Convolutional Neural Networks

Multi-layer Neural Network

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

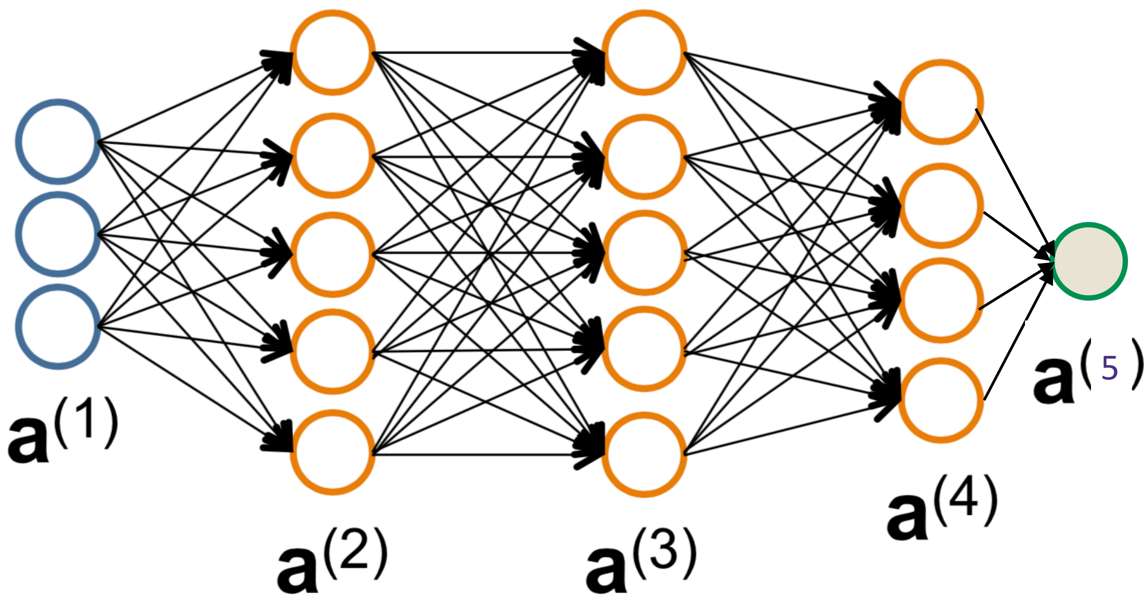
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$



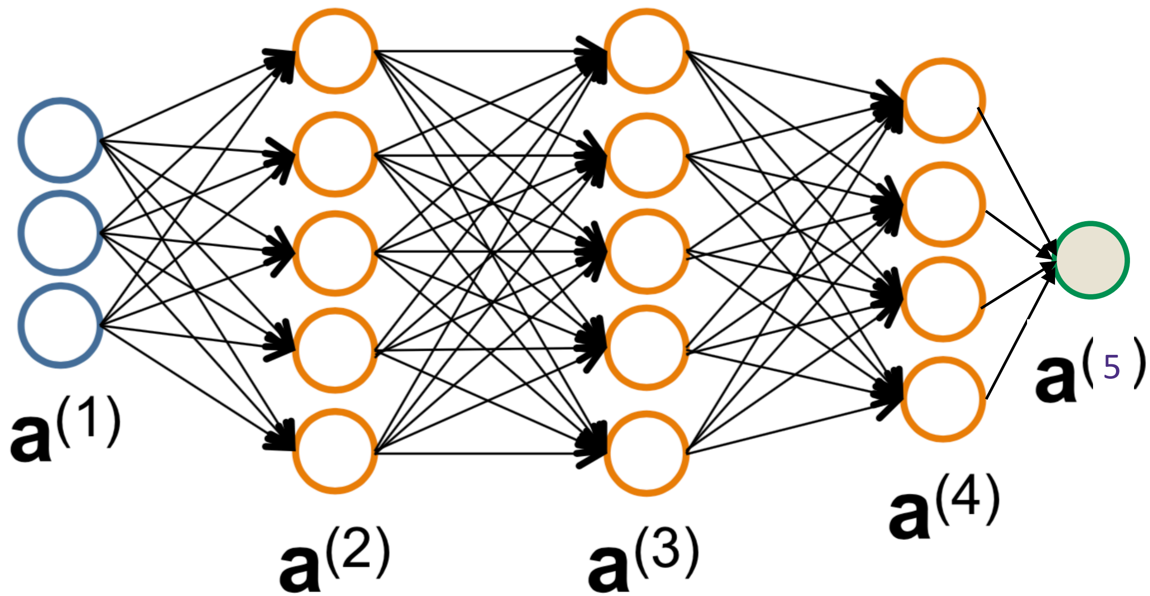
$$L(y, \hat{y}) = y \log(\hat{y}) + (1-y)\log(1-\hat{y})$$

$$g(z) = \frac{1}{1+e^{-z}}$$

Binary Logistic Regression

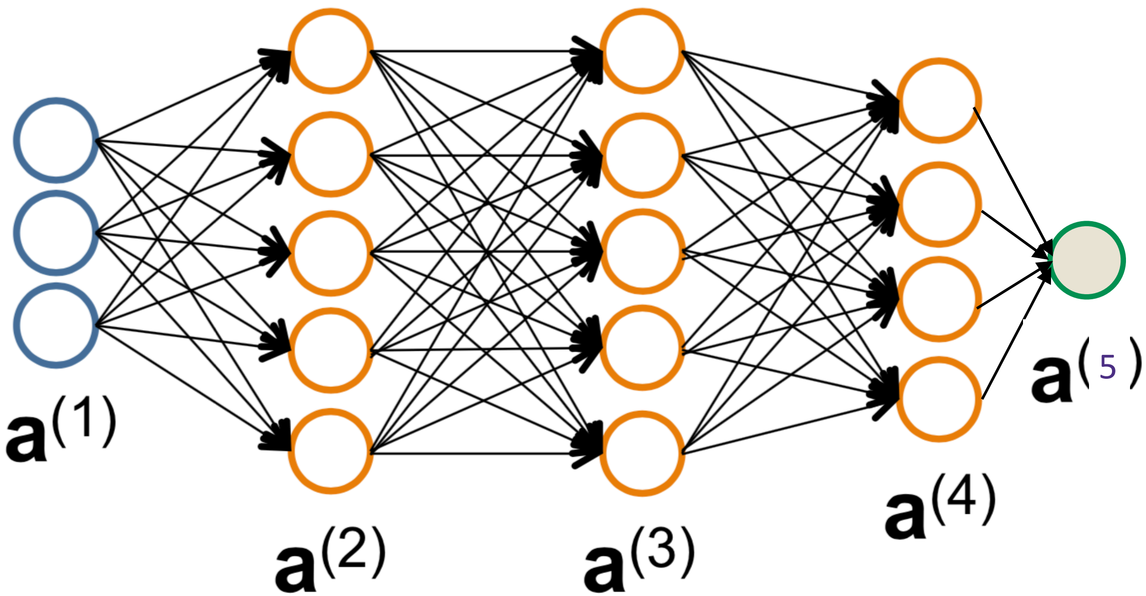
Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.



Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.



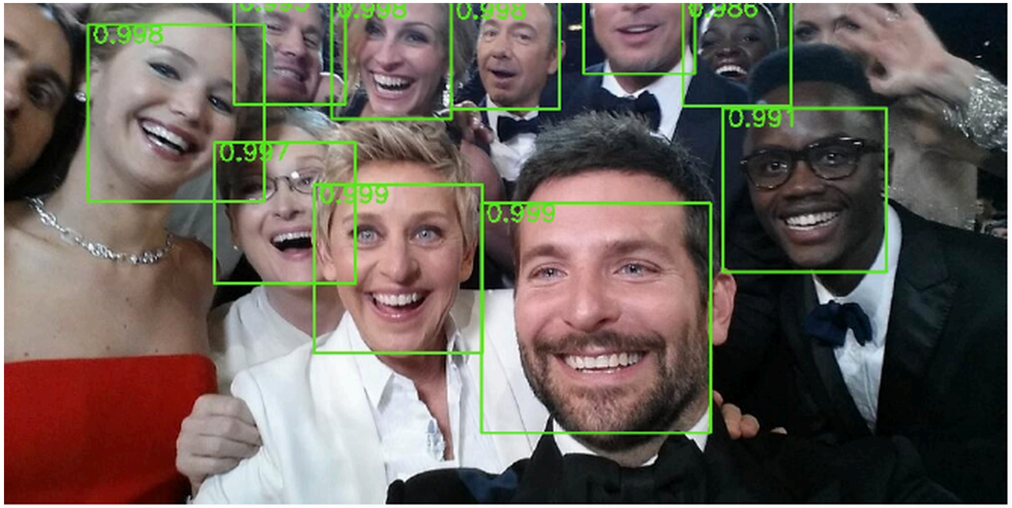
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

$$\mathbf{a}^{(k+1)} = g(\Theta \mathbf{a}^{(k)}) \quad \text{for any } \Theta \in \mathbb{R}^{n_{k+1} \times n_k}$$

A lot of parameters!! $n_1 n_2 + n_2 n_3 + \cdots + n_L n_{L+1}$

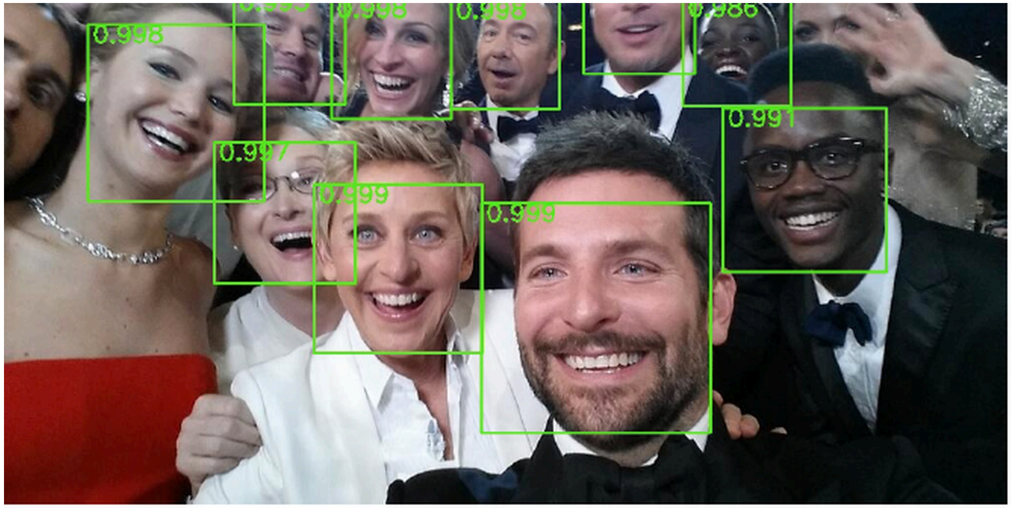
Neural Network Architecture

Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.

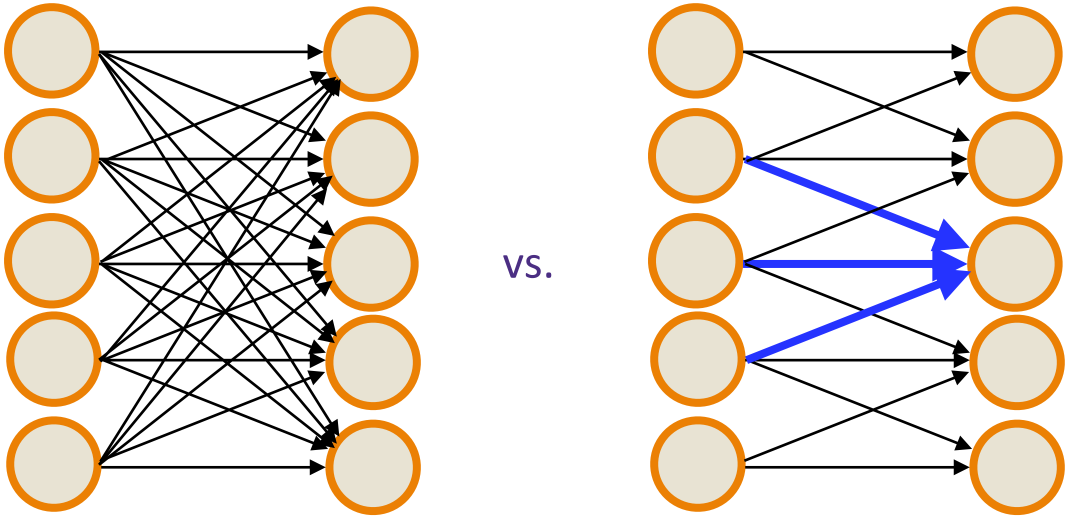


Neural Network Architecture

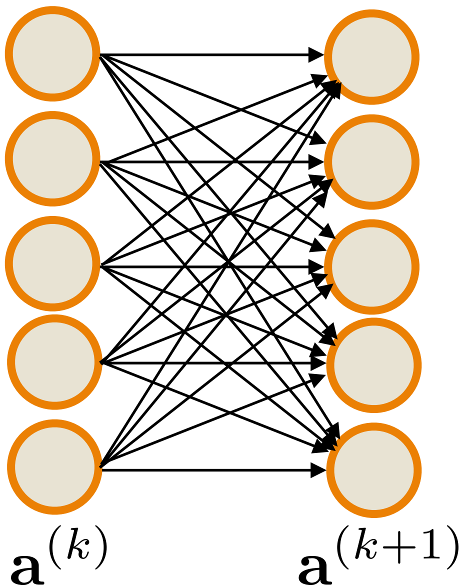
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



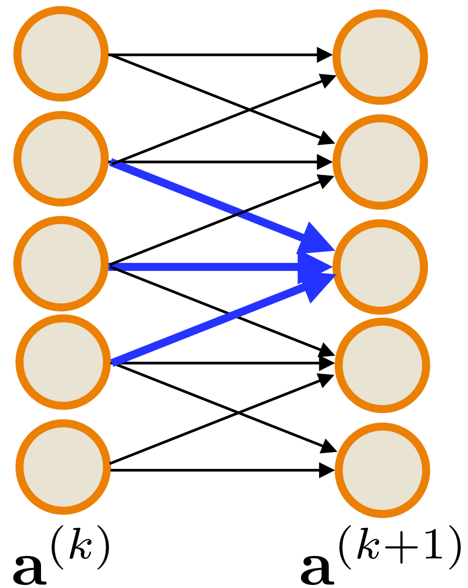
Similarly, to identify edges or other local structure, it makes sense to only look at **local information**



Neural Network Architecture



vs.

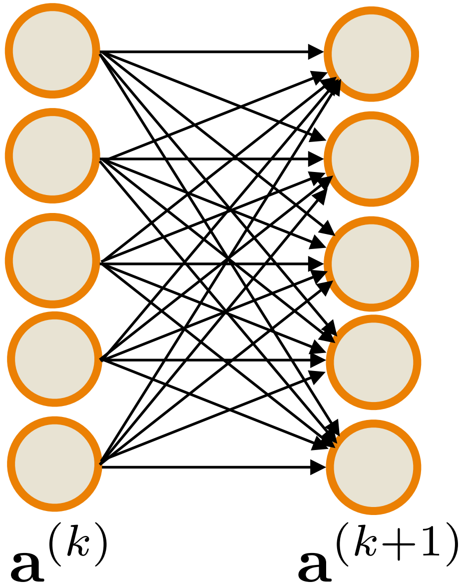


$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \qquad \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

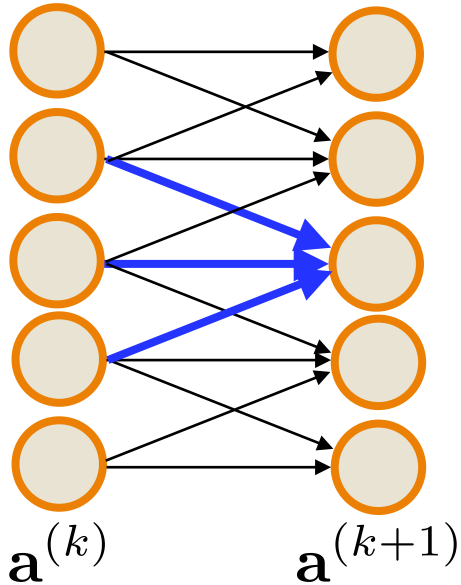
Parameters: n^2 $3n - 2$

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)}\right)$$

Neural Network Architecture



vs.



Mirror/share local weights everywhere (e.g., structure equally likely to be anywhere in image)

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \quad \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \quad \begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

Parameters: n^2 $\quad 3n-2$ $\quad 3$

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j}\mathbf{a}_j^{(k)}\right) \qquad \mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)}\right)$$

Neural Network Architecture

Fully Connected (FC) Layer

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j} \mathbf{a}_j^{(k)}\right)$$

Convolutional (CONV) Layer (1 filter)

$$\begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

m=3

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)}\right) = g([\theta * \mathbf{a}^{(k)}]_i)$$

Convolution*

$\theta = (\theta_0, \ldots, \theta_{m-1}) \in \mathbb{R}^m$ is referred to as a “filter”

Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

1	1	1	0	0

Input $x \in \mathbb{R}^n$

1	0	1

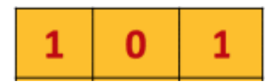
Filter $\theta \in \mathbb{R}^m$

Output $\theta * x$

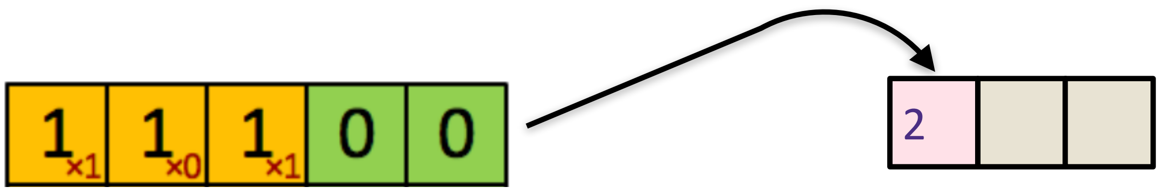
Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

Input $x \in \mathbb{R}^n$



Filter $\theta \in \mathbb{R}^m$

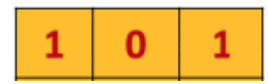


Output $\theta * x$

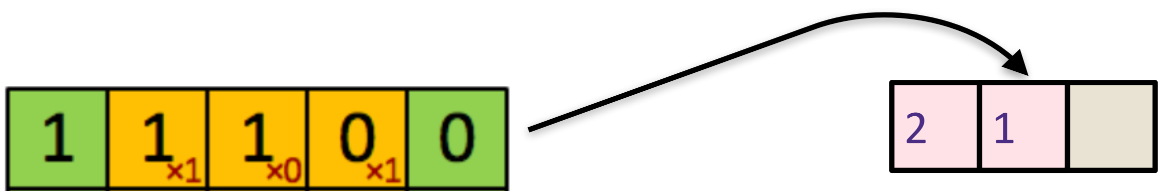
Example (1d convolution)

Input $x \in \mathbb{R}^n$

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$



Filter $\theta \in \mathbb{R}^m$

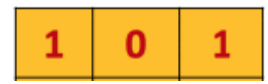


Output $\theta * x$

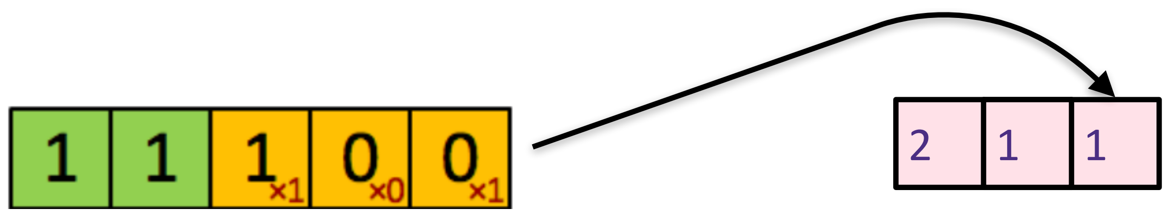
Example (1d convolution)

Input $x \in \mathbb{R}^n$

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$



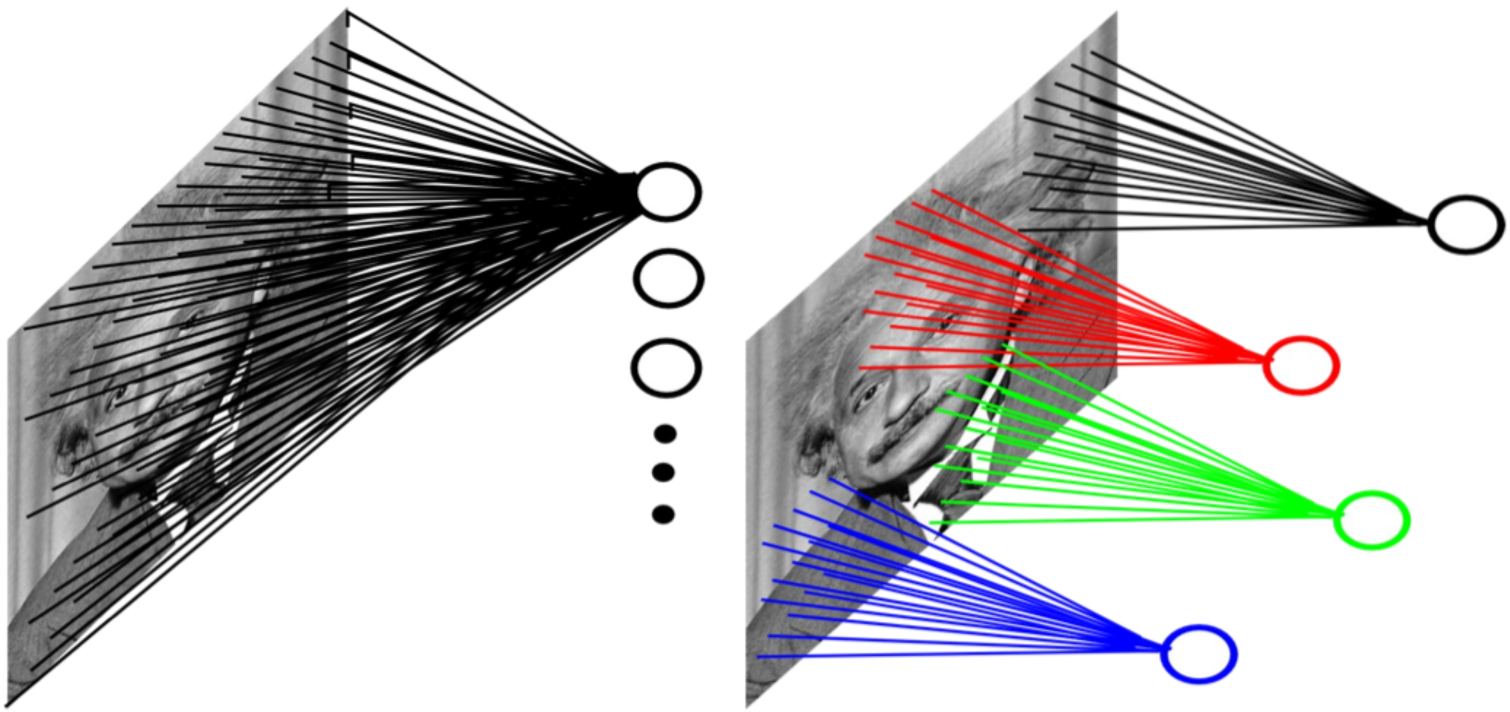
Filter $\theta \in \mathbb{R}^m$



Output $\theta * x$

2d Convolution Layer

- **Example: 200x200 image**
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



Convolution of images (2d convolution)

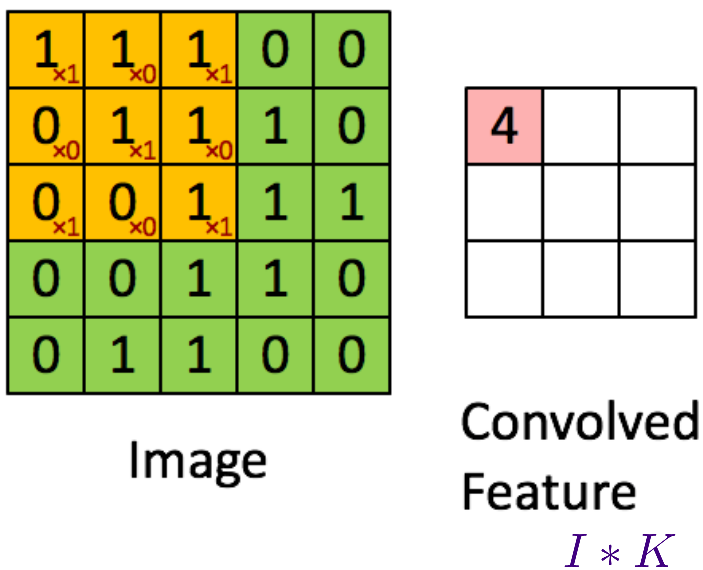
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image I

1	0	1
0	1	0
1	0	1

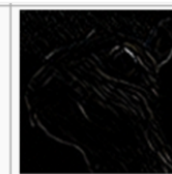
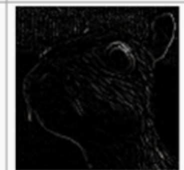
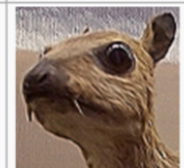
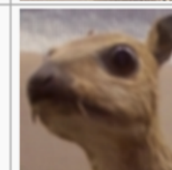
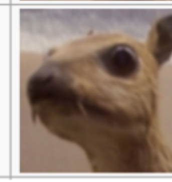
Filter K



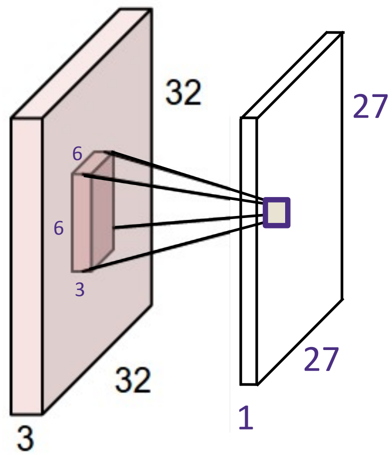
Convolution of images

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Image I

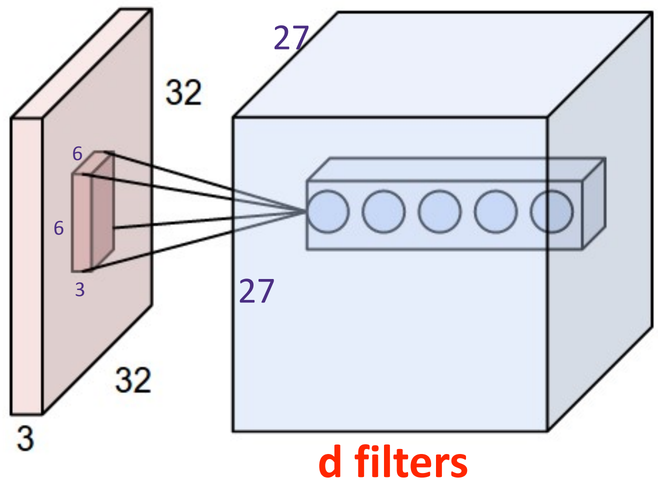
Operation	Filter K	Convolved Image $I * K$
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Stacking convolved images



$$x \in \mathbb{R}^{n \times n \times r}$$

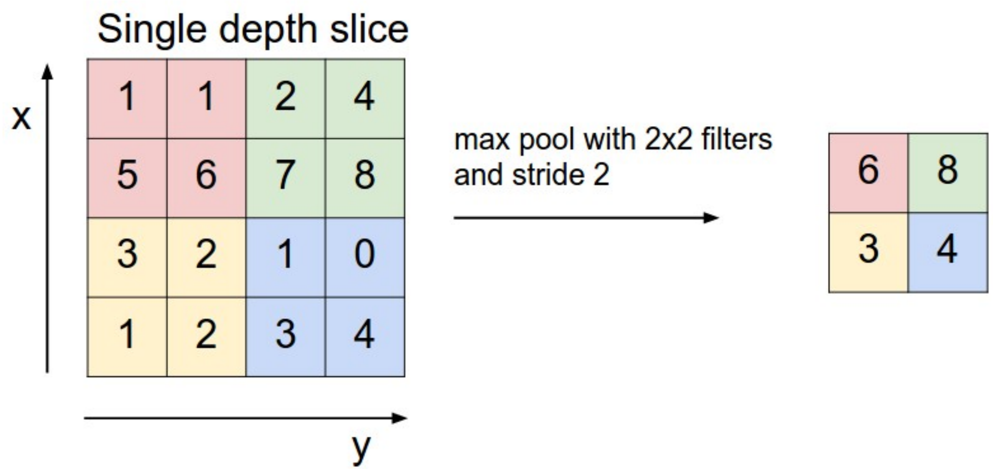
Stacking convolved images

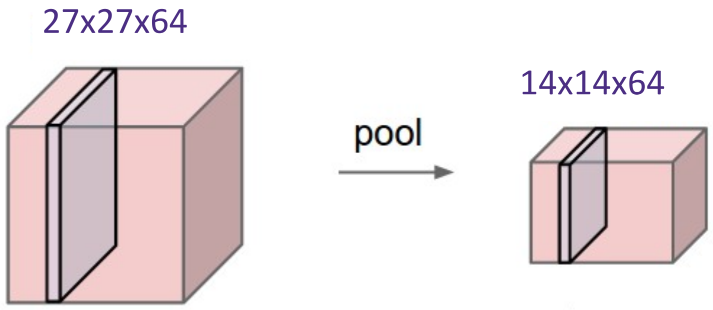


Repeat with d filters!

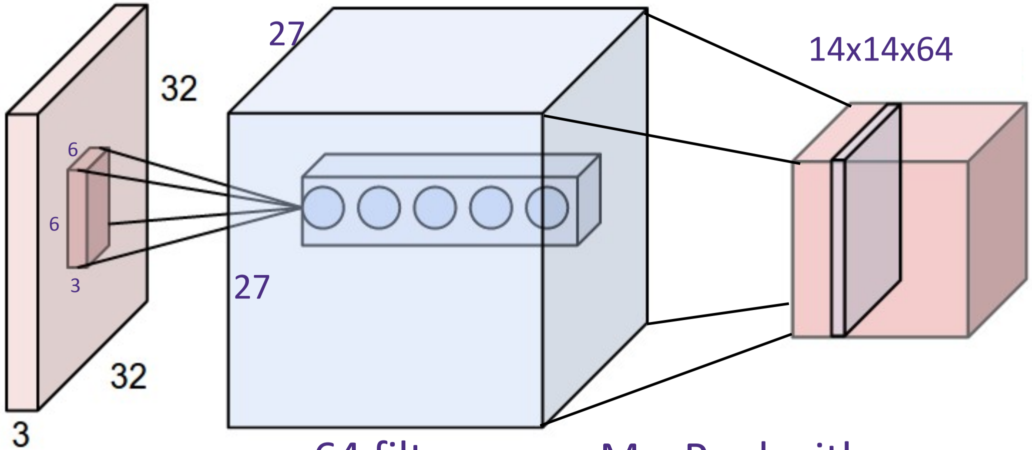
Pooling

Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”





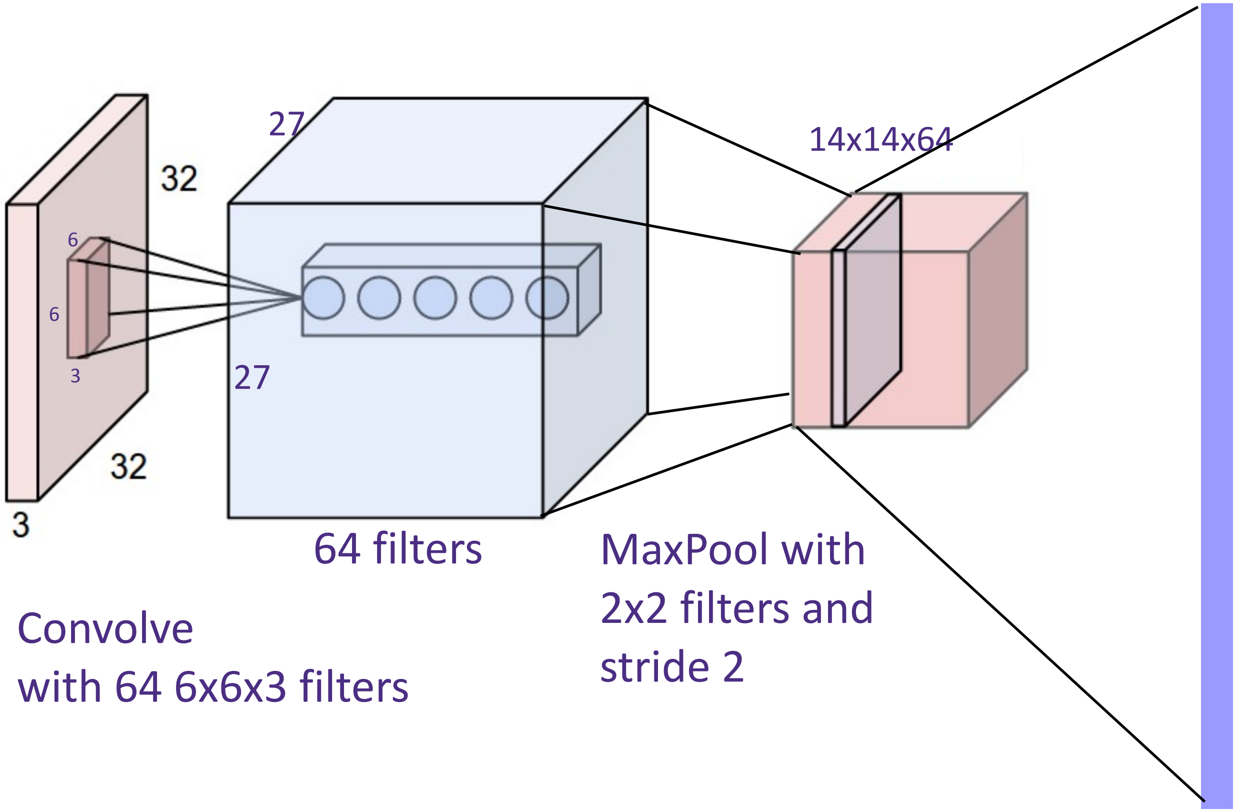
Pooling Convolution layer



MaxPool with
2x2 filters and
stride 2

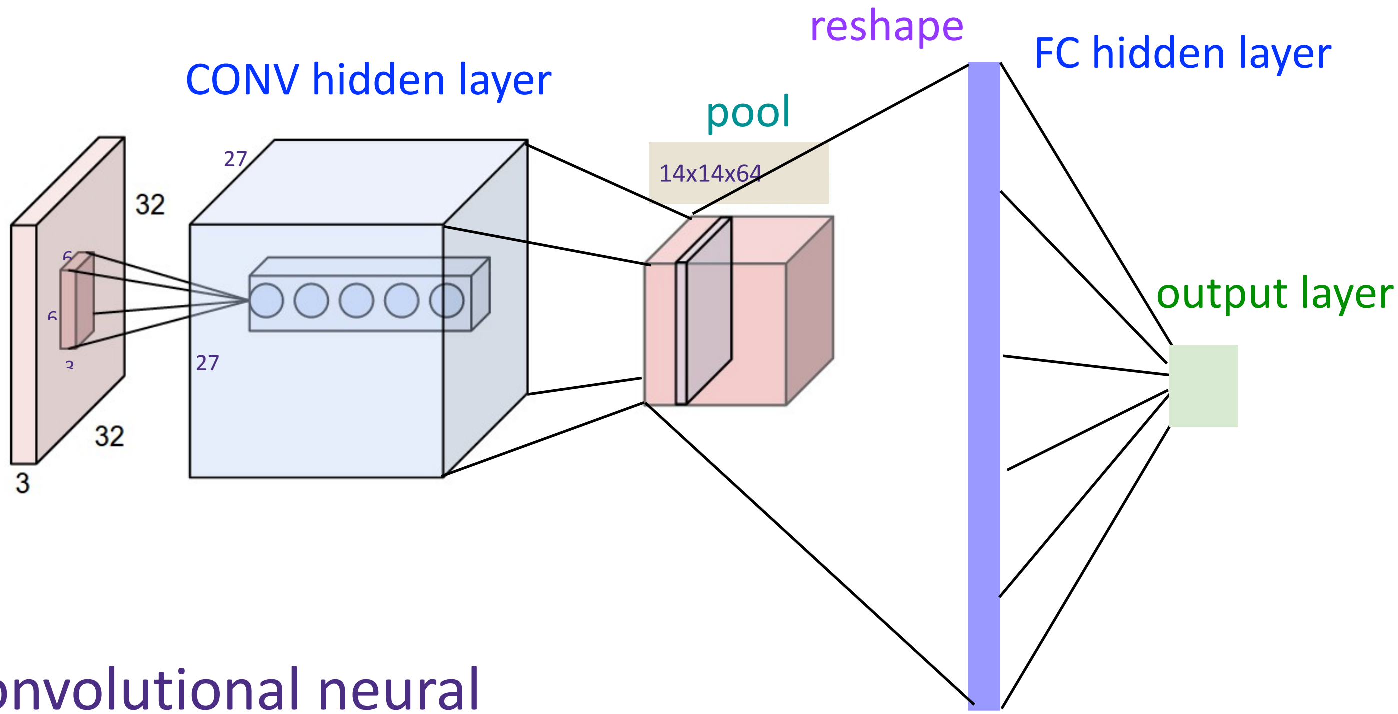
Convolve
with 64 6x6x3 filters

Flattening



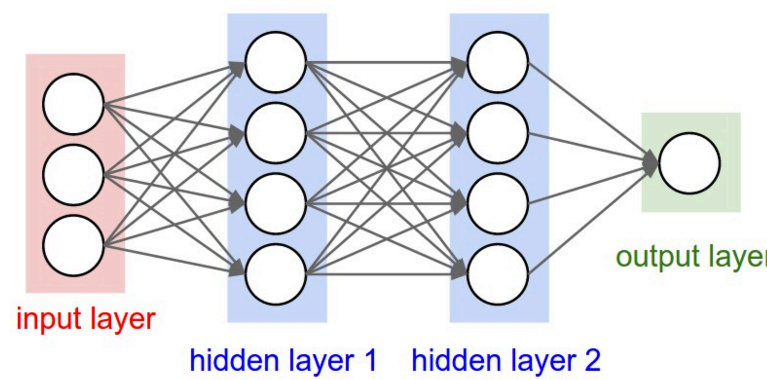
Flatten into a single vector of size 14*14*64=12544

Training Convolutional Networks

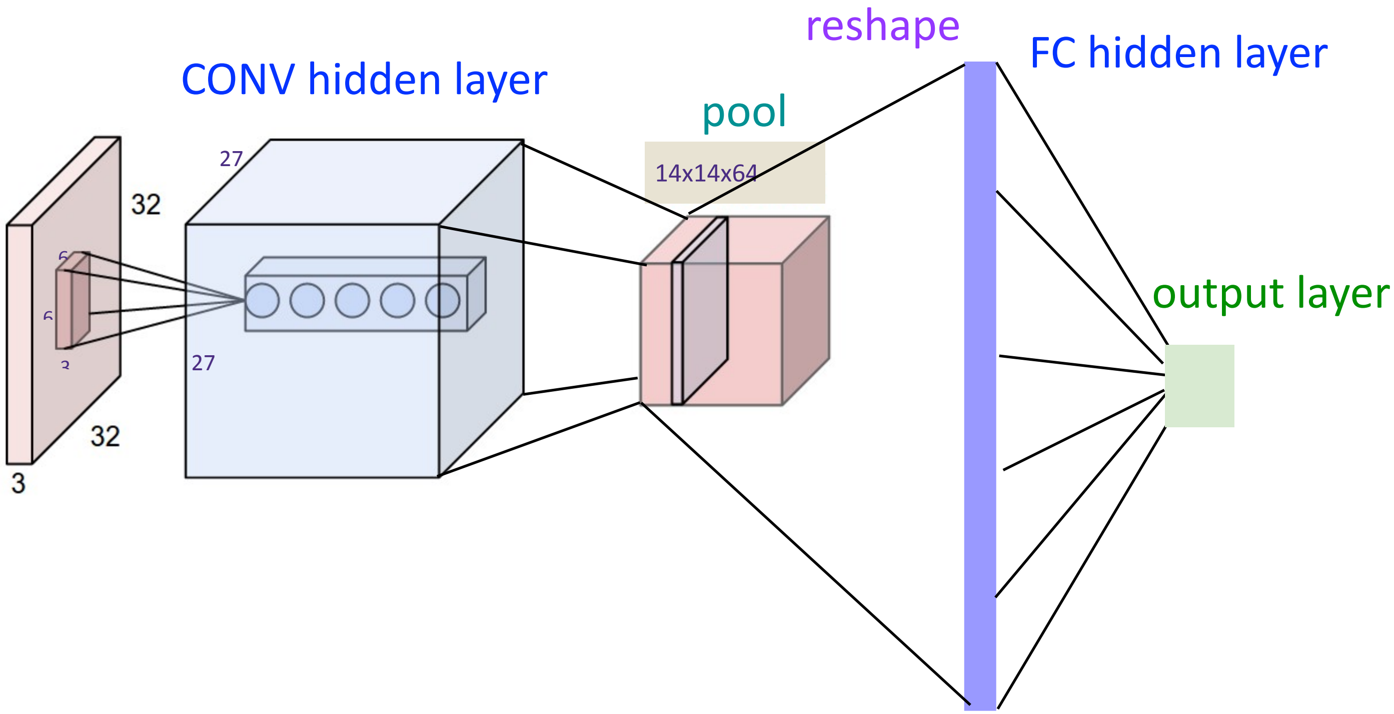


Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.

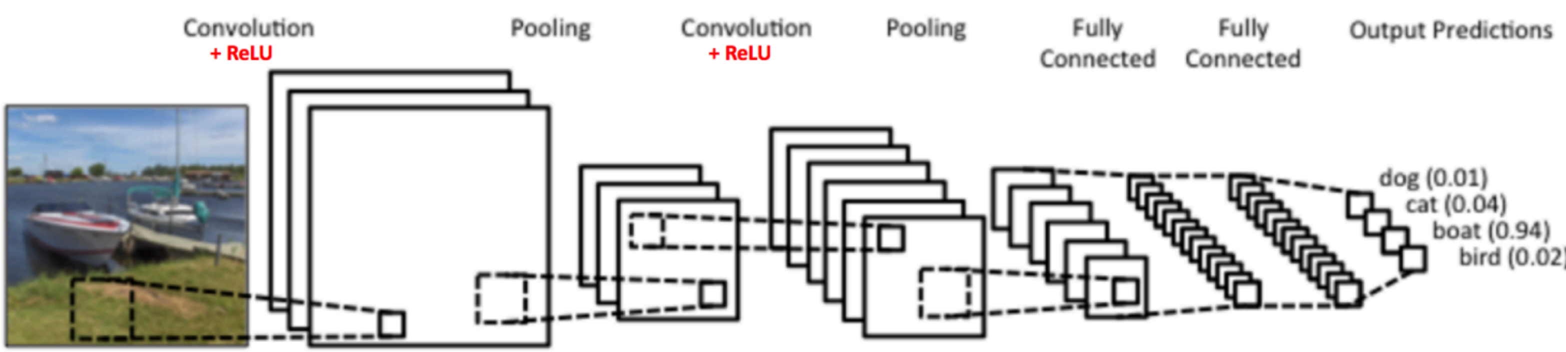
Train with SGD!

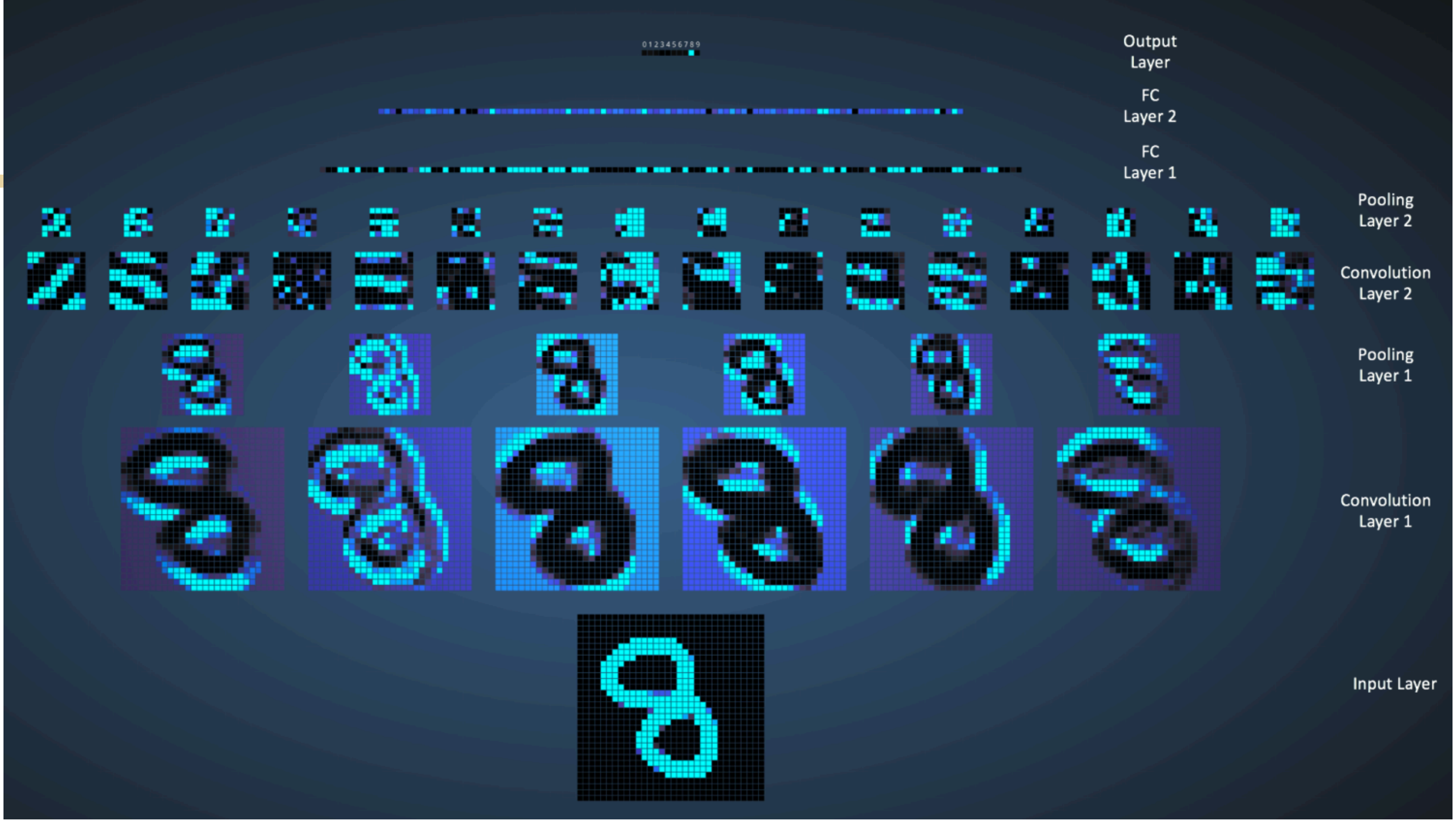


Training Convolutional Networks

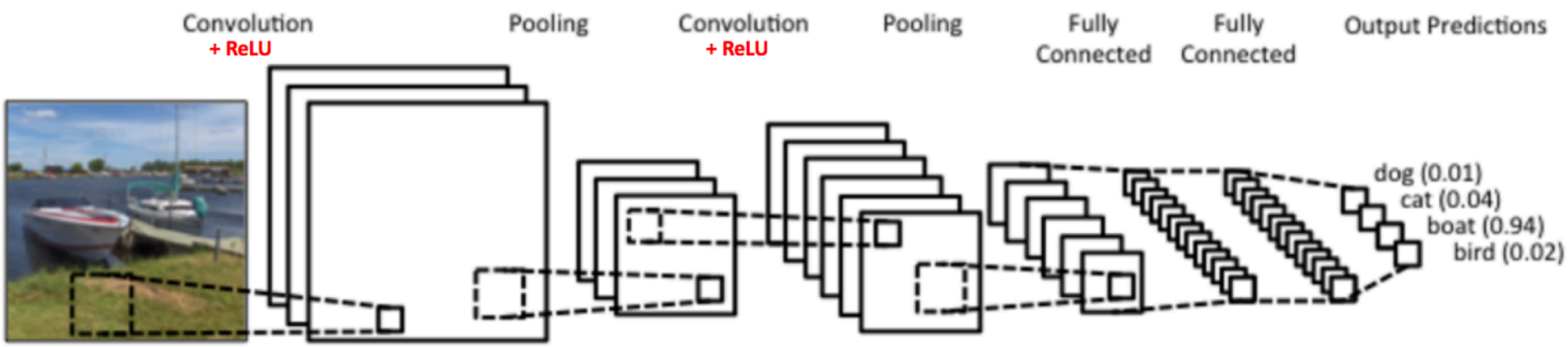


Real example network: LeNet





Real example network: LeNet



Famous CNNs

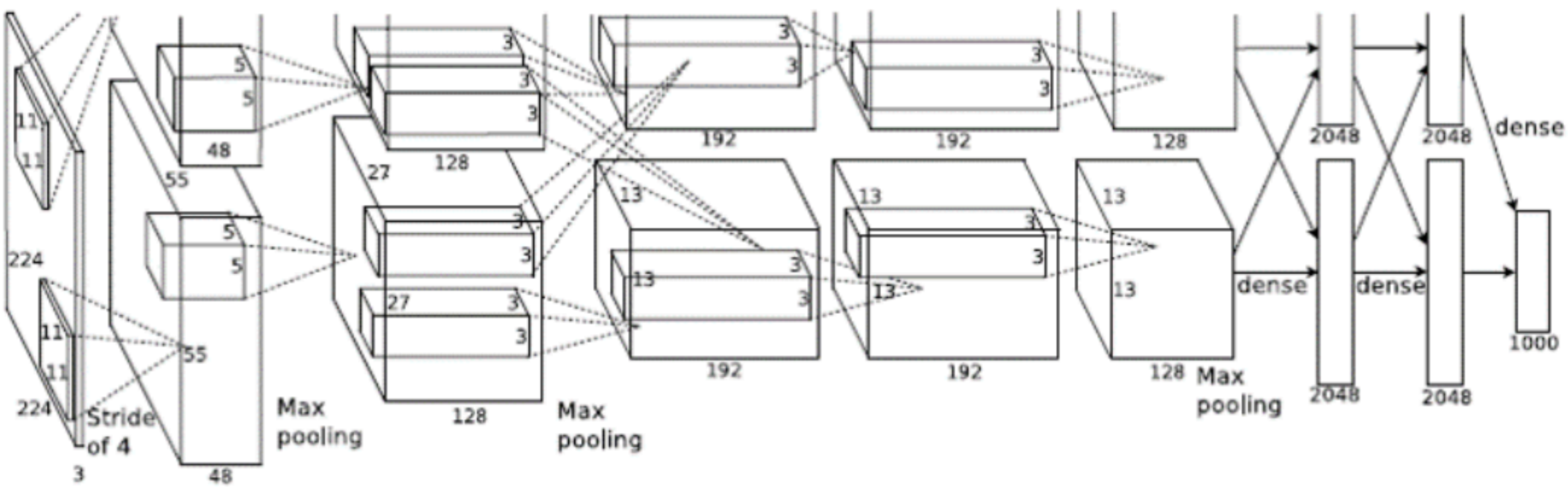
ImageNet Dataset

~14 million images, 20k classes

Deng et al. "Imagenet: a large scale hierarchical image database" '09

AlexNet

Breakthrough on ImageNet: ~the beginning of deep learning era



Krizhevsky, Sutskever, Hinton “ImageNet Claasification with Deep Convolutional Neural Networks”, NIPS 2012.

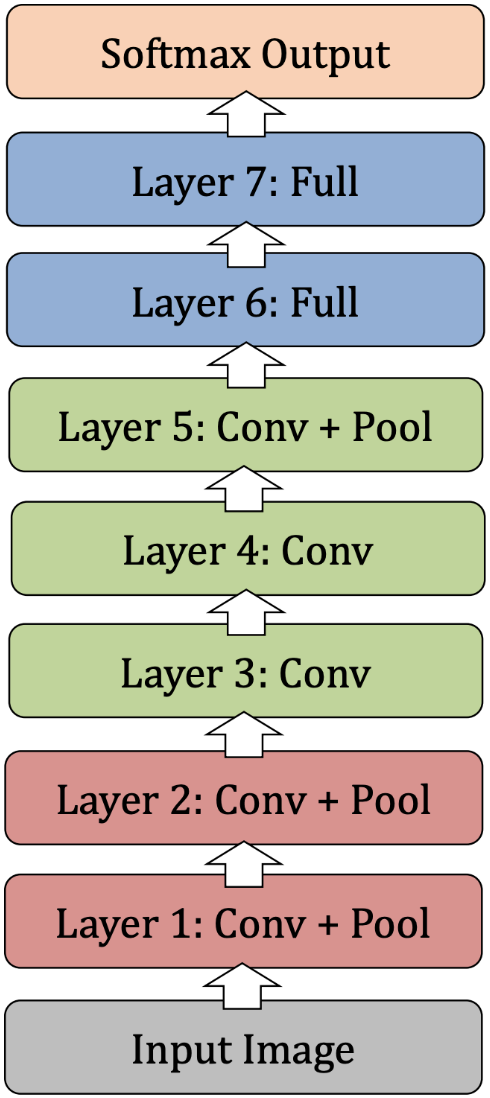
AlexNet

8 layers, ~60M parameters

Top5 error: 18.2%

Techniques used:
ReLU activation, overlapping pooling, dropout, ensemble (create 10 patches by cropping and average the predictions), data-augmentation (intensity of RGB channels)

[From Rob Fergus' CIFAR 2016 tutorial]

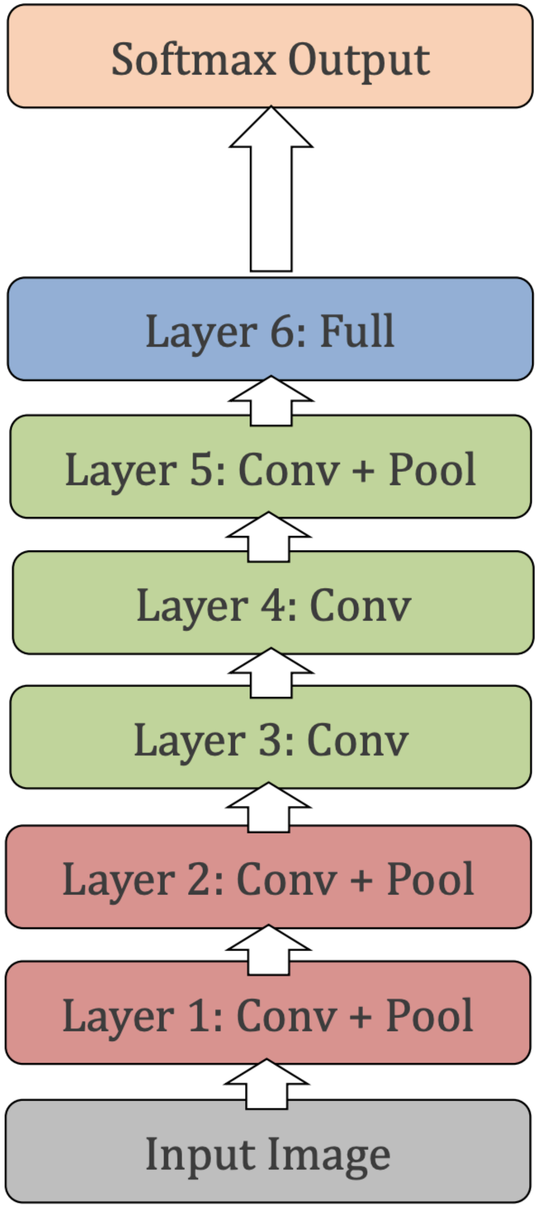


AlexNet

Remove top fully-connected layer 7

Drop ~16 million parameters

1.1% drop in performance



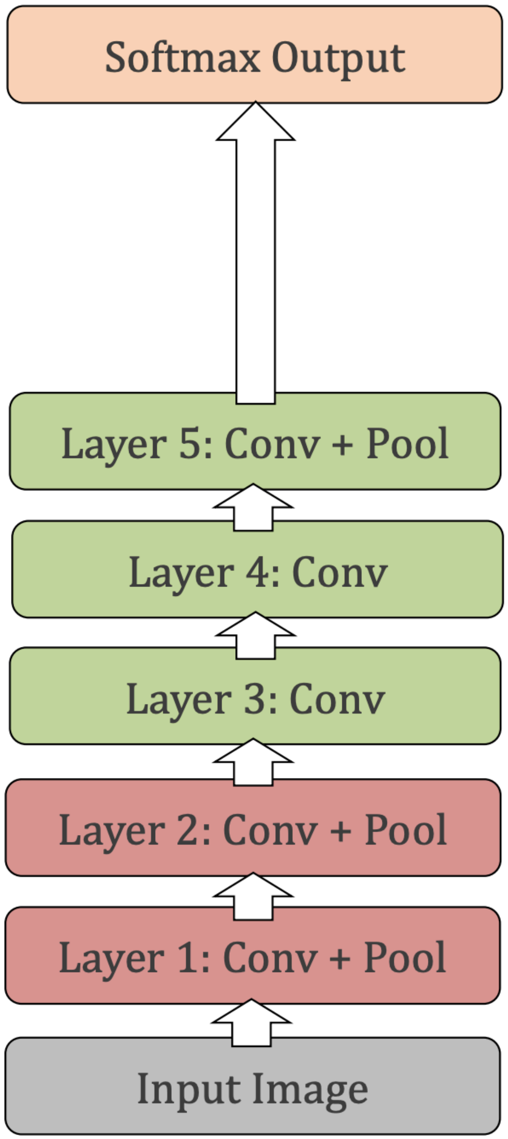
[From Rob Fergus’ CIFAR 2016 tutorial]

AlexNet

Remove both fully connected layers 6 and 7

Drop ~50 million parameters

5.7% drop in performance



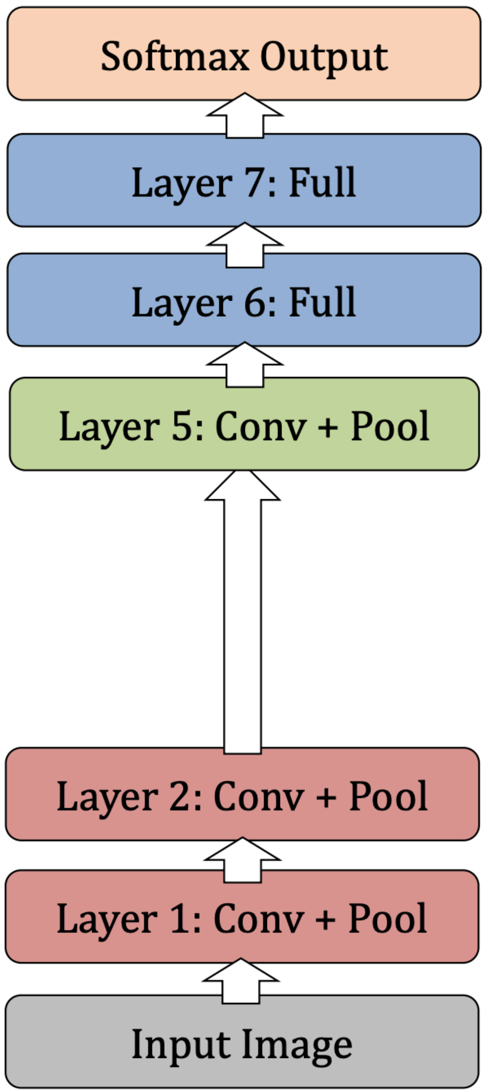
[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

Remove upper convolutio / feature extractor layers (layer 3 and 4)

Drop ~1 million parameters

3% drop in performance



[From Rob Fergus' CIFAR 2016 tutorial]

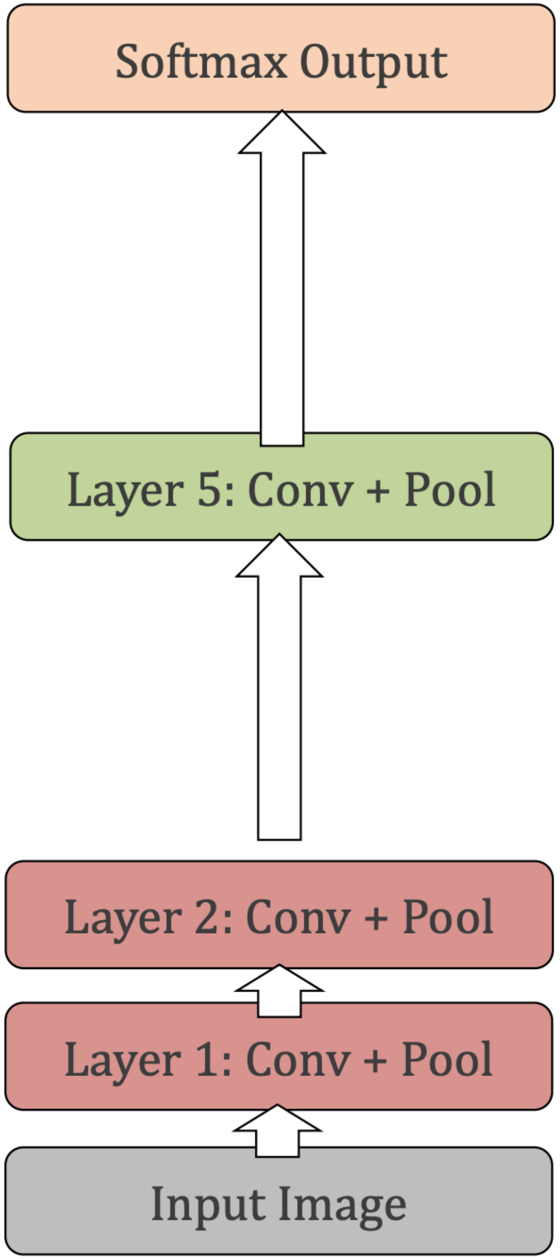
AlexNet

Remove top fully connected layer 6,7 and upper convolution layers 3,4.

33.5% drop in performance.

Depth of the network is the key.

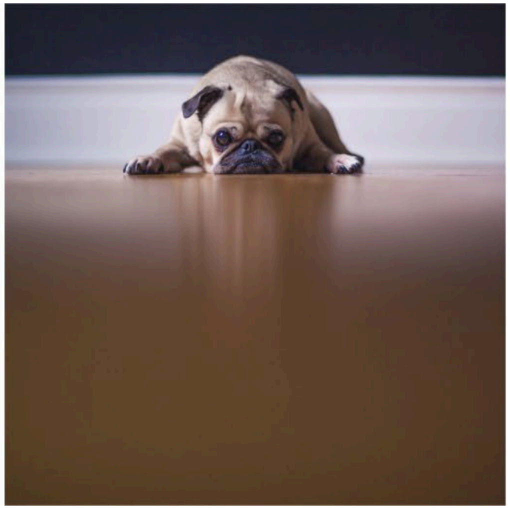
[From Rob Fergus' CIFAR 2016 tutorial]



GoogLeNet

Motivation: multiscale nature of images



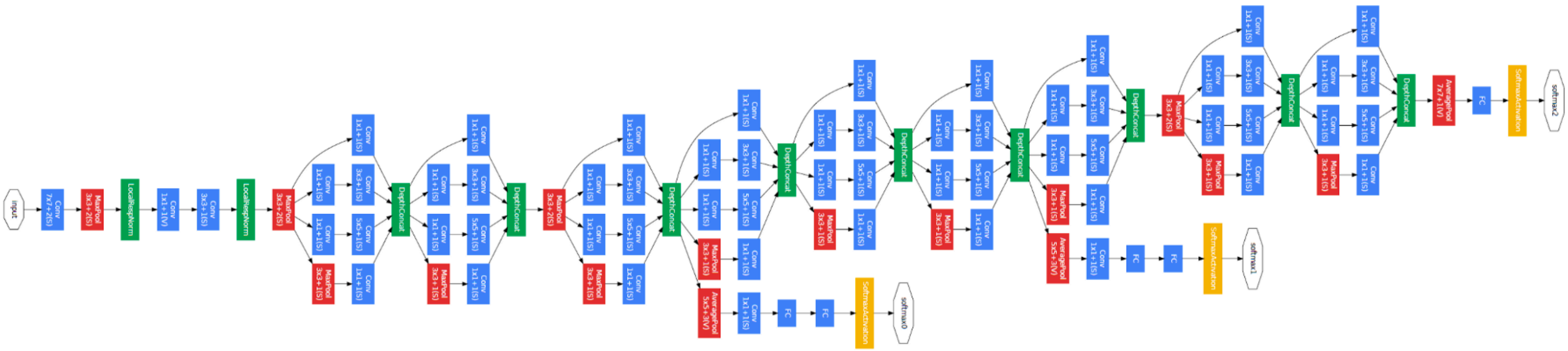


Large kernel for global features, and **smaller kernel** for local features.

Idea: have multiple different-size kernels at any layer.

[Going Deep with Convolutions, Szegedy et al. ’14]

GoogLeNet

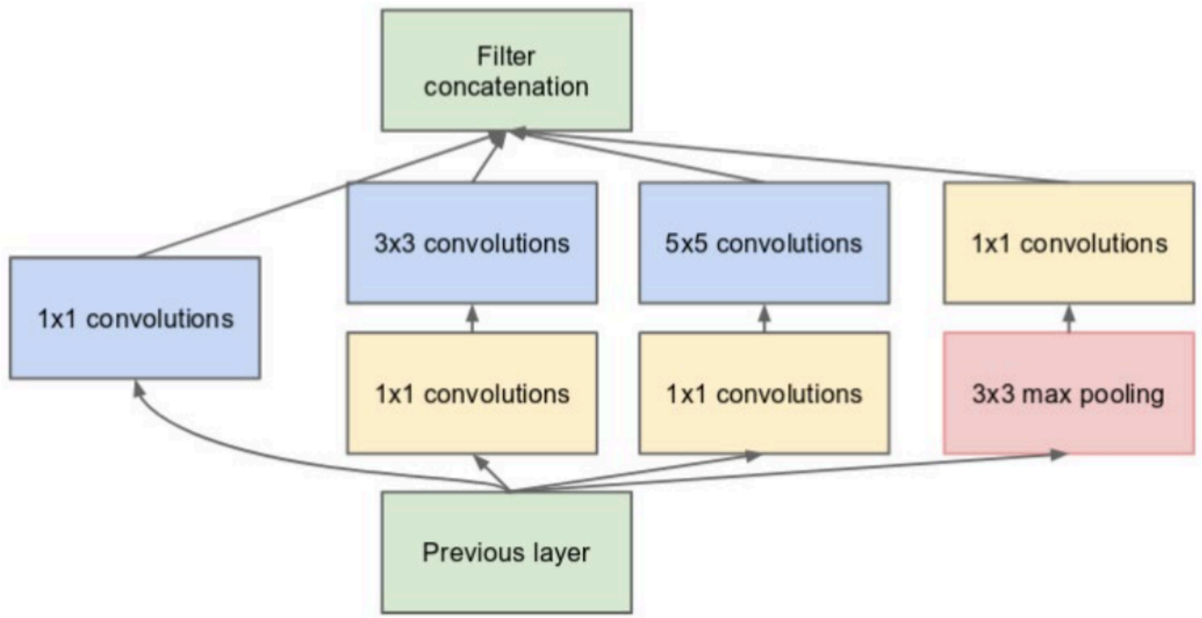


Large kernel for global features, and **smaller kernel** for local features.

Idea: have multiple different-size kernels at any layer.

[Going Deep with Convolutions, Szegedy et al. ’14]

Inception Module



Multiple filter scales at each layer

Dimensionality reduction to keep computational requirements down

[Going Deep with Convolutions, Szegedy et al. ’14]

Residual Networks

Motivation: extremely deep nets are hard to train (gradient explosion/ vanishing)

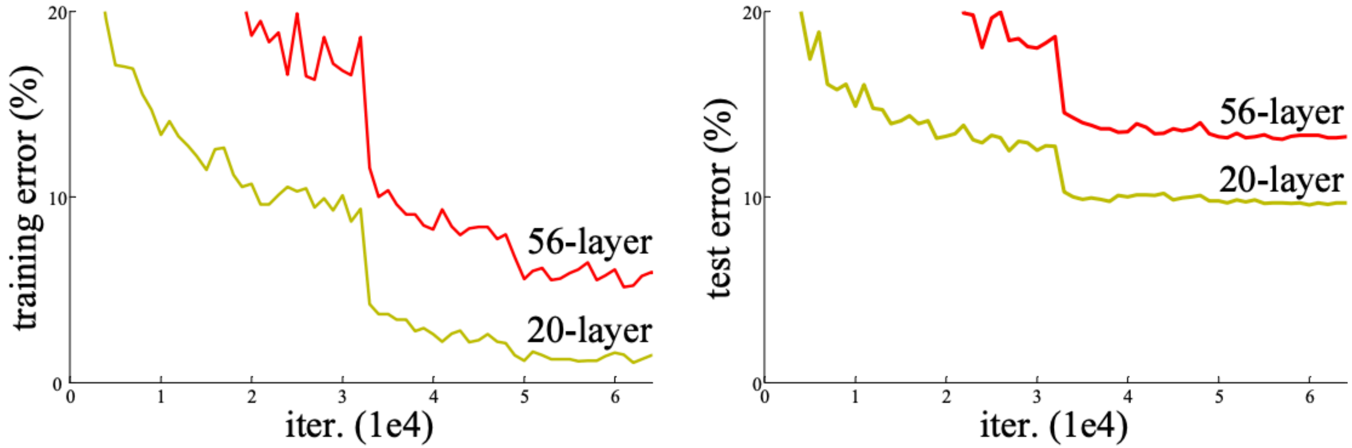


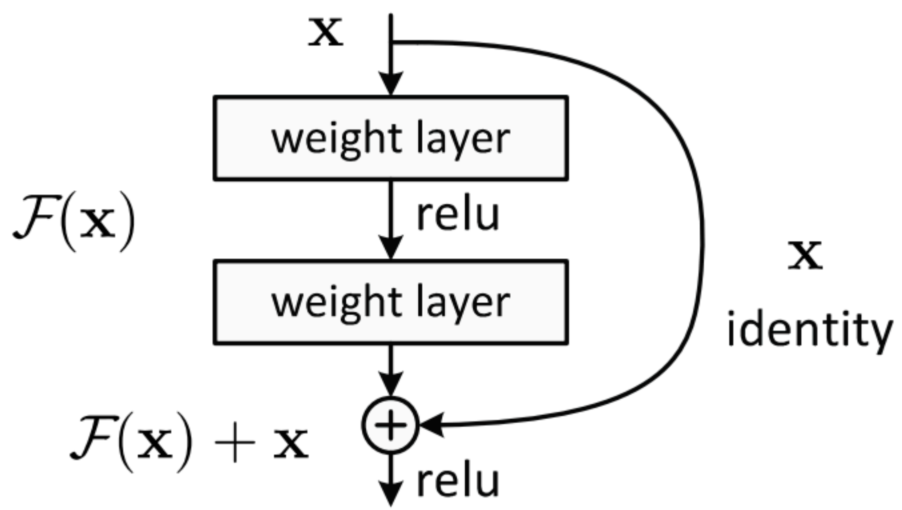
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

[He, Zhang, Ren, Sun, '16]

Residual Networks

Idea: identity shortcut, skip one or more layers.

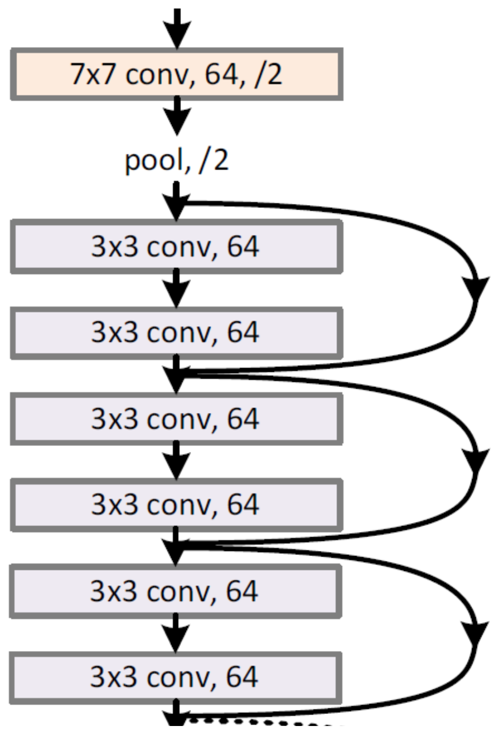
Justification: network can easily simulate shallow network ($F \approx 0$), so performance should not degrade by going deeper.

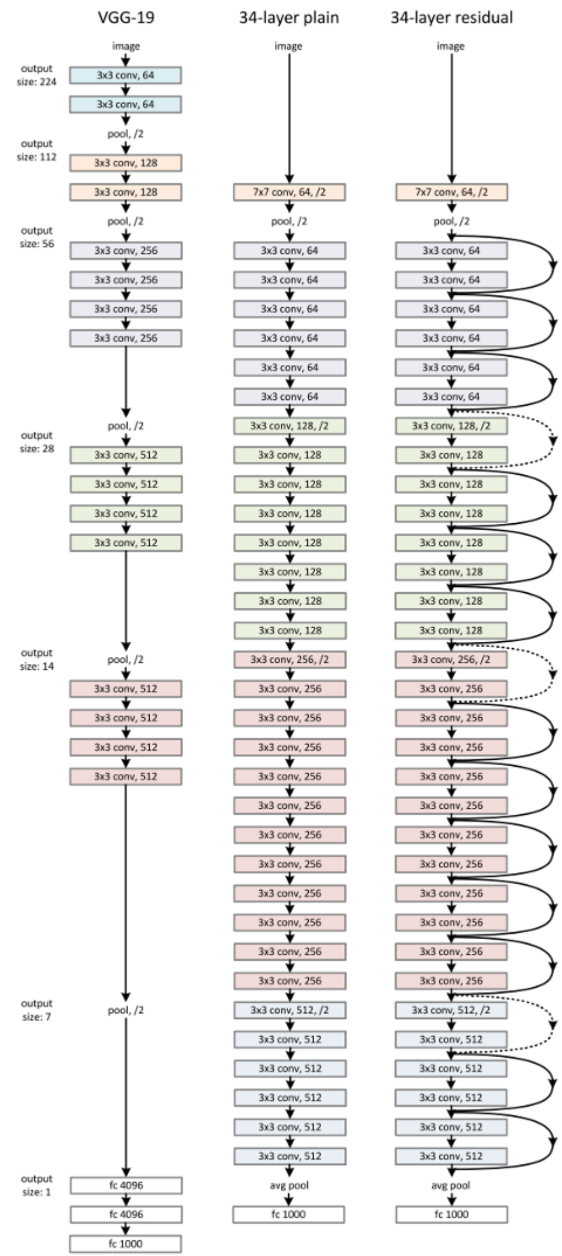


[He, Zhang, Ren, Sun, ’16]

Residual Networks

- 3.57% top-5 error on ImageNet
- First deep network with > 100 layers.
- Widely used in many domains (AlphaGo)





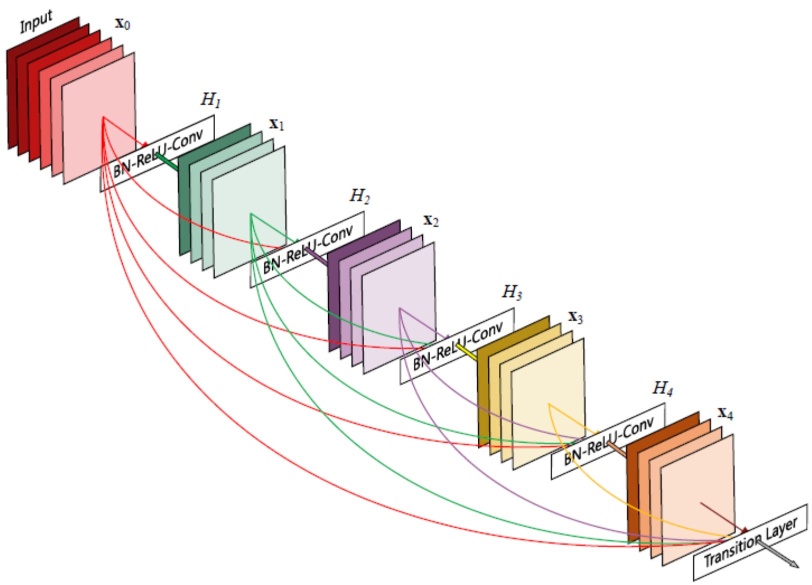
[He, Zhang, Ren, Sun, '16]

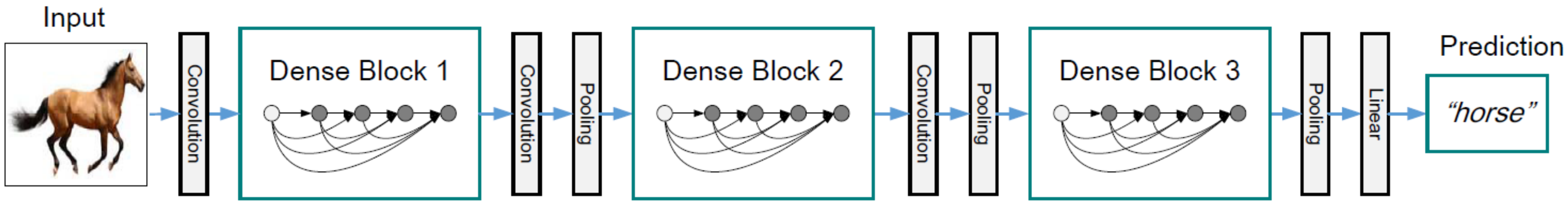
Densely Connected Network

Idea: explicit forward output of layer to all future layers (by concatenation)

Intuition: helps vanishing gradients, encourage reuse features (reduce parameter count)

Issues: network maybe too wide, need to be careful about memory consumption





[He, Zhang, Ren, Sun, '16]

Neural Architecture / Hyper-Parameter Search

Many design choices:

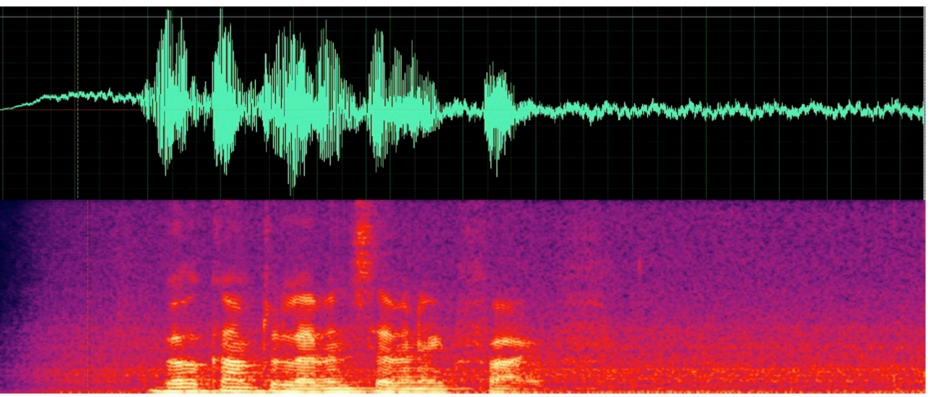
- Number of layers, width, kernel size, pooling, connections, etc.
- Normalization, learning rate, batch size, etc.

Strategies:

- Grid search
- Random search [Bergestra & Bengio ’12]
- Bandit-based [Li et al. ’16]
- Gradient-based (DARTS) [Liu et al. ’19]
- Neural tangent kernel [Xu et al. ’21]
- …

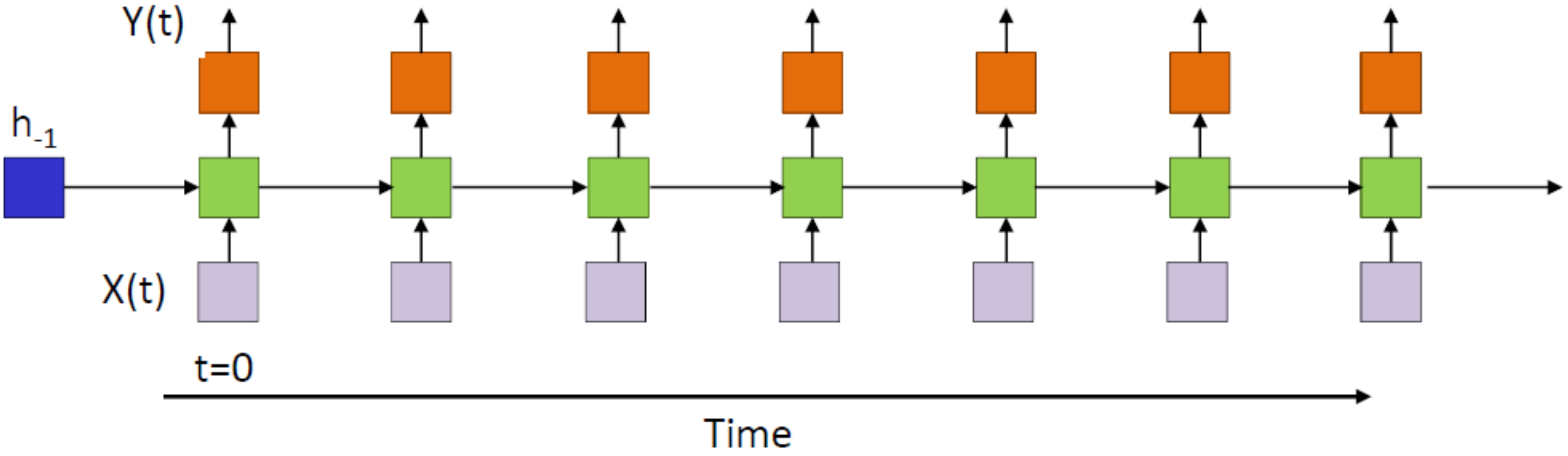
Recurrent Neural Networks

Sequence Data



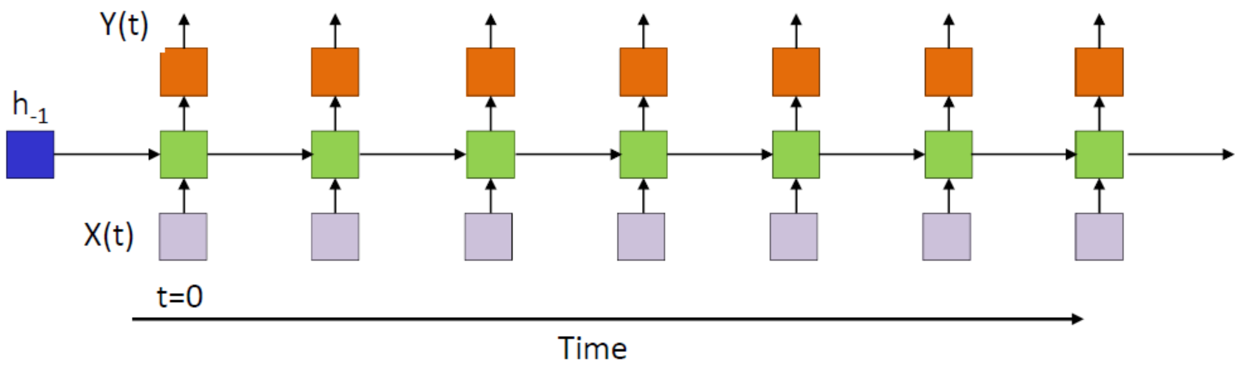
State-Space Model

- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state



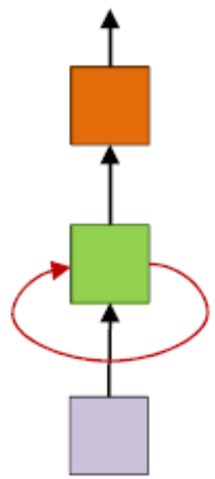
Recurrent Neural Network

- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state



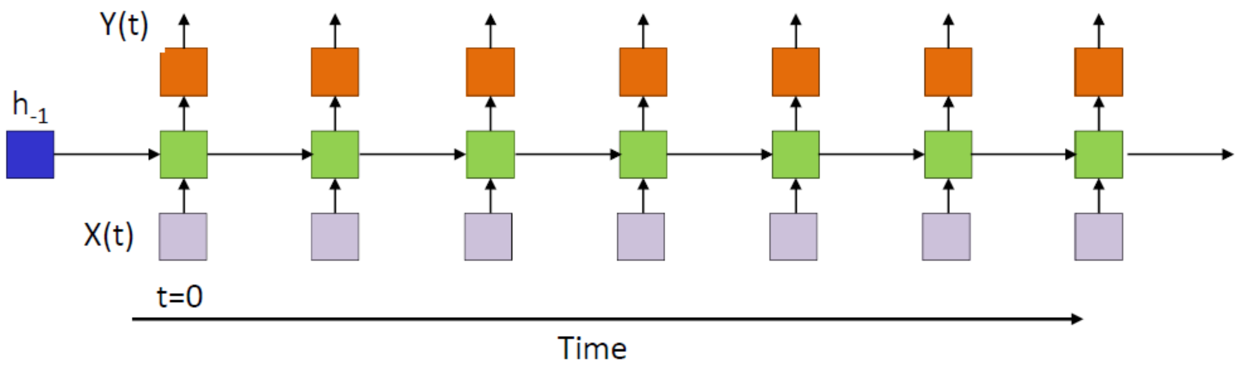
Fully-connect NN vs. RNN

- h_t: a vector summarizes all past inputs (a.k.a. "memory")
- h_{-1} affects the entire dynamics (typically set to zero)
- X_t affects all the outputs and states after t
- Y_t depends on $X_0, \ldots, X_t$



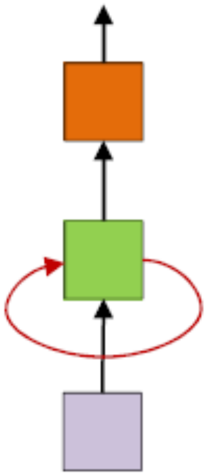
Recurrent Neural Network

- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state

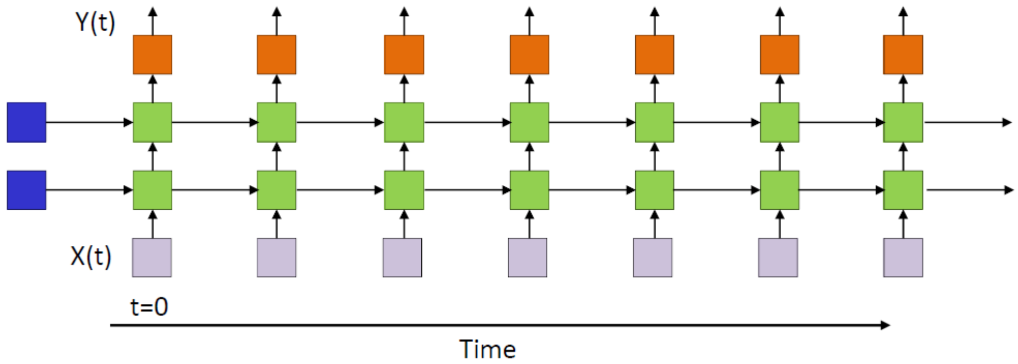


Fully-connect NN vs. RNN

- RNN can be viewed as repeated applying fully-connected NNs
- $h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$
- $Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$
- σ_1, σ_2 are activation functions (sigmoid, ReLU, tanh, etc)

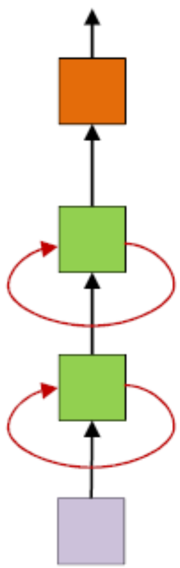


Recurrent Neural Network



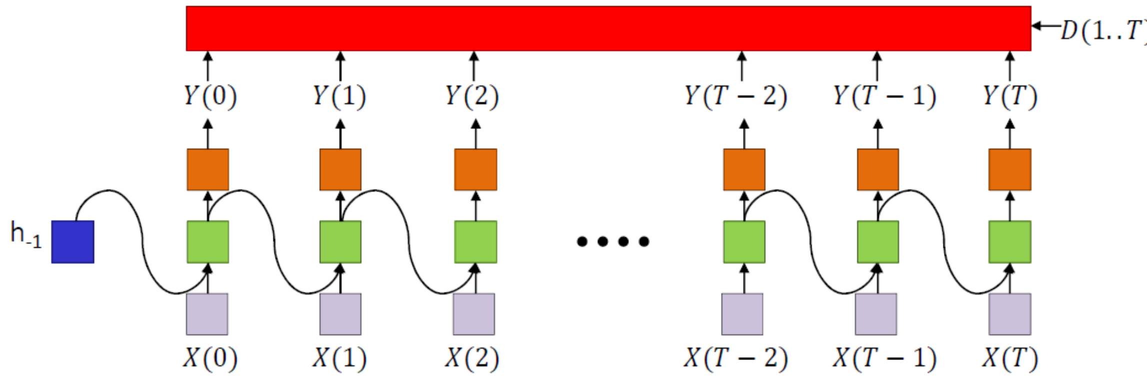
Stack K layers of fully-connected NN

- $h_t^{(k)}$: hidden state
- X_t: input
- Y_t: output
- $h_t^{(1)} = f_1^{(1)}(h_{t-1}^{(1)}, X_t; \theta)$
- $h_t^{(k)} = f_1^{(k)}(h_{t-1}^{(k)}, h_t^{(k-1)}; \theta)$
- $Y_t = f_2(h_t^{(K)}; \theta)$
- $h_{-1}^{(k)}$: initial states



Training Recurrent Neural Network

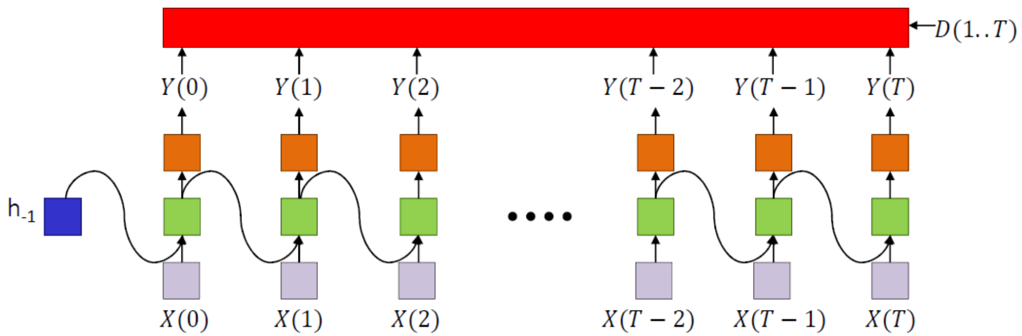
- h_t: hidden state
- X_t: input
- Y_t: output
- $Y_t, h_t = f(h_{t-1}, X_t; \theta)$
- h_{-1}: initial state



- Data: $\{(X_t, D_t)\}_{t=1}^{T}$ (RNN can handle more general data format)
- Loss $L(\theta) = \sum_{t=1}^{T} \ell(Y_t, D_t)$
- Goal: learn θ by gradient-based method
  - Back propagation

Back Propagation Through Time

- $h_t = \sigma_1(W^{(1)}X_t + W^{(11)}h_{t-1} + b^{(1)})$
- $Y_t = \sigma_2(W^{(2)}h_t + b^{(2)})$
- $Z_t^{(1)}$: pre-activation of hidden state
  $(h_t = \sigma_1(Z_t^{(1)}))$
- $Z_t^{(2)}$: pre-activation of output
  $(Y_t = \sigma_2(Z_t^{(2)}))$



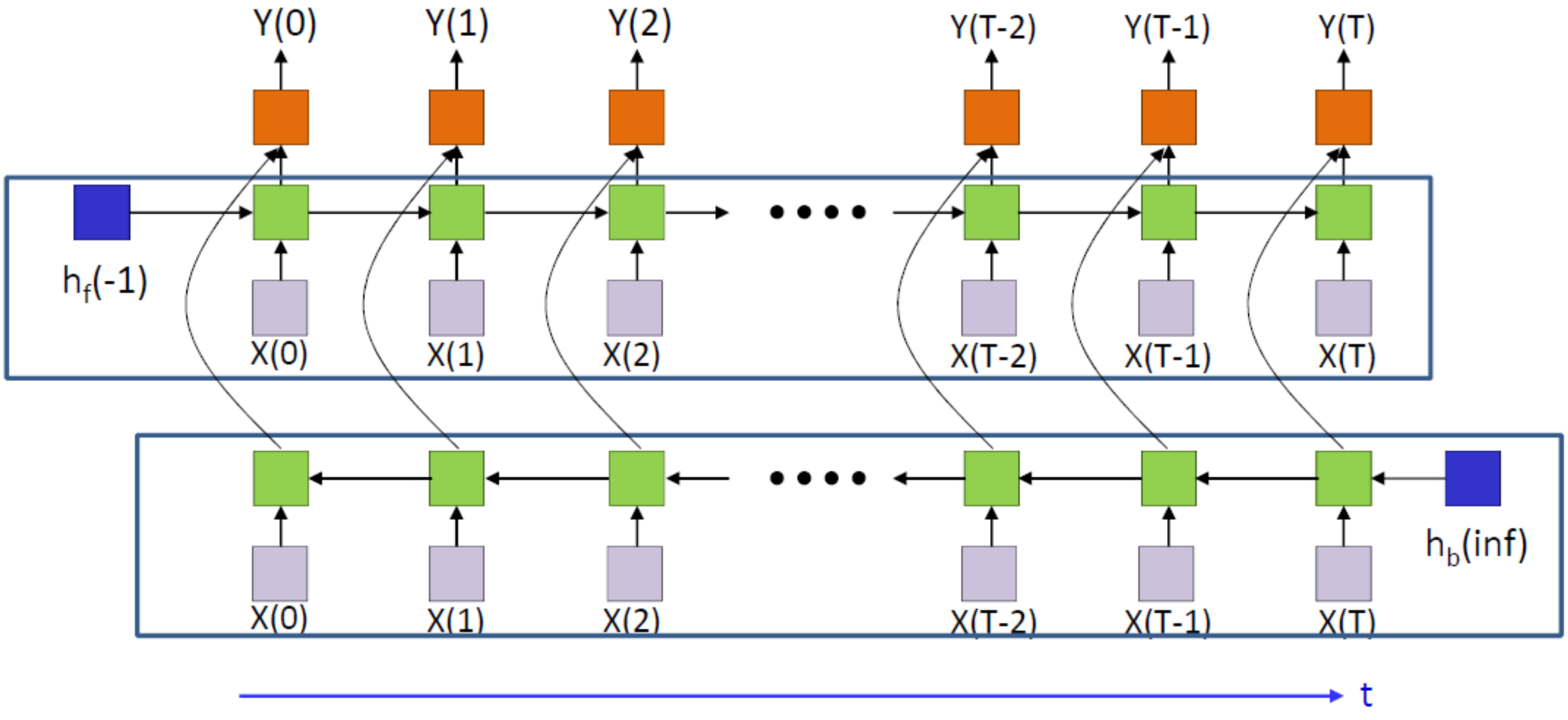
Back Propagation Through Time

Back Propagation Through Time

Extensions

What if Y_t depends on the entire inputs?

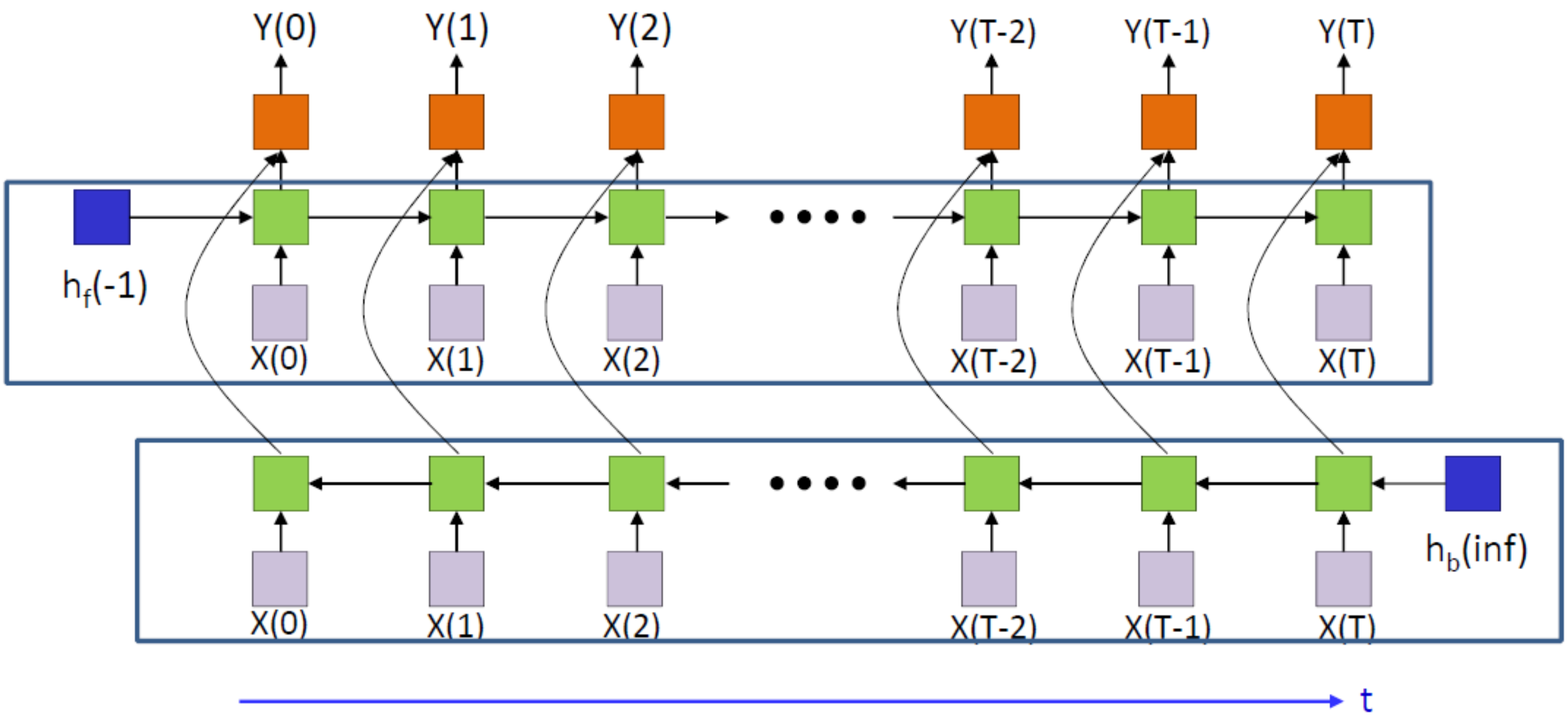
- Biredictional RNN:
    - AN RNN for forward dependencies: t= 0,…,T
    - An RNN for backward dependencies: t= T,…0
    - $Y_t = f_2(h_t^f, h_t^b; \theta)$



Extensions

RNN for sequence classification (sentiment analysis)

- $Y = \max_t Y_t$
- Cross-entropy loss



Practical issues of RNN

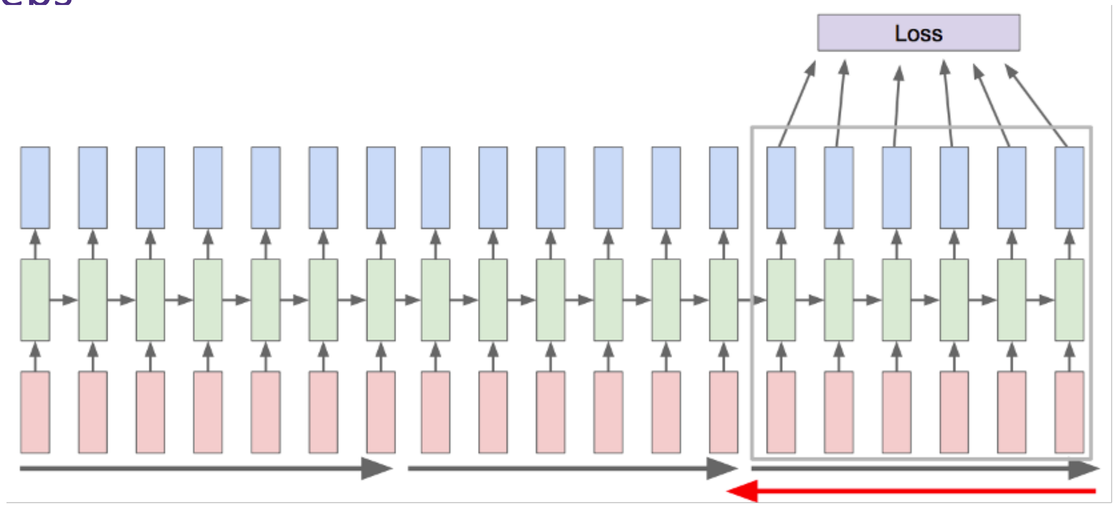
Linear RNN derivation

Practical issues of RNN: training

Gradient explosion and gradient vanishing

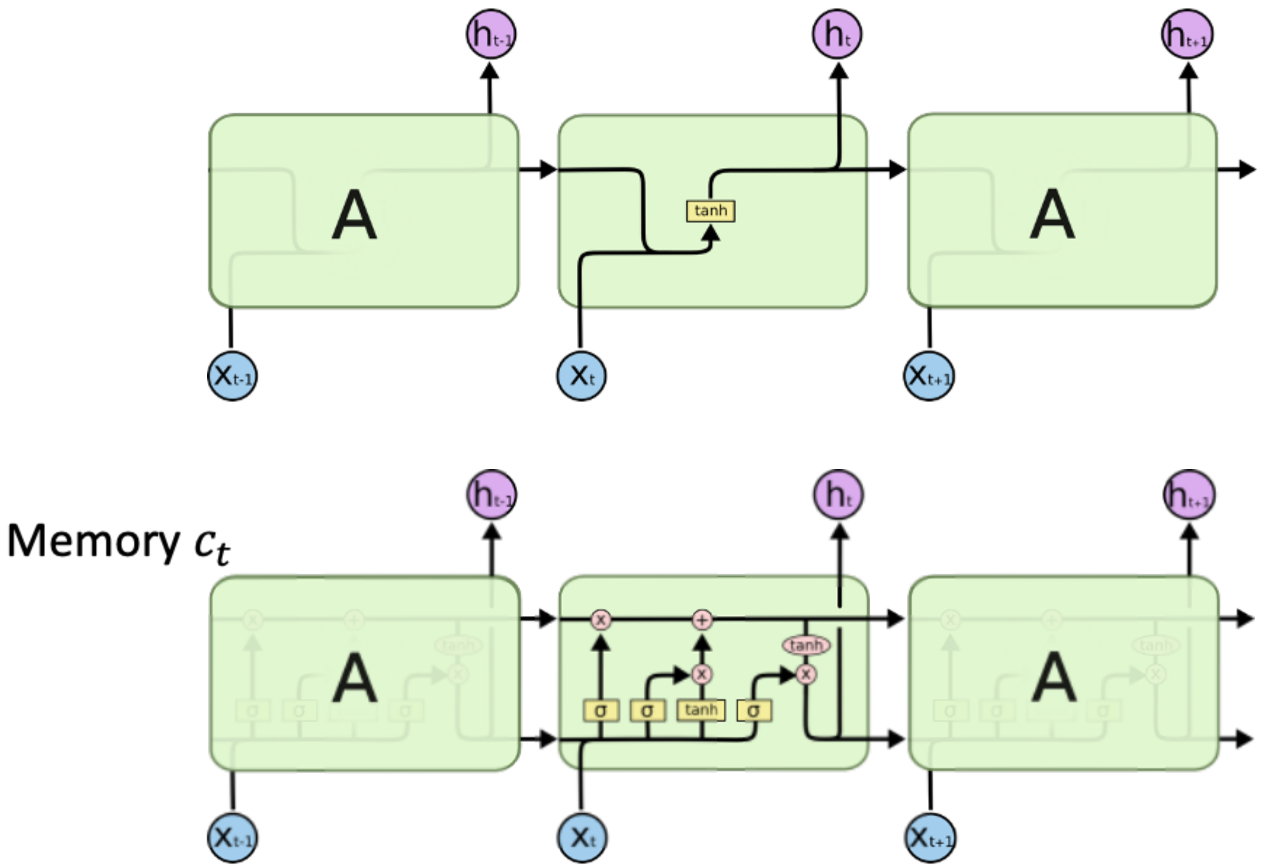
Techniques for avoiding gradient explosion

- Gradient clipping
- Identity initialization
- Truncated backprop through time
  - Only backprop for a few steps



Preserve Long-Term Memory

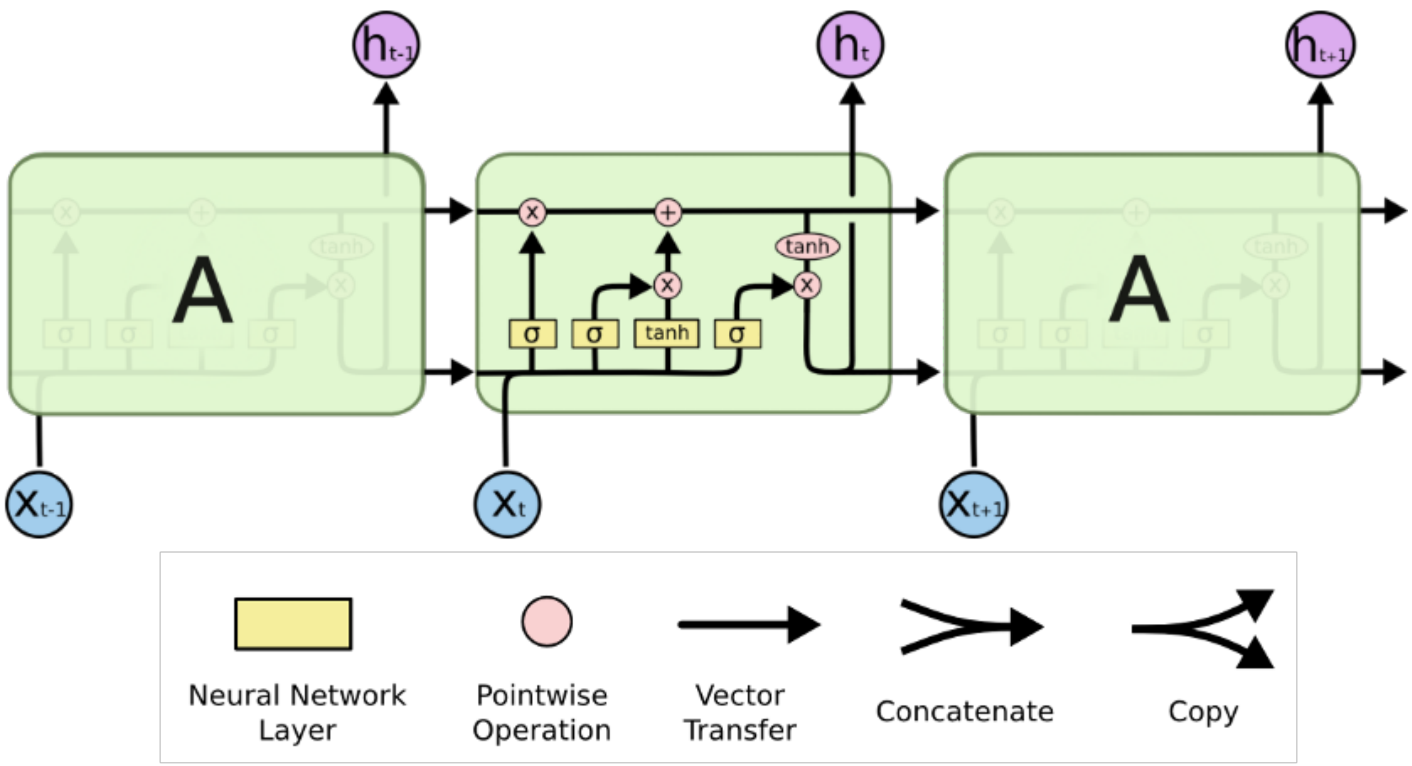
- Difficult for RNN to preserve long-term memory
  - The hidden state h_t is constantly being written (short-term memory)
  - Use a separate cell to maintain long-term memory



Long Short-Term Memory Network

LSTM (Hochreitcher & Schmidhuber, '97)

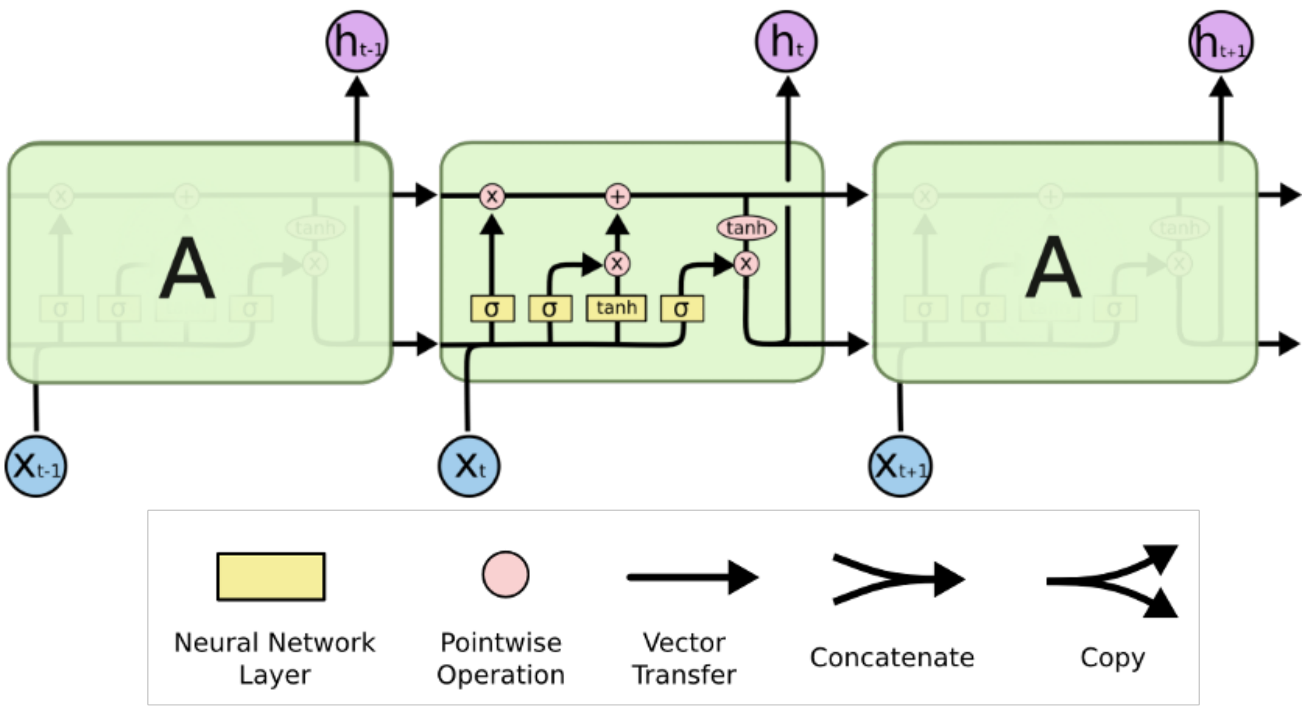
- RNN architecture for learning long-term dependencies
- σ: layer with sigmoid activation



Long Short-Term Memory Network

LSTM (Hochreitcher & Schmidhuber, ’97)

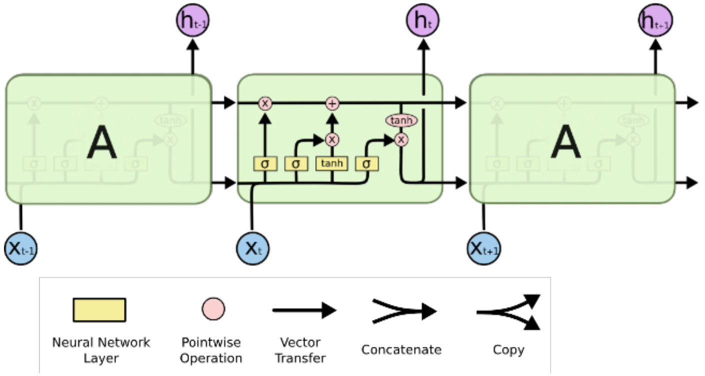
- Core idea: maintain separate state h_t and cell c_t (memory)
- h_t: full update every step
- c_t: only *partially* update through gates
  - σ layer outputs importance ($[0,1]$) for each entry and only modify those entries of c_t

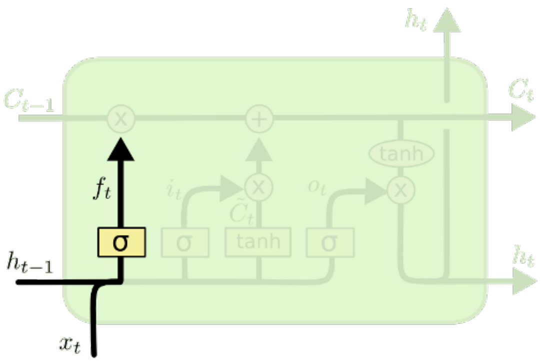


Long Short-Term Memory Network

Forget gate f_t

- f_t outputs whether we want to "forget" things in c_t
  - Compute $c_{t-1} \odot f_t$ (element-wise)
  - $f_t(i) \rightarrow 0$: want to forget $c_t(i)$
  - $f_t(i) \rightarrow 1$: we want to keep the information in $c_t(i)$



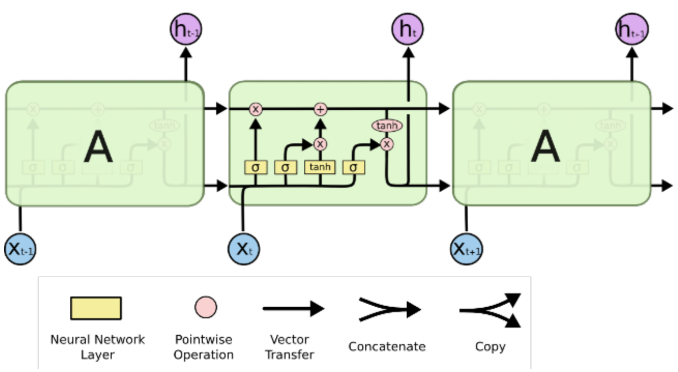


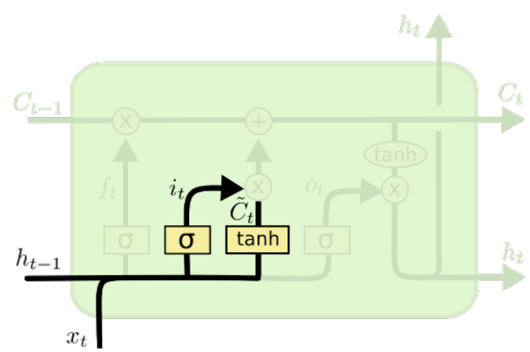
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Long Short-Term Memory Network

Input gate i_t

- i_t extracts useful information from X_t to update memory
  - $\tilde{c}_t$: information from X_t to update memory
  - i_t: which dimension in the memory should be updated by X_t
    - $i_t(j) \to 1$: we want to use the information in $\tilde{c}_t(j)$ to update memory
    - $i_t(t) \to 0$: $\tilde{c}_t(j)$ should not contribute to memory





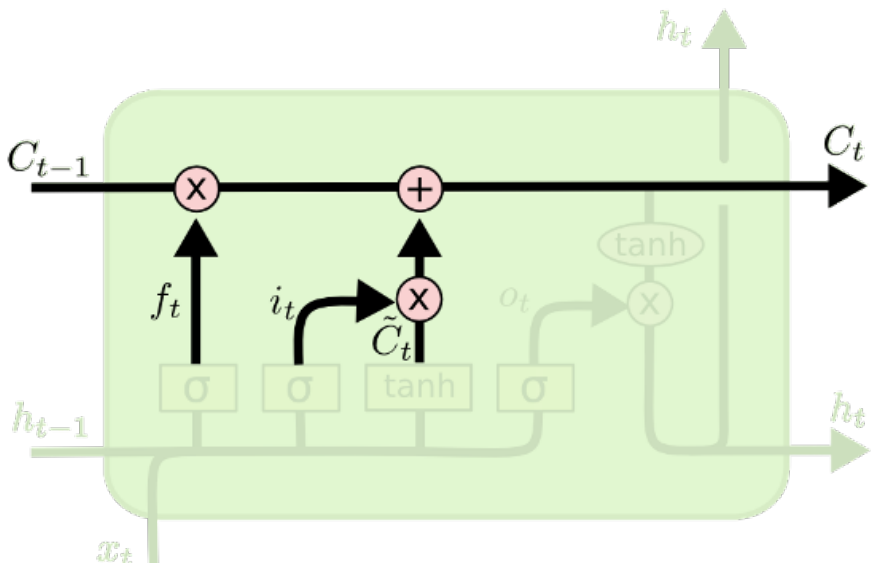
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Long Short-Term Memory Network

Memory update

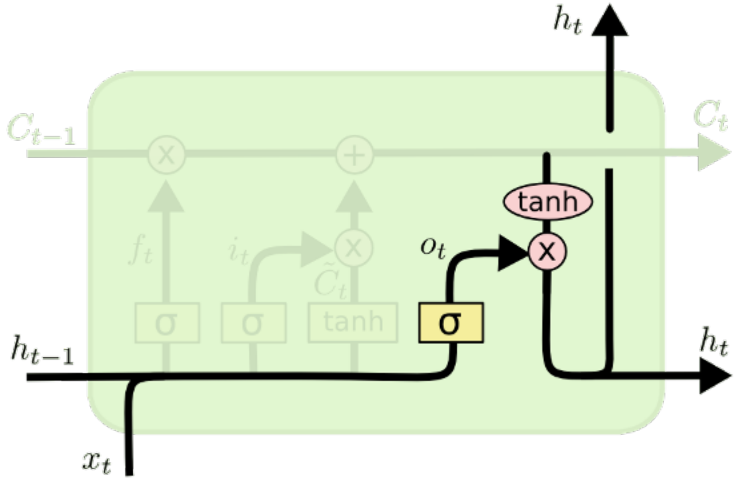
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
- f_t forget gate; i_t input date
- $f_t \odot c_{t-1}$: drop useless information in old memory
- $i_t \odot \tilde{c}_t$: add selected new information from current input



Long Short-Term Memory Network

Output gate o_t

- Next hidden state $h_t = o_t \odot \tanh(c_t)$
  - $\tanh(c_t)$: non-linear transformation over all past information
  - o_t: choose important dimensions for the next state
    - $o_t(j) \to 1$: $\tanh(c_t(j))$ is important for the next state
    - $o_t(j) \to 0$: $\tanh(c_t(j))$ is not important

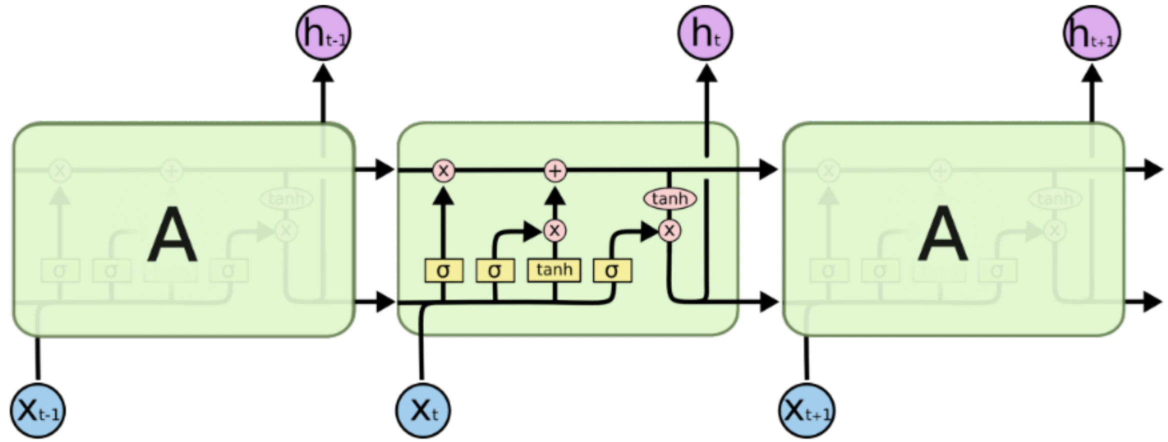


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short-Term Memory Network

- $h_t = o_t \odot \tanh(c_t)$
- $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$
- $Y_t = g(h_t)$



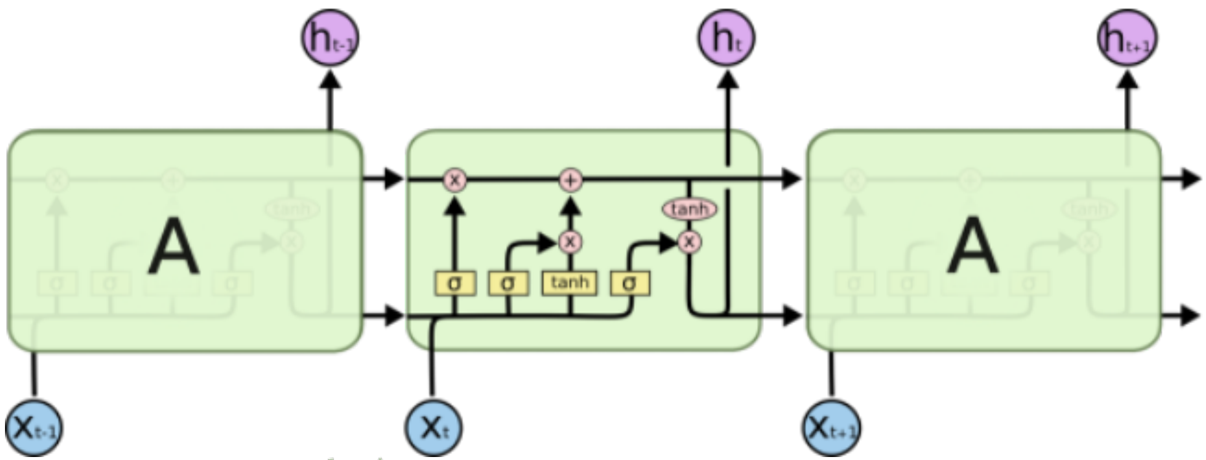
Remarks:

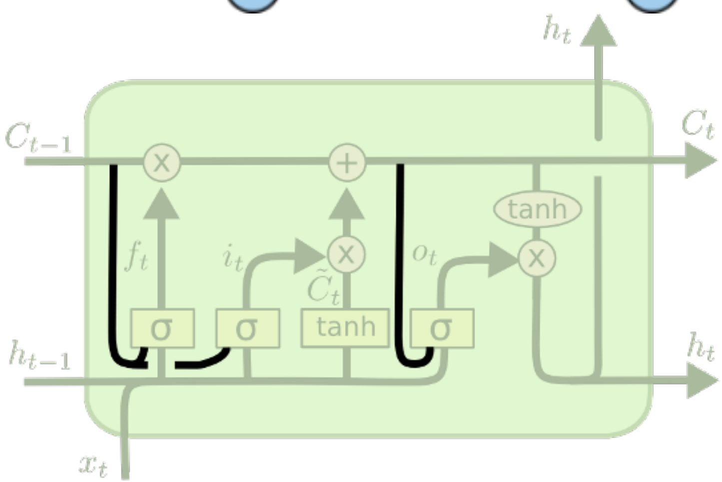
1. No more matrix multiplications for c_t
2. LSTM does not have guarantees for gradient explosion/vanishing
3. LSTM is the dominant architecture for sequence modeling from ’13 - ’16.
4. Why tanh

LSTM Variant

Peephold Connections (Gers & Schmidhuber '00)

- Allow gates to take in c_t information





$$f_t = \sigma\left(W_f \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$

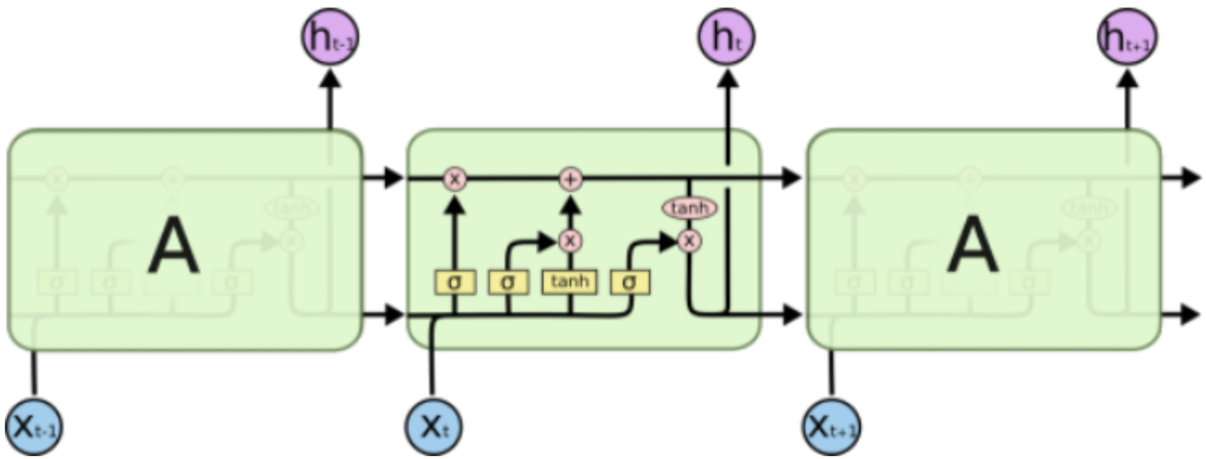
$$i_t = \sigma\left(W_i \cdot [\mathbf{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$

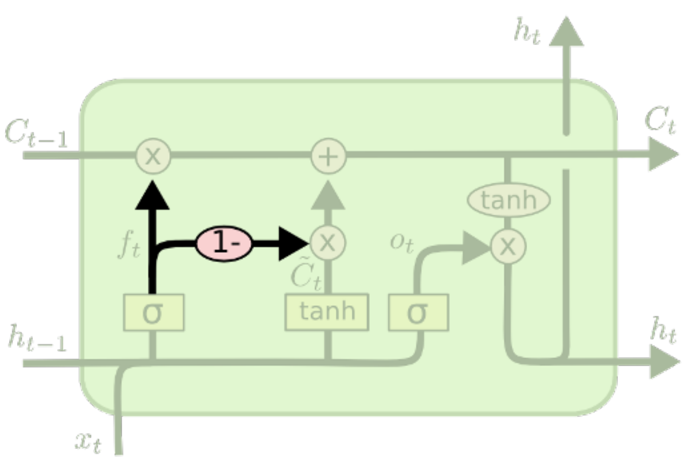
$$o_t = \sigma\left(W_o \cdot [\mathbf{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

LSTM Variant

Simplified LSTM

- Assume $i_t = 1 - f_t$
- Only two gates are needed: fewer parameters



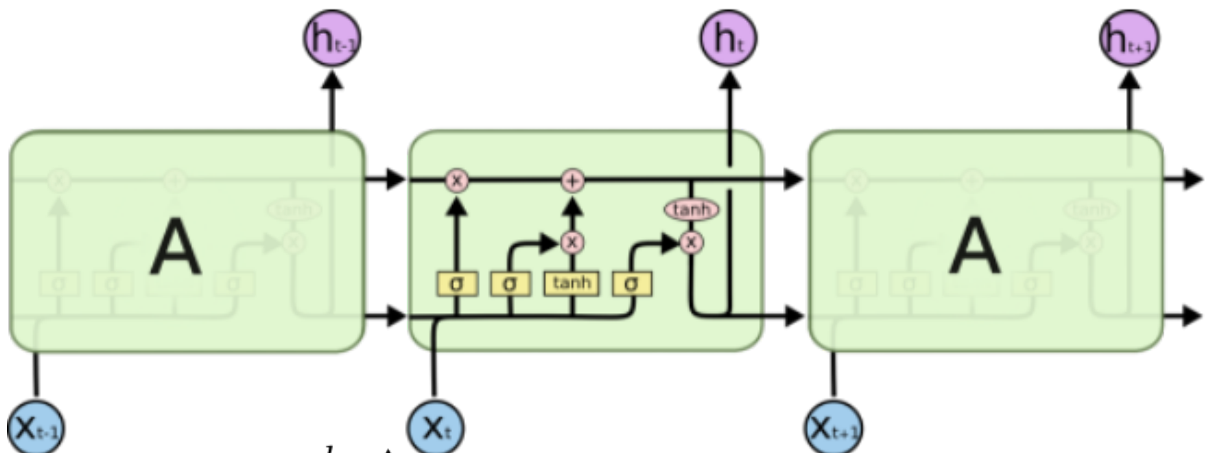


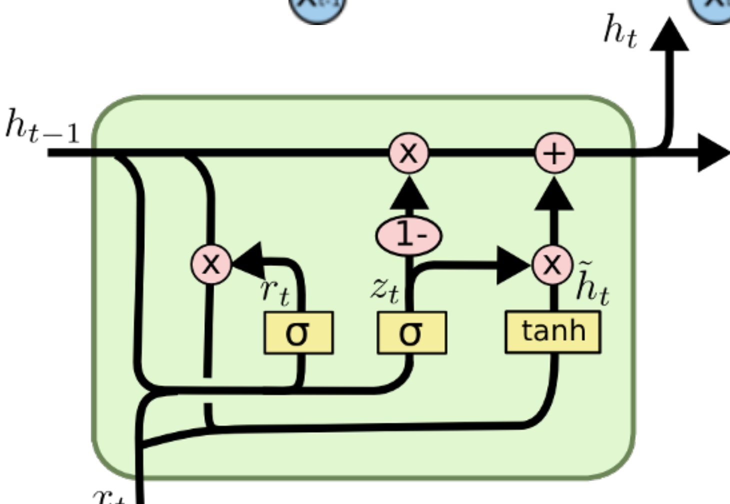
$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

LSTM Variant

Gated Recurrent Unit (GRU, Cho et al. ’14)

- Merge h_t and c_t: much fewer parameters





$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

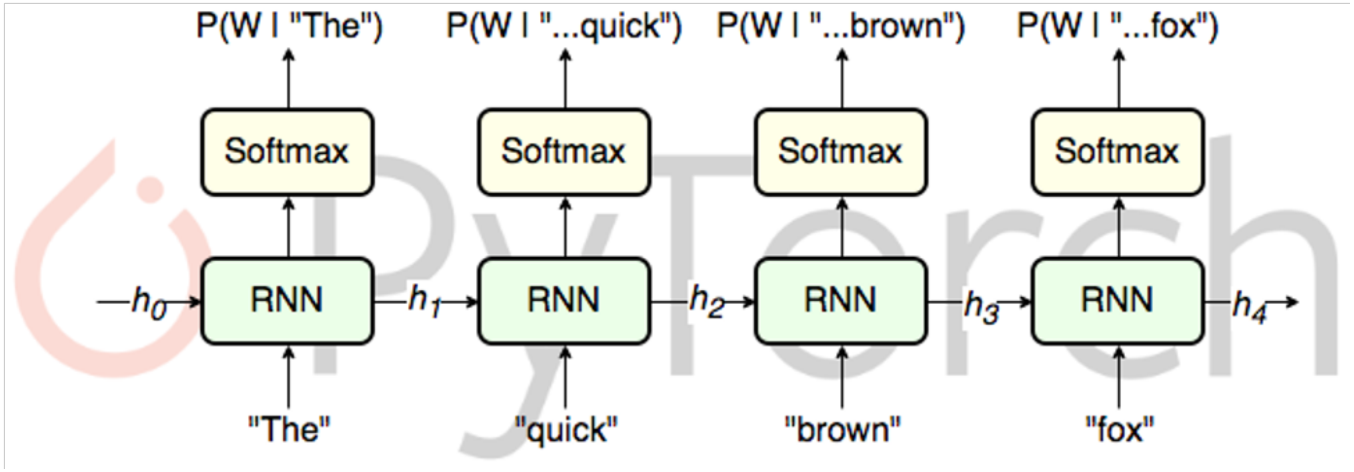
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM application: language model

- Autoregressive language model: $P(X;\theta) = \Pi_{t=1}^{L} P(X_t \mid X_{i<t};\theta)$
  - X: a sentence
  - Sequential generation
- LSTM language model
  - X_t: word at position t.
  - Y_t: softmax over all words
- Data: a collection of texts:
  - Wiki



LSTM application: text classification

Bi-dreictional LSTM and them run softmax on the final hidden state.