Machine Intelligence:: Deep Learning
Week 3

Oliver Dürr

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

Winterthur, 5. March. 2019

Organizational Issues: Projects

- Projects (2-3 People)

- Presented on the last day
  - Spotlight talk (5 Minutes)
  - Poster
- Topics
  - You can choose a topic of your own (have to be discussed with us latest by ~~week4~~ week5)
  - Possible Topics
    - Take part in a Kaggle Competition (e.g. Leaf Classification / Dogs vs. Cats)
    - Overview of google ml learning cloud for deep learning
    - Datasets e.g. http://www.vision.ee.ethz.ch/en/datasets/

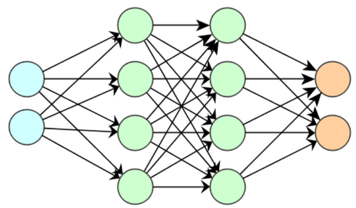
- Please talk to us until week ~~4~~ → 5
- Q&A Session 1h in week 7

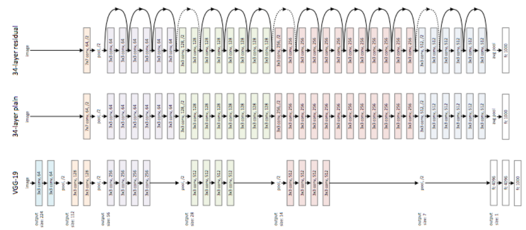
Organizational Issues: Times

- Next times (total 30 minutes break in between, possible different breaks)
  - 09:10 – 10:30 (We start 10 past 9)
  - 11:00 – 12:40

- Please interrupt us if something is unclear!

Learning Objectives





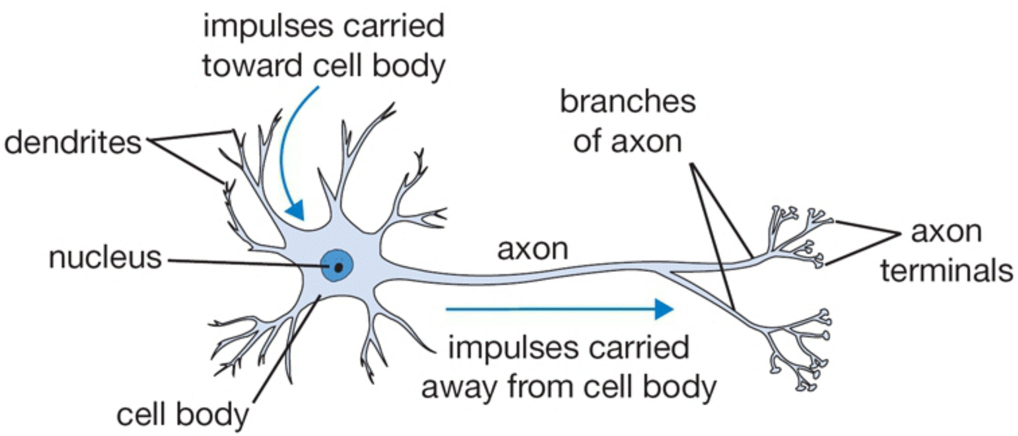
- Increase our knowledge in TF

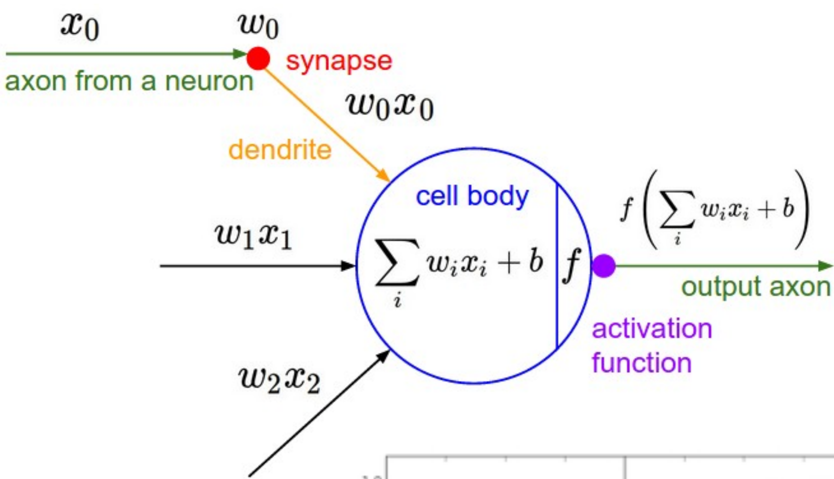
- Foundations of DL
  - **Loss Function (what to minimize)**
    - Cross entropy loss for multinomial logistic regression
    - Two principles to construct loss functions
      - Maximum Likelihood Principle
      - Cross Entropy
  - Deep Neural Networks
    - Fully Connected Networks with hidden layers

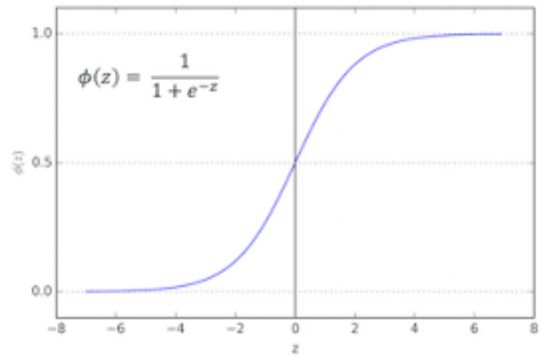
  - Gradient Descent
    - How to calculate the weights efficiently

Biological Interpretation

- In popular media neural networks are often described as a computer model of the human brain.







DL *loosely inspired* by how the brain works.
Biological neurons are much more complicated.

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

Multinomial Logistic Regression

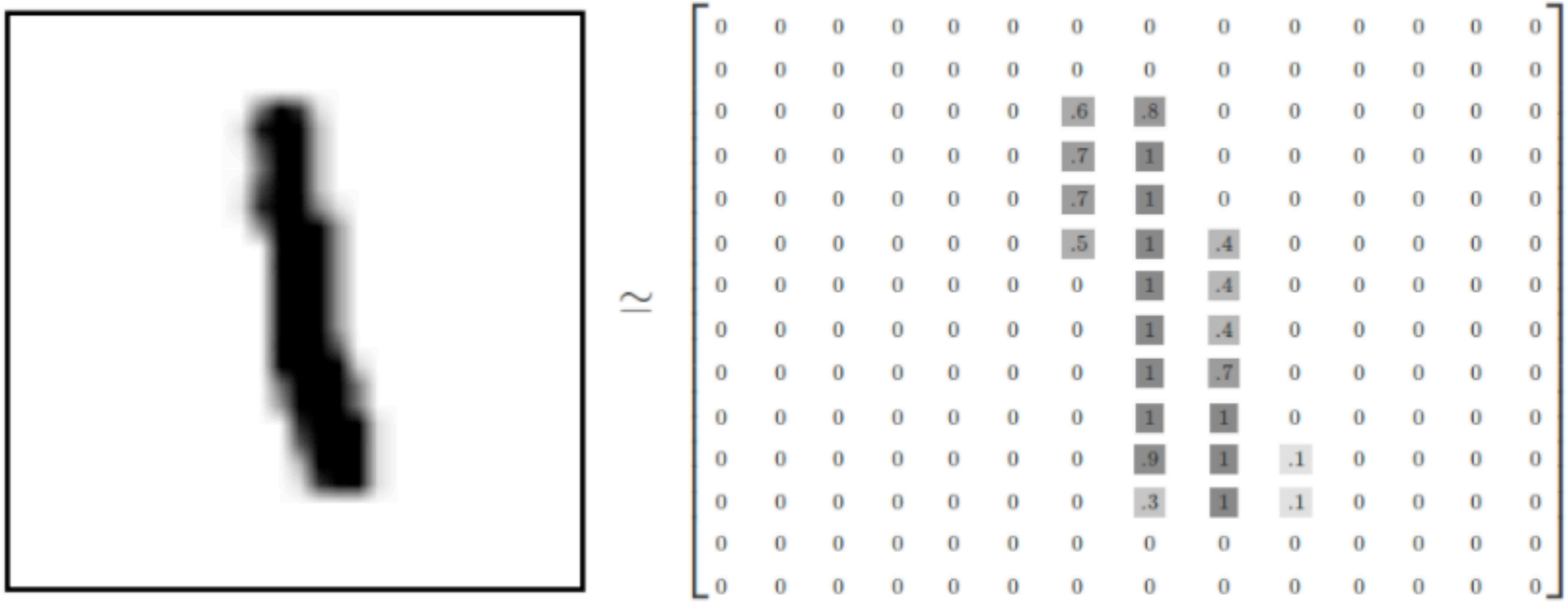
Multinomial logistic regression

- Logistic Regression outputs prob. for class 1
  - So far we can classify into two classes

- We now want to classify more than 2 classes

Exercise: The MNIST Data Set

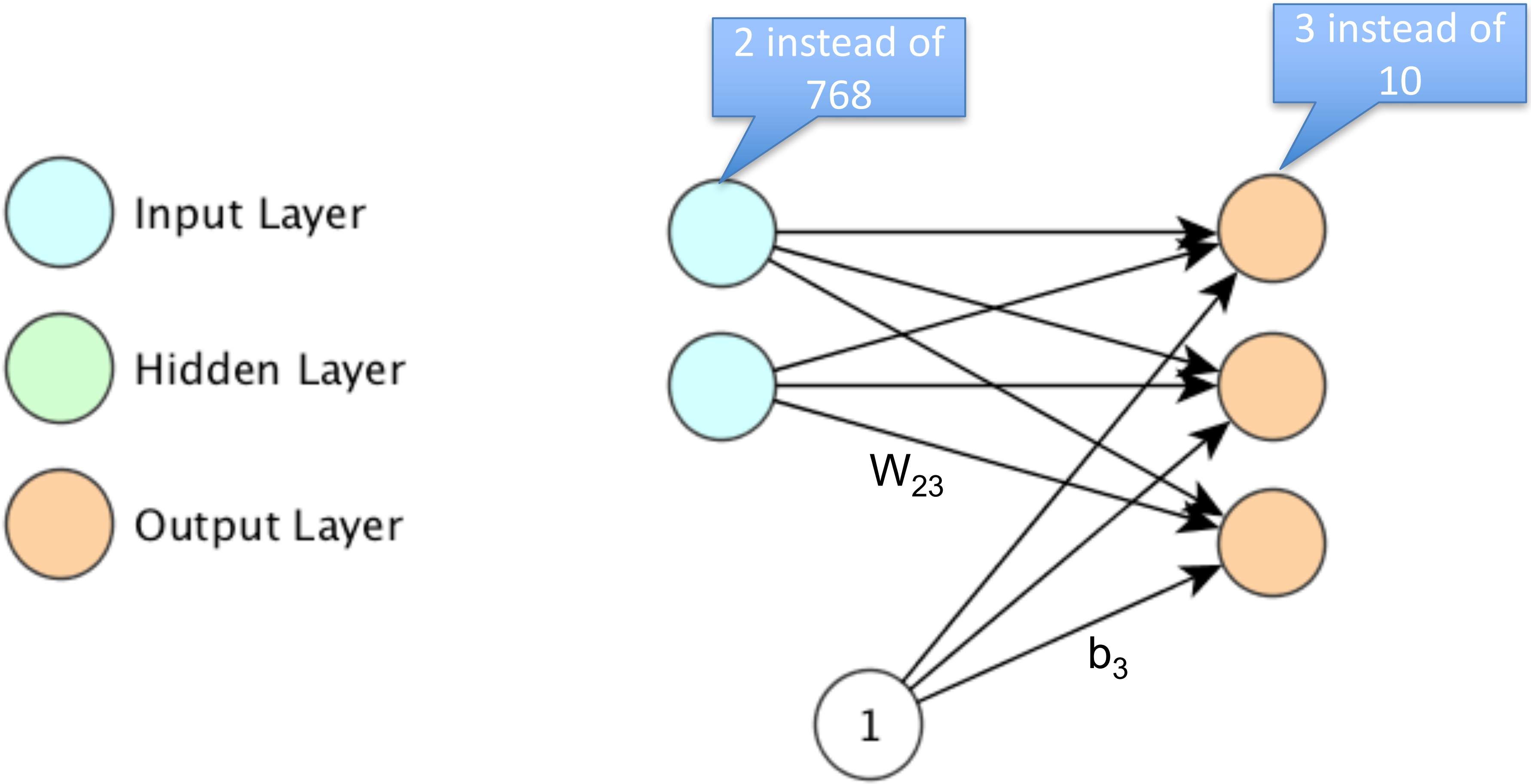
- MNIST the drosophila of all DL-Data sets
  - 50000 handwritten digits to be classified into 10 classes (0-9)



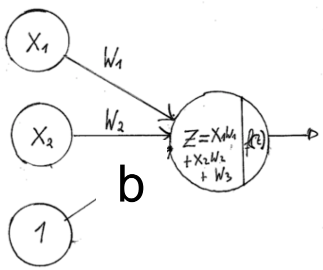
Input tensors: are flattened to 28*28=768 pixels

Image credit: https://www.tensorflow.org/versions/r0.10/images/MNIST-Matrix.png

Multinomial Logistic Regression



Multinominal Regression



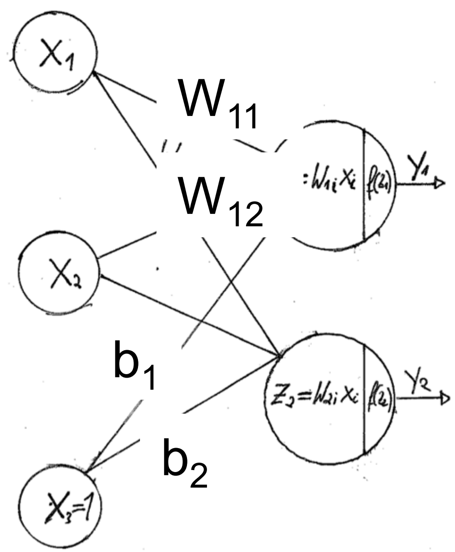
Binary Case

$$P(Y=1 \mid X=x) = \frac{1}{1+\exp(-z)} = \frac{\exp(\sum_i x_i W_i)}{1+\exp(\sum_i x_i W_i)} \propto \exp(\sum_i x_i W_i)$$

W_{12} = reads „from node 2 to 1“

More than one class

called logit



$$p_1 = P(Y_1 = 1 \mid X = x) \propto \exp(\sum_i x_i W_{i1} + b_1)$$

$$p_1 = \frac{\exp(\sum_i x_i W_{i1} + b_1)}{\sum_j \exp(\sum_i x_i W_{ij} + b_j)}$$

$$p_2 = P(Y_2 = 1 \mid X = x) \propto \exp(\sum_i x_i W_{i2} + b_2)$$

Normalisation

$$\sum_{i=1} p_i = 1$$

Multinomial case: just another **non-linearity *softmax***

$$p_1 = P(Y_1 = 1 \mid X = x) = \frac{\exp(\sum_i x_i W_{i1} + b_1)}{\sum_j \exp(\sum_i x_i W_{ij} + b_j)} = \text{softmax}(\sum_i x_i W_{i1} + b_1)$$

Recap: Matrix Multiplication aka dot-product of matrices

We can only multiply matrices if their dimensions are compatible.

A × **B** = **C**

(m × **n**) × (**n** × p) = (m × p)

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

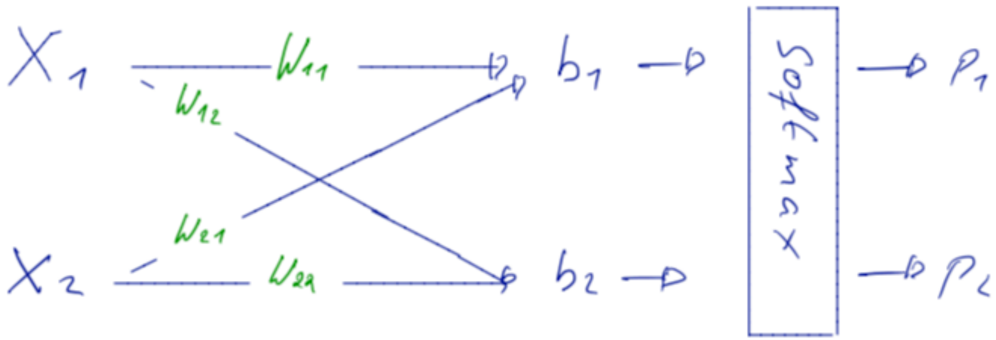
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} x \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32}$$
$$c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31}$$
$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}$$

Example:

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

GPUs love matrices (or tensors)



$$(p_1, p_2) = \text{softmax}(x_1 W_{11} + x_2 W_{21} + b_1, x_1 W_{12} + x_2 W_{22} + b_2)$$

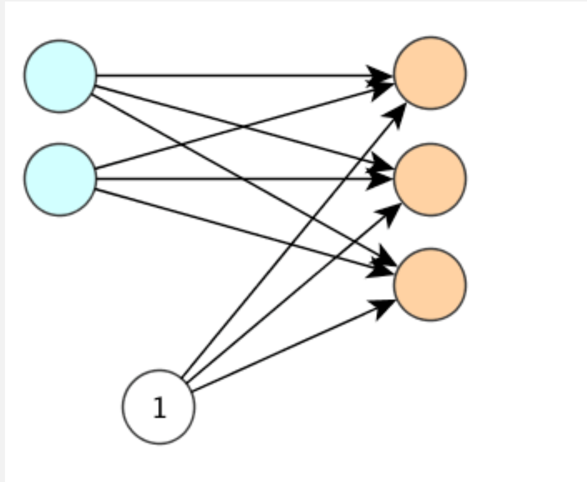
$$= \text{softmax}\left((x_1, x_2)\begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix} + (b_1, b_2)\right)$$

$$p = \text{softmax}(xW + b)$$

$$p_1 = P(Y_1 = 1 \mid X = x) = \frac{\exp(\sum_i x_i W_{i1} + b_1)}{\sum_j \exp(\sum_i x_i W_{ij} + b_j)} = \text{softmax}(\sum_i x_i W_{i1} + b_1)$$

Your turn

- Input x = (1,2)
- W = $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
- b = (1,2,3)



- Calculate the output using numpy:
- Hints:
- `x = np.asarray([[1,2]]) #`
- `np.matmul(.,.) # Matrix multiplication`
- `np.exp(.) # Exponential`
- `np.sum(.) # Sum`

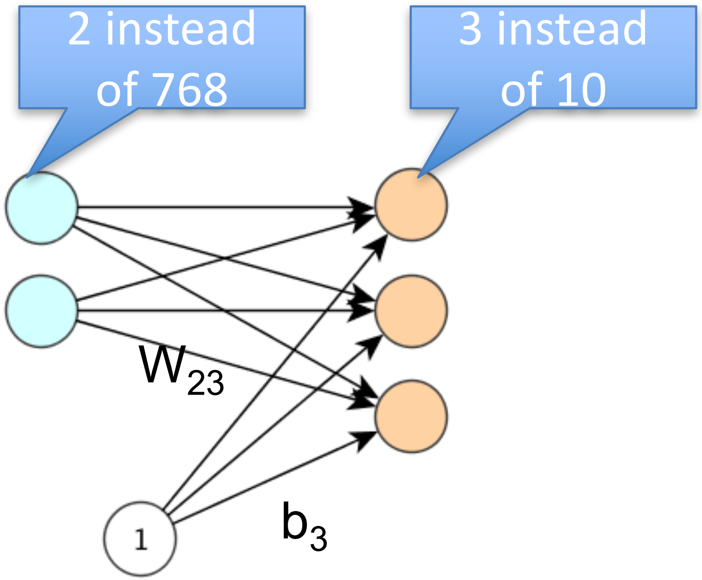
- `#Result:` array([[3.29320439e-04, 1.79802867e-02, 9.81690393e-01]])

GPUs love matrices: Use the source luke

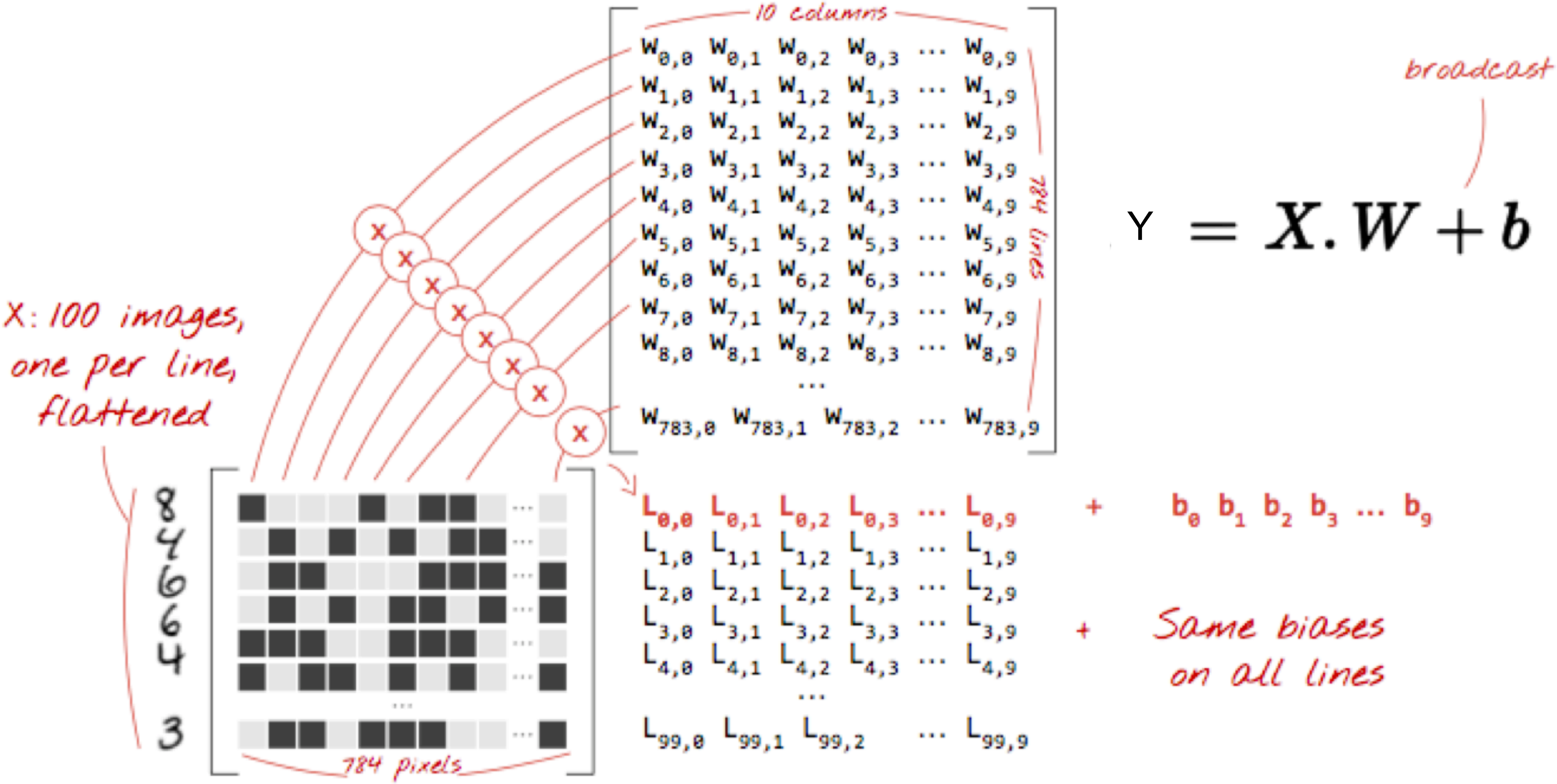
Mini batch size at runtime

```
…
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Data is usually processed in (mini-) batches. Instead of X being a 28*28=784 long vector, we use a batch (e.g. size 100)



GPUs love matrices:



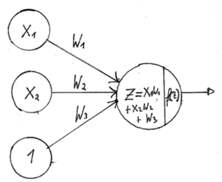

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Slide Credit: Martin Gröner TensorFlow and DL without a PhD https://docs.google.com/presentation/d/1TVixw6ItiZ8igjp6U17tcgoFrLSaHWQmMOwjlgQY9co/pub?slide=id.g110257a6da_0_431

Loss for multinomial regression

This is the prob. the model evaluates for the true class $y^{(i)}$ of training example $x^{(i)}$

Training Examples Y=1 or Y=0



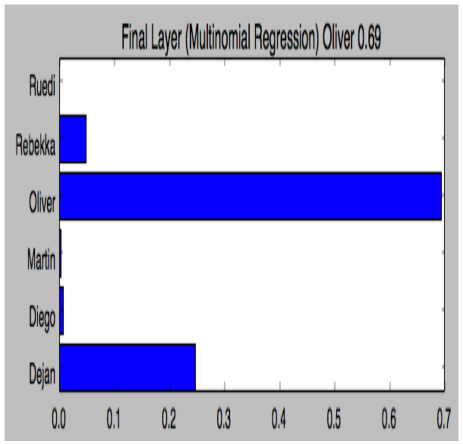
$$\text{loss} = -\frac{1}{N}\sum_{n=1}^{N}\log(p_{\text{model}}(y^{(i)} \mid x^{(i)};\theta))$$

$$\text{loss} = -\frac{1}{N}\sum_{n=1}^{N}\log(p_{\text{model}}(y^{(i)} \mid x^{(i)};\theta)) = -\frac{1}{N}\left(\sum_{i \in \text{All ones}} \log(p_1(x^{(i)})) + \sum_{i \in \text{All zeros}} \log(p_0(x^{(i)}))\right)$$

N Training Examples classes (1,2,3,…,K)

$$\text{loss} = -\frac{1}{N}\sum_{n=1}^{N}\log(p_{\text{model}}(y^{(i)} \mid x^{(i)};\theta)) = -\frac{1}{N}\left(\sum_{i \in y_j=1} \log(p_1(x^{(i)})) + \sum_{i \in y_j=2} \log(p_2(x^{(i)})) + \ldots + \sum_{i \in y_j=K} \log(p_K(x^{(i)}))\right)$$

p_i



Output of last layer

Example: Look at class of single training example. Say it's Dejan, if classified correctly p_dejan = 1 ➔ Loss = 0. Real bad classifier put's p_dejan=0 ➔ Loss = Inf.

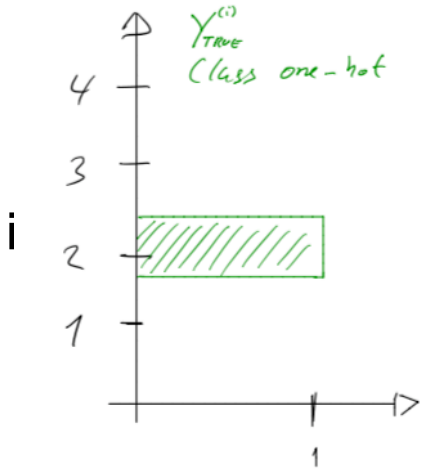
One more Trick: Loss function with indicator function

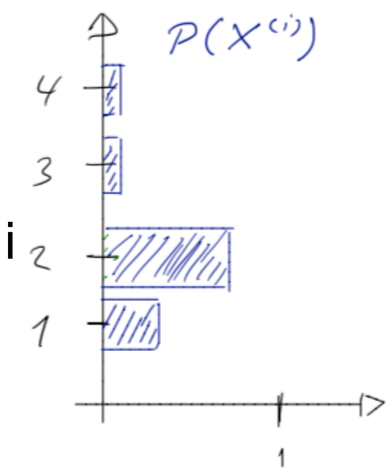
A one-hot-encoded y picks the right class, form all of the K different classes.

For MNIST K=10, so why calculate, 9 logs and through them away?
(Parallel executions)

$$-N \cdot \text{loss} = \sum_{i \in y_j = 1} \log(p_1(x^{(i)})) + \sum_{i \in y_j = 2} \log(p_2(x^{(i)})) + \ldots + \sum_{i \in y_j = K} \log(p_K(x^{(i)})) = \sum_{i=1}^{N} y^{(i)}_{\text{true}} \log(p(x^{(i)})) = \sum_{i=1}^{N} y^{(i)}_{\text{true}} \log(y_i)$$

one-hot-encoded





$$\text{Loss} = \frac{-1}{N} \sum_{i} Y^{(i)}_{TRUE} \ln P(X^{(i)})$$

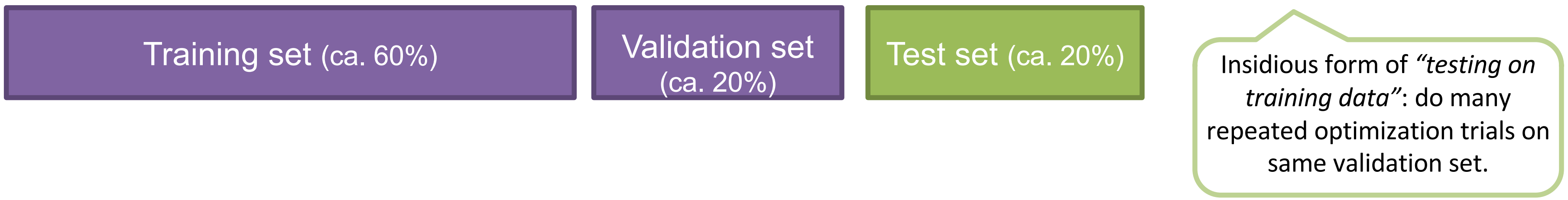
See later crossentropy and KL-Distance between y_i and $p(x^{(i)})$

Training Neural Networks: Split of the data

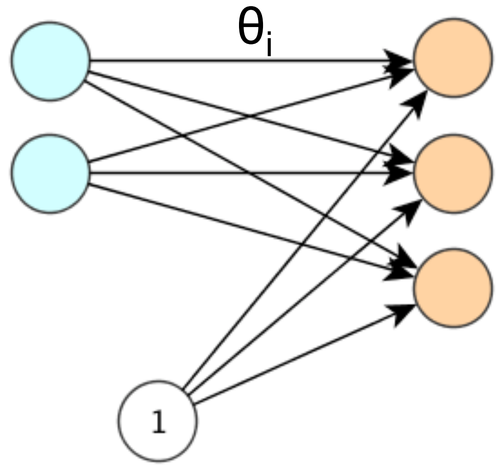
For neural networks usually no cross-validation is done (due to long learning times).

For our use case (4000 images)

- Training set 3000, Test set 1000
- 20% of the Training set is taken as Validation Set



Stochastic gradient descent



- The loss function

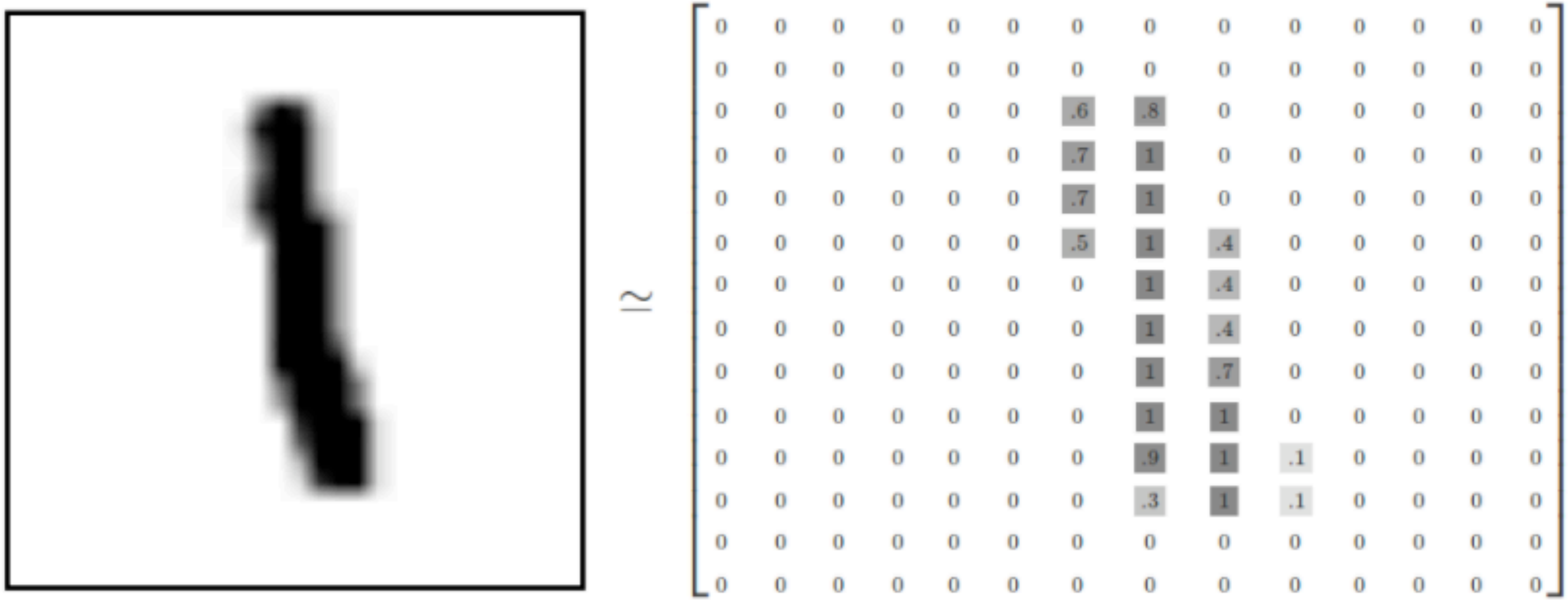
$$\text{loss} = -\frac{1}{N}\sum_{n=1}^{N} \log(p_{\text{model}}(y^{(i)} \mid x^{(i)}; \theta))$$

- A particular weight is updated using the partial derivative of the loss function (the sum) w.r.t θ_i
- The sum is taken over the whole training set of size N. Often the training set is split into mini-batches size of e.g. bs=128 (*)
- These mini-batches are processed one after another
- When all examples have been processed once, we speak of one epoch being finished
- For a new epoch one often reshuffles the data

- The batch size is chosen so that input tensor fits on the GPU.

(*) For some purists only when bs=1 is called stochastic gradient descent

Exercise: The MNIST Data Set



Input tensors

One minibatch has dimension (128, 28, 28, 1) (batch, x,y, color)

or (128, 784) flattened

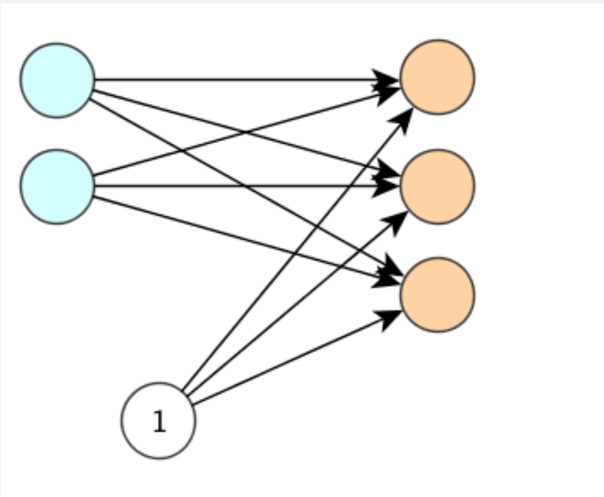
Image credit: https://www.tensorflow.org/versions/r0.10/images/MNIST-Matrix.png

Exercise: Implement multinomial logistic regression

Finish the code in the notebook: **Multinomial Logistic Regression**

- Think about the trick how the loss is calculated!
- Compare the loss and accuracy in the validation set with the loss in the training set. Why is there such a difference?
- Question: How many parameters do we have?

Hints:



$$p_j = \frac{\exp(\sum_i x_i W_{ij} + b_j)}{\sum_{j'} \exp(\sum_i x_i W_{ij'} + b_{j'})} = \left(\text{softmax}(\text{xW+b})\right)_j$$

SOLUTION

- We have
  - For W 28*28*10 = 7840 Parameter
  - For b 10 Parameter
  - Together 7850 Parameters

- Trick with the loss function [Blackboard]
  - `loss = tf.reduce_mean(-tf.reduce_sum(y_true * tf.log(y_pred), reduction_indices=[1]))`

- See: https://github.com/tensorchiefs/dl_course/blob/master/notebooks/05_Multinomial_Logistic_Regression_solution.ipynb

- https://github.com/tensorchiefs/dl_course/blob/master/notebooks_misc/Explanation_of_loss.ipynb

Alternative solution

```
w = tf.Variable(tf.random_normal([784, 10], stddev=0.01))
b = tf.Variable(tf.zeros([10]))
z = tf.matmul(x,w)+b #aka logits
loss = tf.reduce_mean(
        tf.nn.softmax_cross_entropy_with_logits(labels=y_true,logits=z)
)

#Old Solution
prob = tf.nn.softmax(z)
loss_old = tf.reduce_mean(-tf.reduce_sum(y_true * tf.log(prob),
reduction_indices=[1]))
```

For numerical stability, one should use

tf.nn.softmax_cross_entropy_with_logits

There is also a sparse version (no one hot encoded needed)

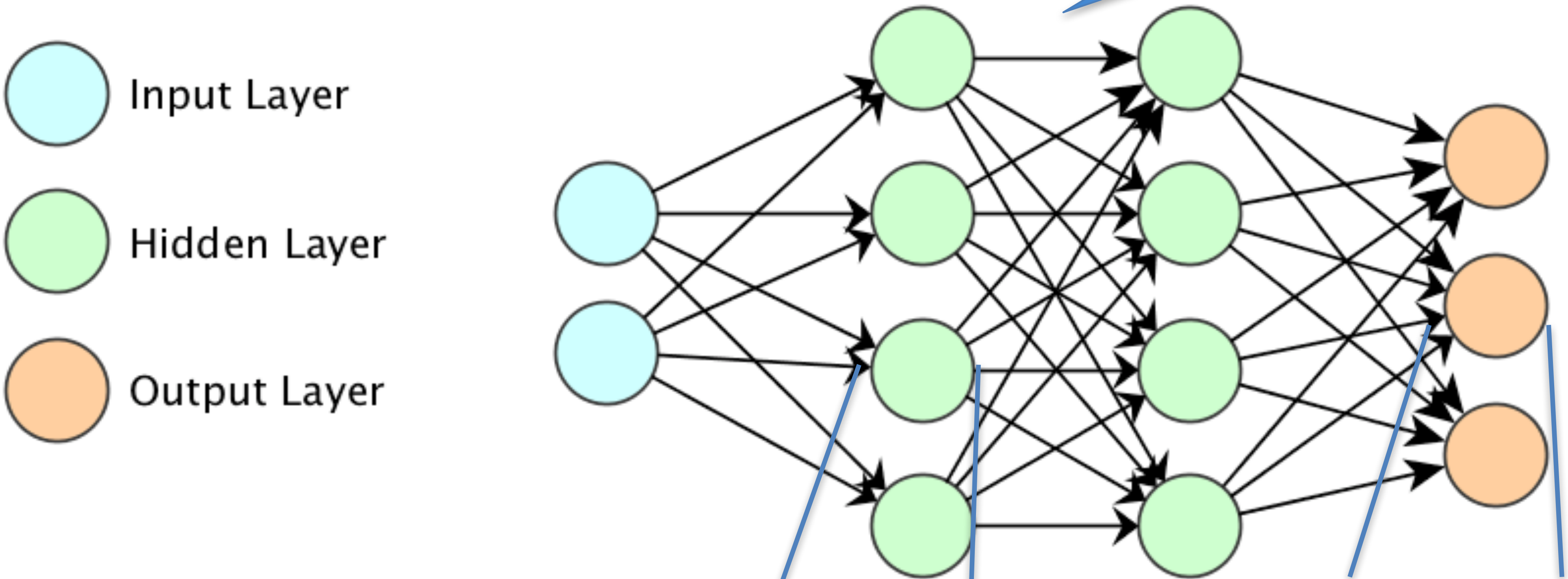
tf.nn.sparse_softmax_cross_entropy_with_logits

Now we are well prepared to entre the realm of deep learning

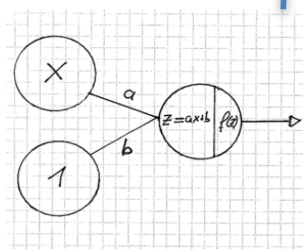

WE NEED TO GO
DEEPER
memegene

Today: Fully Connected Networks

Real networks of course are larger. But this captures the basic structure



We finished with....
Multinomial Logistic Regression



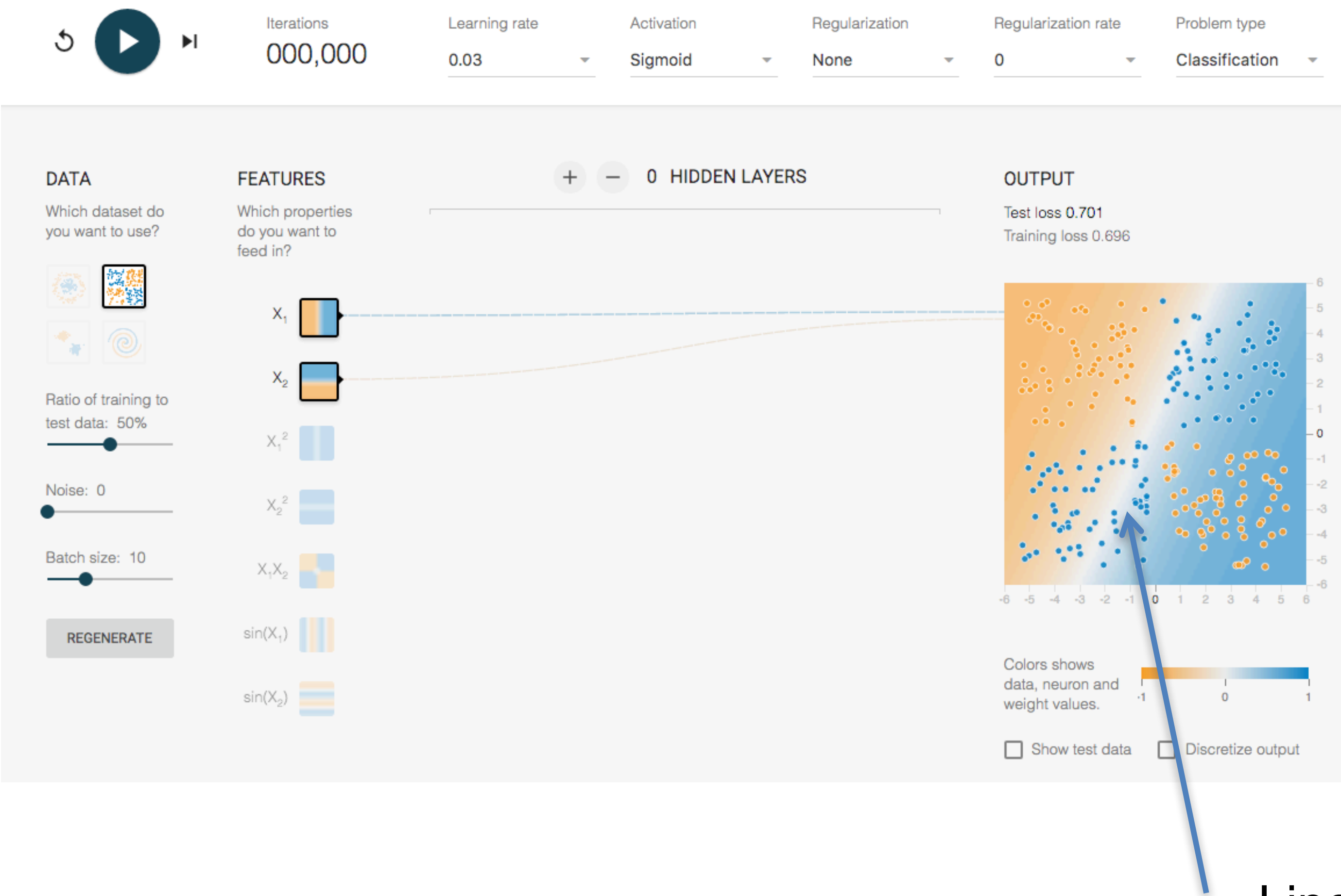
We started with....

1-D Logistic Regression

Networks with hidden layers

Limitations of (multinomial) logistics regression

Logistic regression in NN speak: “no hidden layer”



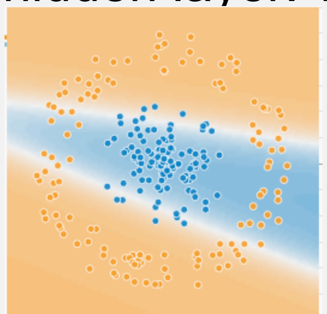
Linear Boundary!

[Network taken from]()

Neural Network with hidden units

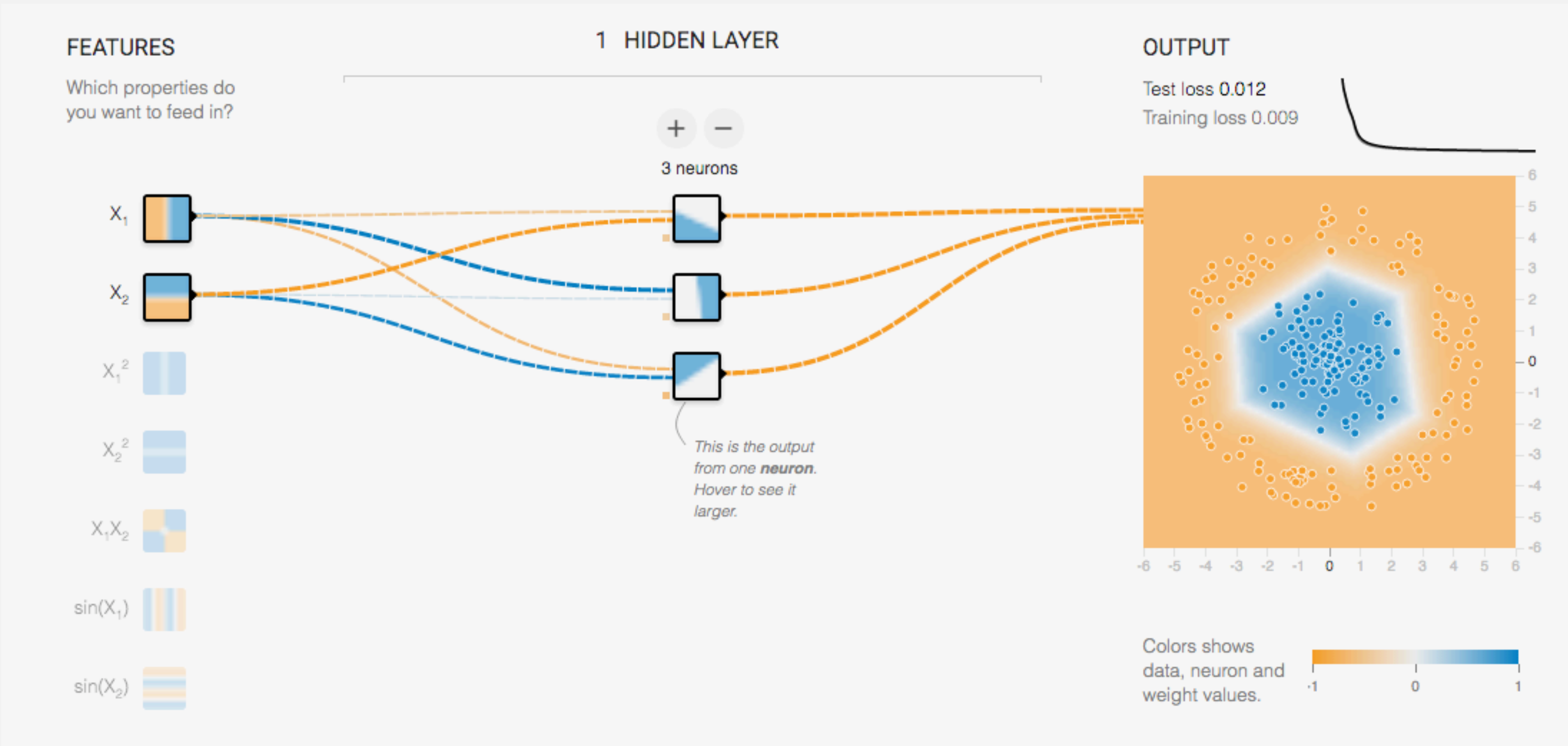
- Go to http://playground.tensorflow.org (https://goo.gl/VR3db5) and train a neural network for the data:

- Start with 0 hidden layers. Increase the number of hidden layers to one, what do you observe?
- Now go to here (https://goo.gl/XwLRKB) and increase the number of neurons in the hidden layer. What do you observe?

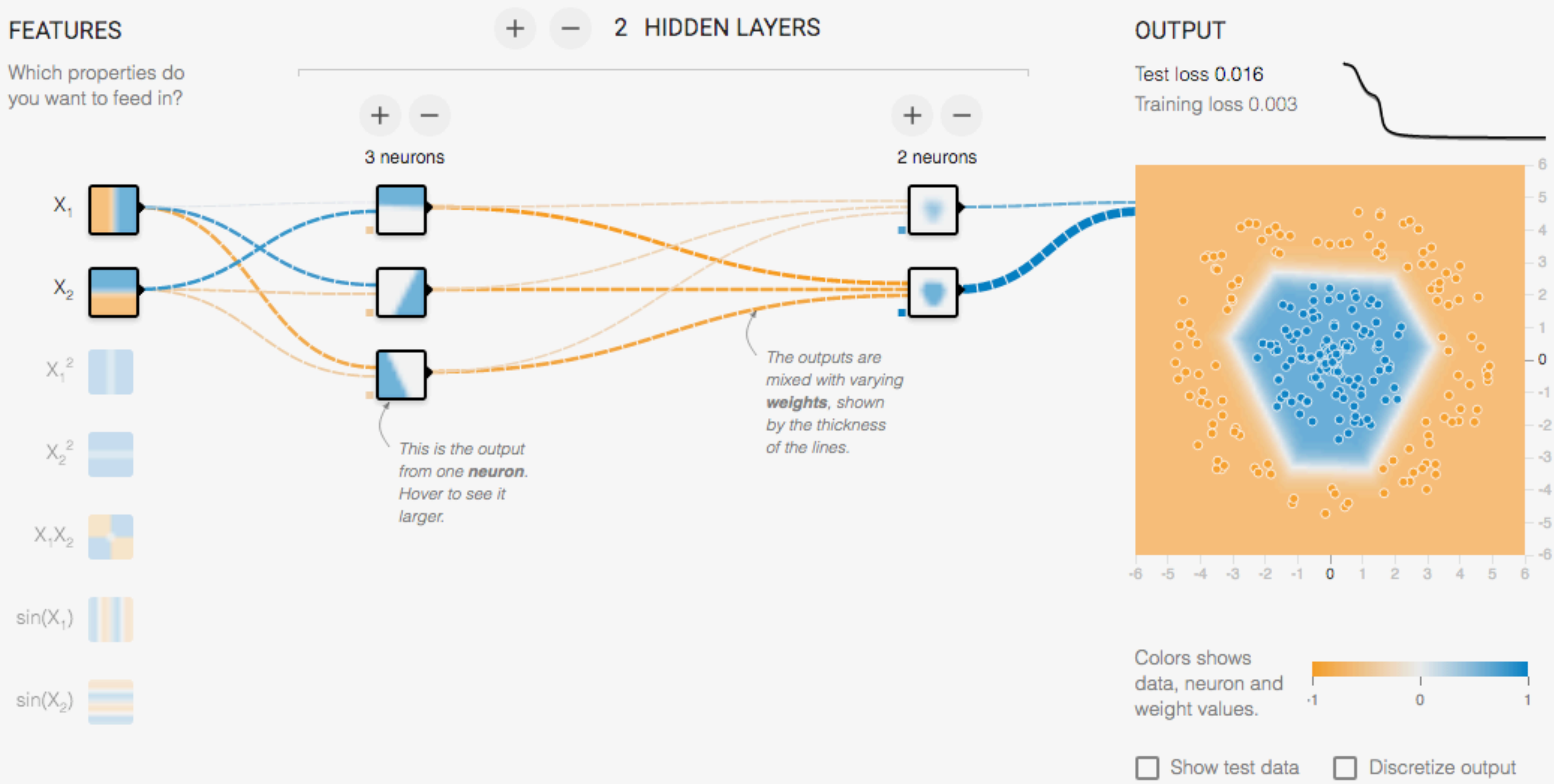


Results

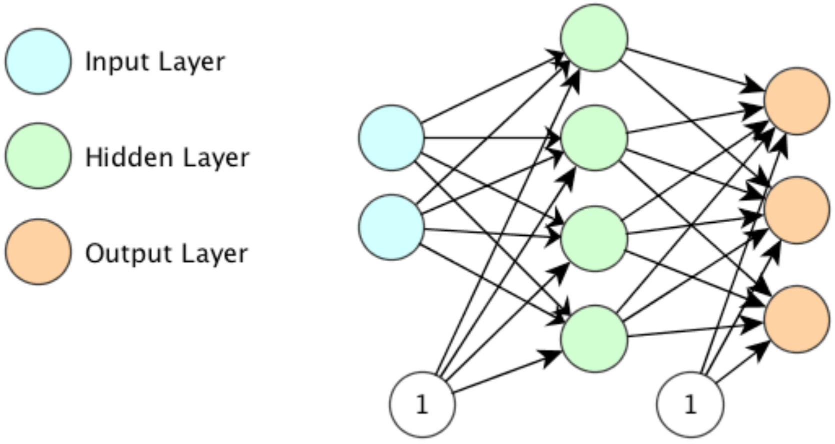
- 0 hidden layers, only a single line
- Many neurons in a hidden layer → also complicated functions



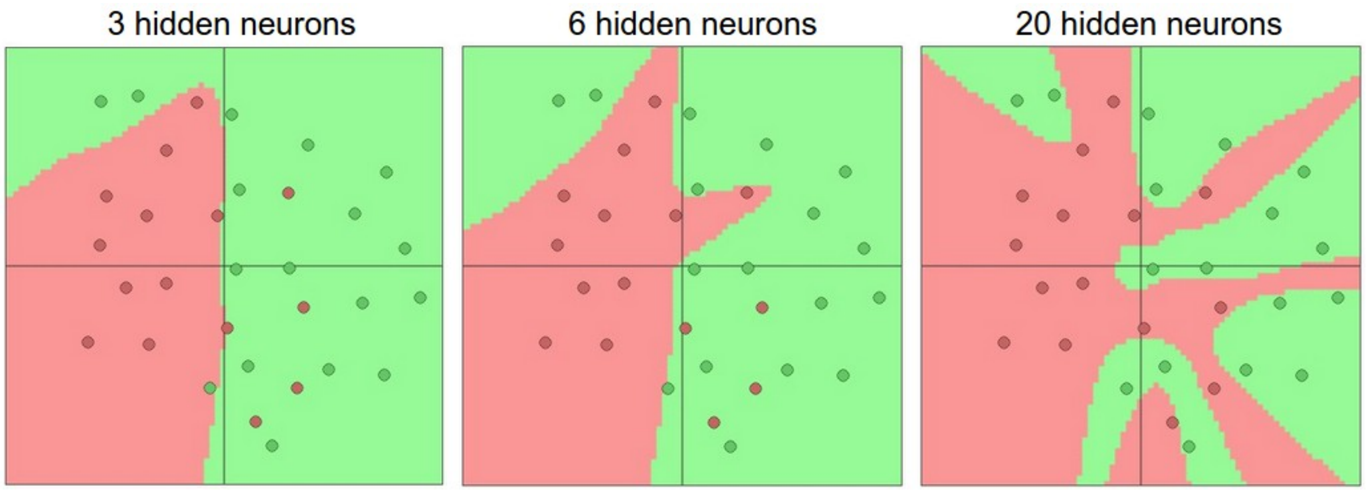
Results (cont)



One hidden Layer



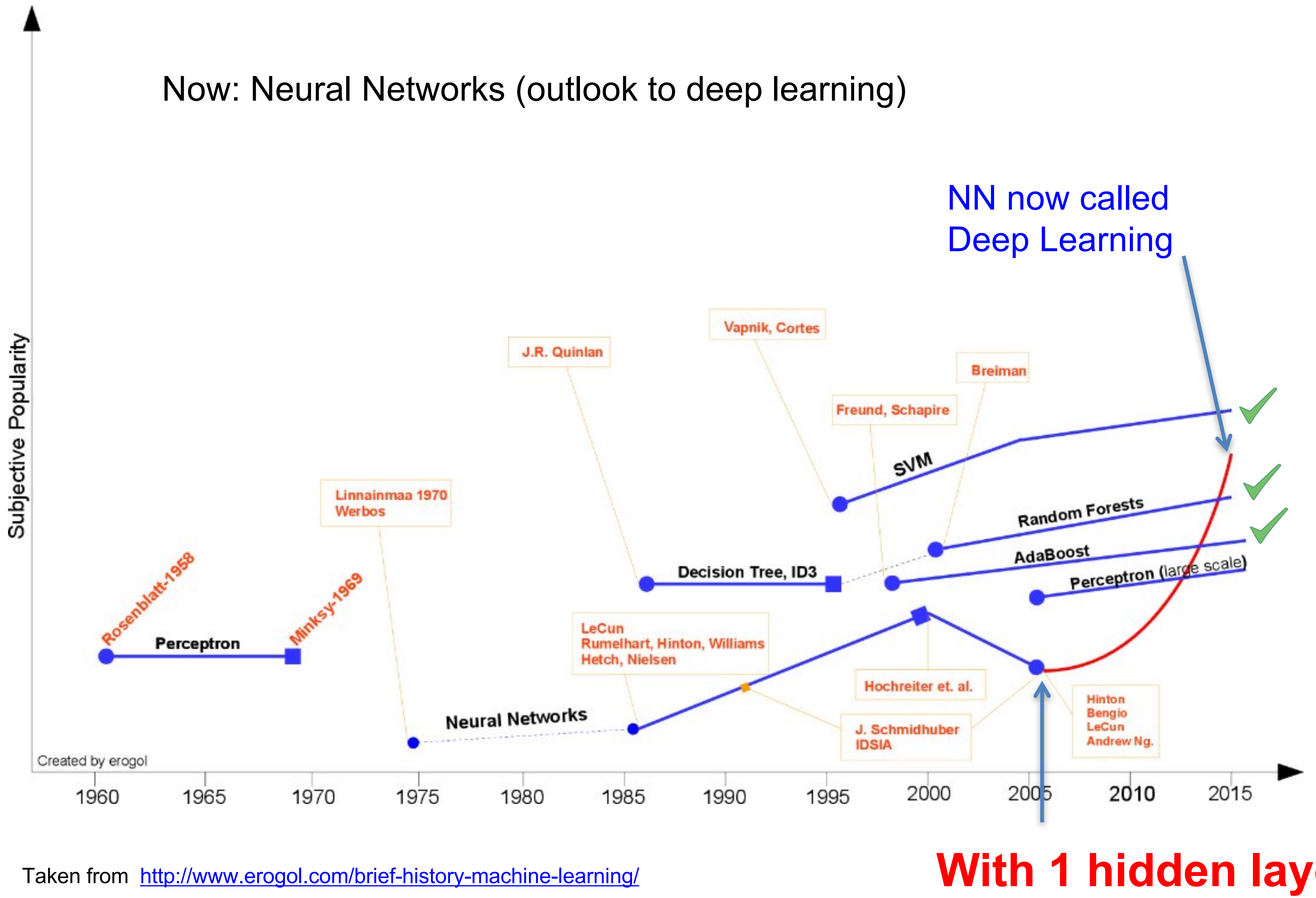
A network with one hidden layer is a universal function approximator!



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

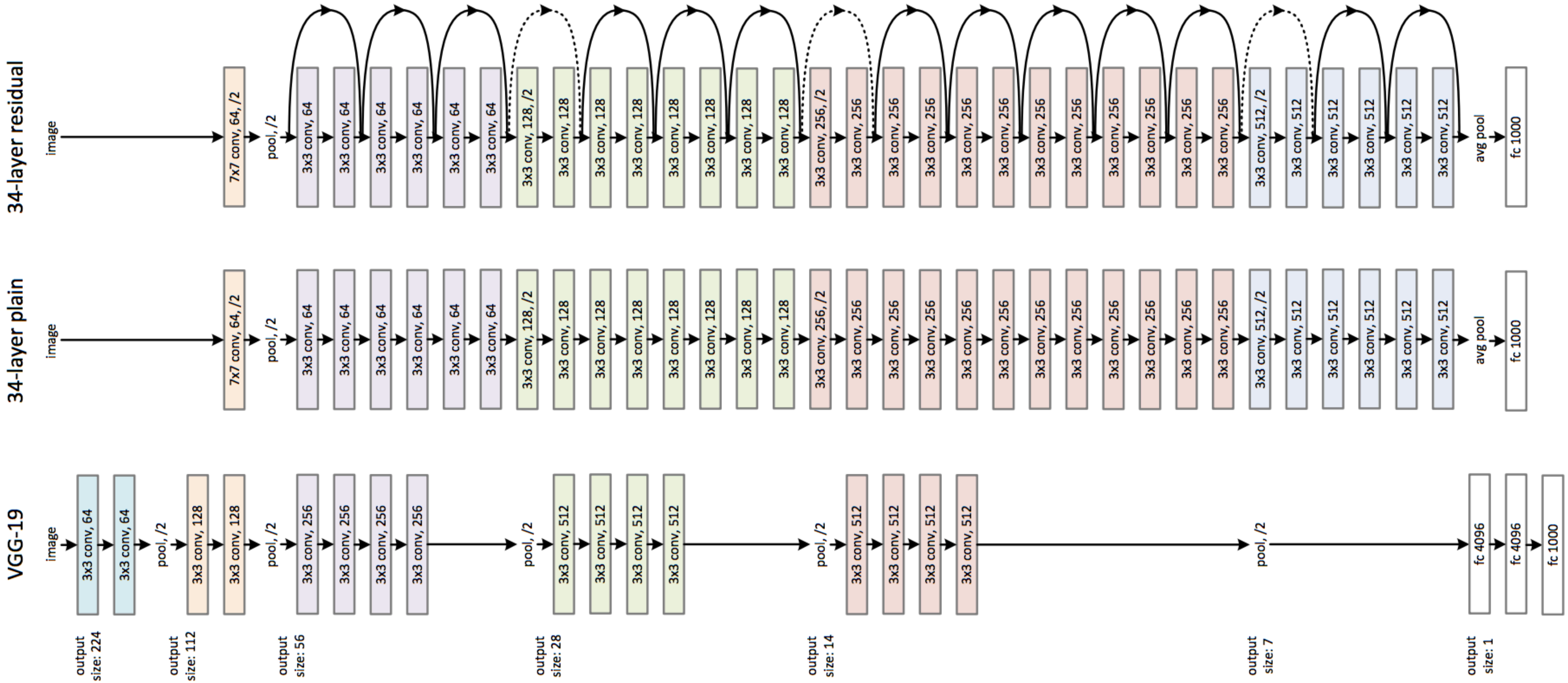
Brief History of Machine Learning (supervised learning)

Now: Neural Networks (outlook to deep learning)



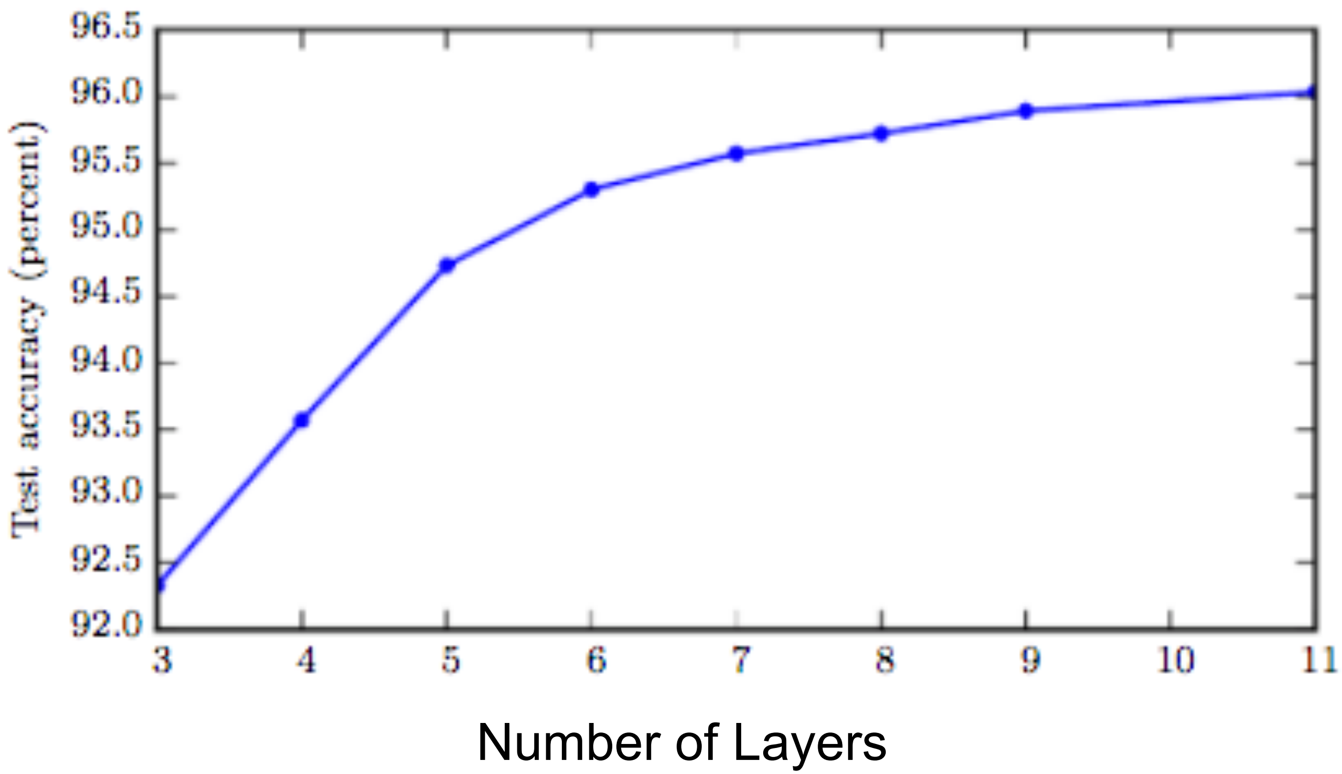
Taken from http://www.erogol.com/brief-history-machine-learning/

Examples of deep architectures



Original Resnet had 152 Layers: https://arxiv.org/abs/1512.03385

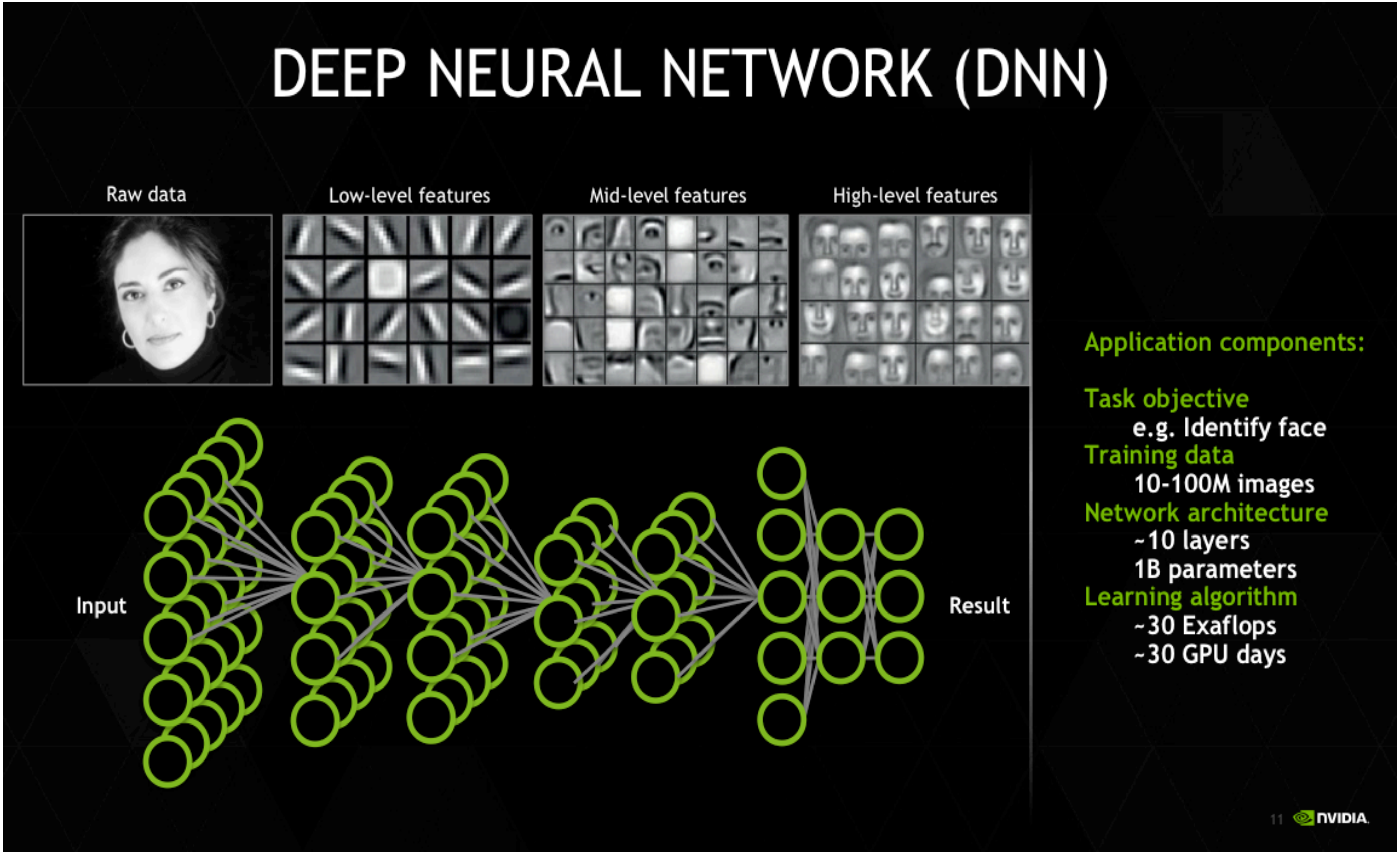
Why going deep: Experimental evidence



The test set accuracy consistently increases with increasing depth. Just increasing model size does not yield the same performance.

Taken from: http://www.deeplearningbook.org/contents/mlp.html

Why Deep: Hierarchy of learned features in Object Detection



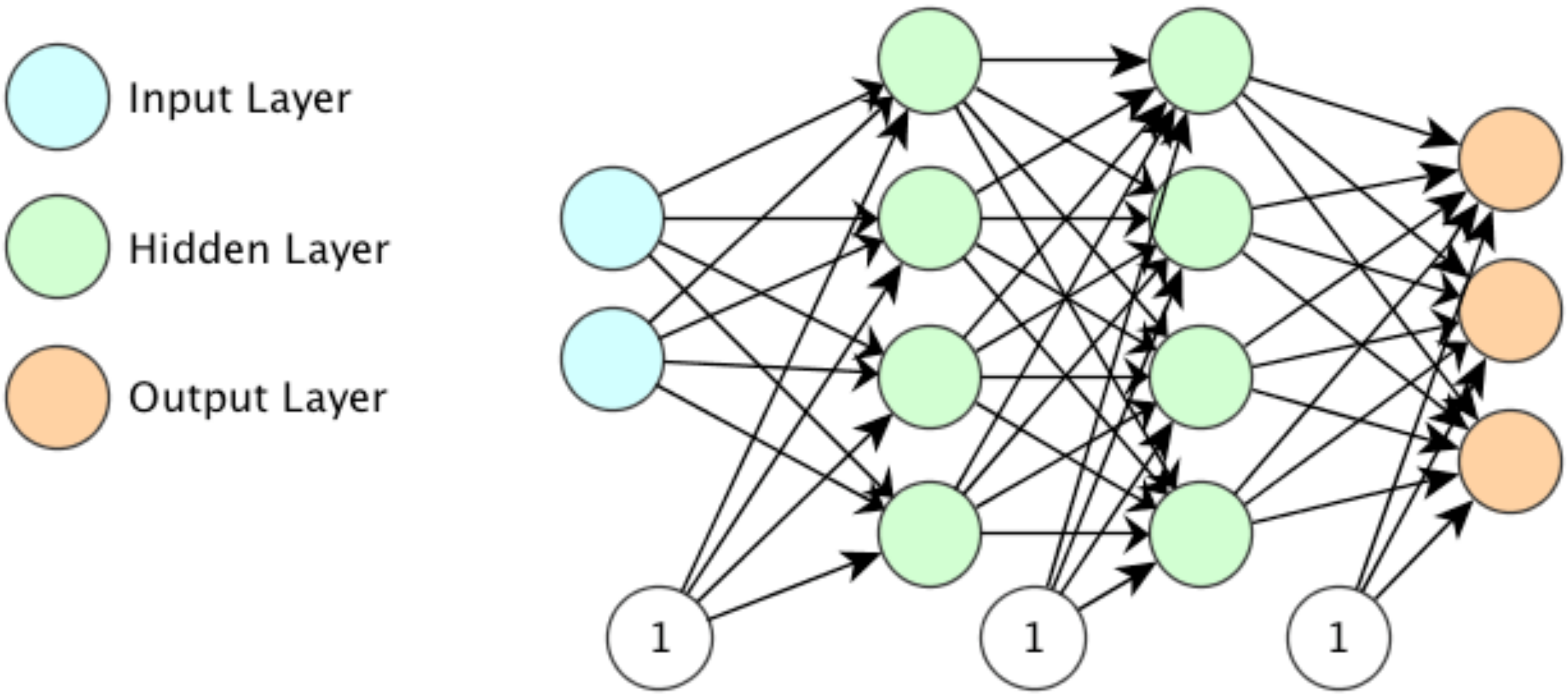
Why deeper (summary)?

- If a network with one hidden layer is a universal function approximator, why bother to go deeper?
  - Step functions are universal function approximators, too. Would you use them?
- Representational power:
  - There is experimental evidence that a 3 layered network needs less weights in total than a network with one hidden layer.
  - Theoretically backed for some functions
- For some applications as image classification there is a natural hierarchy of features to be learned
- More details see: http://cs231n.github.io/neural-networks-1/#power and references therein.
- Still active research area and not solved yet
  - Novel approach Tishby information plane, see e.g. his talk at Yandex https://www.youtube.com/watch?v=bLqJHjXihK8

More than one layer

We have all the building blocks

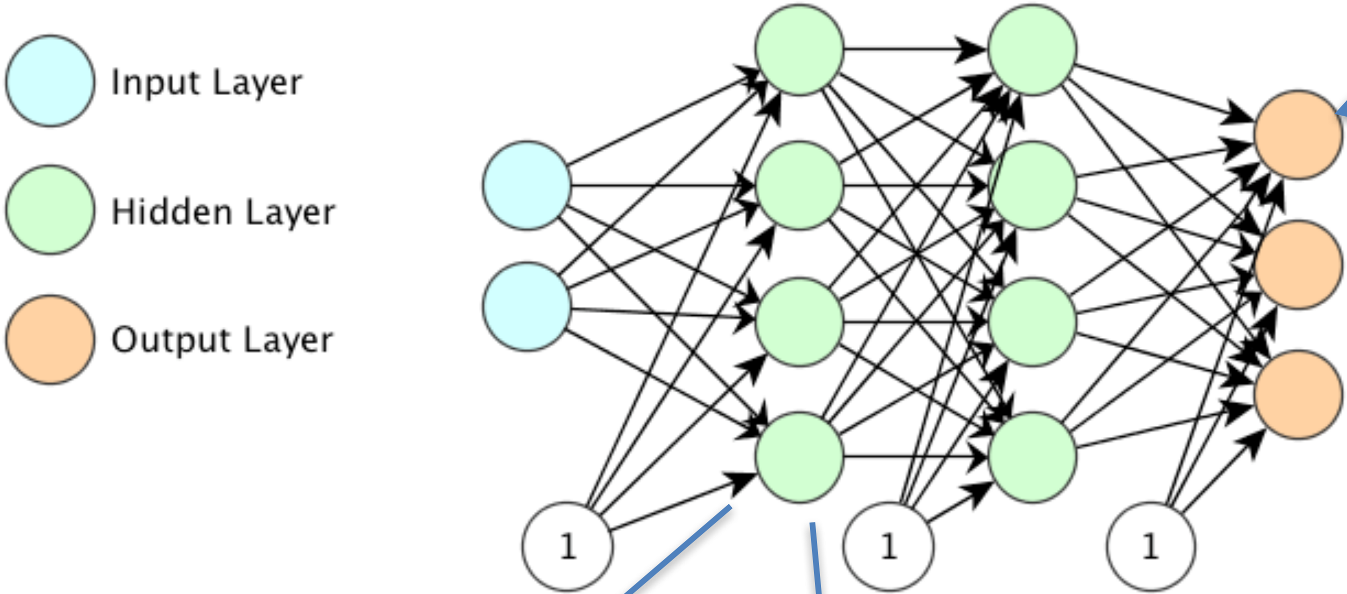
- Use outputs as new inputs
- At the end use multin. logistic regression
- Names:
  - Fully connected network
  - Multi Layer Perceptron (MLP)



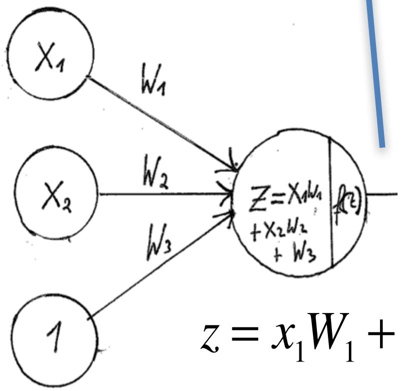
Summary

Softmax in last layer

$$p_1 = \frac{\exp(\sum_i W_{1i} x_i + b_1)}{\sum_j \exp(\sum_i W_{ij} x_i + b_i)}$$



Logistic Regression in hidden layers

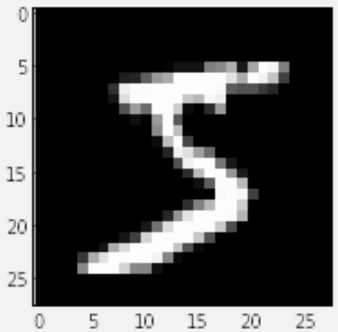


$$f(z) = \frac{\exp(z)}{1 + \exp(z)}$$

$$z = x_1 W_1 + x_2 W_2 + b = Wx + b$$

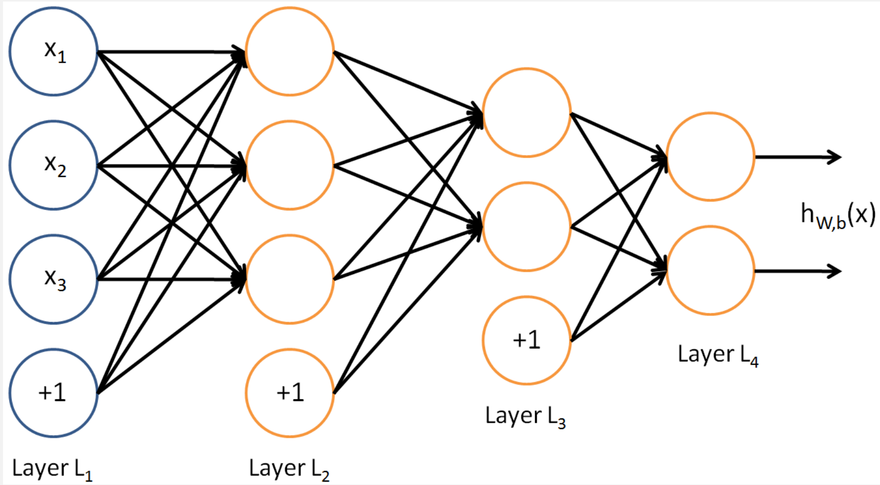
Other activation functions for the hidden layers (see later)

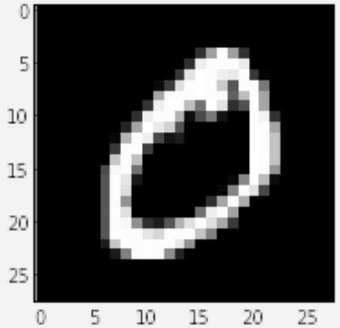
A network for classifying digits



$X^{(1)}$

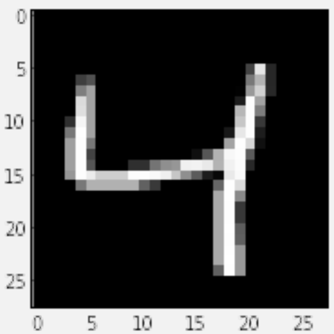
Sketch of the network (not all nodes shown)





$X^{(2)}$

…



$X^{(N)}$

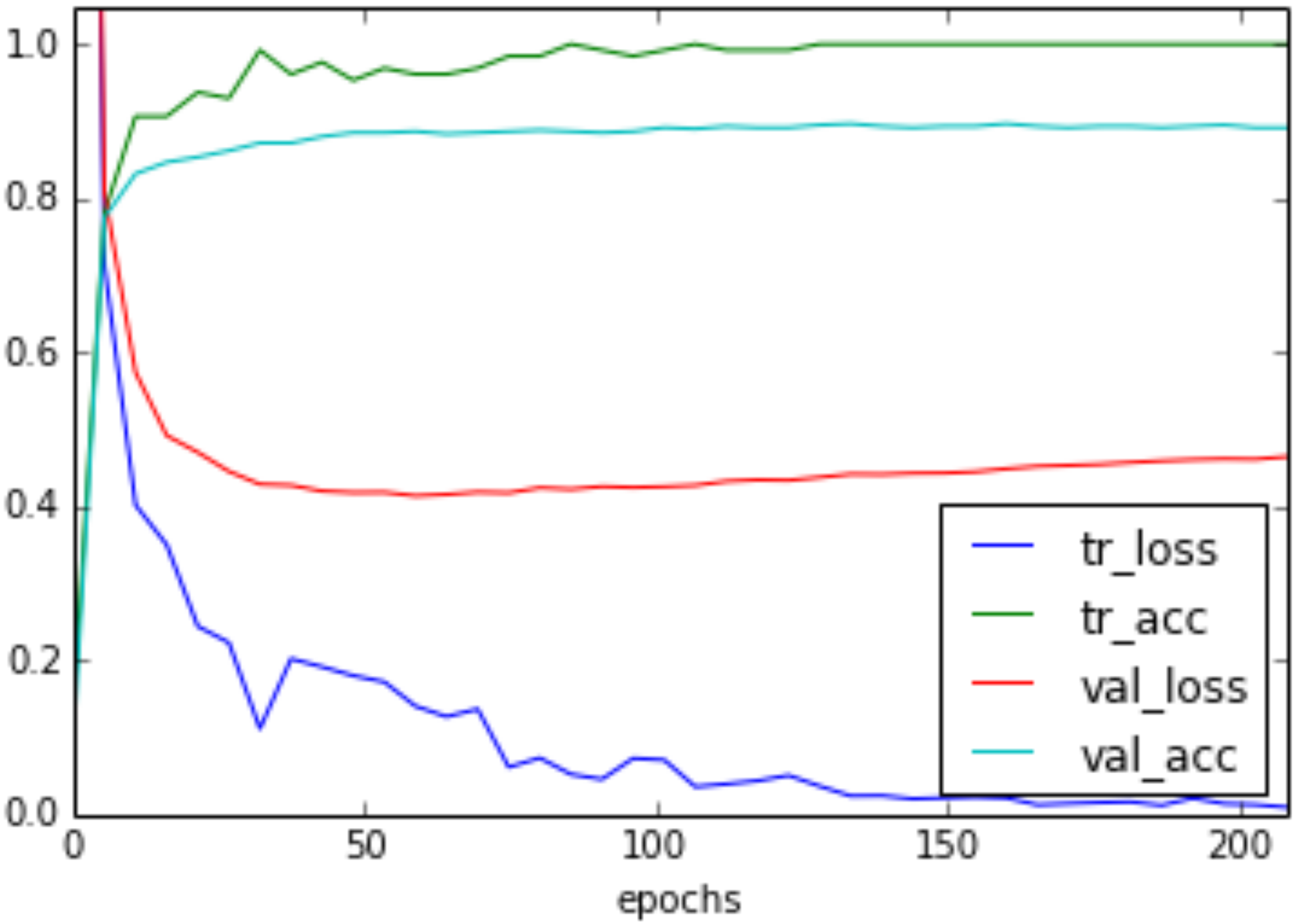
Images 28x28 =784

500 50 10 Number of Nodes

Number of weights to fit:
(785 * 500) + (501 * 50) +(51 * 10) = 418'060

Task: Have a look at the notebook: fcn_MNIST and complete "your code here" parts

Results



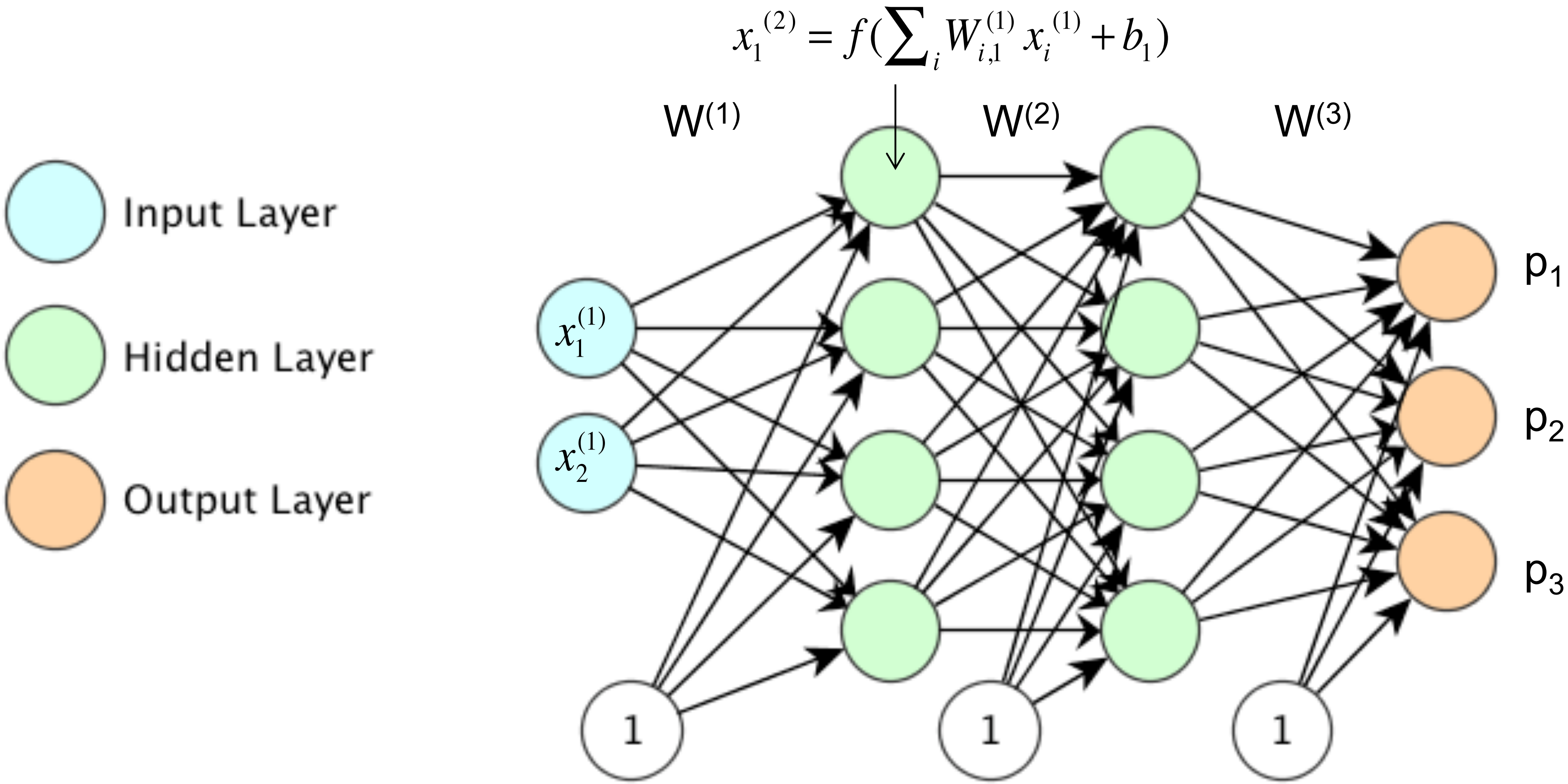
We get an accuracy of about 89% on the validation set.
Training error and loss approach zero. Validation error and loss increase with time (overfitting).

Summary

- Where do we stand?
  - In Principle we now can use deep networks
  - There are some tricks, we learn shortly.
  - To understand those tricks we have to get an understanding how learning works…

- Learning / gradient flow
  - Nowadays networks are learnt with gradient descent
  - For each weight a gradient w.r.t. loss is calculated and the weights are adapted
  - As we see a gradient signal flows from the loss to the input

Layer / chain structure of networks



Simple chaining

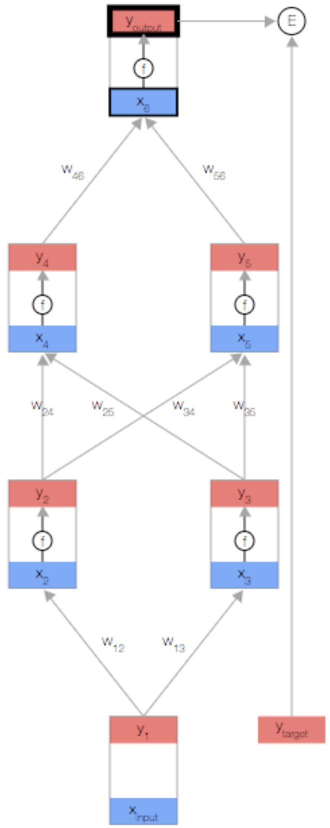
$$p=\mathrm{softmax}(b^{(3)} + W^{(3)} f(b^{(2)} + W^{(2)} f(b^{(1)} + W^{(1)}x^{(1)})))$$

Backpropagation

Slide Credit to Elvis Murina for the great animations

Motivation:
The forward and the backward pass

- https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/



Chain rule recap

- If we have two functions f,g
  $y = f(x)\ and$
  $z = g(y)$
  then y and z are dependent variables.

- And by the chain rule:

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

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

Gradient flow in a computational graph: local junction

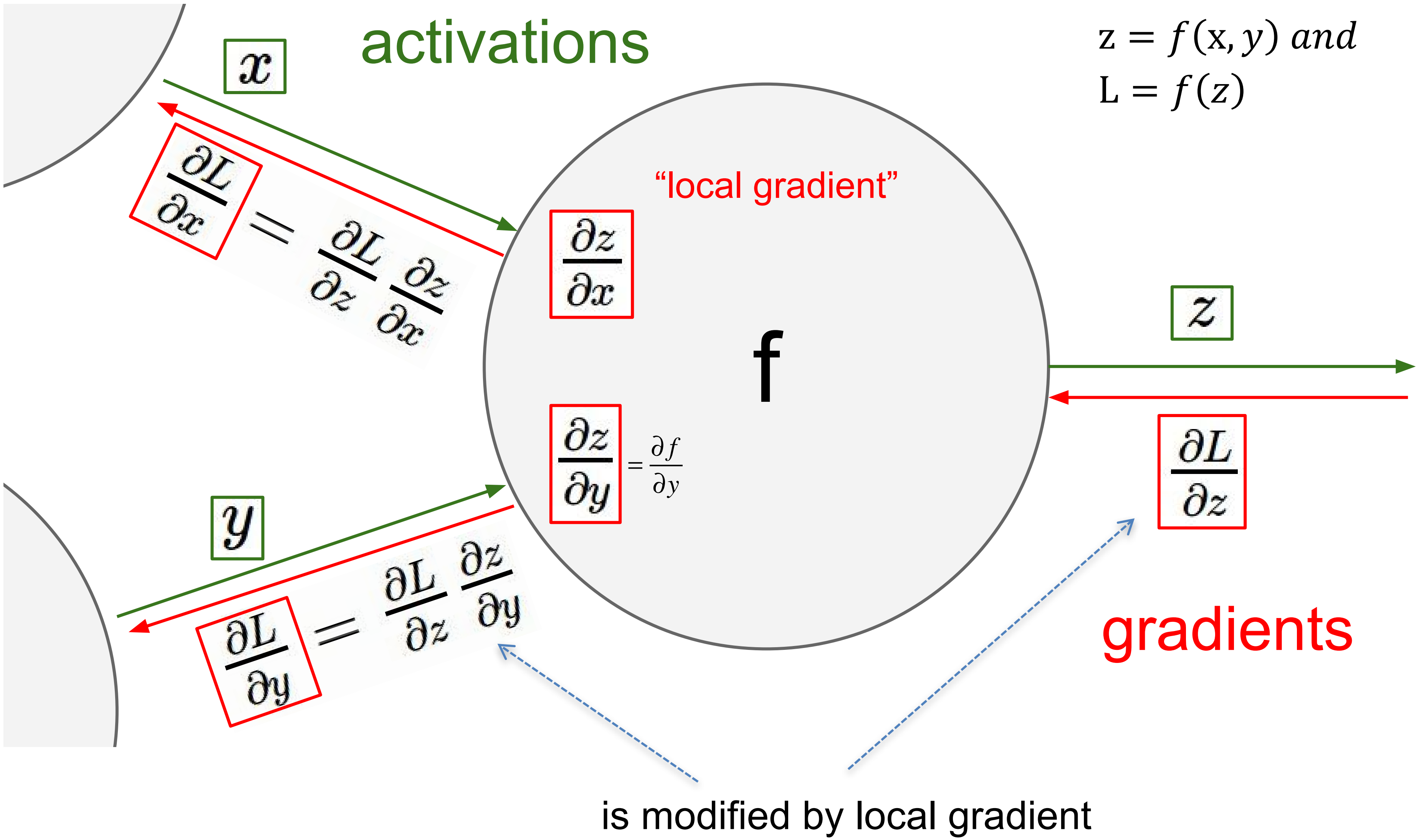
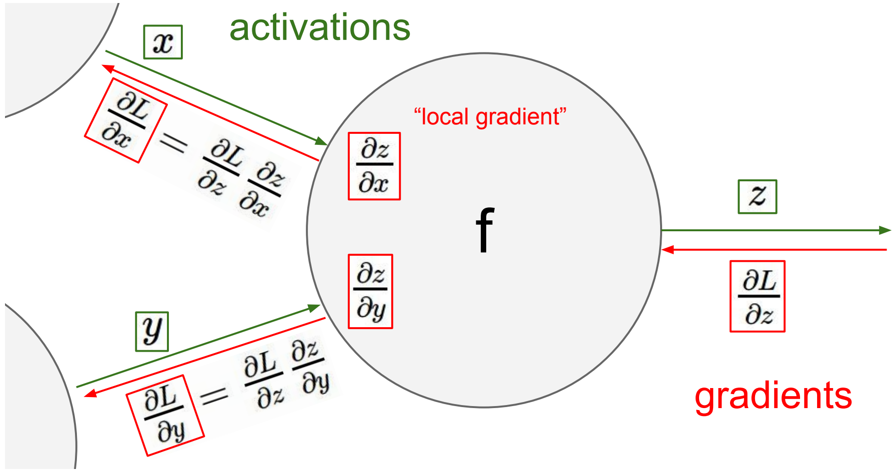


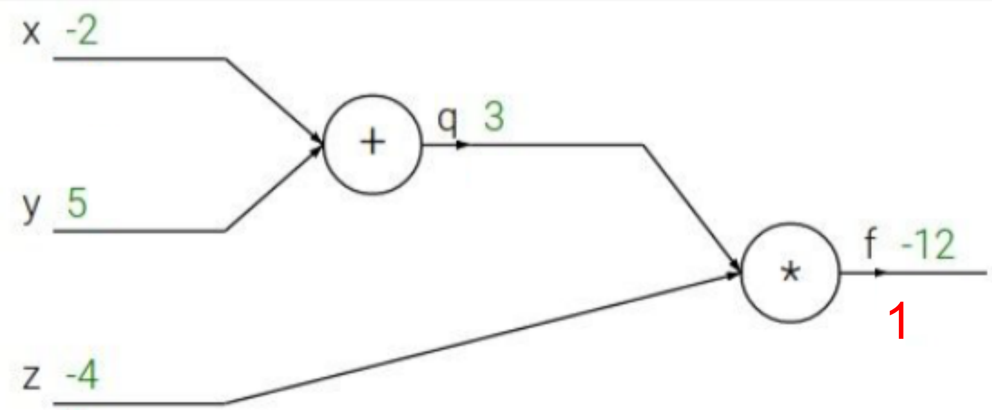
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Example



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

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



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

➔Multiplication do a switch

Forward pass

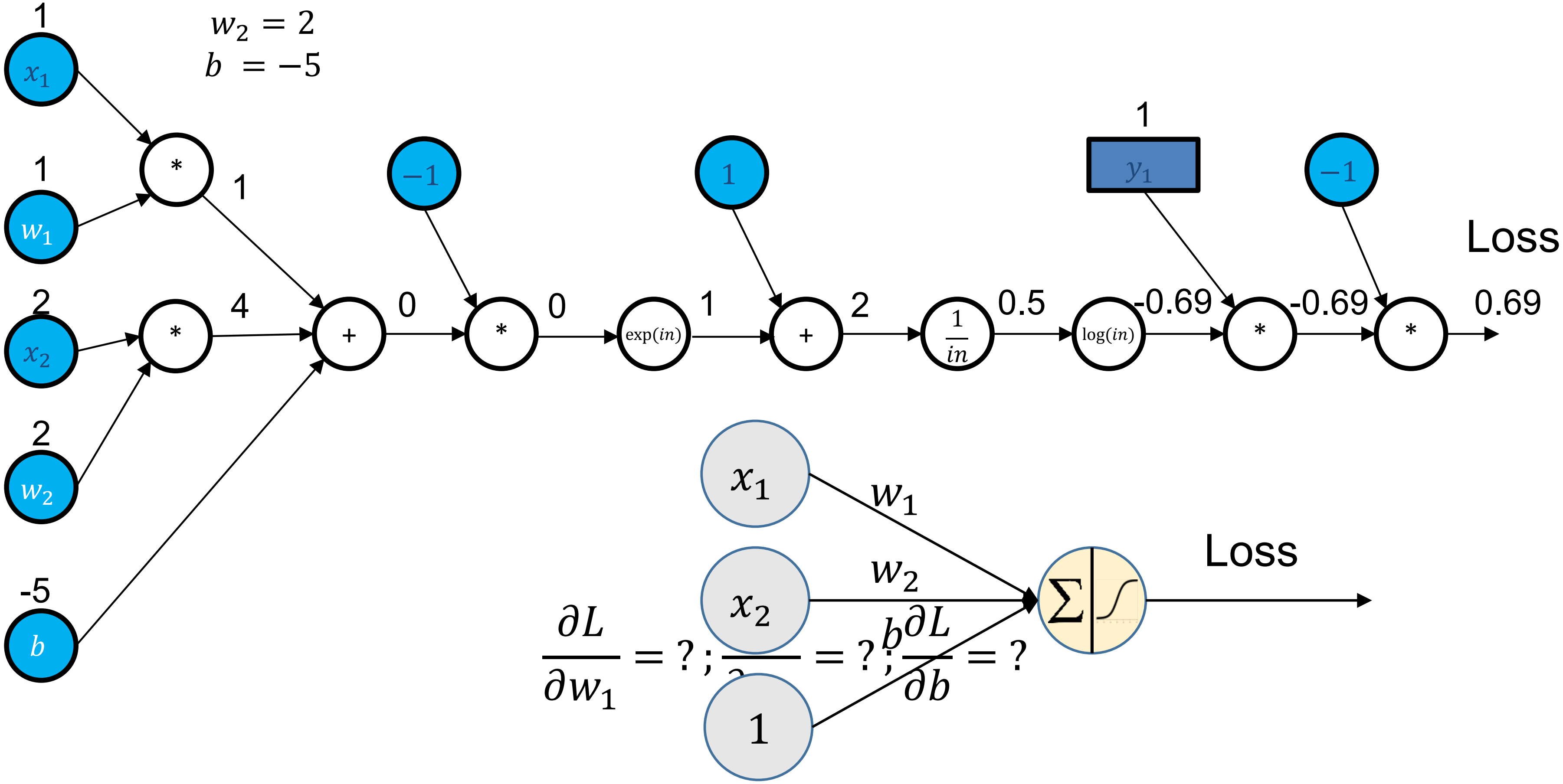
Training data:

$x_1 = 1$
$x_2 = 2$
$y_1 = 1$

Initial weights:

$w_1 = 1$
$w_2 = 2$
$b = -5$

$$p(y = 1|X) = \frac{1}{1 + e^{-(x_1 * w_1 + x_2 * w_2 + b)}}$$



$$\frac{\partial L}{\partial w_1} = ?\, ; \frac{\partial L}{\partial w_2} = ?\, ; \frac{\partial L}{\partial b} = ?$$

Backward pass

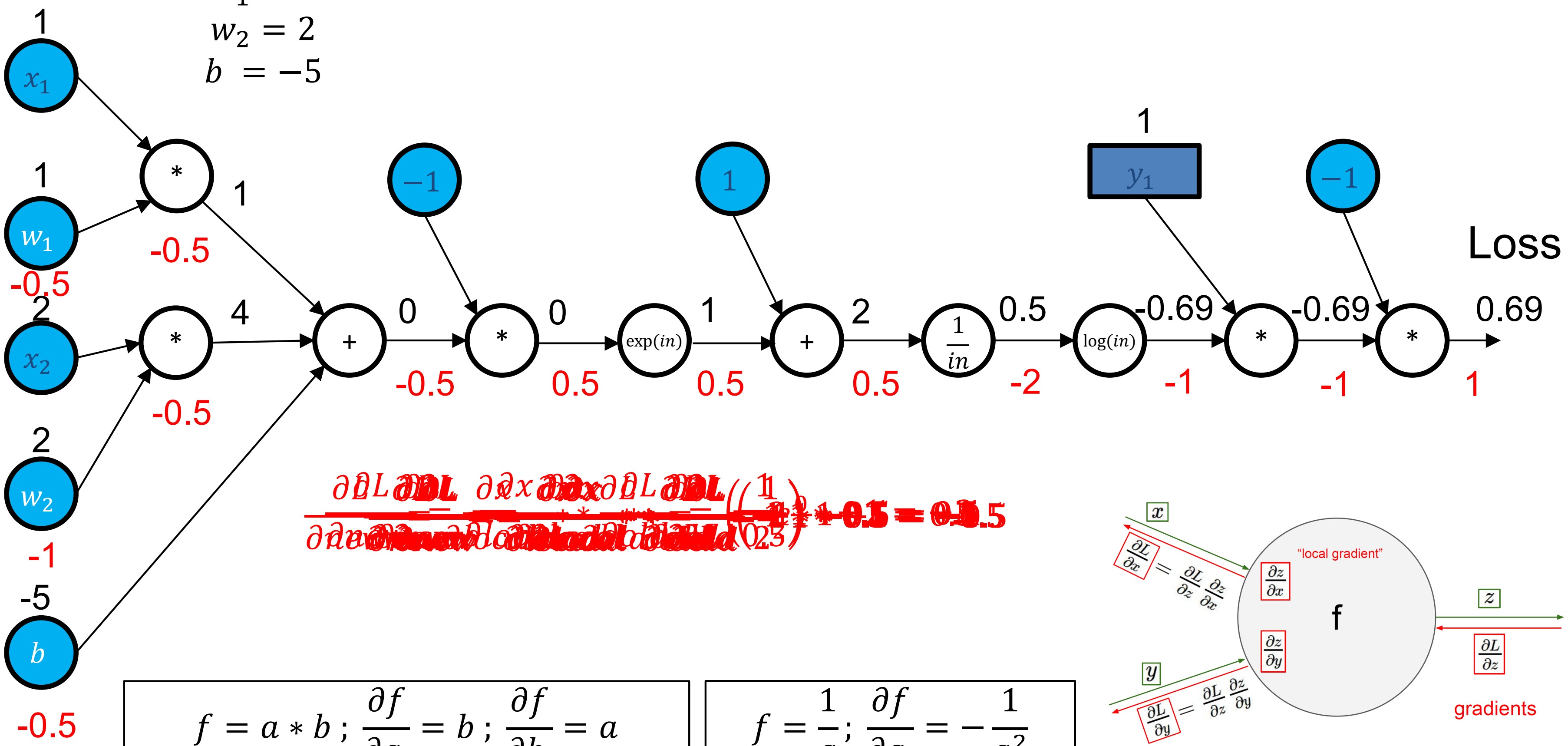
Training data:

$x_1 = 1$
$x_2 = 2$
$y_1 = 1$

Initial weights:

$w_1 = 1$
$w_2 = 2$
$b = -5$

$$p(y = 1|X) = \frac{1}{1 + e^{-(x_1 * w_1 + x_2 * w_2 + b)}}$$



$$f = a * b\,;\ \frac{\partial f}{\partial a} = b\,;\ \frac{\partial f}{\partial b} = a$$

$$f = \frac{1}{a};\ \frac{\partial f}{\partial a} = -\frac{1}{a^2}$$

$$f = a + b\,;\ \frac{\partial f}{\partial a} = 1\,;\ \frac{\partial f}{\partial b} = 1$$

$$f = e^a;\ \frac{\partial f}{\partial a} = e^a$$

$$f = \log(a)\,;\ \frac{\partial f}{\partial a} = \frac{1}{a}$$

Forward pass

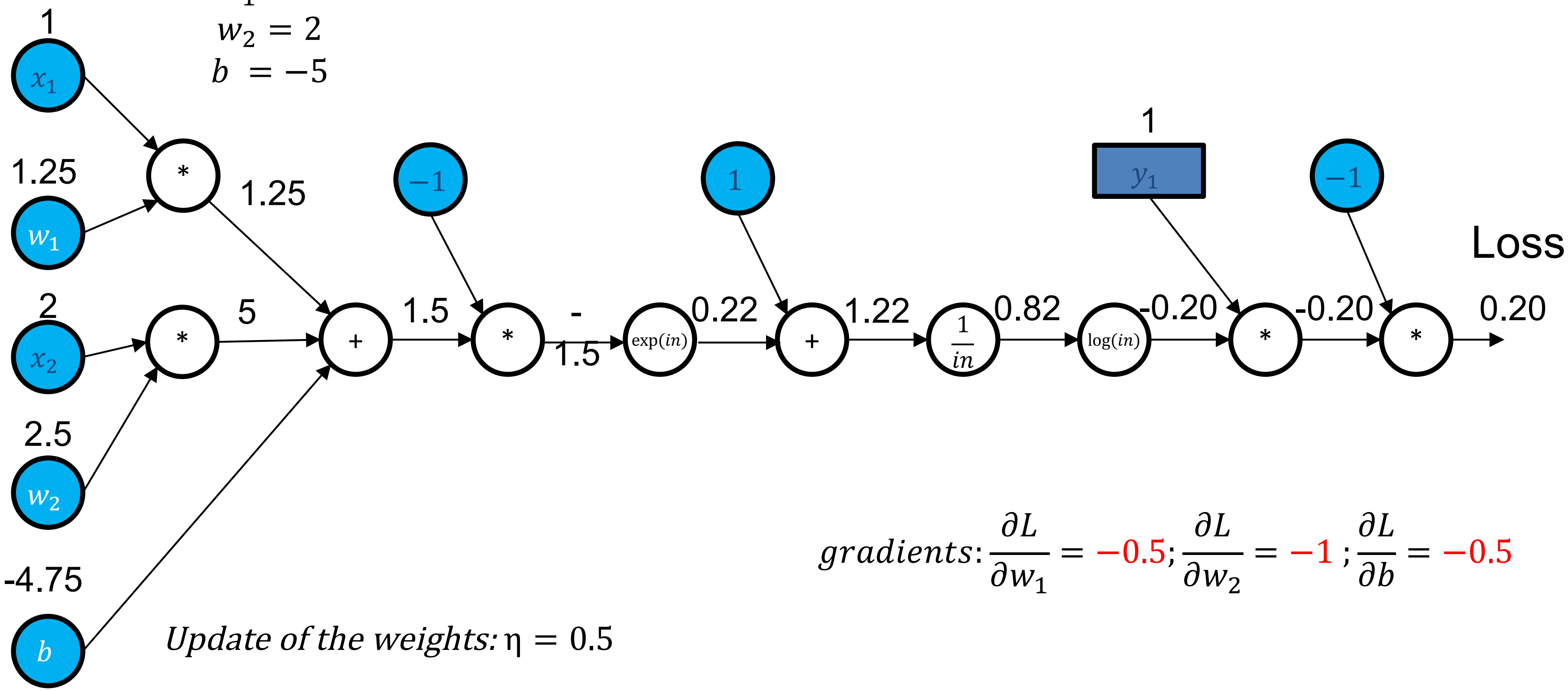
Training data:

$x_1 = 1$
$x_2 = 2$
$y_1 = 1$

Initial weights:

$w_1 = 1$
$w_2 = 2$
$b = -5$

$$p(y = 1|X) = \frac{1}{1 + e^{-(x_1 * w_1 + x_2 * w_2 + b)}}$$



$$gradients: \frac{\partial L}{\partial w_1} = -0.5; \frac{\partial L}{\partial w_2} = -1 ; \frac{\partial L}{\partial b} = -0.5$$

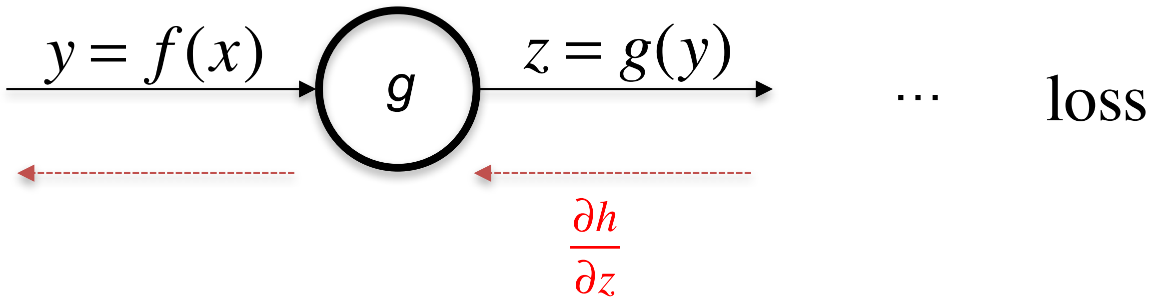
Update of the weights: $\eta = 0.5$

$$w_{1(t+1)} = w_{1(t)} - \eta * \frac{\partial L}{\partial w_1} = 1 - 0.5 * (-0.5) = 1.25$$

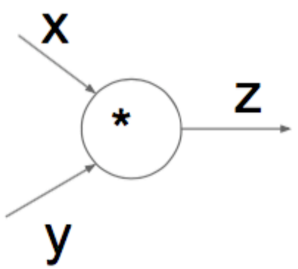
$$w_{2(t+1)} = w_{2(t)} - \eta * \frac{\partial L}{\partial w_2} = 2 - 0.5 * (-1) = 2.5$$

$$b_{(t+1)} = b_{(t)} - \eta * \frac{\partial L}{\partial b} = -5 - 0.5 * (-0.5) = -4.75$$

Side remark:



- Some DL frameworks (e.g. Torch) do not do symbolic differentiation. For these for each operation needs to store only
  - The actual value y coming in and the value of derivative $\left.\frac{\partial g}{\partial y}\right|_y$



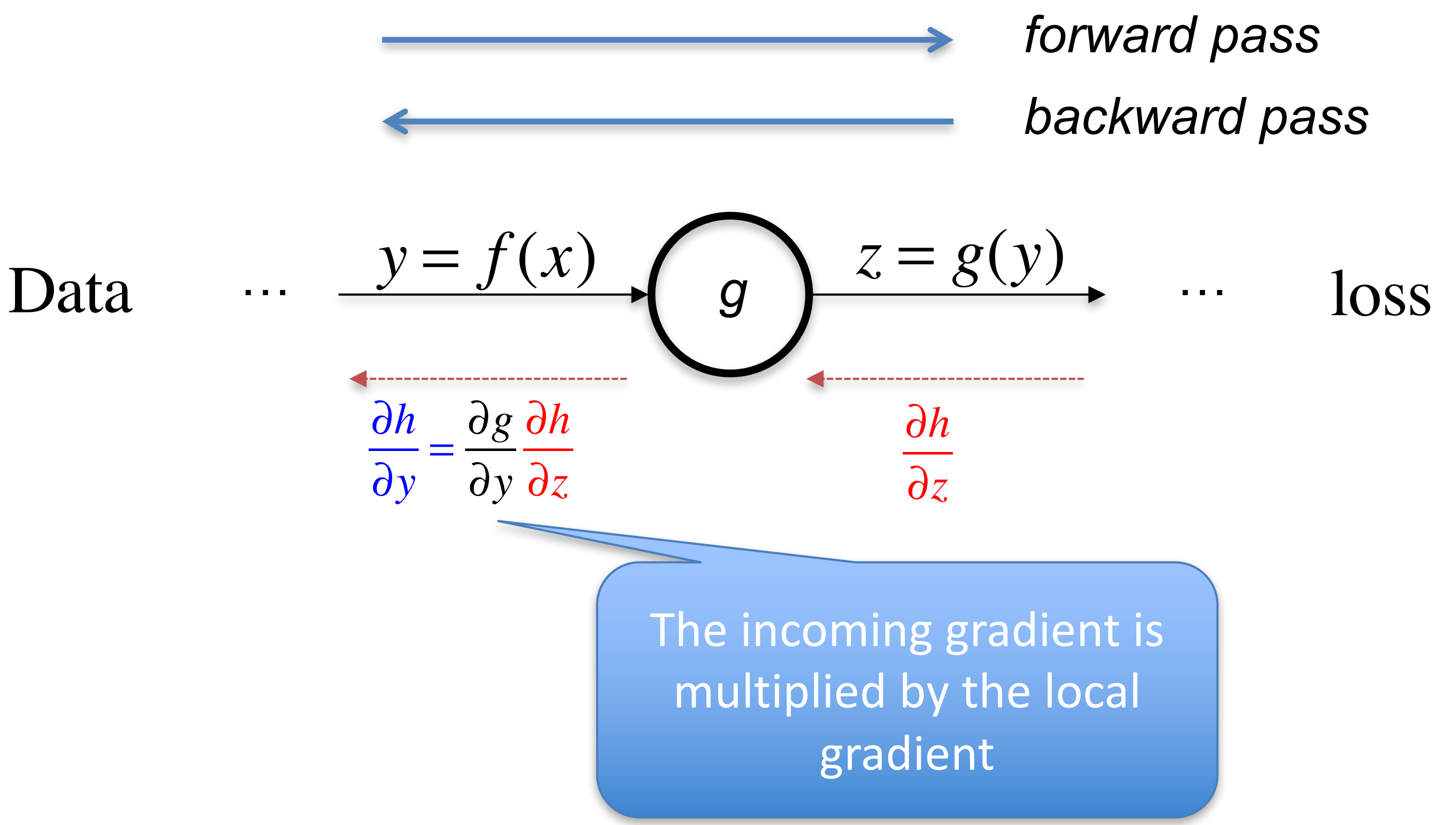

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

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

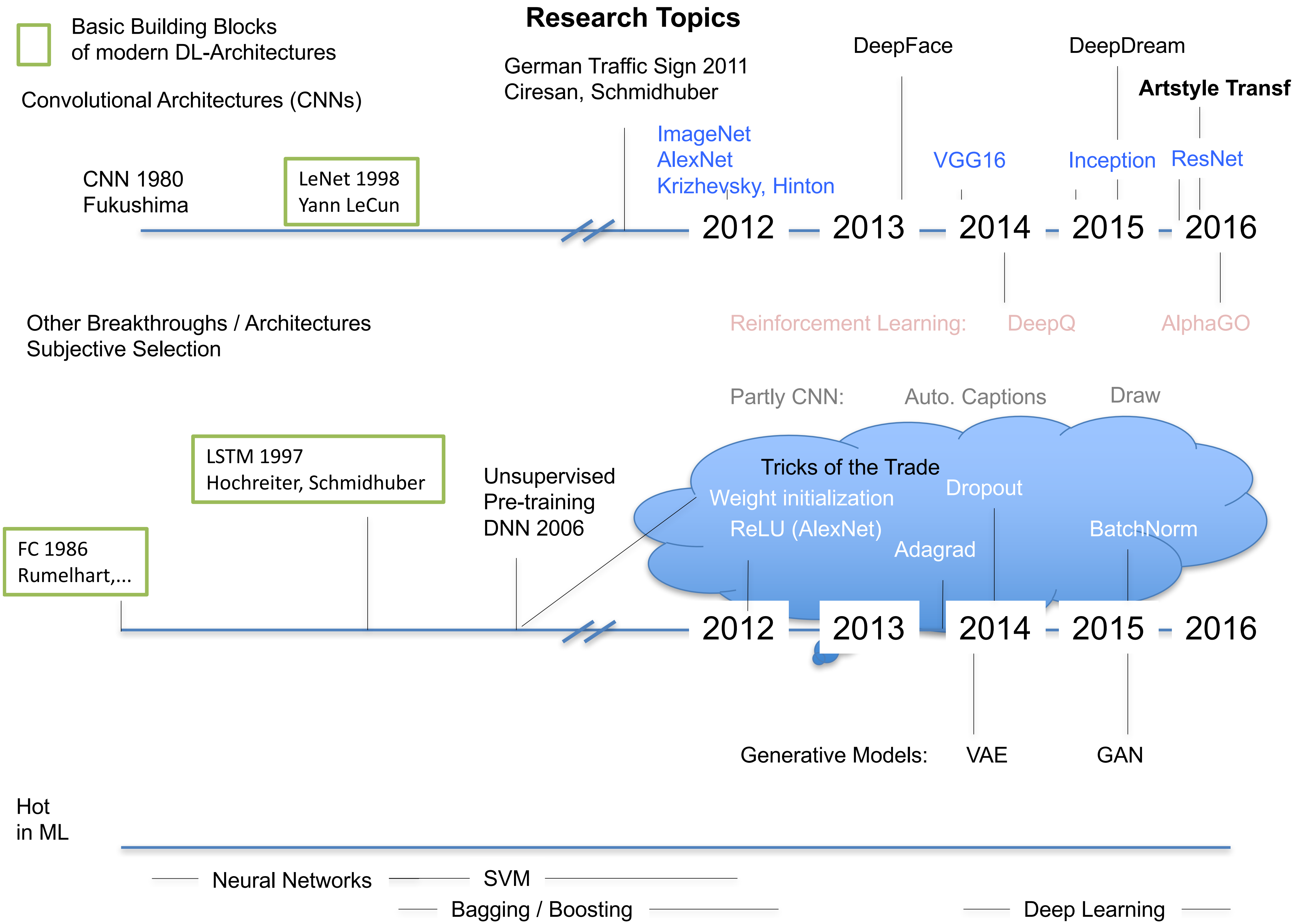
Further References / Summary

- 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!



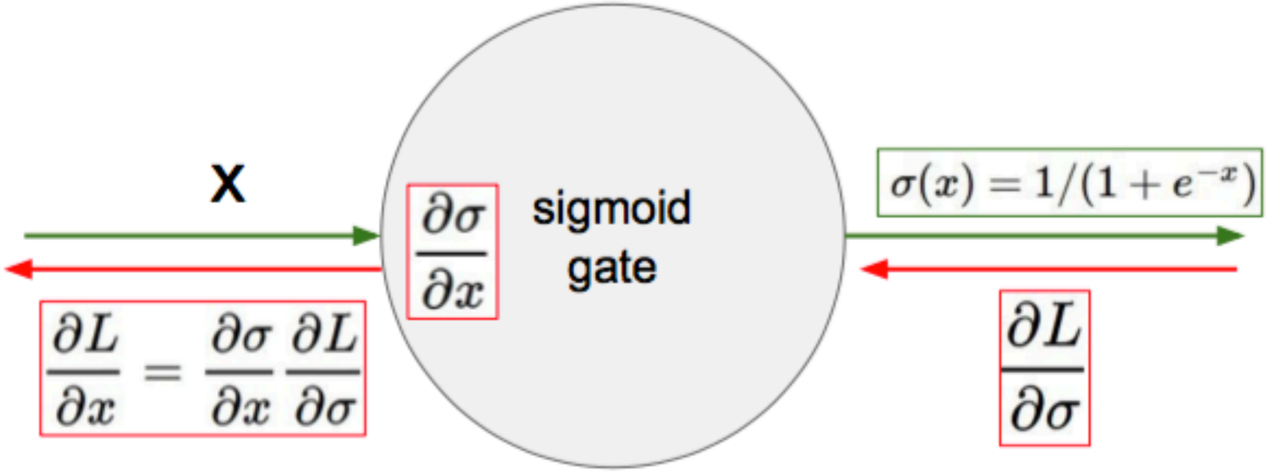
Tricks of the trade

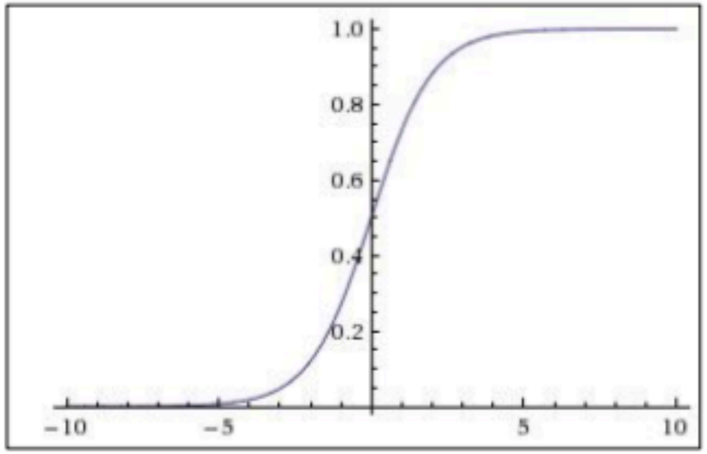


Research Topics
Basic Building Blocks of modern DL-Architectures
Convolutional Architectures (CNNs)
CNN 1980 Fukushima
LeNet 1998 Yann LeCun
German Traffic Sign 2011 Ciresan, Schmidhuber
ImageNet AlexNet Krizhevsky, Hinton
DeepFace
VGG16
DeepDream
Inception
Artstyle Transf
ResNet
2012 2013 2014 2015 2016
Reinforcement Learning:
DeepQ
AlphaGO
Other Breakthroughs / Architectures Subjective Selection
Partly CNN:
Auto. Captions
Draw
LSTM 1997 Hochreiter, Schmidhuber
FC 1986 Rumelhart,...
Unsupervised Pre-training DNN 2006
Tricks of the Trade
Weight initialization
Dropout
ReLU (AlexNet)
Adagrad
BatchNorm
2012 2013 2014 2015 2016
Generative Models:
VAE
GAN
Hot in ML
Neural Networks
SVM
Bagging / Boosting
Deep Learning

Activation Functions

Backpropagation through sigmoid





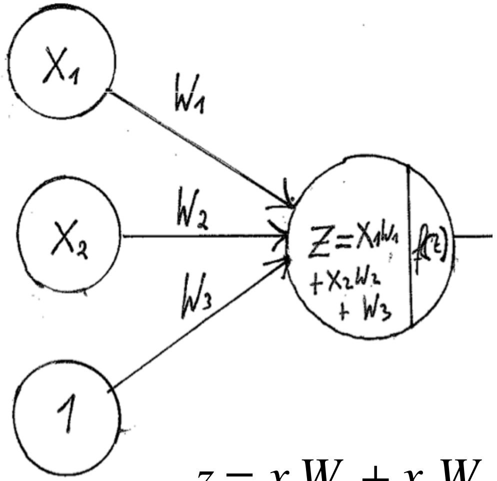
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

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

Slide from: CS231

Different activations in inner layers

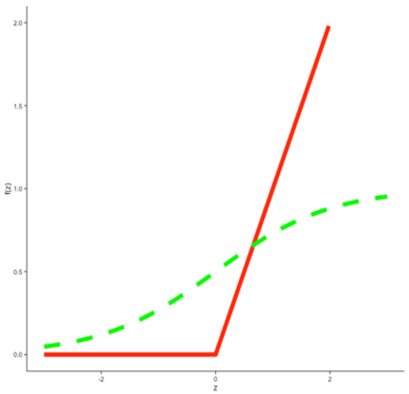
N-D log regression



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

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

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



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

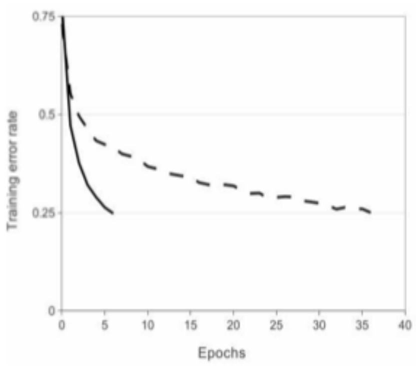


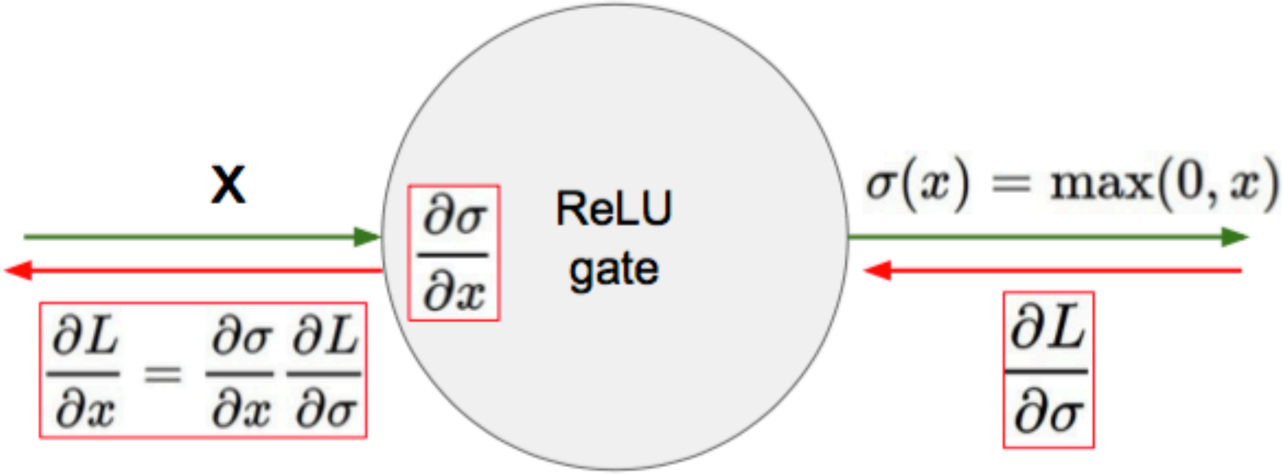
Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

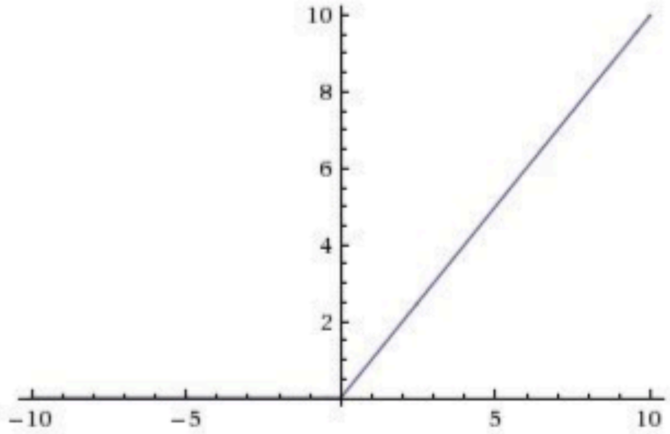
Source:
Alexnet
Krizhevsky et al 2012

There are other alternatives besides sigmoid and ReLU.

Currently ReLU is standard

Backpropagation through ReLU



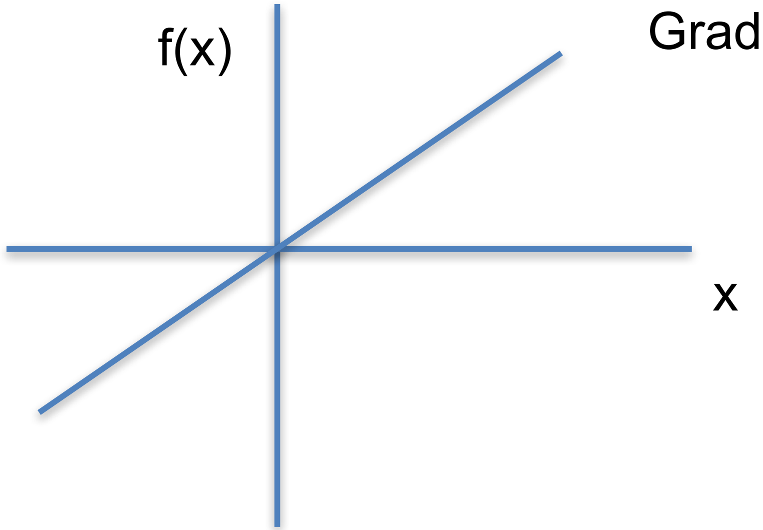


What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

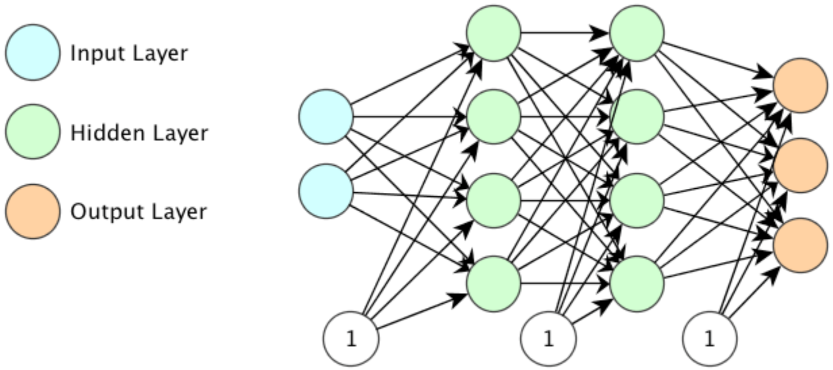
Gradients are killed, only when x < 0

An activation which never gets killed…

- Why just don't take identity?



Gradient is always 1



$$p = \mathrm{softmax}(x^{(1)} f(W^{(1)}) f(W^{(2)}) f(W^{(3)}))$$

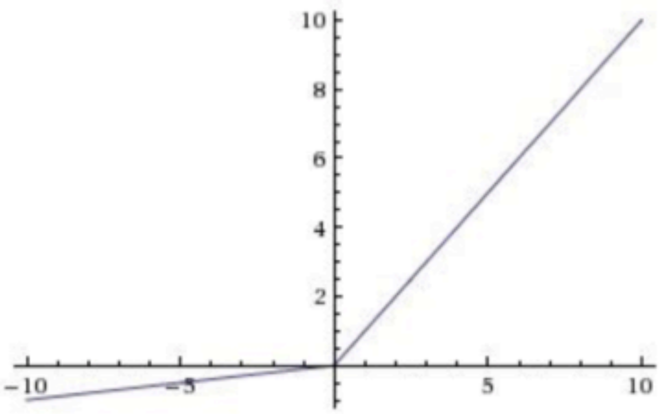
$$x^{(1)} W^{(1)} W^{(2)}\ W^{(3)} = x^{(1)} W$$

If you multiply two matrices $A \cdot B$ you get a new matrix.

Other activations

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die".**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into \alpha (parameter)

Not really established

Slide from: CS231

Initialization

Initialization of weights: Experiment

Layer (type)	Output Shape	Param #	Connected to
dense_1 (Dense)	(None, 100)	78500	dense_input_1[0][0]
dense_2 (Dense)	(None, 100)	10100	dense_1[0][0]
dense_3 (Dense)	(None, 100)	10100	dense_2[0][0]
dense_4 (Dense)	(None, 100)	10100	dense_3[0][0]
dense_5 (Dense)	(None, 100)	10100	dense_4[0][0]
dense_6 (Dense)	(None, 100)	10100	dense_5[0][0]
dense_7 (Dense)	(None, 100)	10100	dense_6[0][0]
dense_8 (Dense)	(None, 100)	10100	dense_7[0][0]
dense_9 (Dense)	(None, 100)	10100	dense_8[0][0]
dense_10 (Dense)	(None, 10)	1010	dense_9[0][0]

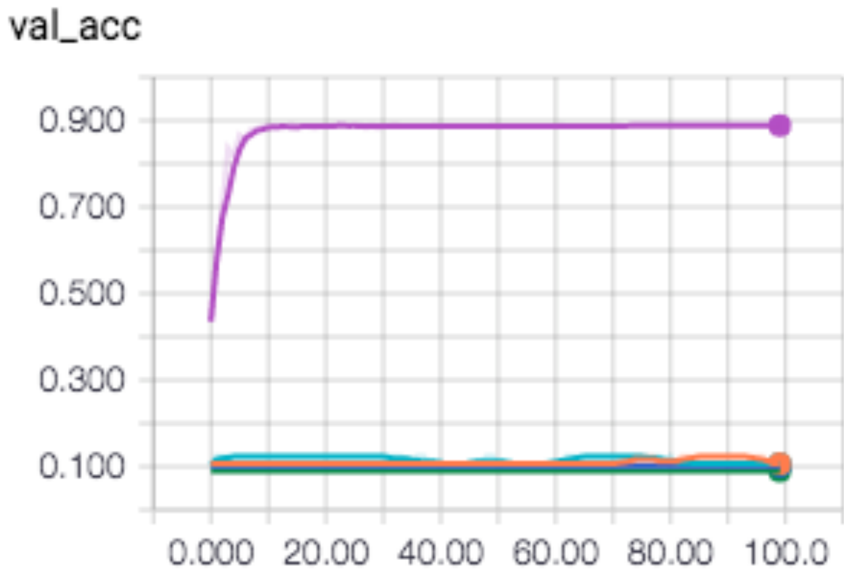
```
Total params: 160,310
Trainable params: 160,310
Non-trainable params: 0
```

Weights are initialized with N(0, sigma)

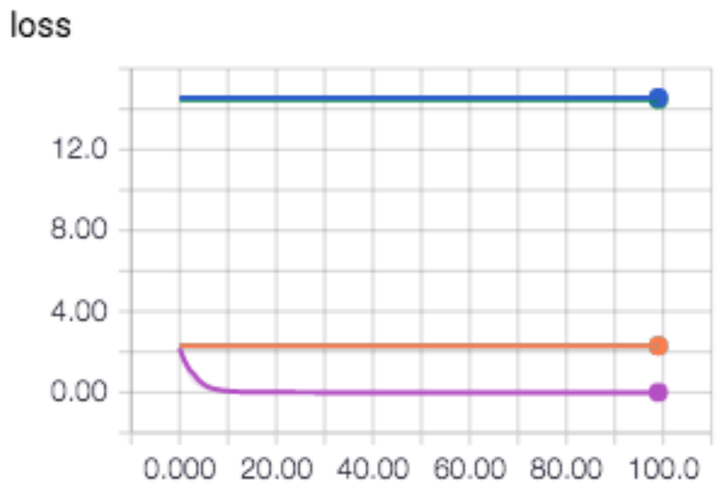
See: https://github.com/tensorchiefs/dl_course/blob/master/notesbooks_misc/weight_initialization.ipynb

Different Initialization: Performance

Name	Smoothed
relu_0.001	0.1067
relu_0.01	0.1233
relu_0.1	0.8855
relu_1.0	0.1017
relu_10	0.09000



Name	Smoothed
relu_0.001	2.300
relu_0.01	2.300
relu_0.1	0.01836
relu_1.0	14.57
relu_10	14.48



Learning happens only for sigma=0.1! Random loss is –ln(1/10)=2.302

Reason for not learning

- Activation values vanished or explode
- No learning since gradient is also vanishing
  - Grad ~ x and thus also near 0

- Historical anecdote
  - Deep Learning started 2006 when Hinton et. all managed to train deep networks unsupervised pre-training
  - Later it turned out that random initialization with the same weight would yield similar results

- For ReLU: He et al., http://arxiv.org/abs/1502.01852
  - sigma = np.sqrt(2. / fan_in)
  - fan_in number of incomming weights (100 in our example)
  - bias to zero

Regularization

Regularisation

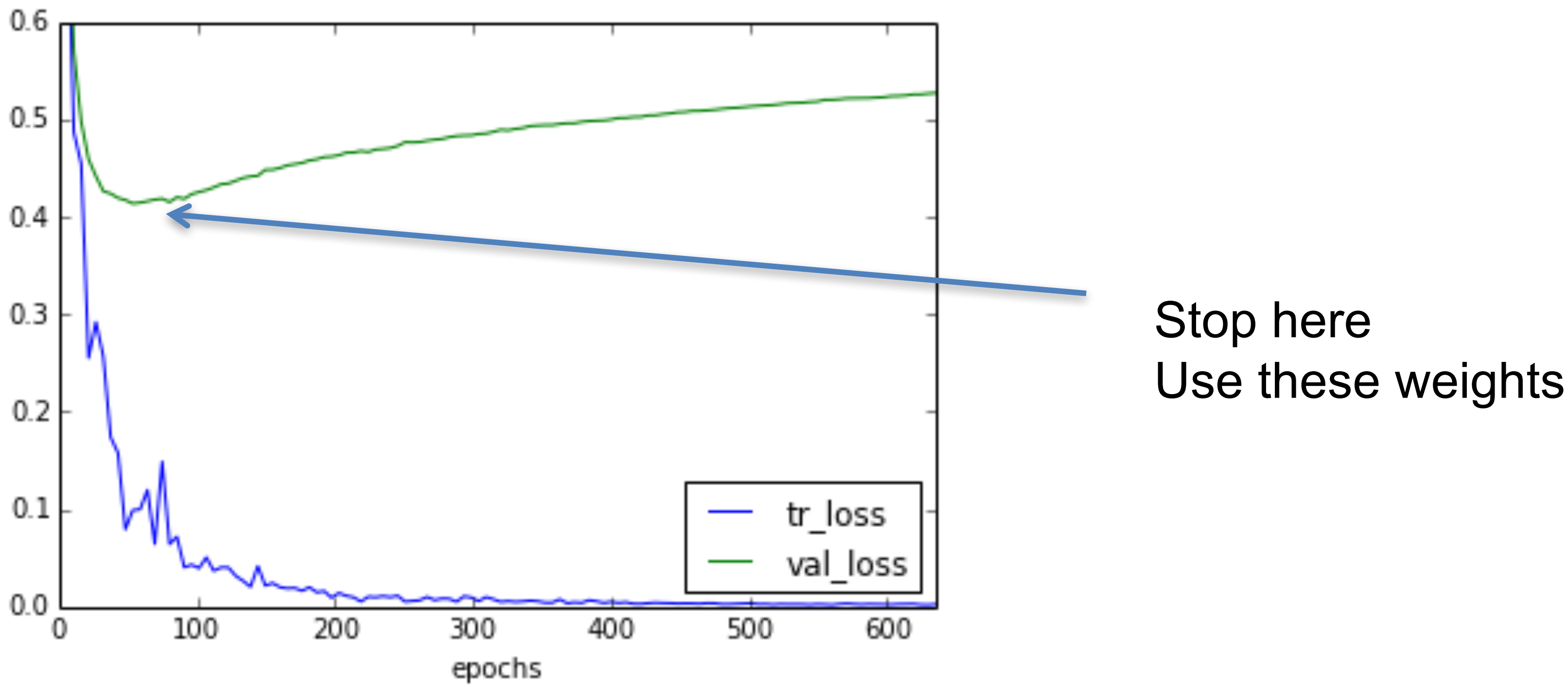
Having more parameters than examples ➔ overfitting becomes a real problem

Several solutions (selection, for complete treatment DL-book chapter 7)

- Early stopping
- Dropout
- Not covered today
  - Penalties on parameter norm (L1, L2 a.k.a. weight decay)
  - Parameter tying and sharing (in the next lectures)
    - Very powerful for special domains
      - Time signals ➔RNN
      - Image like data ➔CNN
  - Dataset Augmentation (in CNN lecture)
  - Semi-supervised learning (use unlabelled data)

Early stopping

- Simply stop (or use the parameters of the network) when validation loss is minimal (hope for the best for the test-set)



- In practice
  – Needs a validation set not used to update the weights
  – Save model weights at different epochs (*checkpoints*)
  – Plot and decide which checkpoint to use (or continue training)

Early stopping (intuition)

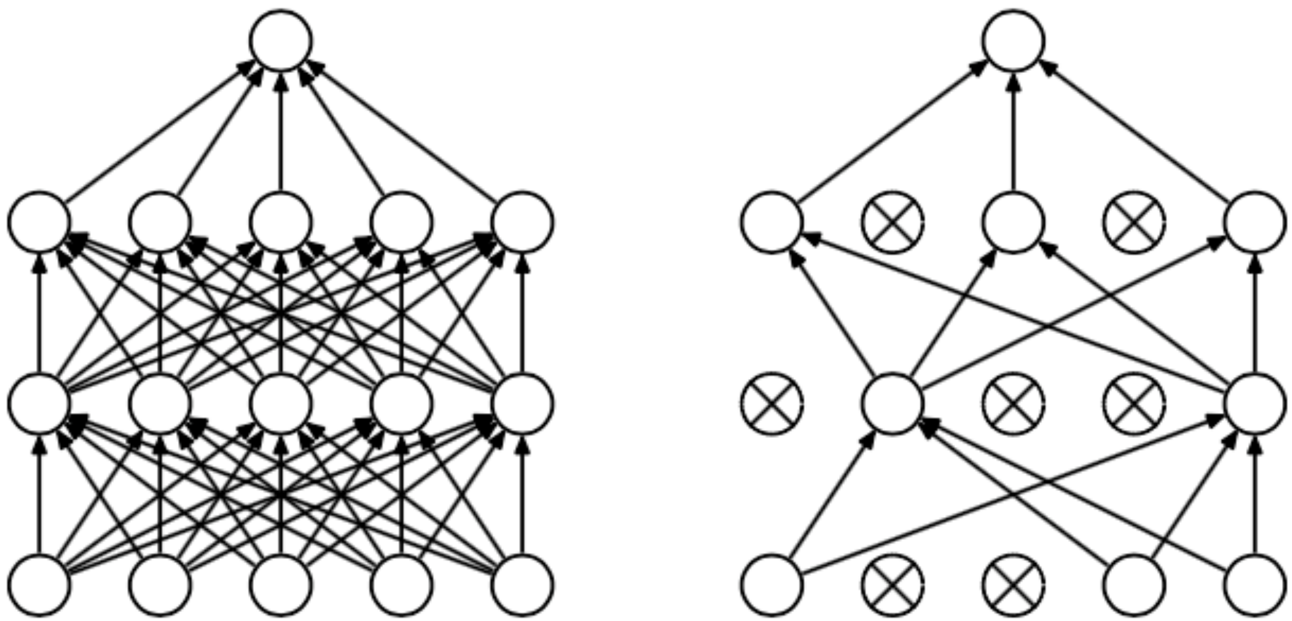
- Early stopping can be seen as a from of regularization
- The optimization procedure cannot explore the whole parameter space
- Cannot adopt too much on the training set

Dropout

- Dropout is a simple and relatively recent regularization technique (Srivastava et al. 2014) which is already widely used.
- It forces the network to learn redundant features
- It averages over many networks

Dropout during training.

Technically: add a layer killing neurons with prob. p



(a) Standard Neural Net

(b) After applying dropout.

Figure: from paper

Dropout: training / testing

- At test time we (usually) want deterministic predictions
  - Later in the course we use them to make stochastic predictions
- Weights (connections) need to be downweighted by *p*
  - During training the connections have not been present with prob. p, they would thus be too strong if always present in test time

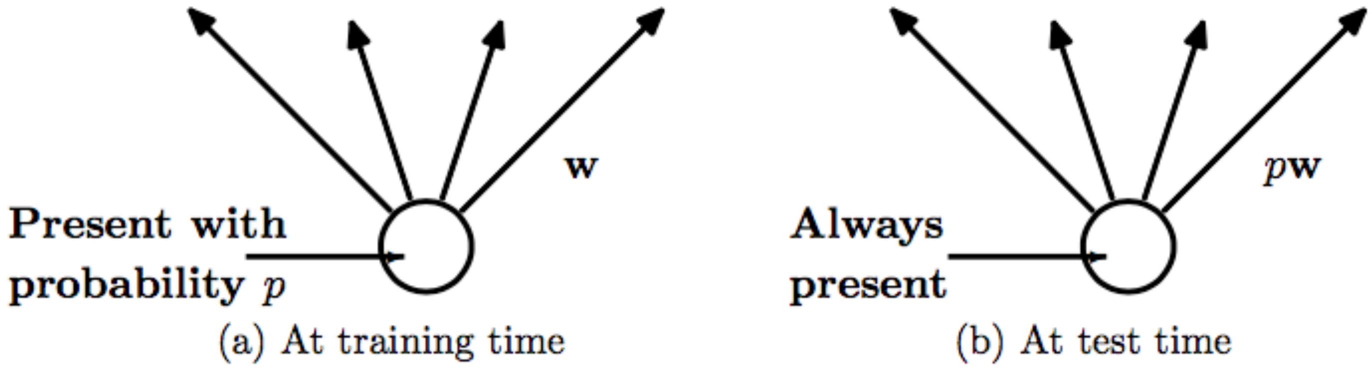
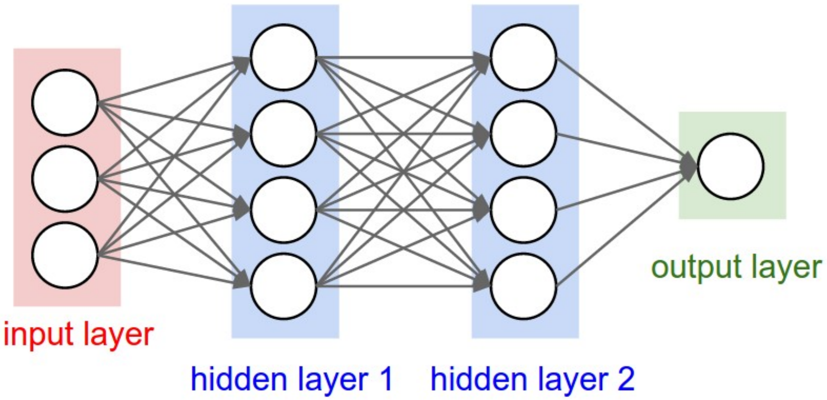


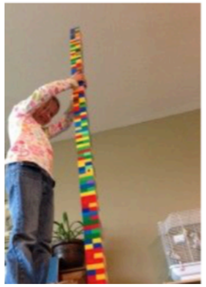
Figure: from paper

- Alternative approach (inverted dropout)
  - Upweight the weights by W/p during training (see also: http://cs231n.github.io/neural-networks-2/)
  - No scaling needed at test-time

Higher level libraries

- Including all the logging and regularisation would require to write lot of code
- There is a multitude of libraries (currently too many!) which help you with training and setting up the networks
- Libraries make use of the Lego like block structure of networks





Have a look at

https://github.com/tensorchiefs/dl_course_2018/blob/master/docs/keras-short-intro.pdf

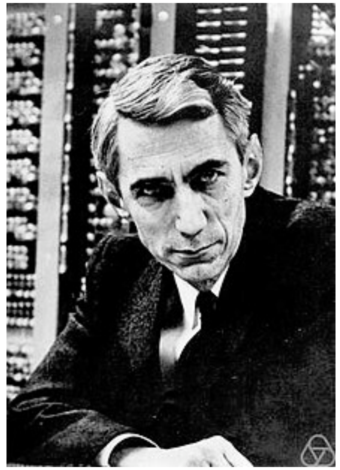
Example in Keras

```
model = Sequential() #We start to build the model in a sequence
model.add(Dense(500, batch_input_shape=(None,
784),activity_regularizer=activity_l2(lambd)))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(50,activity_regularizer=activity_l2(lambd)))
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax',activity_regularizer=activity_l2(lambd)))
# Finishing
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

# Training
history = model.fit(X[0:2400],
          convertToOneHot(y[0:2400],10),
          nb_epoch=500,
          batch_size=128,
          #callbacks=[tensorboard],
          validation_data=[X[2400:3000], convertToOneHot(y[2400:3000],10)])
```

Backup

Why the hack they call it cross entropy?

Entropy and Cross Entropy



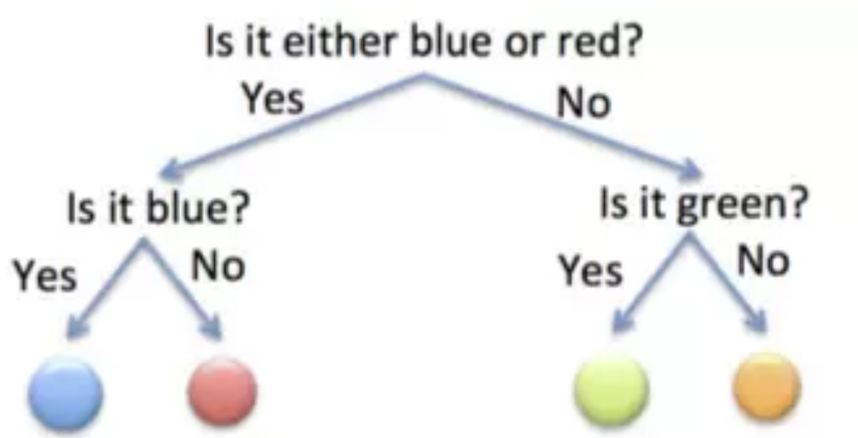
- The central loss function for classification is called cross entropy, why?
- This is a different viewpoint to the max-likelihood approach, we just had
- Let's start by defining the (information) entropy
  - It's somewhat like the amount of surprise you get from a sample.
  - Let's first do an simple example

Image: Wikipedia

Information Content of a single outcome

- 4 Balls each with same probability 25%

- How can your friend ask you which ball you picked, with minimum number of questions?



Let's say we have a red ball.
Two questions need to be ask.

Coding for red ball (yes=1)
10 // Information content 2 bits

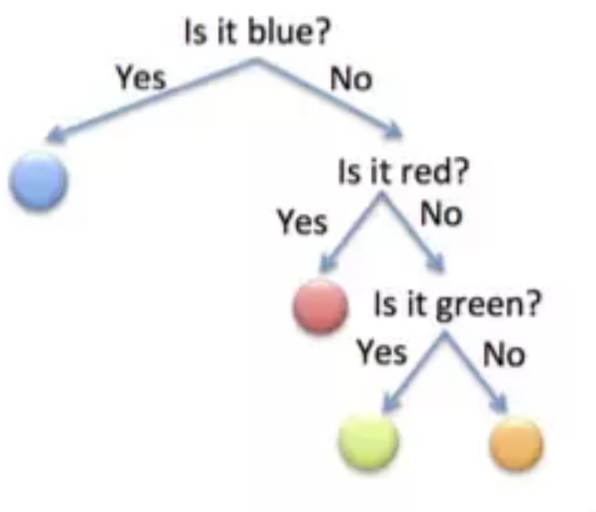
Coding for orange (your turn)
00 // Information content 2 bits

Example form https://www.quora.com/Whats-an-intuitive-way-to-think-of-cross-entropy

Information Content of a single outcome

- 4 Balls each with different probability 50%, 25%, 12.5%, 12.5%

- How can your friend ask you which ball you picked, with minimum number of questions (on average)?



Let's say we have a blue ball.
One questions need to be ask.

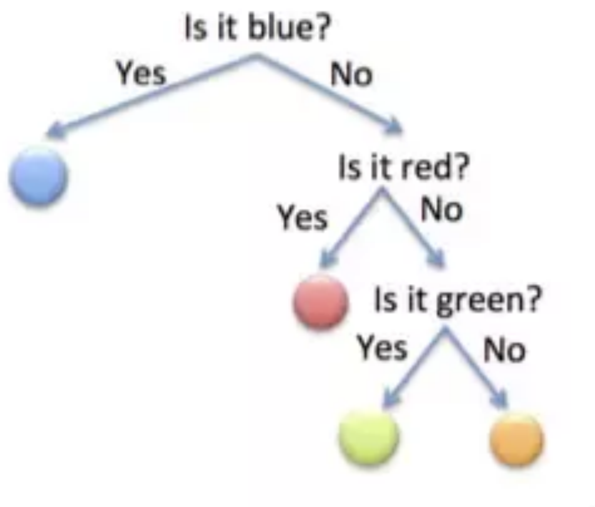
Coding for blue ball (yes=1)
1 // Information content 1 bit

Coding for red (2 questions)
01 // Information content 2 bit

Coding for green (your turn)
001 // Information content 3 bit

Example form https://www.quora.com/Whats-an-intuitive-way-to-think-of-cross-entropy

Information content



On average:
½*1+1/4*2+1/4*3=1.75 bits on average

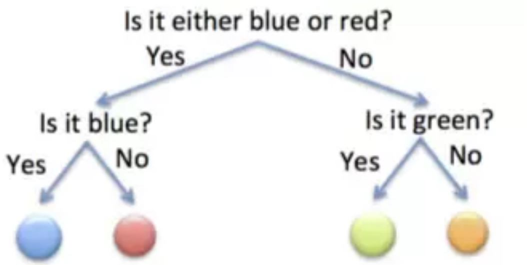
Information Content

- For that easy example, we found the best coding by hand.
- Let's define the (self-) information (Turns out to be the minimal coding length "Shannon's source coding theorem")

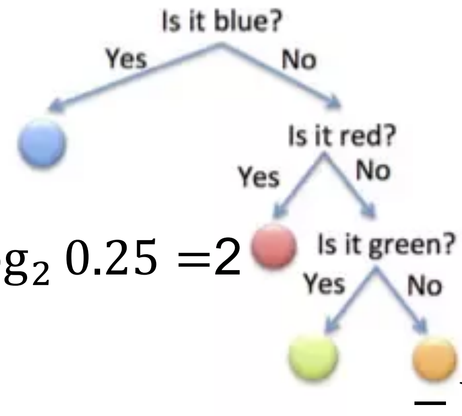
- Requirement for Information (or surpirse)
  - p_i the probability of event i (or prob. that symbol i occurs)
  - Seldom examples should have more surprise.
    - $I(p_i)$ should be monotonic decreasing function
  - Information should be non-negative
    - $I(p_i) \geq 0$
  - Uninformative, or sure events should have no Information
    - $I(p_i) = 0$
  - Information of independent events i, j should add up
    - $I\left(p_{(i,j)}\right) = I\left(p_i p_j\right) = I(p_i) + I(p_j)$
- ➔ $\boldsymbol{I(p) = -\log_2(p)}$
  - (defined up to basis), 2 is often chosen

Information Content → Entropy

- Entropy (average Information Content)
  - $H(p) = \sum p_i I(p_i) = -\sum p_i \log_2(p_i)$



$-\log_2 0.25 = 2$

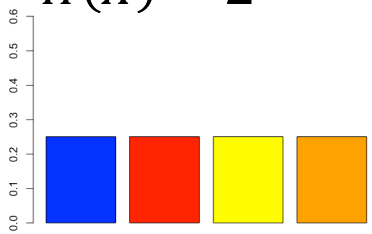


$-\log_2 0.5 = 1$

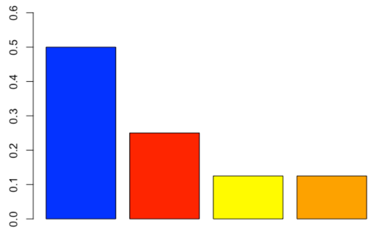
$-\log_2 0.25 = 2$

$-\log_2 0.125 = 3$

$H(X) = 2$



$H(X) = 0.5 + 0.25*2+0.25*3=1.75$



In general: Maximal Entropy if uniform, minimal if peaked (see also in physical Systems)

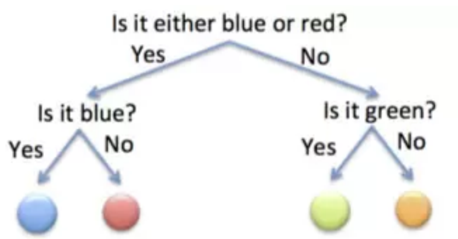
Cross Entropy

- If we know the distribution p, we can find the best coding and need H bits on average

- If we have a “wrong” distribution q how many bits do we need on average

  – $H(p,q) = -\sum p_i log2(q_i) \geq H(p)$

- Example, we think symbols come uniform distributed q. But they come (0.5,0.25,0.125,0.125)



Optimal Coding Scheme for Uniform q

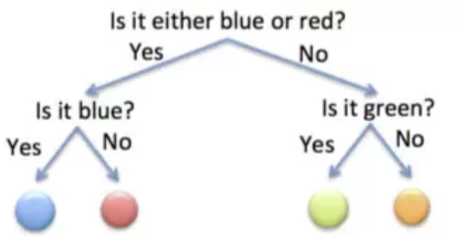
$$H(p,q) = 0.5\ 2 + 0.25\ 2 + 0.125\ * 2 + 0.125\ 2 = 2 > 1.75$$

KL-Divergence

- If we have a “wrong” distribution q how many bits do we have more than the minimal possible amount $H(p)$

$$-\ D_{KL}(p||q) = H(p,q) - H(p) \geq 0$$

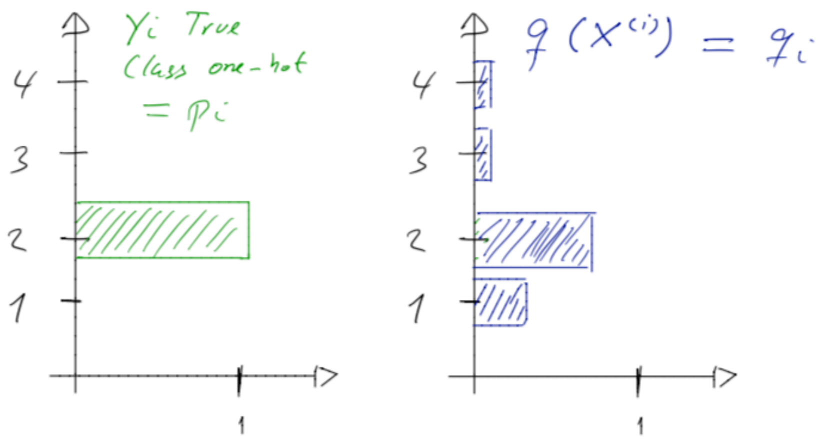
- Example, we think symbols come uniform distributed q. But they come (0.5,0.25,0.125,0.125)



Optimal Coding Scheme
for Uniform q

$$D(p,q) = H(p,q) - H(p) = 2 - 1.75 = 0.25$$

Cross Entropy in DL



$H(p,q) = -\sum p_i ln q_i$ (for one example of the training set)

$H(p,q) = -\sum\sum p_i^{(j)} ln q_i^j$ (for the training set)

We minimize the cross entropy by changing q, the minimum is reached when q is identical to distribution of real labels p

Alternatively we could also minimize the KL-Divergence

Further Resources (cross entropy and information theory)

- https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/
- https://www.quora.com/Whats-an-intuitive-way-to-think-of-cross-entropy
- https://www.khanacademy.org/computing/computer-science/informationtheory/moderninfotheory/v/information-entropy
- https://medium.com/swlh/shannon-entropy-in-the-context-of-machine-learning-and-ai-24aee2709e32