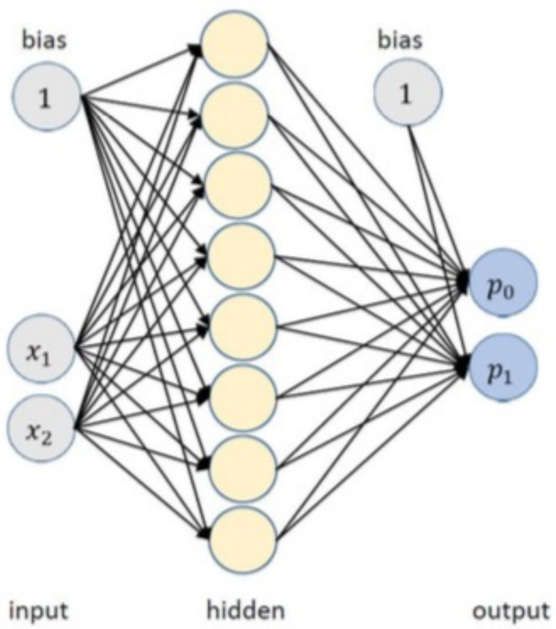
x_i, y_i Training data $i = 1, \ldots N$

$p_0(x_i)$ 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 $y_i = 0$

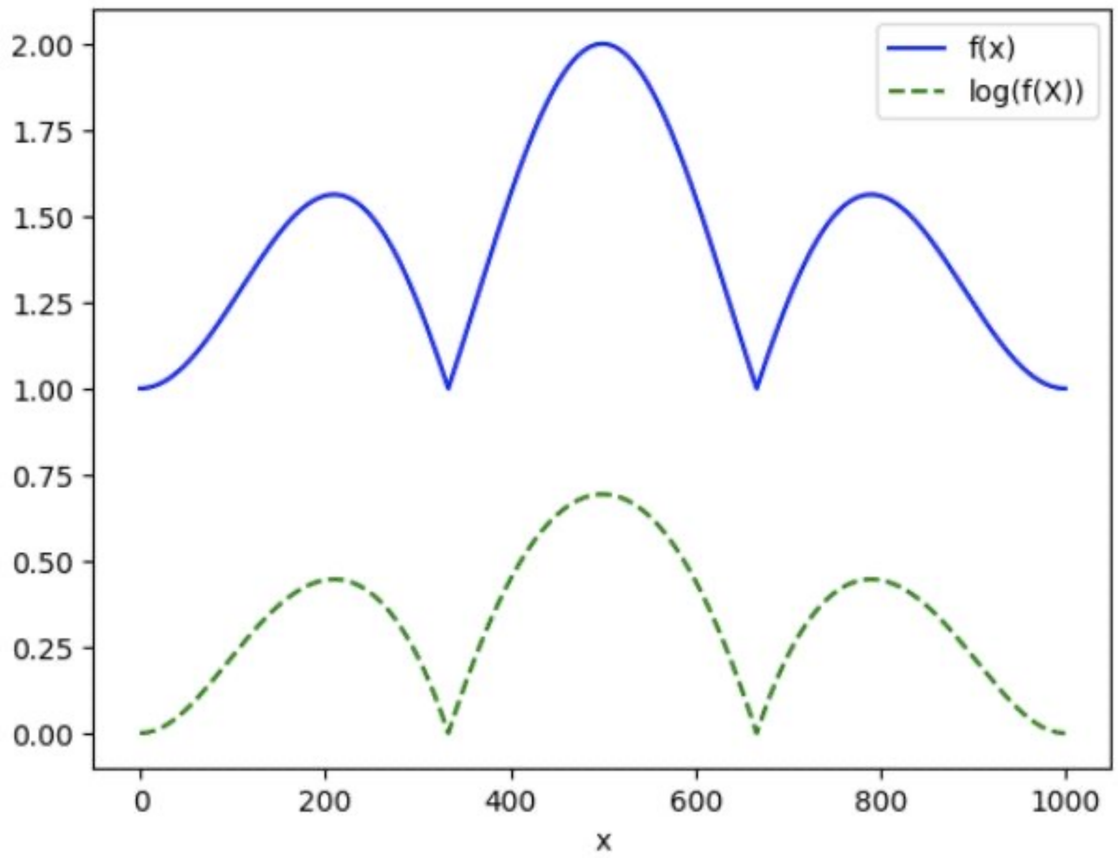
$p_0(x_i)$ is probability for $y_i = 1$

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?

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

Taking the log



To determine the maximal value, taking log is also ok.

Negative Log-Likelihood (NLL)

- Likelihood of training data

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

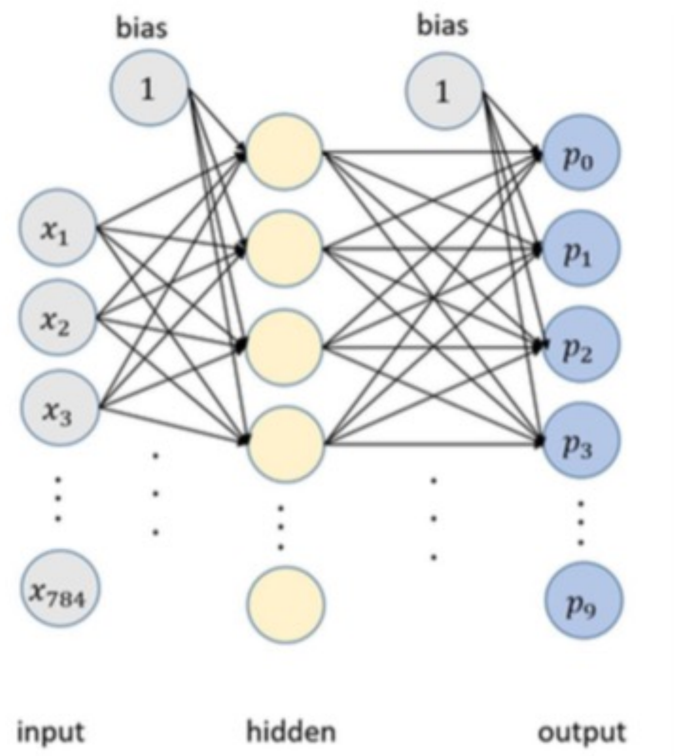
- LogLike

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

- 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



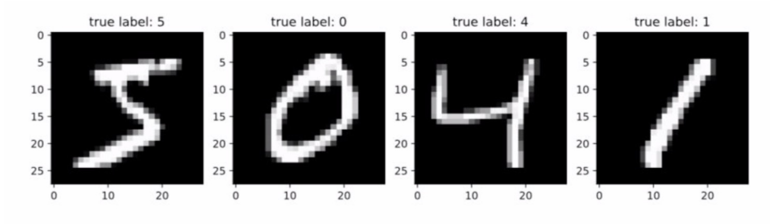


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

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

$$crossentropy = -\frac{1}{n}\sum_{i=1}^{n} {}^{true}p_i \cdot log(p_i)$$