Machine Intelligence:: Deep Learning
Week 3

Oliver Dürr

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

Winterthur, 6. March. 2018

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/

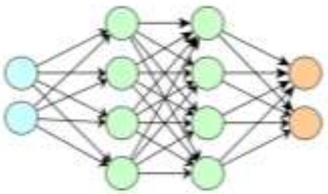
- GPU power:
  - We have a google grant and can reimburse your costs to use the google cloud infrastructure
- Please talk to us until week ~~4~~ → 5
- Q&A Session 1h in week 7

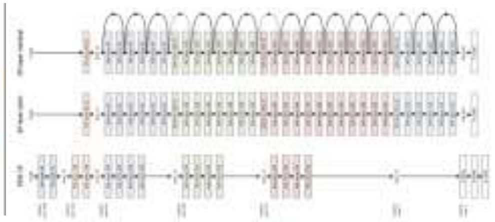
Organizational Issues: Times

- Next 4 times (total 30 minutes break in between, possible different breaks)
  - 09:00 – 10:30
  - 11:00 – 13:30

- Please interrupt us if something is unclear!

Learning Objectives



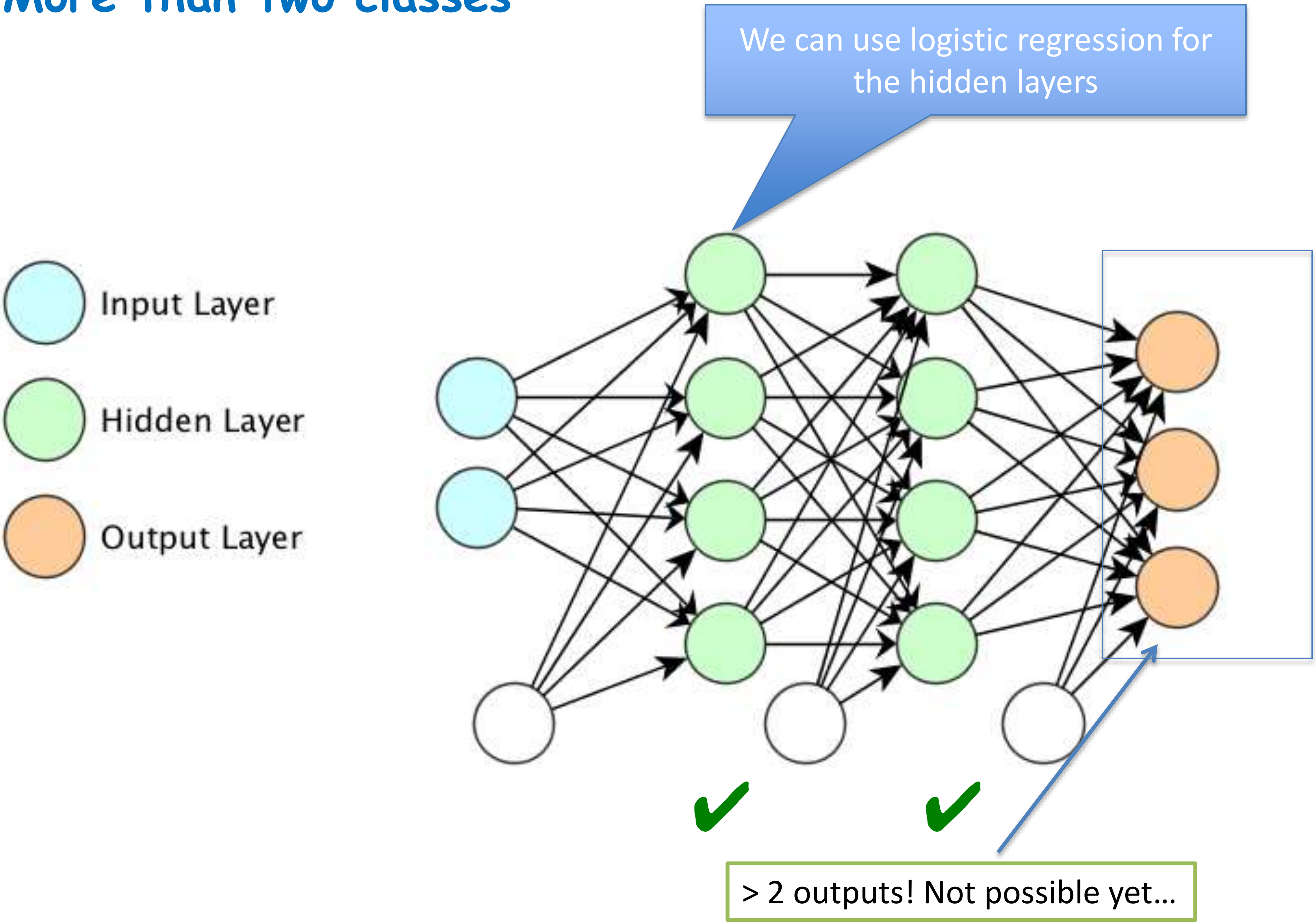


- Increase our knowledge in TF

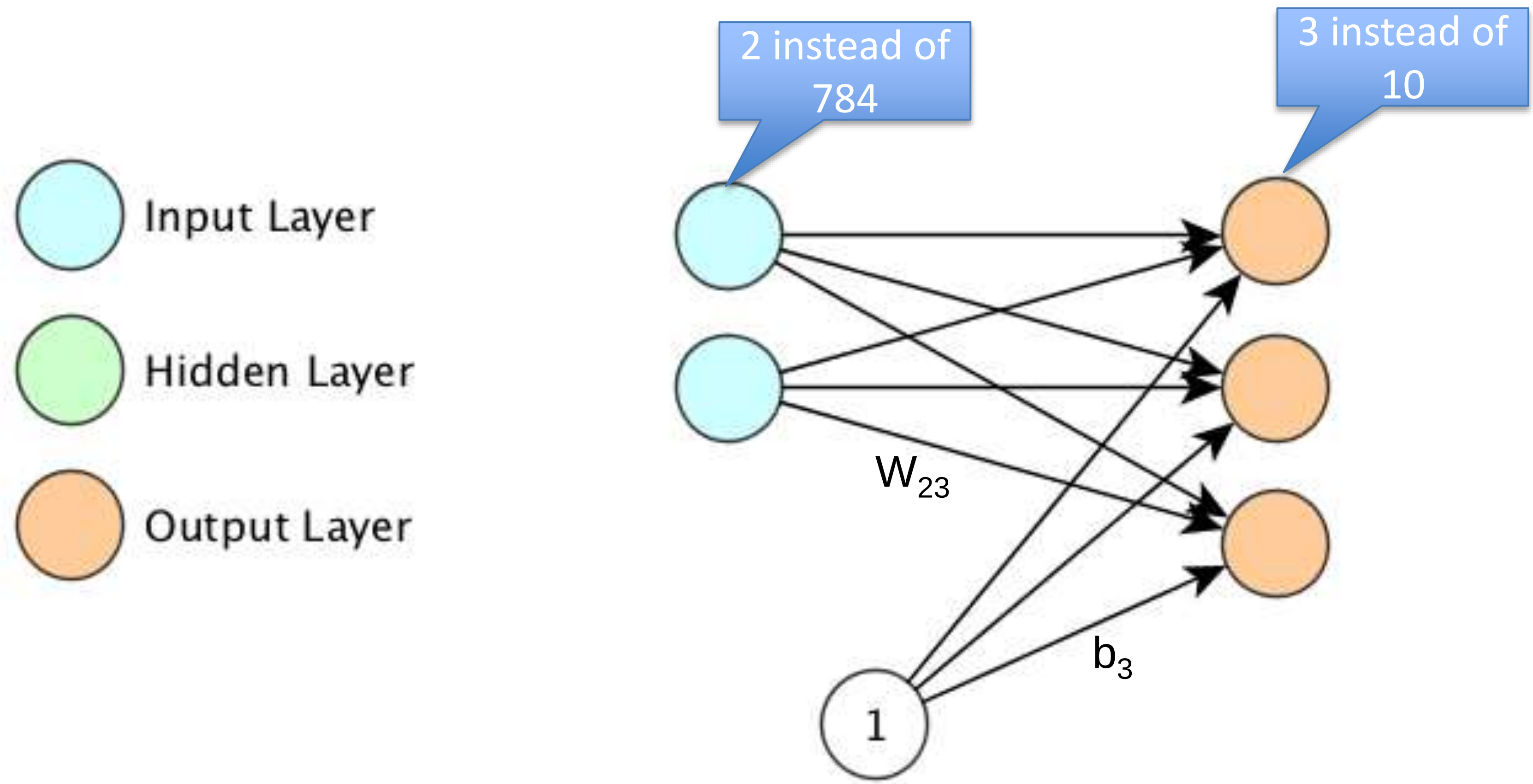
- Foundations of DL
  - **Loss Function (what to minimize)**
    - Cross entropy loss for multinomial logistic regression
    - Two principles to construct loss functions
      - Maximum Likelihood Principle
      - Cross Entropy
  - Deep Neural Networks
    - Fully Connected Networks with hidden layers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

Multinomial Logistic Regression

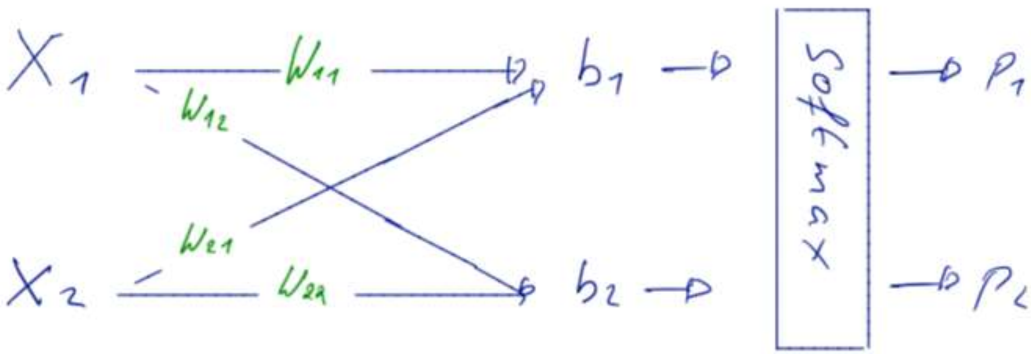
More than two classes



Multinomial Logistic Regression



GPUs love matrices (or tensors)

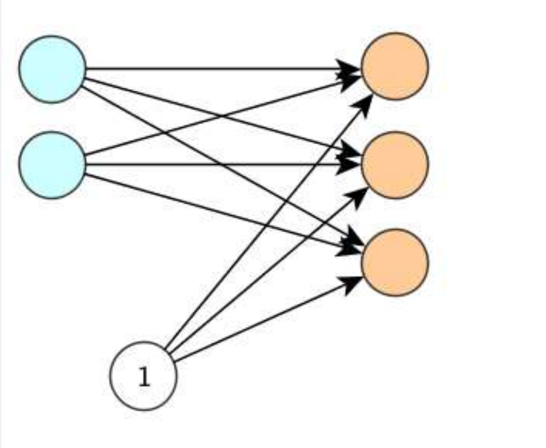


$$(P_1, P_2) = \text{softmax}\left(X_1 W_{11} + X_2 W_{21} + b_1,\ X_1 W_{12} + X_2 W_{22} + b_2 \right)$$

$$= \text{softmax}\left((X_1, X_2) \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix} + (b_1, b_2) \right)$$

$$P = \text{softmax}(X W + b)$$

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]])

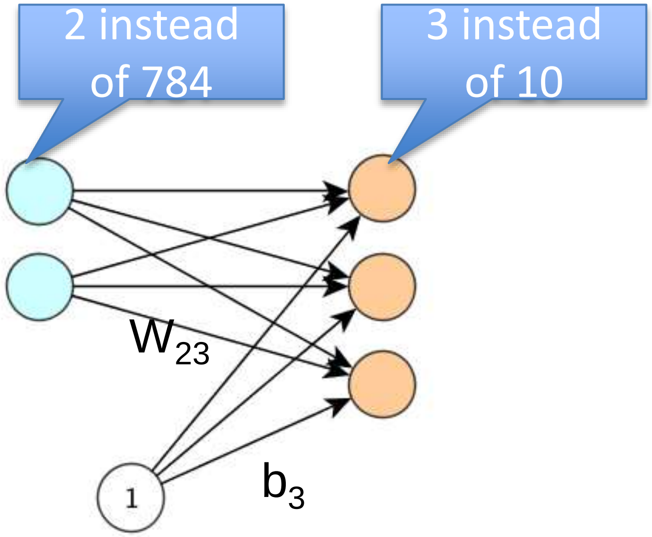
End of Recap

GPUs love matrices: Use the source luke

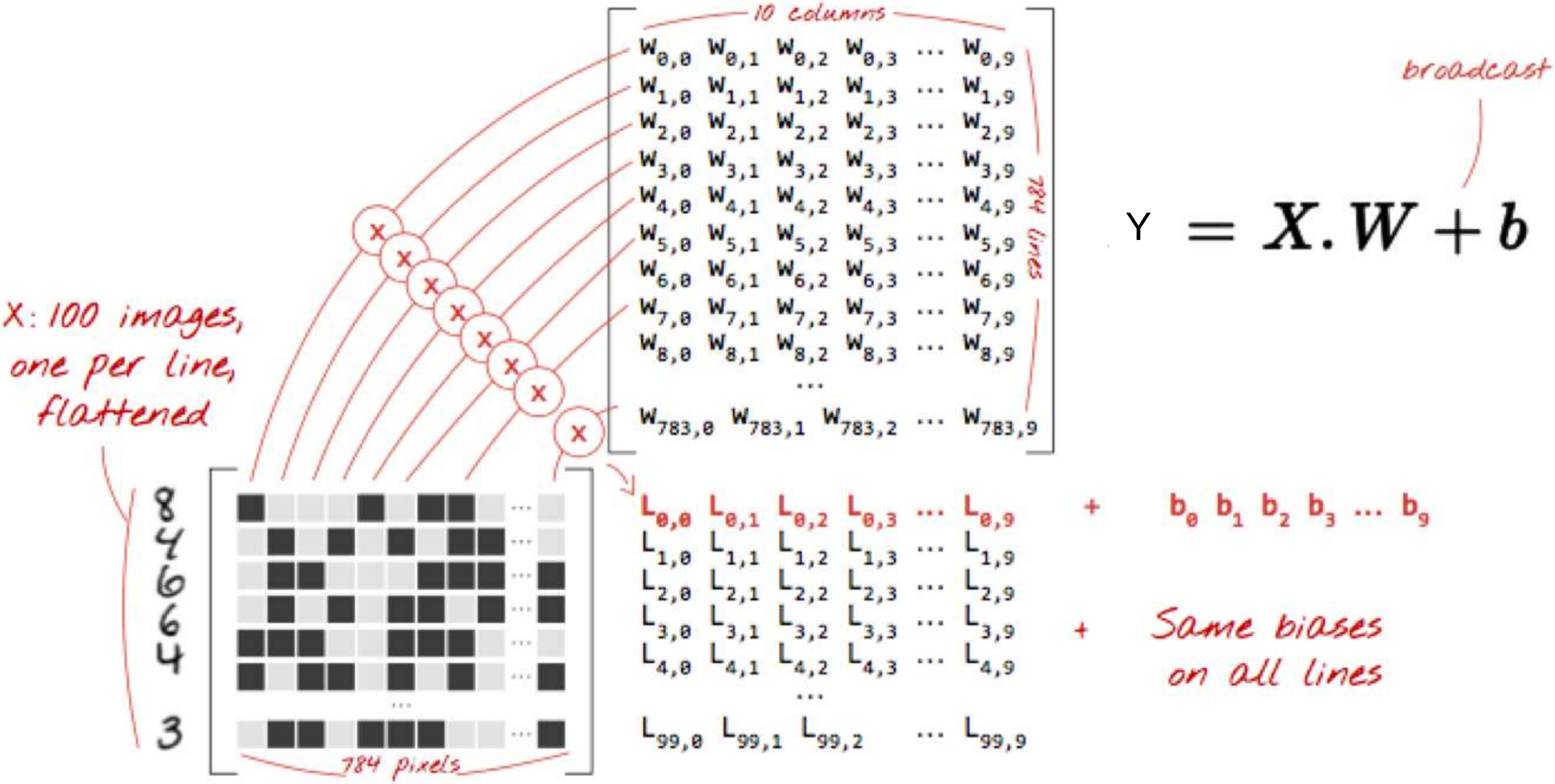
Mini batch size at runtime

```
…
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). x has shape [100, 784]



GPUs love matrices:



$$Y = X.W + b$$

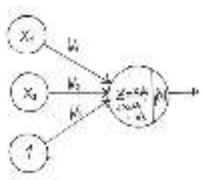
```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

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



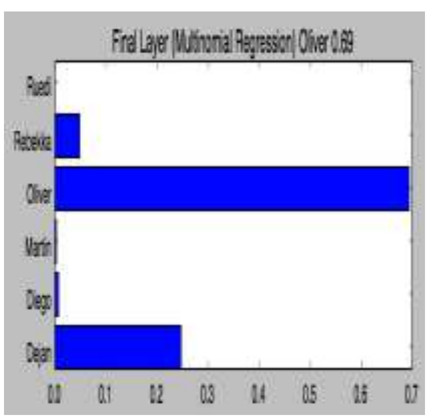
$$\text{loss} = -\frac{1}{N}\sum_{n=1}^{N} \log(p_{\text{model}}(y^{(i)} \mid x^{(i)}; q))$$

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

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

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

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.

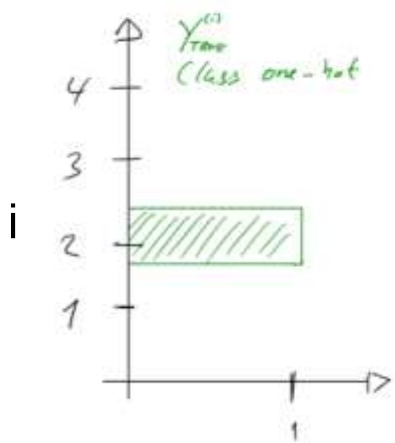
One more Trick: Loss function with indicator function

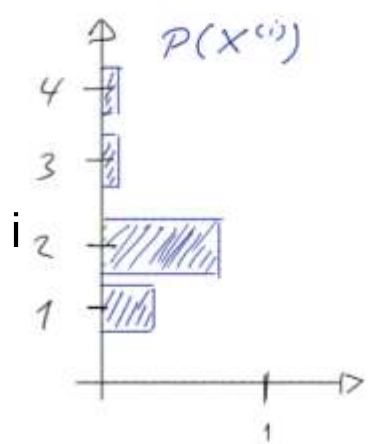
A one-hot-encoded y picks the right class, form all of the K different classes.

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

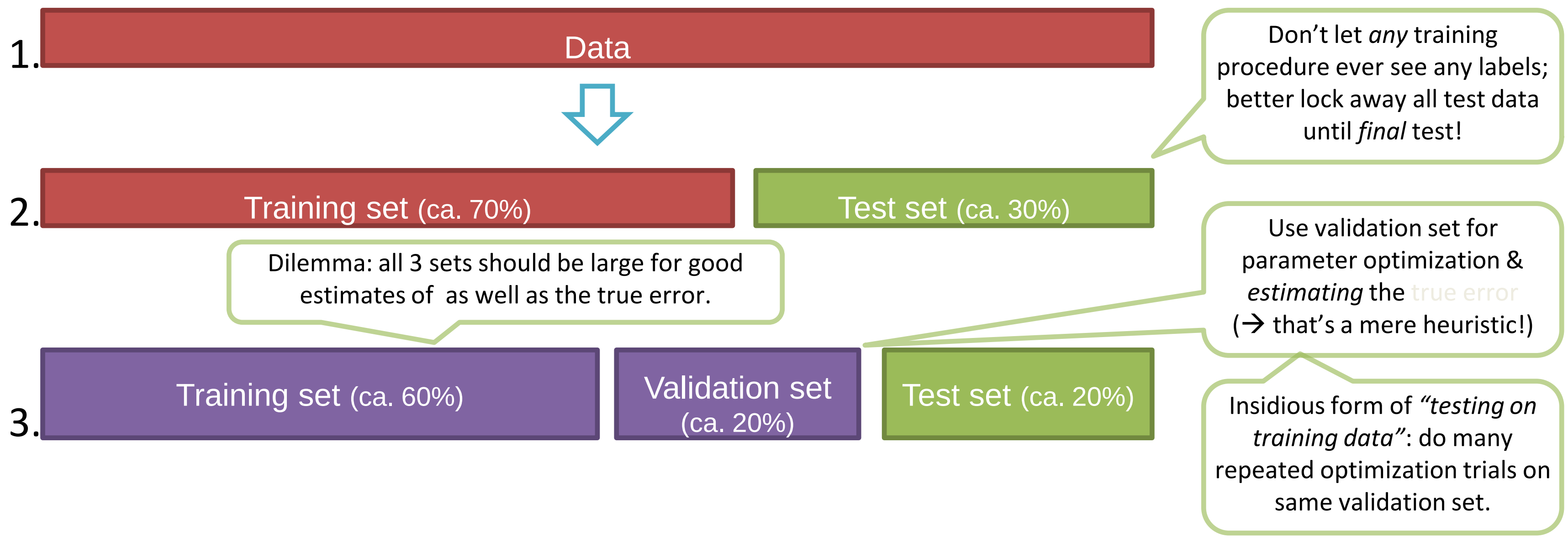




$$\text{Loss} = \frac{-1}{N} \sum_{i} Y^{(i)}_{True} \ln P(x^{(i)})$$

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

Training Neural Networks: Split of the data

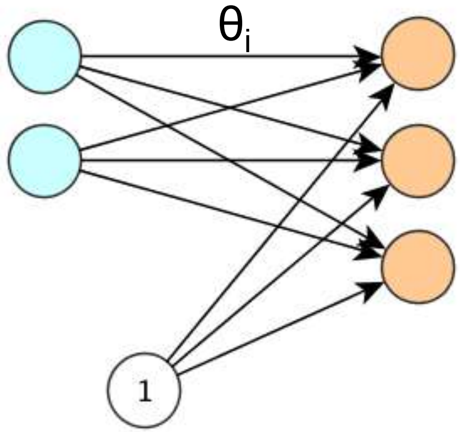


Taken from V03 ModelAssessment. For neural networks no cross-validation is done (long learning times).

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

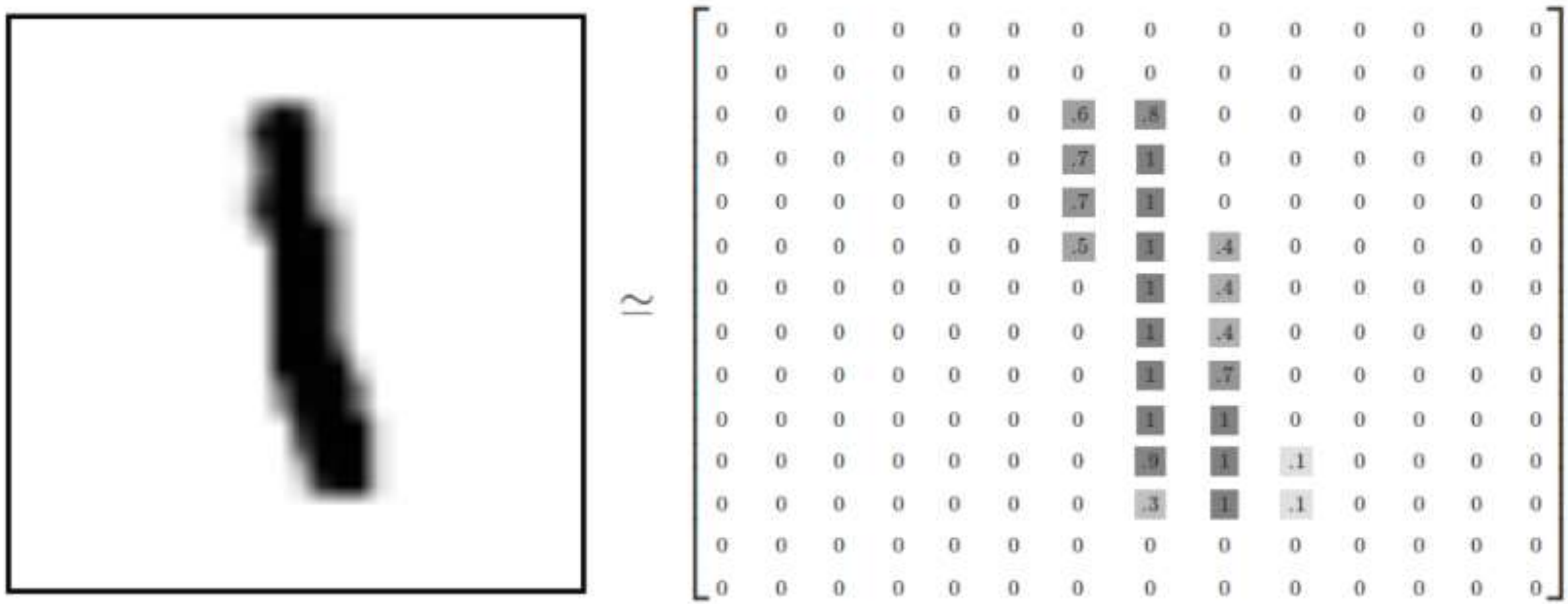
$$\text{loss} = -\frac{1}{N}\sum_{n=1}^{N} \log(p_{\text{model}}(y^{(i)} \mid x^{(i)}; \theta))$$

- 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. b=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 b=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

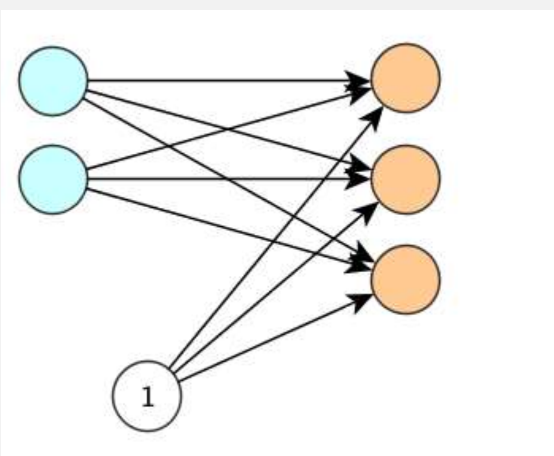
Image credit: https://www.tensorflow.org/versions/r0.10/images/MNIST-Matrix.png

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:



$$p_j = \frac{\exp(\sum_i x_i W_{ij} + b_j)}{\sum_j \exp(\sum_i x_i W_{ij})} = \left(\mathrm{softmax}(xW+b)\right)_j$$

Unten fehlt b

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](https://github.com/tensorchiefs/dl_course/blob/master/notebooks_misc/Explanation_o)

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)

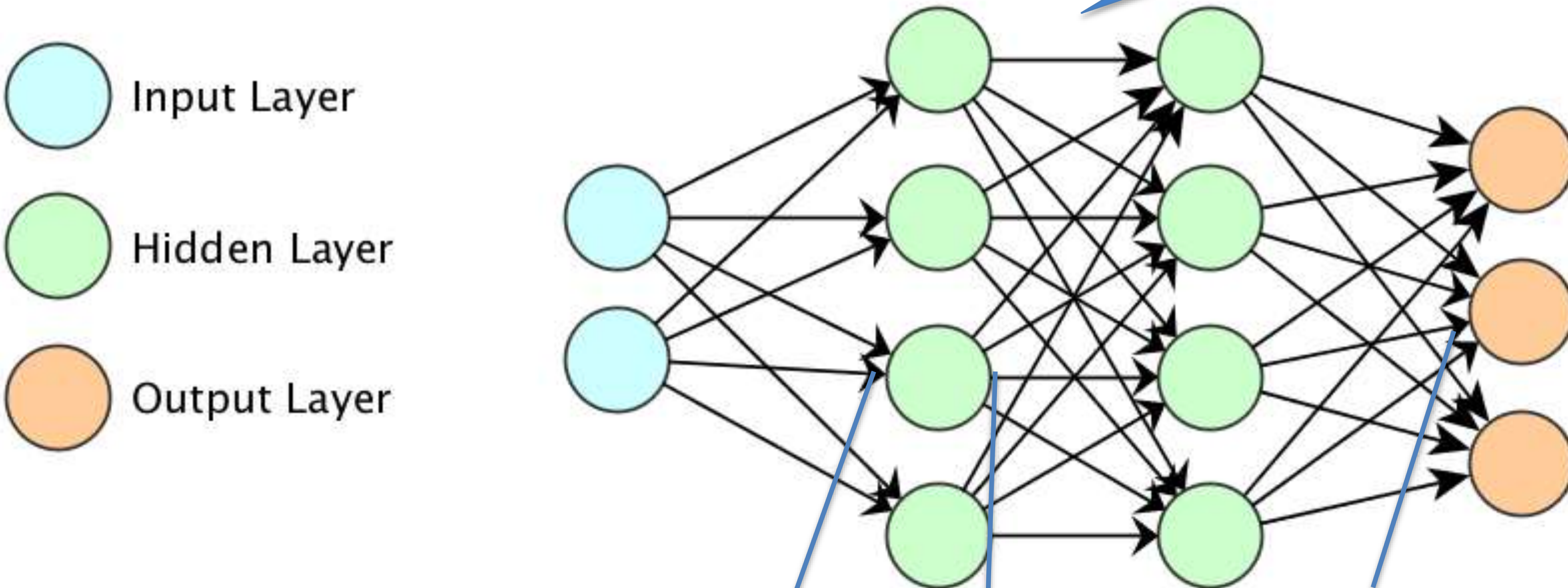
`tf.nn.sparse_softmax_cross_entropy_with_logits`

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

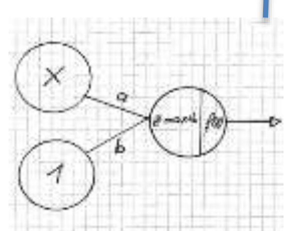

WE NEED TO GO
DEEPER
memegene

Today: Fully Connected Networks

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



We finished with....
Multinomial Logistic Regression



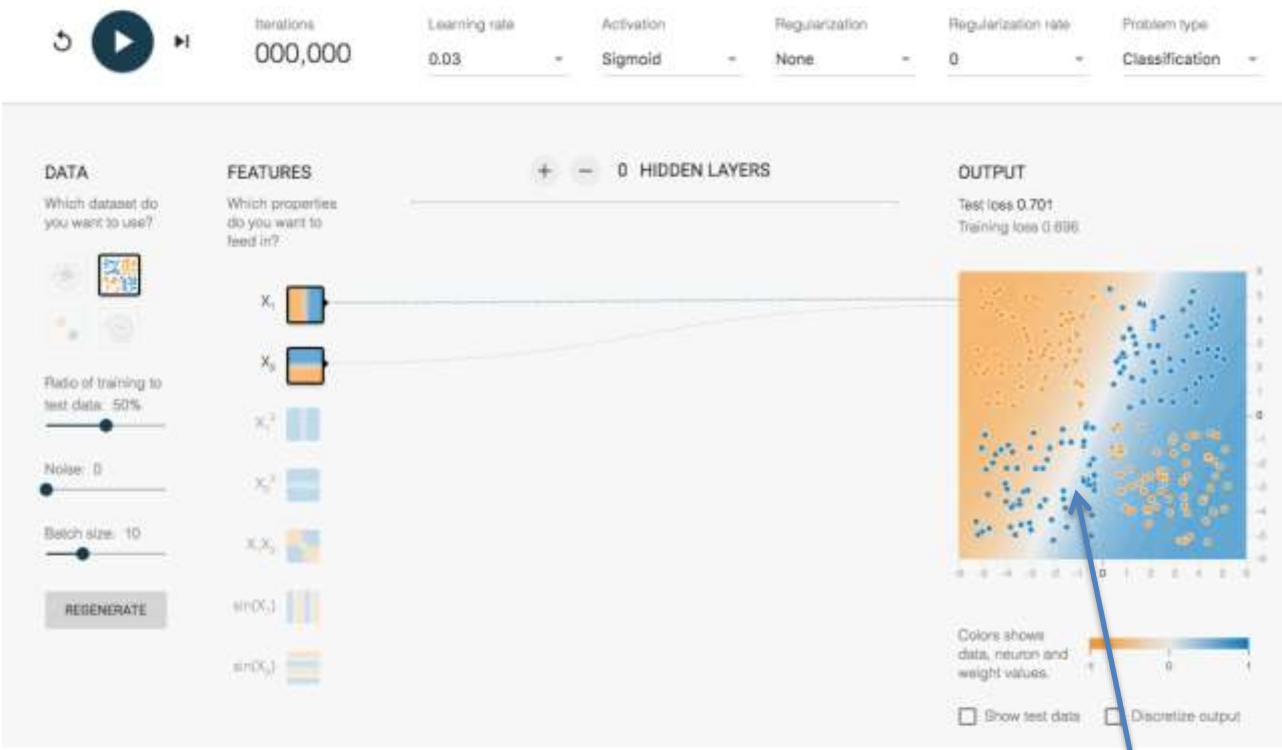
We started with....

1-D Logistic Regression

Networks with hidden layers

Limitations of (multinomial) logistics regression

Logistic regression in NN speak: “no hidden layer”



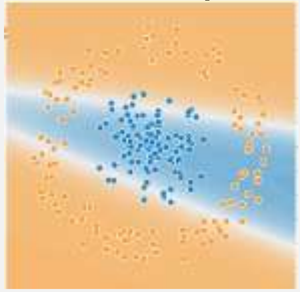
Linear Boundary!

[Network taken from](#)

Neural Network with hidden units

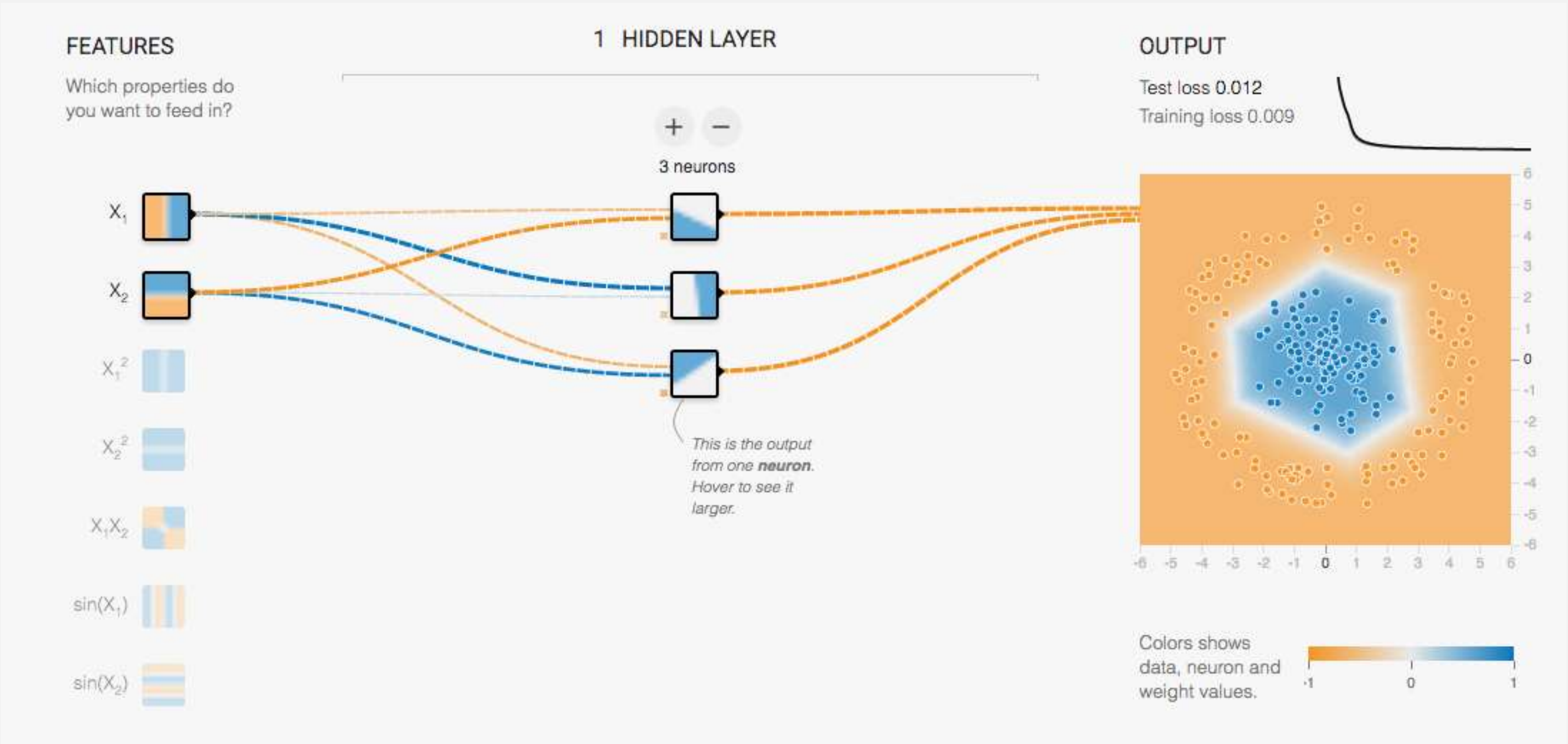
- Go to http://playground.tensorflow.org (https://goo.gl/VR3db5) and train a neural network for the data:

- 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?

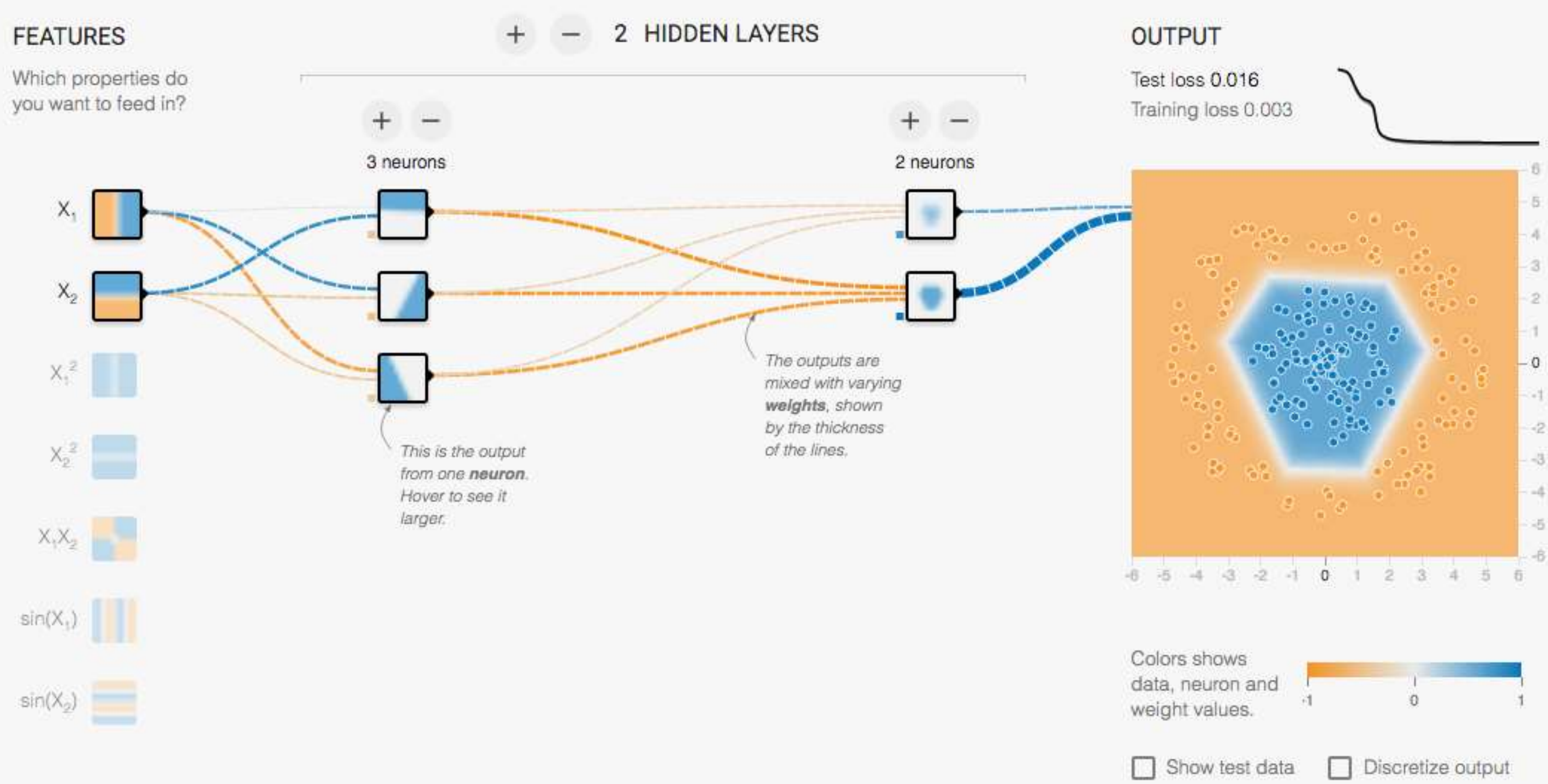


Results

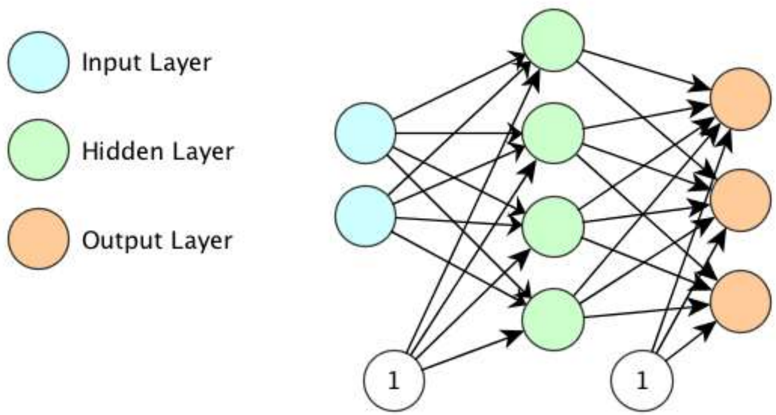
- 0 hidden layers, only a single line
- Many neurons in a hidden layer → also complicated functions



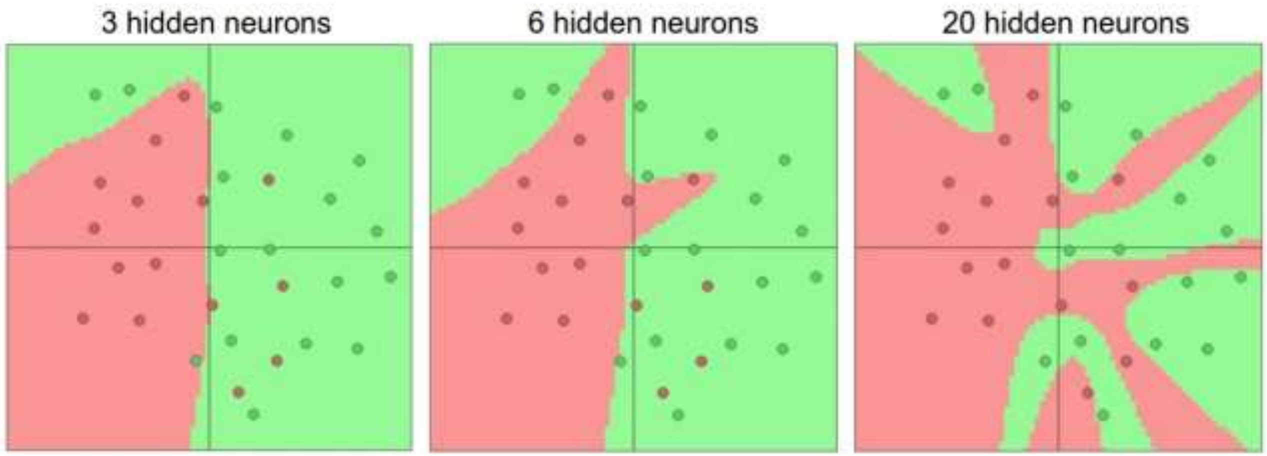
Results (cont)



One hidden Layer



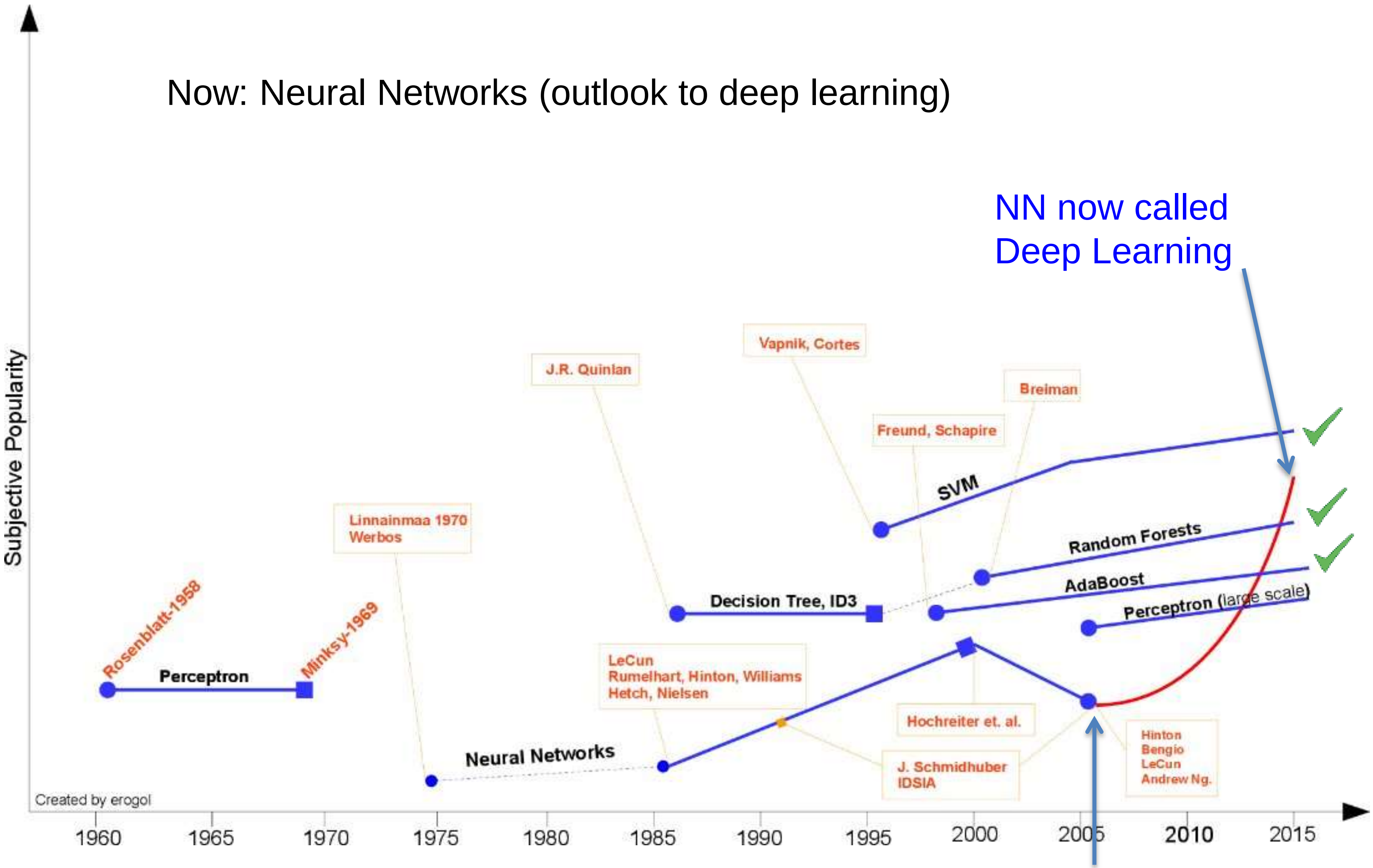
A network with one hidden layer is a universal function approximator!



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

Brief History of Machine Learning (supervised learning)

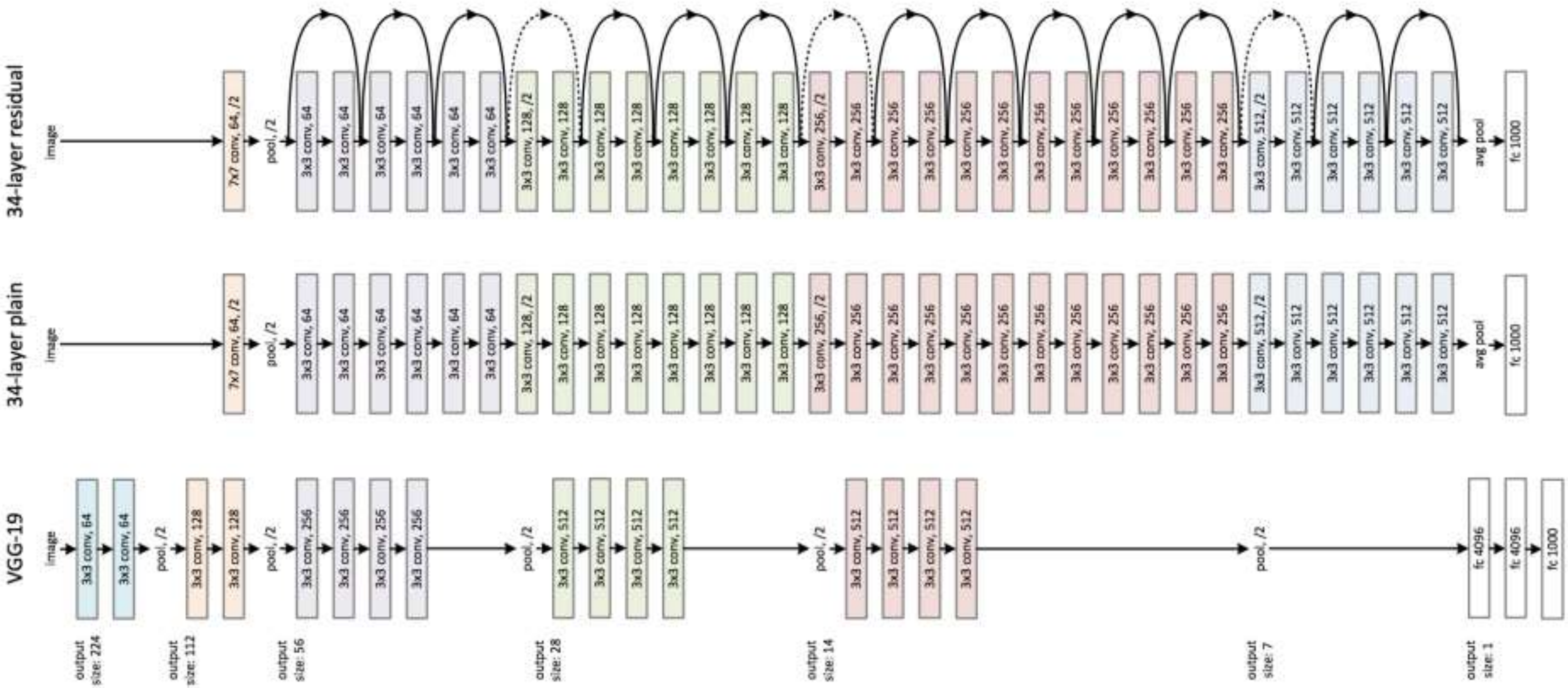
Now: Neural Networks (outlook to deep learning)



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

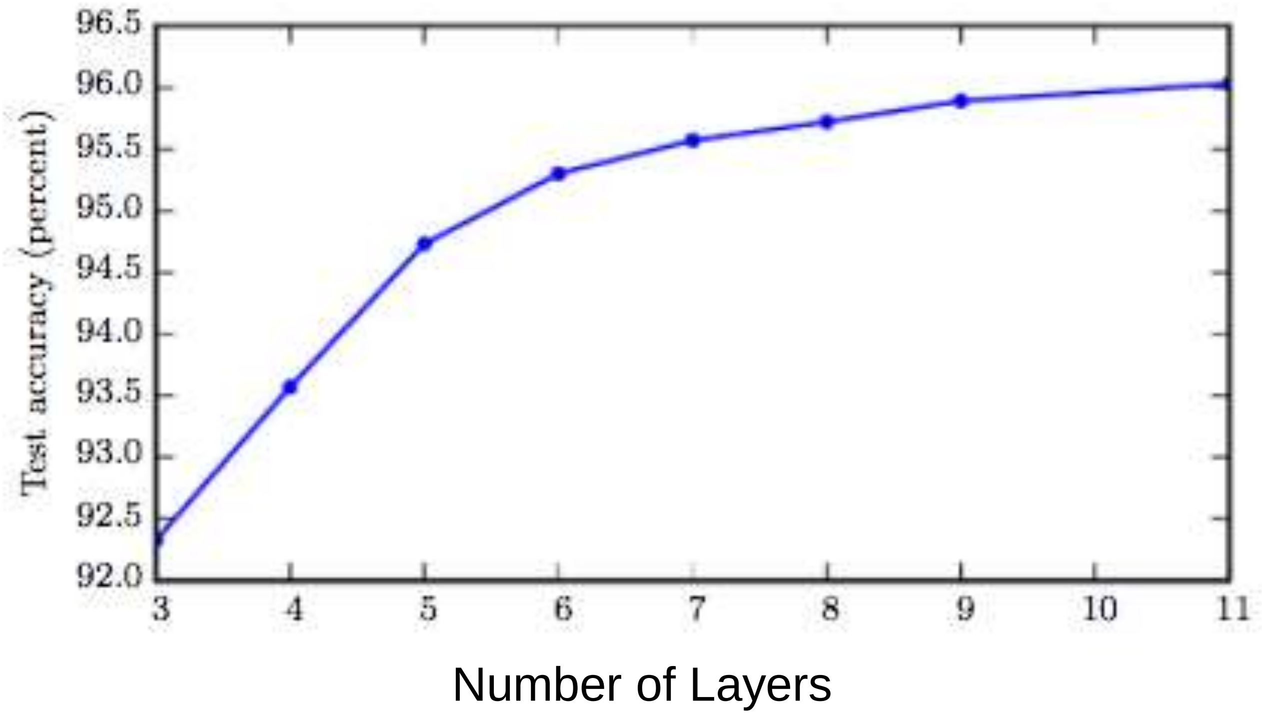
With 1 hidden layer

Examples of deep architectures



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

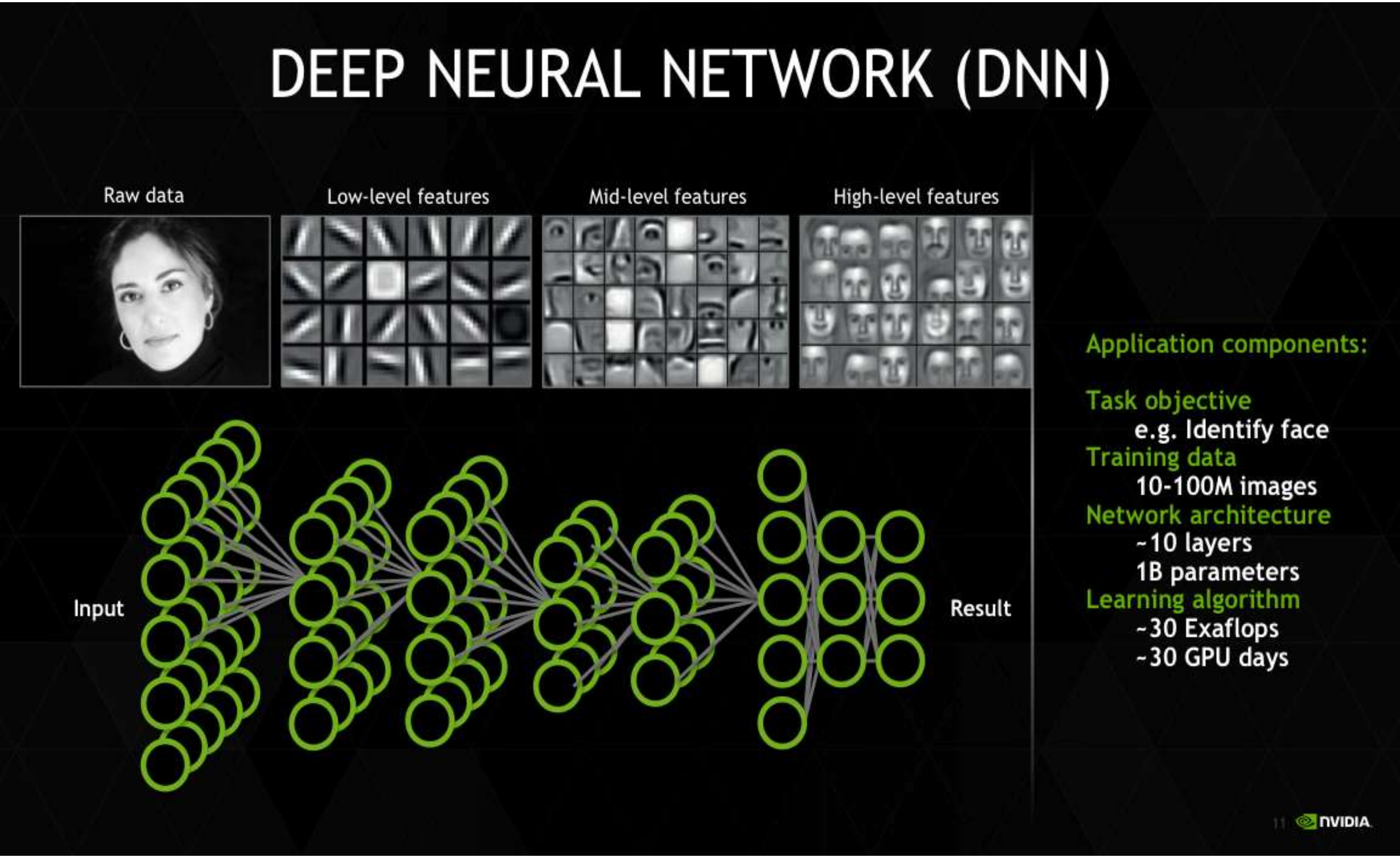
Experimental evidence



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



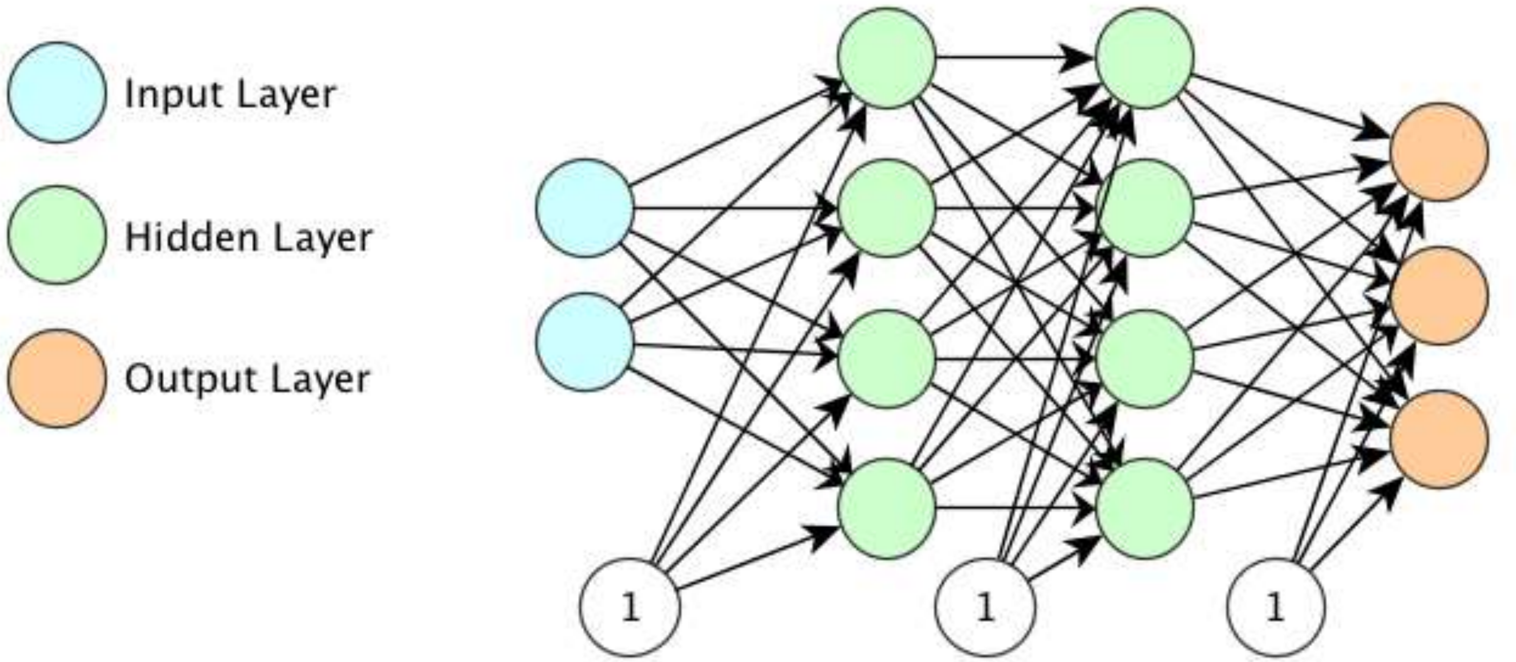
Why deeper (summary)?

- 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

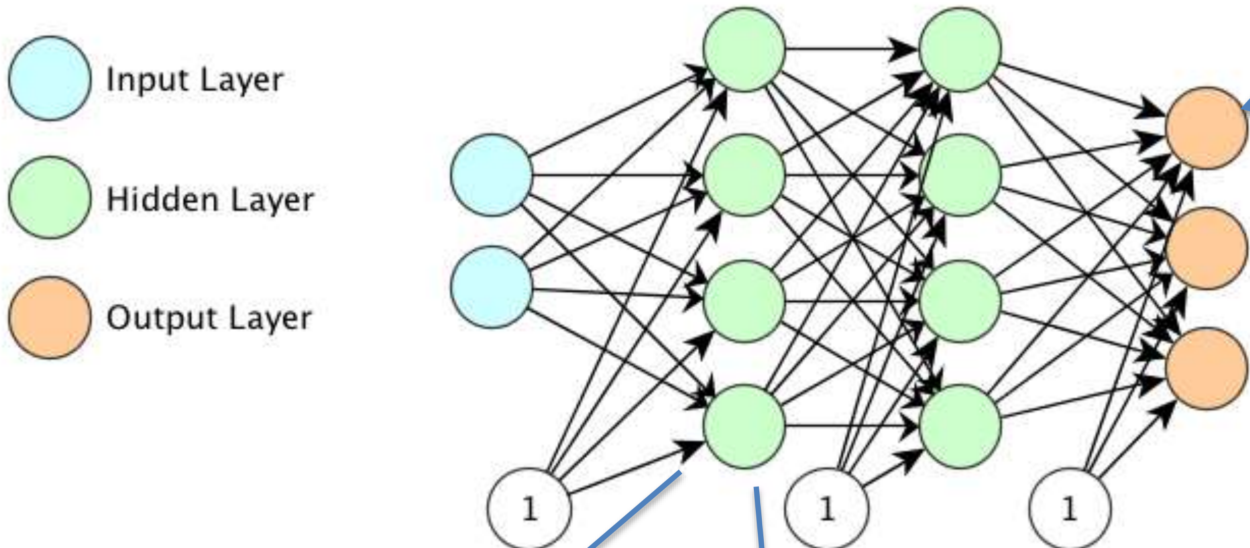
- Use outputs as new inputs
- At the end use multin. logistic regression
- Names:
  - Fully connected network
  - Multi Layer Perceptron (MLP)



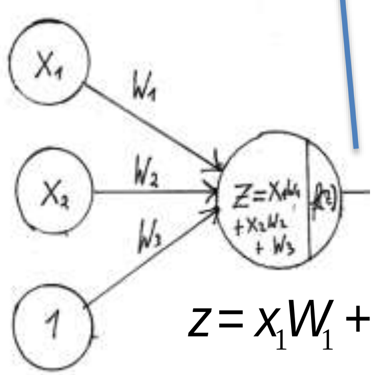
Summary

Softmax in last layer

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



Logistic Regression in hidden layers

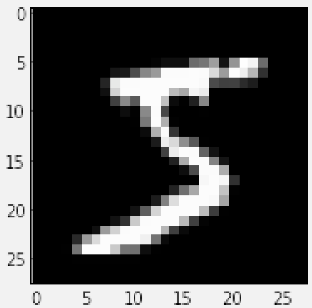


$$f(z) = \frac{\exp(z)}{1 + \exp(z)}$$

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

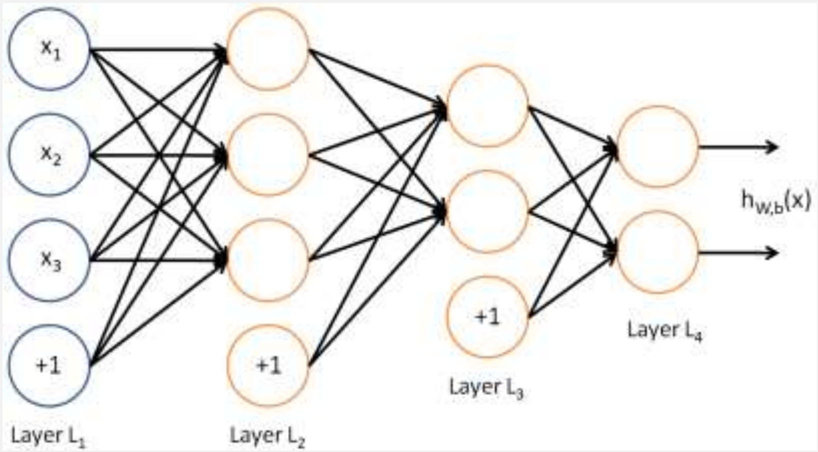
Other activation functions for the hidden layers (see later)

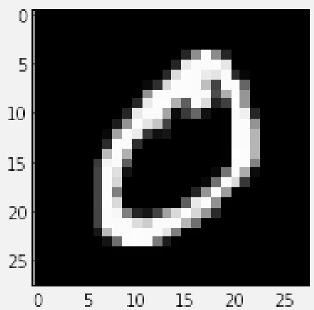
A network for classifying digits



$X^{(1)}$

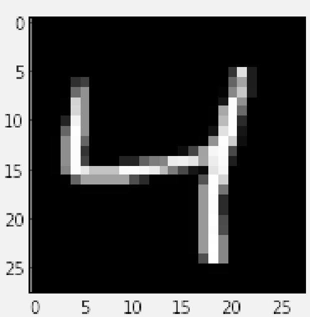
Sketch of the network (not all nodes shown)





$X^{(2)}$

…



$X^{(N)}$

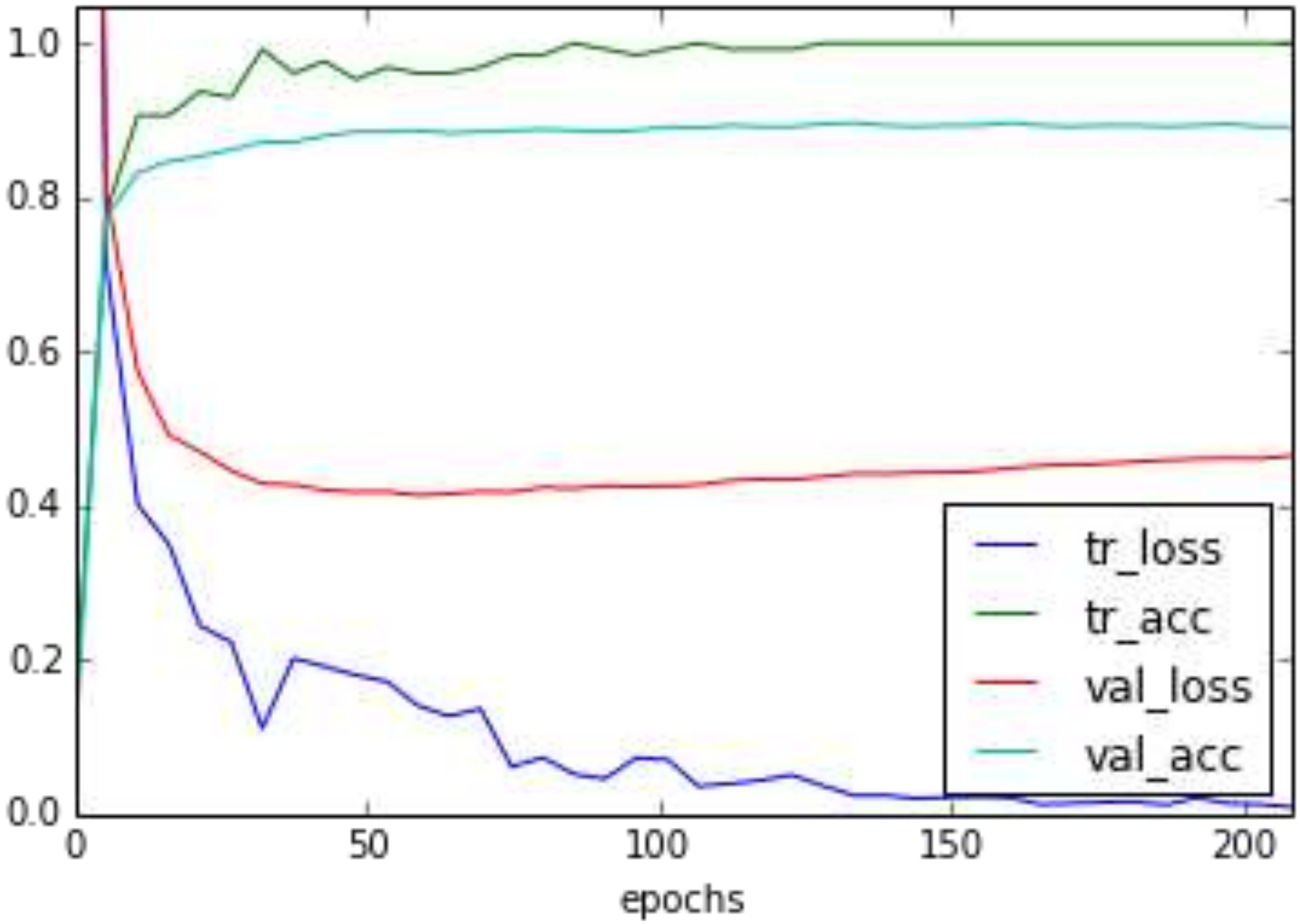
Images 28x28 =784

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).

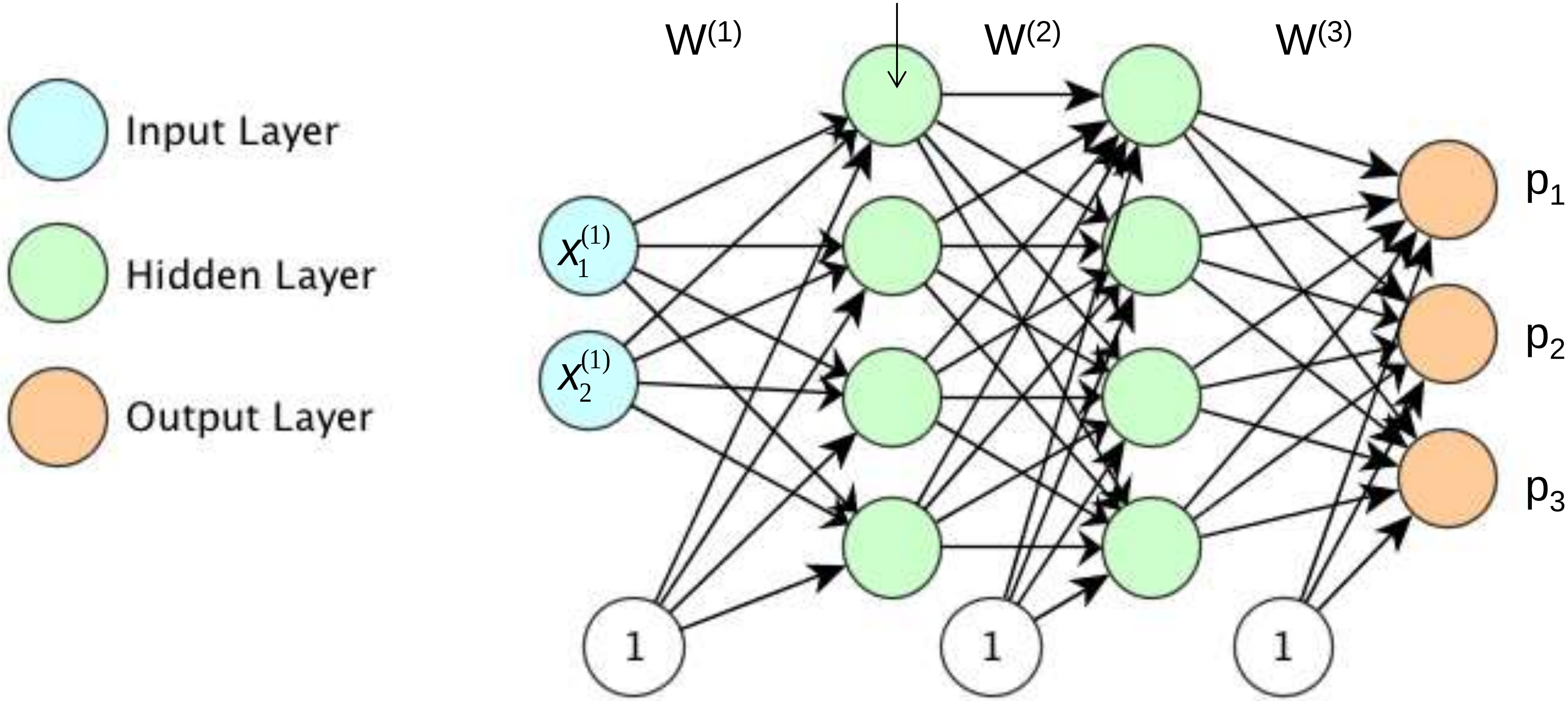
Summary

- 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

$$x_1^{(2)} = f\left(\sum_i W_{i,1}^{(1)} x_i^{(1)} + b_1\right)$$

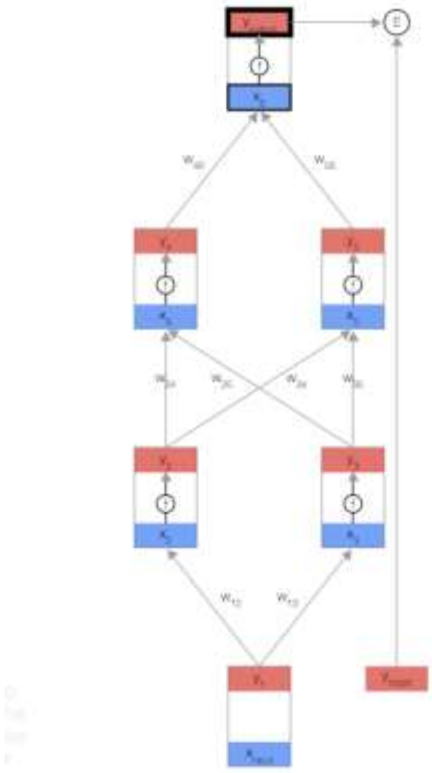


Simple chaining

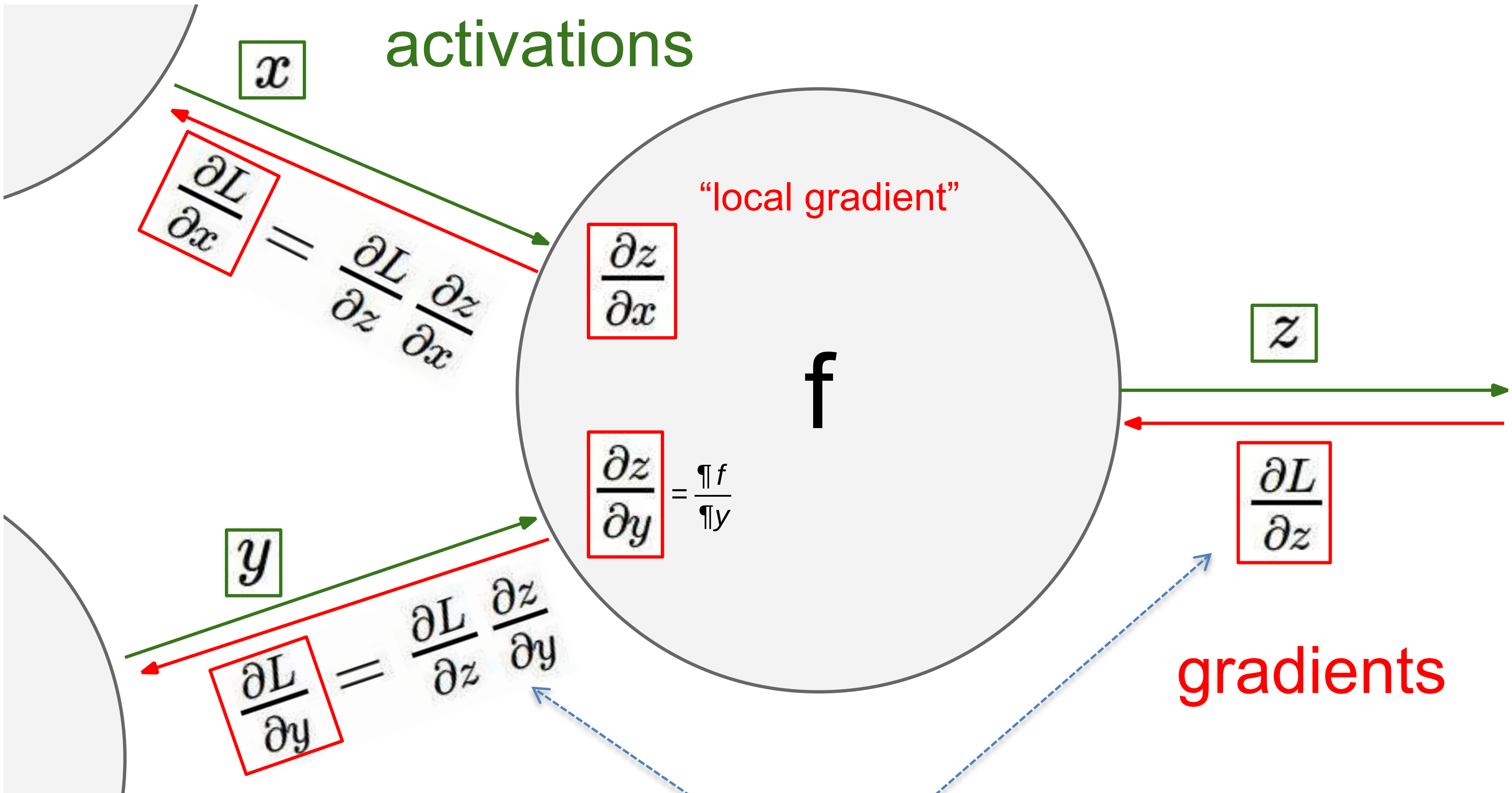
$$p = \mathrm{softmax}(b^{(3)} + W^{(3)} f(b^{(2)} + W^{(2)} f(b^{(1)} + W^{(1)} x^{(1)})))$$

The forward and the backward pass

- https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/
- Here I just give an idea.
- Please read through this example as a homework!
  - Note that they use loss for linear regression



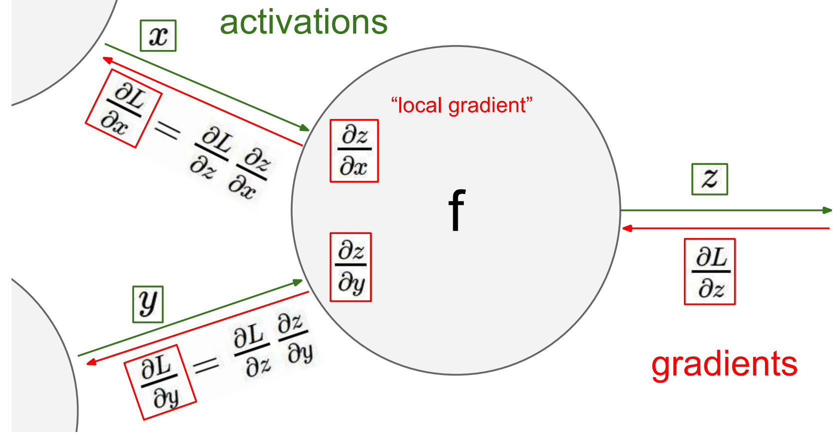
Gradient flow in a computational graph: local junction



is modified by local gradient

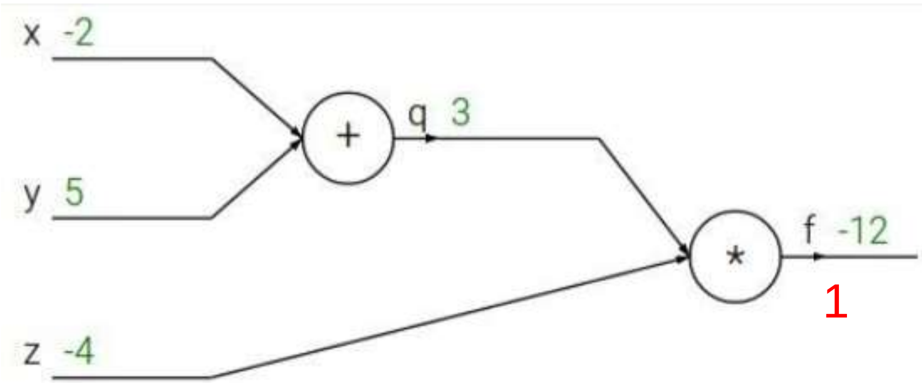
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

Example



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

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



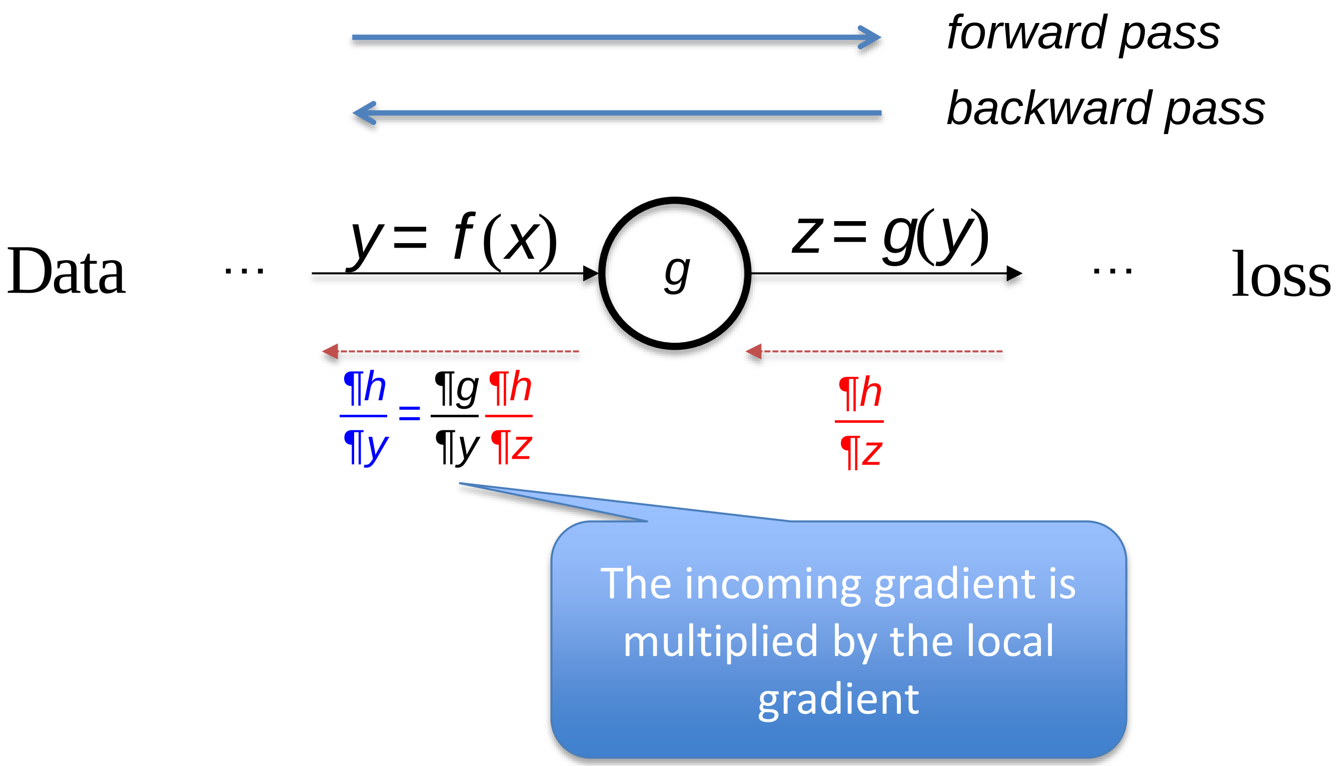
$$\frac{\partial(a+b)}{\partial a} = 1 \qquad \frac{\partial(a*b)}{\partial a} = b$$

➔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!



Gradient flow in a computational graph

How is h effected by x?

$$h(z) = h(g(f(x))$$

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

$$\frac{\partial h}{\partial x} = \frac{\partial f(x)}{\partial x}\frac{\partial g(y)}{\partial y}\frac{\partial h(z)}{\partial z}$$

$$\frac{\partial h}{\partial x} = \frac{\partial f(x)}{\partial x}\frac{\partial h(z)}{\partial y}$$

How is h effected by y?

$$h(z) = h(g(y))$$

$$\frac{\partial h}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial h}{\partial z}$$

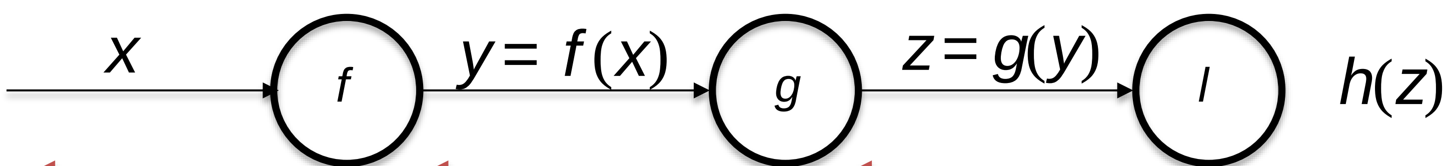
$$\frac{\partial h}{\partial y} = \frac{\partial g(y)}{\partial y}\frac{\partial h(z)}{\partial z}$$

How is h effected by z?

$$h(z) = h(z)$$

$$\frac{\partial h}{\partial z}$$

For DL: h is Loss



$$\frac{\partial h}{\partial x} = \frac{\partial f}{\partial x}\frac{\partial g}{\partial y}\frac{\partial h}{\partial z} = \frac{\partial f}{\partial x}\frac{\partial h}{\partial y}$$

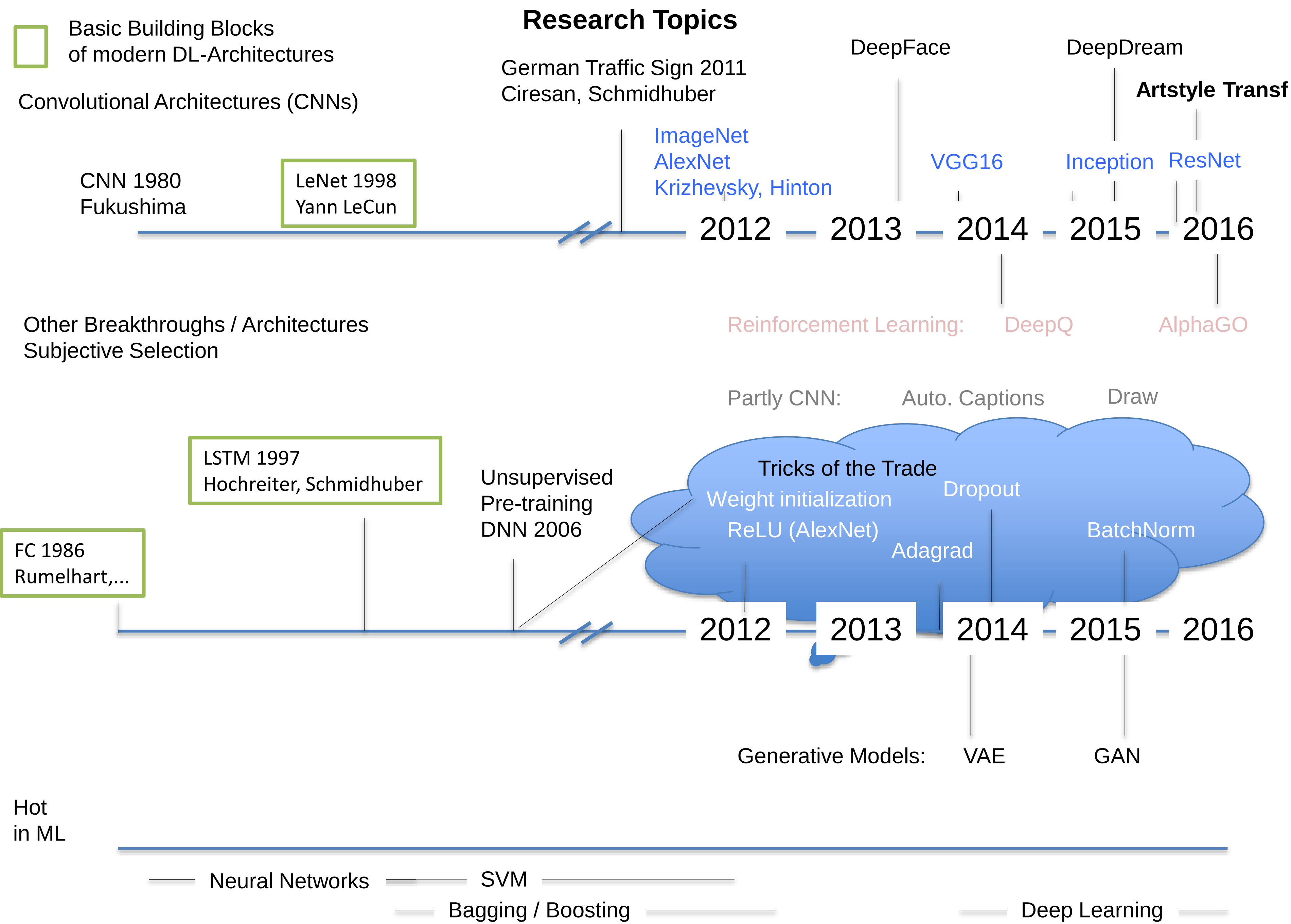
$$\frac{\partial h}{\partial y} = \frac{\partial g}{\partial y}\frac{\partial h}{\partial z}$$

$$\frac{\partial h}{\partial z}$$

incoming gradient

outgoing gradient of an operation f = $\frac{\partial f}{\partial x} \cdot$ incoming gradient

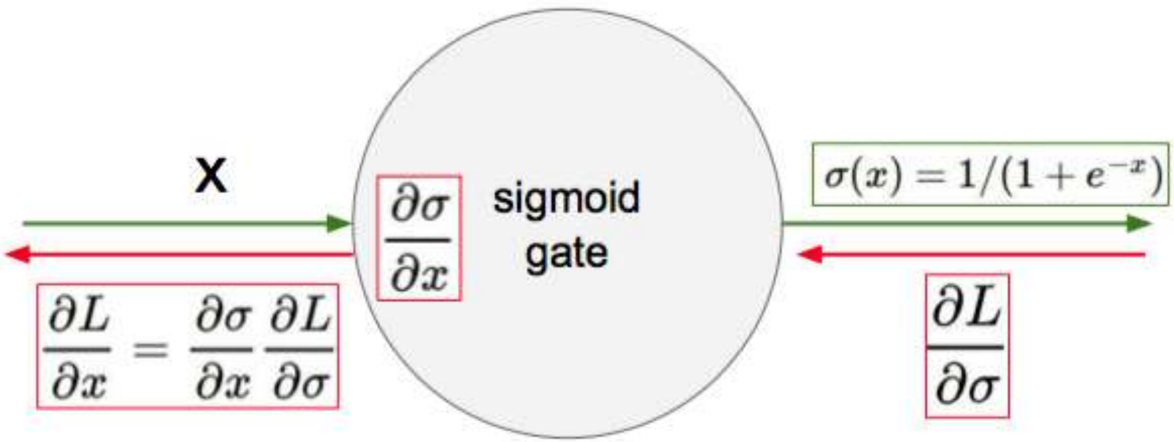
Tricks of the trade

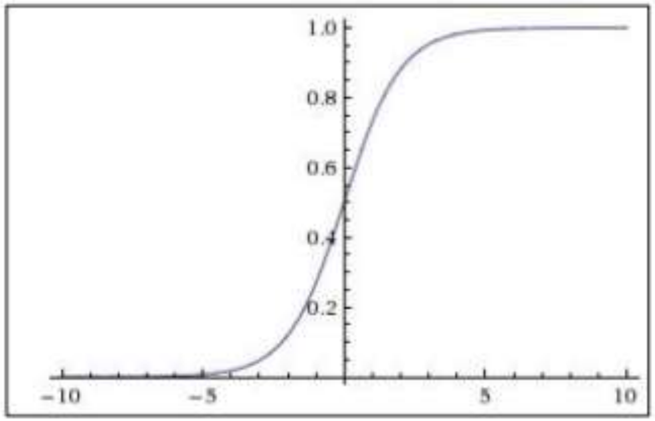


Research Topics
Basic Building Blocks
of modern DL-Architectures
Convolutional Architectures (CNNs)
CNN 1980
Fukushima
LeNet 1998
Yann LeCun
German Traffic Sign 2011
Ciresan, Schmidhuber
ImageNet
AlexNet
Krizhevsky, Hinton
DeepFace
VGG16
DeepDream
Inception
Artstyle Transf
ResNet
2012
2013
2014
2015
2016
Reinforcement Learning:
DeepQ
AlphaGO
Other Breakthroughs / Architectures
Subjective Selection
Partly CNN:
Auto. Captions
Draw
LSTM 1997
Hochreiter, Schmidhuber
Unsupervised
Pre-training
DNN 2006
FC 1986
Rumelhart,...
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





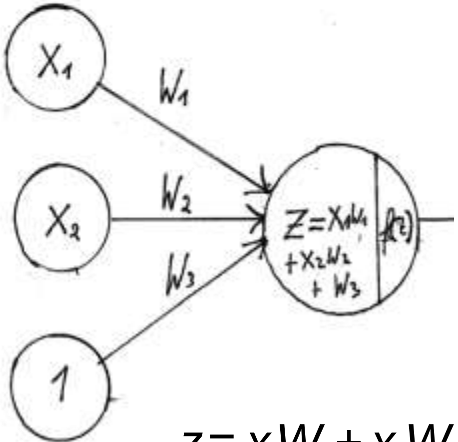
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

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

Slide from: CS231

Different activations in inner layers

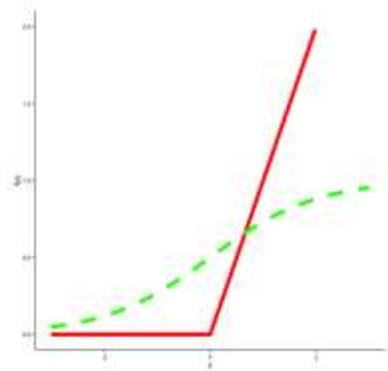
N-D log regression



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

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

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



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

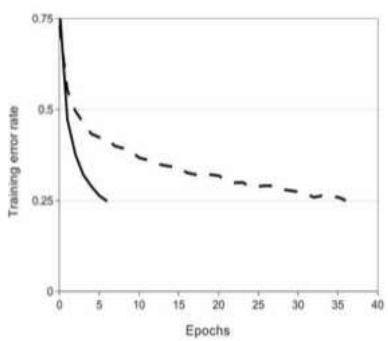


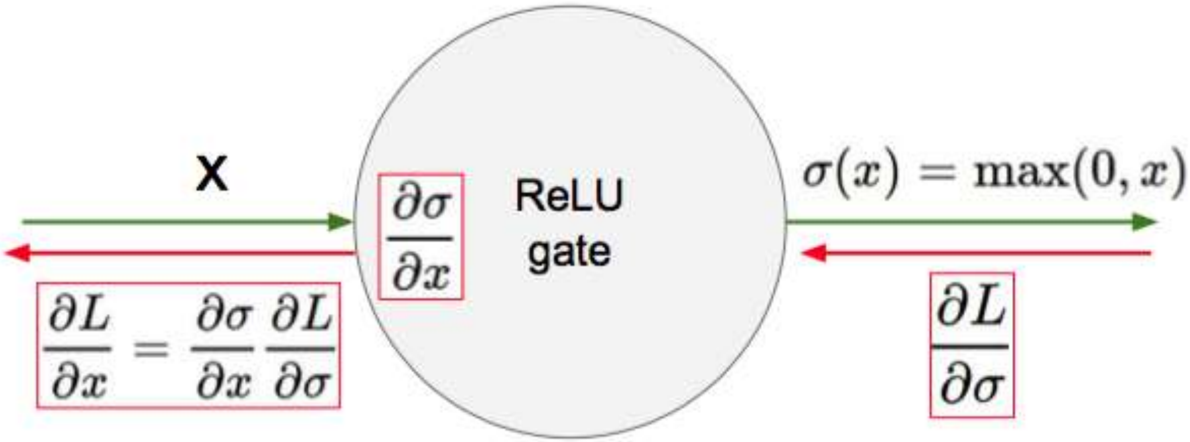
Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

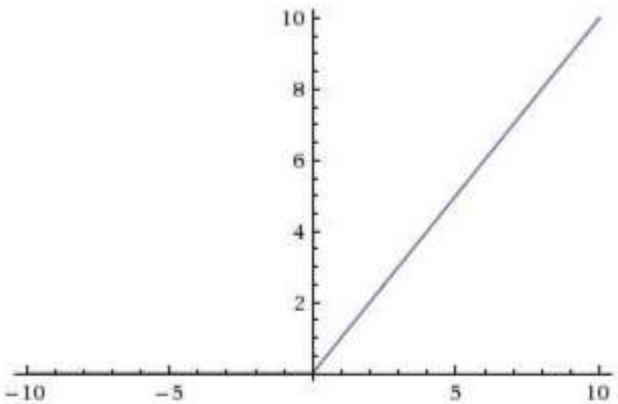
Source:
Alexnet
Krizhevsky et al 2012

There are other alternatives besides sigmoid and ReLU.

Currently ReLU is standard

Backpropagation through ReLU





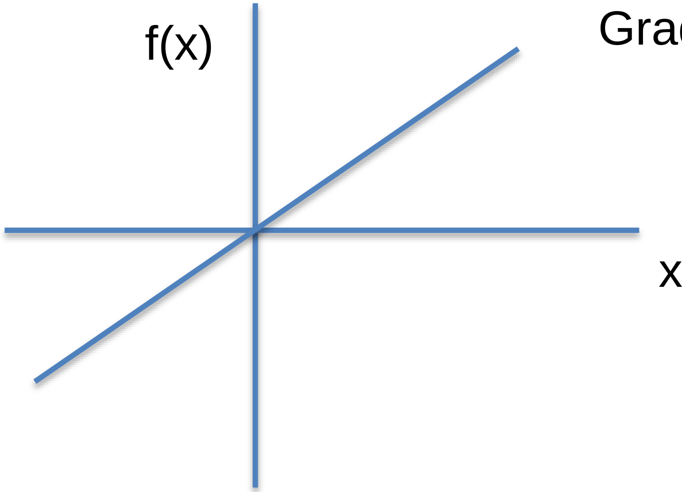
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Gradients are killed, only when x < 0

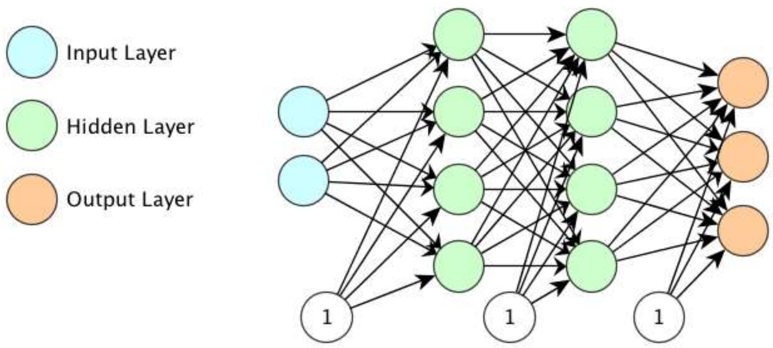
Slide from: CS231

An activation which never gets killed…

- Why just don't take identity?



Gradient is always 1



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

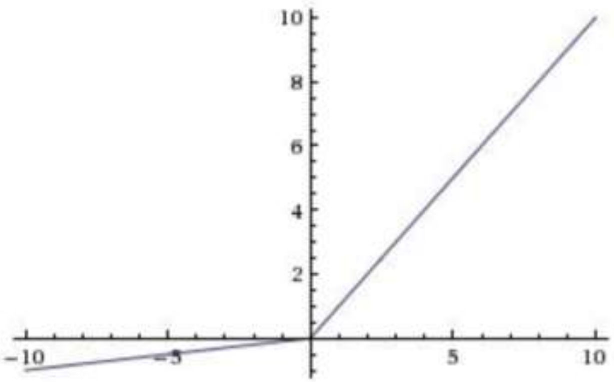
$$x^{(1)} W^{(1)} W^{(2)}\, W^{(3)} = x^{(1)} W$$

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