

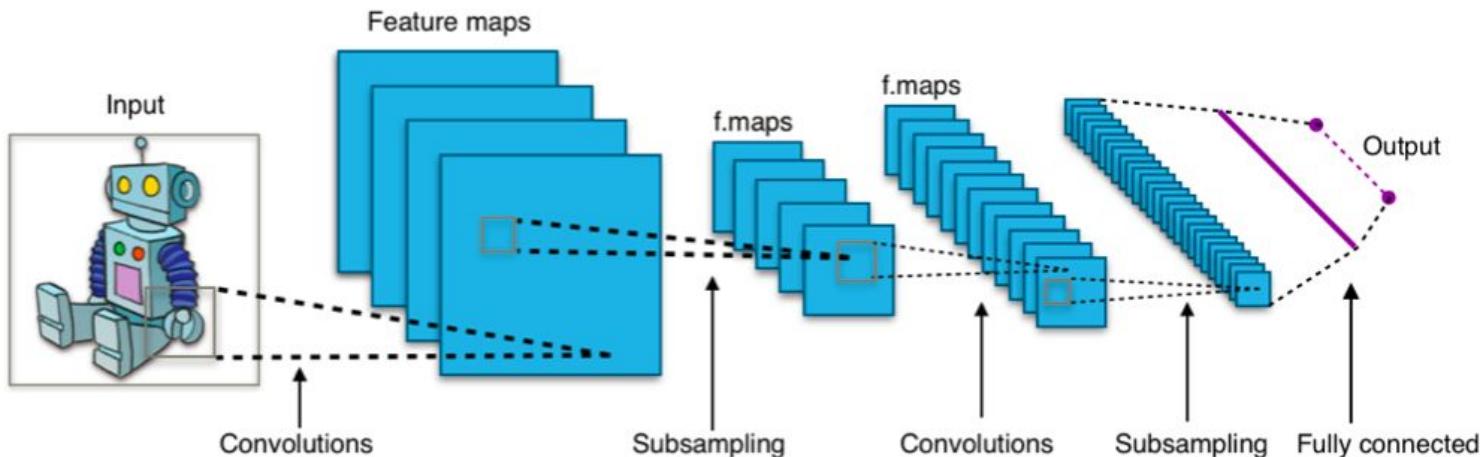
CONVOLUTIONAL AND RECURRENT NEURAL NETWORKS

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

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

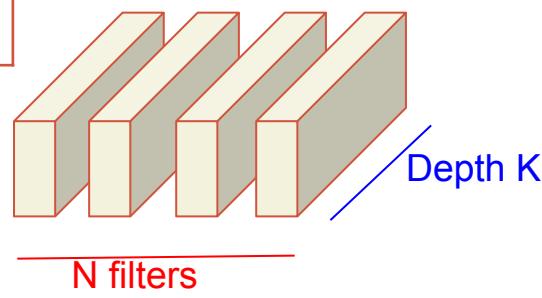
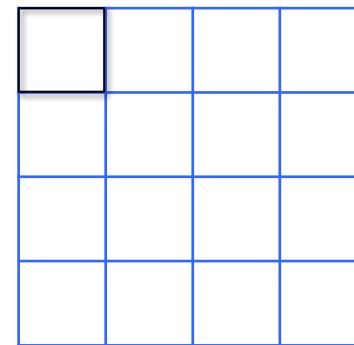
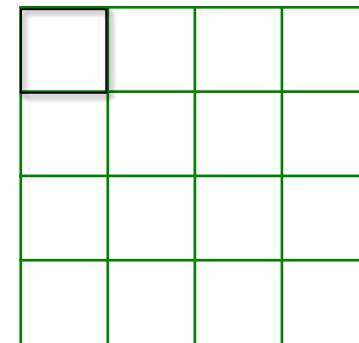
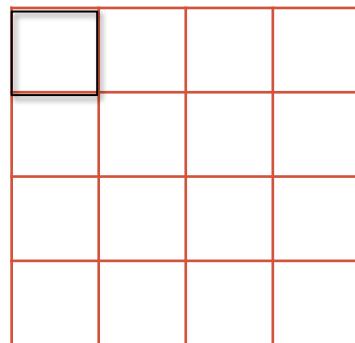


Common schemes

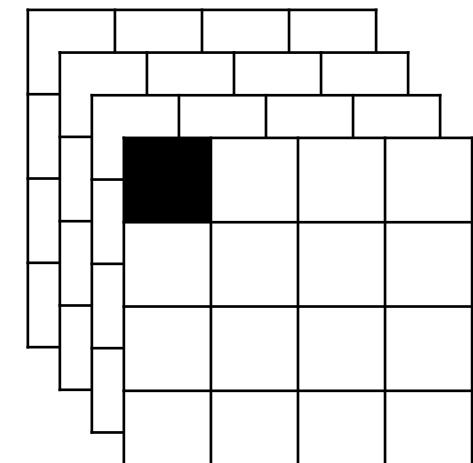
- INPUT -> [CONV -> RELU -> POOL]^{*N} -> [FC -> RELU]^{*M} -> FC
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]^{*N} -> [FC -> RELU]^{*M} -> FC
- If you working with images, just use a winning architecture.

1x1 convolution

Reduces the dimension of feature maps

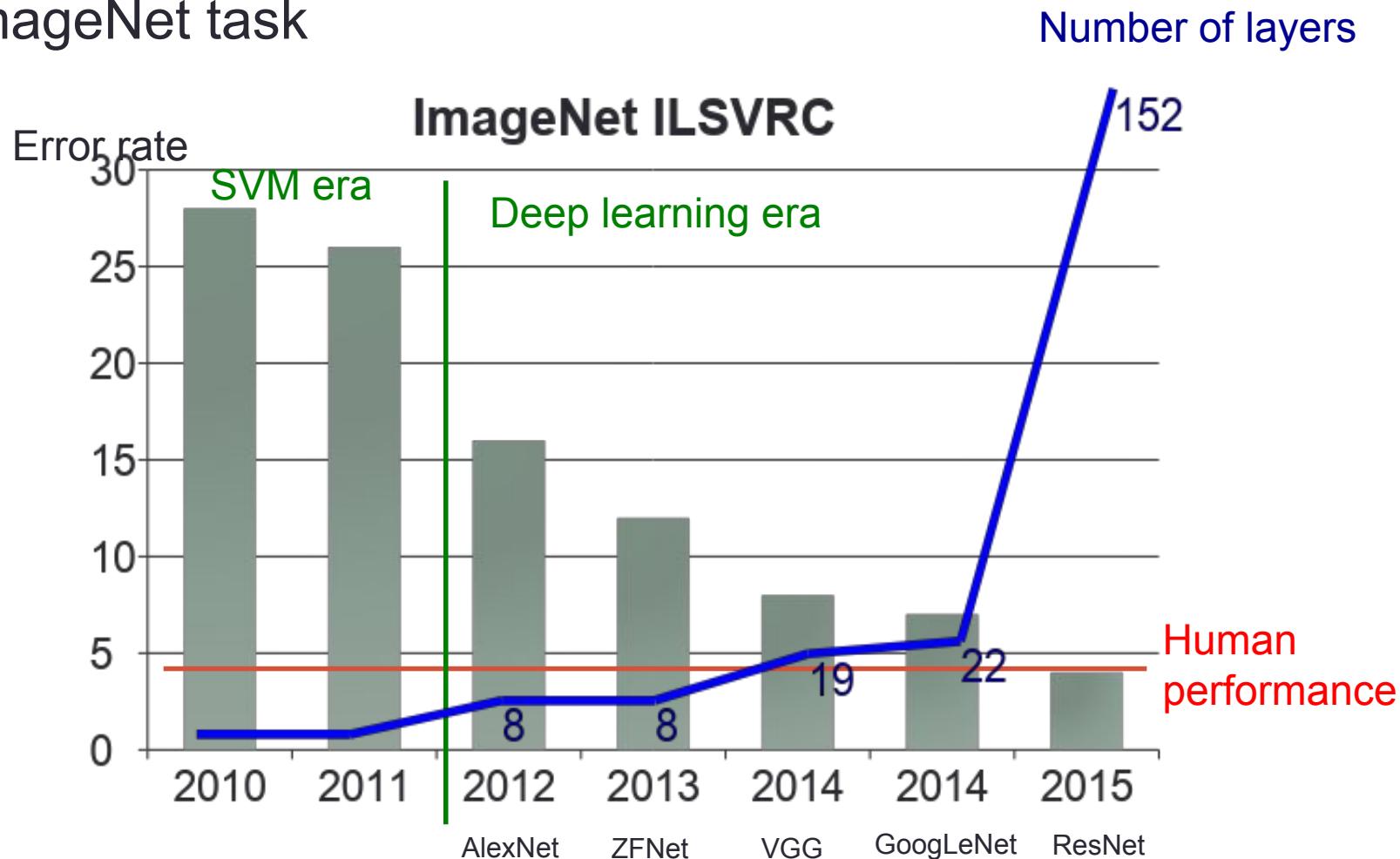


The filter is actually $1 \times 1 \times K$



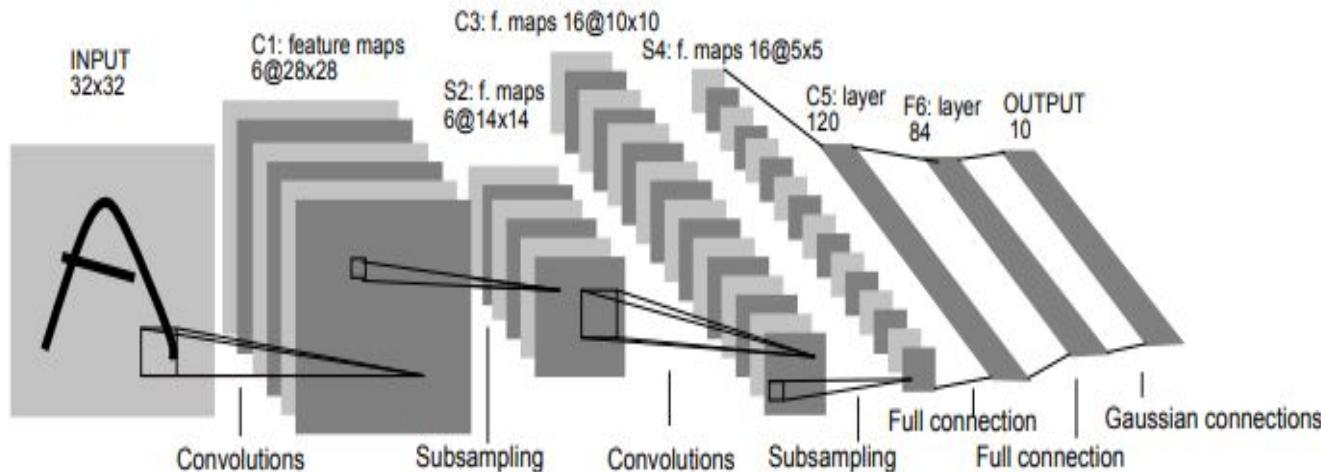
A brief history of imagenet architectures

- ImageNet task



LeNet

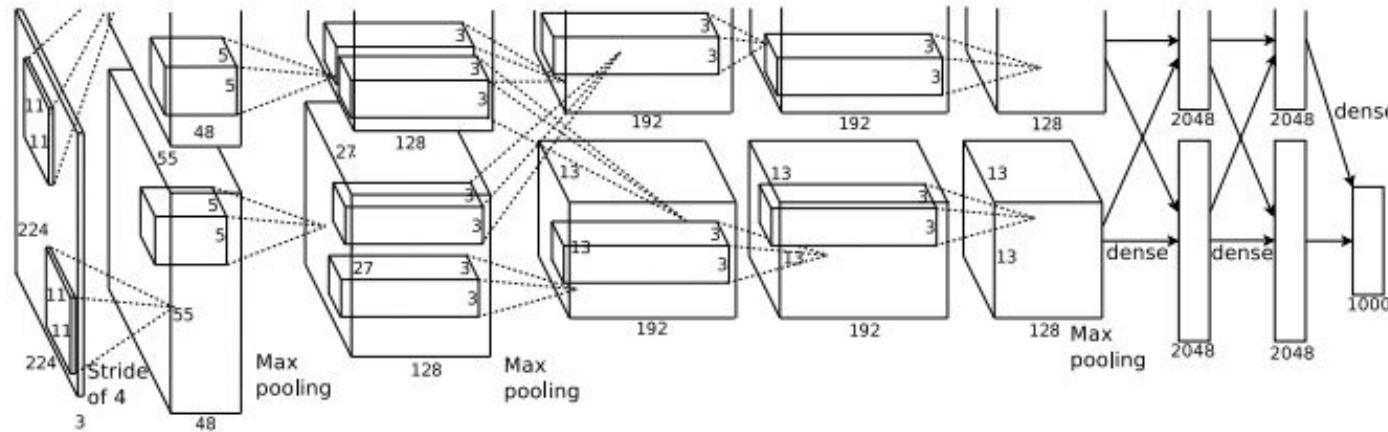
Convolutions and poolings followed by fully connected layers
Tanh activations
Ability to handle larger images limited by compute



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition," 1998.

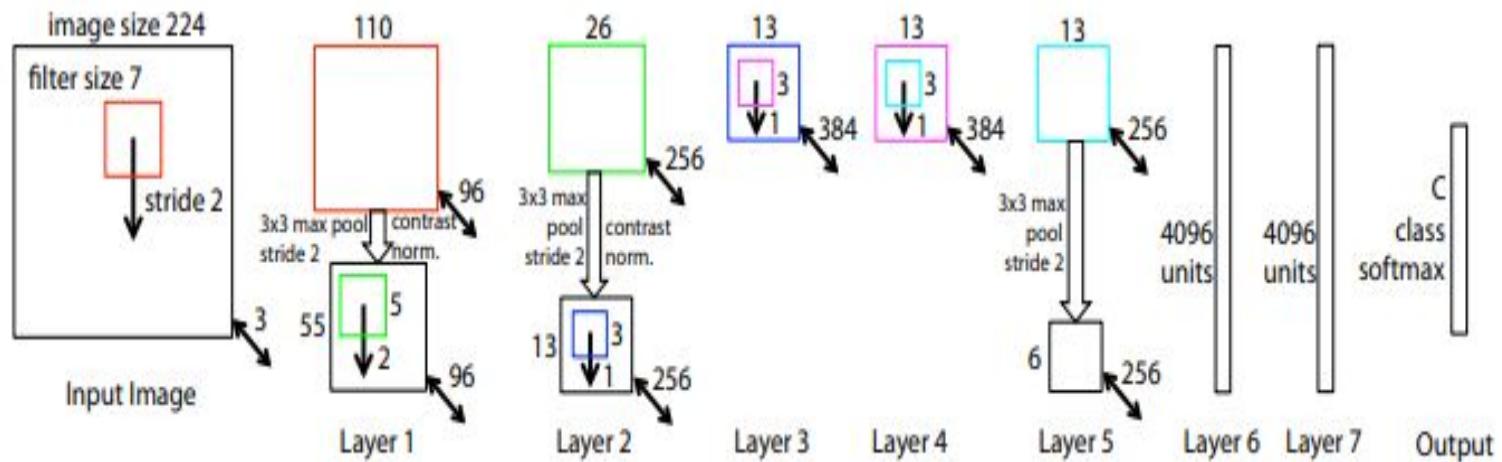
AlexNet

Convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum
Two pipelines to fit into two GPUs



ZFNet

Tweaking hyperparameters



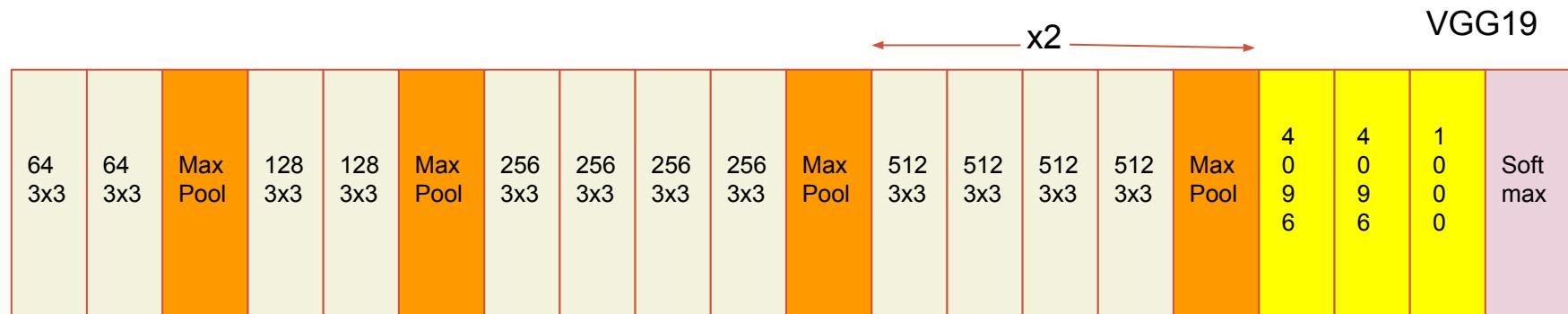
Matthew D. Zeiler, Rob Fergus, "Visualizing and Understanding Convolutional Networks," 2013

VGG

Uniform 3x3 convolutional filters

19 layers! Pushing the limits of conventional wisdom at that time.

Used by many since pre-train weights are publically available

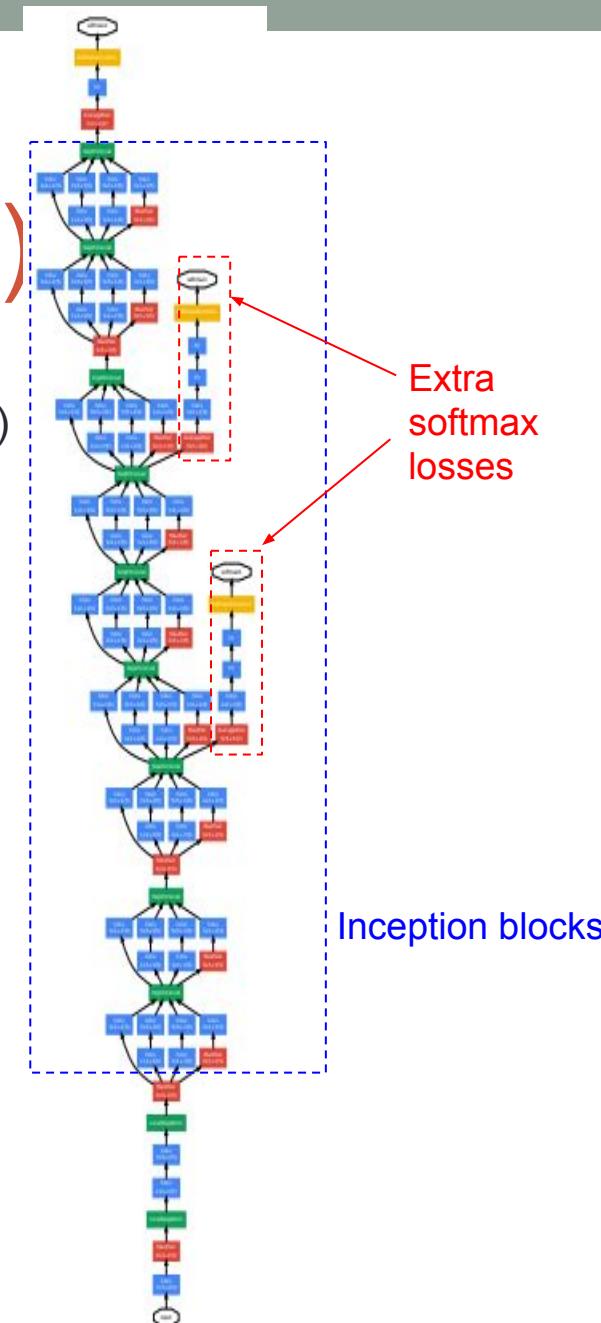
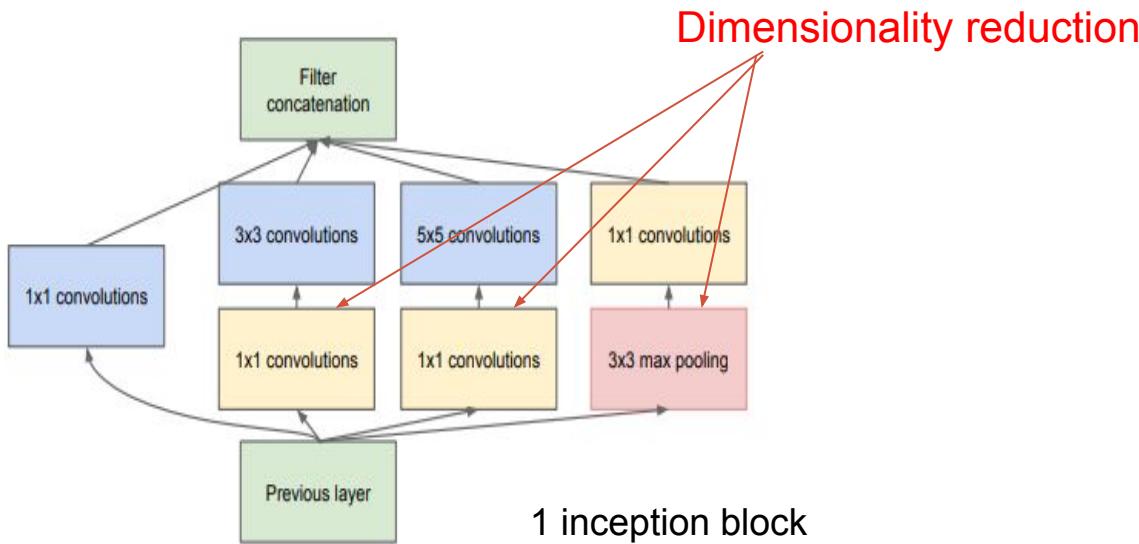


GoogLeNet (Inception v1)

Multiple filter sizes per layer (objects come in different scales)

Dimensionality reduction via 1x1 convolution

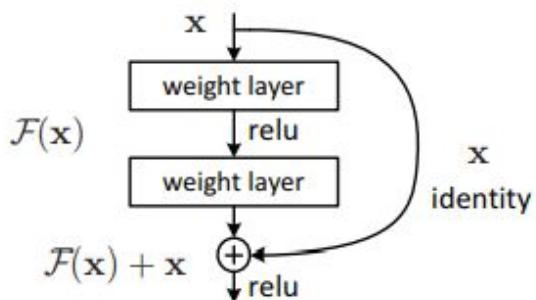
Multiple softmax losses to help the gradient problem



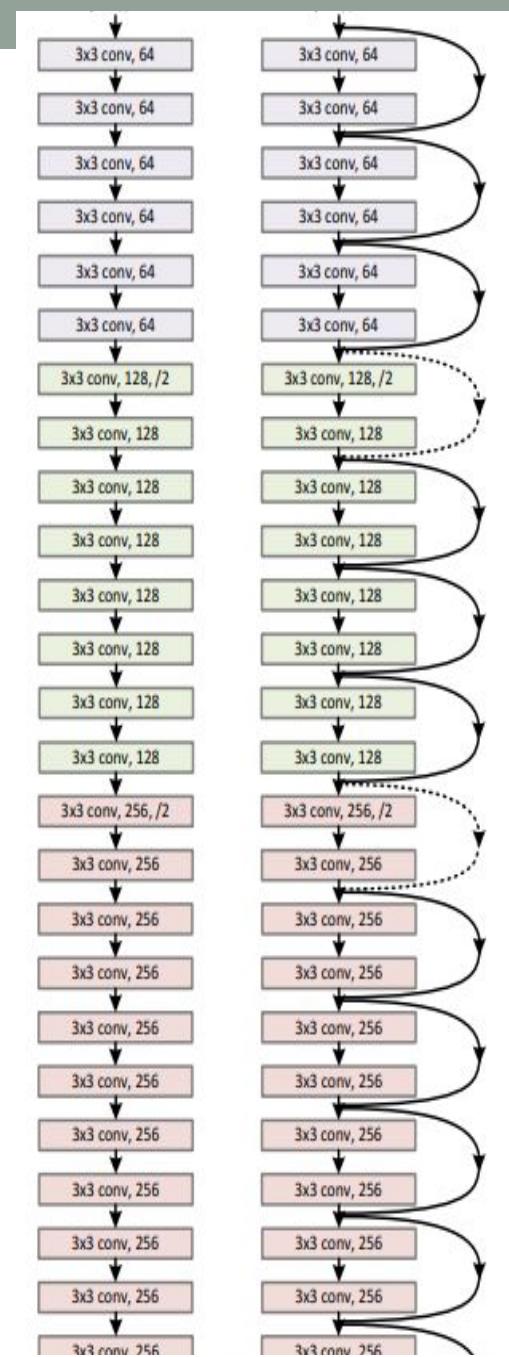
ResNet (Residual Network)

Batch norm

Extra “skip connections” to reduce
the vanishing gradient problem

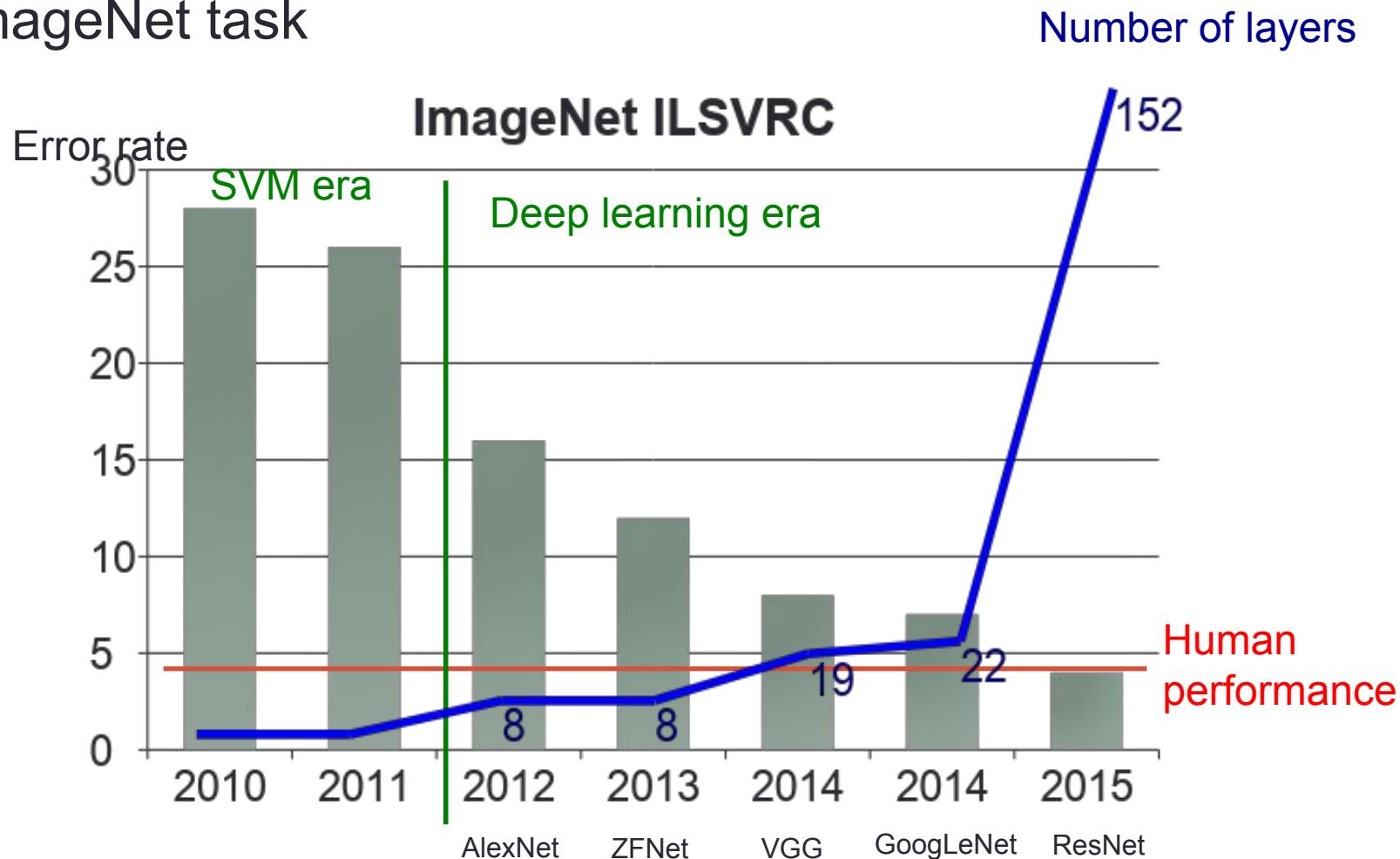


Kaiming He, et al. “Deep Residual Learning for Image Recognition” 2015



A brief history of imagenet architectures

- ImageNet task



Inception v2+v3

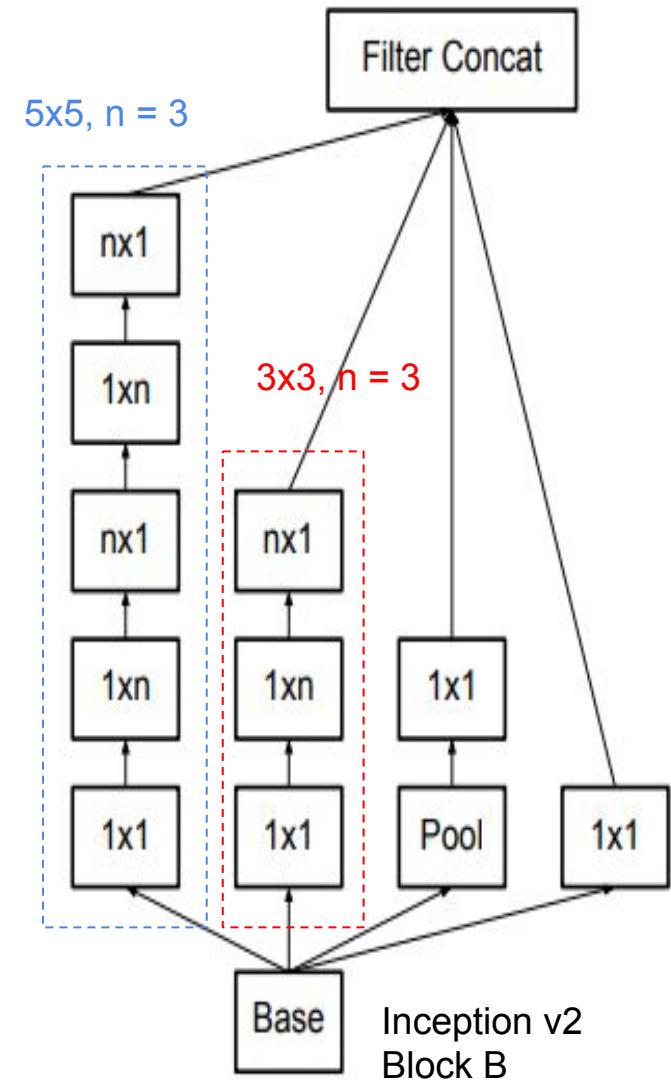
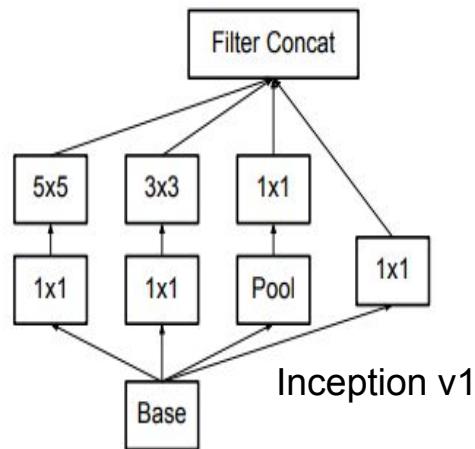
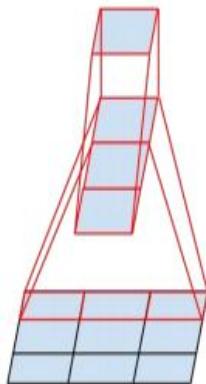
Implement 5x5 with two 3x3s

Factorized convolution

3x3 -> 3x1 and 1x3

3 types of inception blocks

RMSprop, Batch norm, label smoothing



Christian Szegedy, "Rethinking the Inception Architecture for Computer Vision," 2015

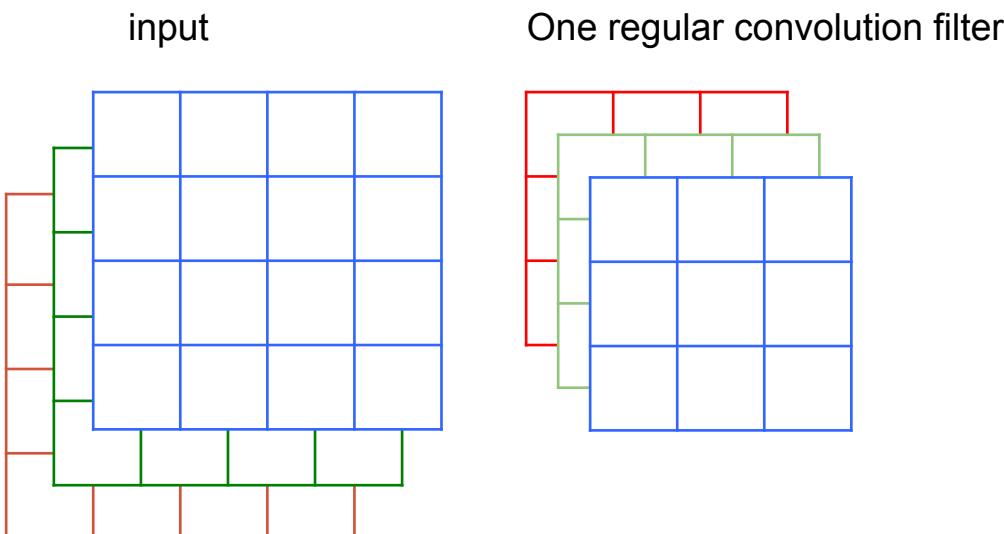
Xception

Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

Typical convolution 3x3 filter is
3x3xinput channel

Depthwise convolution 3x3 filter is
3x3x1
1 filter per input channel



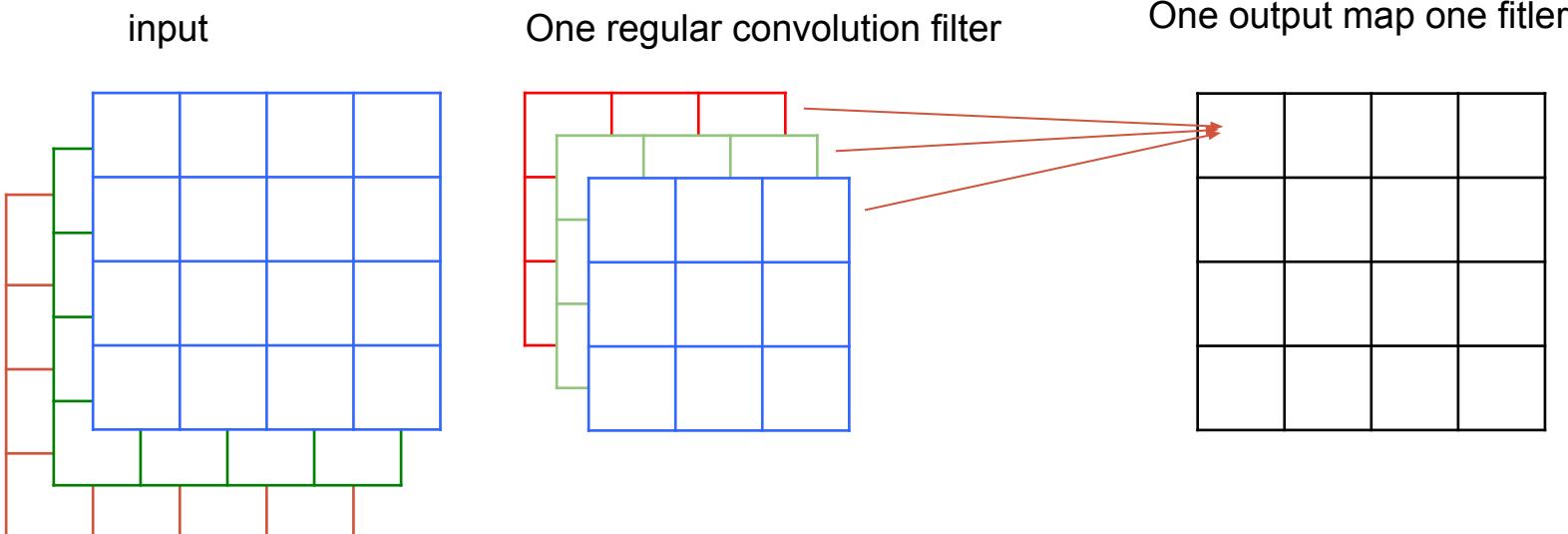
Xception

Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

Typical convolution 3x3 filter is
3x3xinput channel

Depthwise convolution 3x3 filter is
3x3x1
1 filter per input channel

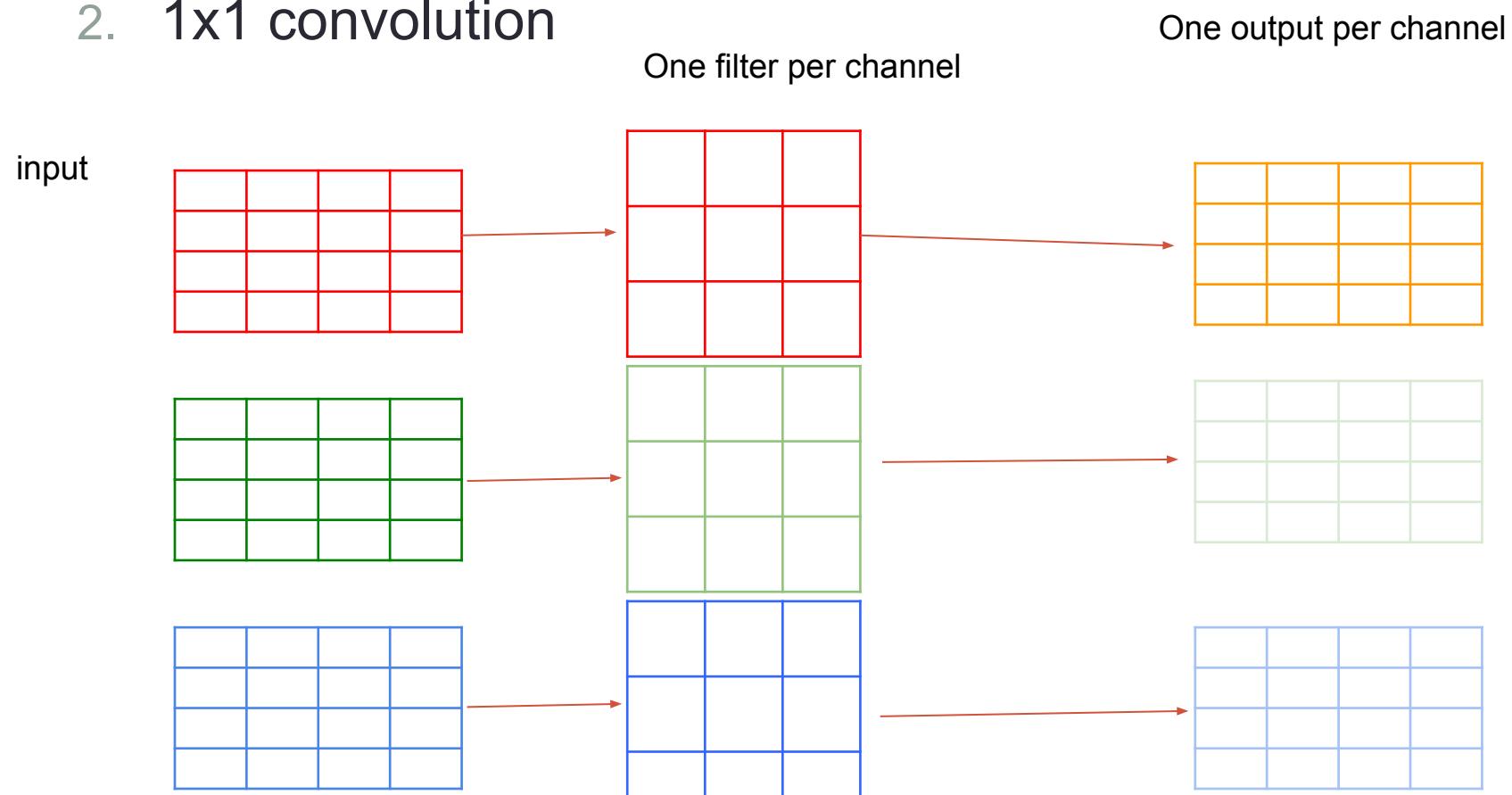


Xception

Depthwise separable convolution: two-step convolution

1. Depthwise convolution

2. 1x1 convolution

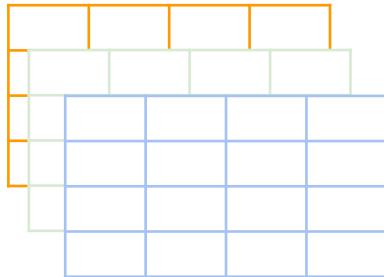


Xception

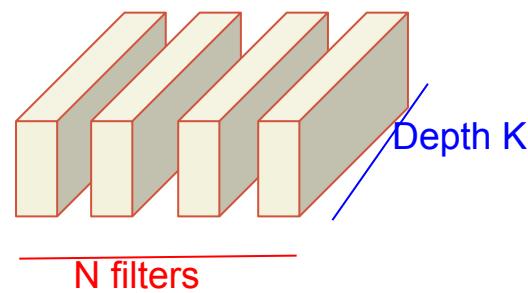
Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. **1x1 convolution**

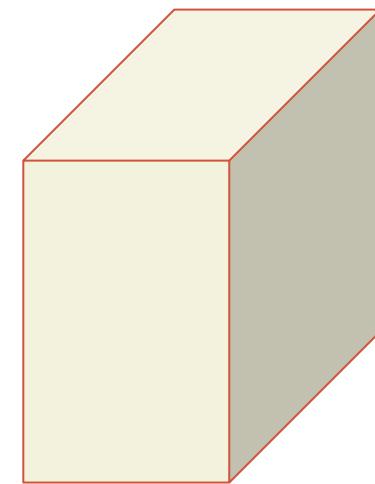
Output from depthwise convolution



1x1 filters



Final output



Xception

Replace convolutions in inception with depthwise separable convolutions

Smaller model

Faster compute

Comparable with Inception v3 while much faster

Used in MobileNet and other models

François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions” 2016.

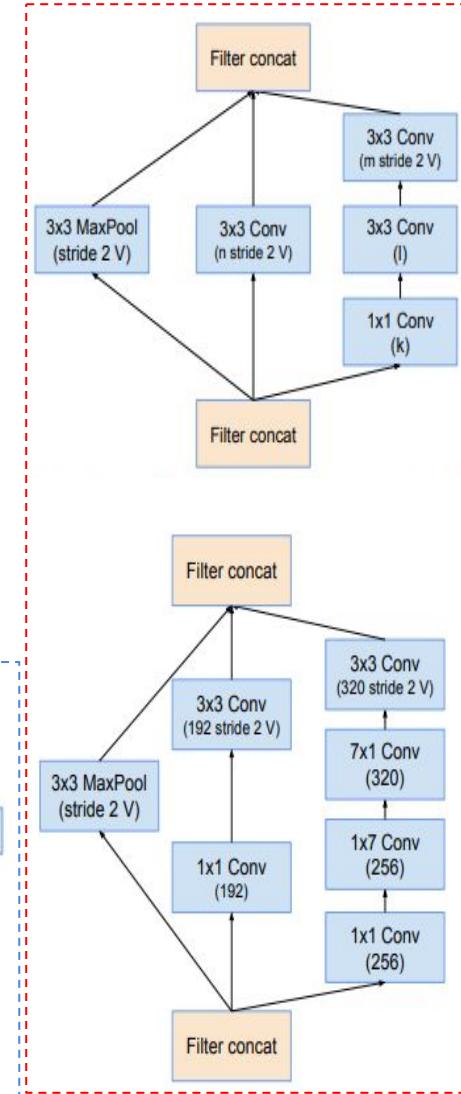
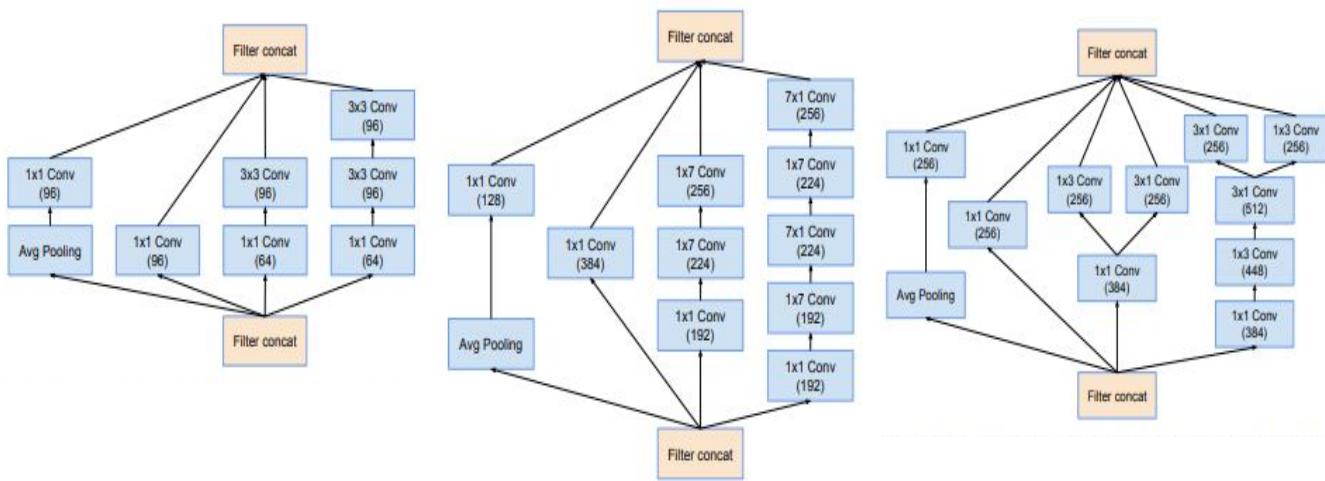
Andrew G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” 2017.

Inception v4

Same three types of inception blocks from v3

Add two types of **reduction blocks** for reducing the size of the grid (super pooling blocks)

Inception blocks



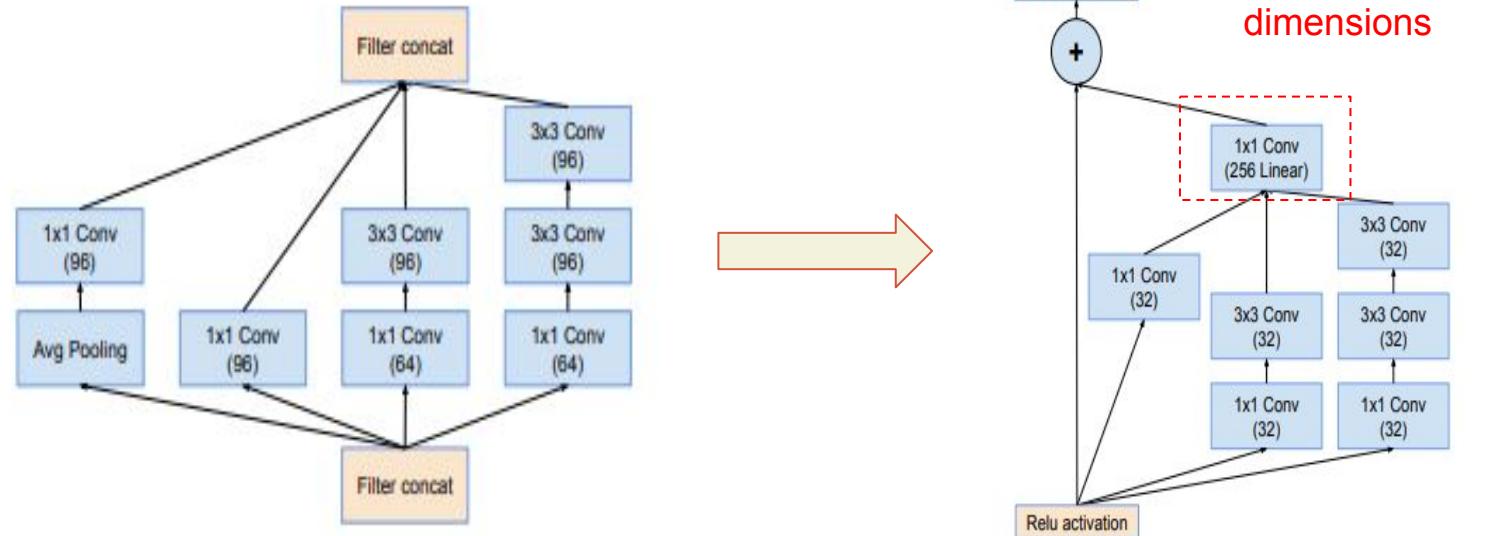
Reduction blocks

Inception-ResNet v1-v2

Introduce residual connections into inception blocks

Poolings changed to additions, 1x1 added to keep dimensions for the residual

Similar idea to ResNeXt



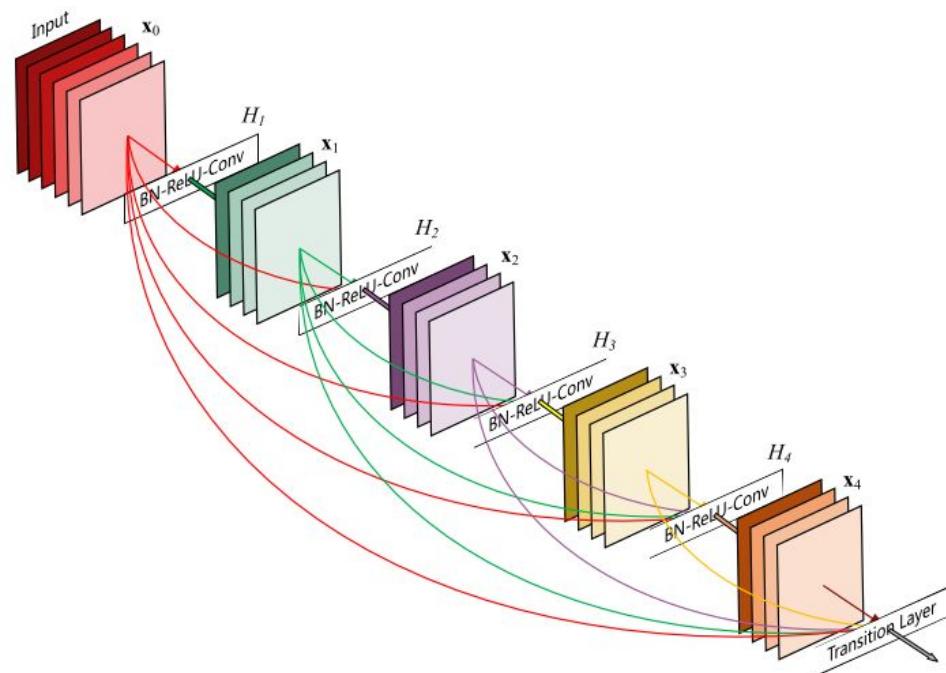
DenseNet

Instead of adding the residuals, concatenates the feature maps from previous layers

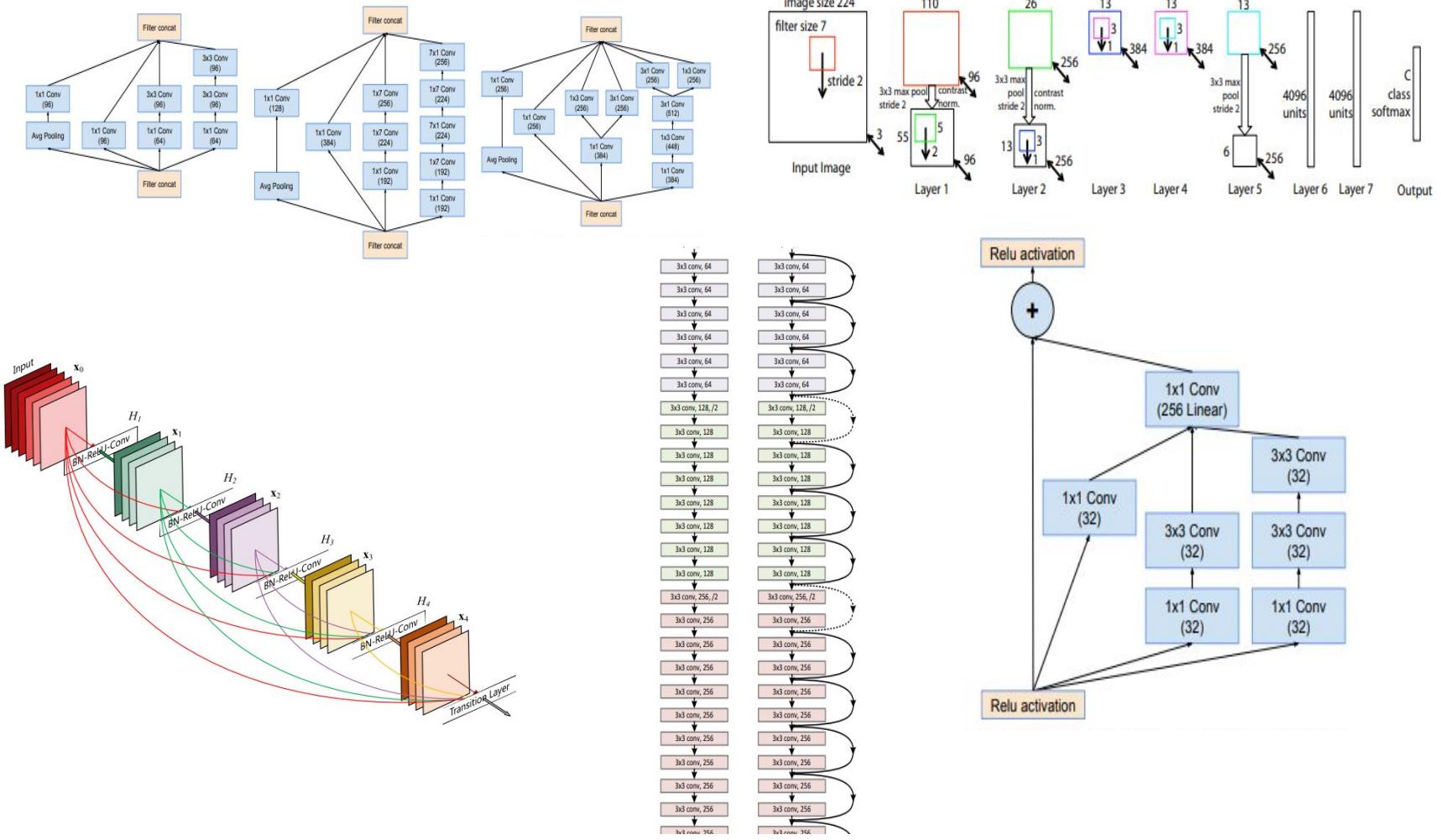
Densely connect multiple layers

Multi-scale feature maps

Smaller model due to smaller number of channels



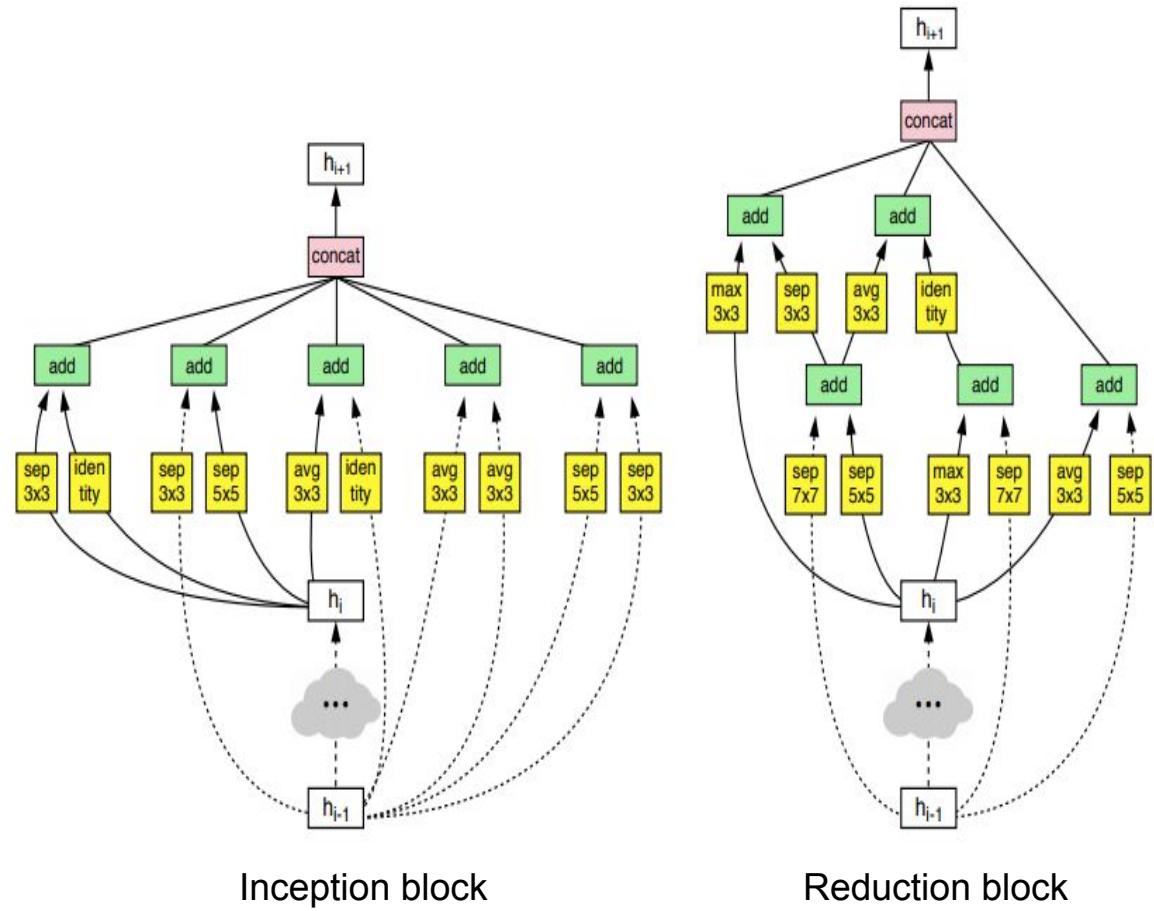
Do people keep tweaking network architectures until the end of time?



NasNet

Neural Architecture Search Network

Use reinforcement learning to search for the best network configuration
Lowers compute while maintaining high accuracy

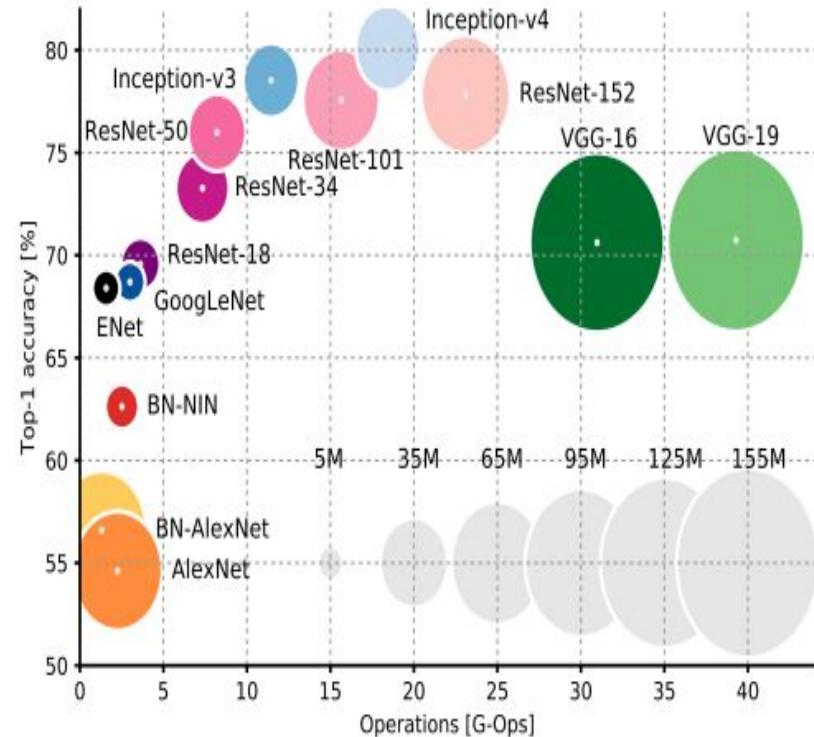


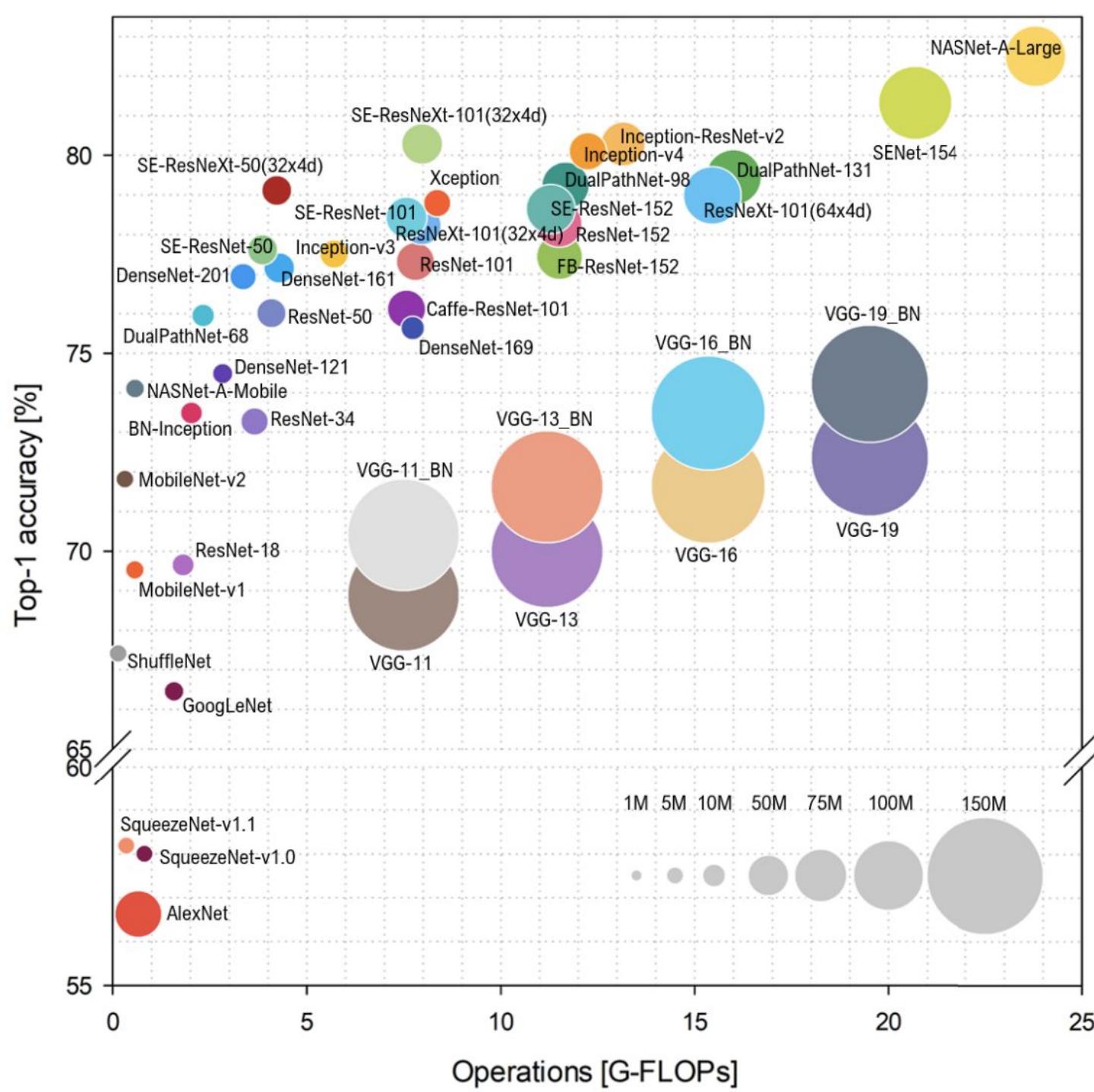
Object classifiers summary

Most successful tricks:
Dropout, residual, batch
norms, multi-path, data
augmentation

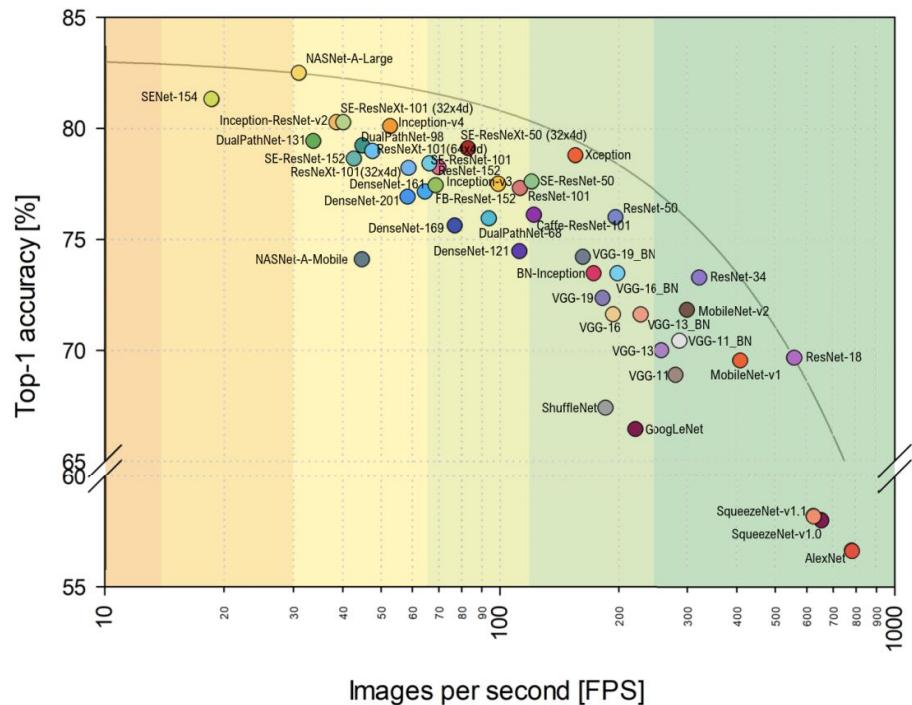
These object classifiers are
often called **backbones** and
used by other models

Comparing accuracy, model
size, transferability and
compute.

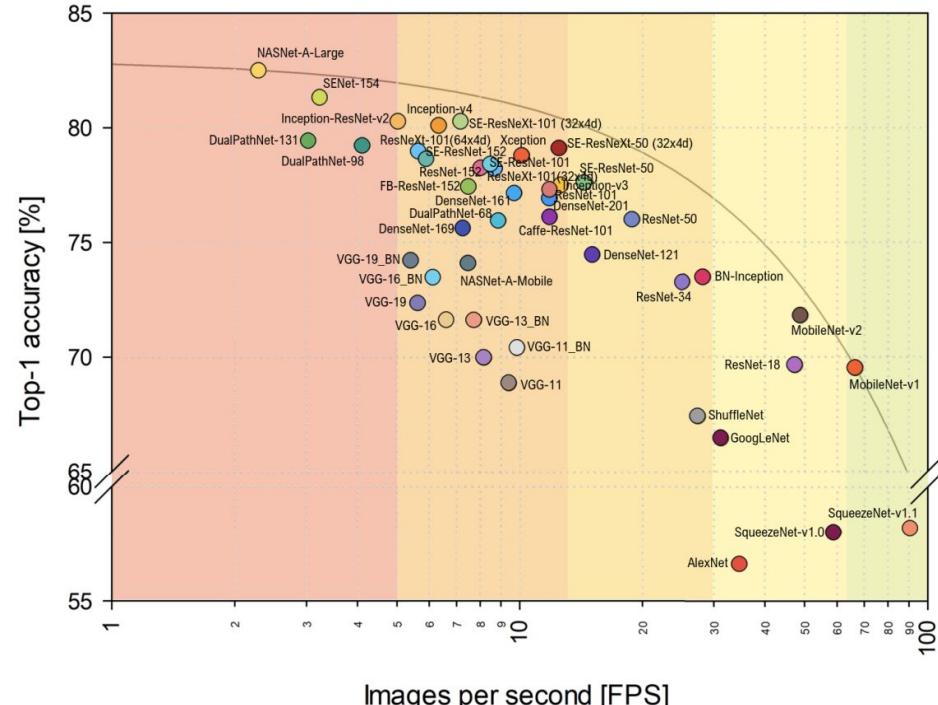




Simone Bianco, et al.,
Benchmark Analysis of
Representative Deep
Neural Network
Architectures, 2019



(a)



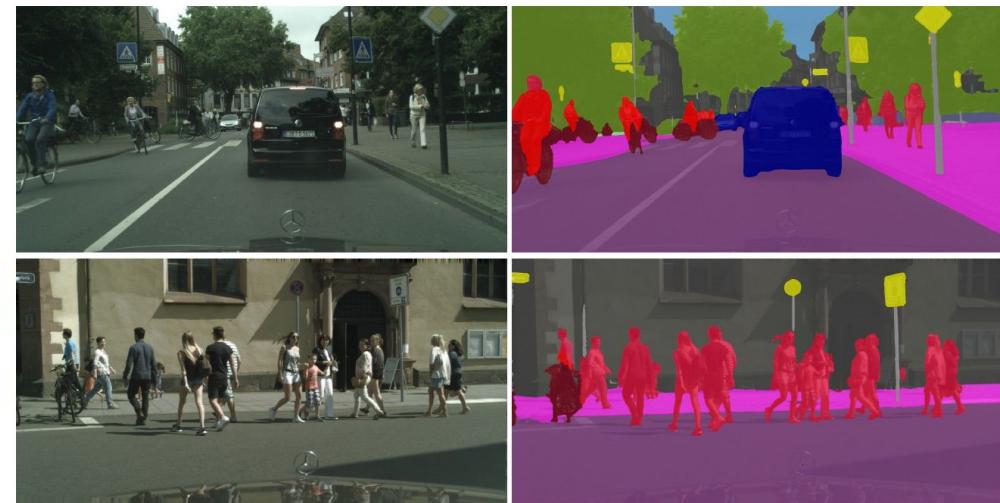
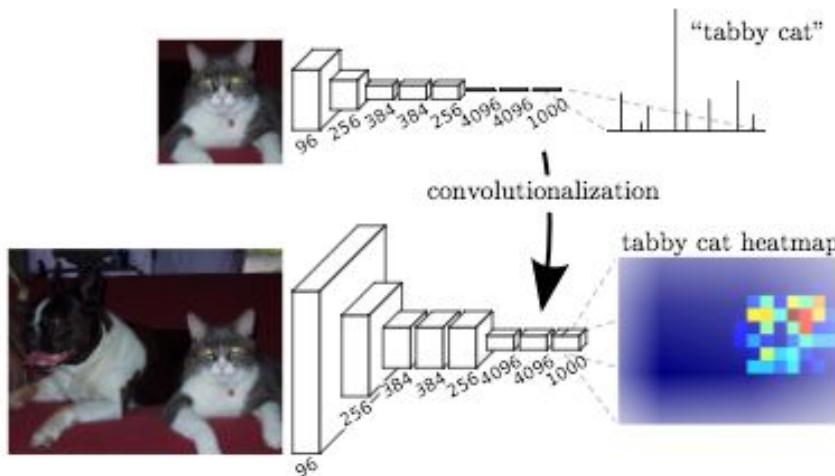
(b)

FIGURE 3. Top-1 accuracy vs. number of images processed per second (with batch size 1) using the Titan Xp (a) and Jetson TX1 (b).

Simone Bianco, et al.,
Benchmark Analysis of
Representative Deep
Neural Network
Architectures, 2019

Sometimes you want to increase the size of your feature map

Image segmentation



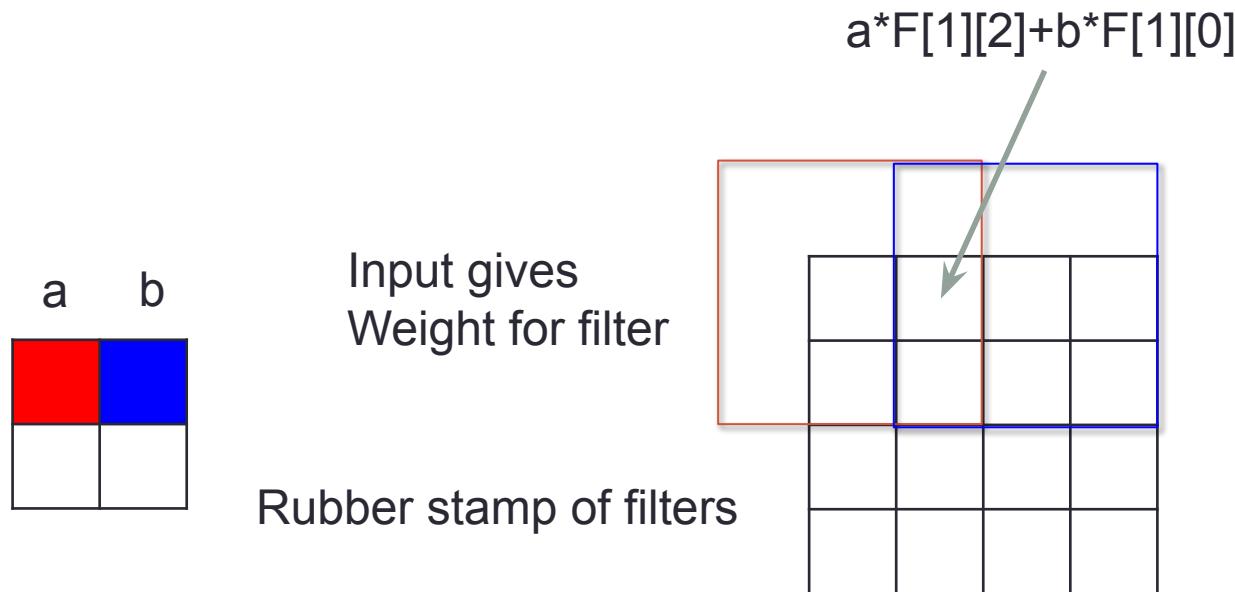
2 main approaches to upsample
De-convolution
resize (unpooling) + convolution

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

<http://vladlen.info/publications/feature-space-optimization-for-semantic-video-segmentation/>

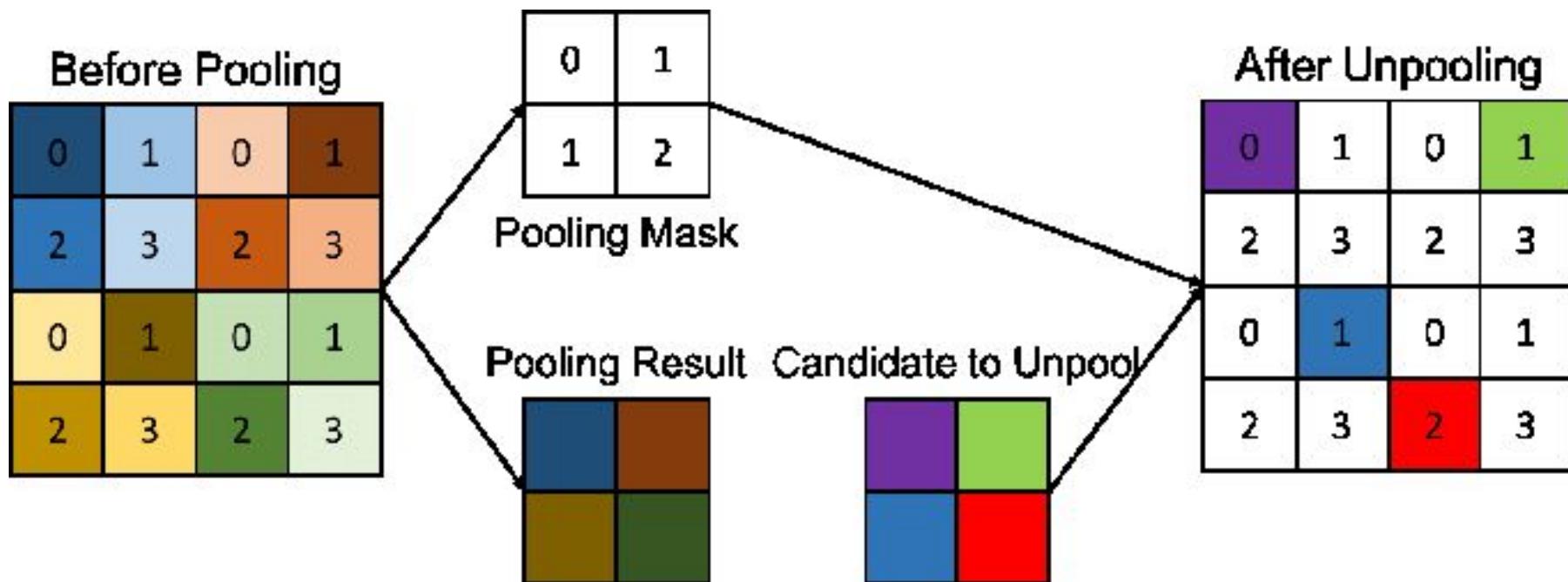
De-convolution (Upsampling)

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



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

Unpooling + conv



Remember the location of the max value

Put the value to unpool at that location

Fill the rest with zeroes

Follow by regular convolution to smooth the image

Upsampling notes

Deconvolution filter size should be a multiple of the stride to avoid **checkerboard** artifacts

Unpooling can be replaced with regular image resizing techniques (interpolation)

In keras, upsampling2d -> conv2d

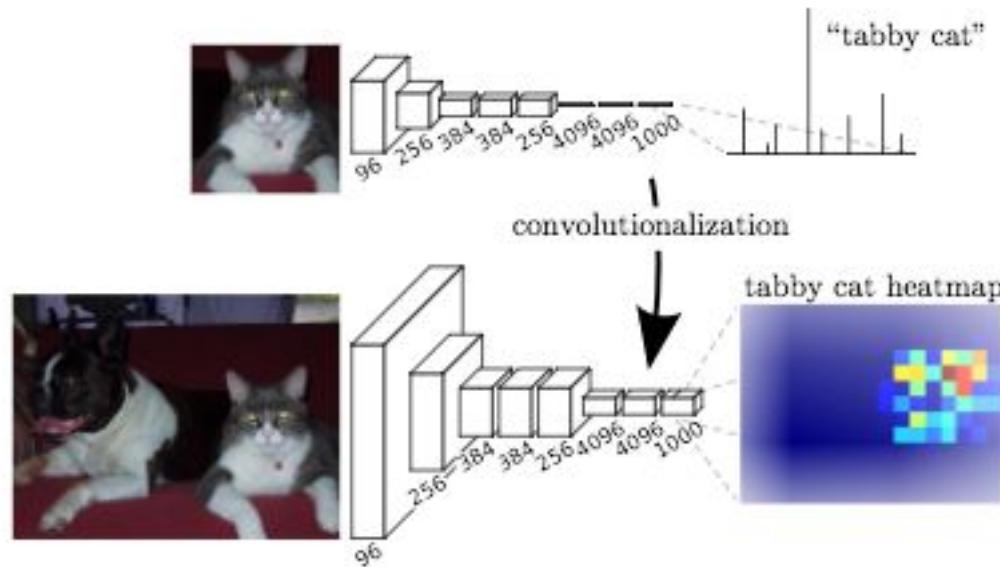


Image using deconv

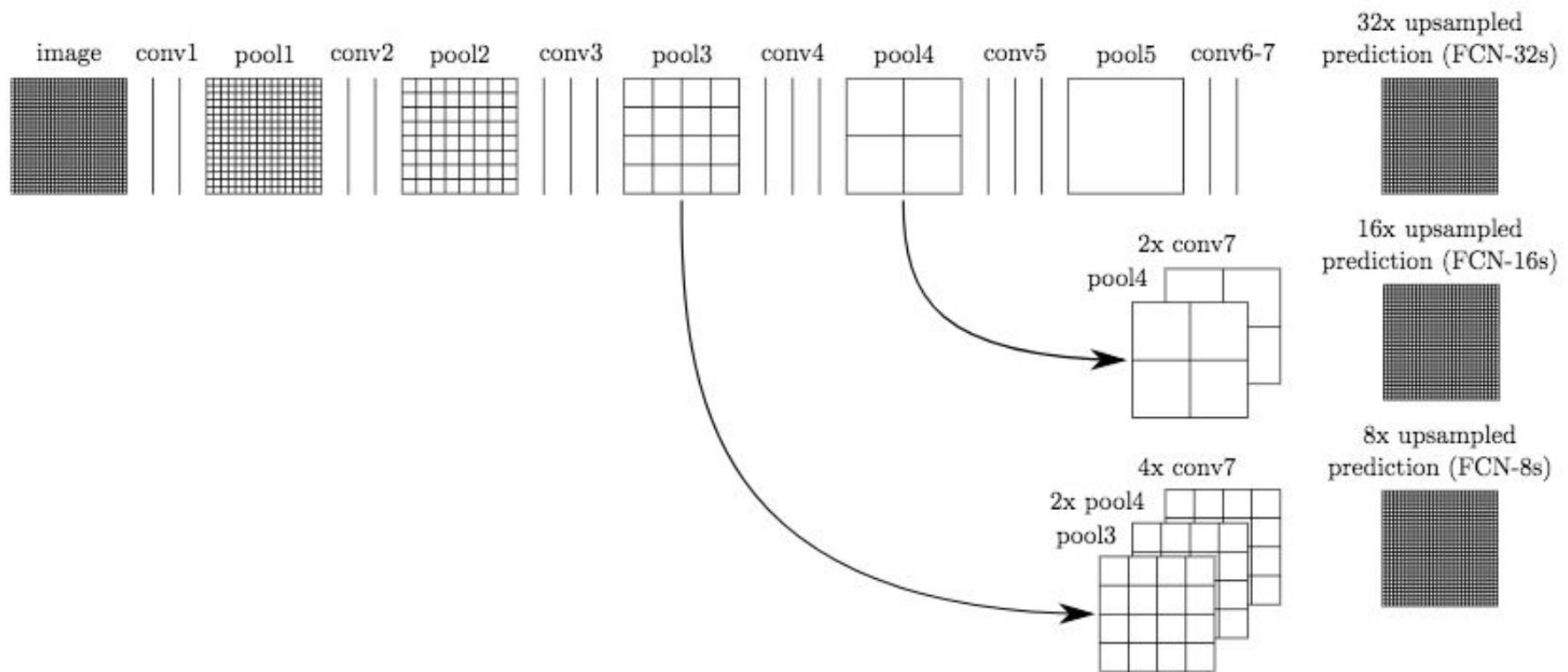


Image using resize upsampling

De-convolution for segmentation



De-convolution for segmentation



De-convolution for segmentation

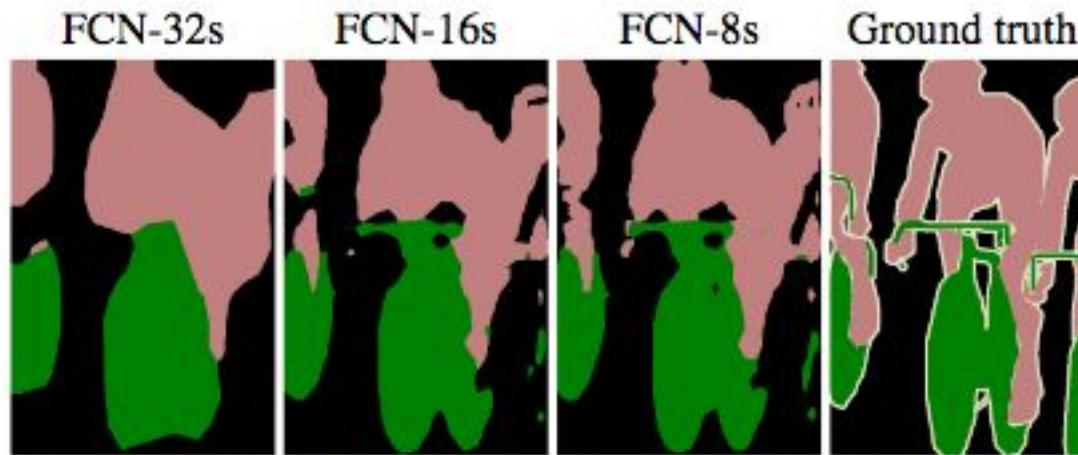


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

Other stuff to look for

Layers

Dilated convolution

<https://towardsdatascience.com/understanding-2d-dilated-convolution-operation-with-examples-in-numpy-and-tensorflow-with-d376b3972b25>

Mobile inverted BottleNeck (inverted linear residual)

<https://arxiv.org/abs/1801.04381>

Backbones

Feature Pyramid Networks (multi-scale backbone)

<https://arxiv.org/abs/1612.03144>

SqueezeNet (small network) <https://arxiv.org/abs/1602.07360>

EfficientNet <https://arxiv.org/abs/1905.11946>

CNN Summary

Building blocks

Matched filters and pooling

Filter factorization

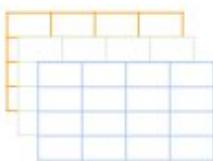
$5 \times 5 \rightarrow$ two 3×3

depthwise vs matrix factorization

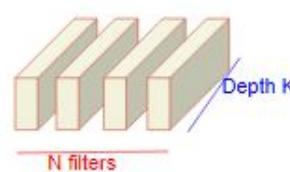
1×1 convolution

Deconvolution and upsampling

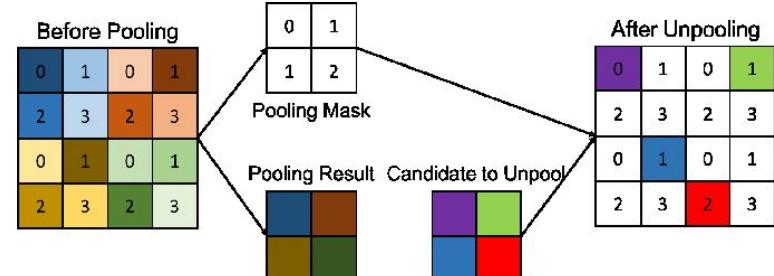
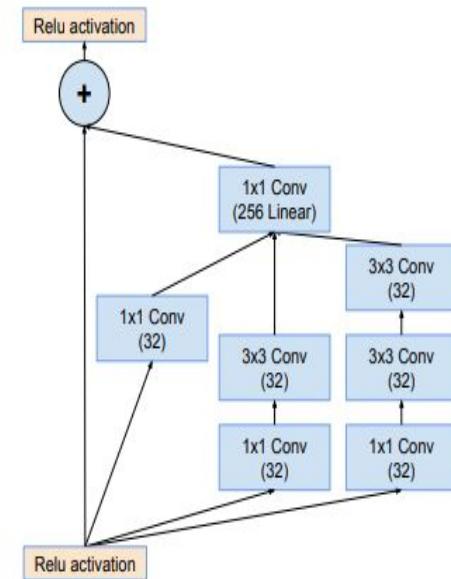
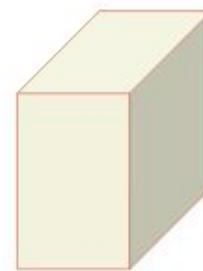
Output from depthwise convolution



1×1 filters



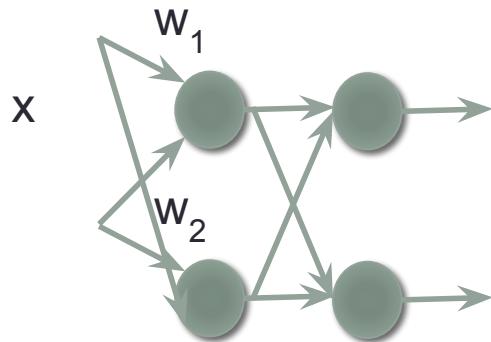
Final output



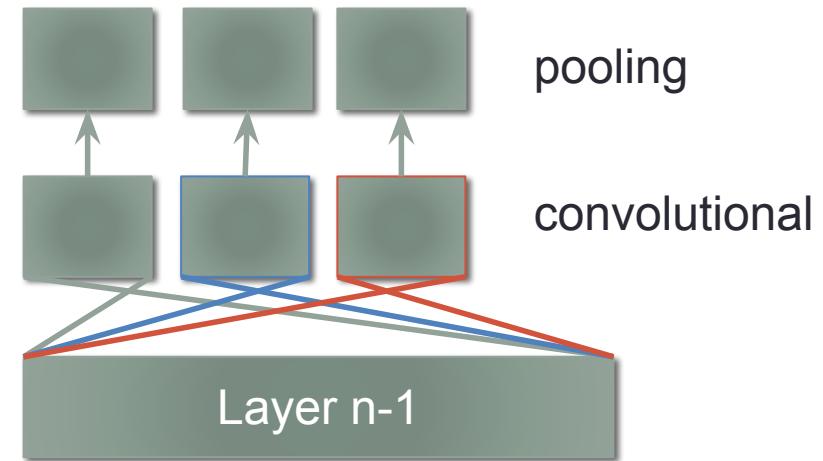
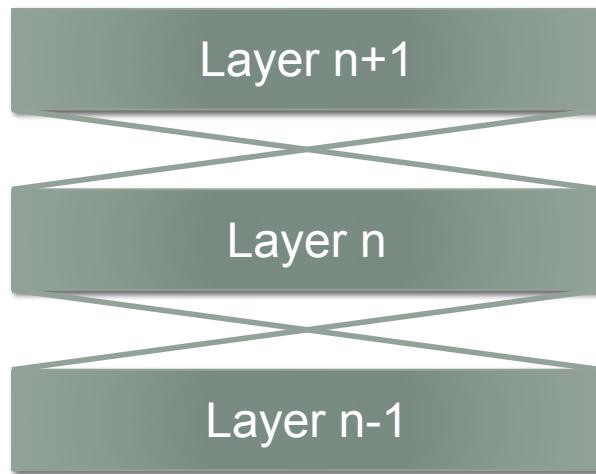
Parameter sharing in convolution neural networks

- $W^T x$

- CNN shares parameters in space

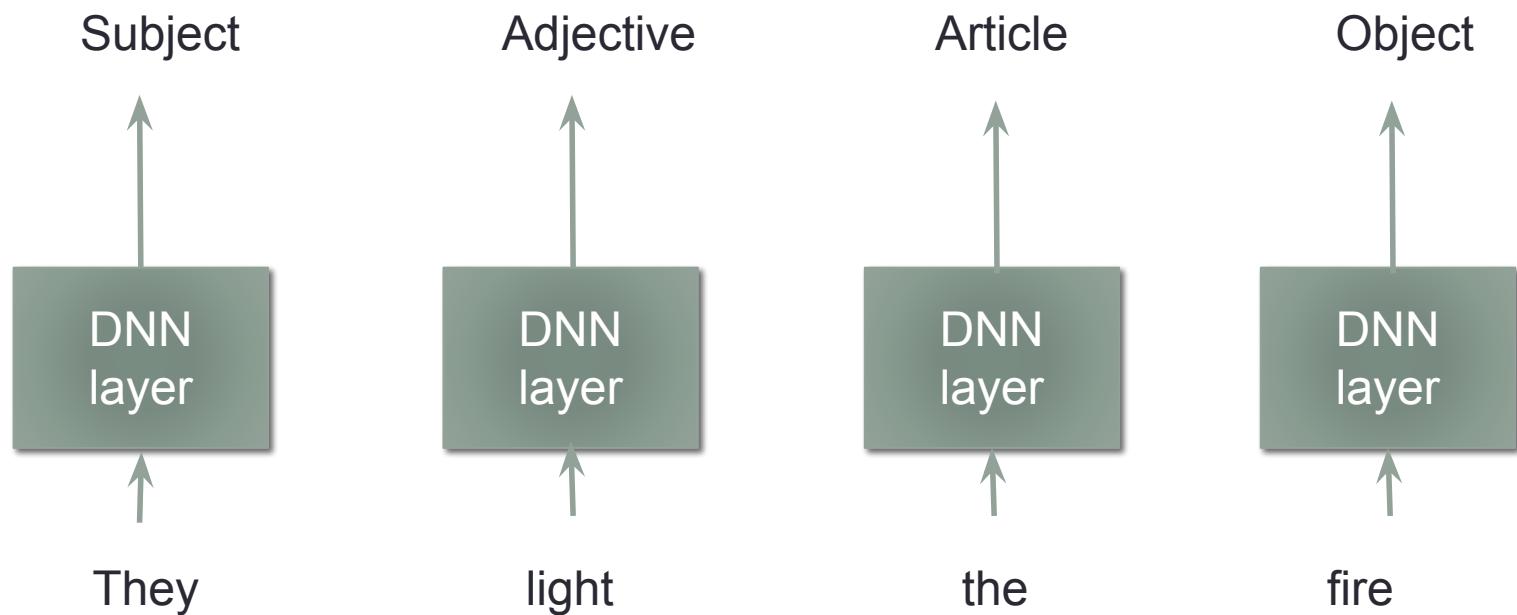


- What about sharing parameters in time?



Recurrent neural network (RNN)

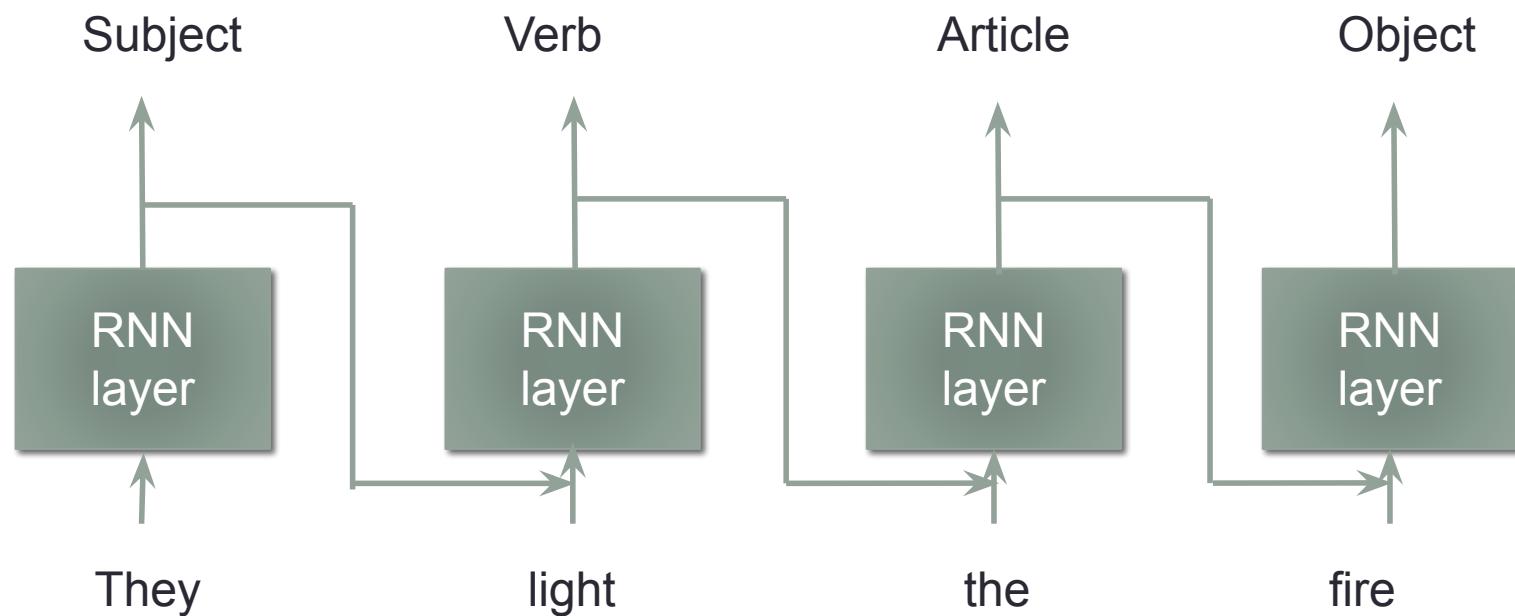
- DNN framework



Problem 1: need a way to remember the past

Recurrent neural network (RNN)

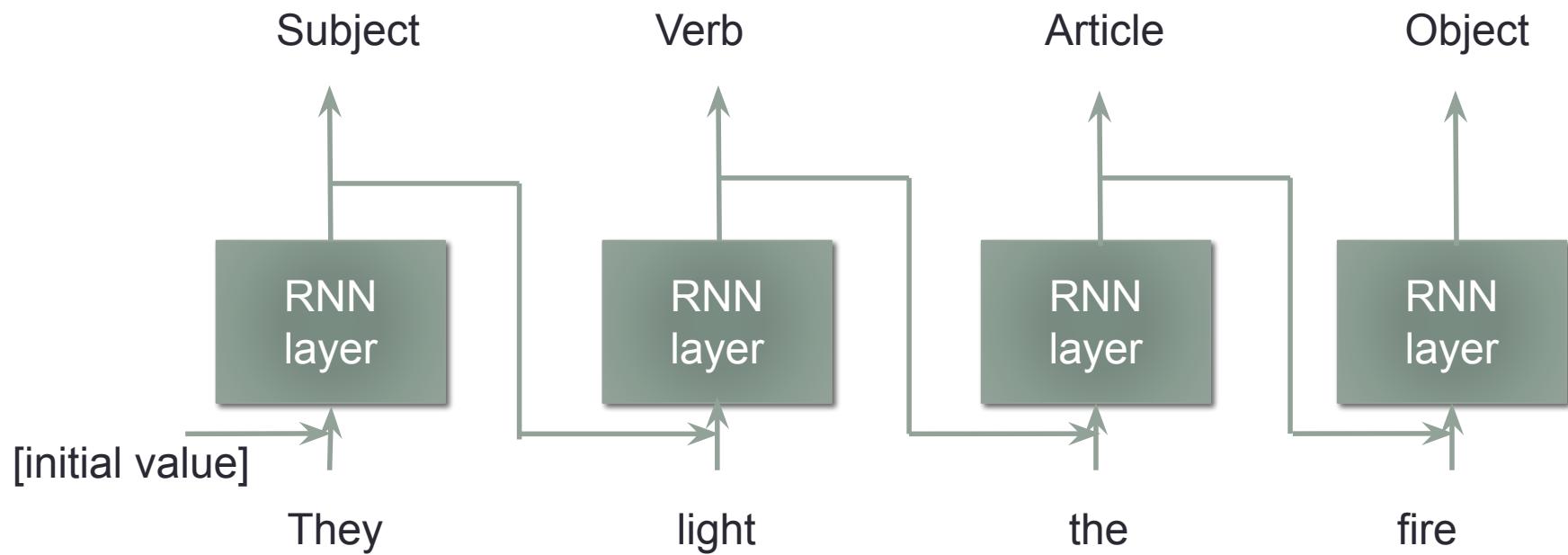
- RNN framework



Output of the layer encodes something meaningful about the past

Recurrent neural network (RNN)

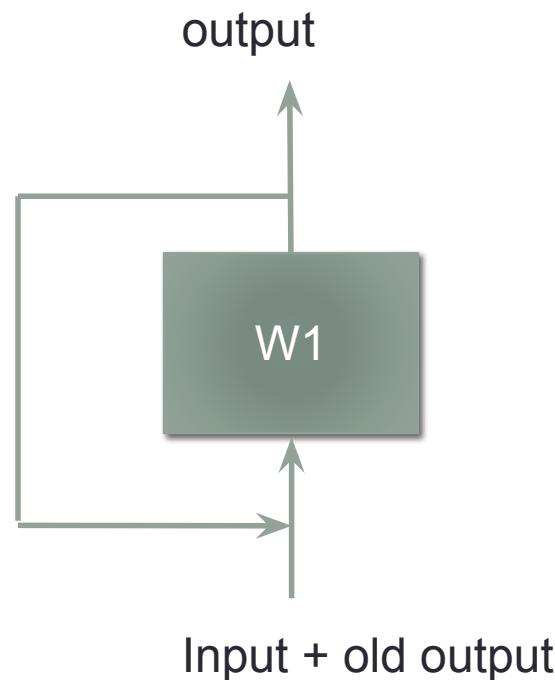
- RNN framework



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

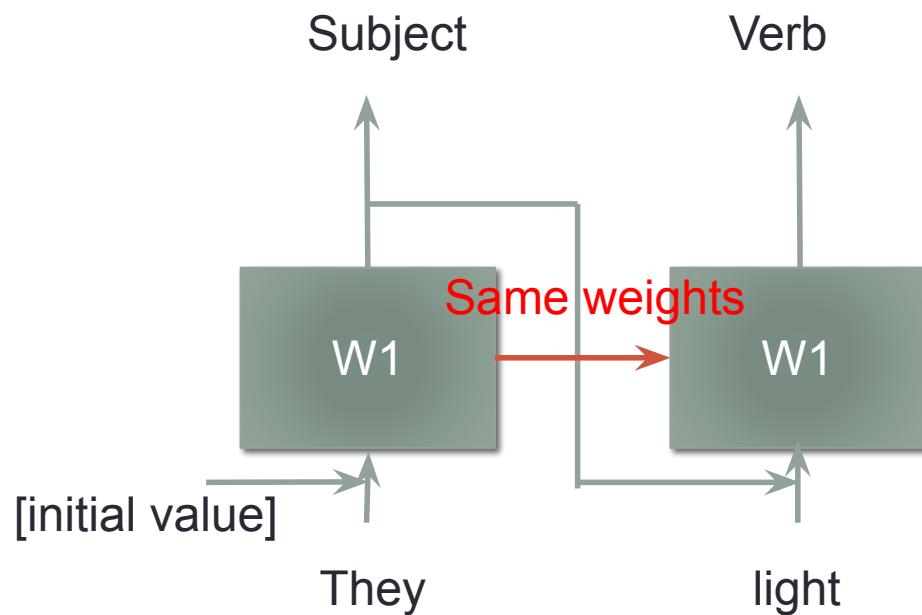
Recurrent neural network (RNN)

- Unrolling of a recurrent layer.



Recurrent neural network (RNN)

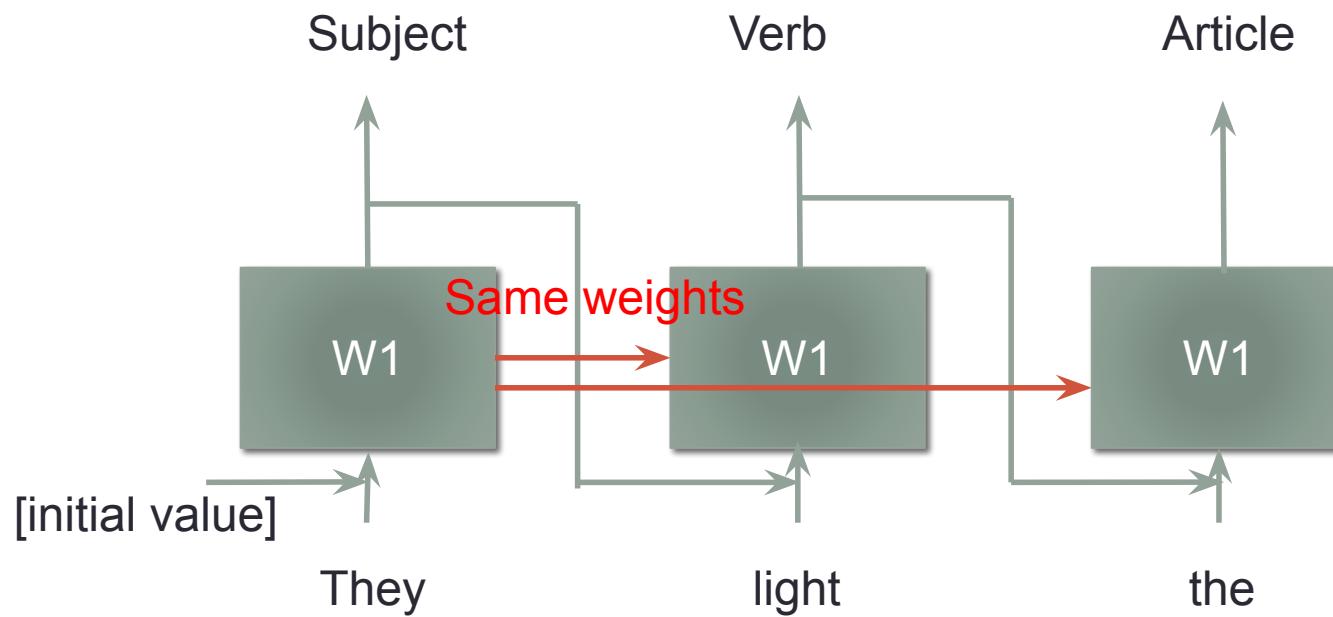
- Unrolling of a recurrent layer.



Parameter sharing across time

Recurrent neural network (RNN)

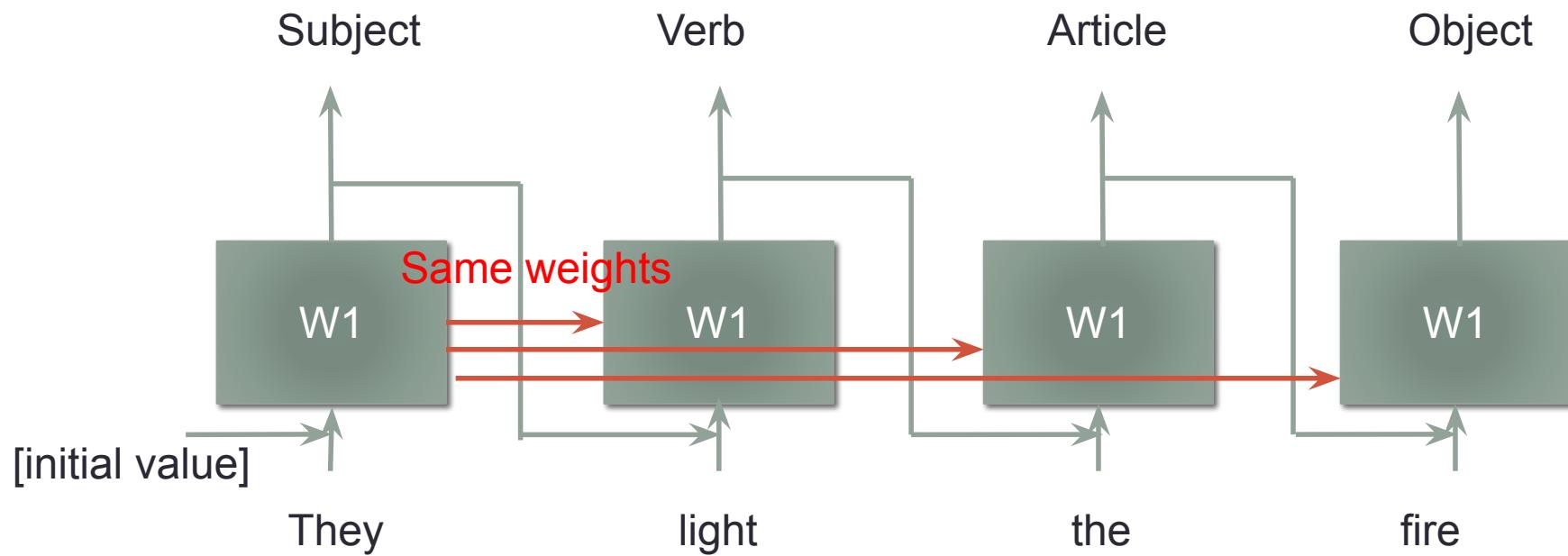
- Unrolling of a recurrent layer.



Parameter sharing across time

Recurrent neural network (RNN)

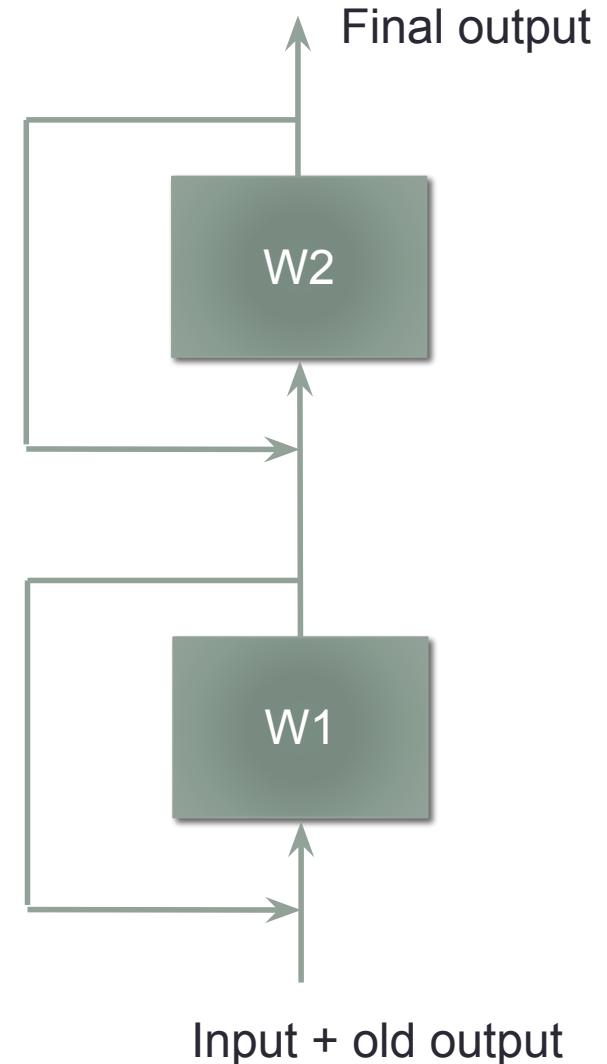
- Unrolling of a recurrent layer.



Parameter sharing across time

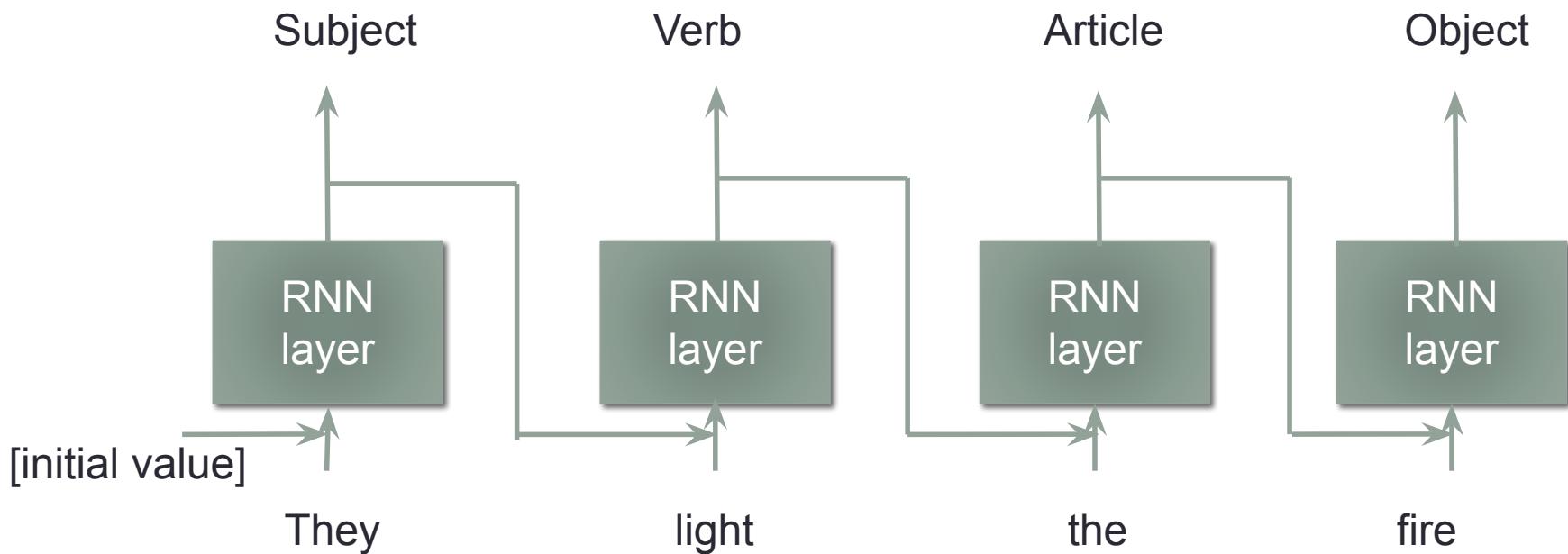
Recurrent neural network (RNN)

- Stacks of recurrent layer



Training a recurrent neural network

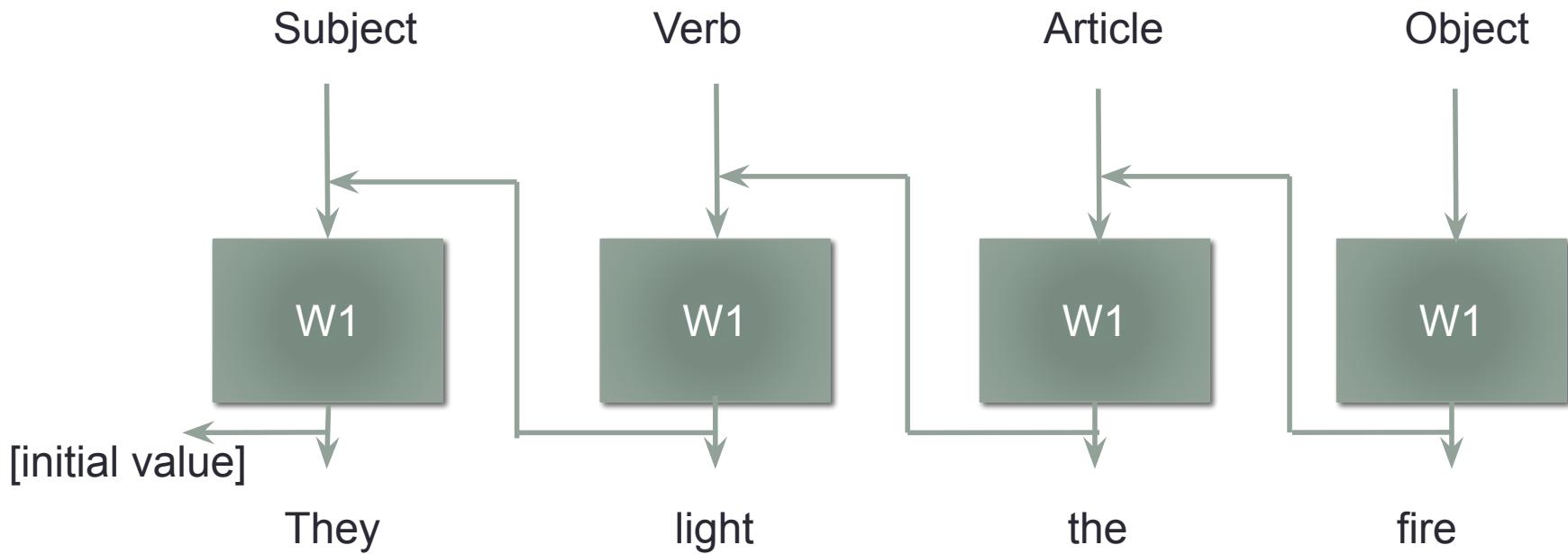
- RNN framework



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

Training a recurrent neural network

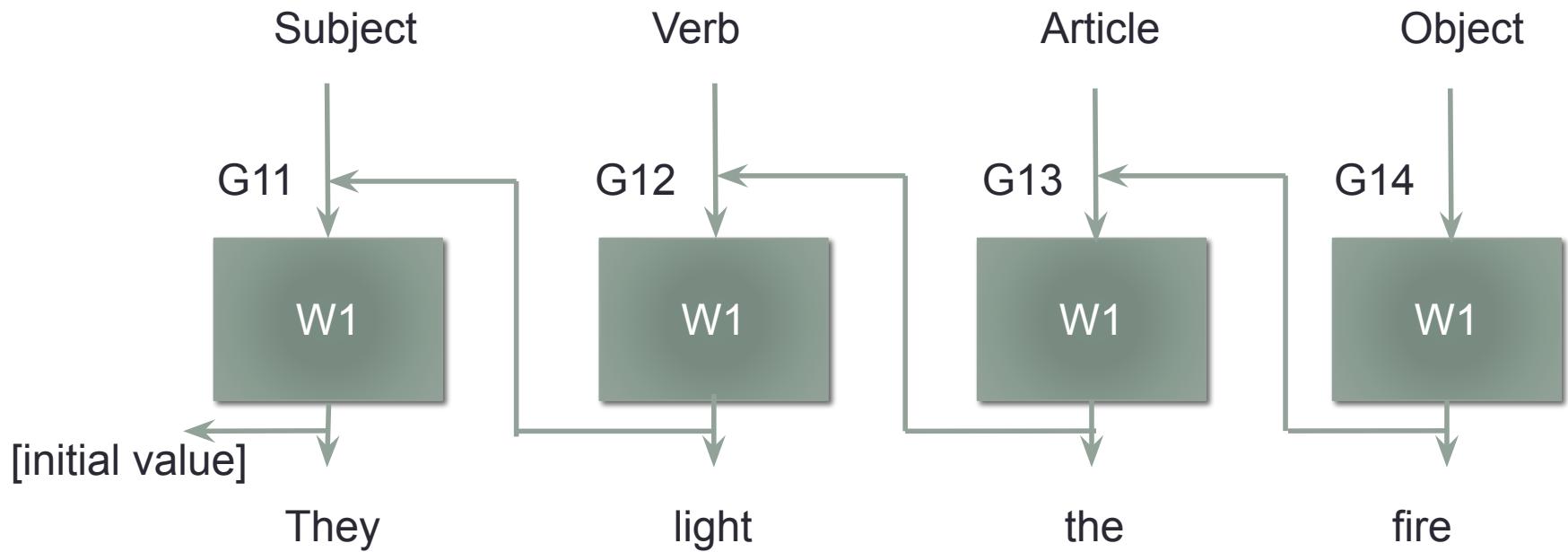
- Backward Computation graph



Backpropagation through time (BPTT)

BPTT

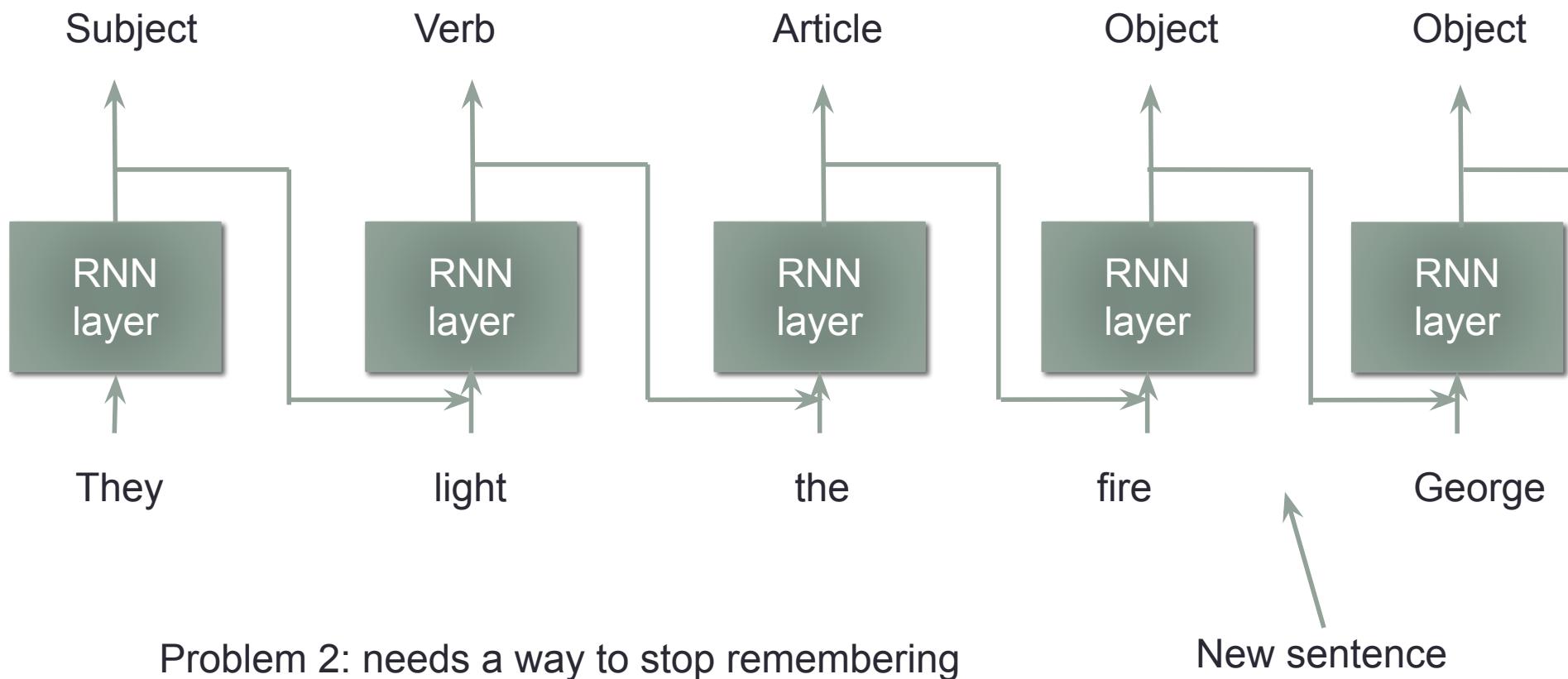
- Backward Computation graph



$$W_1 \leftarrow W_1 + G_{11} + G_{12} + G_{13} + G_{14}$$

Problem 1: cannot deal with infinitely long recurrent
Gradient explosion, vanishing gradient

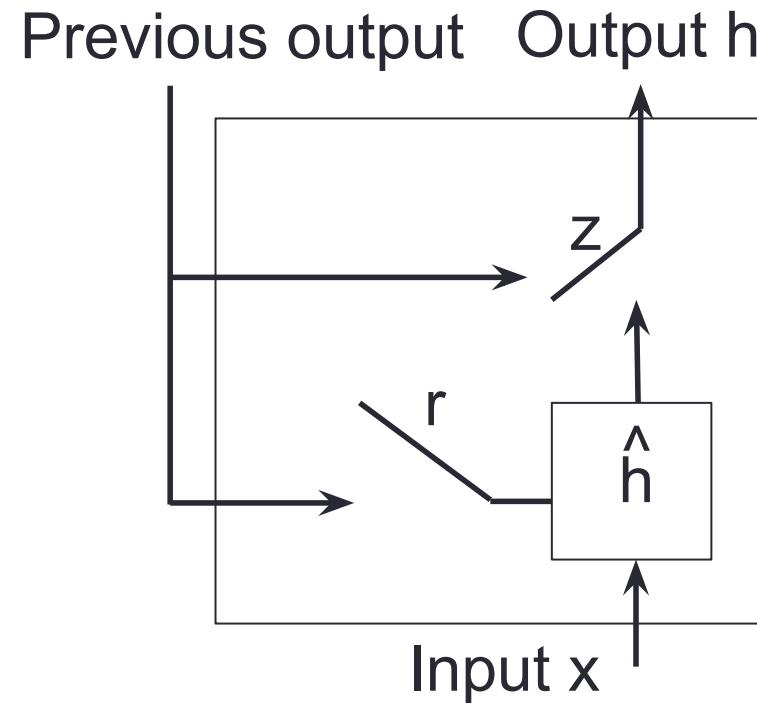
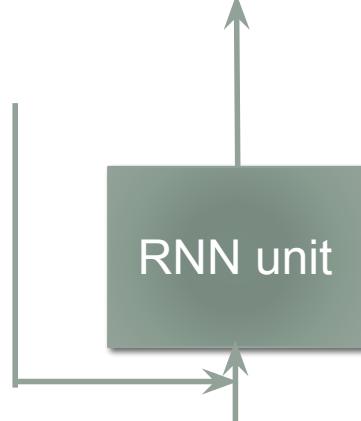
Recurrent neural network (RNN)



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

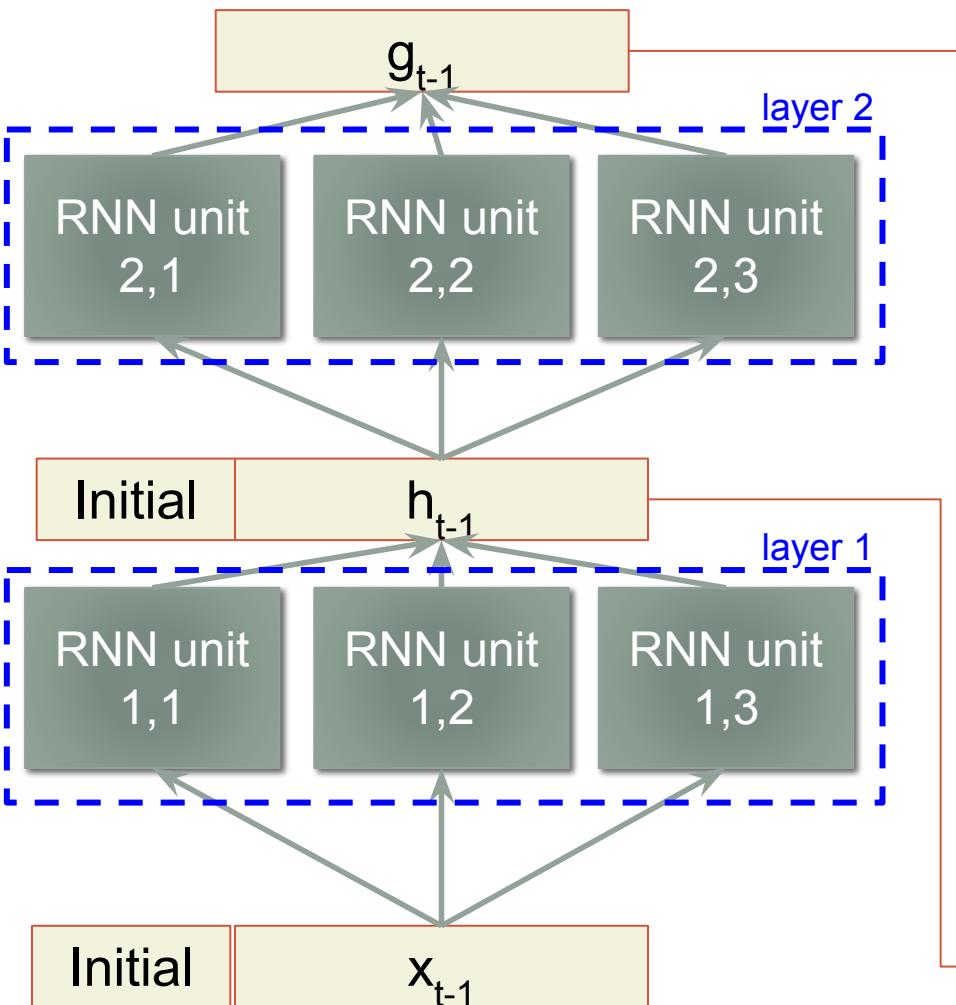
Gated Recurrent Unit (GRU)

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

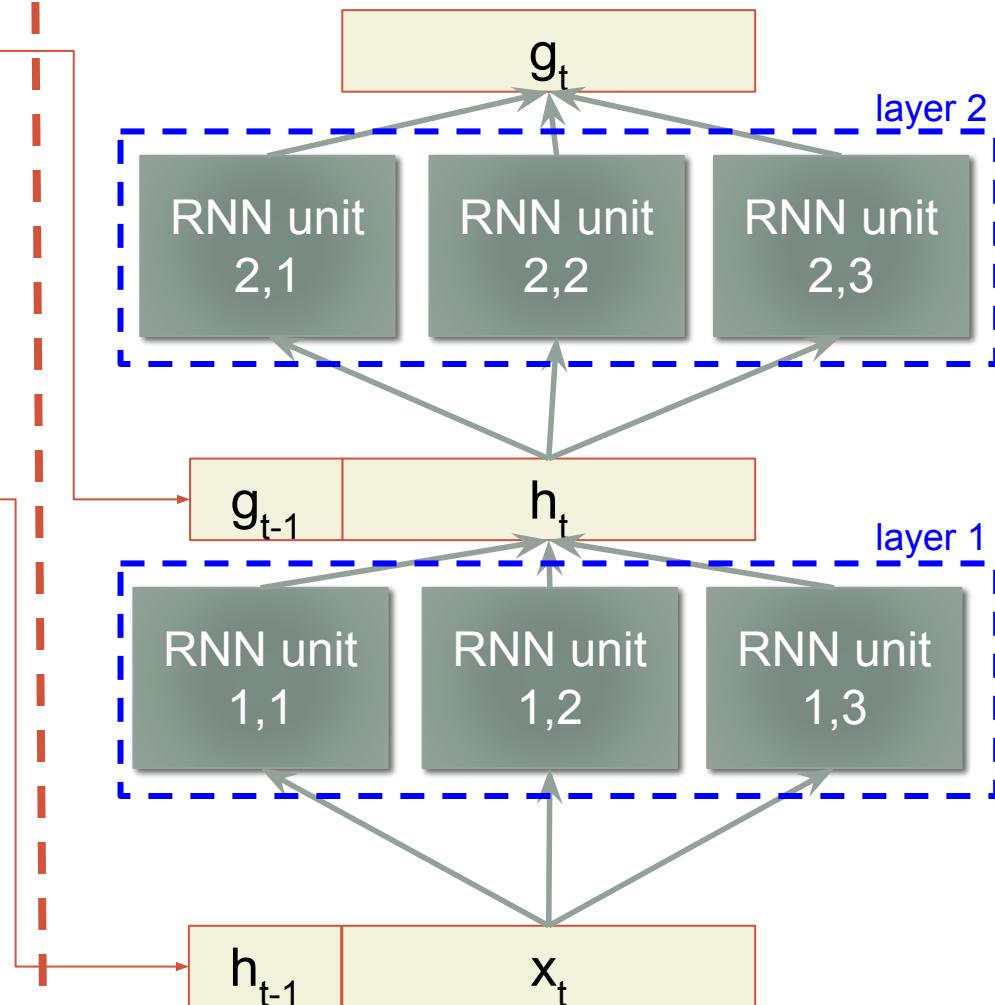


Gated Recurrent Unit (GRU) layer

Time step t-1

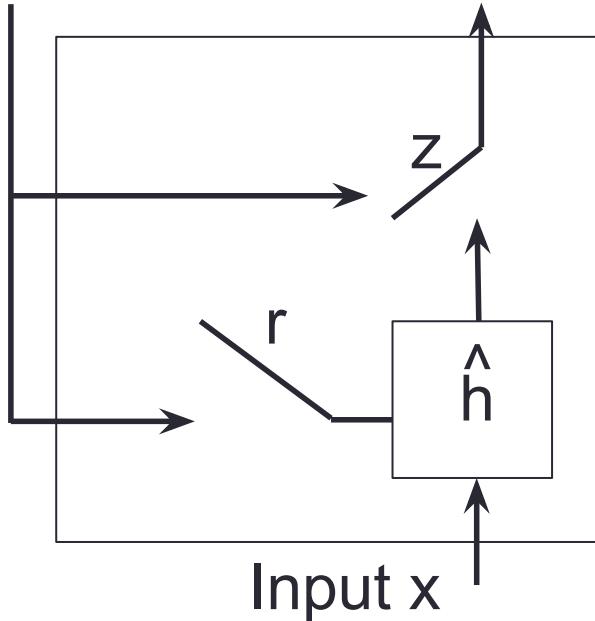


Time step t



Gated Recurrent Unit (GRU)

Previous output Output h

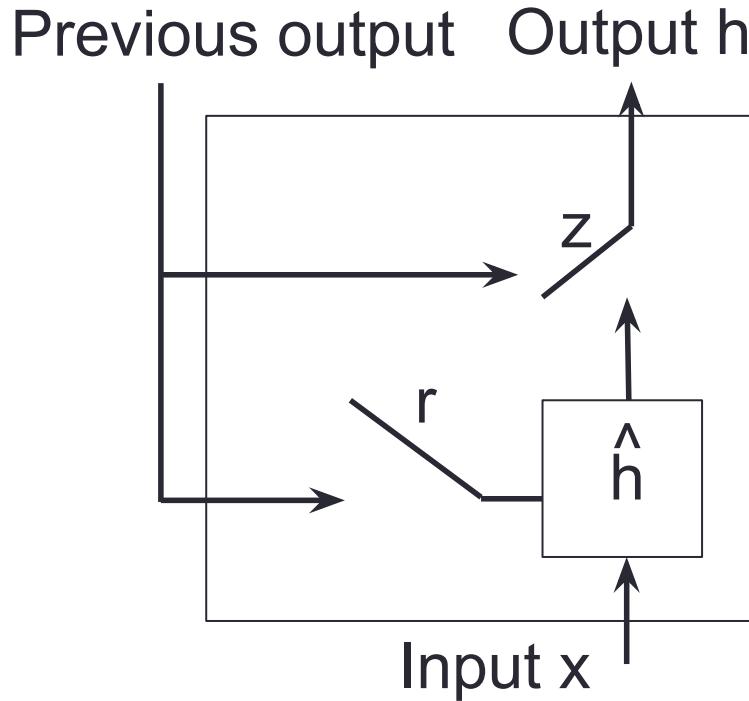


Neuron index

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

time index

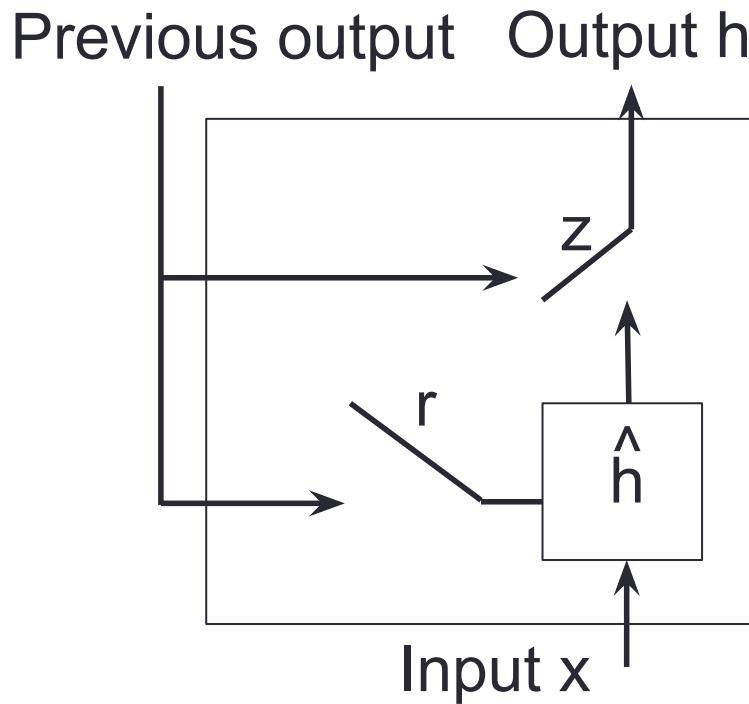
Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

One GRU neuron output (scalar)

Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

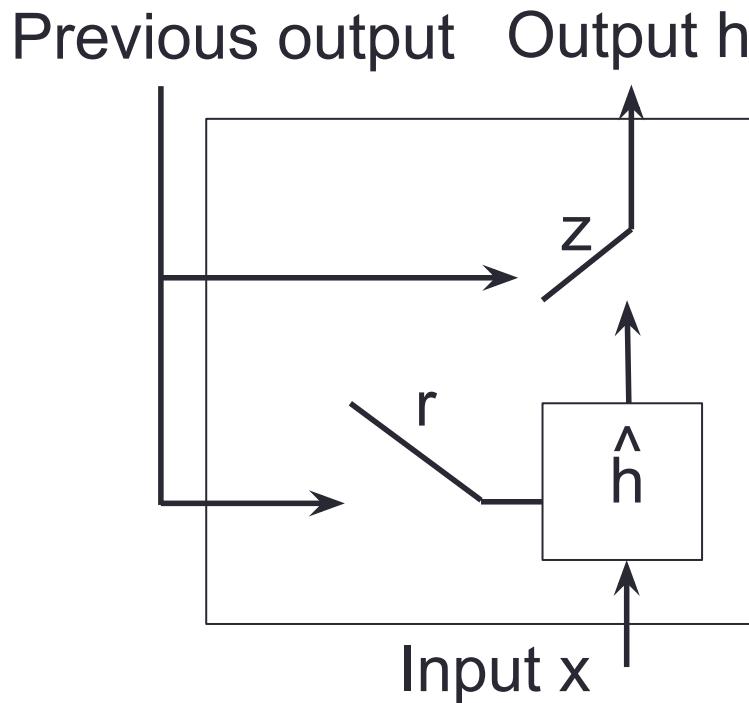
$$\hat{h}_t^j = \tanh^j (W \boxed{\mathbf{x}_t} + U (\boxed{\mathbf{r}_t} \odot \underline{\mathbf{h}_{t-1}}))$$

Linear transform with matrix multiply

Vector (each value from each GRU unit in the previous layer)

$$\mathbf{x}_t^j = \mathbf{h}_t^j$$

Gated Recurrent Unit (GRU)



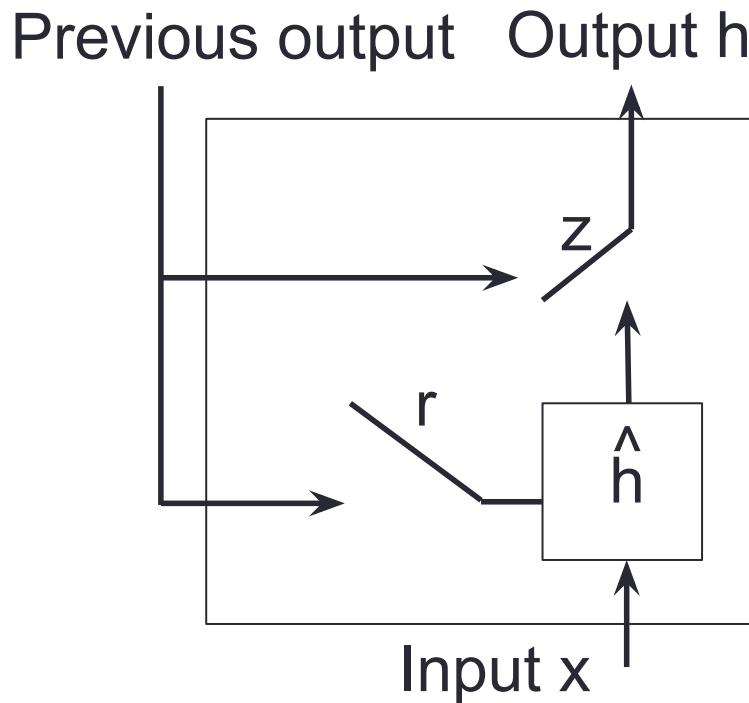
$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \underline{\tanh^j}(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

Takes the j-th element

Bounds the output

Gated Recurrent Unit (GRU)



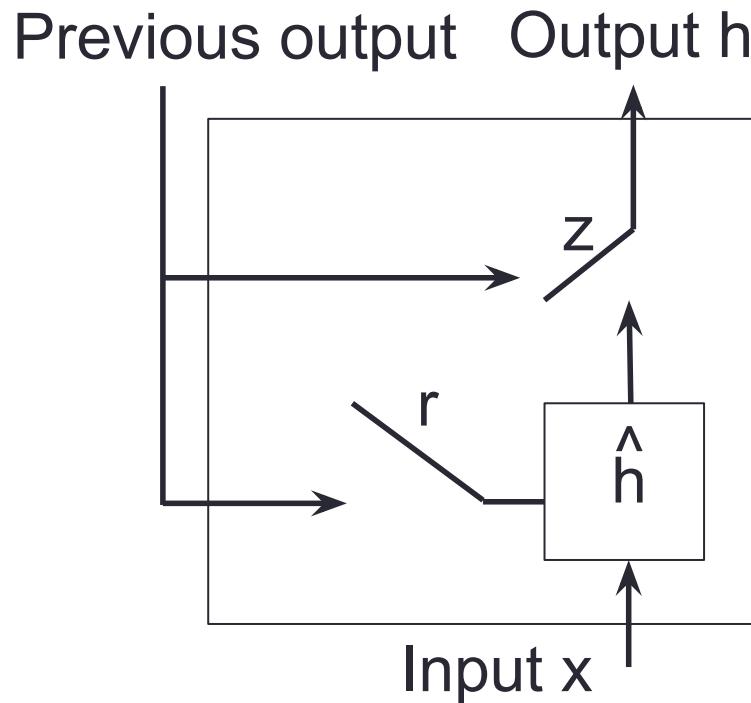
$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = \text{sigmoid}^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

Indicates a different set of weights

Gated Recurrent Unit (GRU)



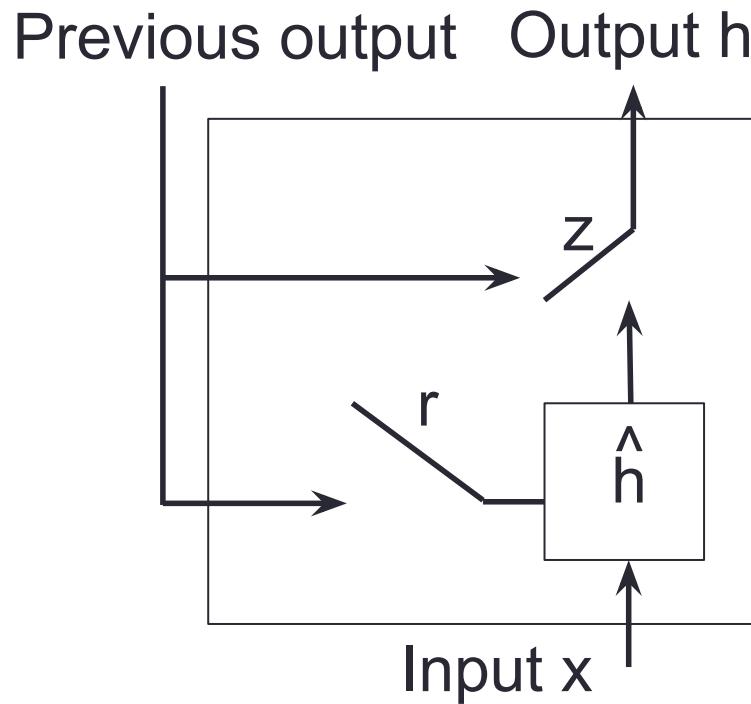
$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

$$z_t^j = \text{sigmoid}^j(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1})$$

Bounds the output to 0 to 1 for interpolation

Gated Recurrent Unit (GRU)



$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \hat{h}_t^j$$

$$\hat{h}_t^j = \tanh^j(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))$$

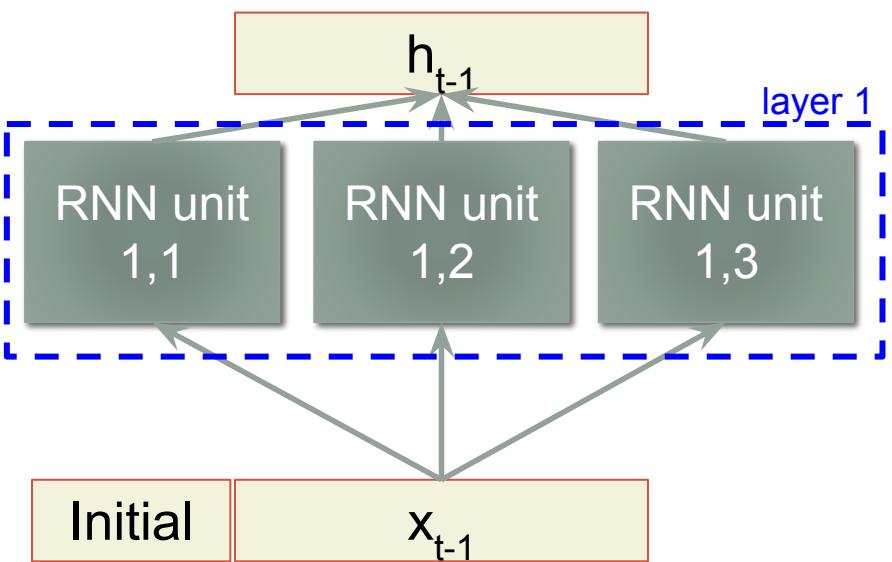
$$z_t^j = \text{sigmoid}^j(W_z\mathbf{x}_t + U_z\mathbf{h}_{t-1})$$

$$r_t^j = \text{sigmoid}^j(W_r\mathbf{x}_t + U_r\mathbf{h}_{t-1})$$

Gated Recurrent Unit (GRU) layer

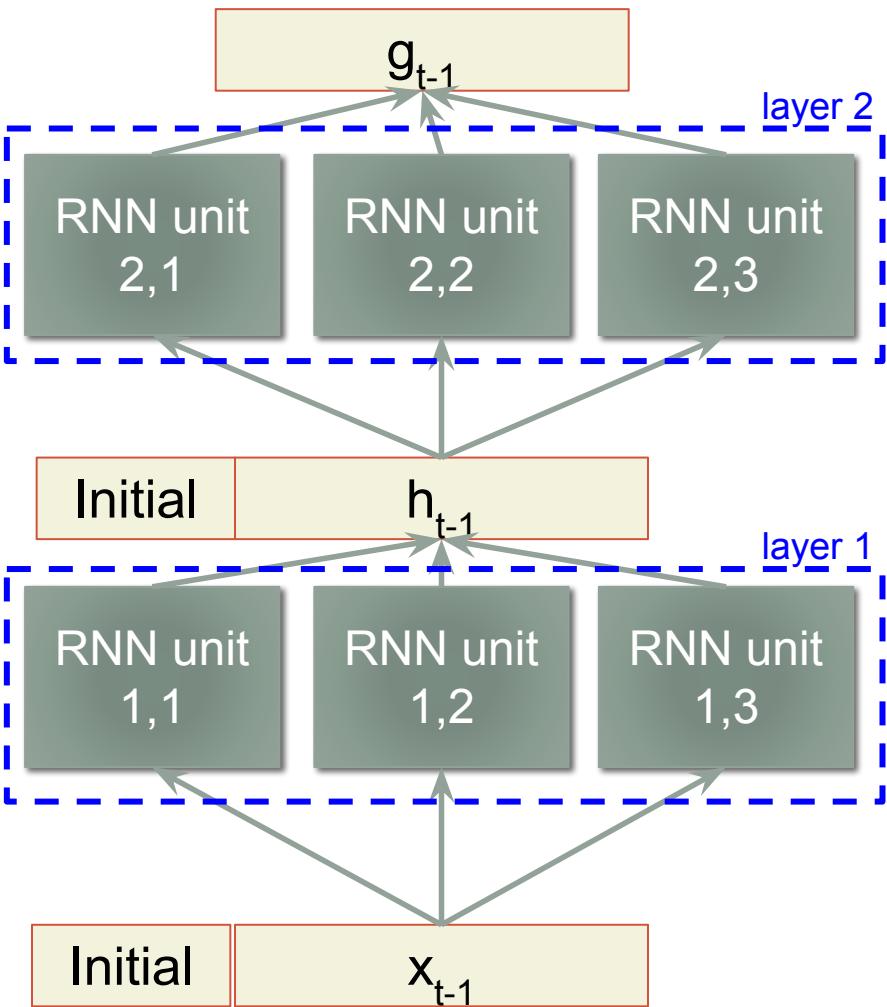
Time step 1

Time step 2



Gated Recurrent Unit (GRU) layer

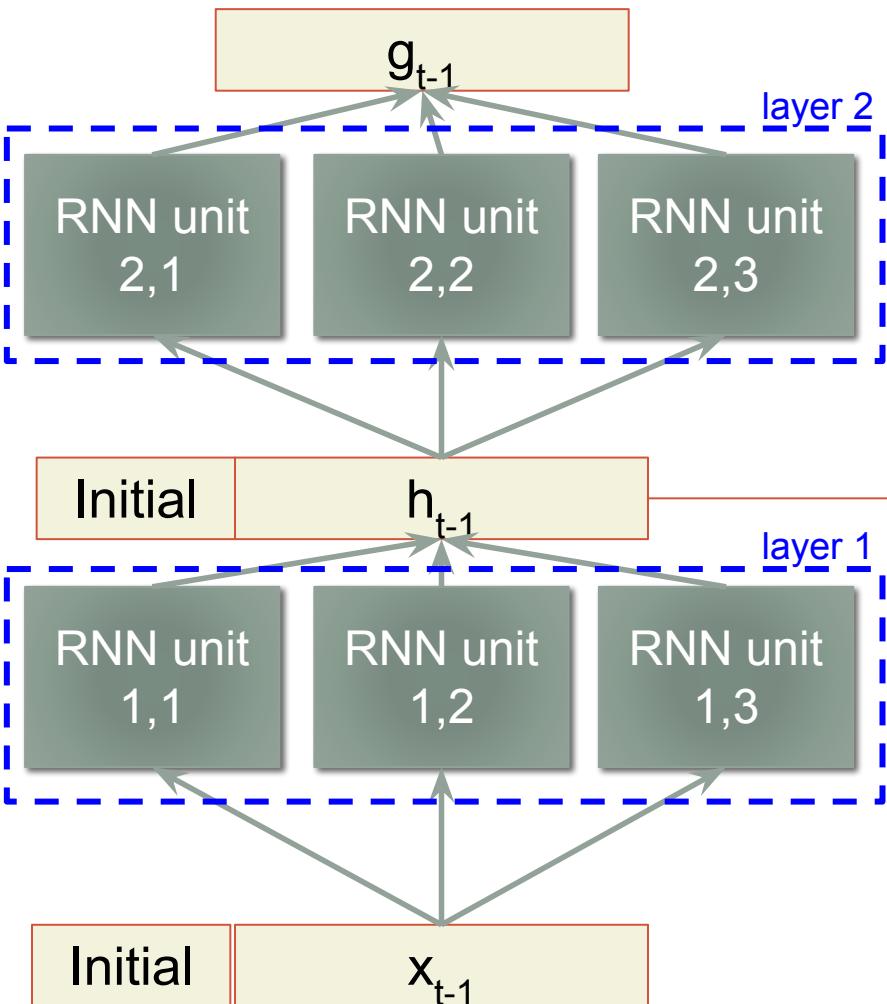
Time step 1



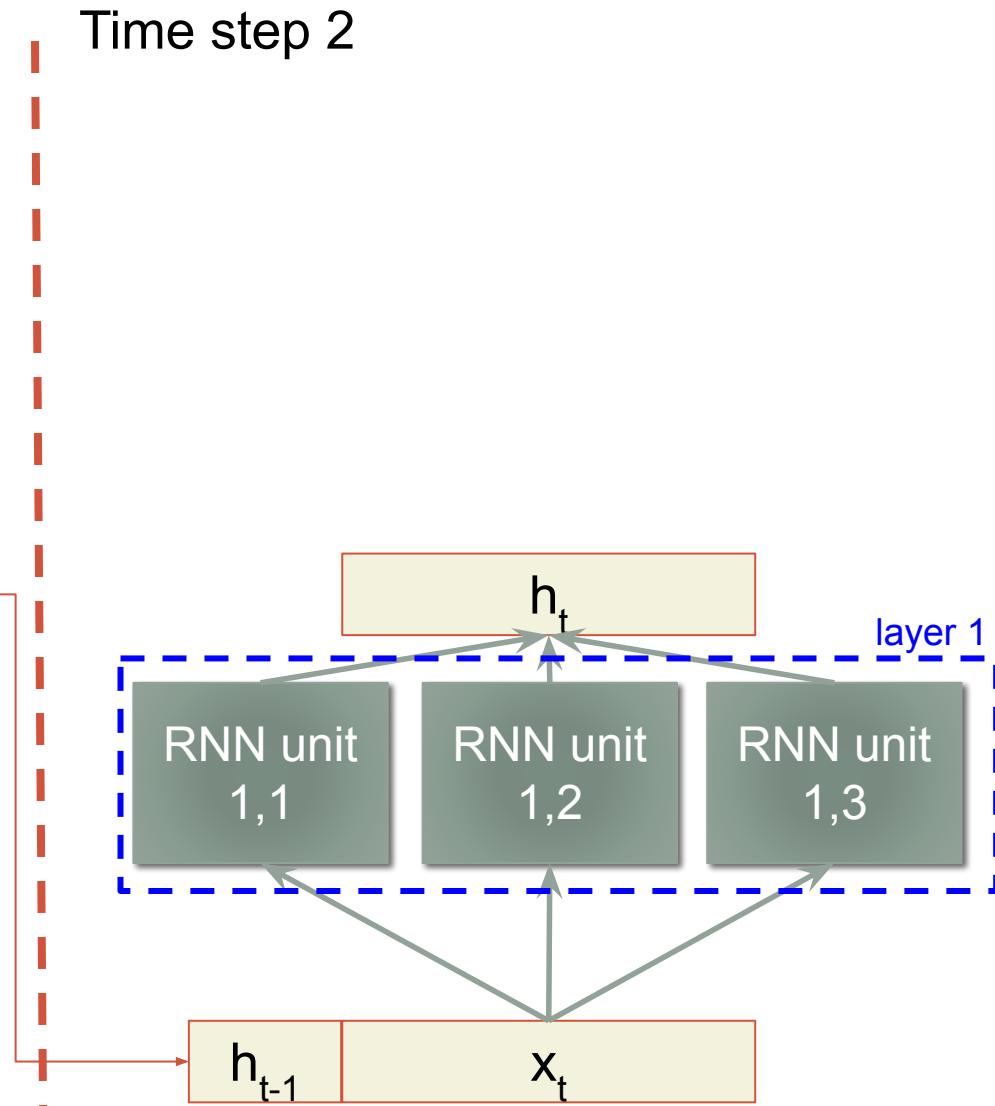
Time step 2

Gated Recurrent Unit (GRU) layer

Time step 1

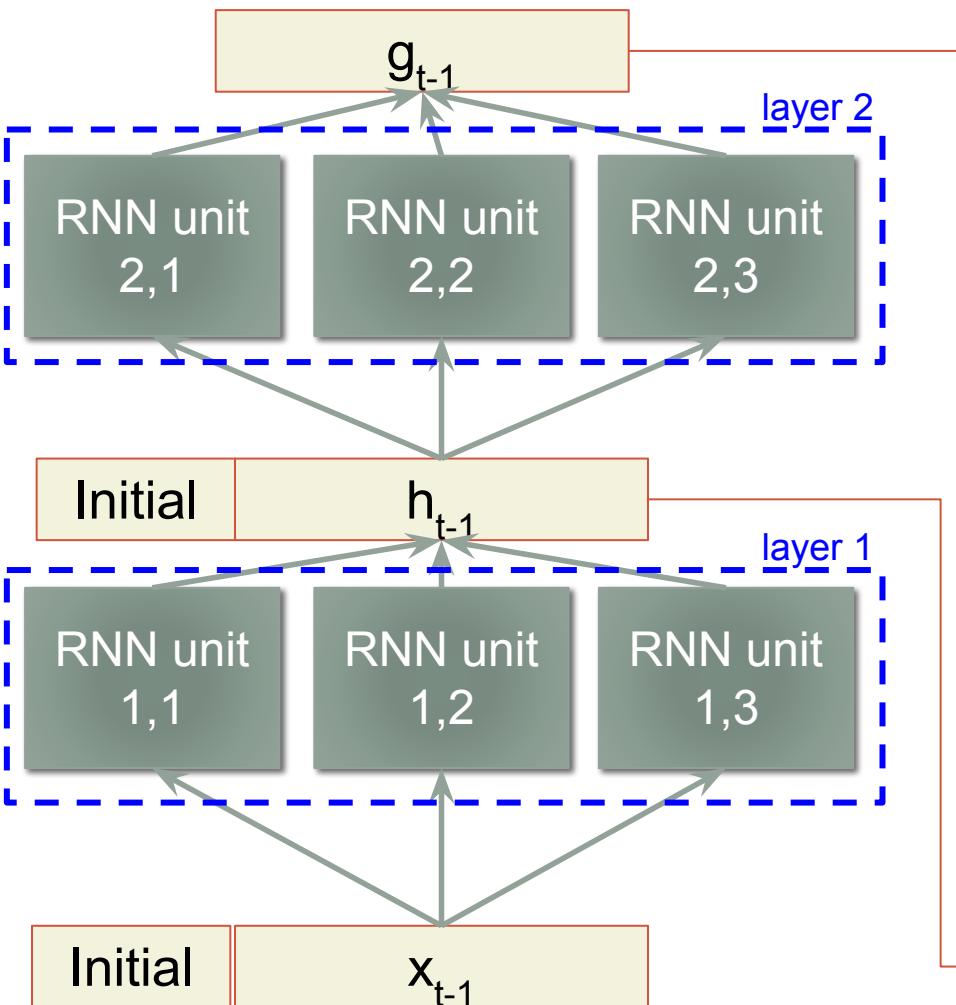


Time step 2

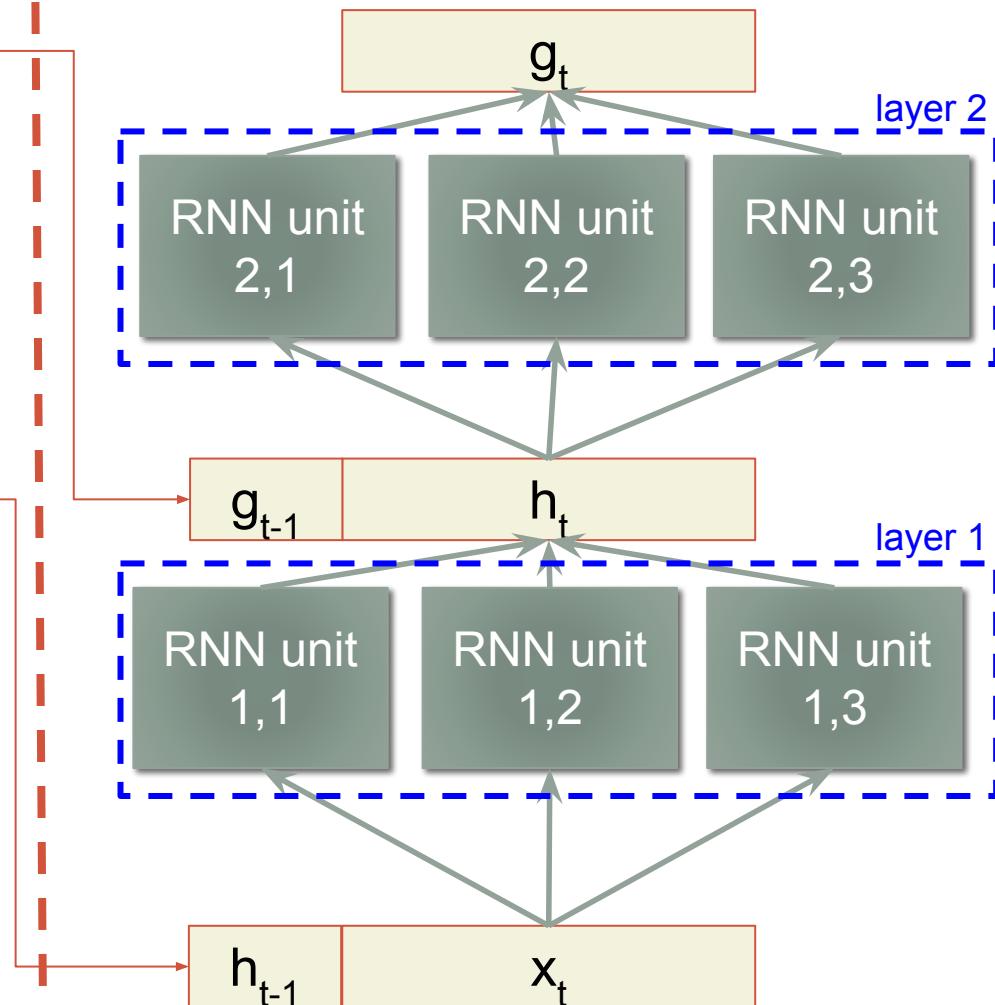


Gated Recurrent Unit (GRU) layer

Time step 1

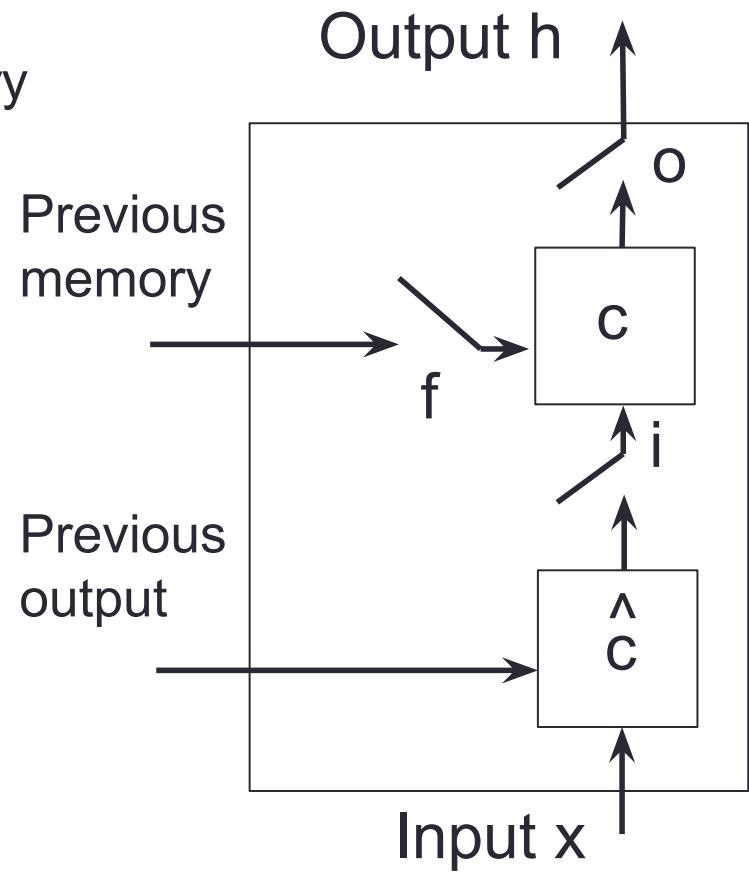
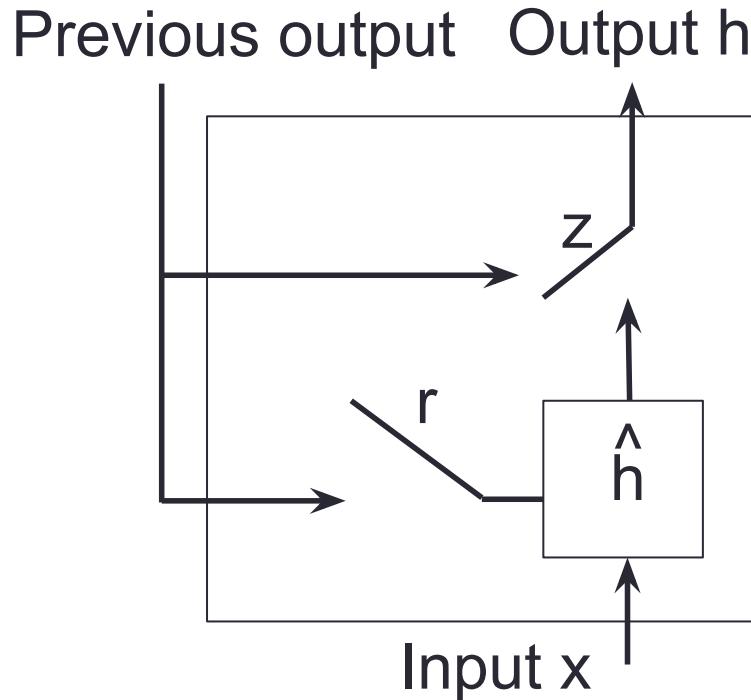


Time step 2

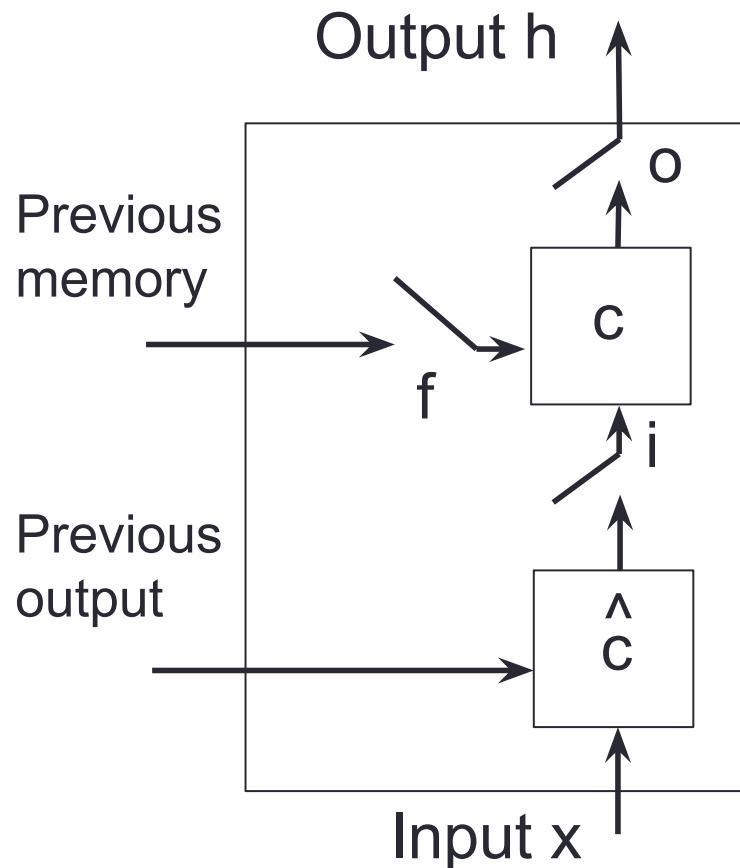


Long Short-Term Memory (LSTM)

- Have 3 gates, forget (f), input (i), output (o)
- Has an **explicit memory cell** (c)
 - Does not have to output the memory



Long Short-Term Memory (LSTM)

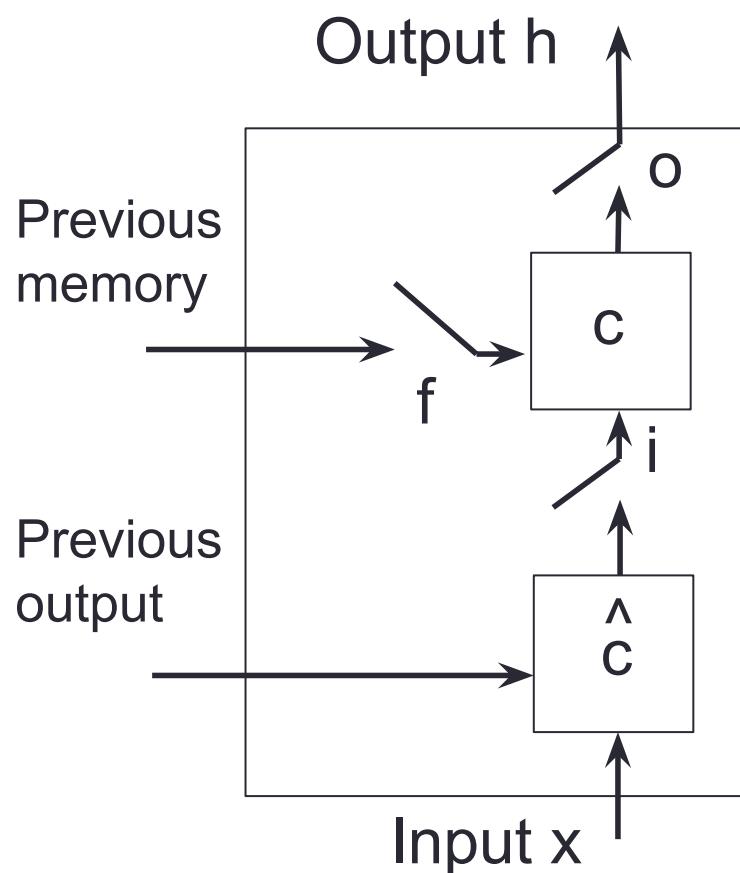


$$i_t^j = F^j(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + V_i \mathbf{c}_{t-1})$$
$$o_t^j = F^j(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t)$$
$$f_t^j = F^j(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_j \mathbf{c}_{t-1})$$

Contribution from memory “Peephole connection”

Vs are diagonal matrices(Each cell can only see its own memory)

Long Short-Term Memory (LSTM)



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

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

$$f_t^j = F^j(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_j \mathbf{c}_{t-1})$$

$$h_t^j = o_t^j \tanh(c_t^j)$$

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

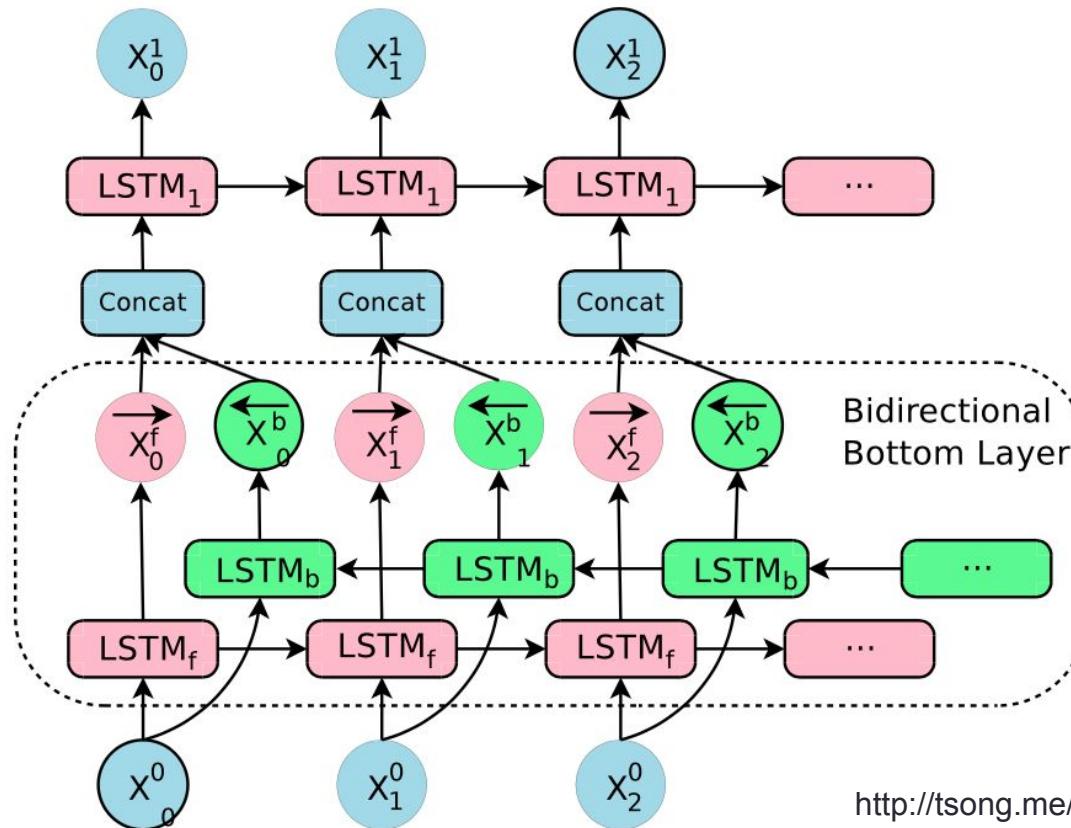
$$\hat{c}_t^j = \tanh^j(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1})$$

GRU vs LSTM

- GRU and LSTM offers the same performance with large dataset
 - GRU better for smaller dataset (less parameters)
 - GRU faster to train and faster runtime (smaller model)
- Use GRUs!

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



LSTM remembers meaningful things

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Embeddings

- A way to encode information to a lower dimensional space
 - PCA
 - We learn about this lower dimensional space through data

One hot encoding

- Categorical representation is usually represented by **one hot encoding**
- Categorical representations examples:
 - Words in a vocabulary, characters in Thai language

Apple -> 1 -> [1, 0, 0, 0, ...]

Bird -> 2 -> [0, 1, 0, 0, ...]

Cat -> 3 -> [0, 0, 1, 0, ...]

- **Sparse** representation
 - Sparse means most dimension are zero

One hot encoding

- Sparse – but lots of dimension
 - Curse of dimensionality
- Does not represent meaning.

Apple -> 1 -> [1, 0, 0, 0, ...]

Bird -> 2 -> [0, 1, 0, 0, ...]

Cat -> 3 -> [0, 0, 1, 0, ...]

$$|\text{Apple} - \text{Bird}| = |\text{Bird} - \text{Cat}|$$

Getting meaning into the feature vectors

- You can add back meanings by hand-crafted rules
- Old-school NLP is all about feature engineering
- Word segmentation example:
 - Cluster Numbers
 - Cluster letters
- Concatenate them
- 𠂇 = [0 0 0 0 1 0 0 0, 1, 0]
- 𠂈 = [0 0 0 1 0 0 0 0, 0, 1]
- 𠂉 = [1 0 0 0 0 0 0 0, 0, 2]
- Which rules to use?
 - Try as many as you can think of, and do feature selection or use models that can do feature selection

Dense representation

- We can encode sparse representation into a lower dimensional space
 - $F: \mathbb{R}^N \rightarrow \mathbb{R}^M$, where $N > M$

Apple -> 1 -> [1, 0, 0, 0, ...] -> [2.3, 1.2]

Bird -> 2 -> [0, 1, 0, 0, ...] -> [-1.0, 2.4]

Cat -> 3 -> [0, 0, 1, 0, ...] -> [-3.0, 4.0]

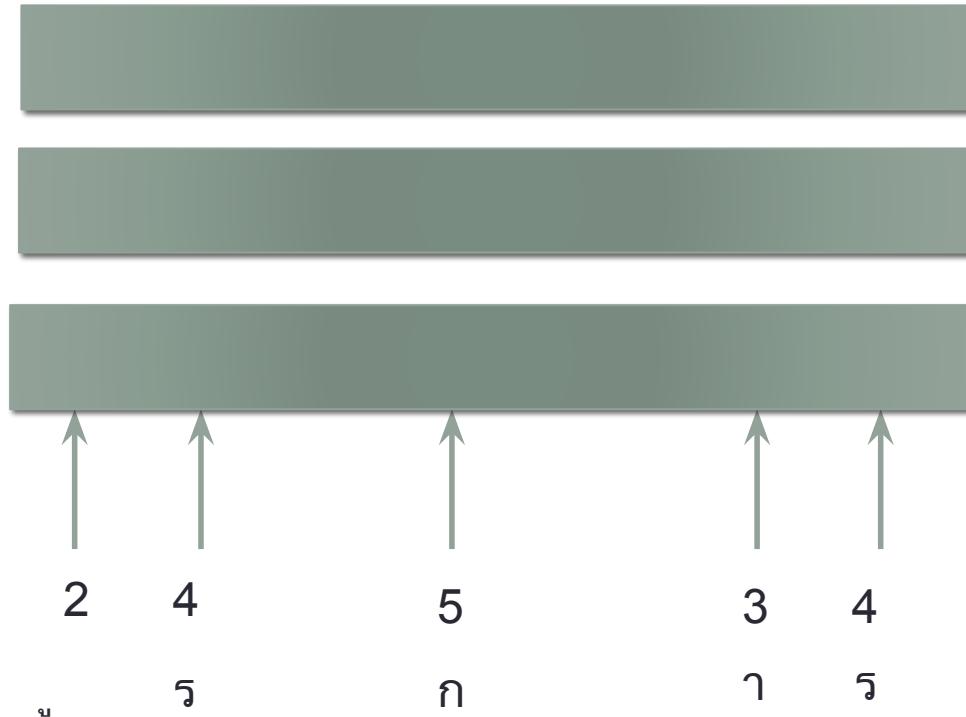
- We can do this by using an embedding layer

Word segmentation with fully connected networks

1 = word beginning, 0 = word middle



Logistic function



Adding embedding layer



Embedding layer
shares the same
weights



Parameter sharing!



[1, -1] [3, -2] [5.3, -2.1] [5.3, -3.1] [3, -2]



More on embeddings
in the next two
lectures!

2

4

5

3

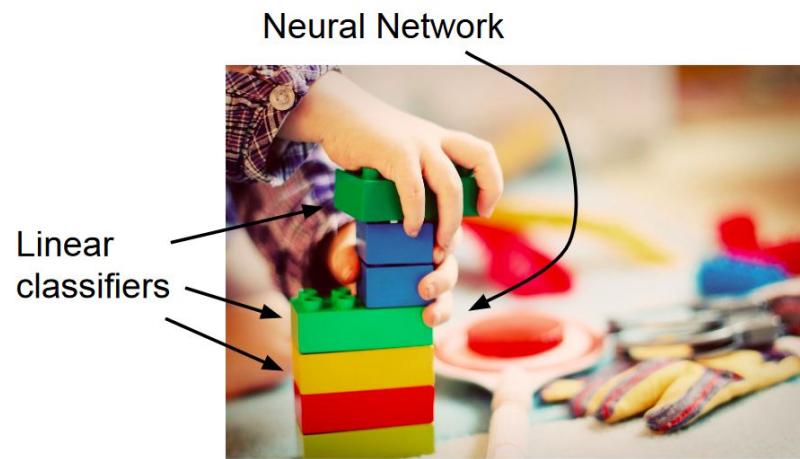
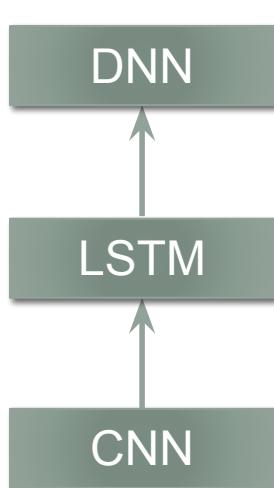
4

Embedding vs PCA

- PCA – unsupervised method
- Embedding – train supervised with the task
 - Embedding should be superior with the task
- PCA – linear
- Embedding – potentially non-linear
 - Should be more powerful
- You can learn an embedding on one task (with lots of data) and use it on another task
 - Embedding learns meaningful feature representations

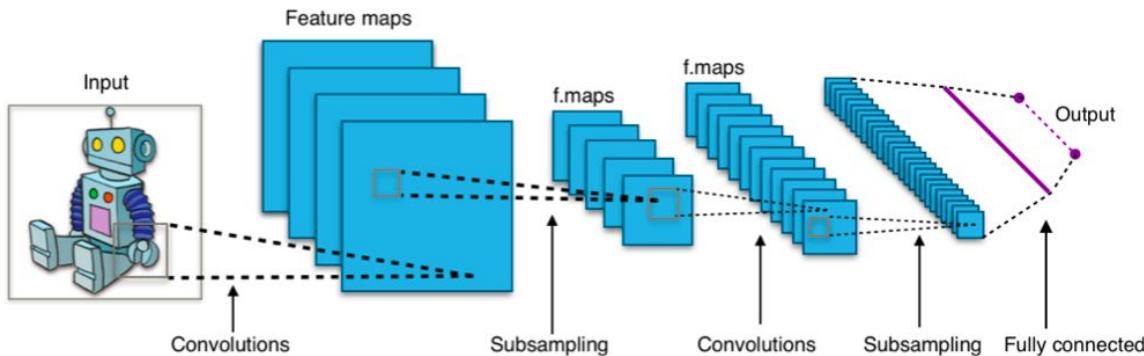
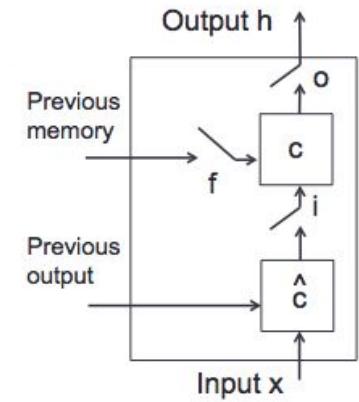
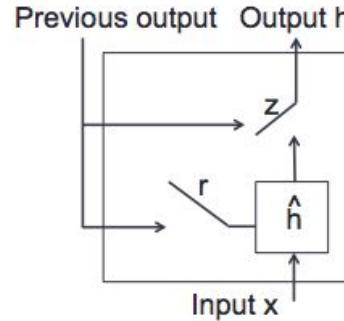
DNN Legos

- 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



Neural networks

- Fully connected networks
 - SGD, backprop
- CNN
- RNN, LSTM, GRU



Attention modeling
Object detection

← Next lecture