Machine Intelligence:: Deep Learning
Week 4

Oliver Dürr

Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Winterthur, 12. March. 2019

Outline

Zurich University of Applied Sciences

zh aw

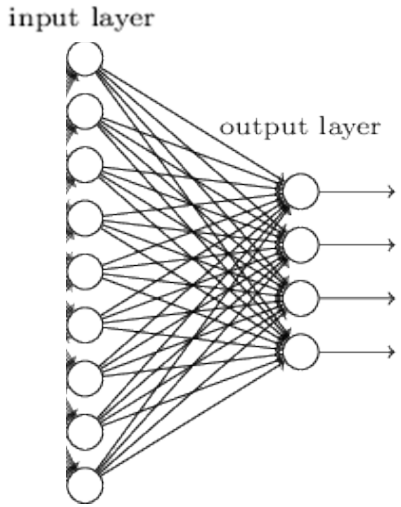
- Homework: Questions and results

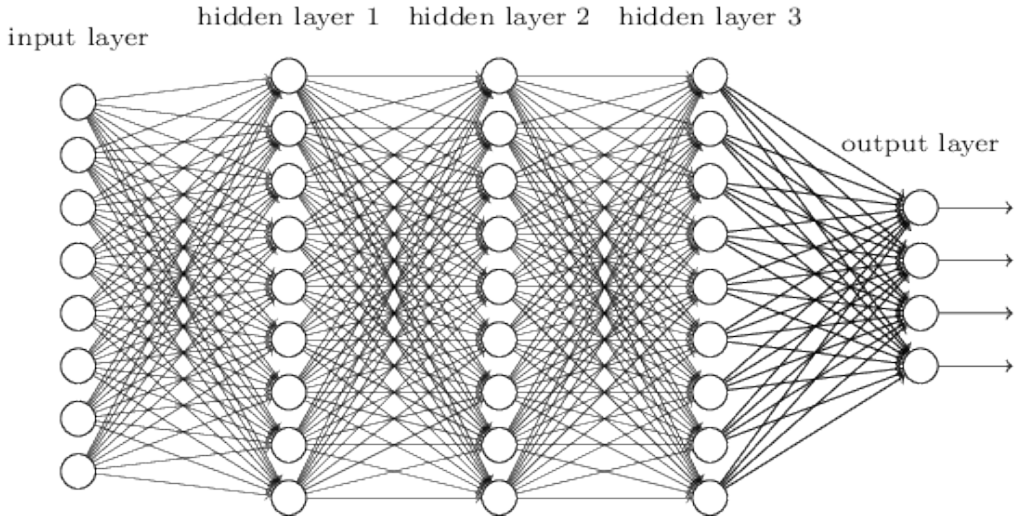
- Tricks of the trade
  - Batchnorm
- Backpropagation

- Motivation of convolutional neural networks (CNNs)
- What is convolution?
- How is convolution performed over several channels/stack of images?
- How does a classical CNN look like?
- Do a CNN yourself

We will go from fully connected NNs to CNNs

Fully connected Neural Networks (fcNN) without and with hidden layers:





Convolutional Neural Network:

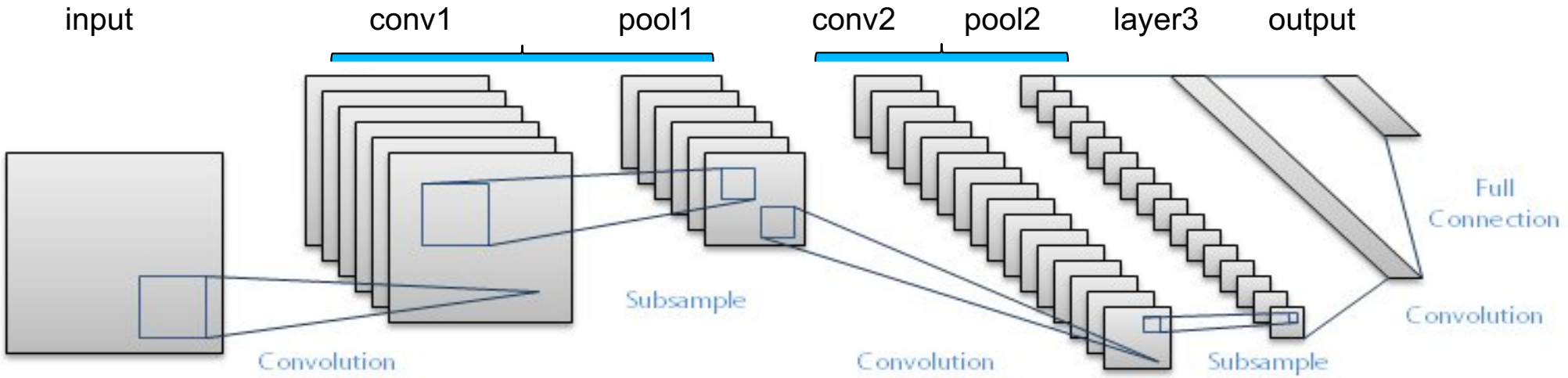


Image credits: http://neuralnetworksanddeeplearning.com/chap6.html, http://www.cemetech.net/projects/ee/scouter/thesis.pdf

At the end of the day

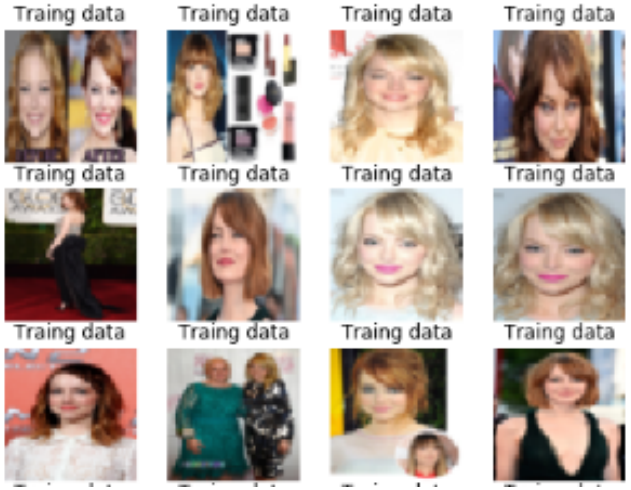
Develop a DL model to solve this task:

For a given image from the internet, decide which out of 8 celebrities is on the image.

Example images:

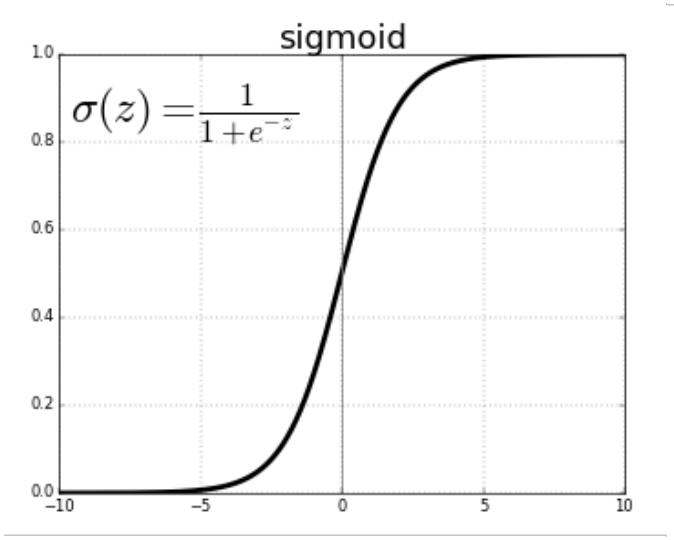
Label: Steve Jobs (entrepreneur)

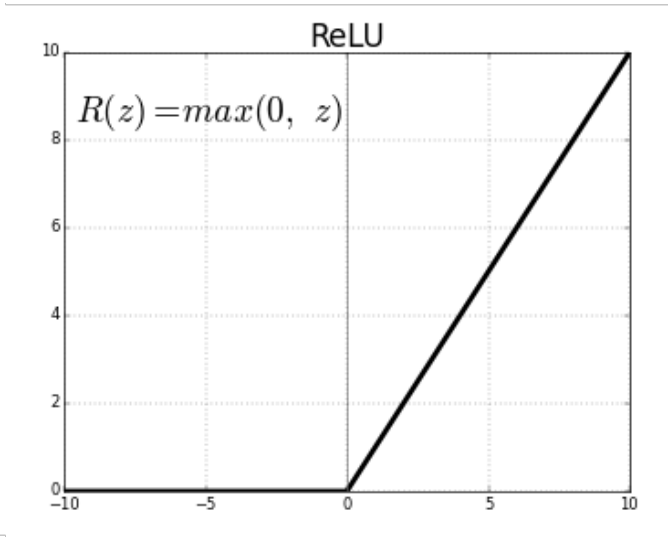
Label: Emma Stone (actress)

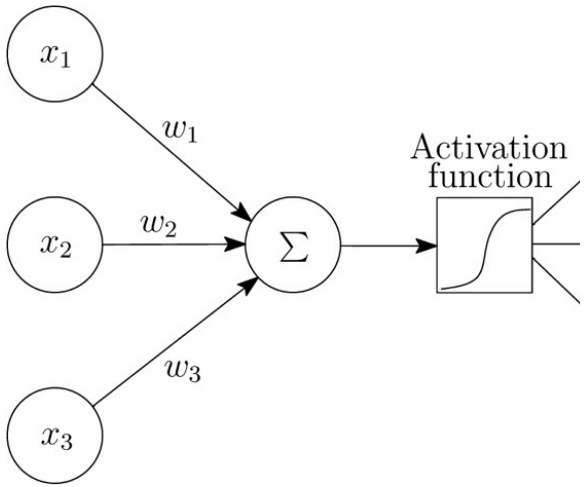


Homework

Question in home work:
Which activation function should we use? Does it matter?

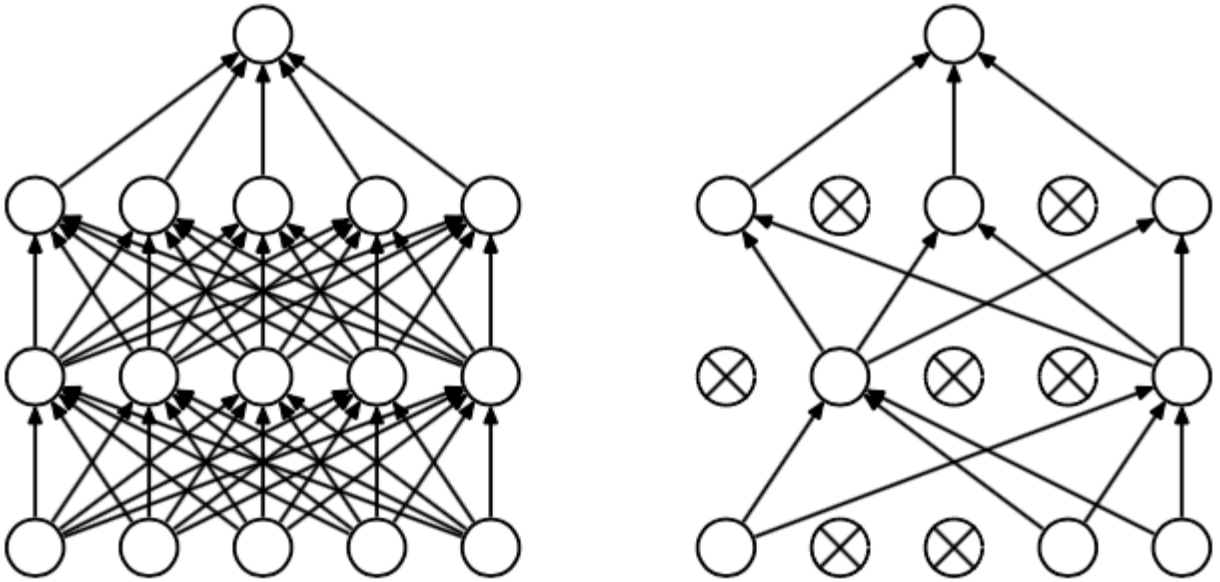






Question in home work: Dropout – does it help?

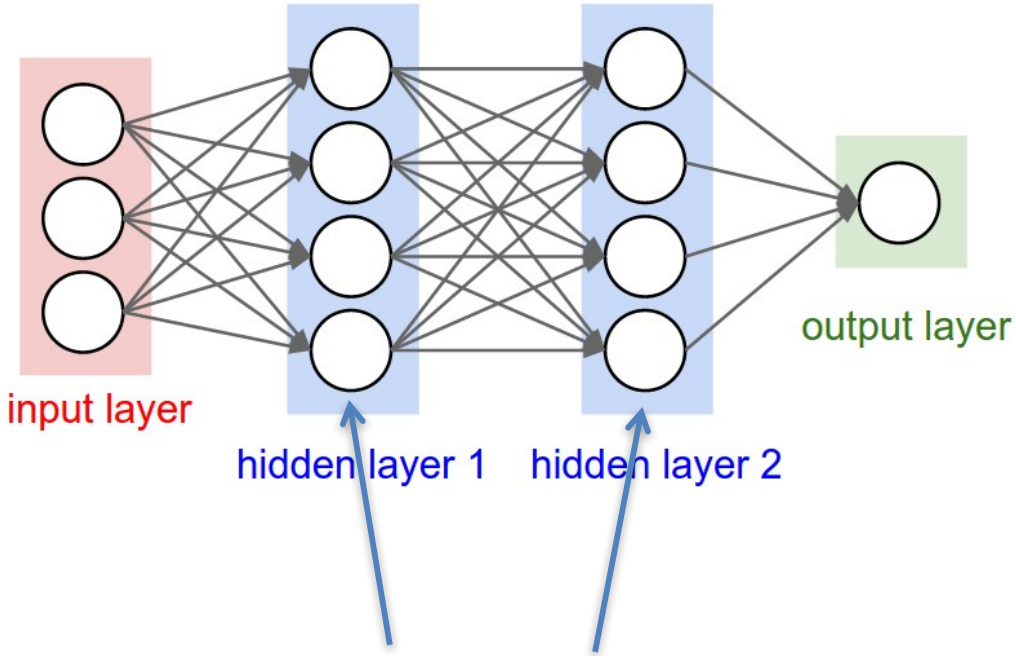
- After each weight update, we randomly "delete" a certain percentage of neurons, which will not be updated in the next step – than repeat.
- In each training step we optimize a slightly different NN model.



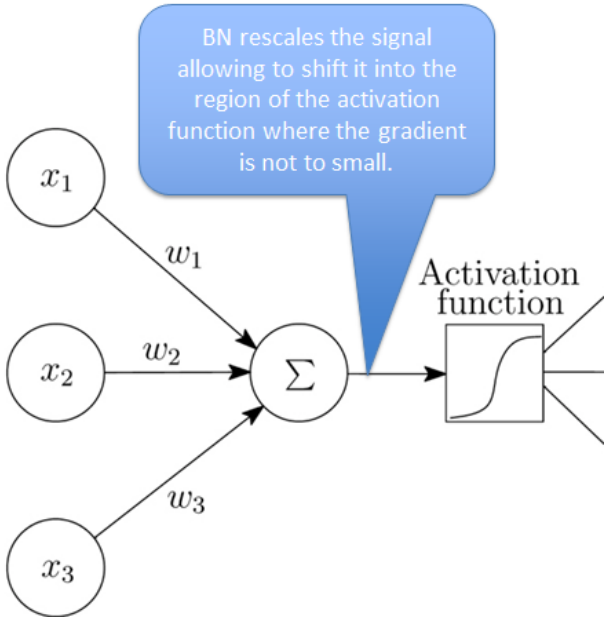
(a) Standard Neural Net

(b) After applying dropout.

Question in home work: Should we allow the NN to normalize the data between layers (batch_norm)? Does it matter?

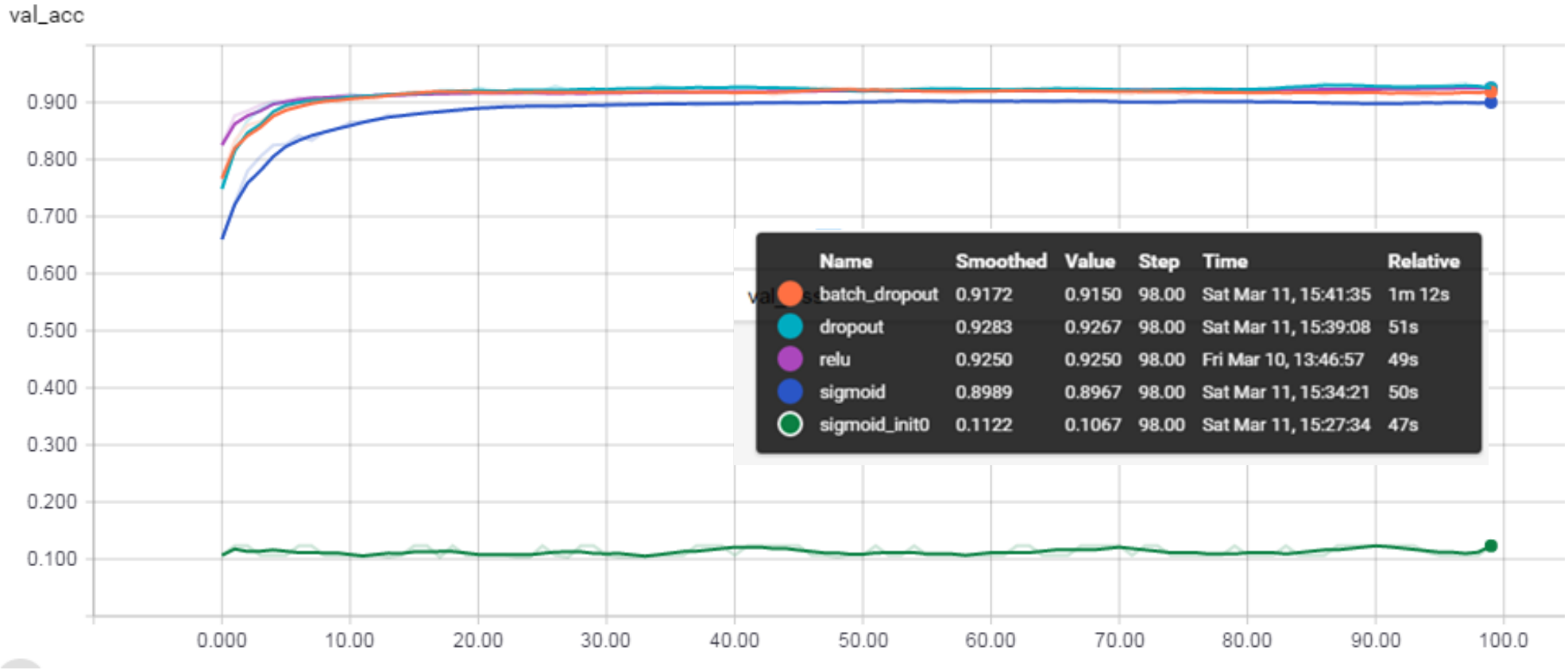


Should we allow the NN to normalize intermediate data (activations), so that they have mean=0 and sd=1?



Home work - main result

With this small training set of 4000 images and 100 epochs training we get the best test accuracy of ~92% when working with random initialization (reducing weights if number of input data increases), ReLu, dropout and BN (here BN does not improve things – in many applications it does!).



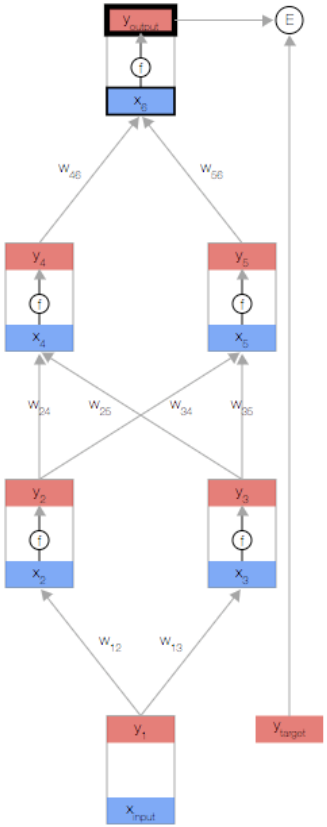
Why did ReLU help so much. Why is it a bad idea to have too high weights

Backpropagation

Slide Credit to Elvis Murina for the great animations

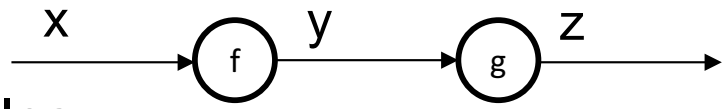
Motivation:
The forward and the backward pass

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



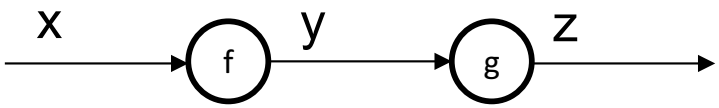
Chain rule recap

- 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

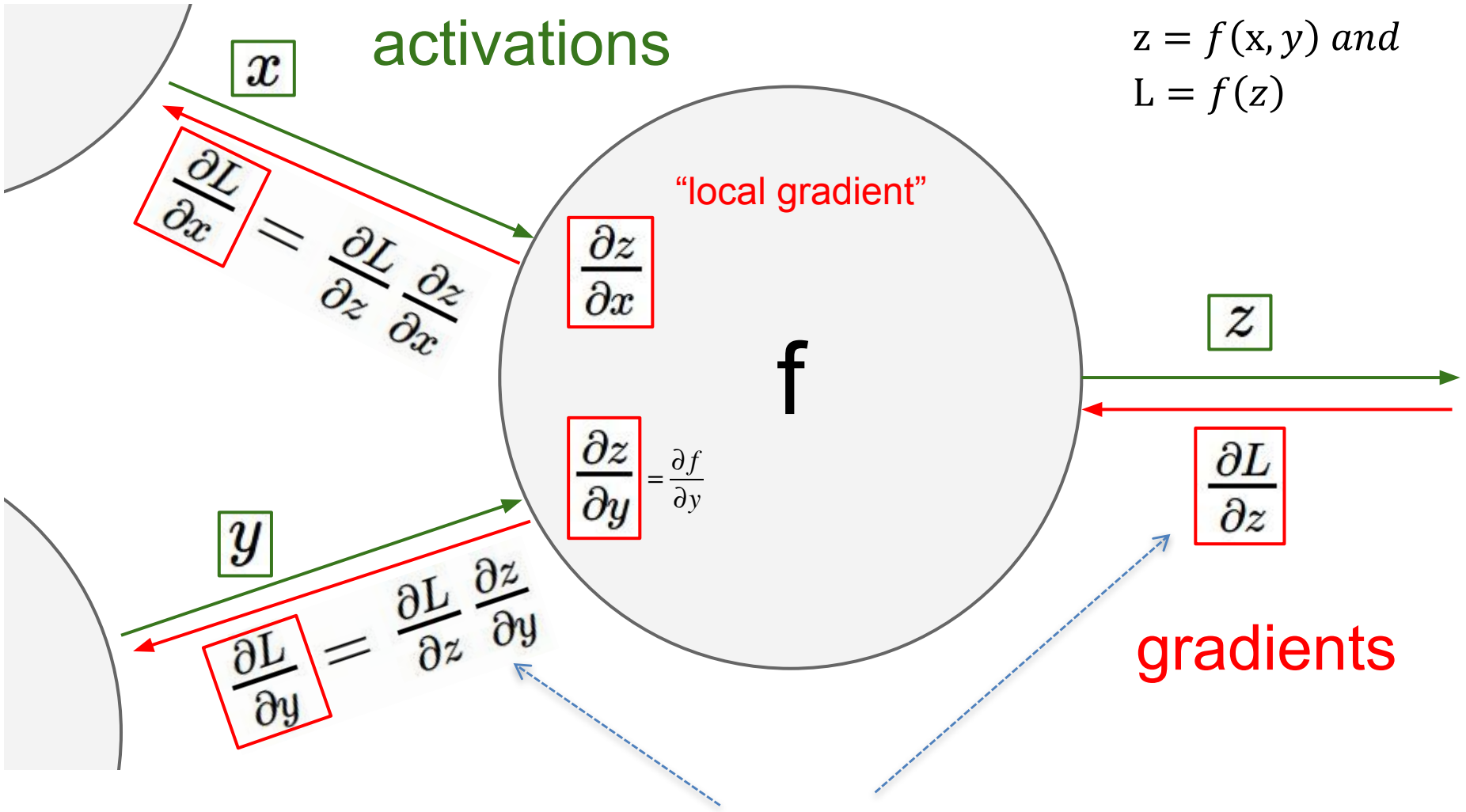
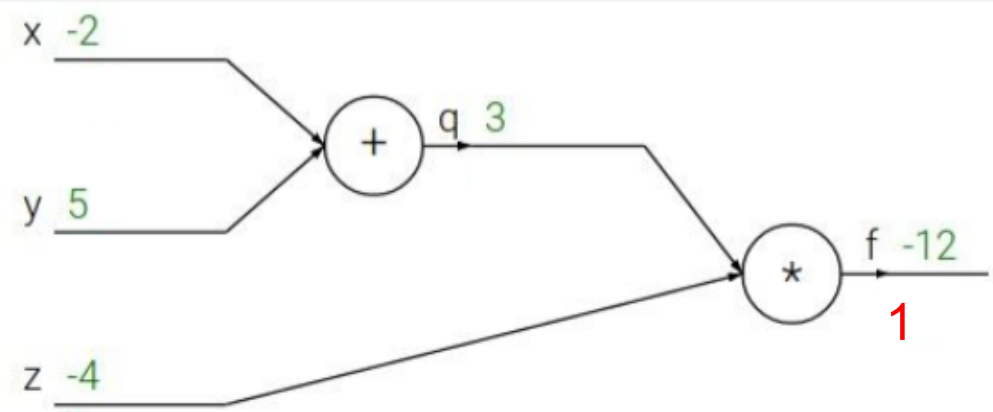


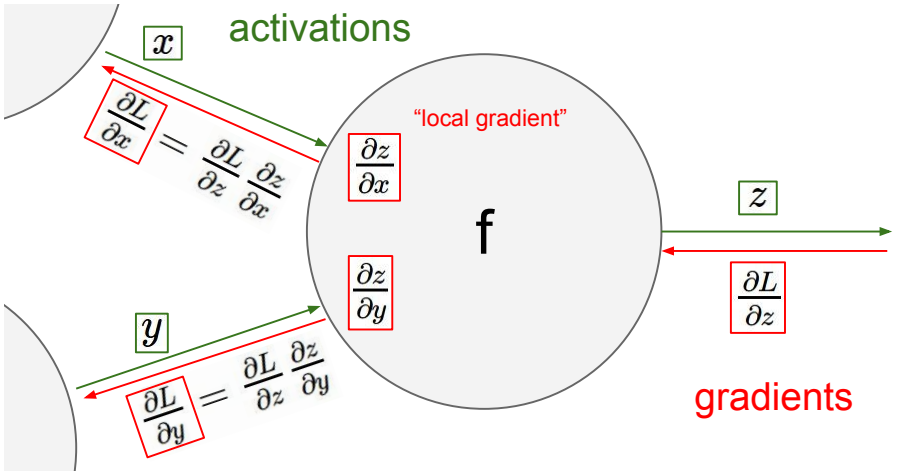
Illustration: http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

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:

$x_1 = 1$

$x_2 = 2$

$y_1 = 1$

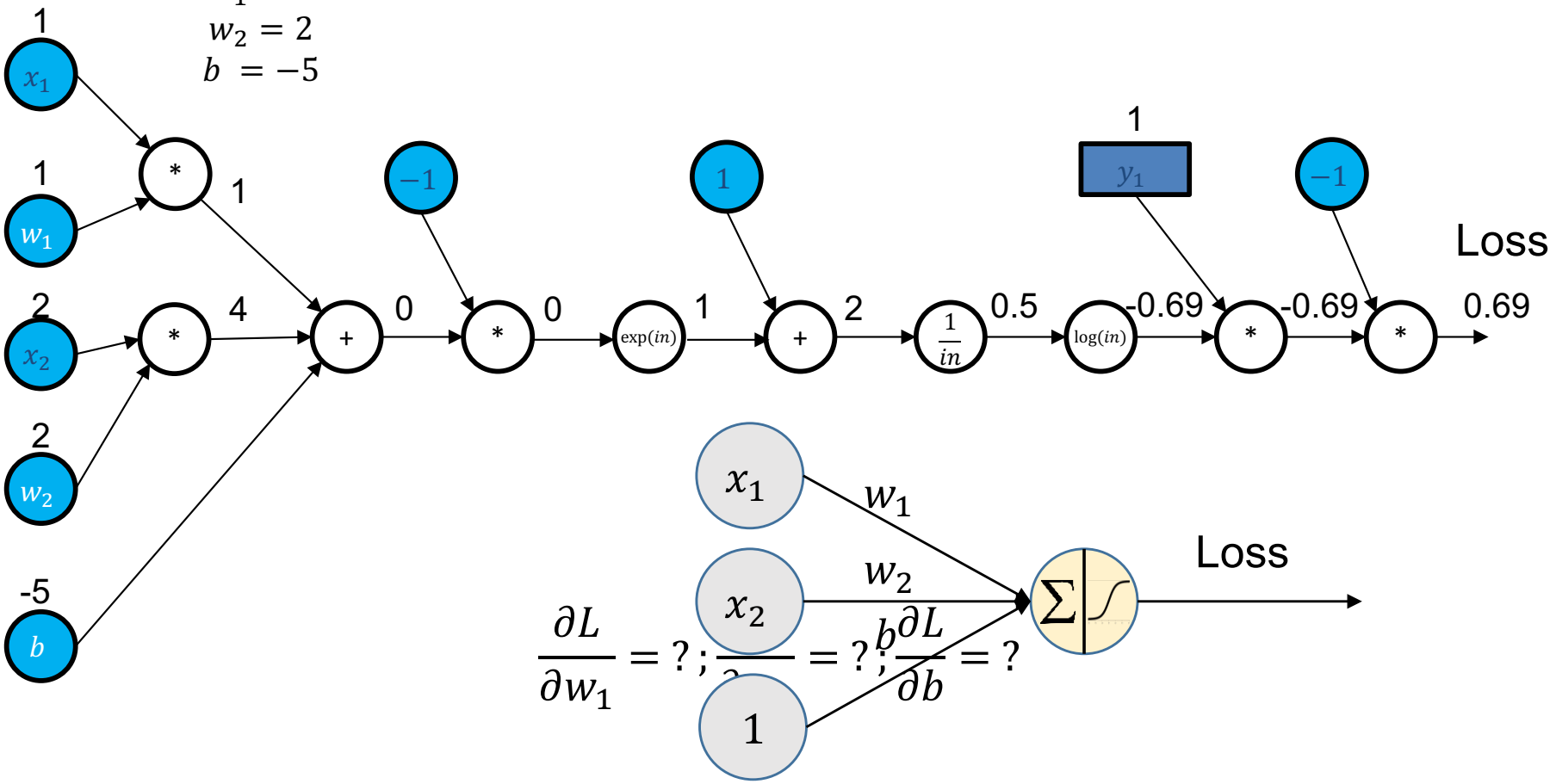
Initial weights:

$w_1 = 1$

$w_2 = 2$

$b = -5$

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



Backward pass

Training data:

$x_1 = 1$

$x_2 = 2$

$y_1 = 1$

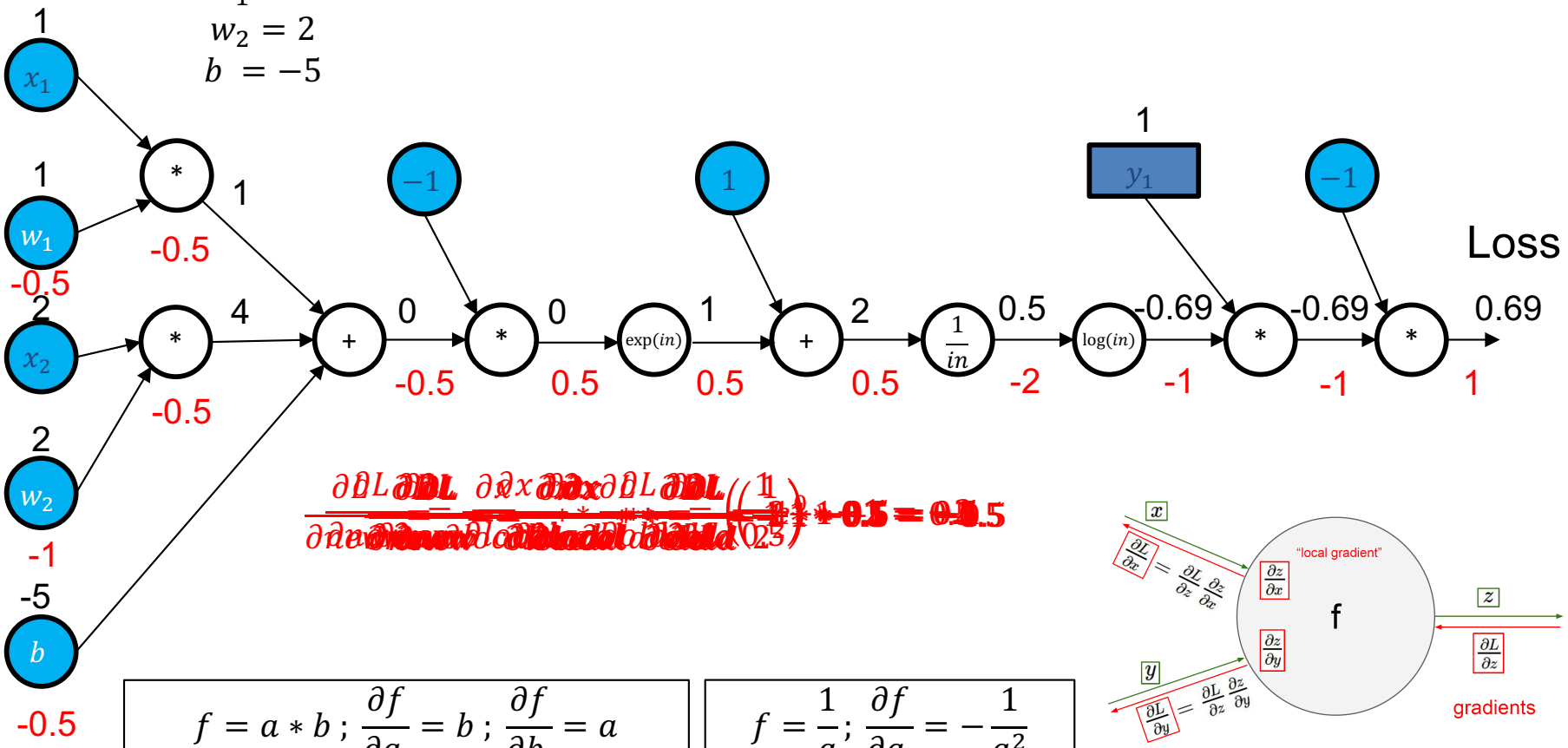
Initial weights:

$w_1 = 1$

$w_2 = 2$

$b = -5$

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



$f = a * b\,;\ \frac{\partial f}{\partial a} = b\,;\ \frac{\partial f}{\partial b} = a$	$f = \frac{1}{a};\ \frac{\partial f}{\partial a} = -\frac{1}{a^2}$	
$f = a + b\,;\ \frac{\partial f}{\partial a} = 1\,;\ \frac{\partial f}{\partial b} = 1$	$f = e^a;\ \frac{\partial f}{\partial a} = e^a$	$f = \log(a)\,;\ \frac{\partial f}{\partial a} = \frac{1}{a}$

Forward pass

Training data:

$x_1 = 1$

$x_2 = 2$

$y_1 = 1$

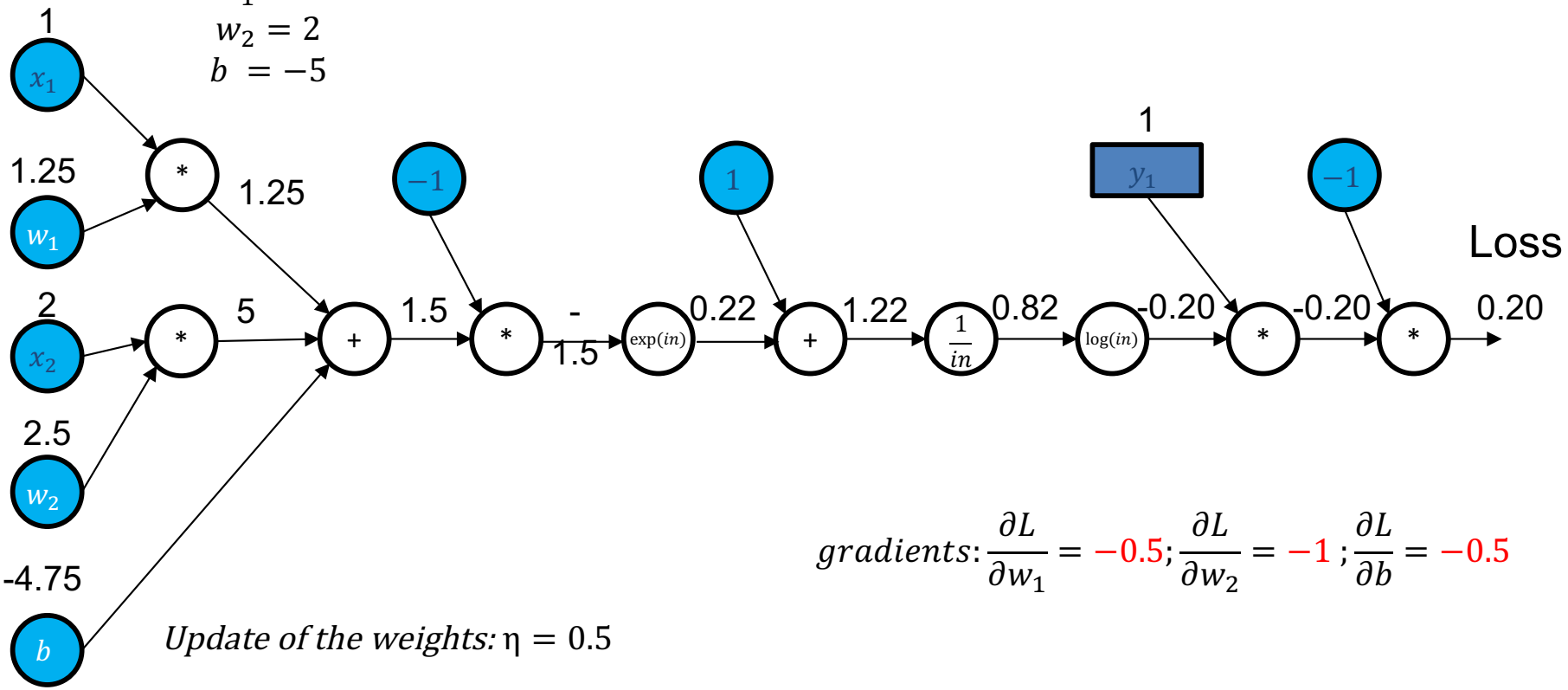
Initial weights:

$w_1 = 1$

$w_2 = 2$

$b = -5$

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



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

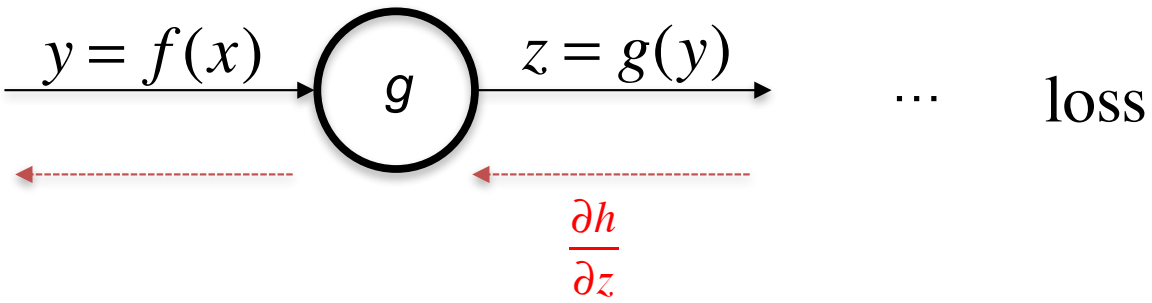
Update of the weights: $\eta = 0.5$

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

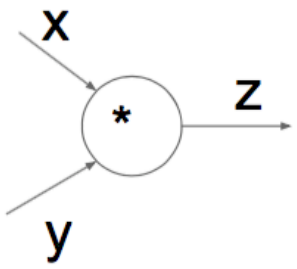
$$w_{2(t+1)} = w_{2(t)} - \eta * \frac{\partial L}{\partial w_2} = 2 - 0.5 * (-1) = 2.5$$

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

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 $\frac{\partial g}{\partial y}\Big|_y$



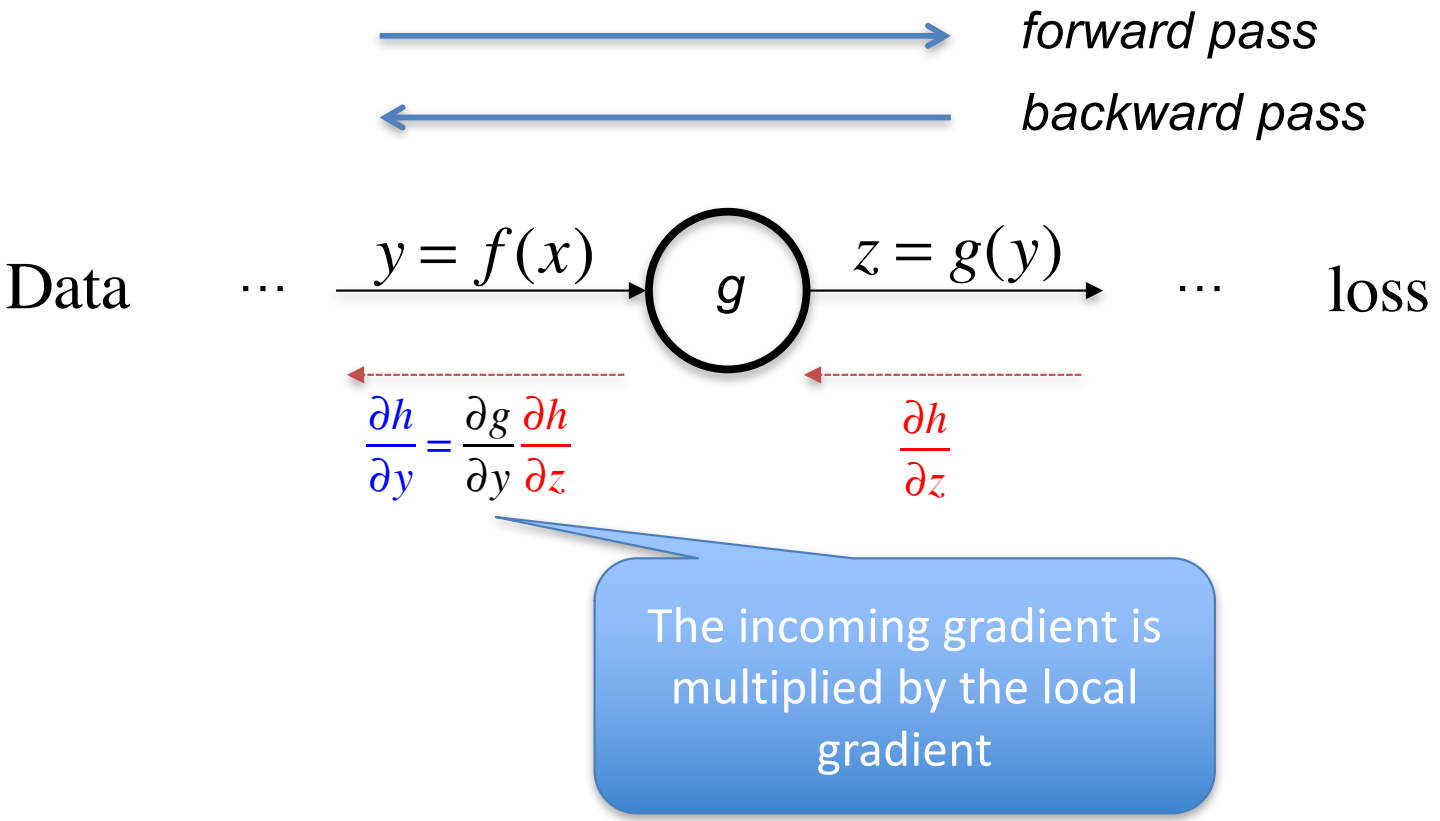

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

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

Further References / Summary

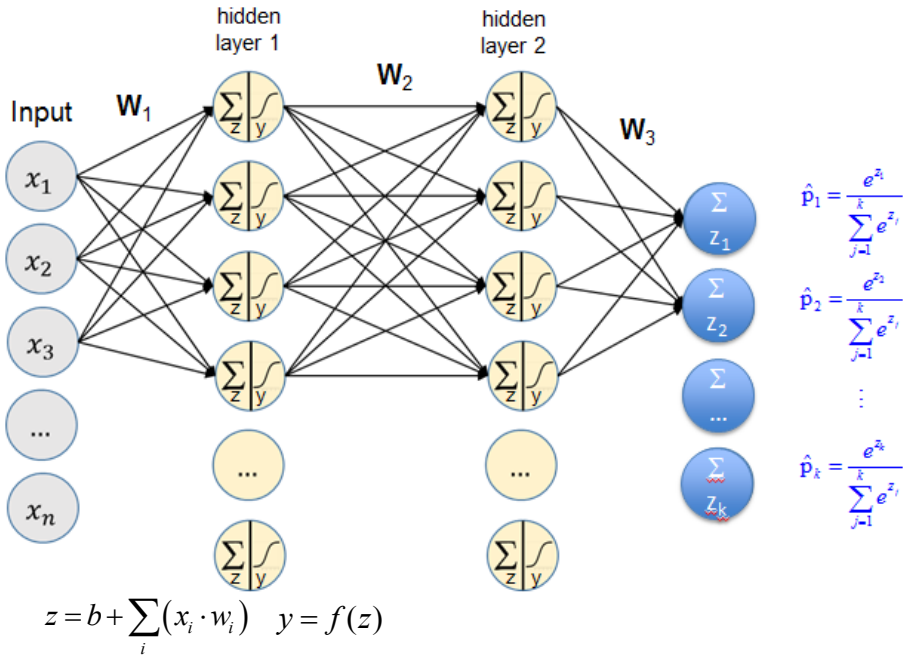
- For a more in depth treatment have a look at
  - Lecture 4 of http://cs231n.stanford.edu/
  - Slides http://cs231n.stanford.edu/slides/winter1516_lecture4.pdf

- Gradient flow is important for learning: remember!



Consequences of the chain rule

Bad idea: initializing all weights with the same value



$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial C(\mathbf{w})}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$

chain rule:

$$\left. \frac{\partial C}{\partial w_{2n}} \right|_{\text{current NN values}} = \left. \frac{\partial C}{\partial p_k} \right|_{cv} \cdot \left. \frac{\partial \hat{p}_k}{\partial z_{2m}} \right|_{cs} \cdot \left. \frac{\partial z_{2m}}{\partial y_{2m}} \right|_{cv} \cdot \left. \frac{\partial y_{2m}}{\partial z_{1n}} \right|_{cv} \cdot \left. \frac{\partial z_{1n}}{\partial w_{2n}} \right|_{cv}$$

Forward pass: initialize all weights with the same value

⇒ all units get the same values $y_j = f(z_j) = f(b + \sum_i x_i w_i)$

⇒ … all outputs are the same.. (Initializing all weights=0 will give all units the value 0!)

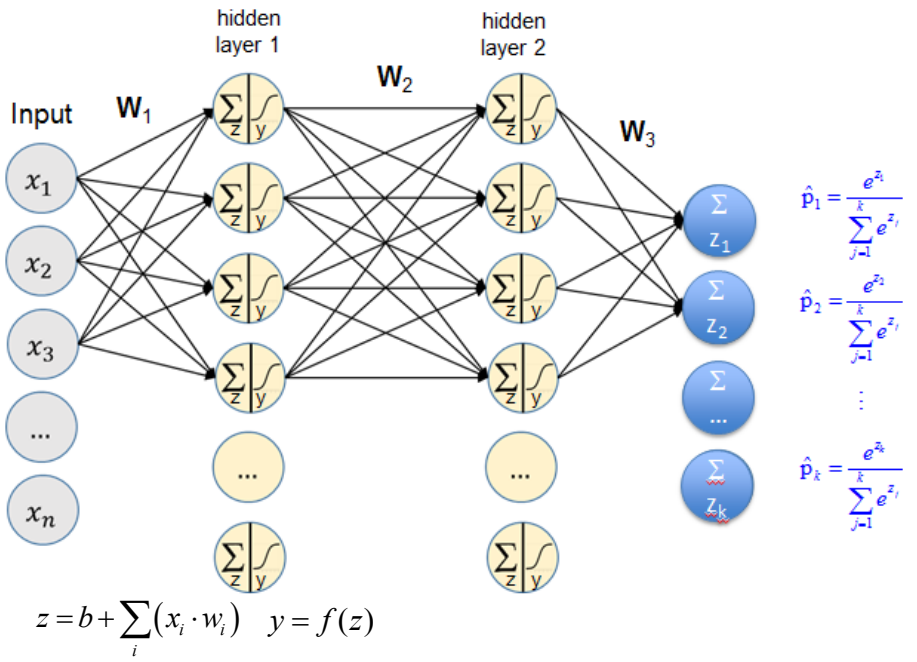
Backward pass: all weights and units have same values & all functions same

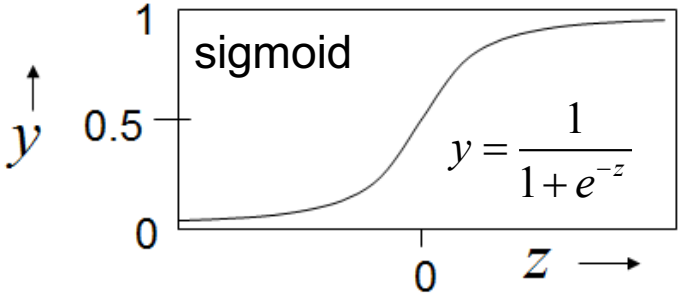
⇒ all gradients are the same

⇒ all weights get the same update and get again the same value!

⇒ no learning

Bad idea: initializing with high values





Initialize weights with partly large values.
⇒ large absolute z-values
⇒ flat parts of activation function

⇒ According chain rule we multiply with $\frac{\partial y}{\partial z} \approx 0$
⇒ gradient is zero we cannot update the weights ⇒ no learning

$$w_i^{(t)} = w_i^{(t-1)} - \varepsilon^{(t)} \left. \frac{\partial C(\mathbf{w})}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$

What is the default initializer in Keras?

Dense https://keras.io/layers/core/ [source]

Also in conv layer 'glorot_uniform' is used as default initializer.

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_i
```

Recommended 2010 by Glorot & Bengio
http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

glorot_uniform

```
glorot_uniform(seed=None)
```

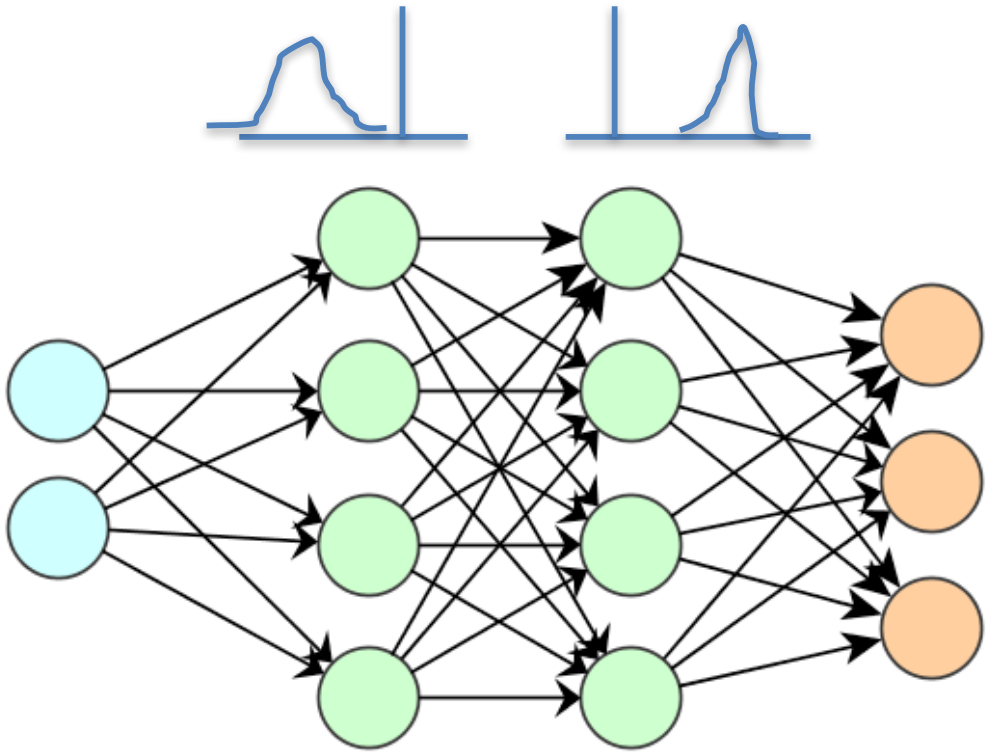
Glorot uniform initializer, also called Xavier uniform initializer.

It draws samples from a uniform distribution within [-limit, limit] where `limit` is `sqrt(6 / (fan_in + fan_out))` where `fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

guarantees random small numbers

Batch Normalization*

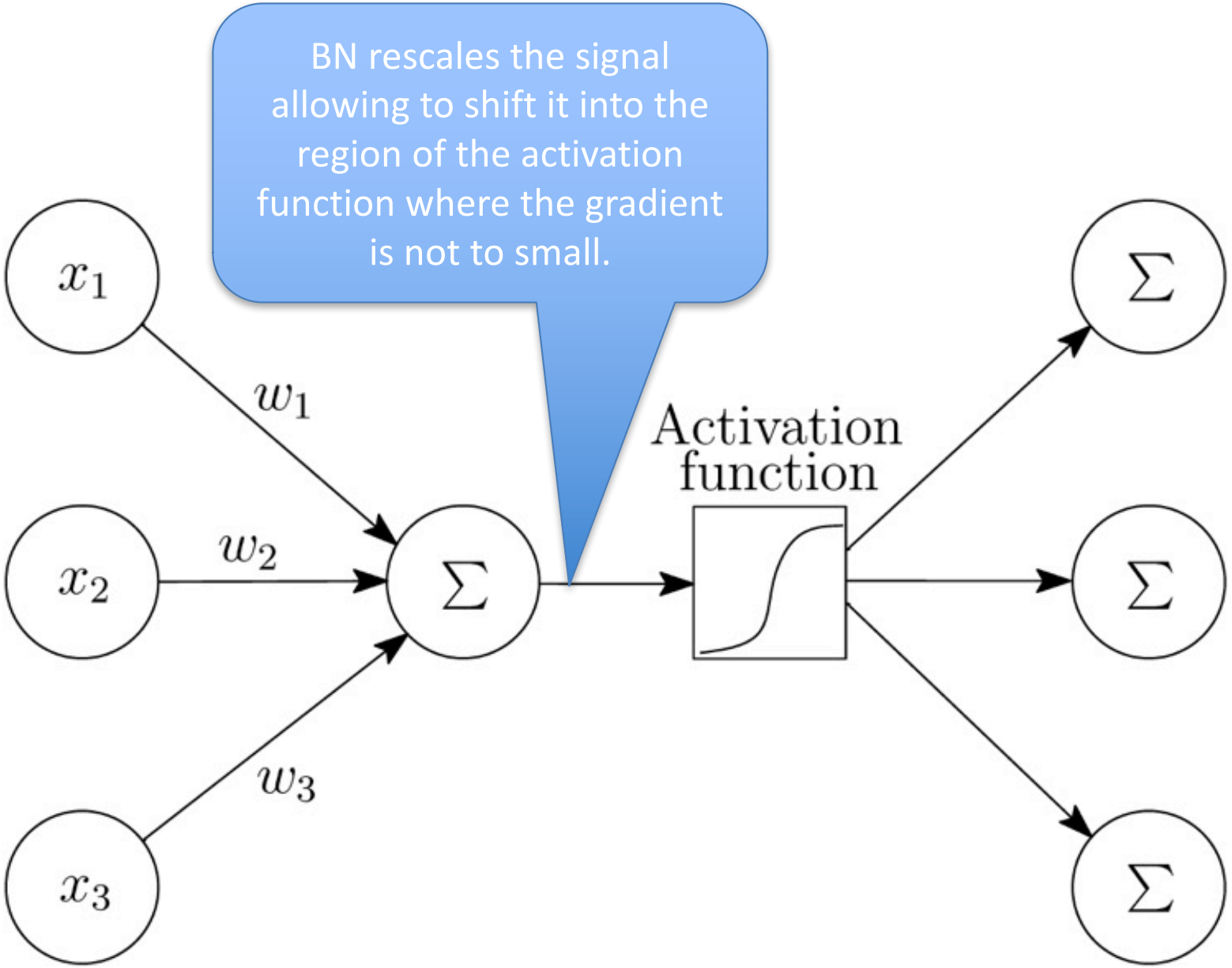
- Not a regularisation per se, but speeds up learning in practice
- Data fluctuates around different means with different variances in the layers



- Problematic for non-linarites which have sweet spot around 0
- Or ReLus with activation < 0 ➔ dying gradient
- Too much changes down stream

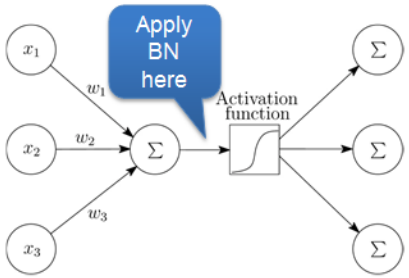
* (Ioffe, Szegedy, 2015) Batch Normalization. Accelerating Deep Network Training by Reducing Internal Covariate Shift

What is the idea of Batch-Normalization (BN)



Batch Normalization

- Idea: Allow before each activation (non-linear transformation) to standardize the ingoing signal, but also allow to learn redo (partly) the standardization if it is not beneficial.



A BN layer performs a 2-step procedure with α **and** β **as learnable** parameter:

Step 1: $\hat{x} = \dfrac{x - avg_{batch}(x)}{stdev_{batch}(x) + \epsilon}$

$avg(\hat{x}) = 0$

$\text{stdev}(\hat{x}) = 1$

Step 2: $BN(x) = \alpha\hat{x} + \beta$

α learned

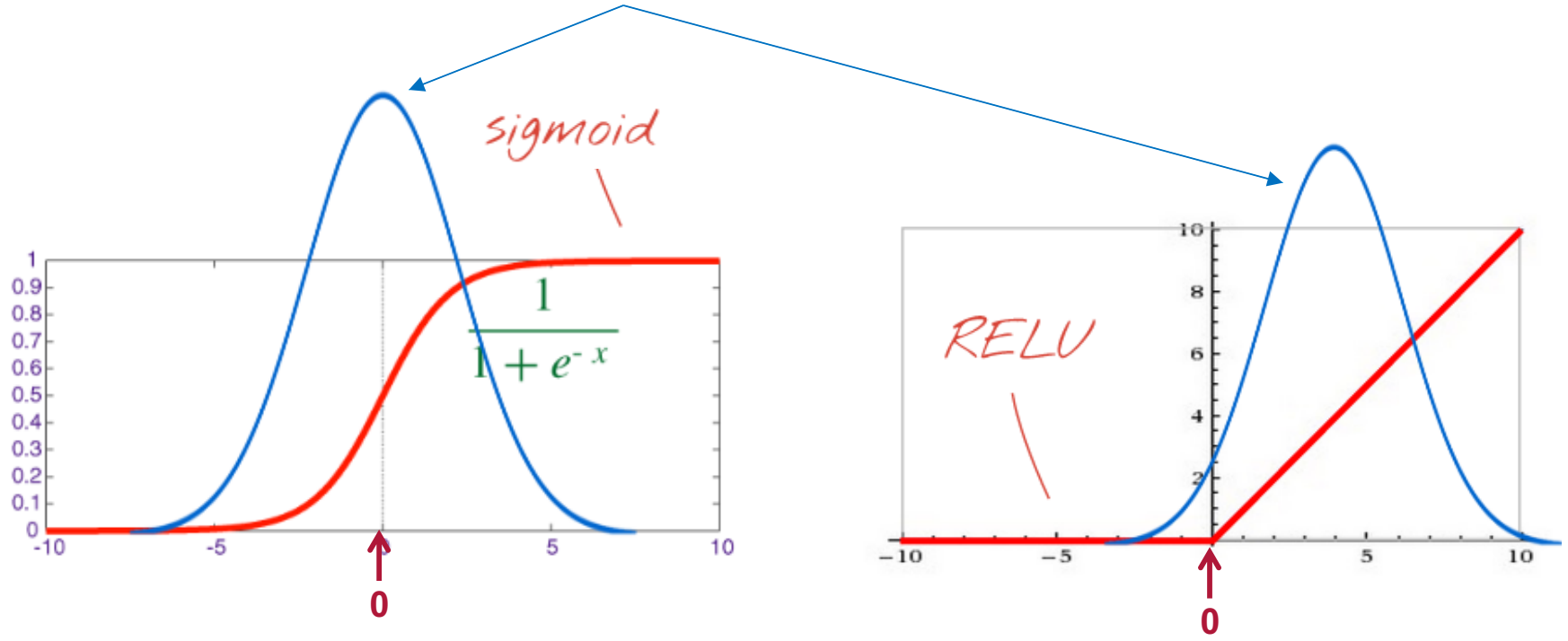
β learned

The learned parameters α and β determine how strictly the standardization is done. If the learned α=stdev(**x**) and the learned β=avg(**x**), then the standardization performed in step 1 is undone (for $\varepsilon \approx 0$) in step 2 and BN(**x**)=**x**.

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

Summary

Fully Connected Network

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

- Gradient Flow in Network
- Tricks:
  - ReLU instead of sigmoid activation
  - Regularization: early stopping, dropout (no detailed knowledge needed)
  - Batchnormalization for faster training (you just need to know how to apply it)
  - [Better random initialization]

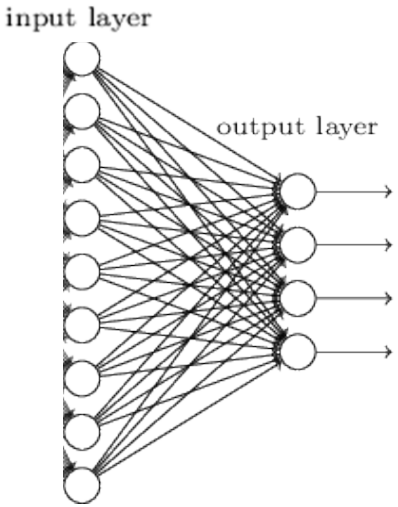
Next :

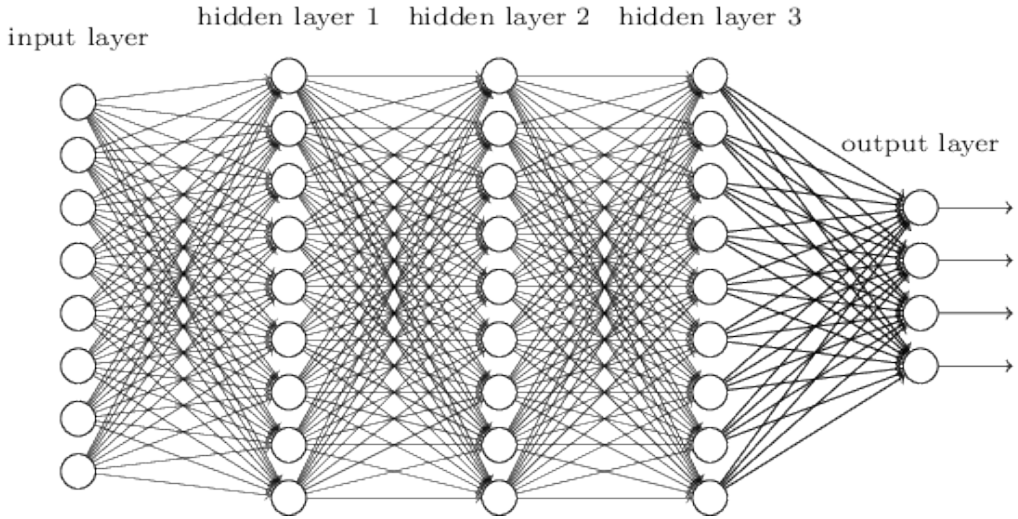
- Convolutional Neural Networks
  - The network starting the current hype in 2012

Convolutional Neural Networks

Today: We will go from fully connected NNs to CNNs

Fully connected Neural Networks (fcNN) without and with hidden layers:





Convolutional Neural Network:

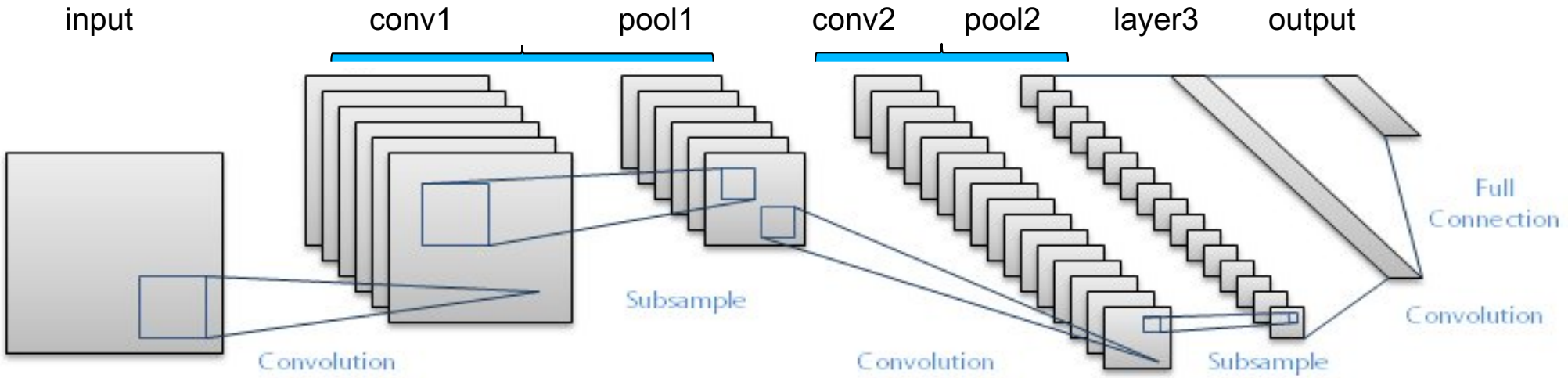


Image credits: http://neuralnetworksanddeeplearning.com/chap6.html, http://www.cemetech.net/projects/ee/scouter/thesis.pdf

Live Demo

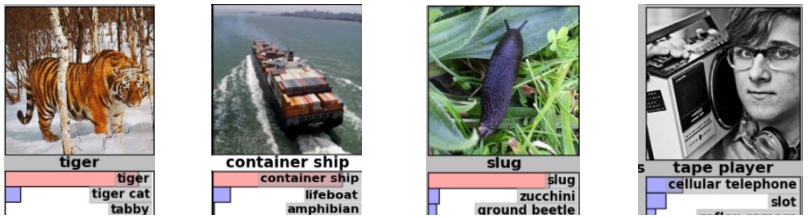
http://cs231n.stanford.edu/

History, milestones of CNN

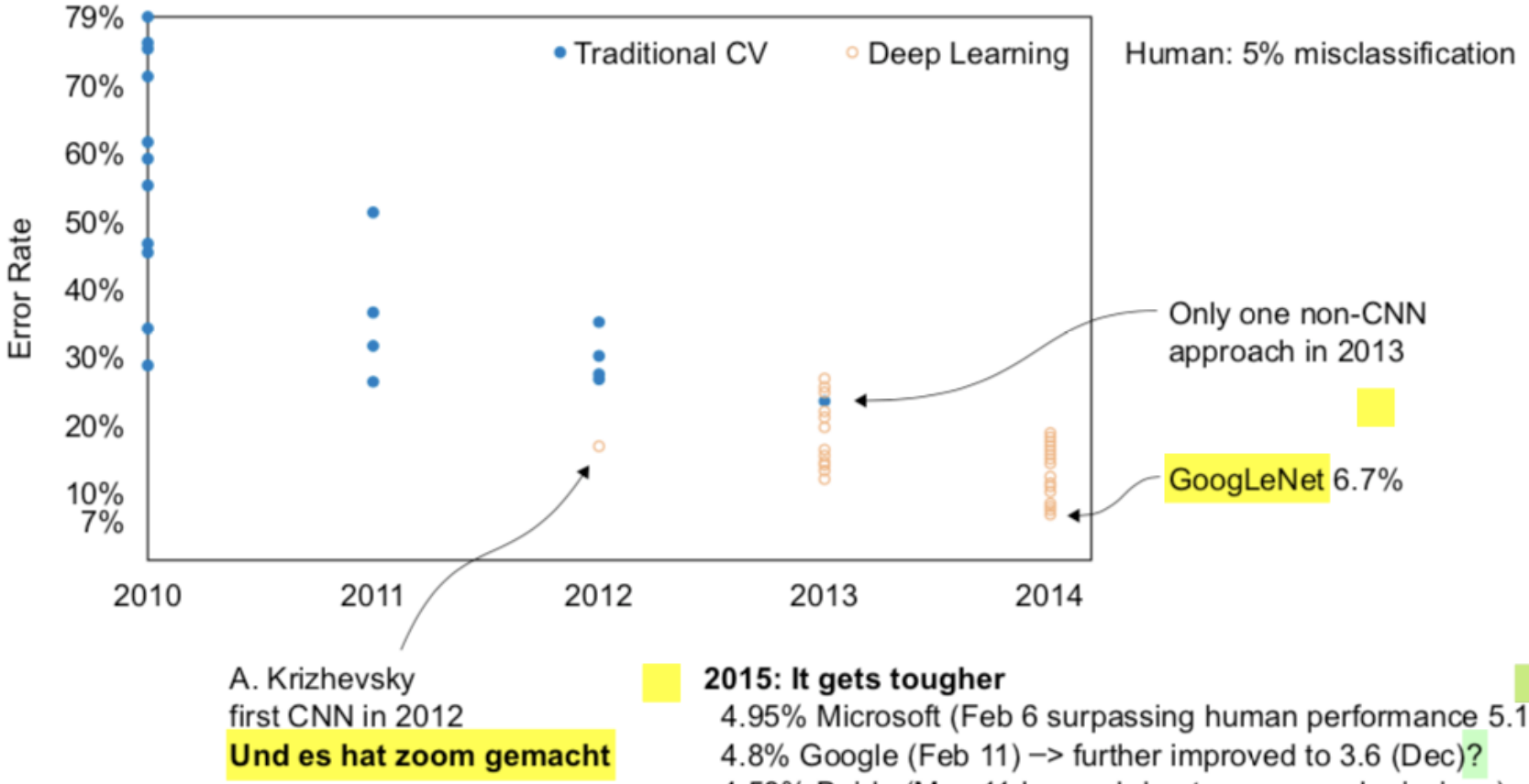
- 1980 Introduced by Kunihiko Fukushima
- 1998 LeCun (Backpropagation)
- Many contests won (IDSIA Jürgen Schmidhuber and Dan Ciresan et al.)
  - 2011& 2014 MINST Handwritten Dataset
  - 201X Chinese Handwritten Character
  - 2011 German Traffic Signs
- ImageNet Success Story
  - Alex Net (2012) winning solution of ImageNet…

Why DL: Imagenet 2012, 2013, 2014, 2015

1000 classes
1 Mio samples

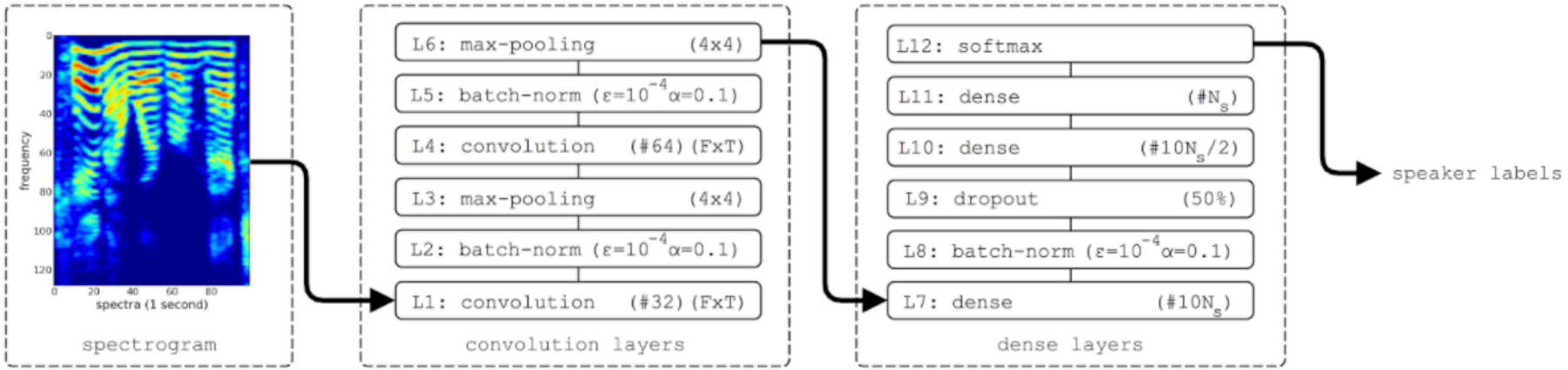


...



Project: Speaker Diarization (Who spoke when)

- Deep Learning can be used for audio
- While the big players solve speech recognition, we focus on a different problem to predict who spoke when



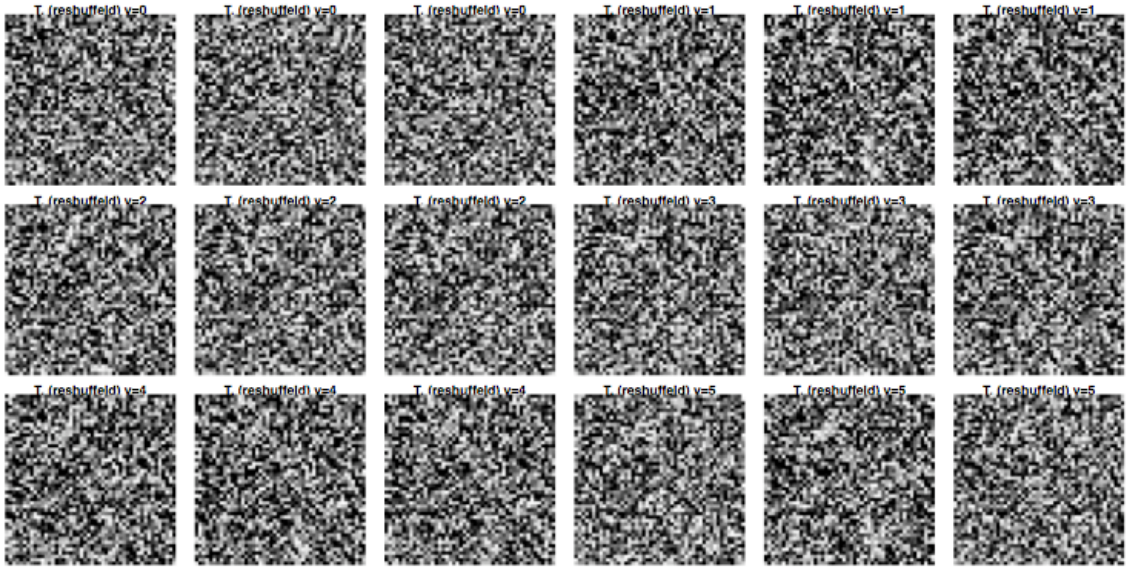
- Several project and bachelor thesis

Lukic, Yanick; Vogt, Carlo; Dürr, Oliver; Stadelmann, Thilo (2016). Speaker Identification and Clustering using Convolutional Neural Networks. In: Proceedings of IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2016)

How stupid is the machine II (revisited)

Aligned

Also for FCNN



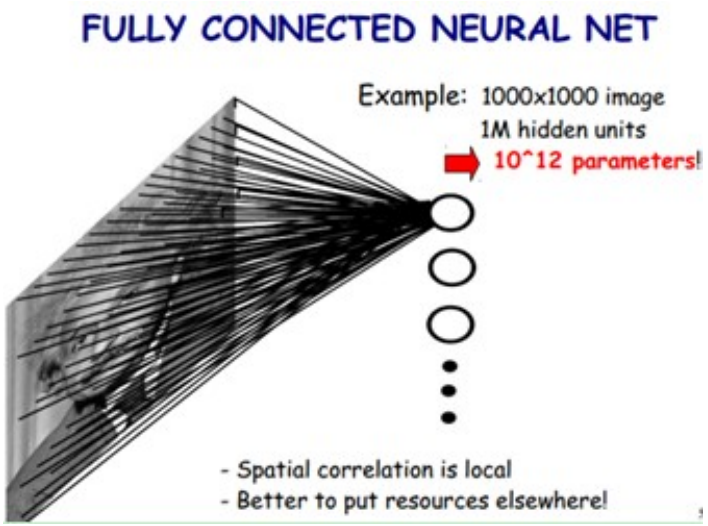
Aligned & Shuffeld

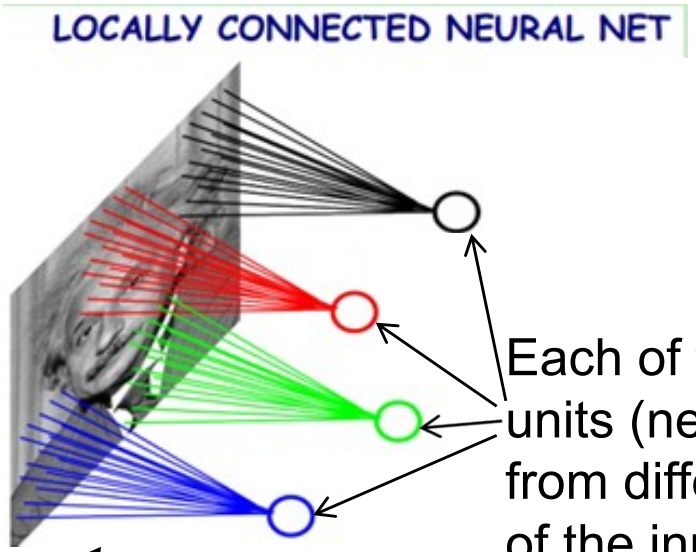
Guess the performance?

.	Eigenfaces	Fisher	SVM
Reshuffeled	0.7349	0.9163	0.9023
Original	0.7349	0.9163	0.9023

The previous algorithms are not robust against translations and don‘t care about the locality of the pixels!

Convolution extracts local information using few weights





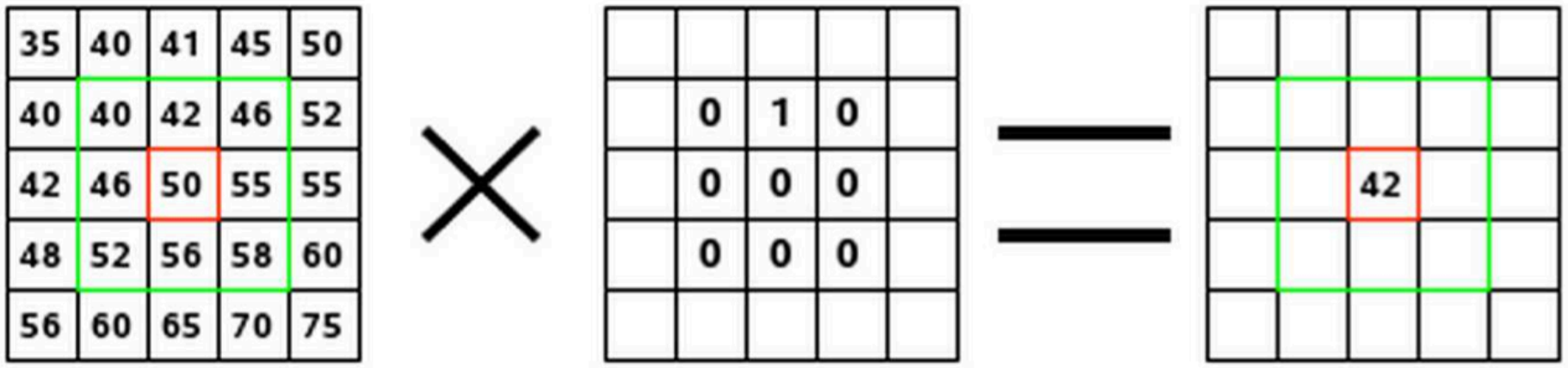
Each of these neural net units (neurons) extracts from different positions of the input image information about the presence of the same feature type, e.g. an edge.

Shared weights:

by using the same weights for each patch of the image we need much less parameters than in the fully connected NN and get from each patch the same kind of local feature information such as the presence of a edge.

CNN Ingredient I: Convolution

What is convolution?

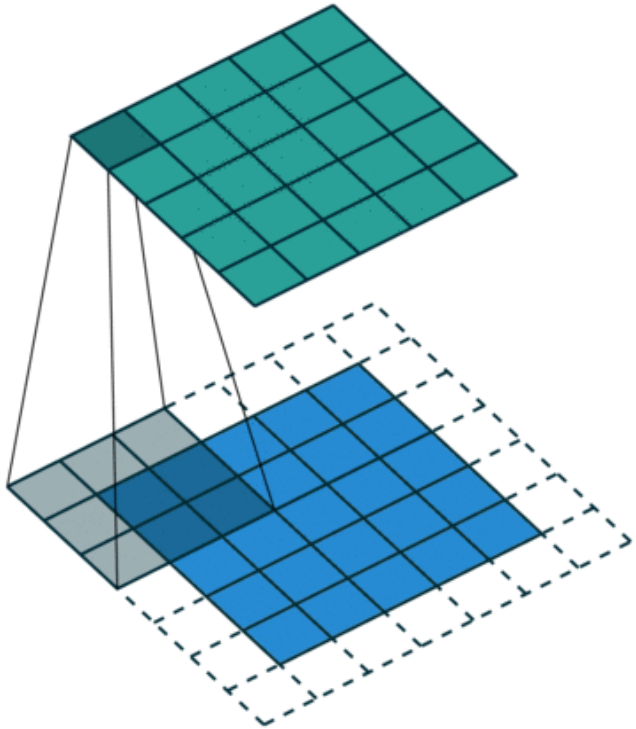


The 9 weights W_{ij} are called kernel (or filter)

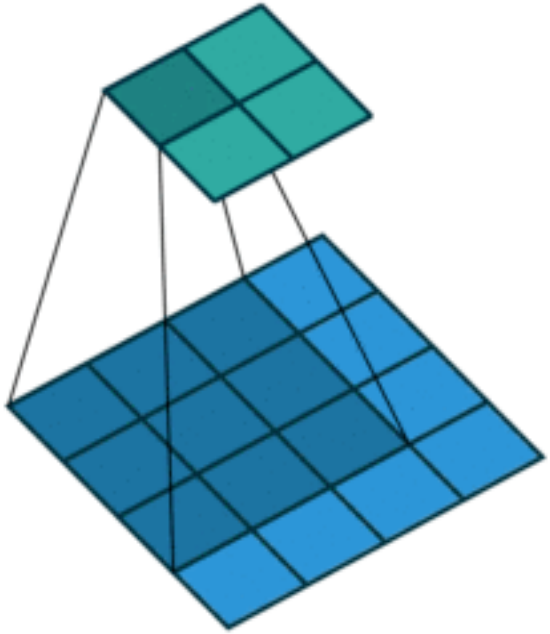
The weights are not fixed, they are learned!

Gimp documentation: http://docs.gimp.org/en/plug-in-convmatrix.html

CNN Ingredient I: Convolution



padding



no padding

The *same* weights are slid over the image

Illustration: https://github.com/vdumoulin/conv_arithmetic

CNN: local connectivity and weight sharing feature maps

In a locally connected network the calculation rule

$$z = b + \sum_i x_i w_i$$

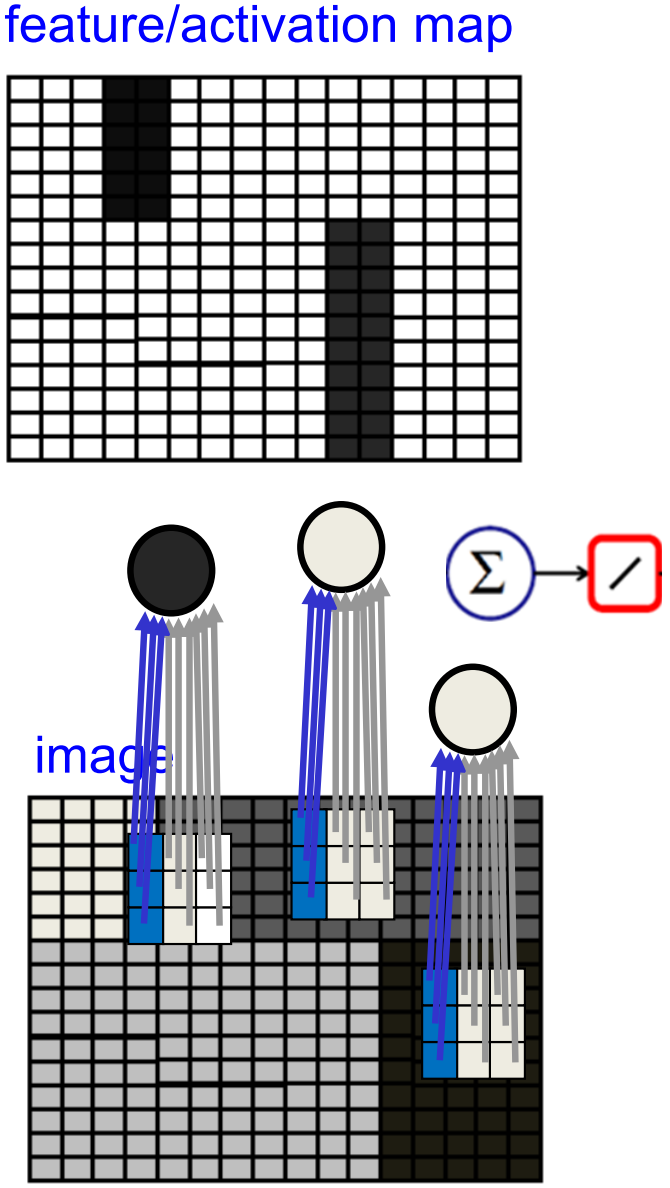
Corresponds to convolution of a filter with the image and the pattern of weights represent a filter.

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

0.9	-0.9	-0.9
0.9	-0.9	-0.9
0.9	-0.9	-0.9

The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

The results form again an image called feature map (=activation map) which shows at which position the feature is present.



Example of designed Kernel / Filter

Edge enhance
Filter

	0	0	0	
	-1	1	0	
	0	0	0	

But again!
The weights are not fixed. They are learned!

If time: http://setosa.io/ev/image-kernels/

Gimp documentation: http://docs.gimp.org/en/plug-in-convmatrix.html

One kernel or filter searches for specific local feature

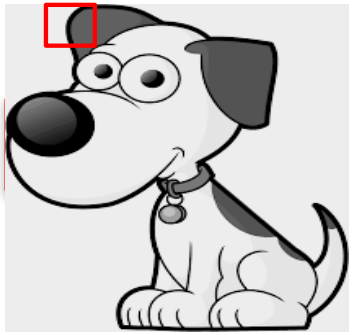


image patch

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

filter/kernel: curve detector

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

=6600

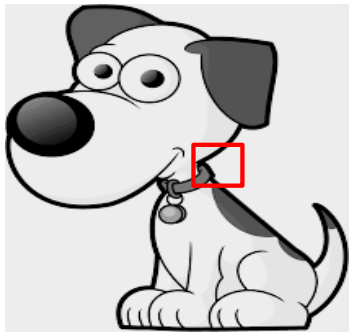


image patch

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

filter/kernel: curve detector

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

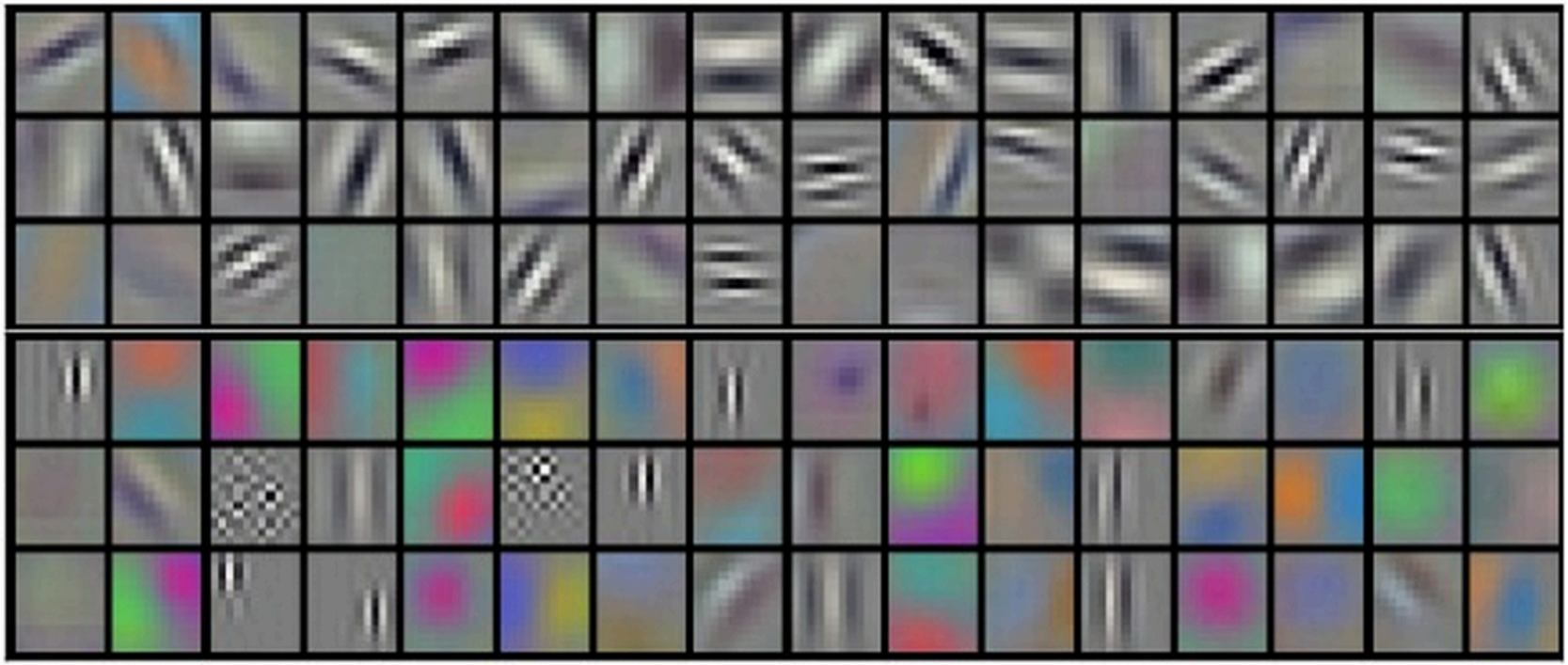
=0

We get a large resulting value if the filter resembles the pattern in the image patch on which the filter was applied.

credits: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

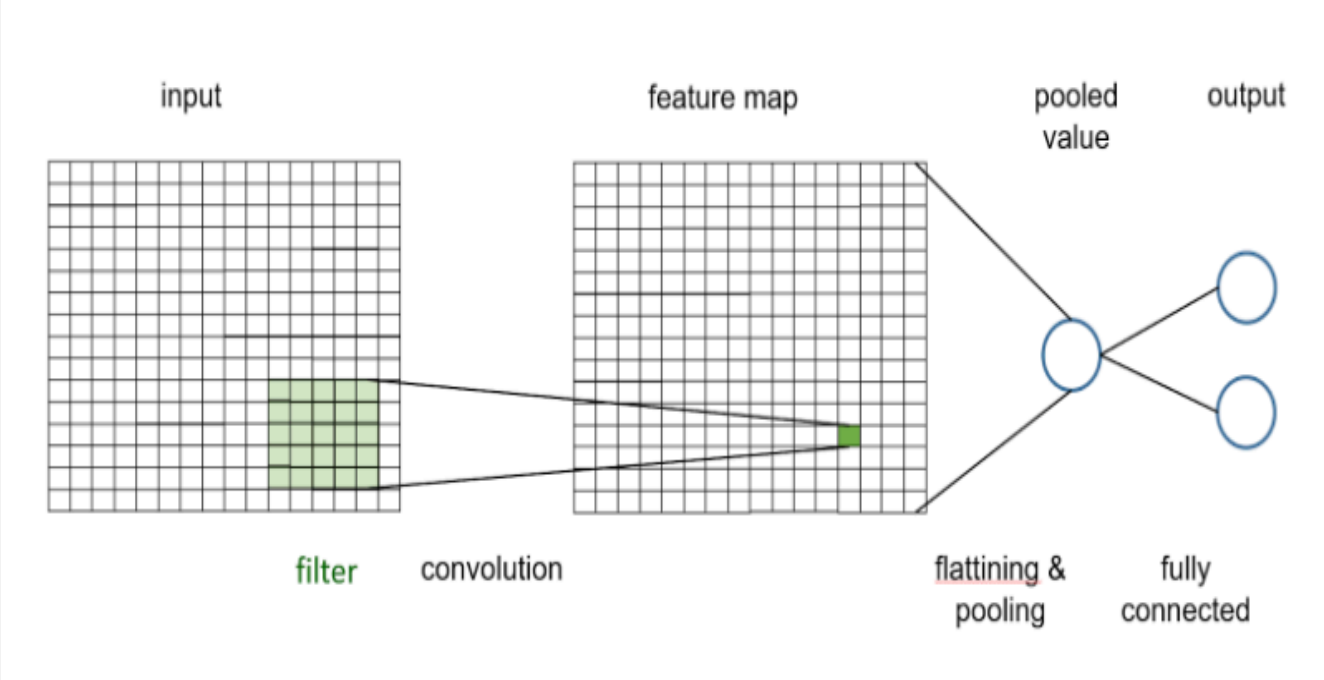
Example of learning filters

First Layer (11,11,3)



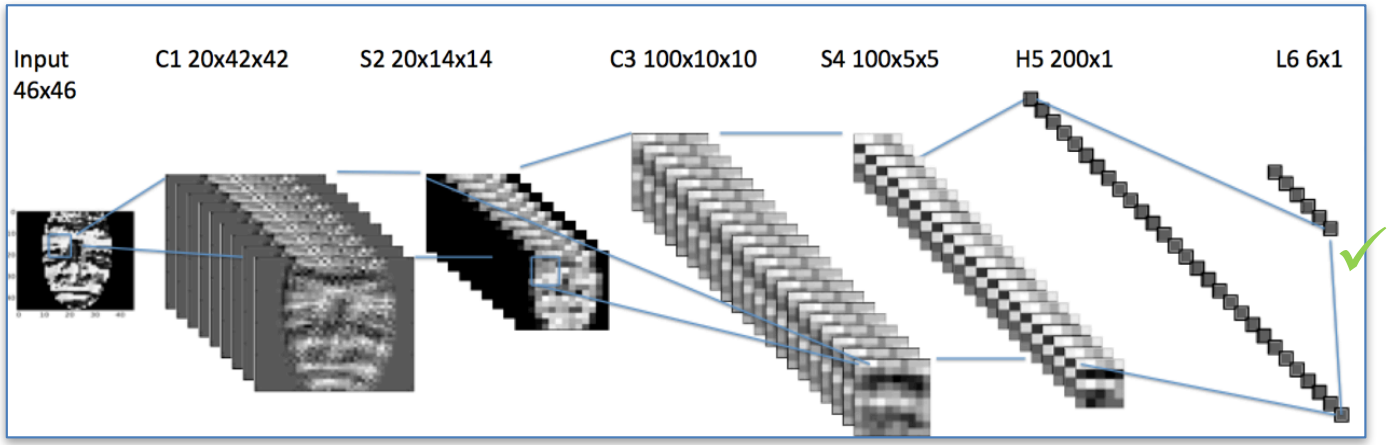
Examples of the 11x11x3 filters learned (Taken from Krizhevsky et al. 2012). Looks pretty much like old fashioned filters.

Exercise: Artstyle Lover



Open NB in: https://github.com/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_03.ipynb

Maxpooling Building Block



4x4

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

→

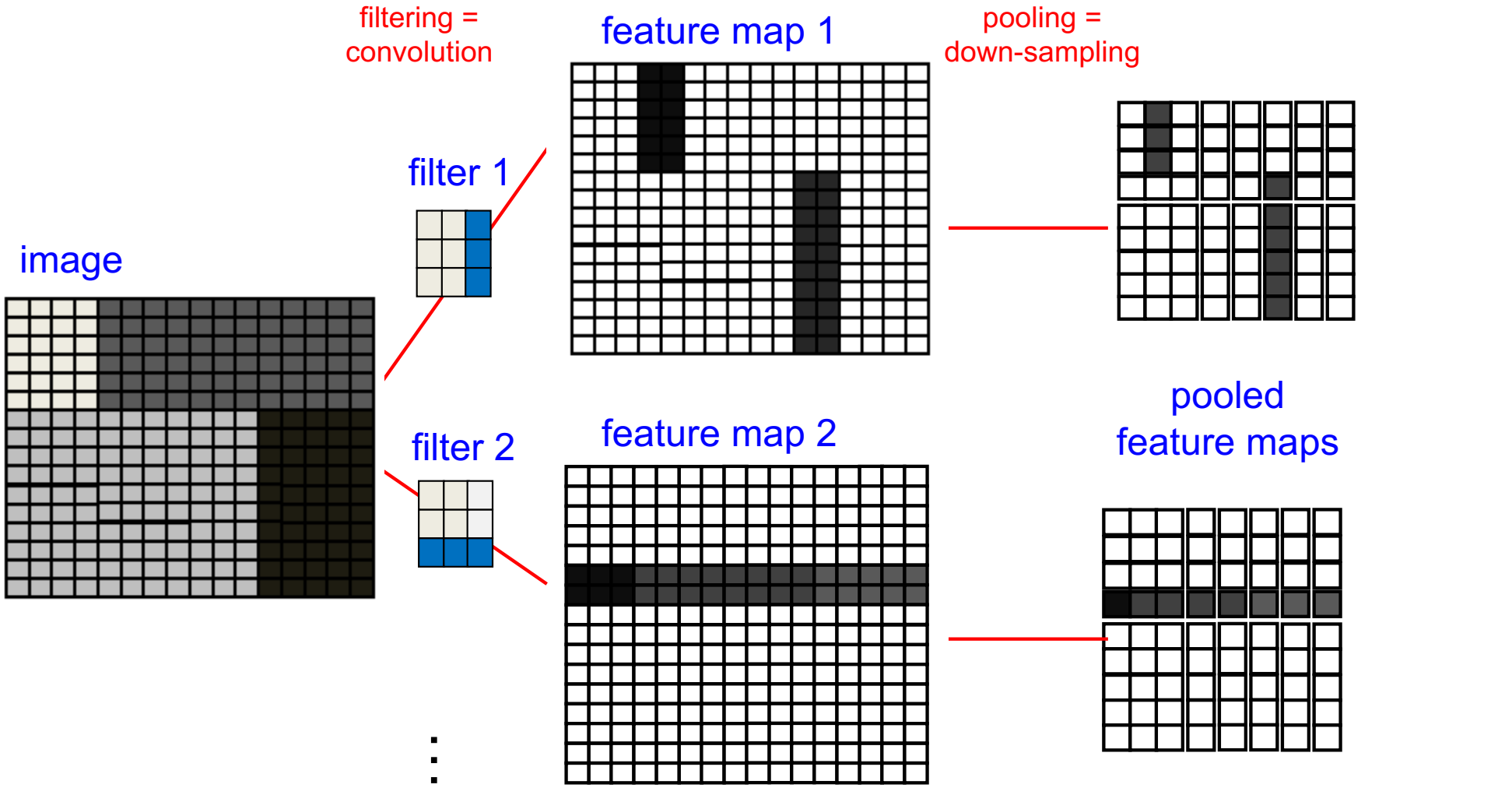
2x2

20	30
112	37

Simply join e.g. 2x2 adjacent pixels in one.

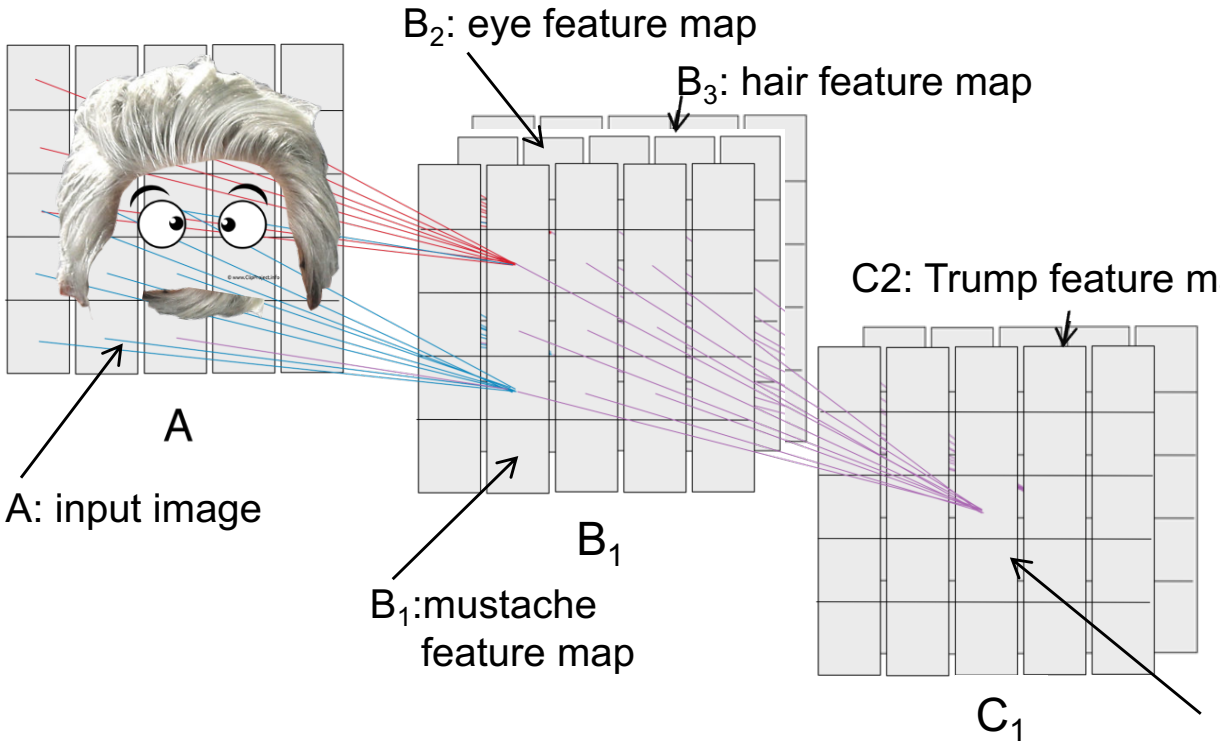
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

Propagating the features down



Slide thanks to Beate

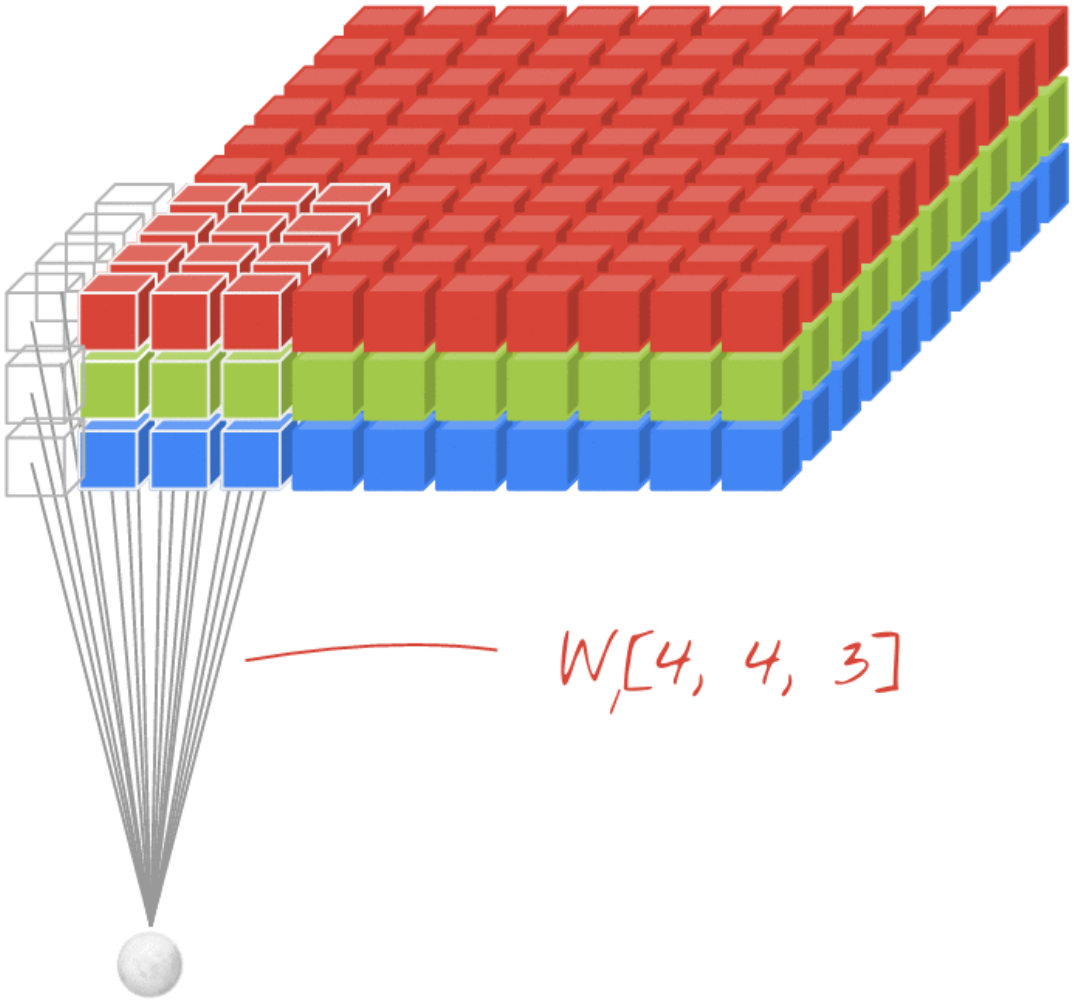
A simplified view: hierarchy of features



Filter cascade across different channels can capture relative position of different features in input image.

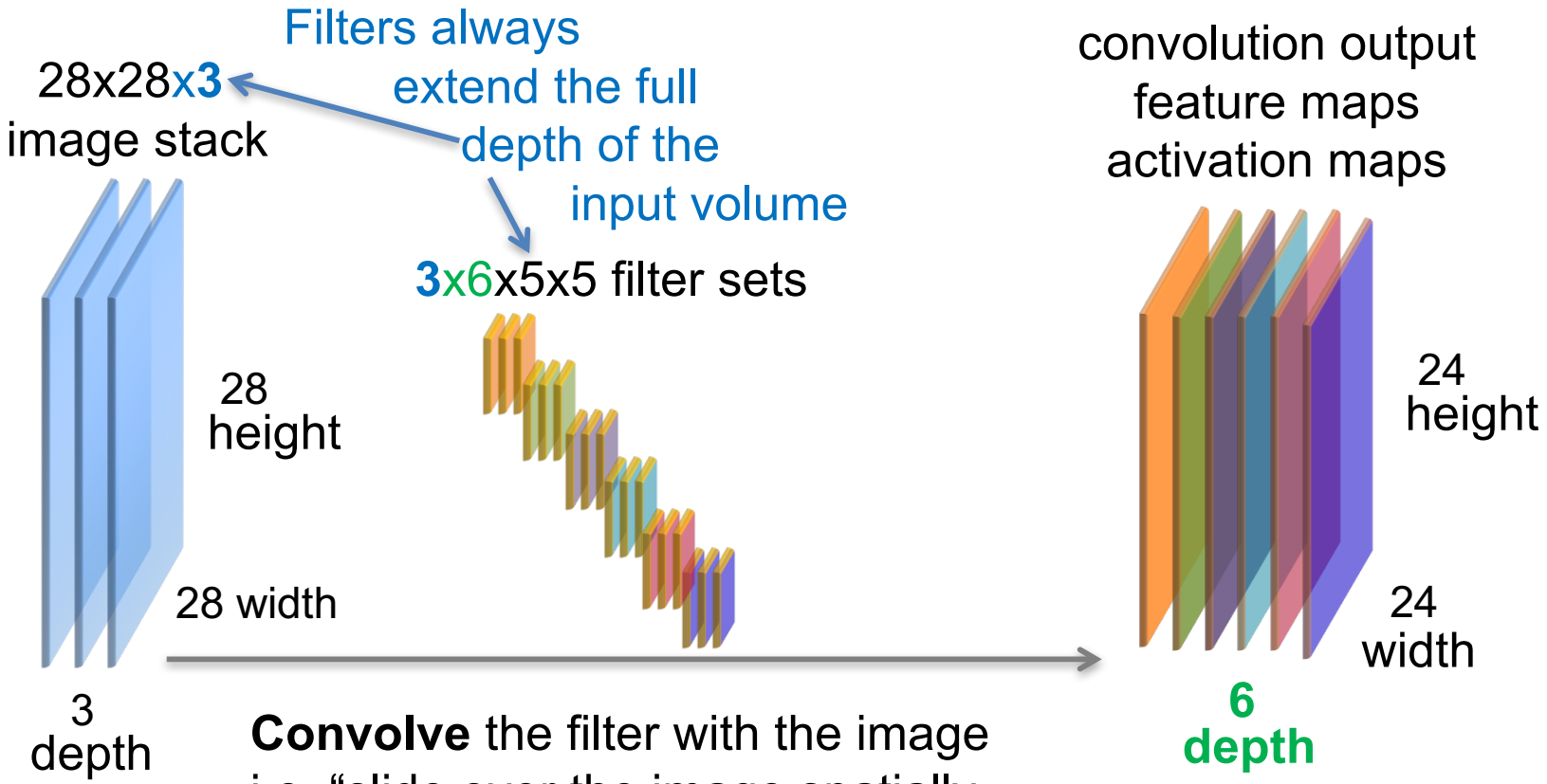
Einstein-face-filter will have a high value at expected mustache position.

Animated convolution with 3 input channels



Animation credits: M.Gorner, https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#10

Stacking it together filters are rank 3 tensors



Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products at each position"

Output size in case of
6 5x5x3 filter
no zero-padding
stride=1

Output: 24x24x6

Convention: This is called adding **6 filters**

Do the math

Keras code:

We add 32 convolutional filters (3x3)

```
model = Sequential()
model.add(Convolution2D(32, (3, 3), input_shape=(28, 28, 1)))
```

Input (None, 28, 28, 1)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320

Can you explain 320?

Do the math

Keras code:

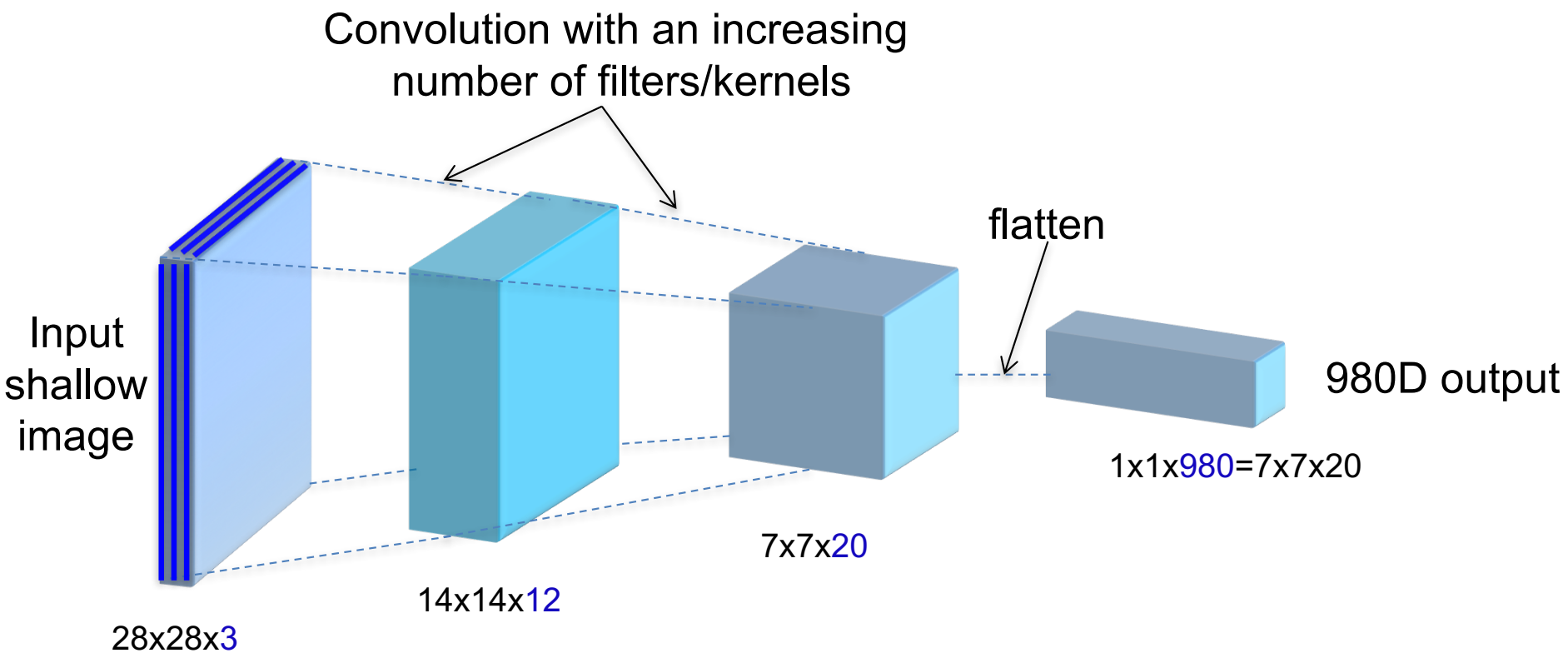
```
model = Sequential()
model.add(Convolution2D(16, (3,3))
```

(None, 14, 14, 8)

max_pooling2d_1 (MaxPooling2	(None, 14, 14, 8)	0
conv2d_3 (Conv2D)	(None, 14, 14, 16)	1168

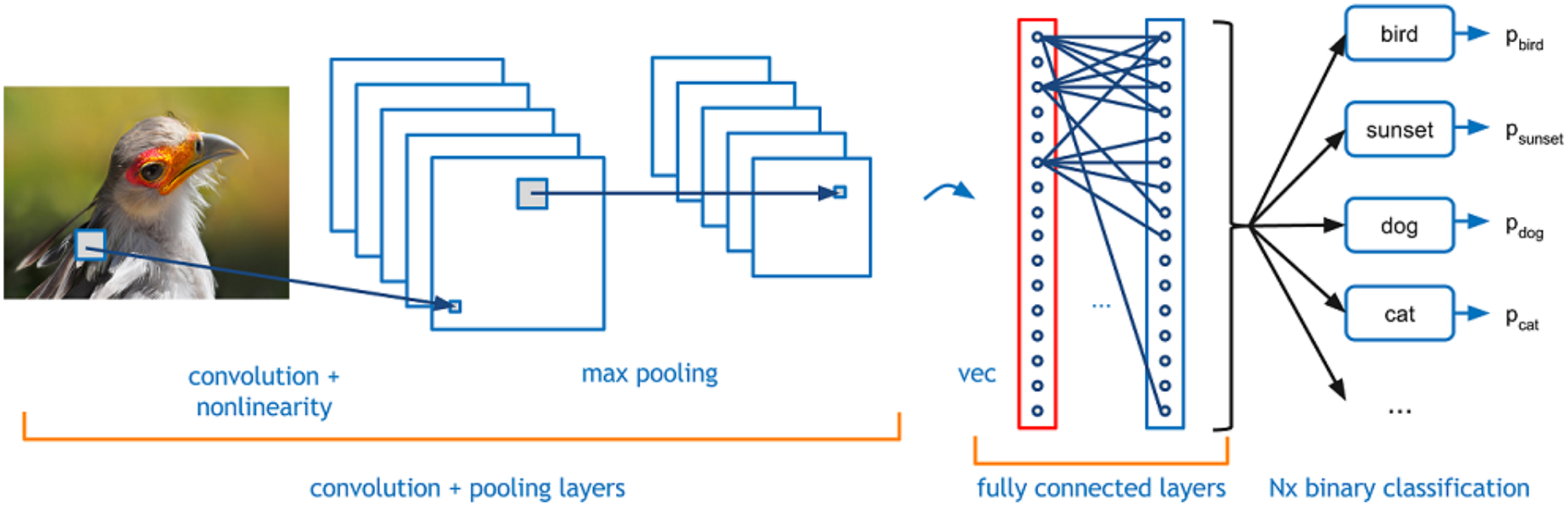
Can you explain 1168?

Typical shape of a classical CNN



Spatial resolution is decreased e.g. via max-pooling while more abstract image features are detected in deeper layers.

A classical CNN has fc layers at the end



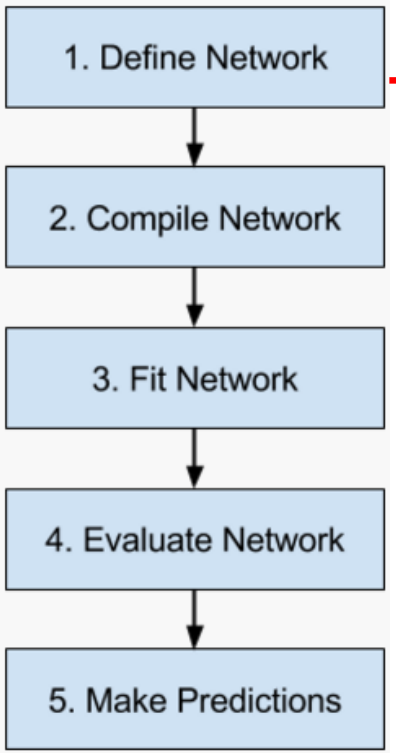
In a classical CNN we start with convolution layers and end with fc layers.

The task of the convolutional layers is to extract useful features from the image which might be appear at arbitrary positions in the image.

The task of the fc layer is to use these extracted features for classification.

Image credits:
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Keras: A high level API with best practice defaults



Number filter/feature maps (activation maps) in first hidden layers

```
model = Sequential()
# this applies 32 convolution filters of size 3x3 each.

model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape=(28, 28, 1)))
model.add(Activation('relu'))

model.add(Flatten())

model.add(Dense(10))
model.add(Activation('softmax'))
```

Will arrange the 32x28x28 = 25088 neurons, that are located in 32 feature maps of dim 28x28, in one vector.

```
model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
convolution2d_2 (Convolution2D)	(None, 28, 28, 32)	320	convolution2d_input_2[0][0]
activation_3 (Activation)	(None, 28, 28, 32)	0	convolution2d_2[0][0]
flatten_2 (Flatten)	(None, 25088)	0	activation_3[0][0]
dense_2 (Dense)	(None, 10)	250890	flatten_2[0][0]
activation_4 (Activation)	(None, 10)	0	dense_2[0][0]

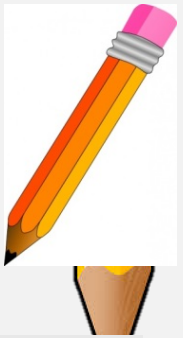
Exercise on setting up a simple CNN in keras

Check out the architecture of the CNN described in "live cnn in browser"

https://transcranial.github.io/keras-js/#/mnist-cnn

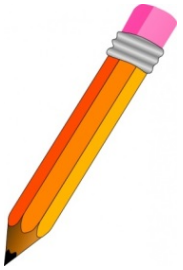
And fill in the pieces to get a Keras code for a model with this architecture
Check out the default parameters and the keras syntax starting from:

https://keras.io/layers/convolutional/ https://keras.io/layers/core/ or google…




```
model = Sequential()
model.add(Conv2D(filters= ...,
                 kernel_size=(..., ...),
                 input_shape=(..., ..., ...))
model.add(Activation('...'))
model.add(Conv2D(filters= ...,
                 kernel_size=(..., ...))
model.add(Activation(...))
model.add(MaxPooling2D(pool_size=(..., ...)))
model.add(Dropout(...))
model.add(Flatten())
model.add(Dense(...))
model.add(Activation('...'))
model.add(Dropout(...))
model.add(Dense(...))
model.add(Activation('softmax'))
```

Exercise on setting up a simple CNN in keras

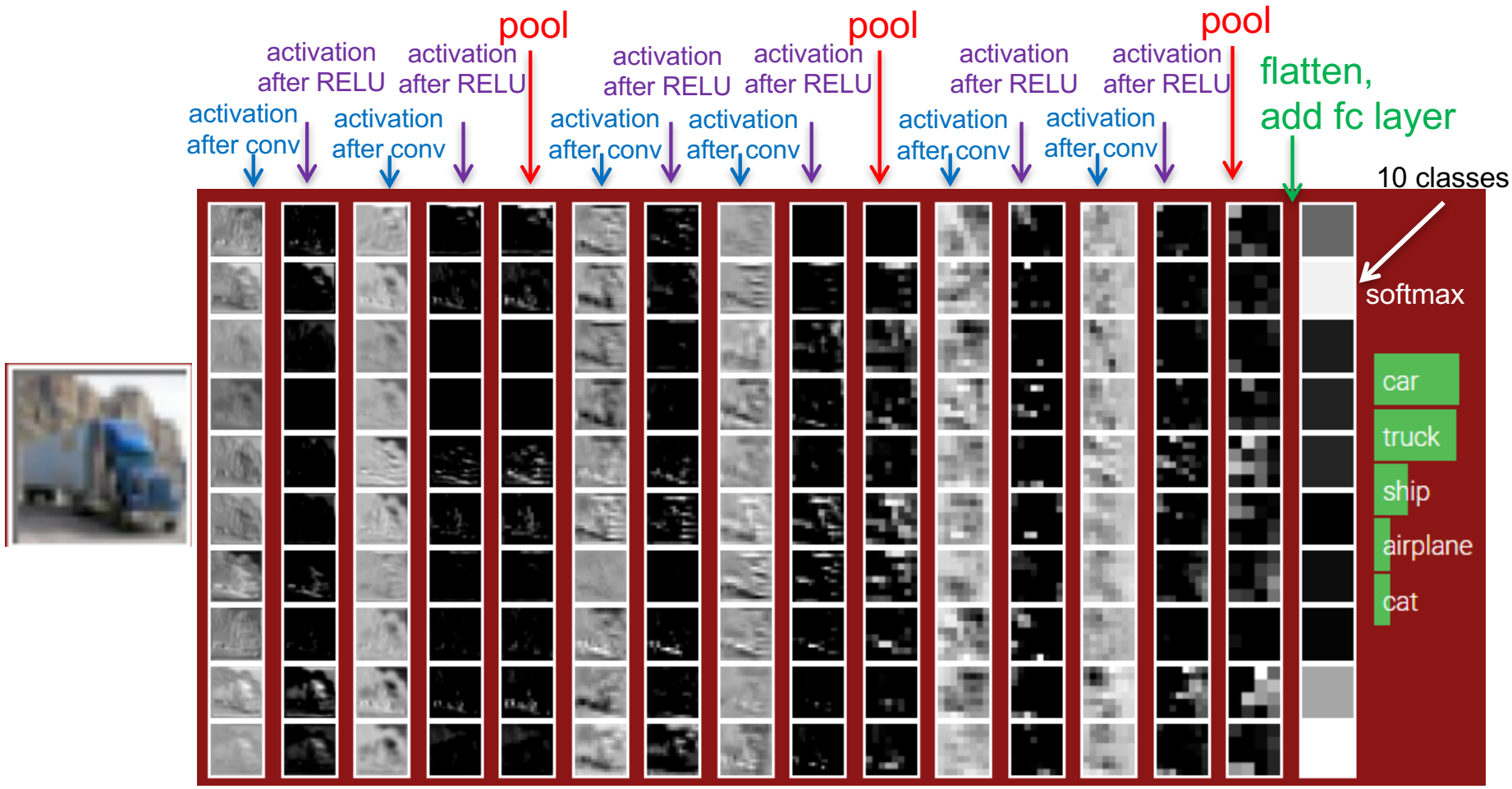


Check out the architecture of the CNN described in "live cnn in browser"
And fill in the pieces to get a Keras code for a model with this architecture

Solution:

```
model = Sequential()
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 input_shape=(28,28,1))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Appearance of activation/feature maps in different layers



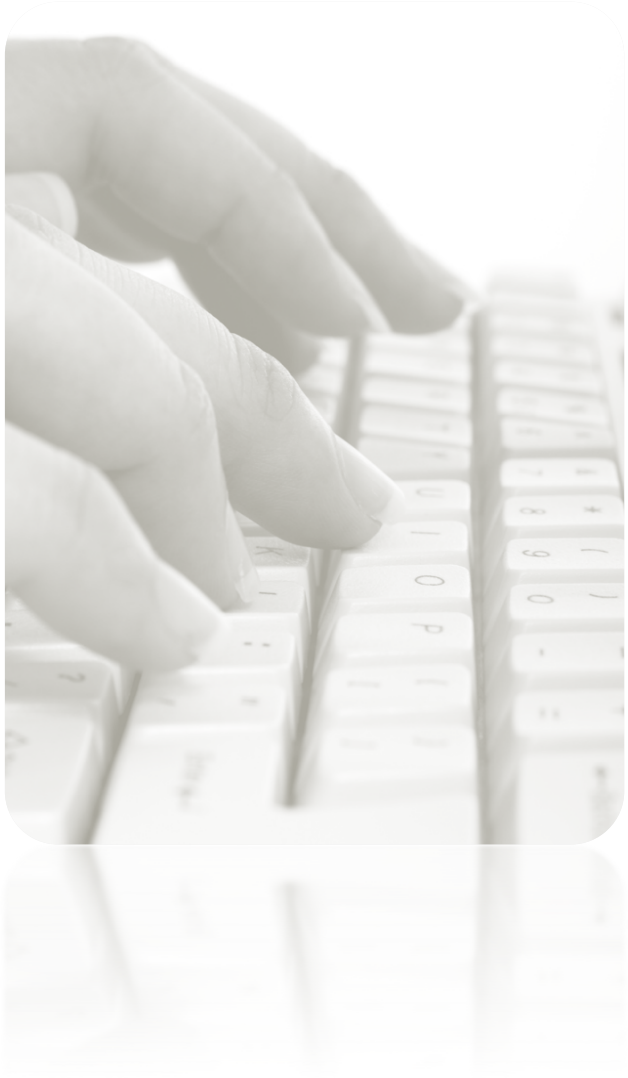
Activation maps give insight on the spatial positions where the filter pattern was found in the input one layer below (in higher layers activation maps have no easy interpretation)
-> only the activation maps in the first hidden layer correspond directly to features of the input image.

http://cs231n.stanford.edu/

See next lecture for understanding higher layers.

Exercise: use CNN for mnist classification

- Work through the instructions in 07 and 08 CNN Exercises **in day4** and use the ipython notebooks that are referred to.



Wrapping up today's story
Why going from fully connected NN to convolutional NN?

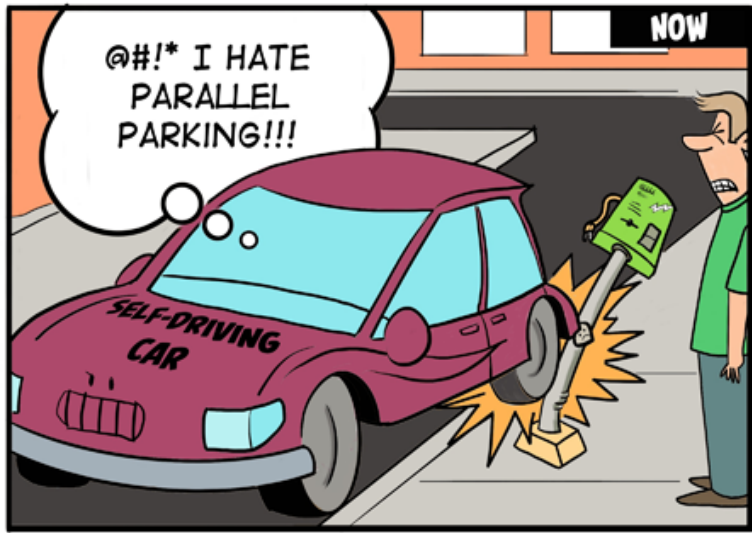
- We need to learn the features that a image is composed of.
- The classification should not depend so much on the location of the object in the image
- We want to exploit the information that is contained in the neighborhood structure of pixels in a image.

What kind of tasks can be tackled by CNNs?

Convolutional Neural Nets are used for **detecting patterns** in images, videos, sounds and texts.

Where are CNNs used already?

- Recommendation at Spotify, Amazon …
  (http://benanne.github.io/2014/08/05/spotify-cnns.html)
- Google, Facebook for image interpretation
  e.g. PlaNet—Photo Geolocation
  (http://arxiv.org/abs/1602.05314)
- Who else is using CNNs?
  (https://www.quora.com/Apart-from-Google-Facebook-who-is-commercially-using-deep-recurrent-convolutional-neural-networks)



Spotify amazon Google facebook …

Homework: Do some real stuff

Team-up for your first real DL project:

Develop a DL model to solve this task:

For a given image, decide which out of 8 celebrities is on the image.

Data:

For each of the 8 celebrities you get 250 images in the training data set, 50 images in the validation data set and 50 images in test data set.

Special challenge:

The images come from the OXFORD VGG Face dataset. The images were derived from the internet and automatically labeled. The data set contains also mislabeled images or ambiguous images.

Example images:

Label: Steve Jobs (entrepreneur)

Label: Emma Stone (actress)