CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

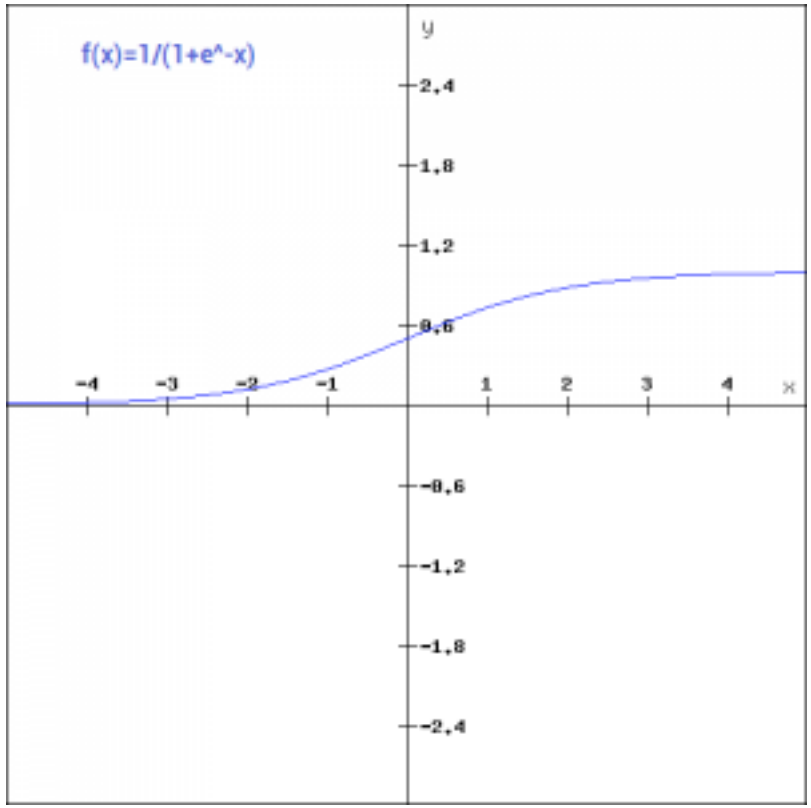
Neural networks

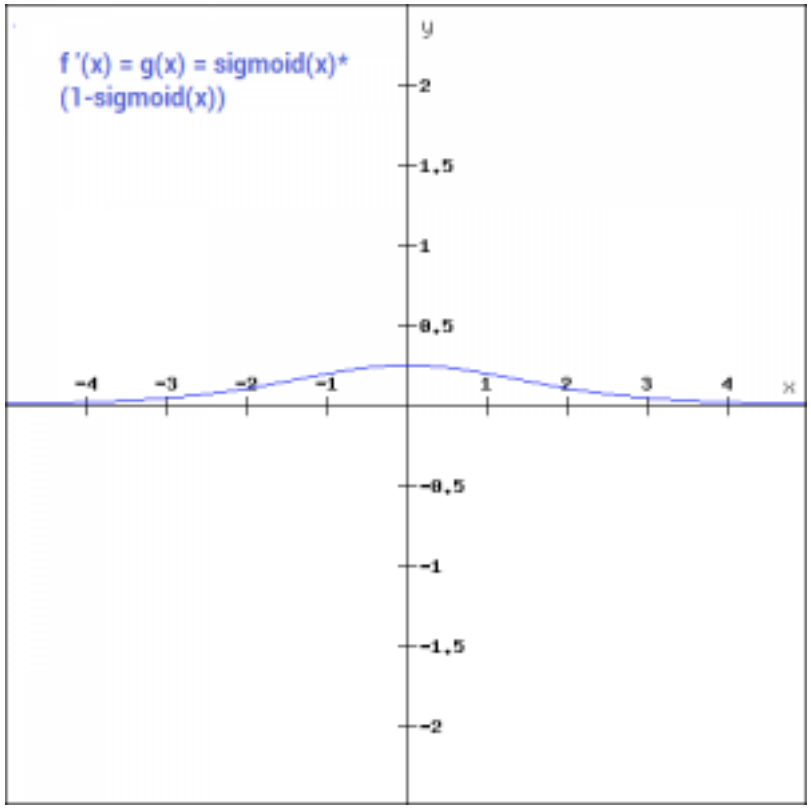
- Fully connected networks
  - Neuron
  - Non-linearity
  - Softmax layer
- DNN training
  - Loss function and regularization
  - SGD and backprop
  - Learning rate
  - Overfitting – dropout, batchnorm
- CNN, RNN, LSTM, GRU <- This class

Notes on non-linearity

- Sigmoid

  Models get stuck if fall go far away from 0. Output always positive



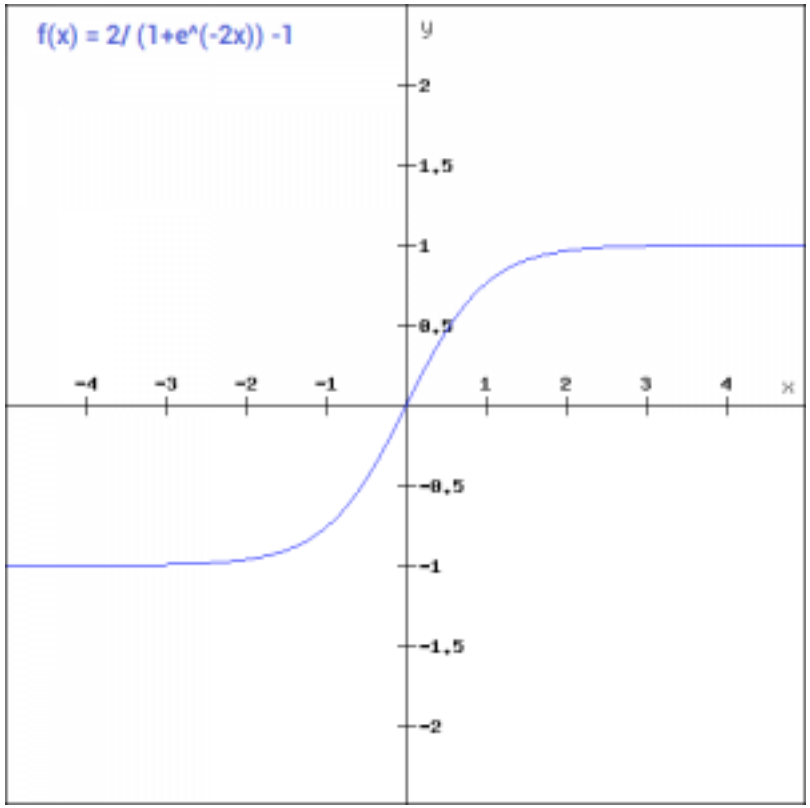


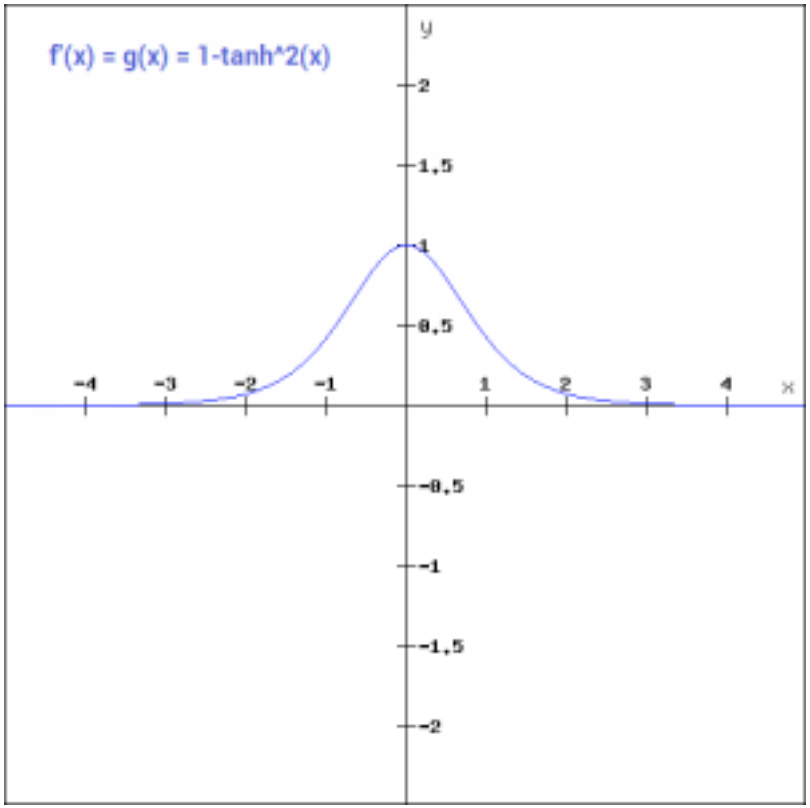
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

Notes on non-linearity

- Tanh

Output can be +-. Models get stuck if fall go far away from 0



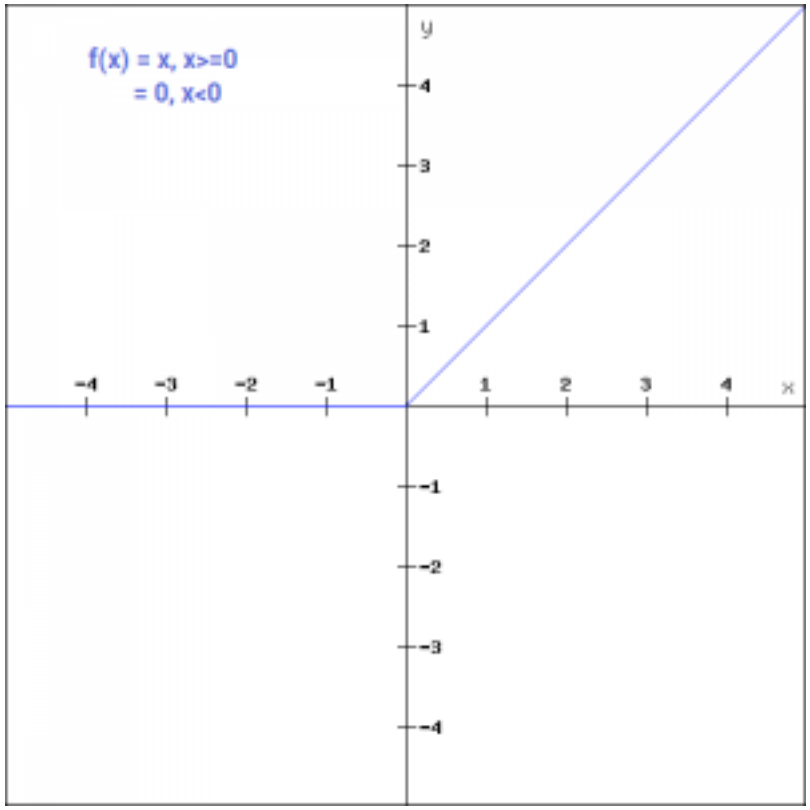


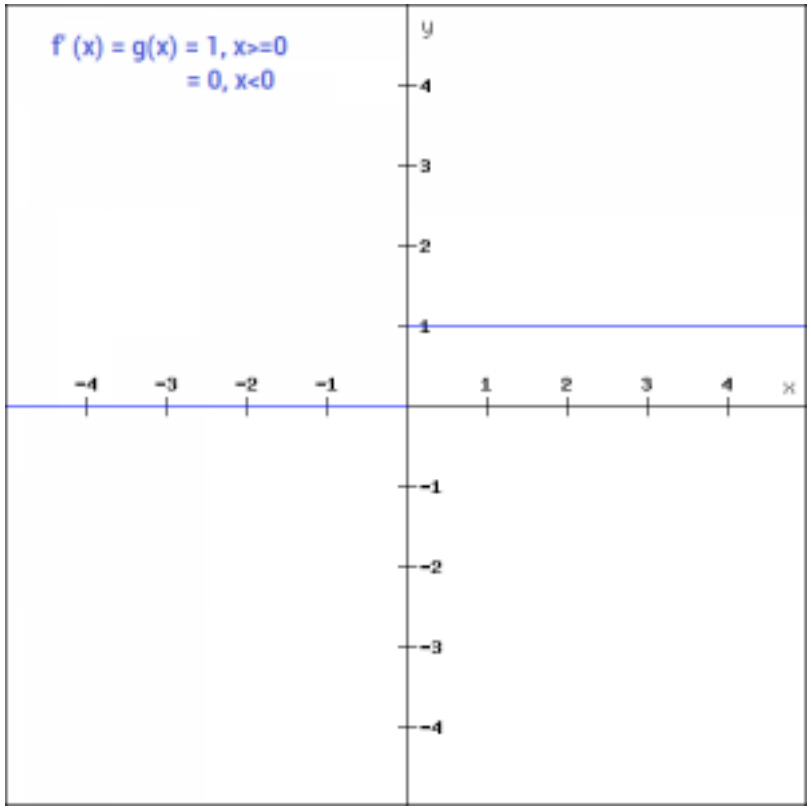
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

Notes on non-linearity

- ReLU

  High gradient in positive. Fast compute. Gradient doesn't move in negative



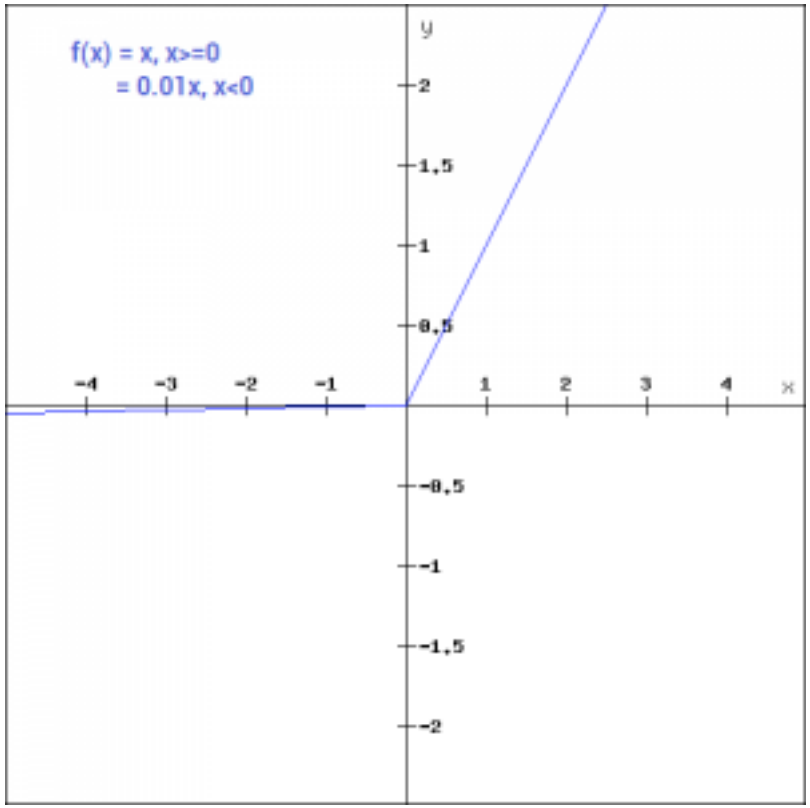


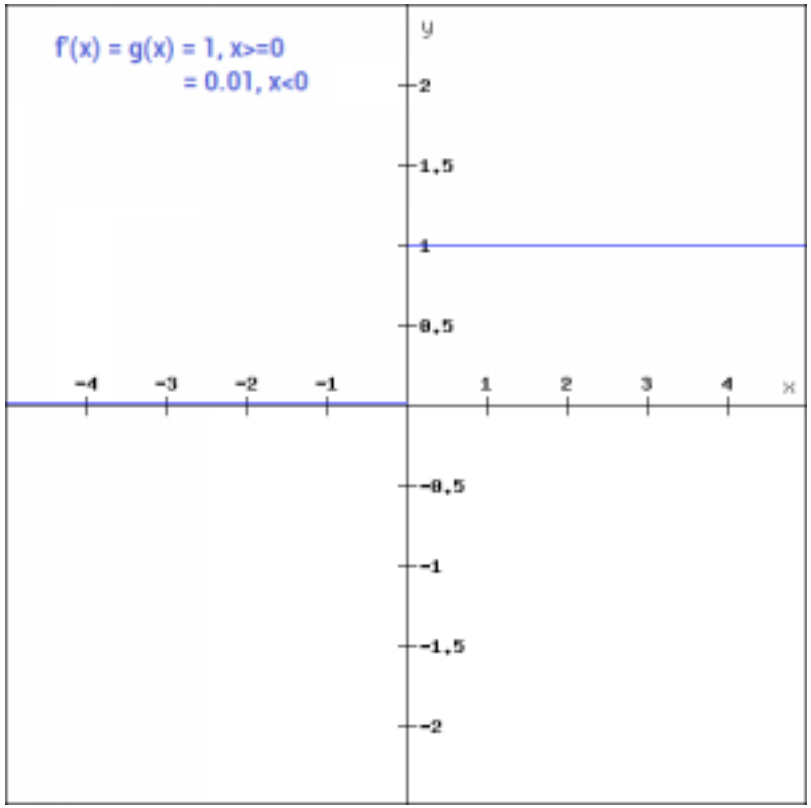
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

Notes on non-linearity

- Leaky ReLU

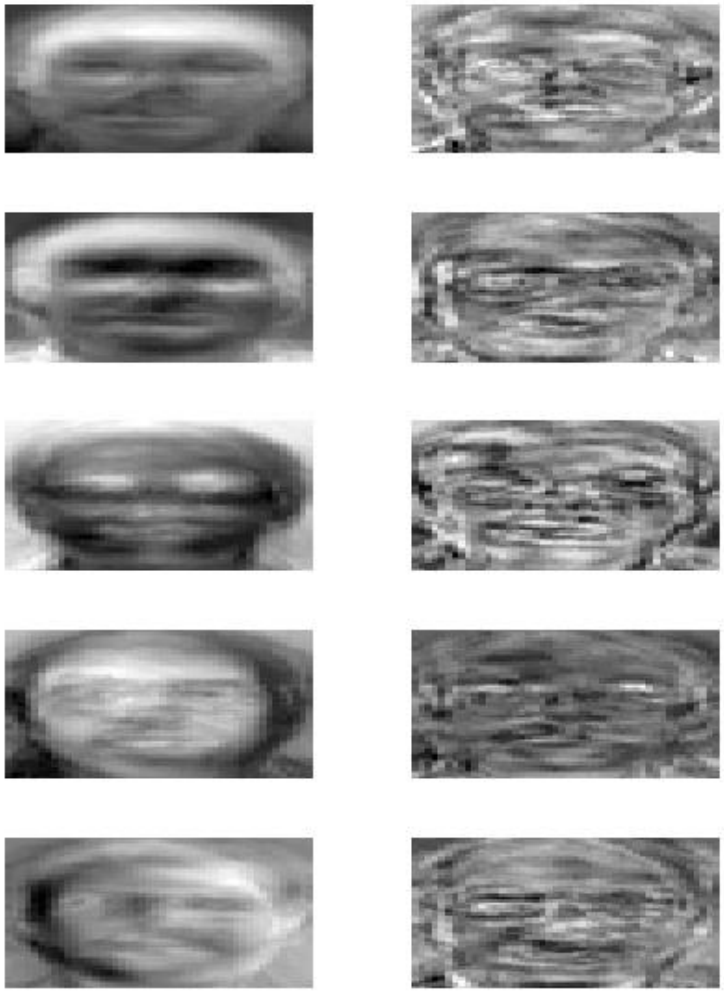
  Negative part now have some gradient. Real task doesn't seem that much better than ReLU





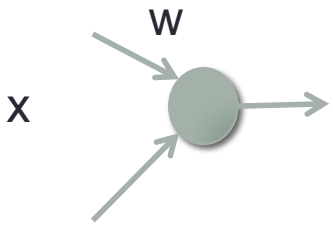
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

So we learn about basis and projections



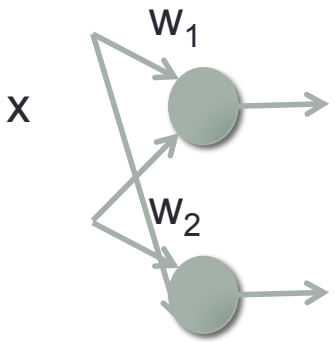
Projections and Neural network weights

- $w^T x$



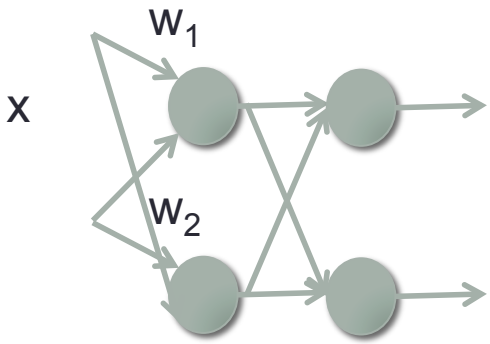
Projections and neural network weights

- $W^T x$



Projections and neural network weights

- $W^T x$



fisher projection = $V^T W^T x$
= $(WV)^T x$

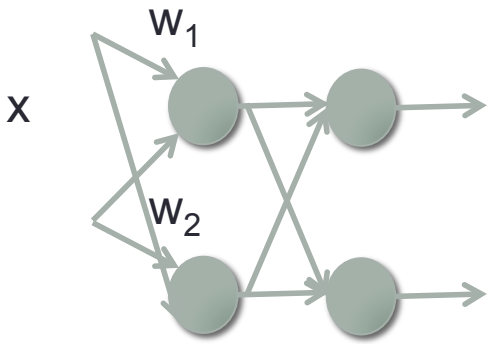
x = y

LDA projections

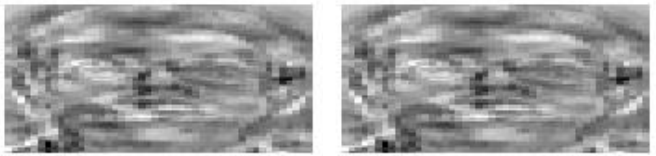
y

Projections and neural network weights

- $W^T x$



fisher projection = $V^T W^T x$
$= (WV)^T x$



LDA projections

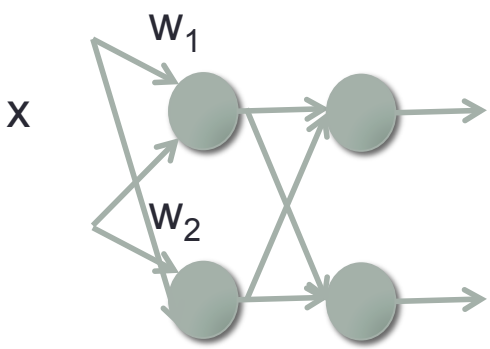
x = y

y

Projections and neural network weights

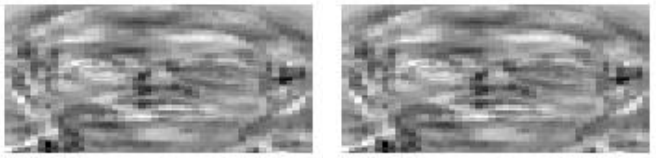
- $W^T x$

- Neural network layers as feature transform
- Non-linearity prevents merging of layers



fisher projection = $V^T W^T x$
$= (WV)^T x$

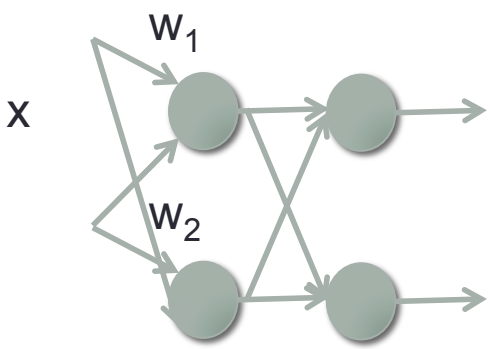
Wv_1 Wv_2



LDA projections

Shift in feature space

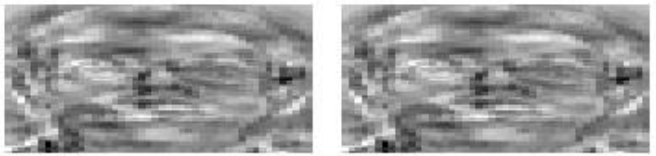
- $W^T x$



- What happens if I have a person that is off-frame?
- Need another filter that is shifted
- Can we do better?

fisher projection = $V^T W^T x$
$= (WV)^T x$

Wv_1 Wv_2



LDA projections

x = y

y

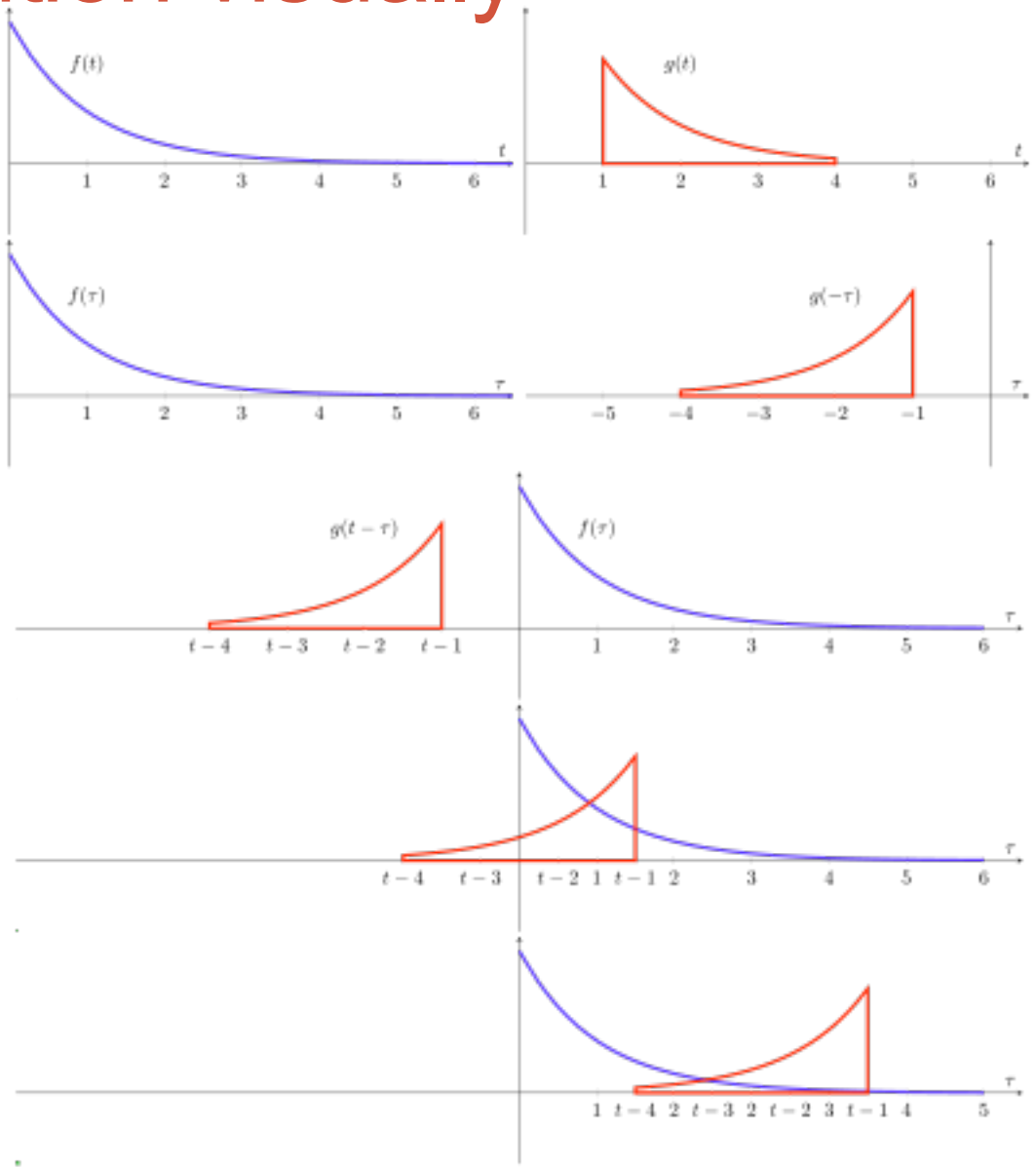
Convolution

- Continuous convolution

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$
$$= \int_{-\infty}^{\infty} f(t - \tau) g(\tau)\, d\tau.$$

- Meaning of (t-T) : Flip then shift

Convolution visually



Demo

https://en.wikipedia.org/wiki/Convolution

Convolution discrete

- Discrete convolution

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$
$$= \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

- Continuous convolution

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$
$$= \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau.$$

- Same concept as continuous version

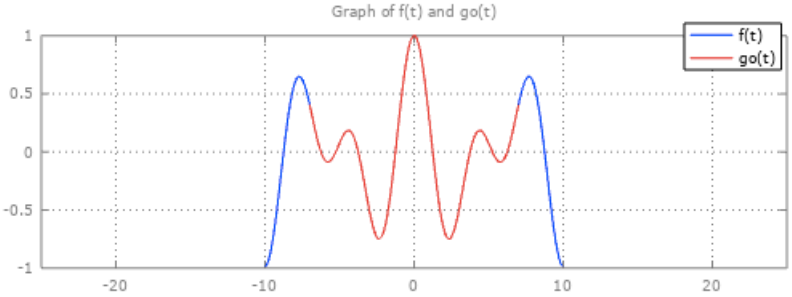
Matched filters

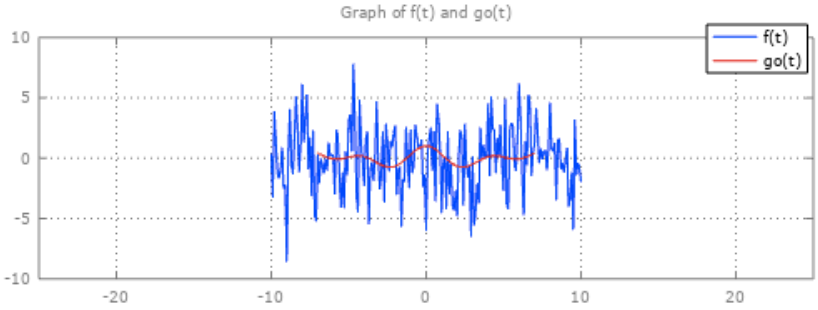
- We can use convolution to detect things that match our pattern
- Convolution can be considered as a filter (Why? Take ASR next semester ☺)
- If the filter detects our pattern, it will show up as a nice peak even if there are noise.

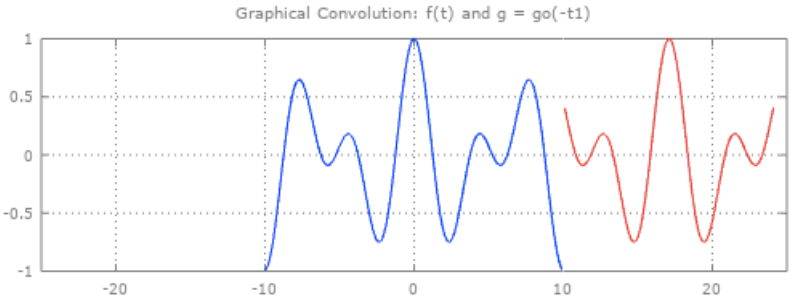
- Demo

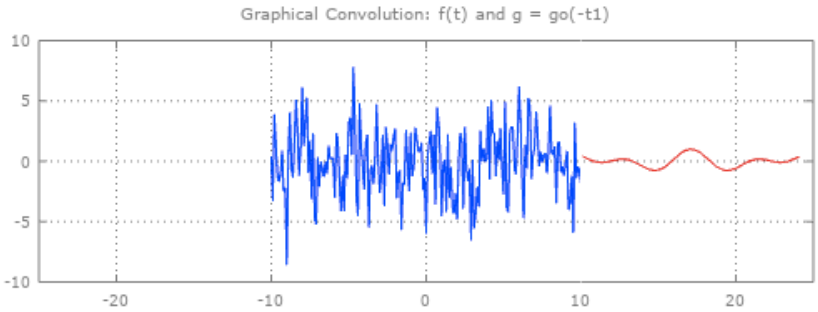
Matched filters

Red: matched filter
Blue: signal

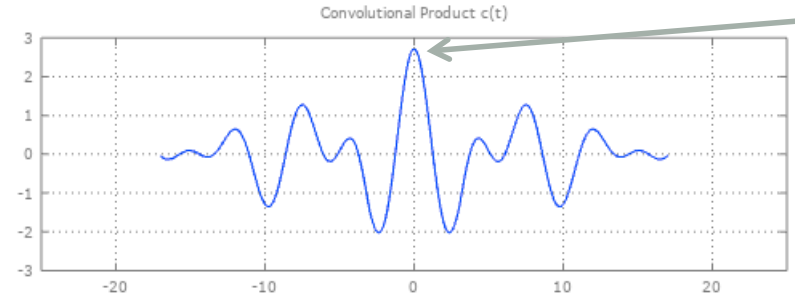


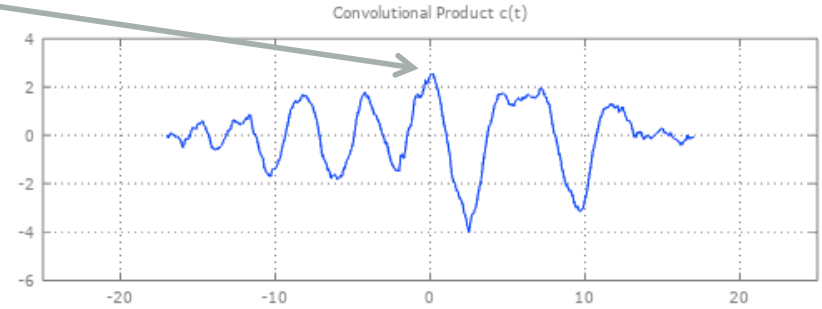






Matched peak





Convolution and Cross-Correlation

- Convolution

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau = \int_{-\infty}^{\infty} f(t-\tau) g(\tau)\, d\tau.$$

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m] g[m]$$

- (Cross)-Correlation

$$(f \star g)(\tau) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f^*(t)\, g(t+\tau)\, dt,$$

$$(f \star g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\, g[m+n].$$

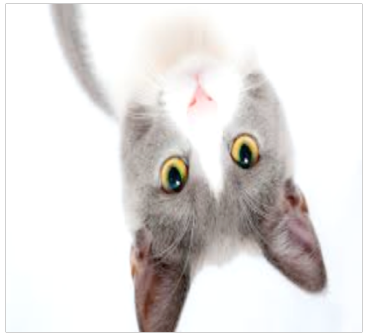
Convolution and cross-correlation are the same if g(t) is symmetric (even function).

For some unknown reason, people use convolution in CNN to mean cross-correlation.

From this point onwards, when we say convolution we mean cross-correlation.

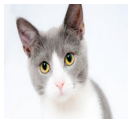
2D convolution

- Flip and shifts in 2D



- But, we no longer flips

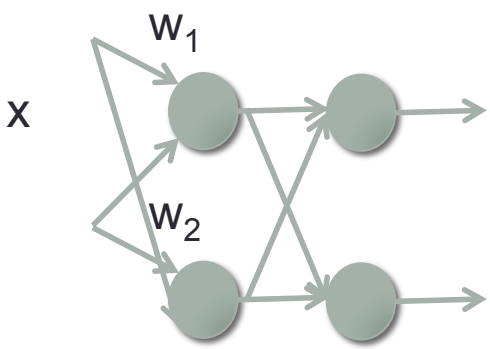
Our match filter



Will get some peak here

Shift in feature space

- $W^T x$

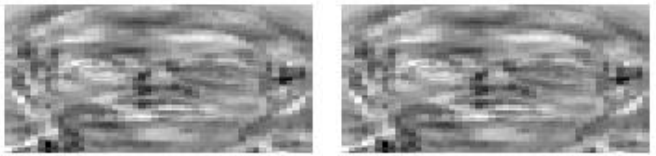


- What happens if I have a person that is off-frame?
- Need another filter that is shifted

w_1 w_2 x = y

fisher projection = $V^T W^T x$
= $(WV)^T x$

Wv_1 Wv_2



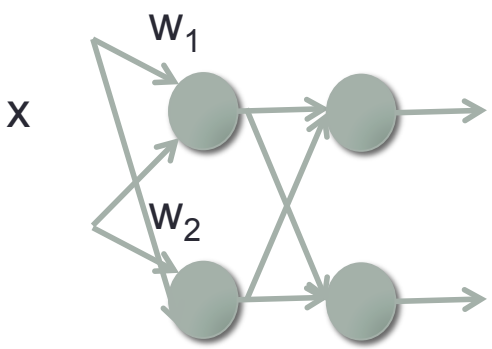
LDA projections

v_1 v_2 y

Shift in feature space

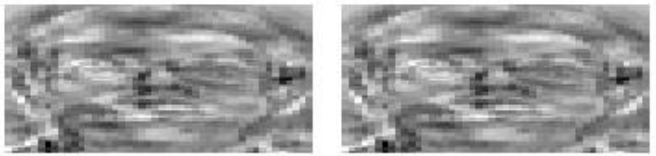
- $W^T x$

- What happens if I have a person that is off-frame?
- Ans: Convolution with W as filter



fisher projection = $V^T W^T x$
$= (WV)^T x$

Wv_1 Wv_2



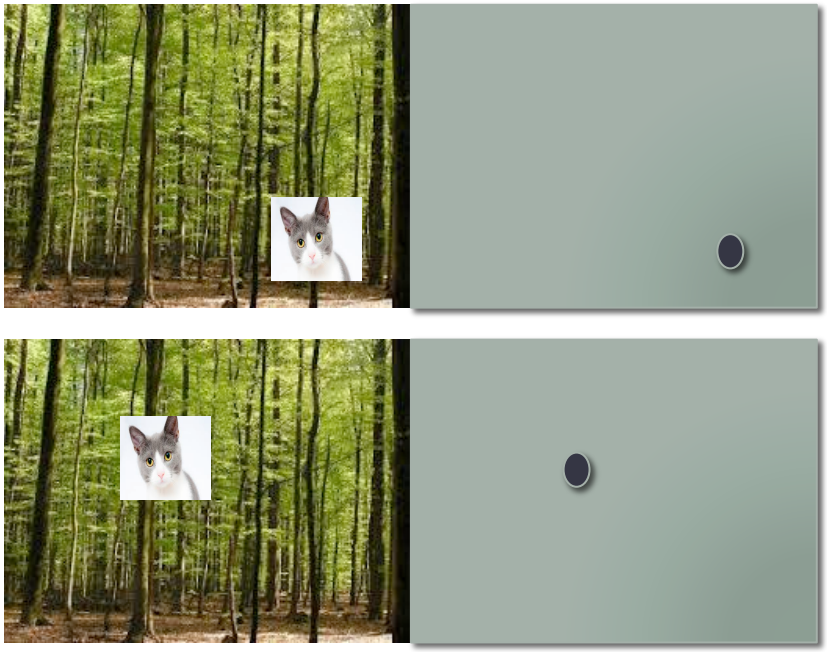
LDA projections

x = y

y

Convolutional Neural networks

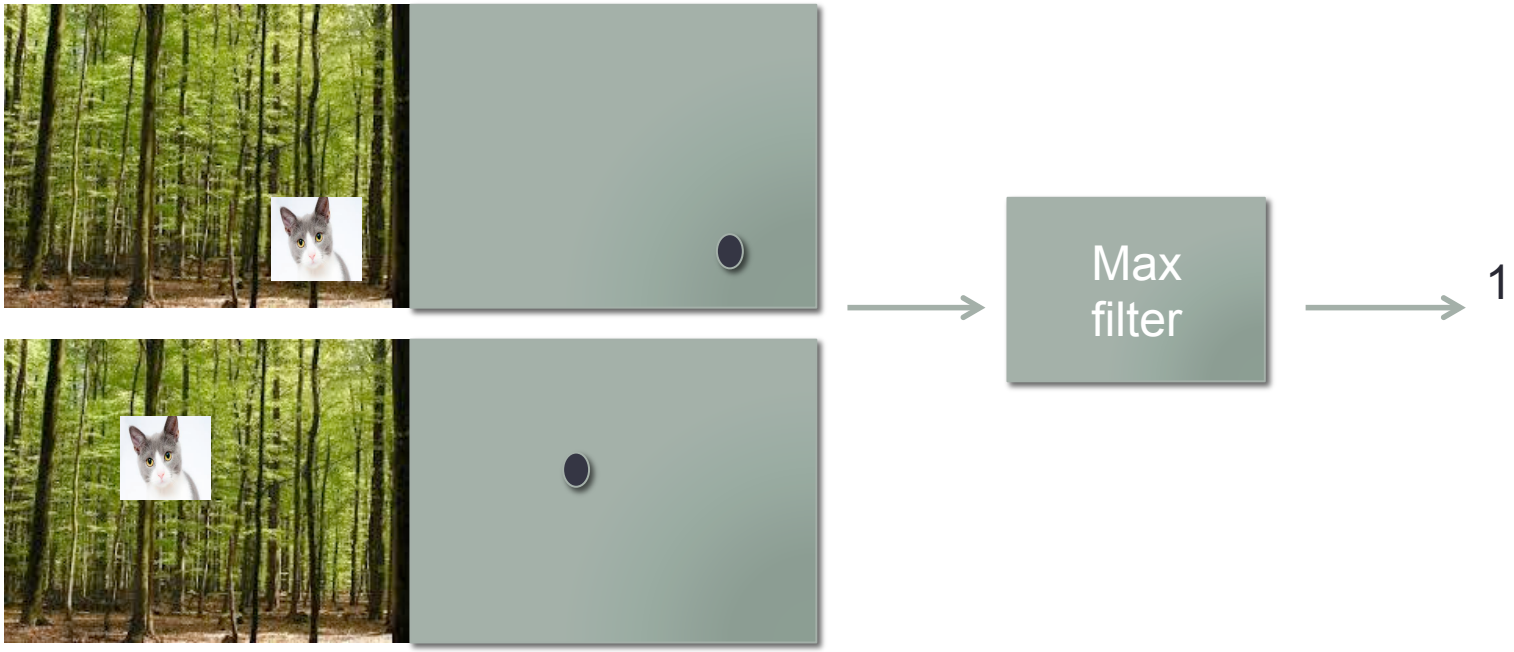
- A neural network with convolutions! (cross-correlation to be precise)
- But we have peaks at different location



From the point of view of a network, these are two different things.

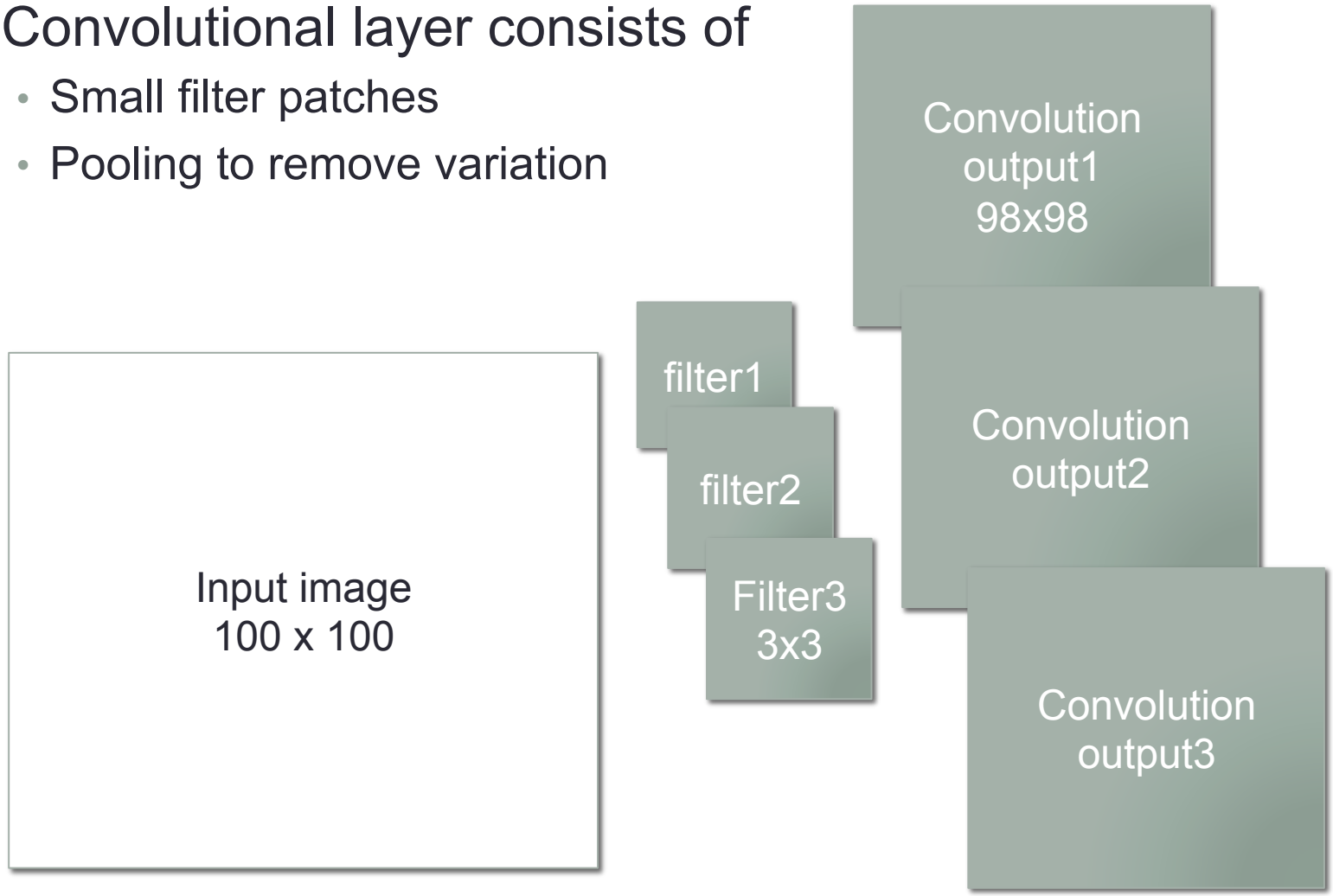
Pooling layers/Subsampling layers

- Combine different locations into one
- One possible method is to use a max
  - Interpretation: Yes, I found a cat somewhere

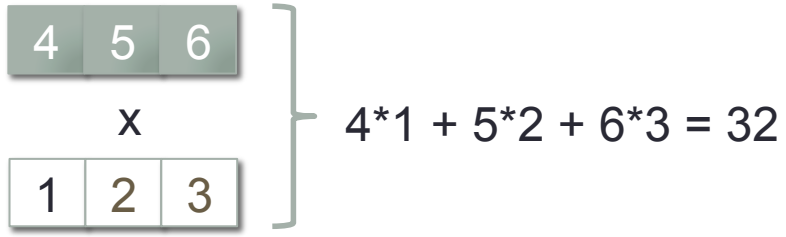


Convolutional filters

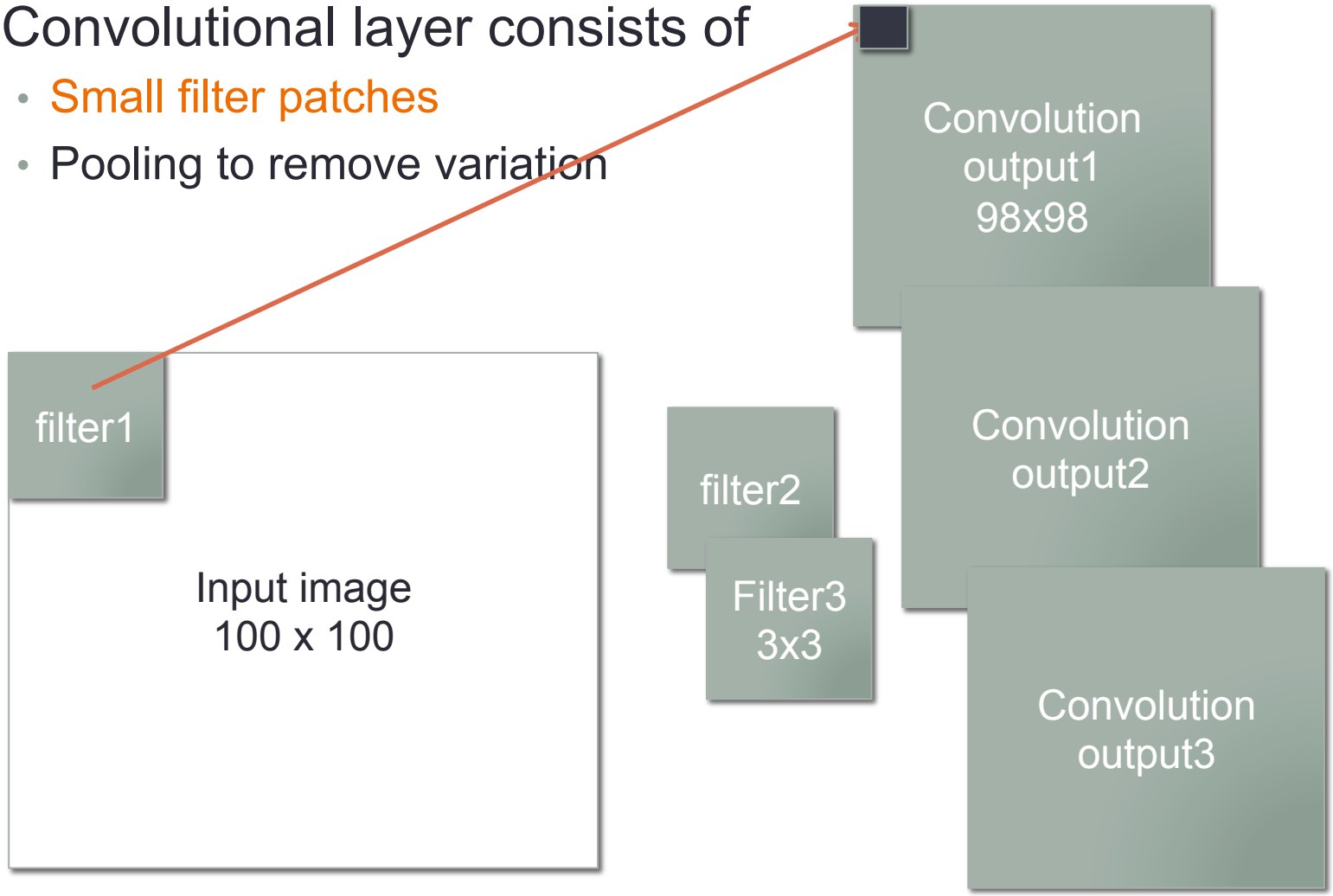
- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



Convolutional filters

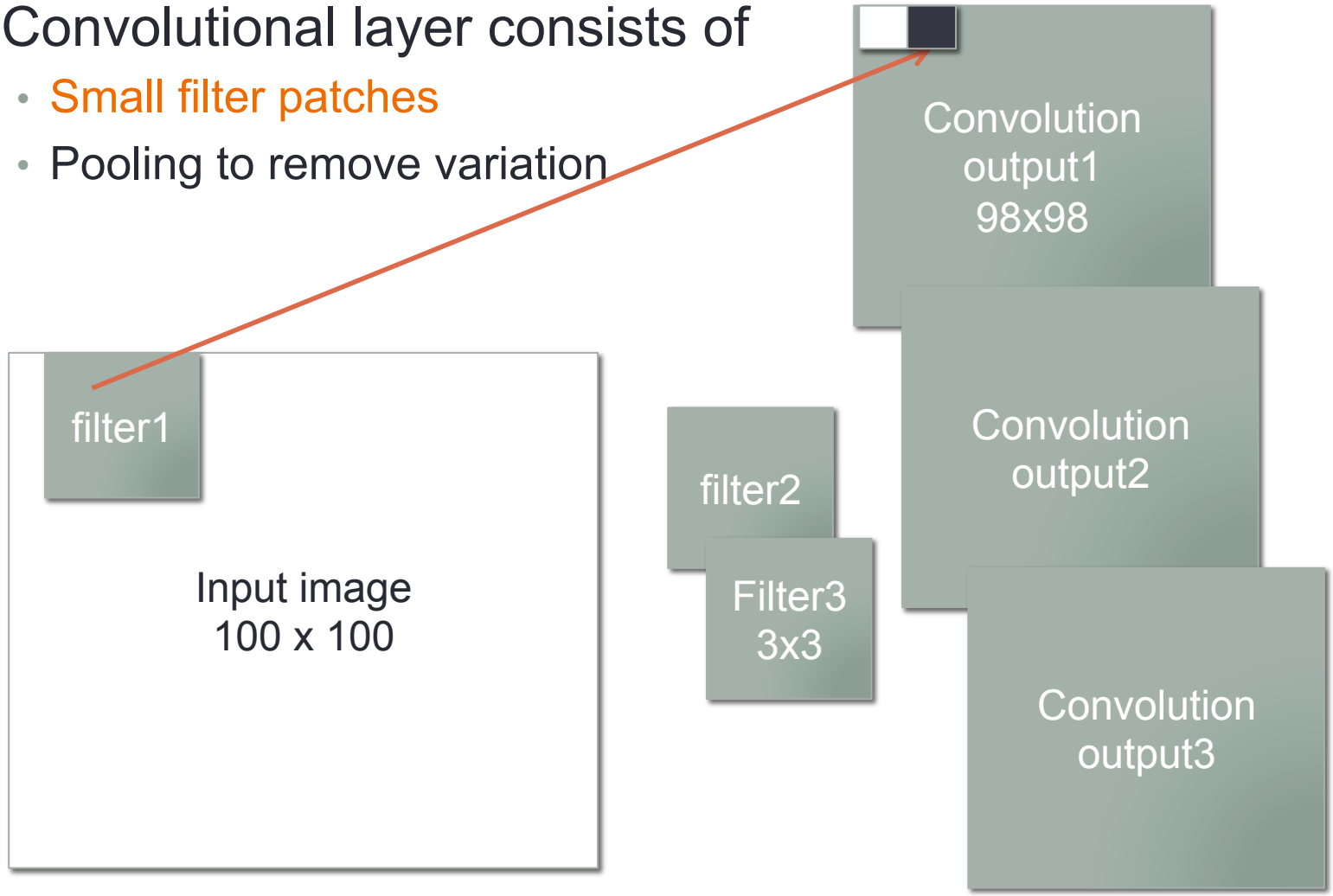


- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



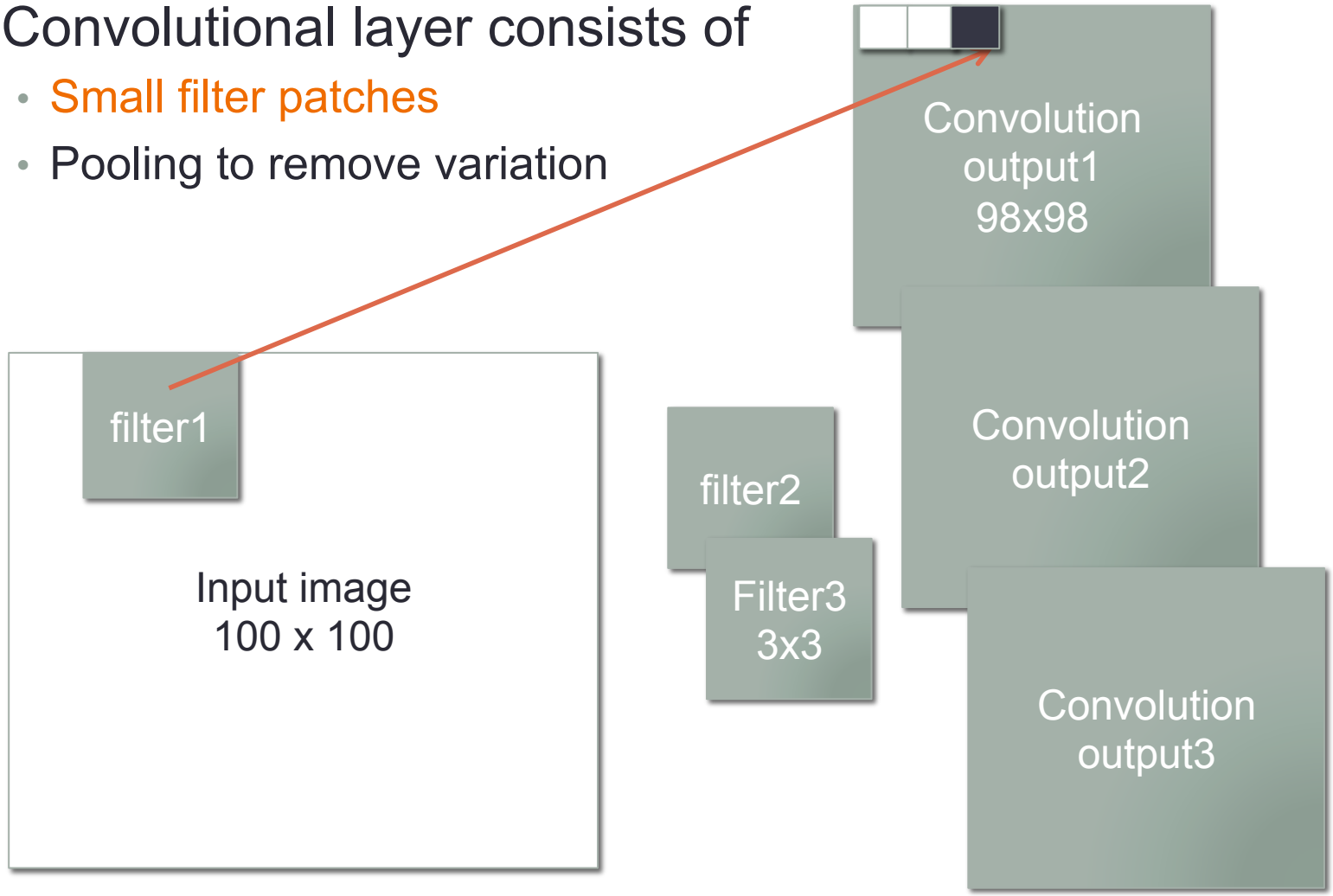
Convolutional filters

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



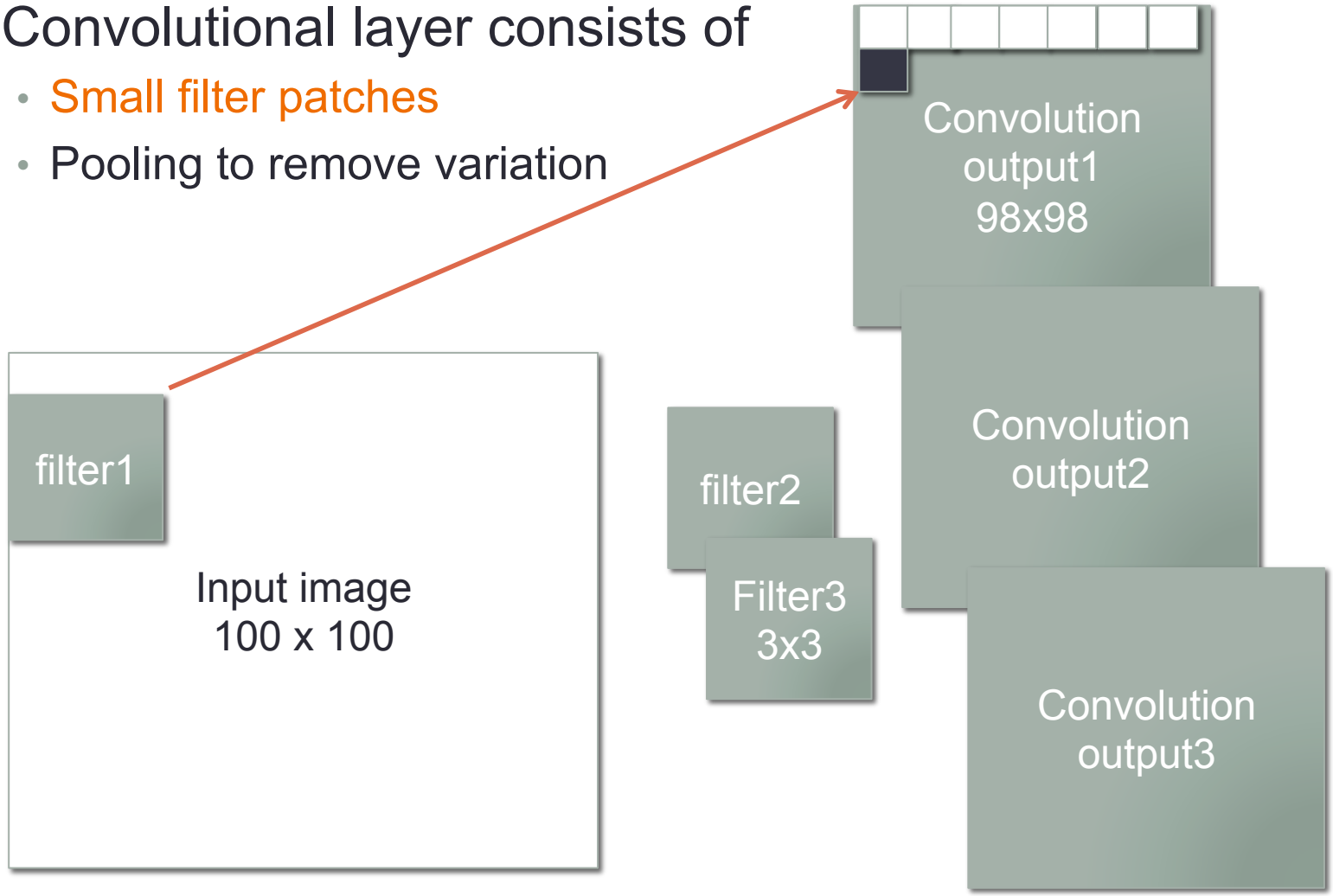
Convolutional filters

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



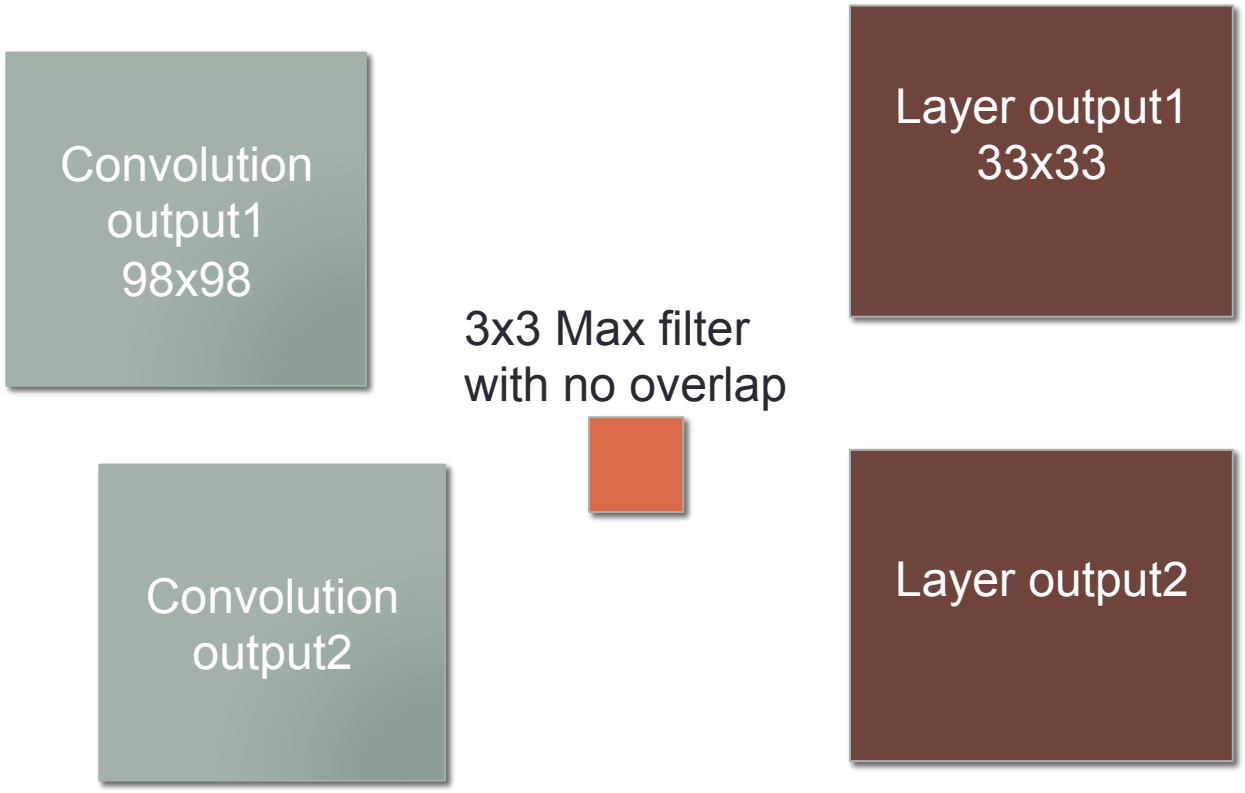
Convolutional filters

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



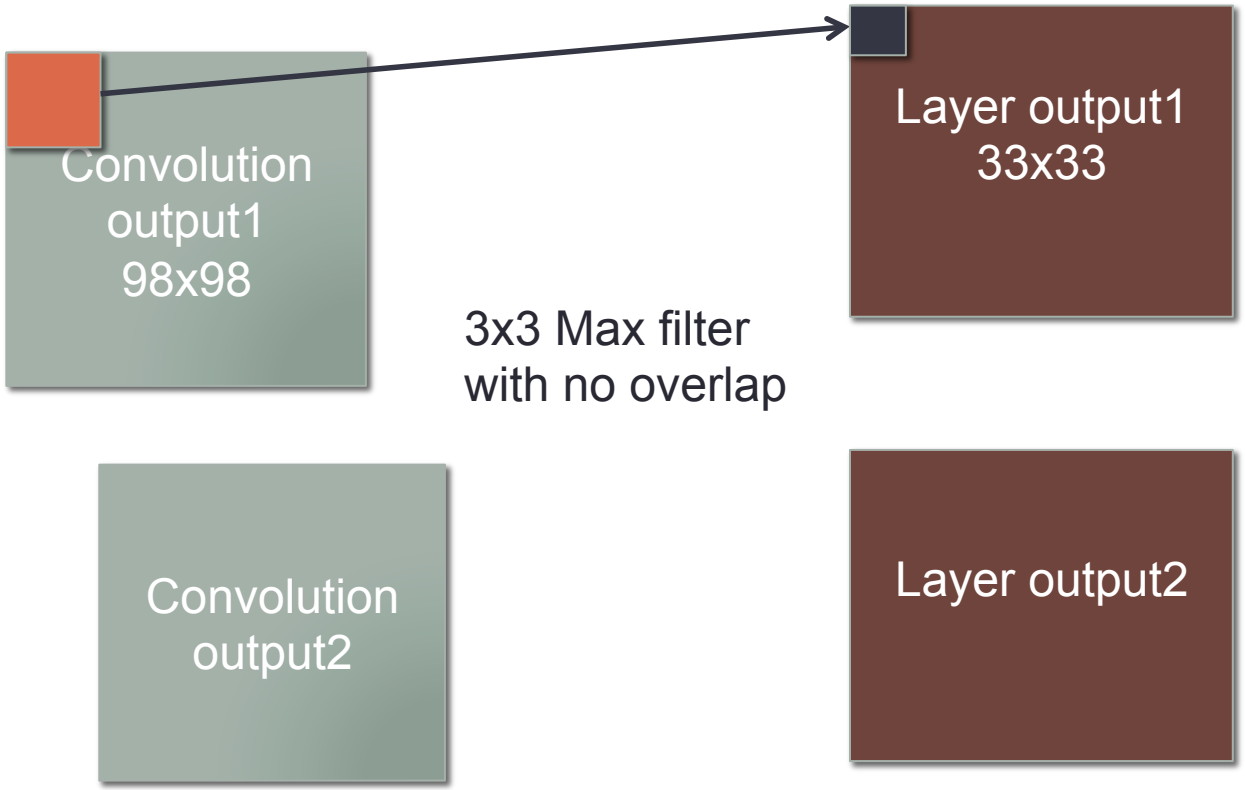
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



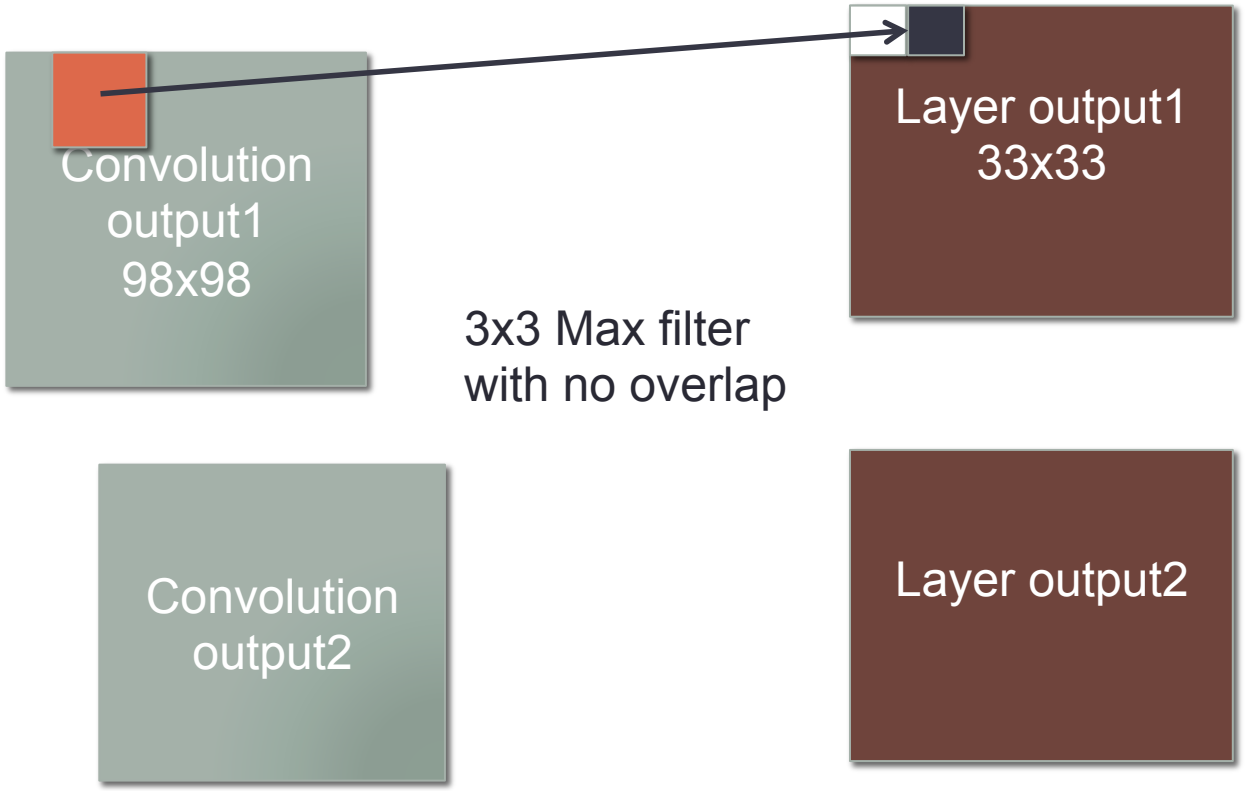
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



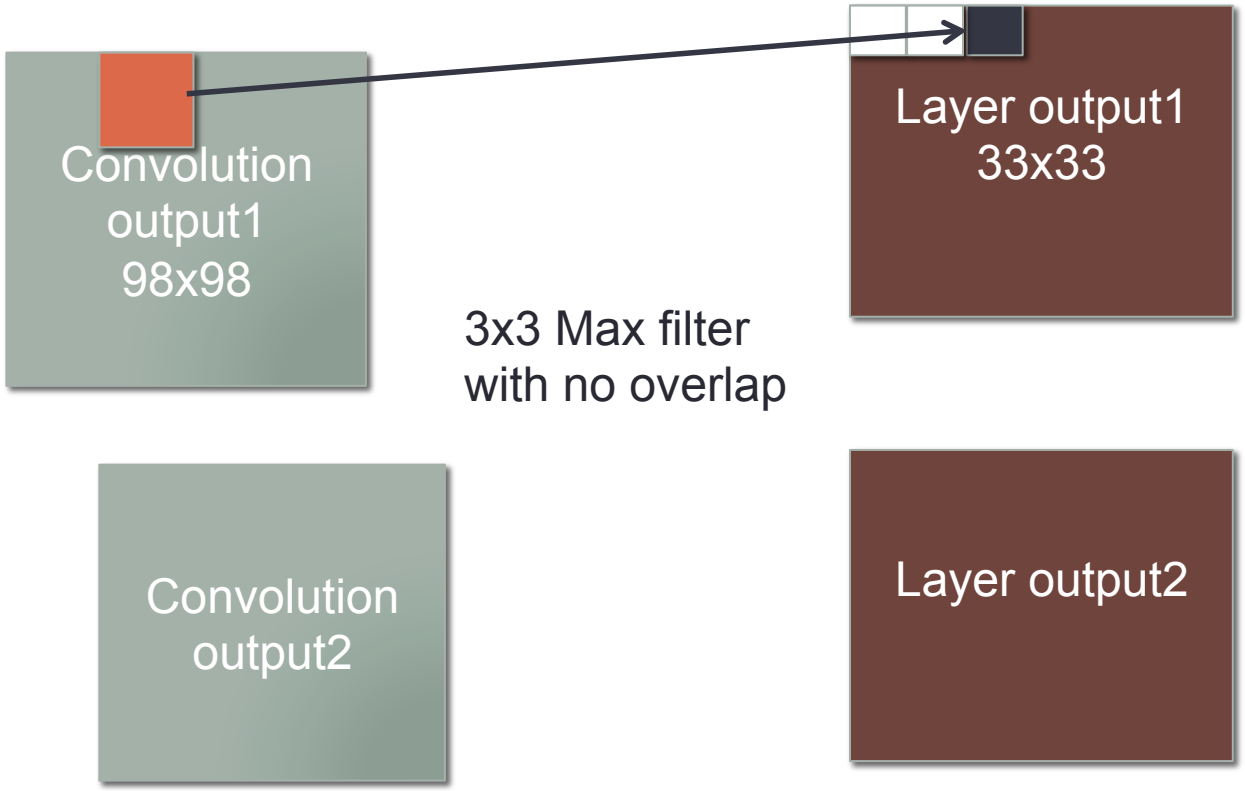
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



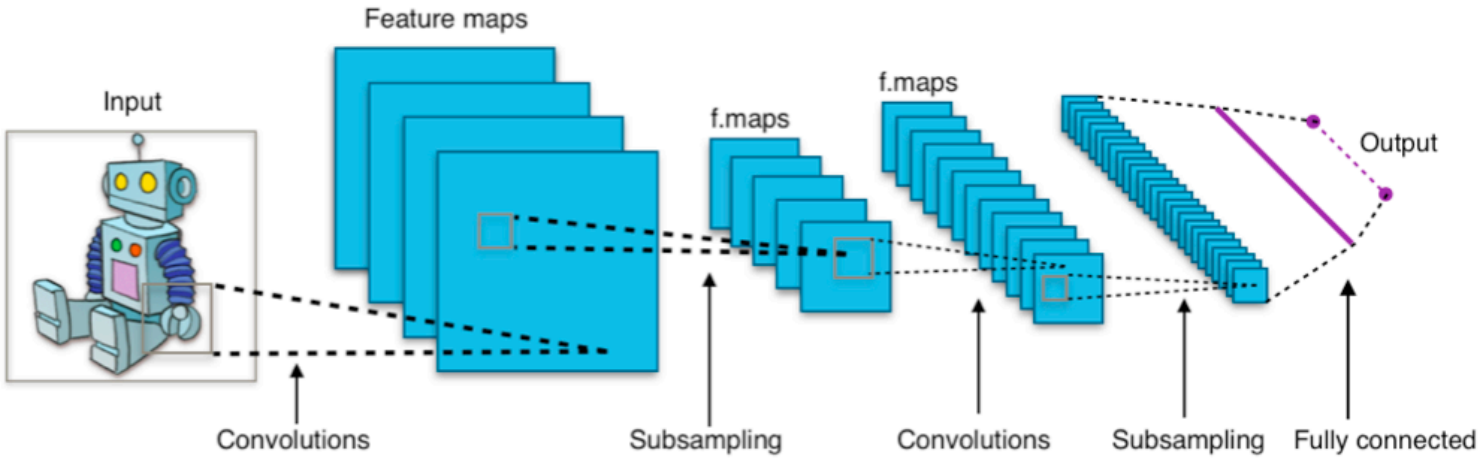
Pooling/subsampling

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation



CNN overview

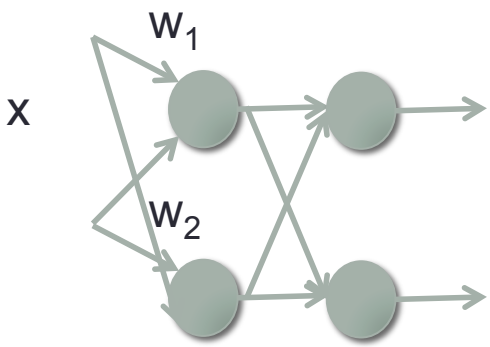
- Filter size, number of filters, filter shifts, and pooling rate are all parameters
- Usually followed by a fully connected network at the end
  - CNN is good at learning low level features
  - DNN combines the features into high level features and classify



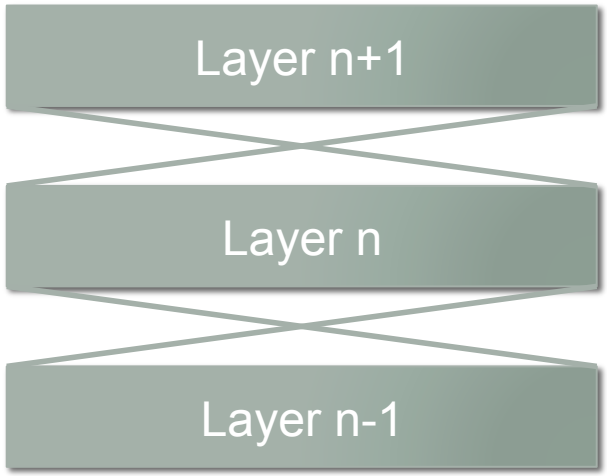
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

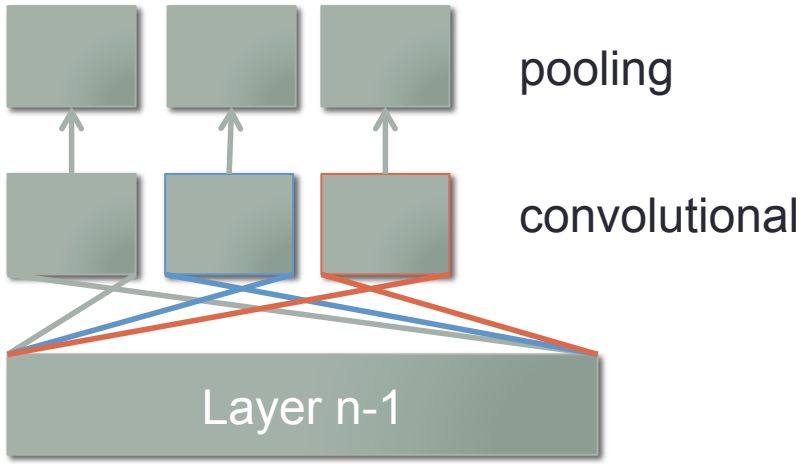
Parameter sharing in convolution neural networks

- $W^T x$



- Cats at different location might need two neurons for different locations in fully connect NNs.
- CNN shares the parameters in 1 filter
- The network is no longer fully connected





Pooling/subsampling

- Max filter -> Maxout
  - Backward pass?
    - Gradient pass through the maximum location, 0 otherwise

Norms (p-norm or Lp-norm)

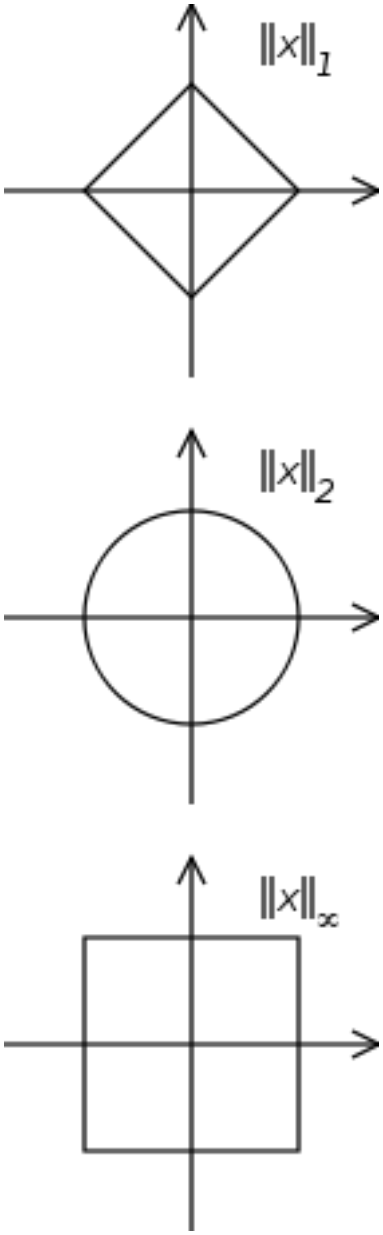
- For any real number p > 1

$$||\mathbf{x}||_p = (|x_1|^p + |x_2|^p + \ldots + |x_n|^p)^{\frac{1}{p}}$$

- For p = ∞

$$\|x\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$$

- We'll see more of p-norms when we get to neural networks



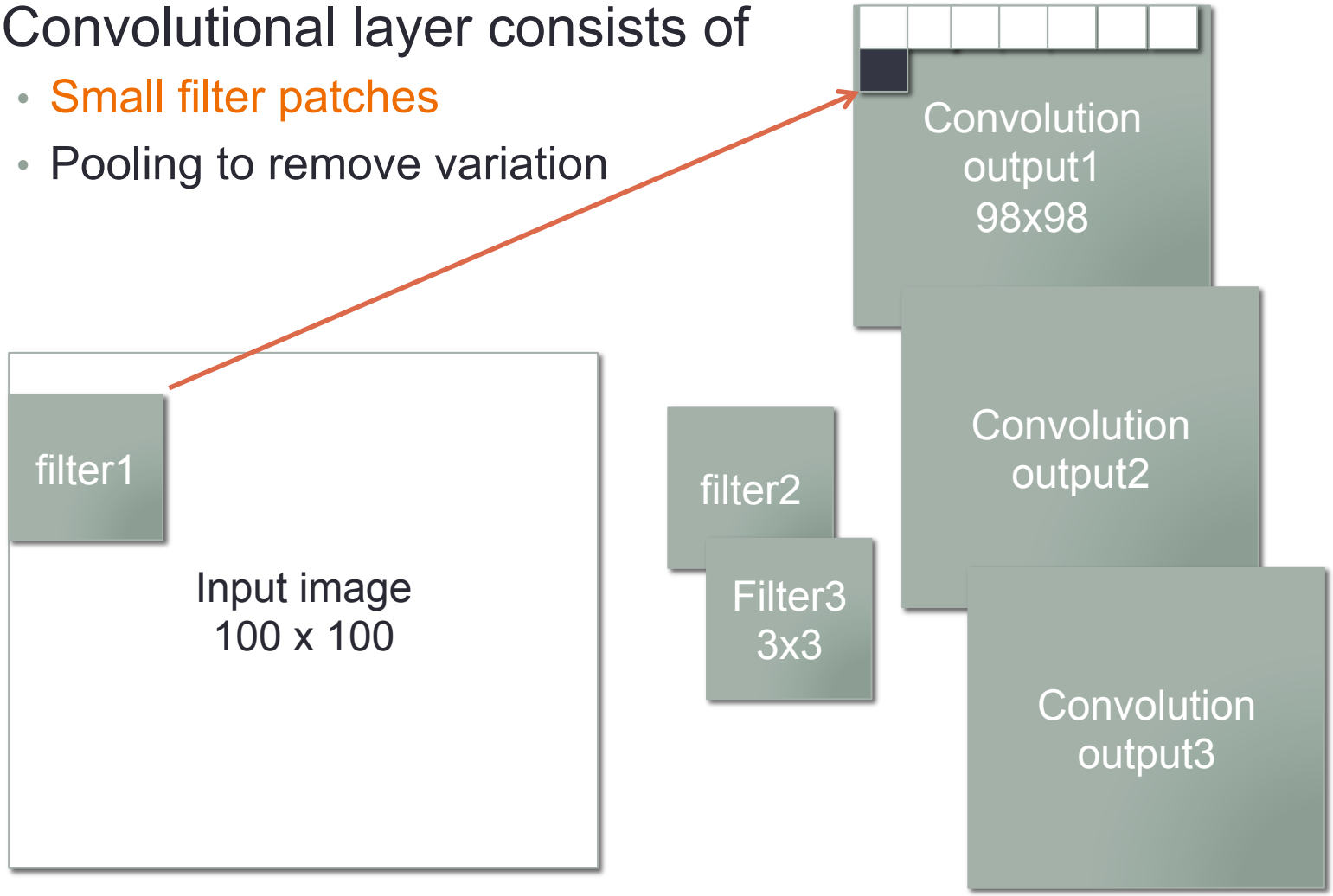
https://en.wikipedia.org/wiki/Lp_space

Pooling/subsampling

- Max filter -> Maxout
  - Backward pass?
    - Gradient pass through the maximum location, 0 otherwise
- P-norm filter
- Fully connected layer – (1x1 convolutions)
- Recently, people care less about the meaning of pooling as way to introduce a shift invariance, but more as a dimension reduction (since conv layers usually has a higher dimension than the input)

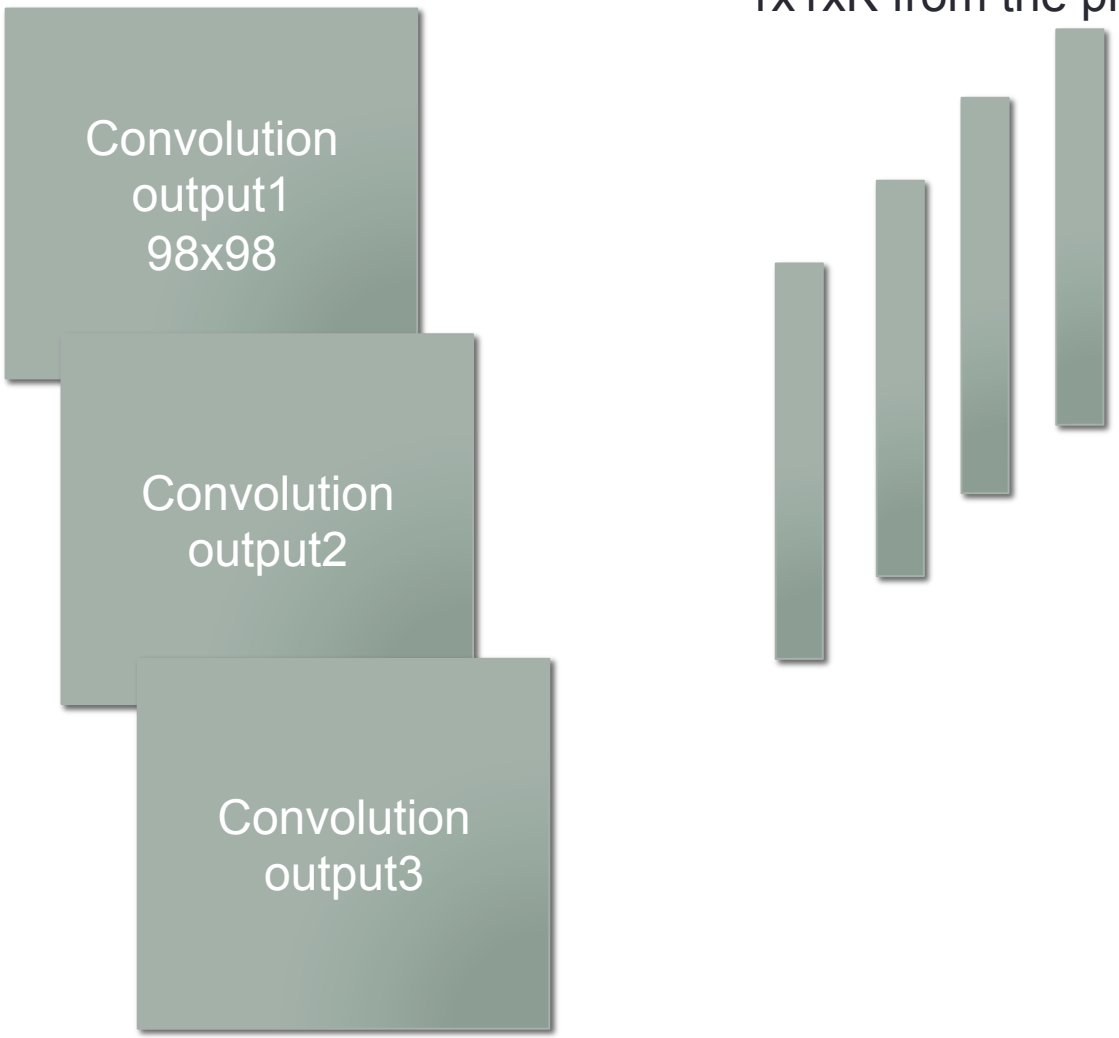
1x1 Convolutions

- Convolutional layer consists of
  - Small filter patches
  - Pooling to remove variation

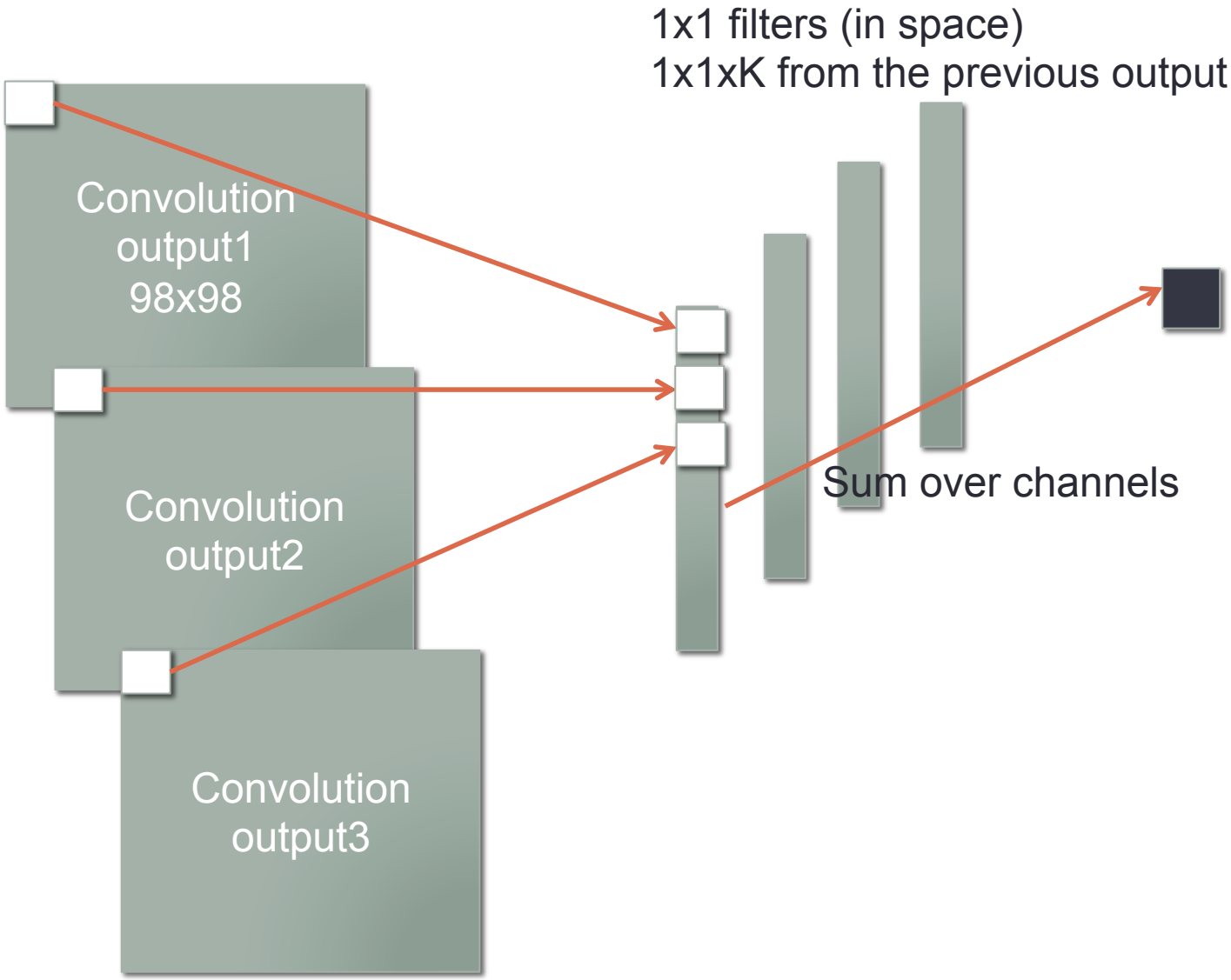


1x1 Convolutions

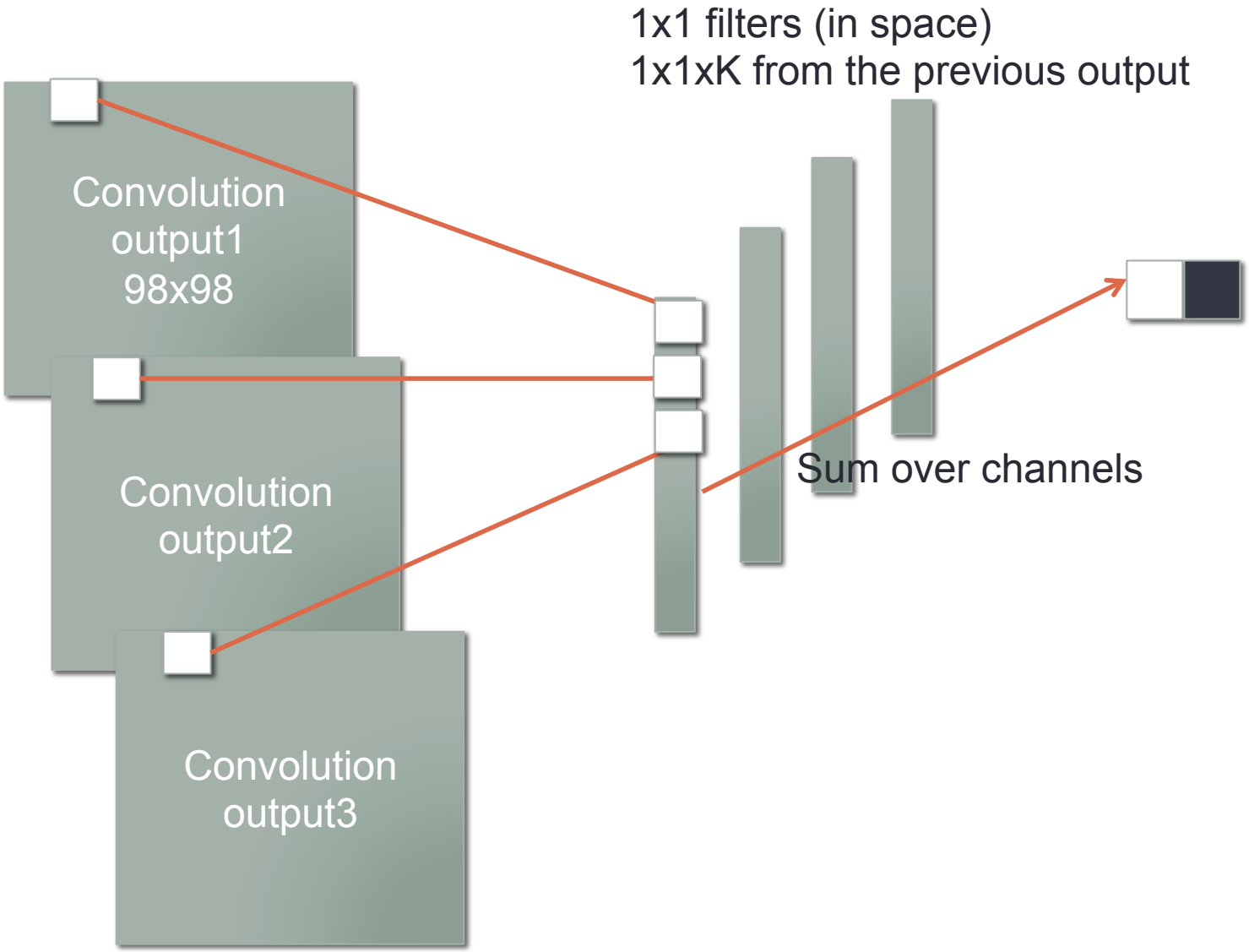
1x1 filters (in space)
1x1xK from the previous output



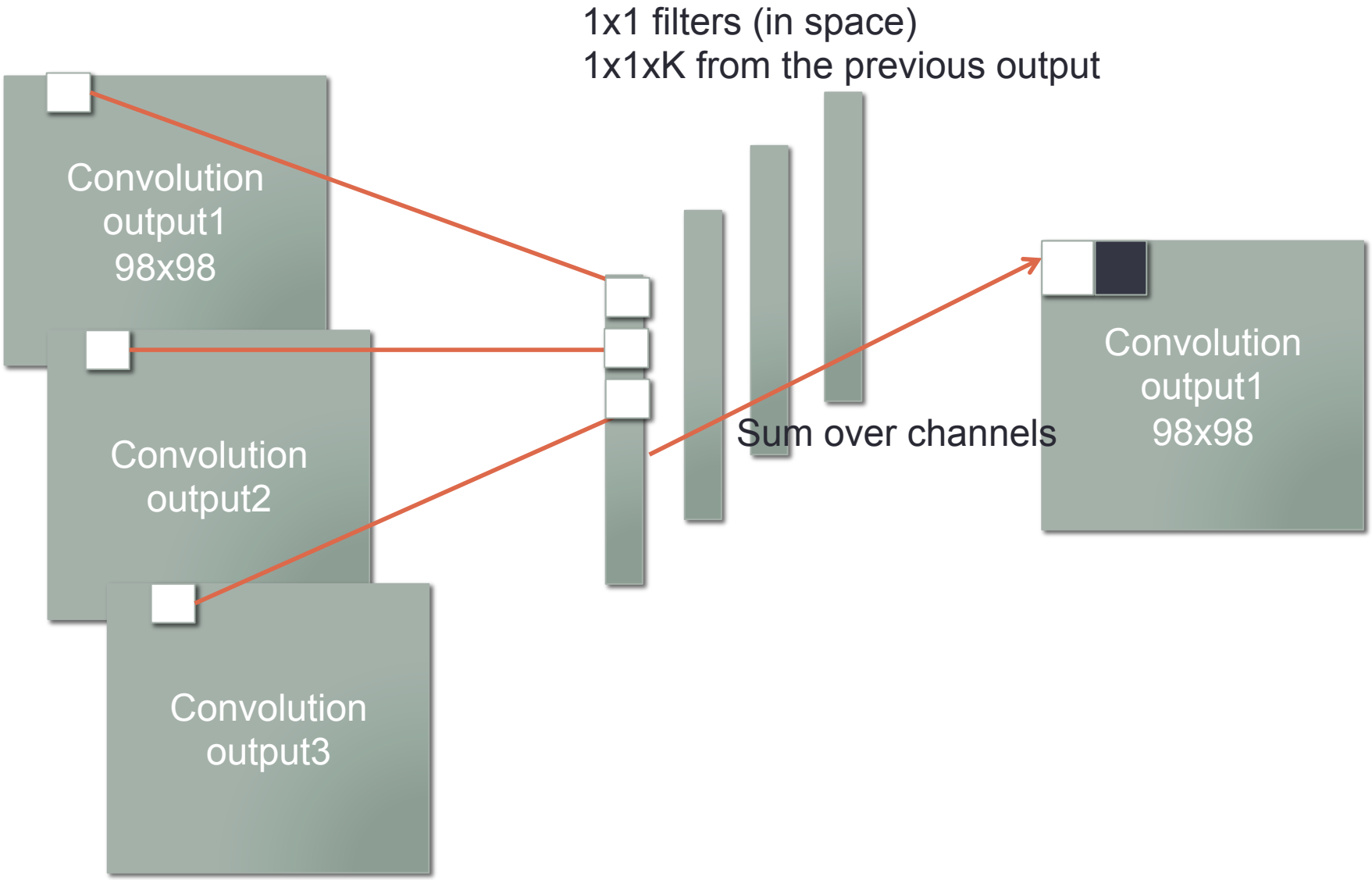
1x1 Convolutions



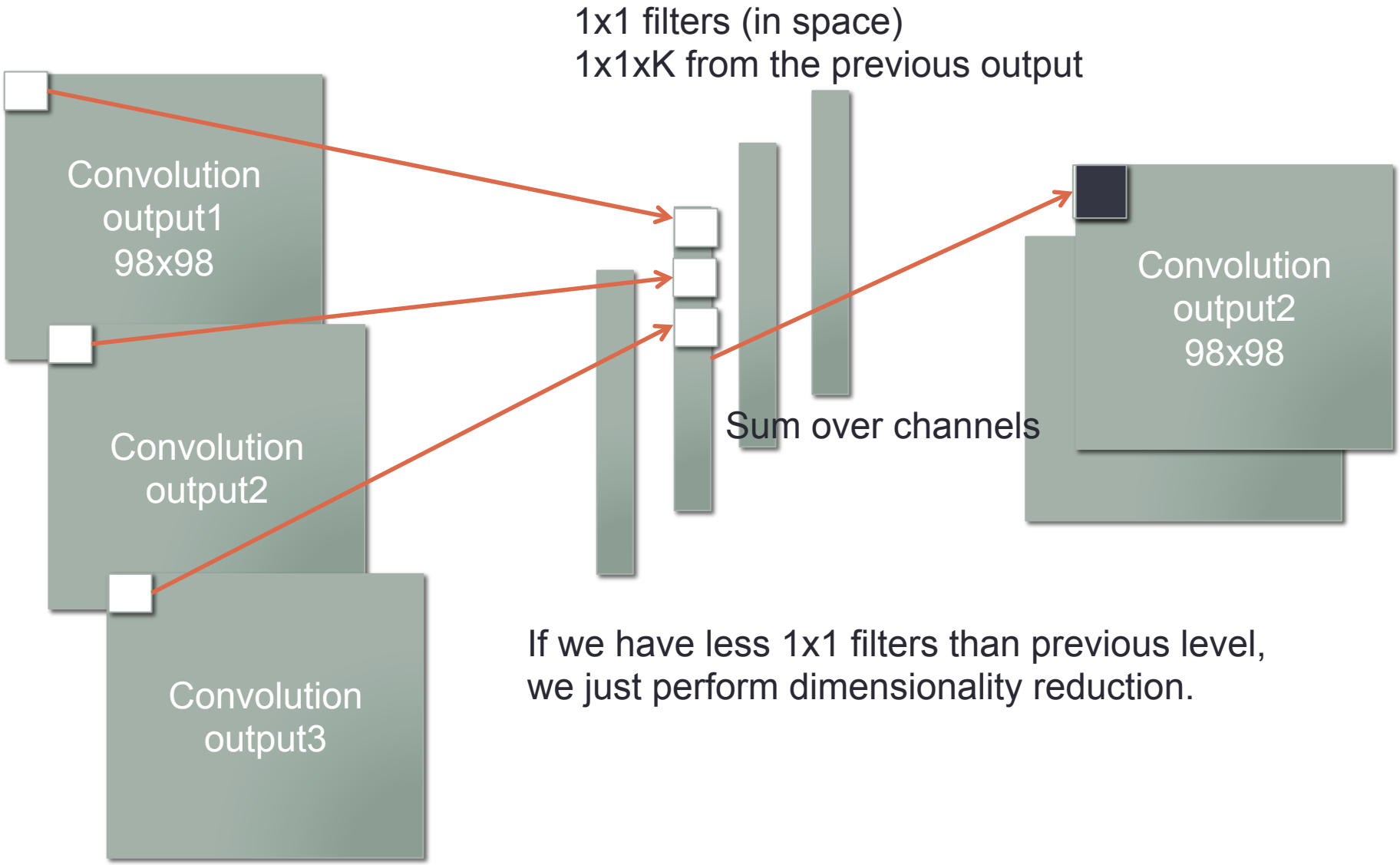
1x1 Convolutions



1x1 Convolutions



1x1 Convolutions



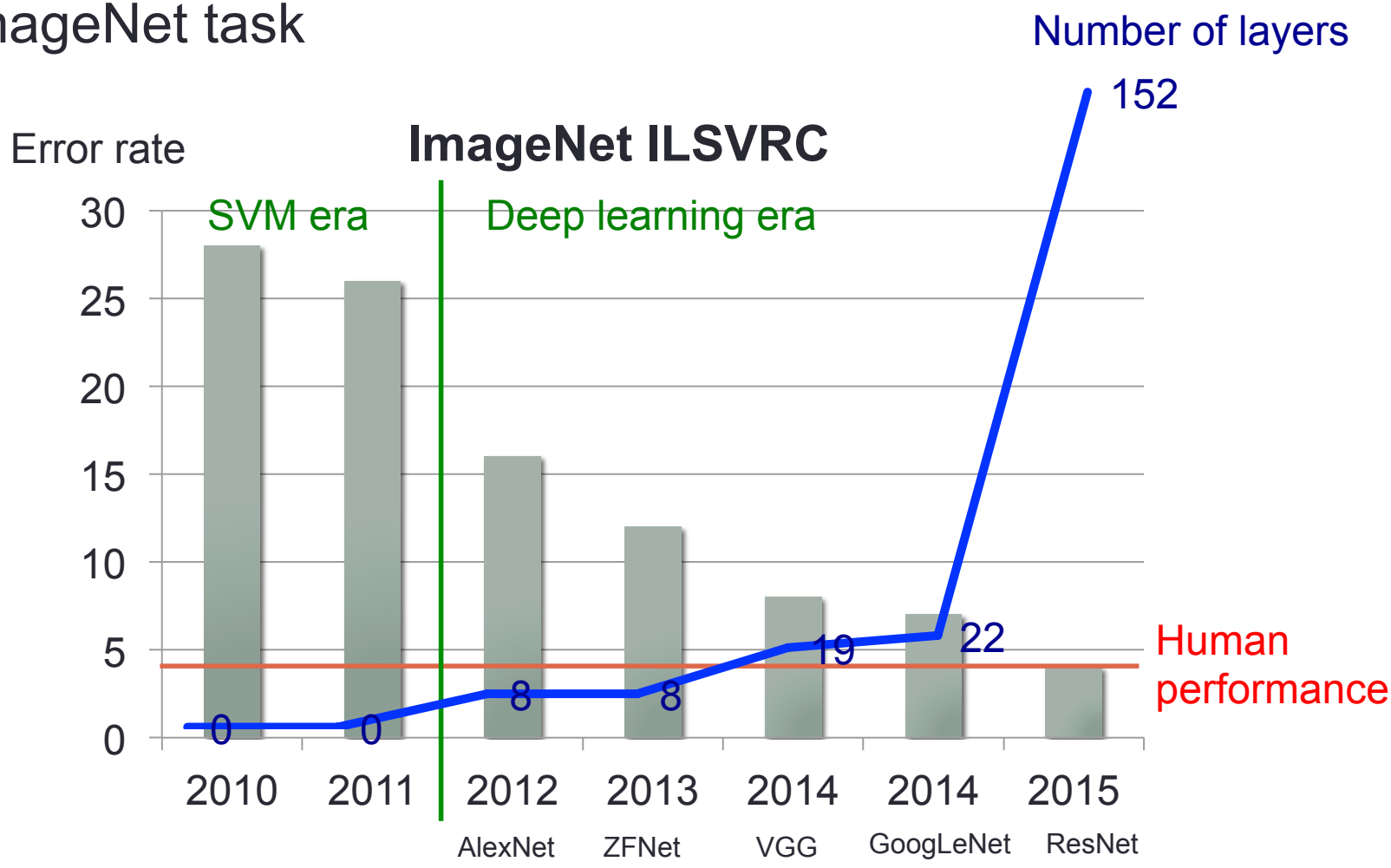
If we have less 1x1 filters than previous level, we just perform dimensionality reduction.

Common schemes

- INPUT -> [CONV -> RELU -> POOL]*N -> [FC -> RELU]*M -> FC
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*N -> [FC -> RELU]*M -> FC
- If you working with images, just use a winning architecture.

Wider and deeper networks

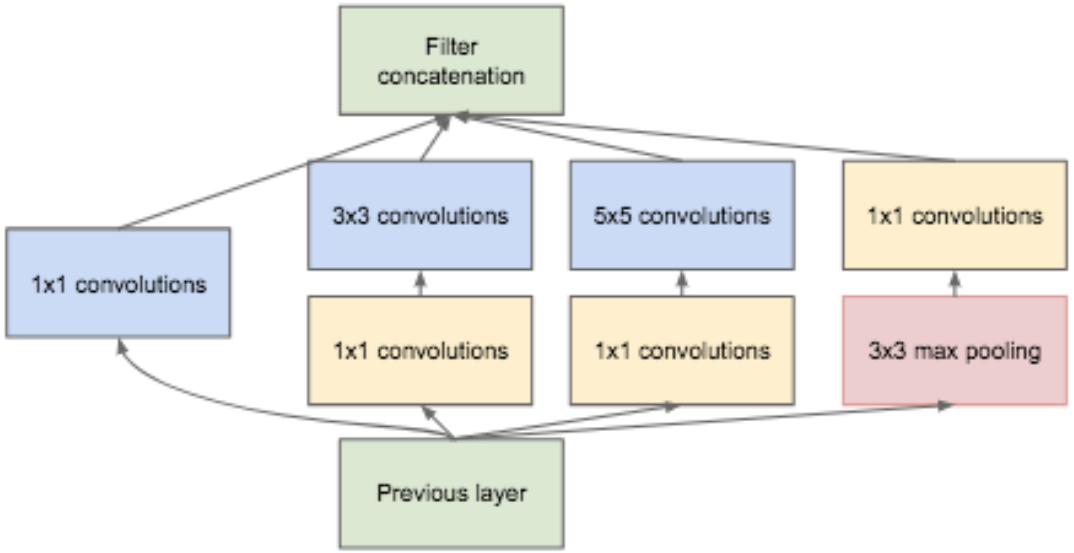
- ImageNet task



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

ImageNET prize winner

- AlexNet – first deep model based on LeNET (simple CNN from 1990s)
- ZFNet – AlexNET just deeper, and better tuned hyperparameters
- GoogLeNet (Inception Model)



ImageNET prize winner 2

- VGGNet - just a bigger network. Notable mention because model available
- ResNet – Next class

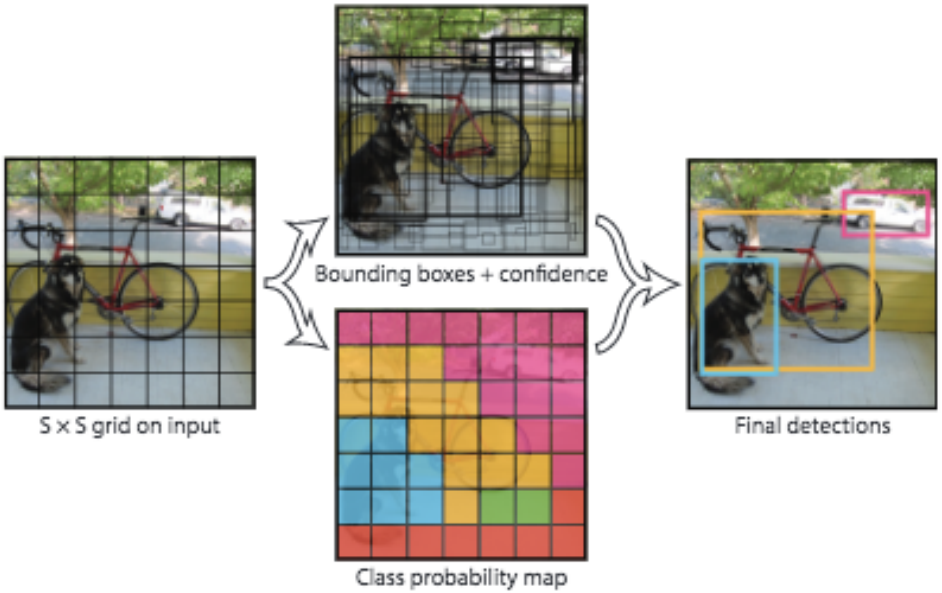
Other models

- YOLO: Frame object classification to regression
  - Regression: coordinates + class probabilities
  - Traditional object detector segments the original images into patches and scan the patches – each different patch is scanned many times

https://pjreddie.com/media/files/papers/yolo_1.pdf

YOLO

- YOLO: Frame object classification to regression
  - Regression: coordinates + class probabilities
  - You Only Look Once: output possible bounding boxes and class
  - A post-processing step is used to merge the bounding boxes
  - Fast model for real time object detectors
  - Model is just VGG



https://pjreddie.com/media/files/papers/yolo_1.pdf

Faster R-CN

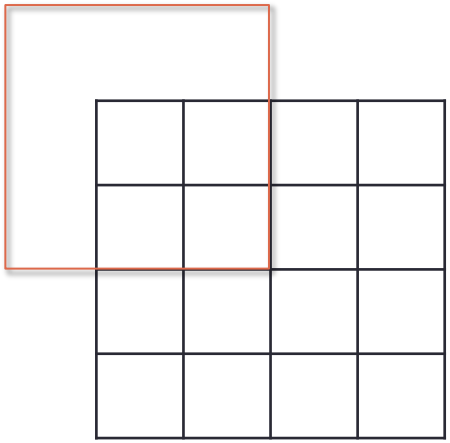
- Another model used for object detection
- Current state of the art
- Details – next class

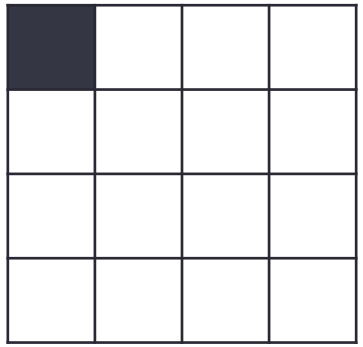
De-convolution layers

- Yet another abused of notation made by vision folks
  - Conventional meaning:
    - A method to reverse an effect of a filter
    - Blurred image -> de-convolution -> Good image
  - Neural network meaning:
    - Something that reverse the order of convolution computation
    - Backward pass of a convolutional layer
- Used for upsampling

Convolution

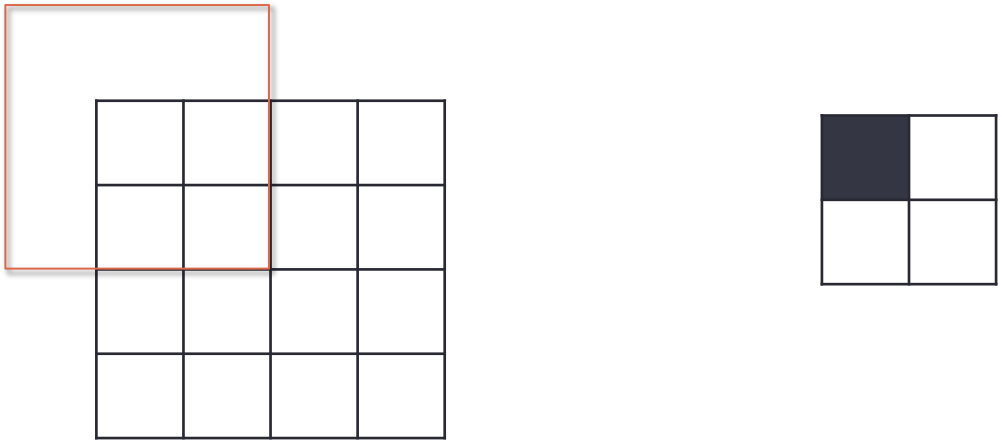
- 3x3 filter pad, stride 1, pad 1





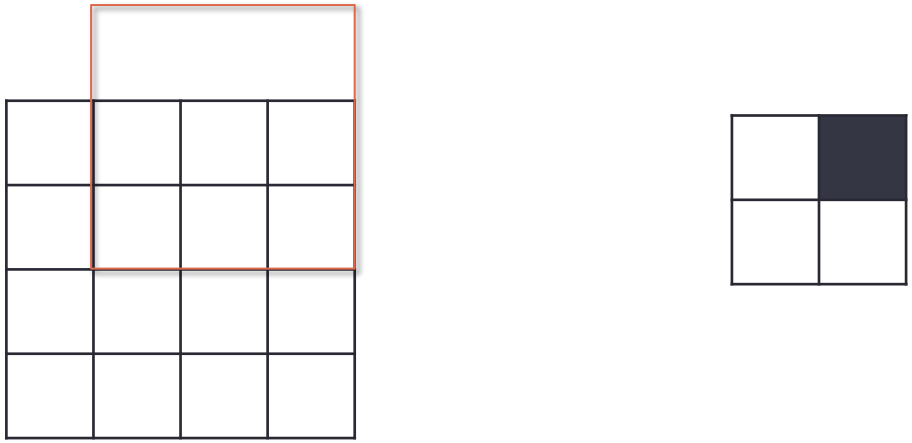
Convolution

- 3x3 filter pad, stride 2, pad 1



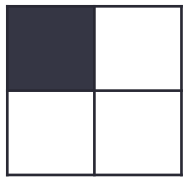
Convolution

- 3x3 filter pad, stride 2, pad 1

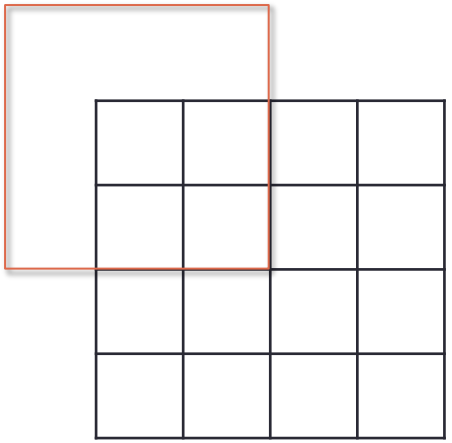


De-convolution

- 3x3 de-convolution filter, stride 2, pad 1



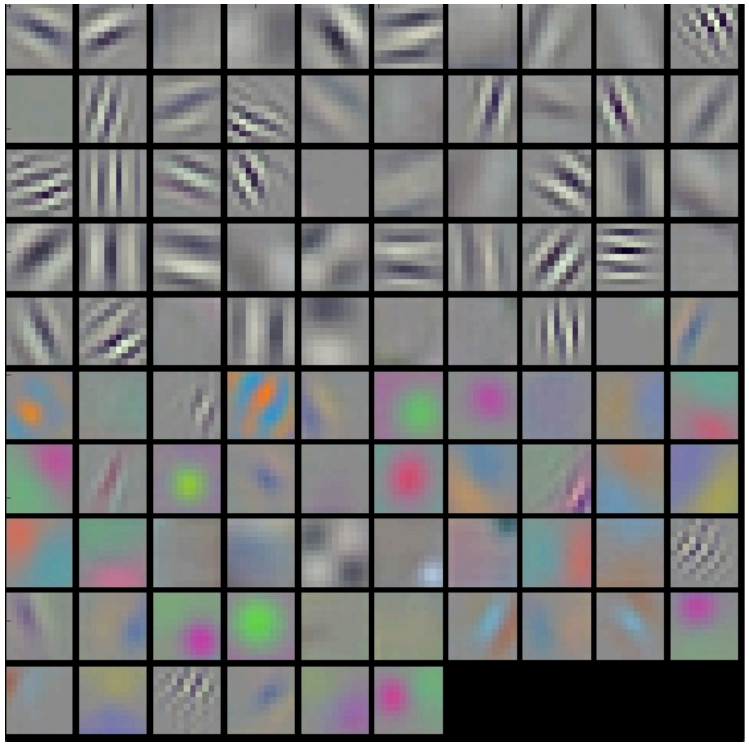
Input gives
Weight for filter



Rubber stamp

Visualizing convolutional layers

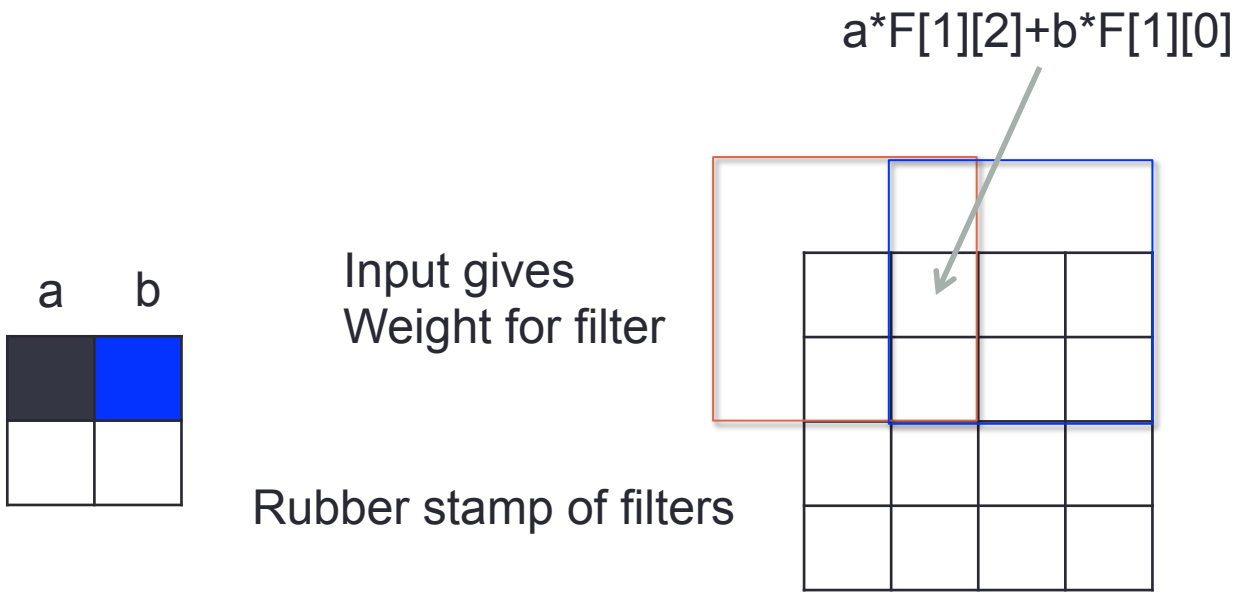
- Just like PCA, we can visualize the weights of a transform
- “Matched filters”



http://cs231n.github.io/understanding-cnn/

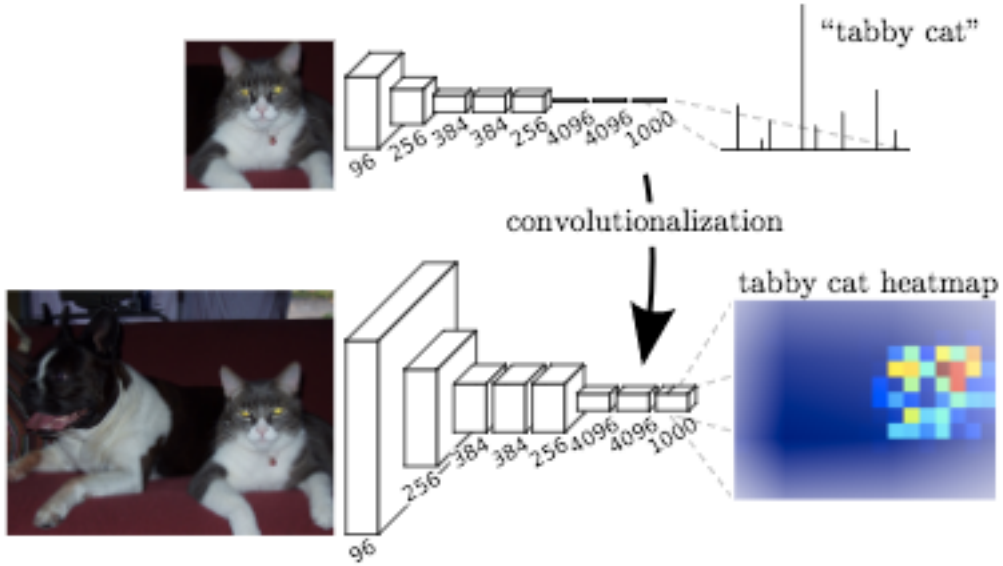
De-convolution

- 3x3 deconvolution filter, stride 2, pad 1



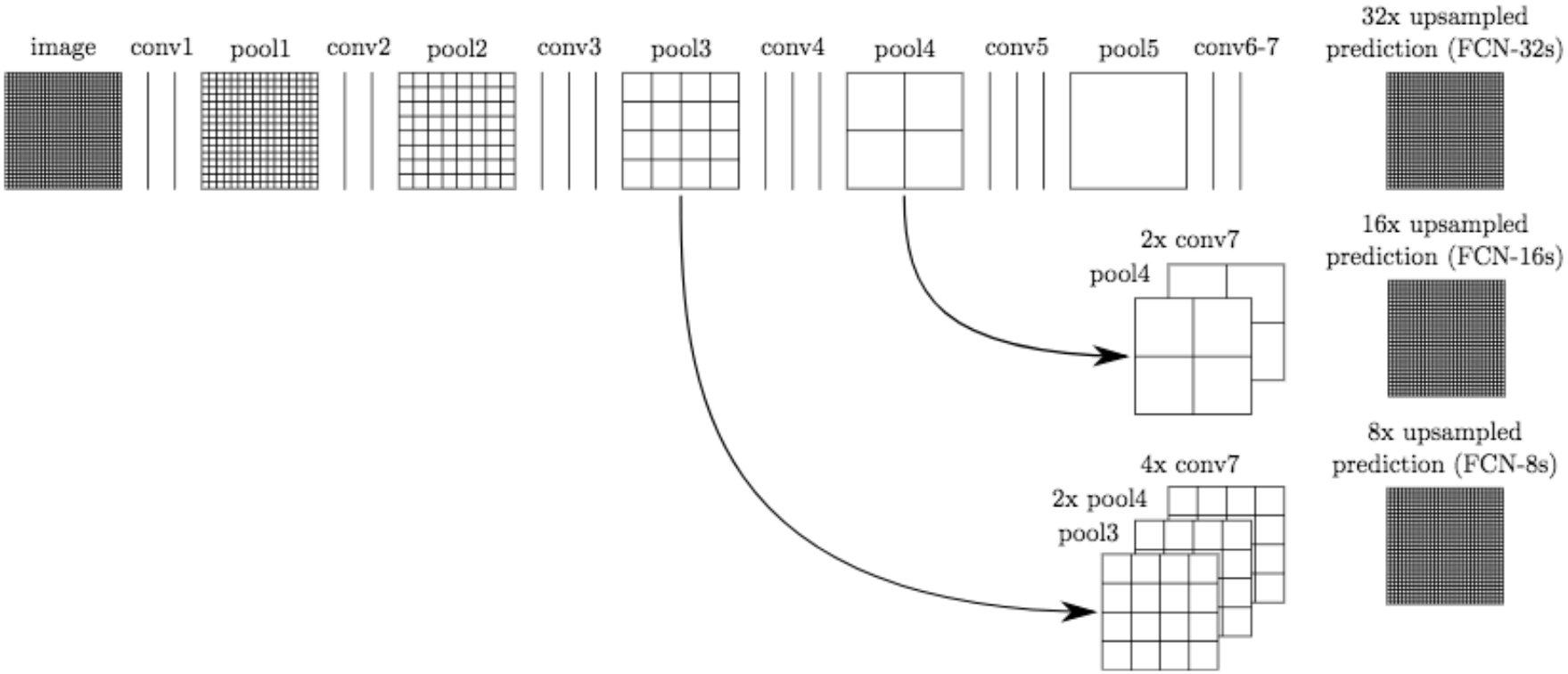
Other names because this name sucks (for me)
- Convolution transpose, upconvolution, backward strided convolution

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation



https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

De-convolution for segmentation

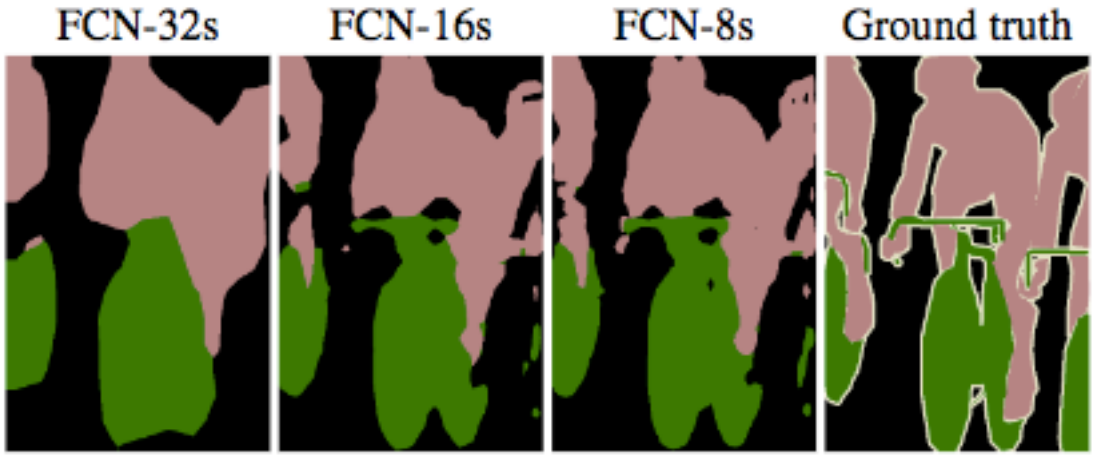


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

CNN

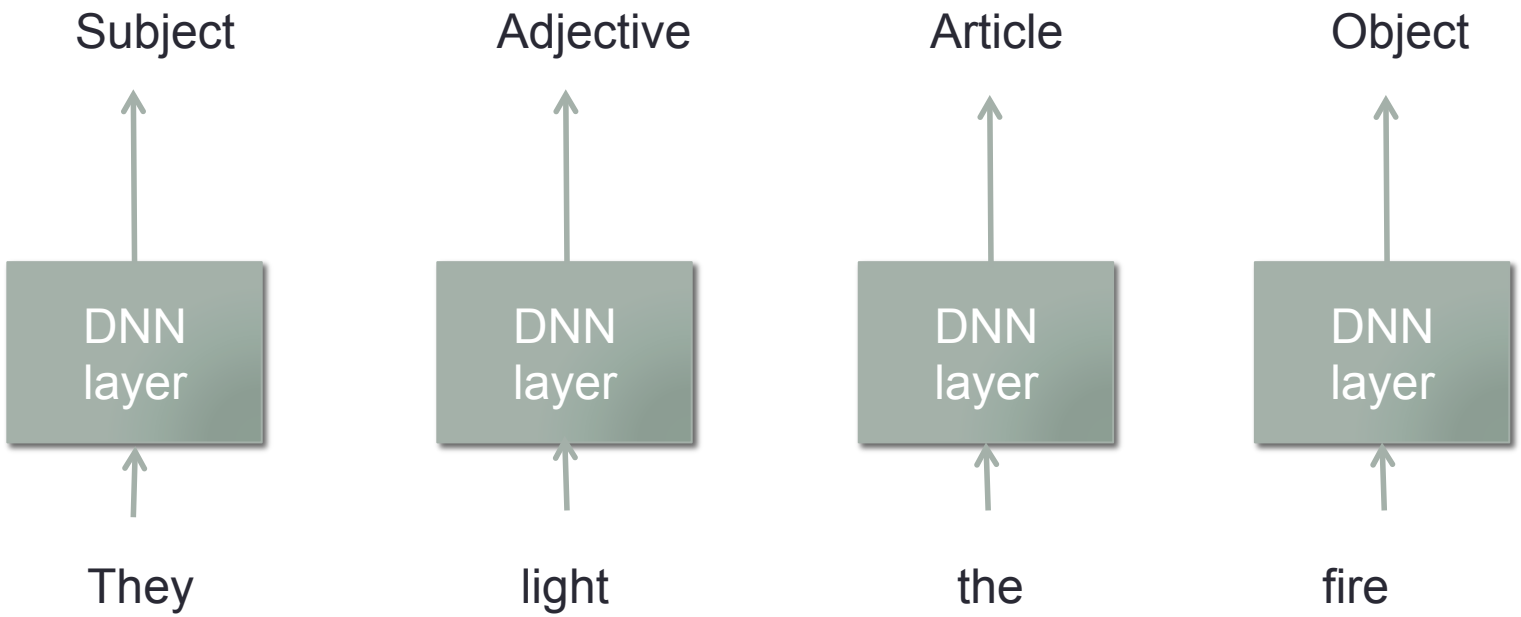
- Convolutional layer
- Subsampling

- Sharing of parameters in space

- Sharing of parameters in time?

Recurrent neural network (RNN)

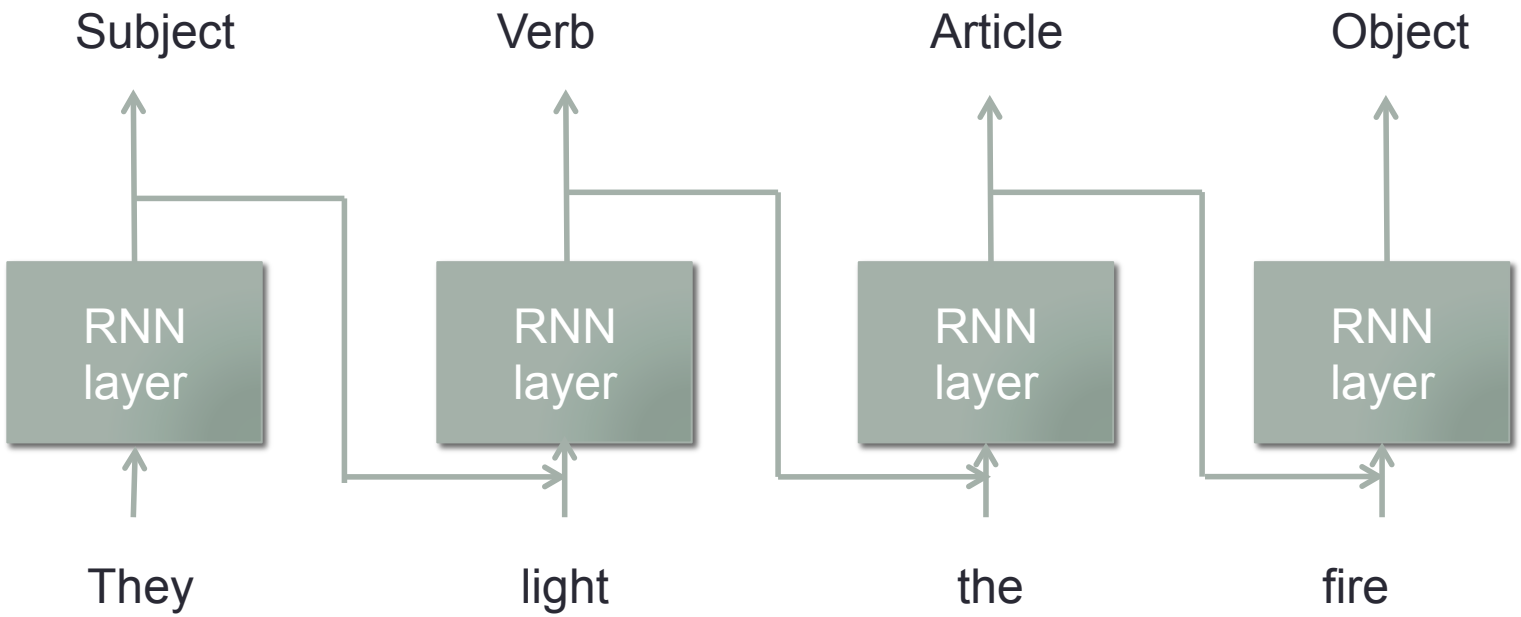
- DNN framework



Problem: need a way to remember the past

Recurrent neural network (RNN)

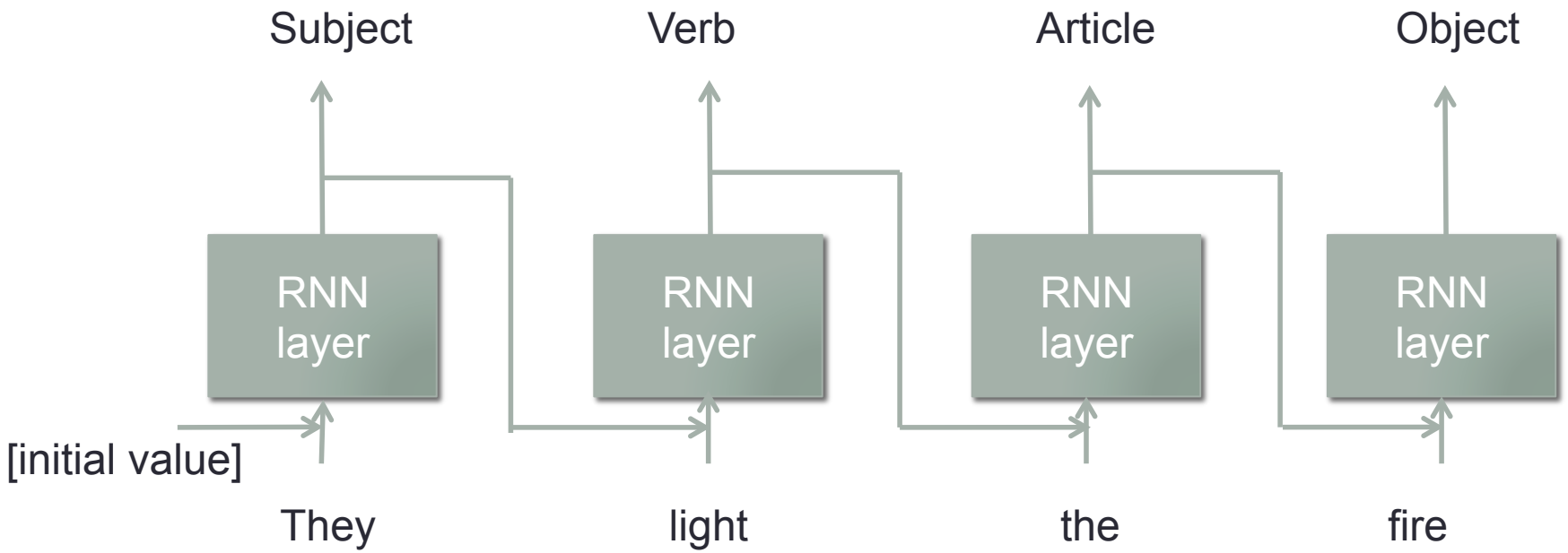
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

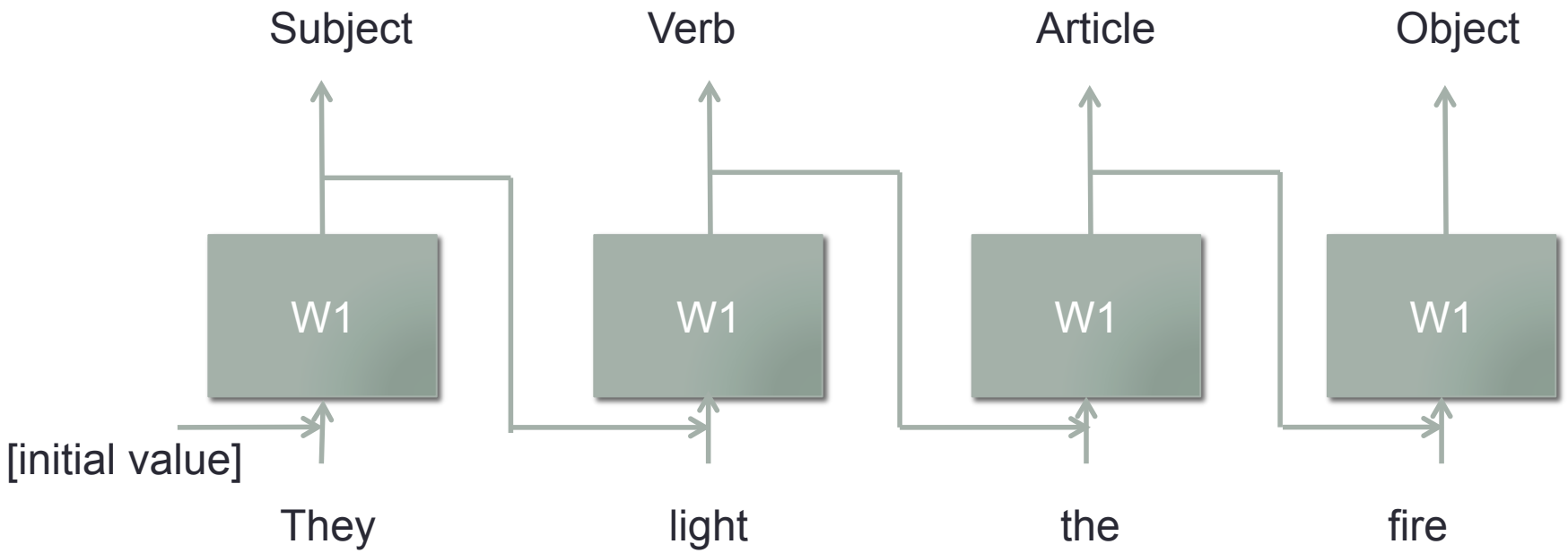
- RNN framework



New input feature = [original input feature, output of the layer at previous time step]

Training a recurrent neural network

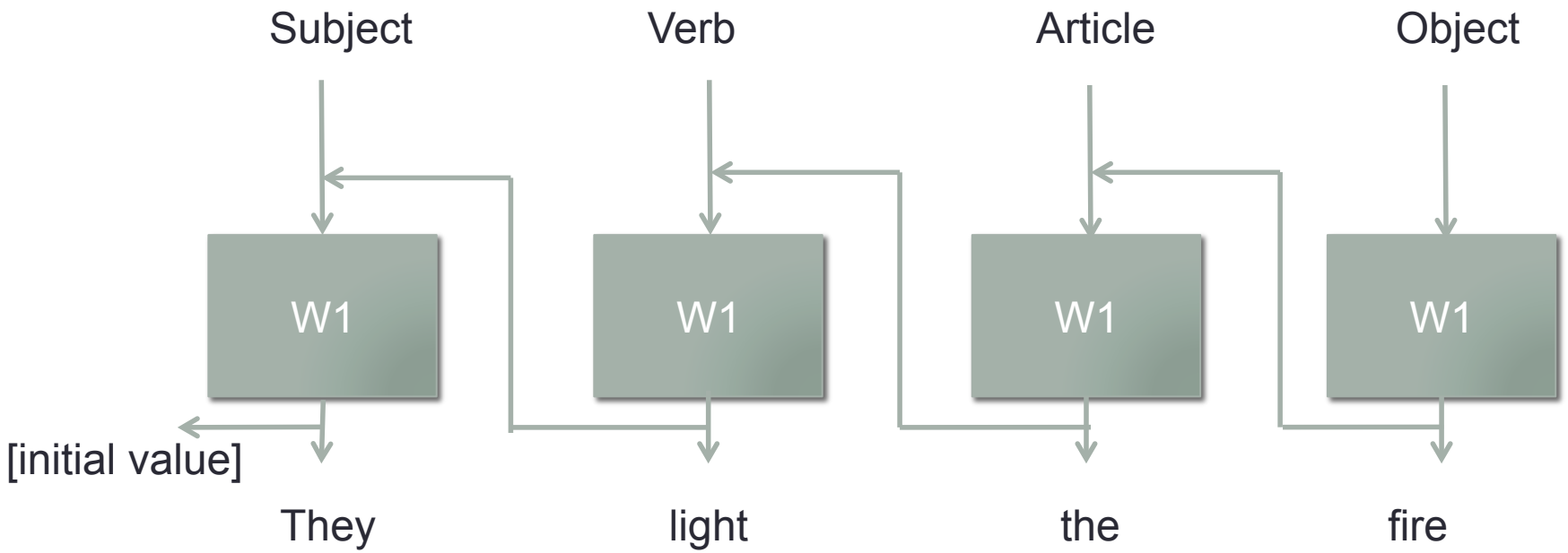
- Computation graph



Parameter sharing

Training a recurrent neural network

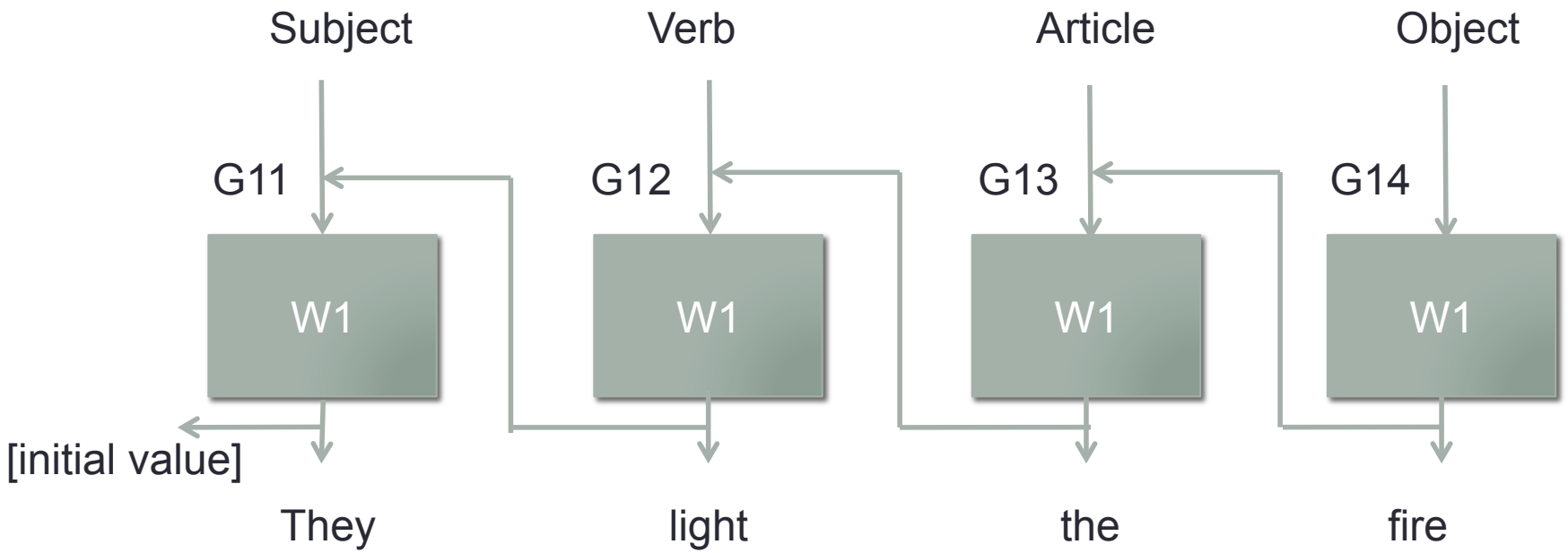
- Backward Computation graph



Backpropagation through time (BPTT)

BPTT

- Backward Computation graph

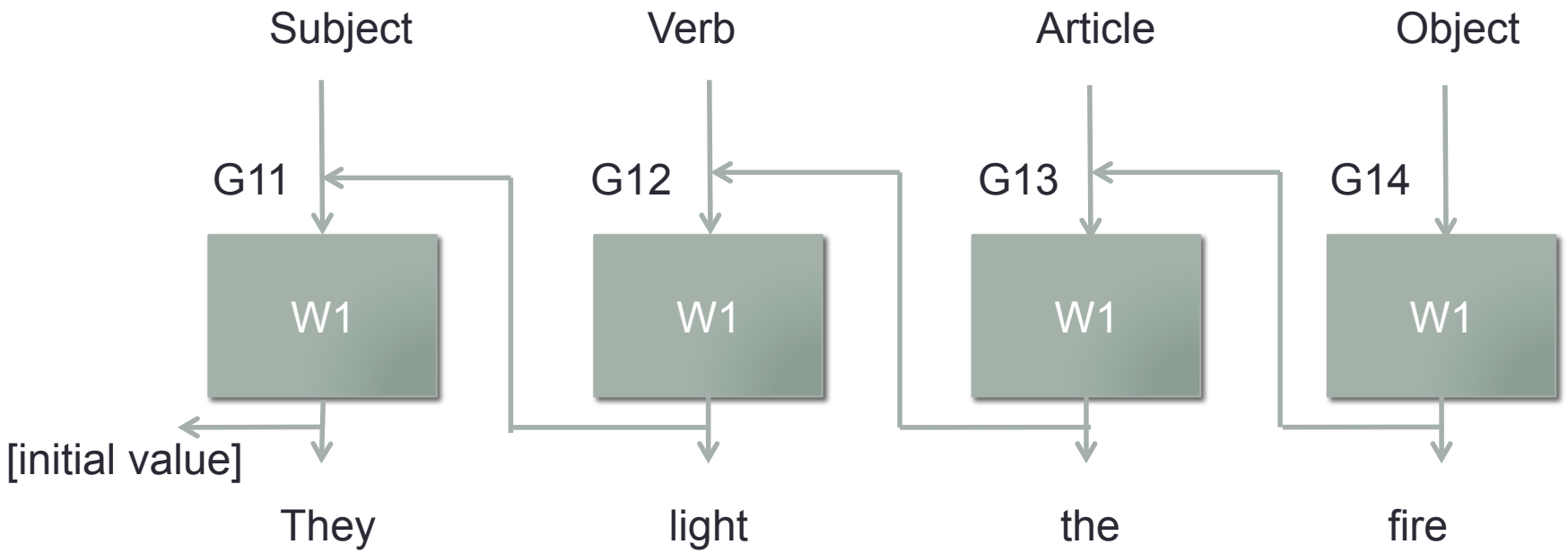


W1 <- W1 + G11 + G12 + G13 + G14

Cannot deal with infinitely long recurrent
Gradient explosion, vanishing

Truncated BPTT

- Backward Computation graph

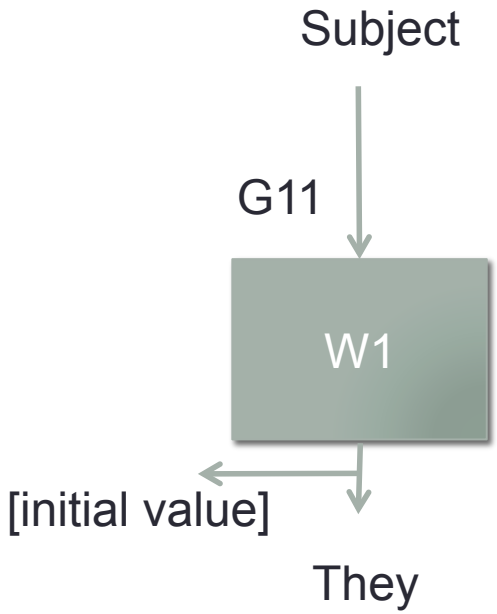


W1 <- W1 + G13 + G14

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

- Backward Computation graph

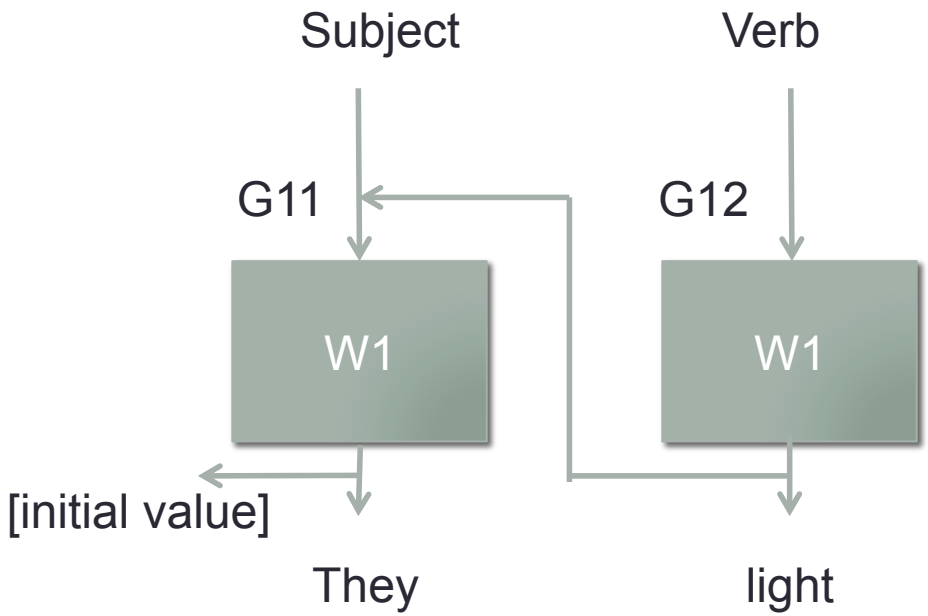


W1 <- W1 + G11

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

- Backward Computation graph

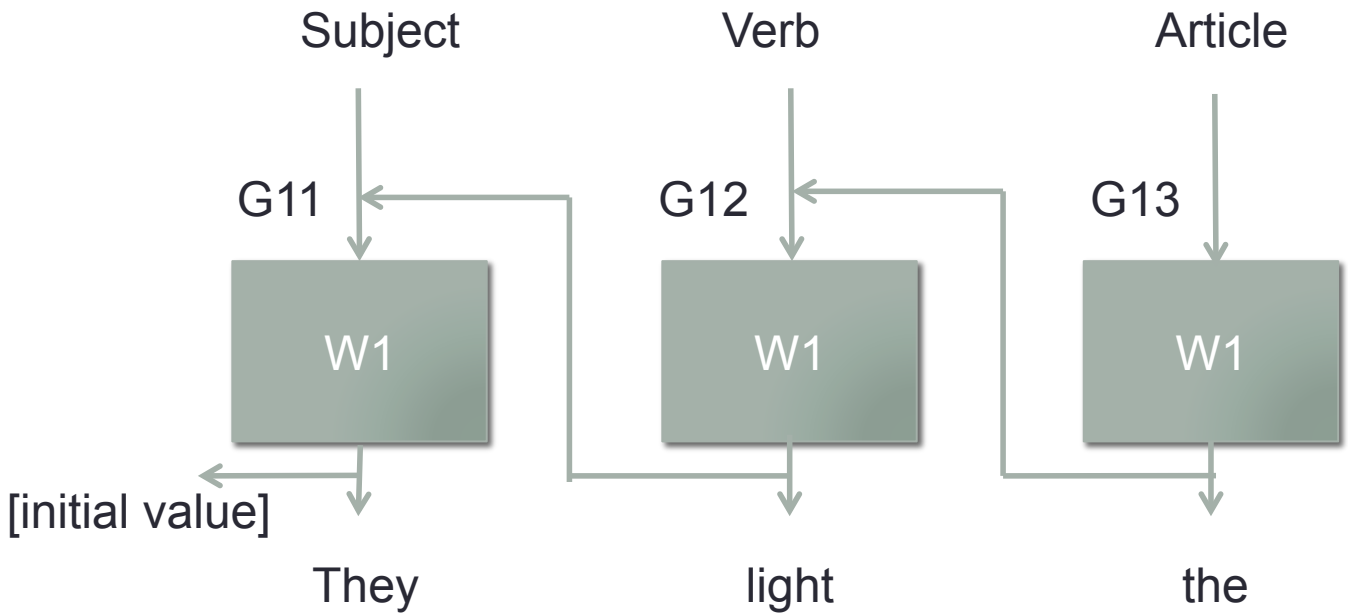


W1 <- W1 + G11 + G12

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

- Backward Computation graph

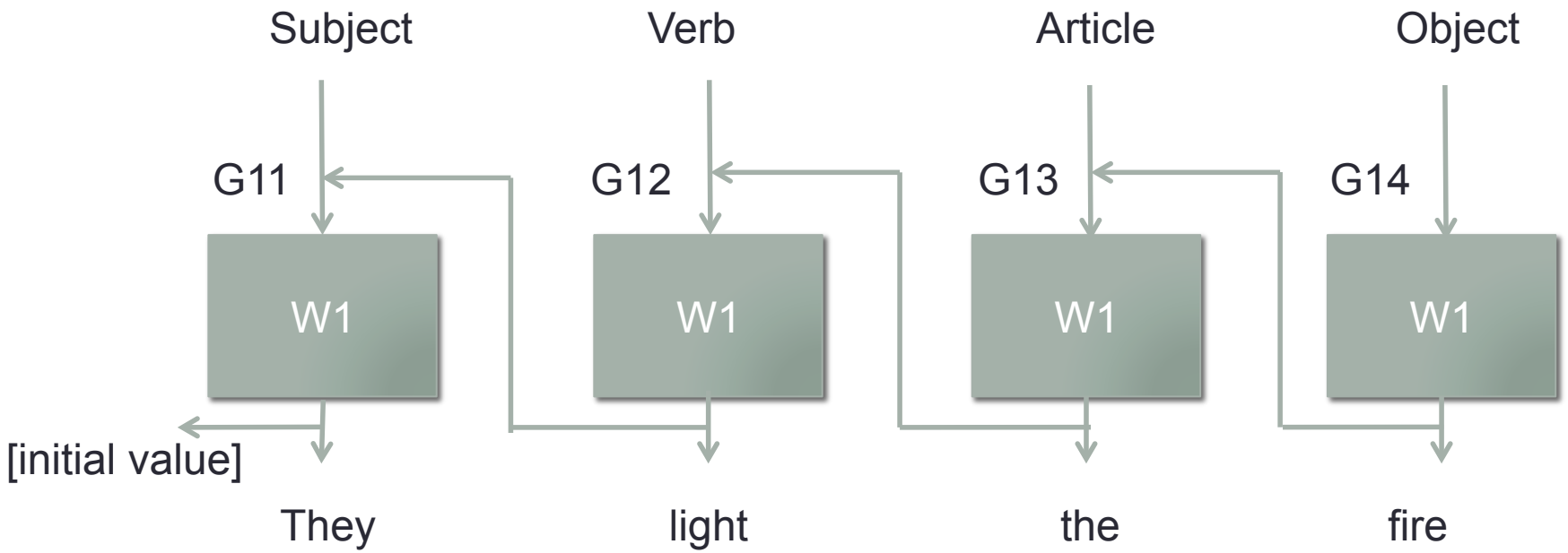


W1 <- W1 + G12 + G13

Pick a maximum number of time steps and go backwards that much

Truncated BPTT

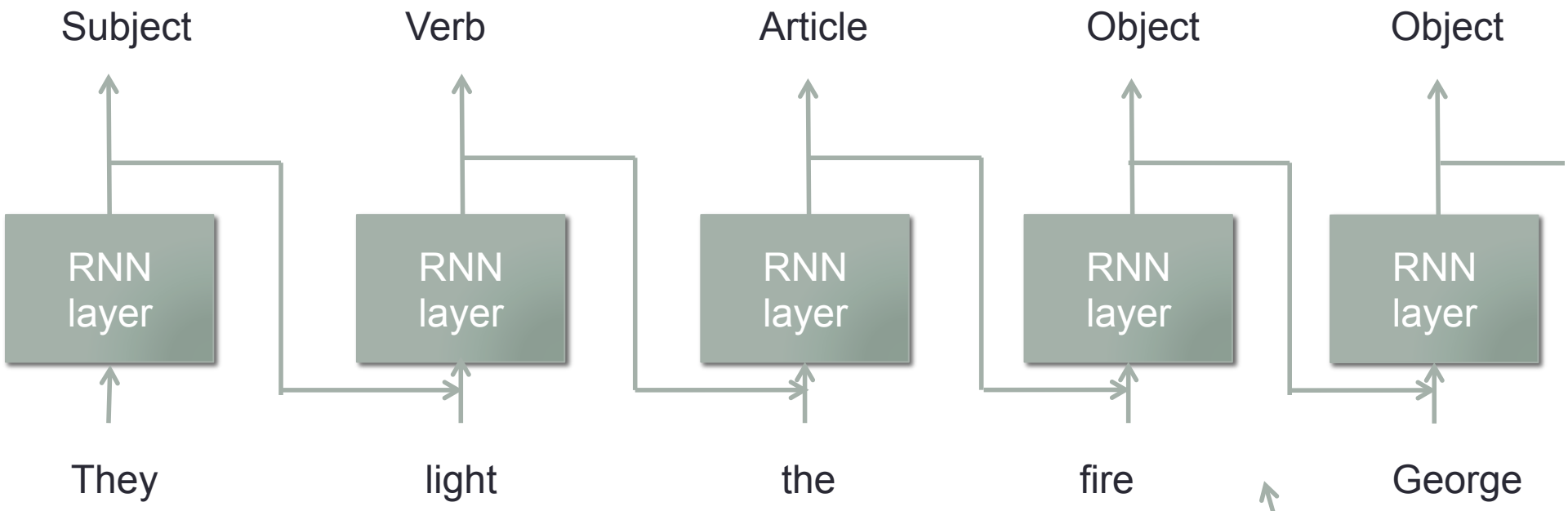
- Backward Computation graph



W1 <- W1 + G13 + G14

Pick a maximum number of time steps and go backwards that much

Recurrent neural network (RNN)

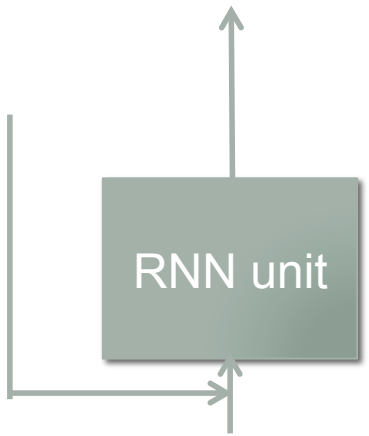


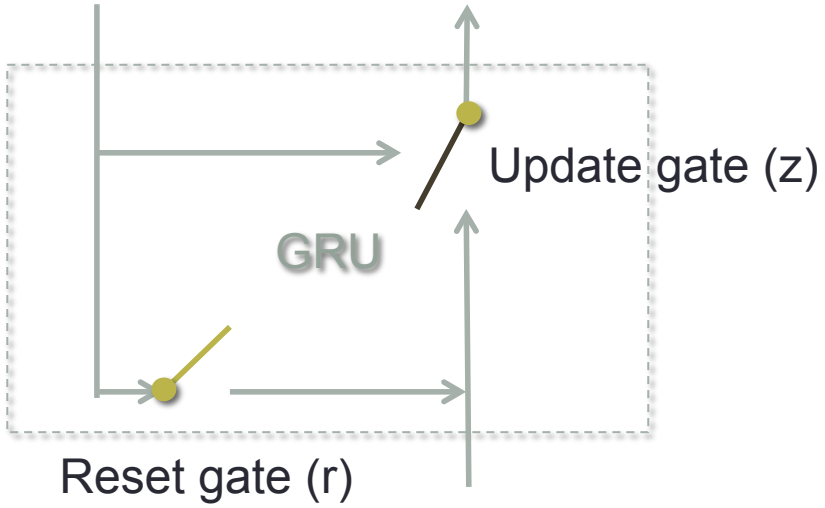
Problem2: needs a way to stop remembering

Can the network learn when to start and stop remembering things?

Gated Recurrent Unit (GRU)

- Forms a Gated Recurrent Neural Networks (GRNN)
- Add gates that can choose to reset (r) or update (z)

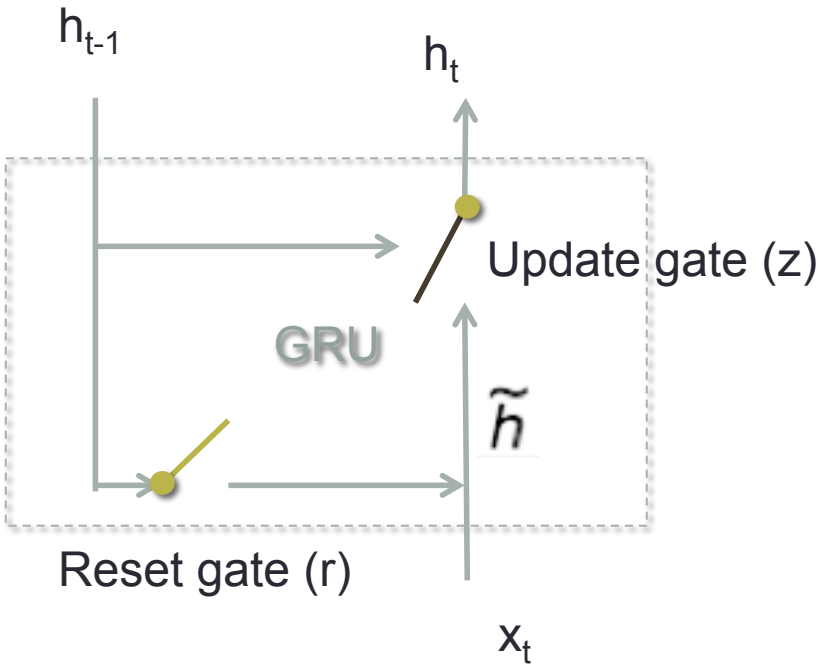




Gated Recurrent Unit (GRU)

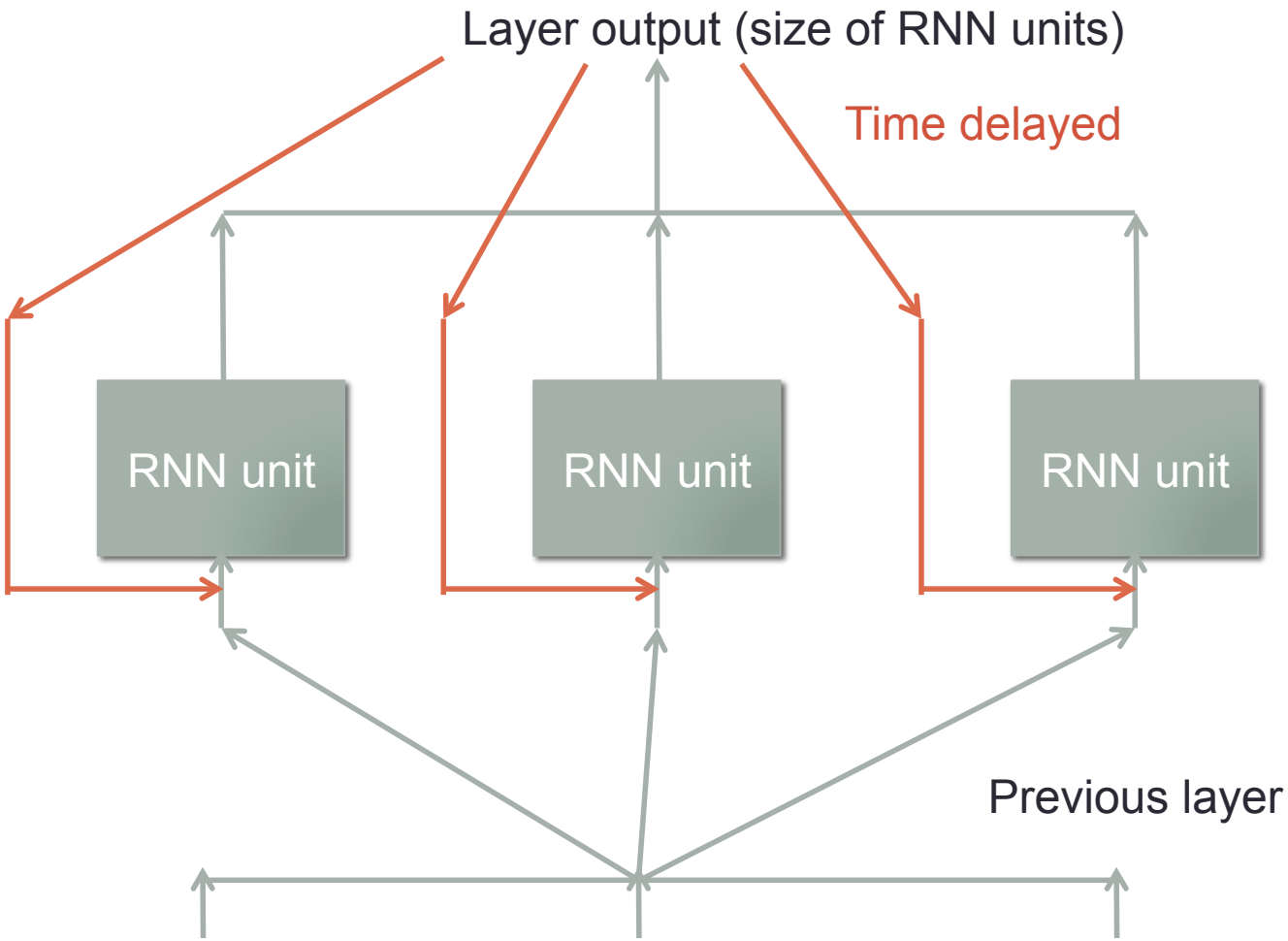
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

$\tilde{h}$



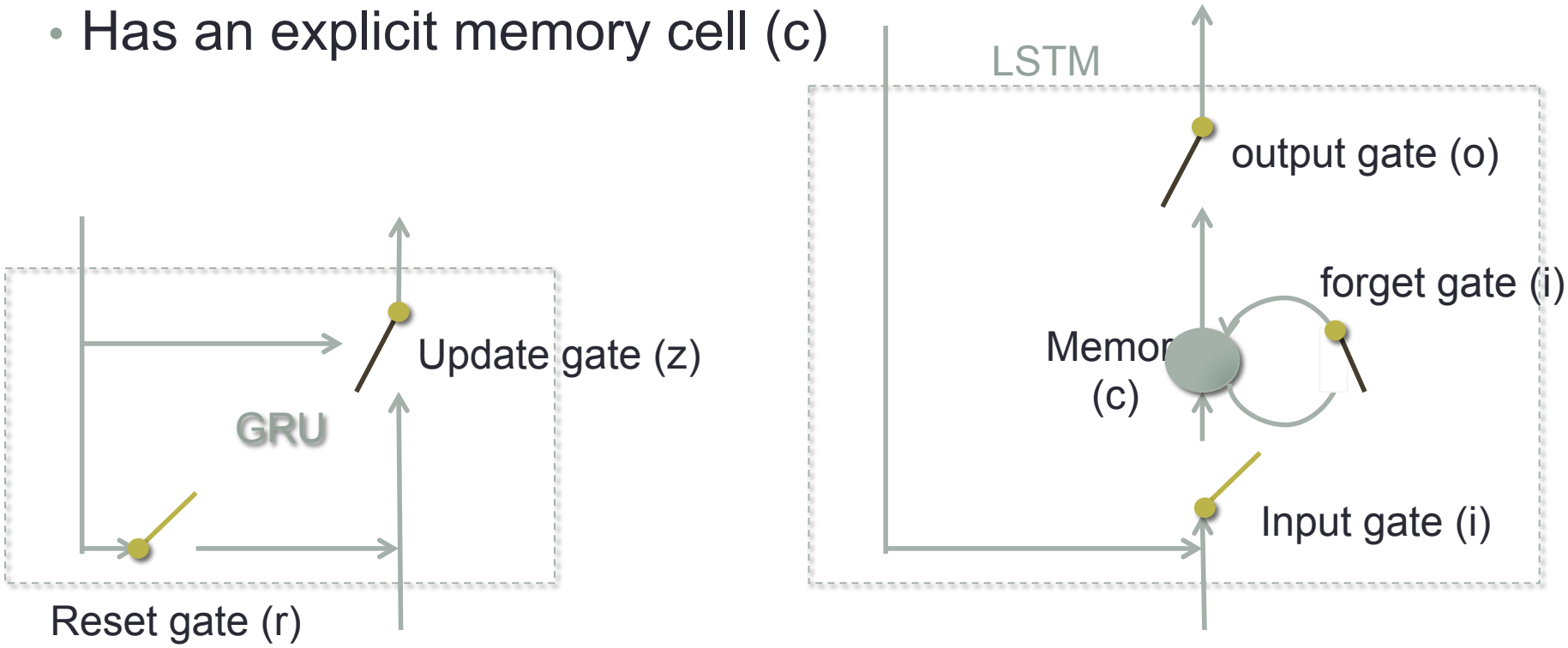
Note: non-linearity of GRU and LSTM are usually sigmoid/tanh – not ReLU

Gated Recurrent Unit (GRU) layer



Long Short-Term Memory (LSTM)

- Have 3 gates, forget (f), input (i), output (o)
- Has an explicit memory cell (c)



Both works for data with time dependency. Try GRU first

Long Short-Term Memory (LSTM)

j is the index of the LSTM cell

$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$

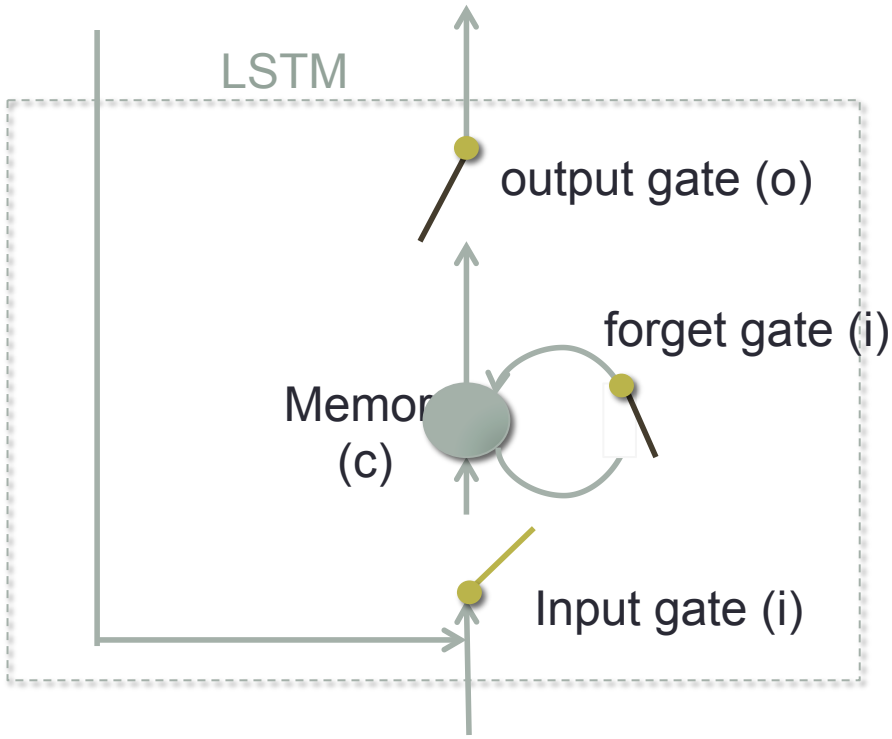
$$f_t^j = \sigma\left(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1}\right)^j$$

$$i_t^j = \sigma\left(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1}\right)^j .$$

$$\tilde{c}_t^j = \tanh\left(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1}\right)^j$$

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j$$

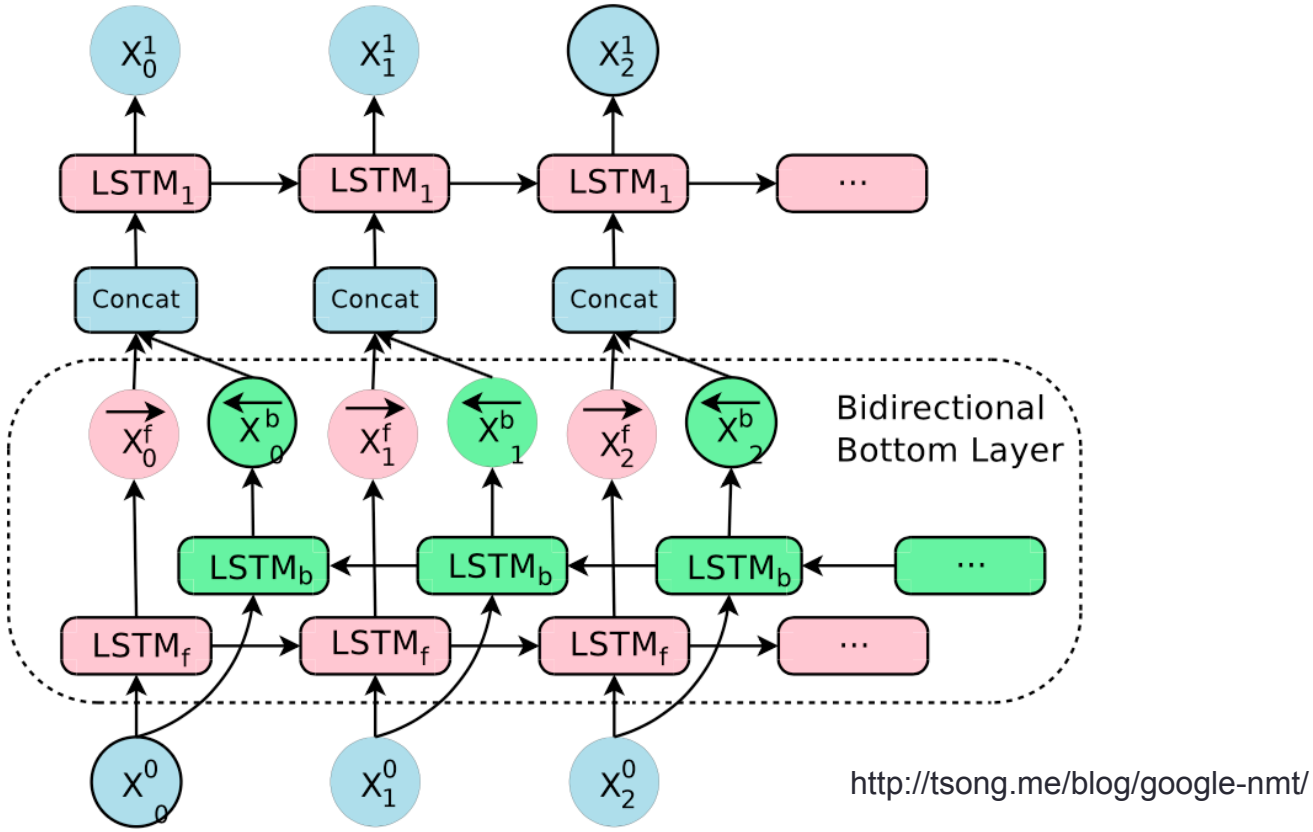
$$o_t^j = \sigma\left(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t\right)^j$$



Note V is diagonal (output of the cell is related to the memory of its own cell)

Bi-directional LSTM

- The previous GRU/LSTM only goes backward in time (uni-directional)
- Most of the time information from the future is useful for predicting the current output



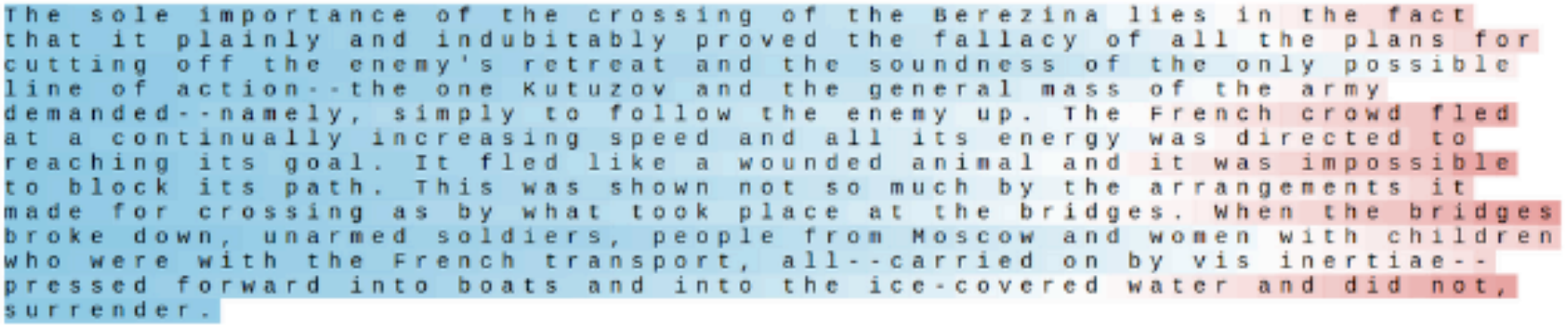
http://tsong.me/blog/google-nmt/

Real time bi-directional LSTM

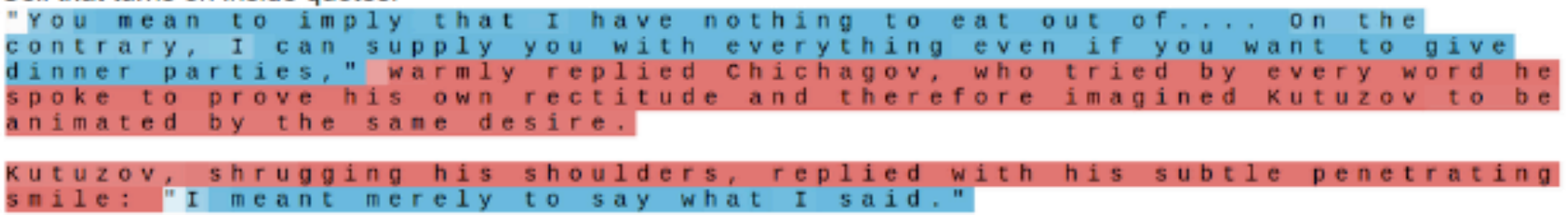
- For real time applications, only “look ahead” a certain amount of time steps
- Still helpful

LSTM remembers meaningful things

Cell sensitive to position in line:



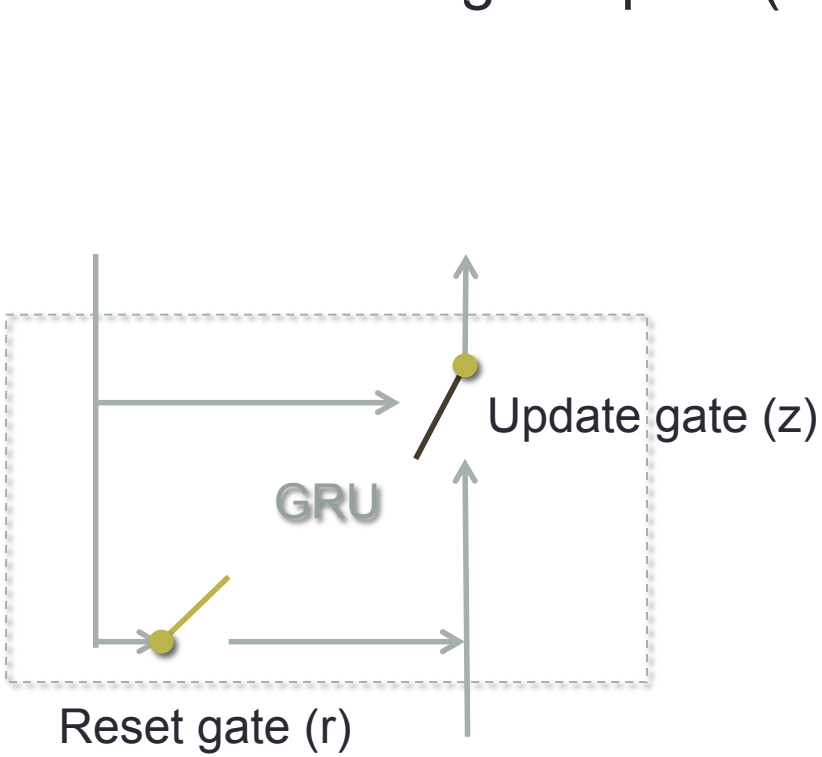
Cell that turns on inside quotes:

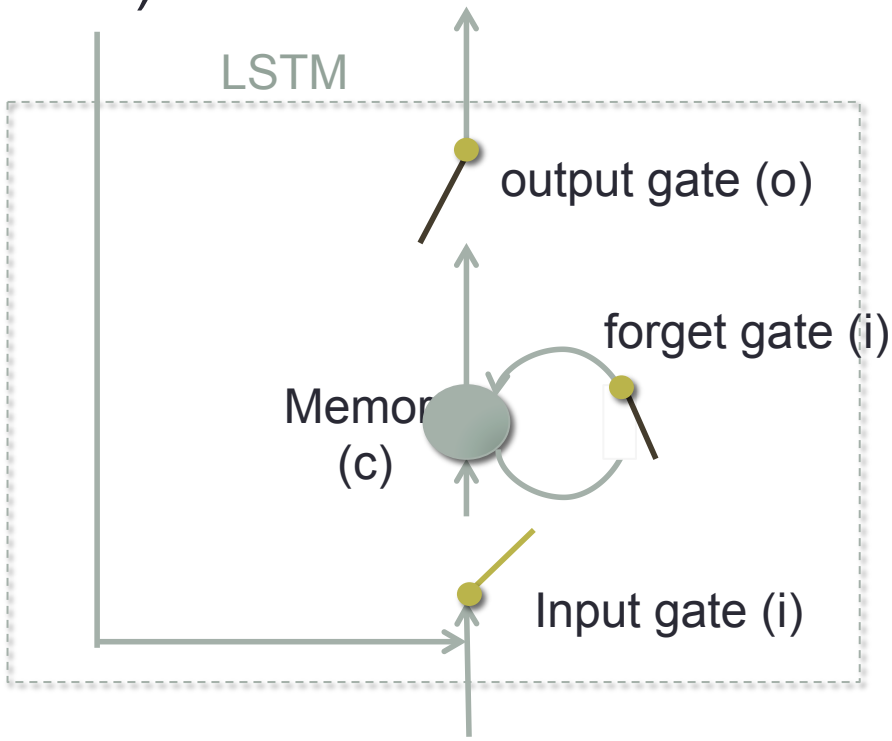


Andrej Karpathy, Visualizing and Understanding Recurrent Networks, 2015, https://arxiv.org/abs/1506.02078

LSTM/GRU summary

- Sharing of parameters across time
- Remembering the past (and future)





DNN Legos

- Typical models now consists of all 3 types
  - CNN: local structure in the feature. Used for feature learning.
  - LSTM: remembering longer term structure or across time
  - DNN: Good for mapping features for classification. Usually used in final layers

Tensorboard demo

Project doodle

- Build your group on courseville (under project)
- Submit a proposal. Due next Tuesday.
  - Bullet points
  - What do you want to do?
  - What is the data?
  - How to evaluate the task?
- Sign up for time slots (next weeks' office hour)
  - Sign up sheet TBA.