CHULA ENGINEERING
Foundation toward Innovation

COMPUTER

Chula Big Data and IoT
Center of Excellence
(CUBIC)

Data Mining Group
Machine Intelligence and Knowledge Discovery Lab
Chulalongkorn University

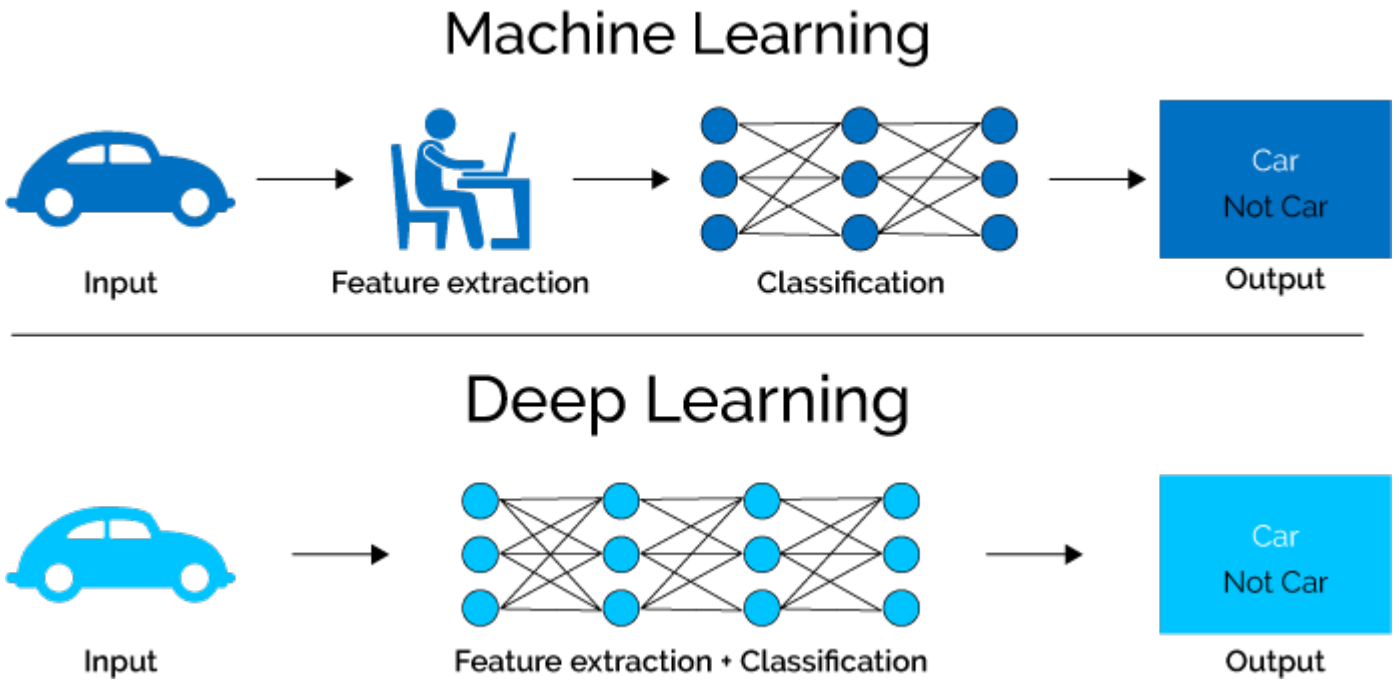
Practical Deep Learning
Python for Data Analytics

Peerapon Vateekul, Ph.D.
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University
Peerapon.v@chula.ac.th

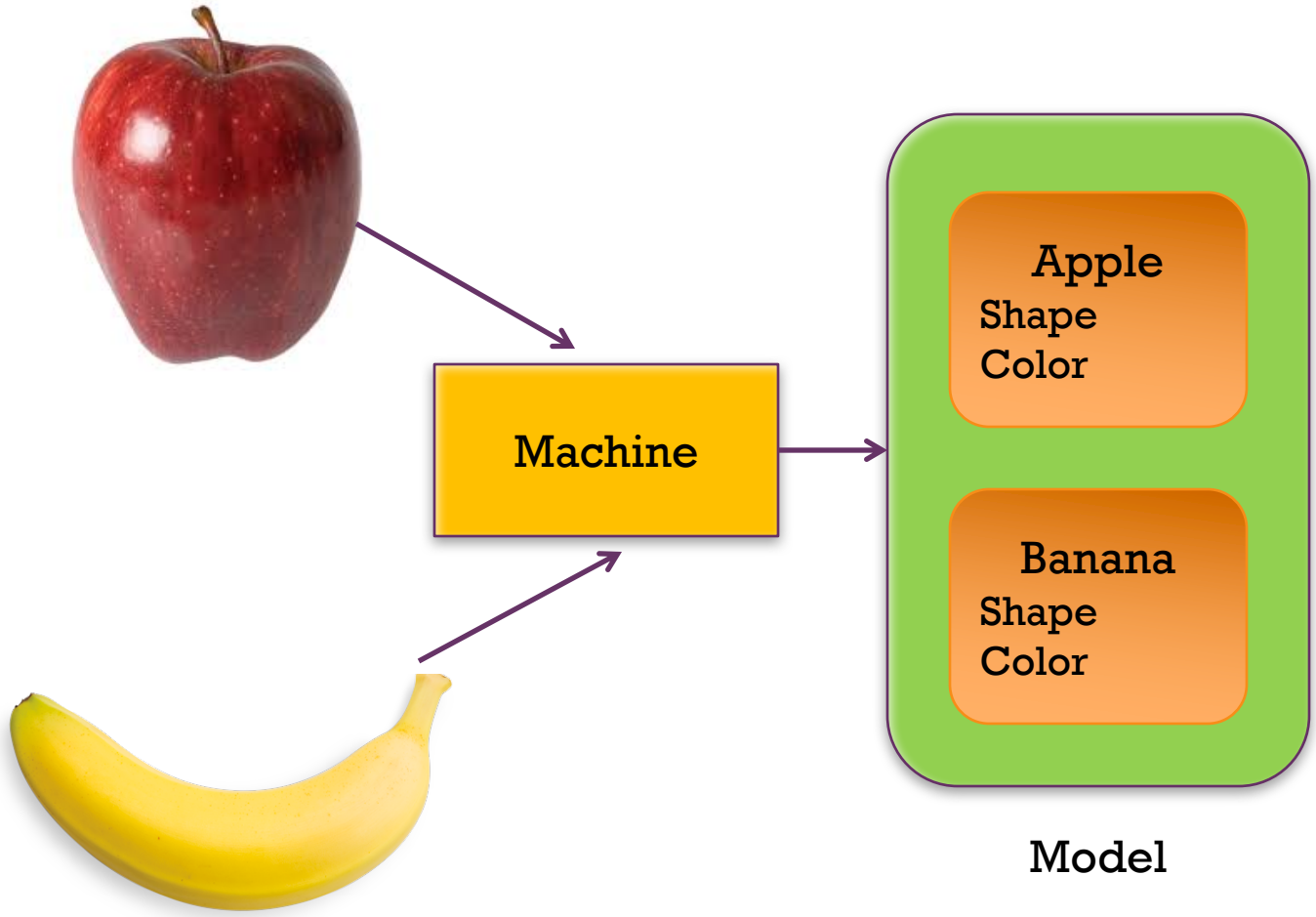
Outlines

- Introduction to Deep Learning
- Convolutional Neural Network (CNN)
- Word Embedding
- Long-Short Term Memory (LSTM)
- Current Research in Deep Learning

Introduction to Deep Learning



Traditional Supervised Learning

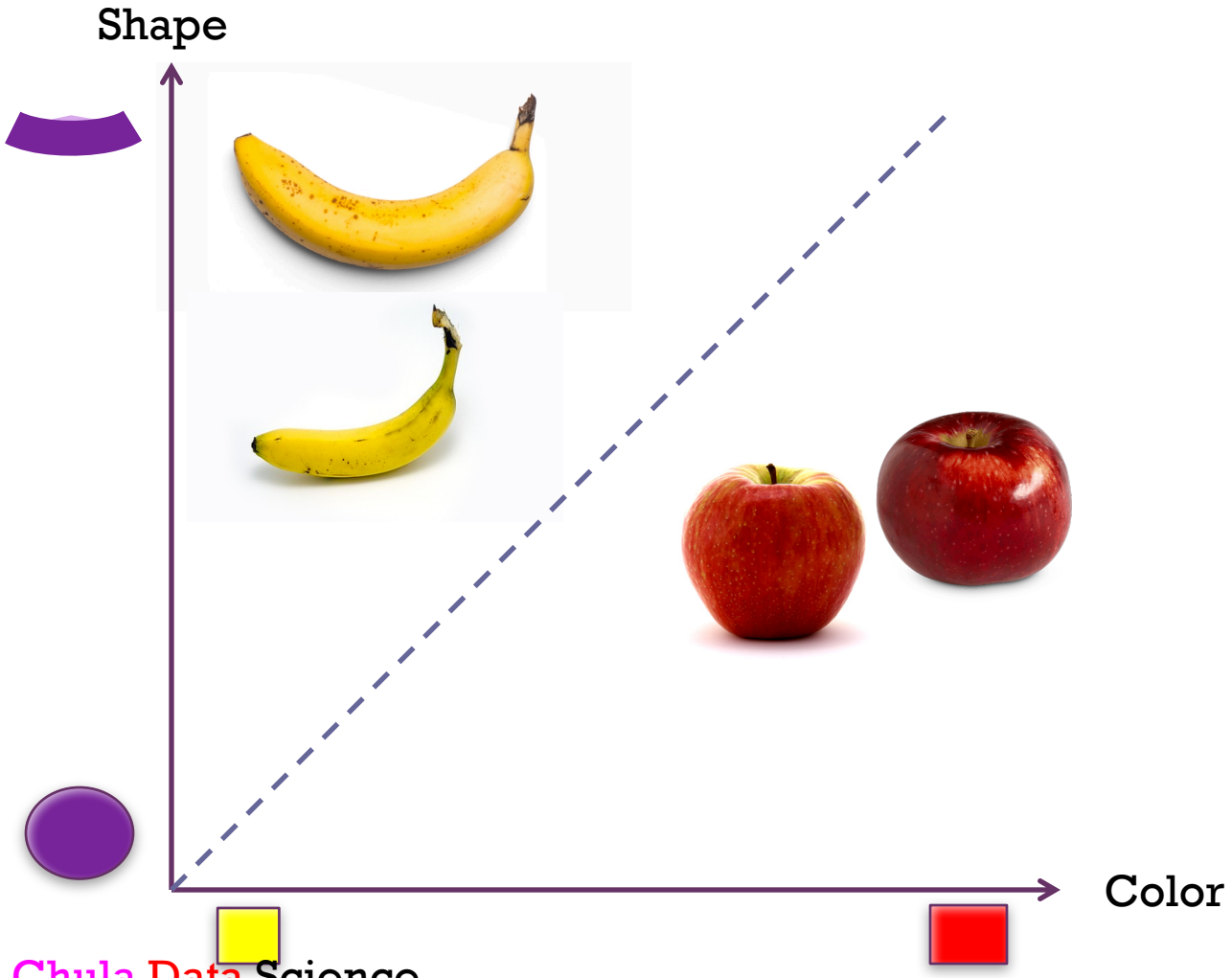


Supervised Learning

- Learn from labeled examples
- Example: Learn to classify apples from bananas
- Model: Criteria for classification

1. Training Phase
2. Testing Phase

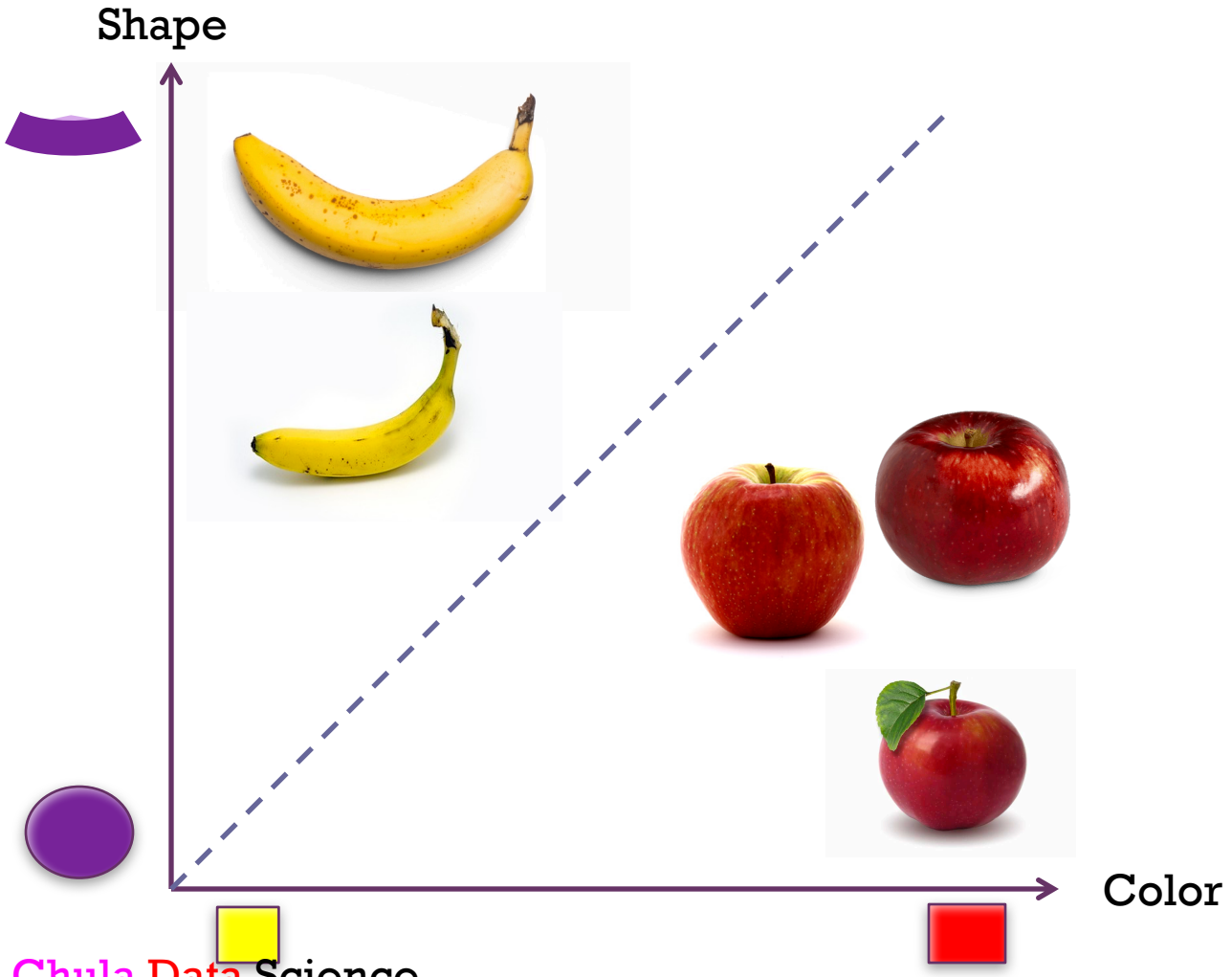
Traditional Supervised Learning
Training Phase → classification model



Supervised Learning

- Learn from labeled examples
- Example: Learn to classify apples from bananas
- Model: Criteria for classification

Traditional Supervised Learning
Testing Phase: case 1



Supervised Learning

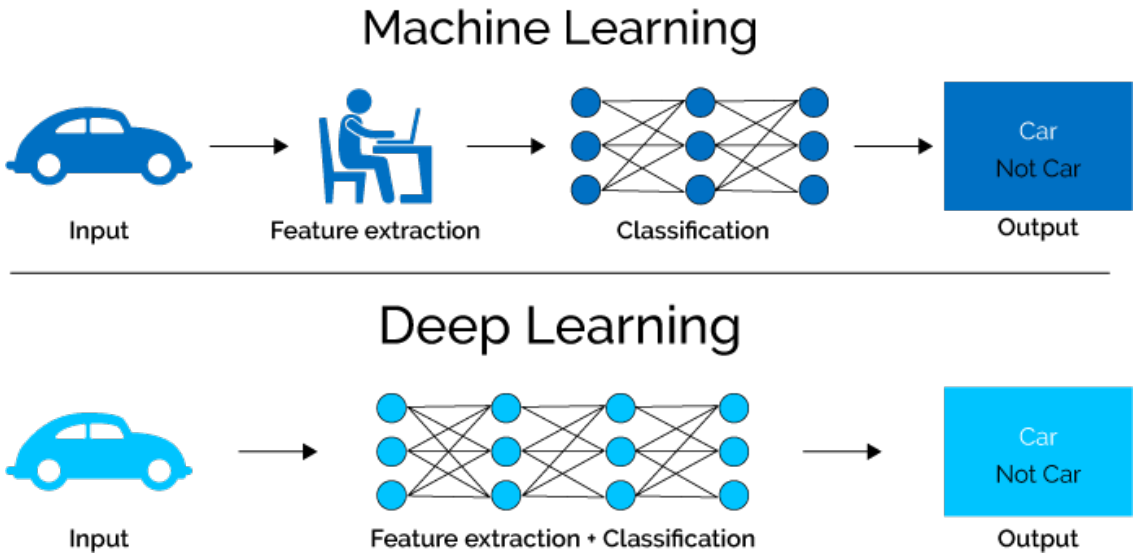
- Learn from labeled examples
- Example: Learn to classify apples from bananas
- Model: Criteria for classification

Feature is the most important factor!

What is Deep Learning?

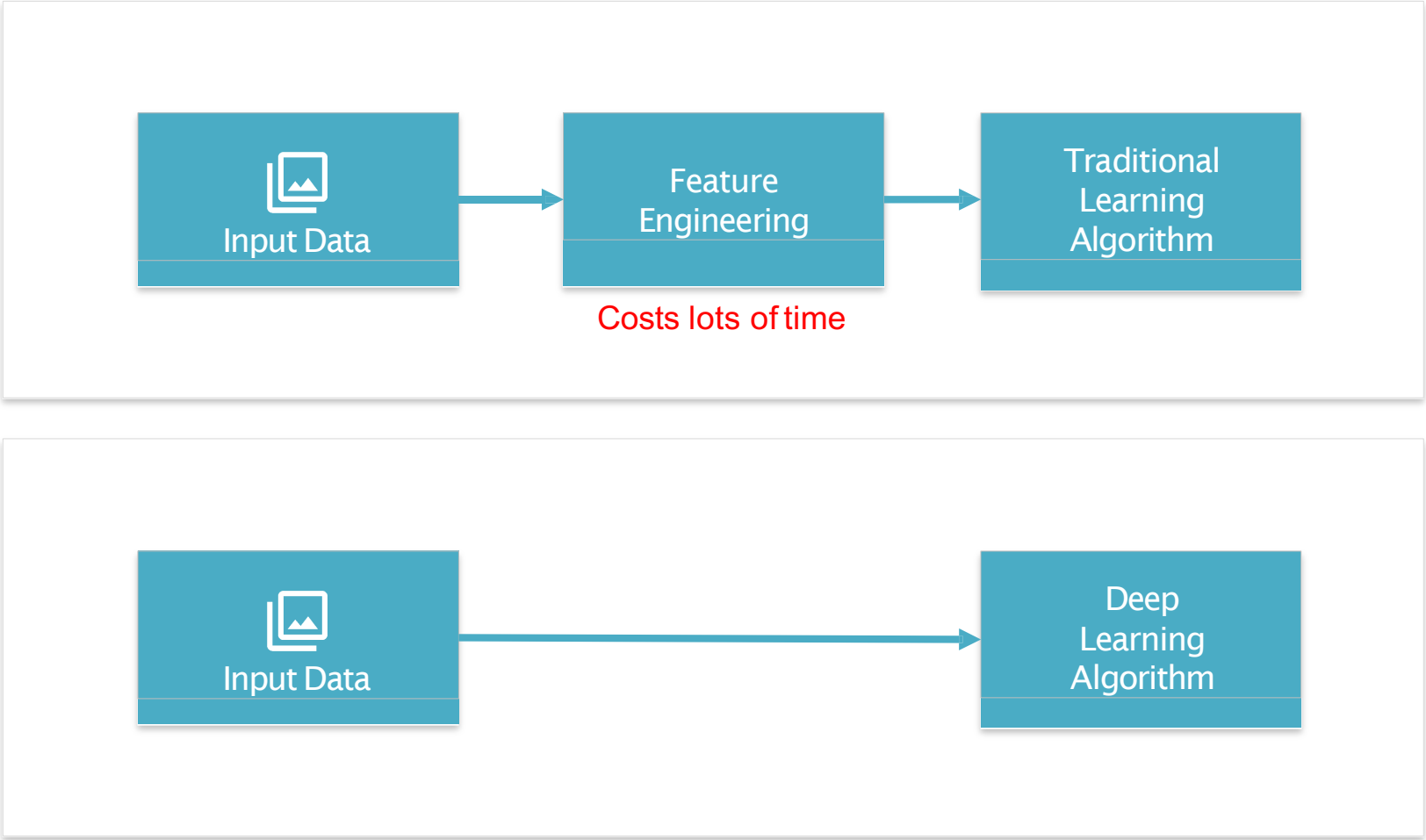
Part of the machine learning field of learning representations of data. Exceptional effective at learning patterns.

Utilizes learning algorithms that derive meaning out of data by using a hierarchy of multiple layers that mimic the neural networks of our brain.



Deep Learning - Basics

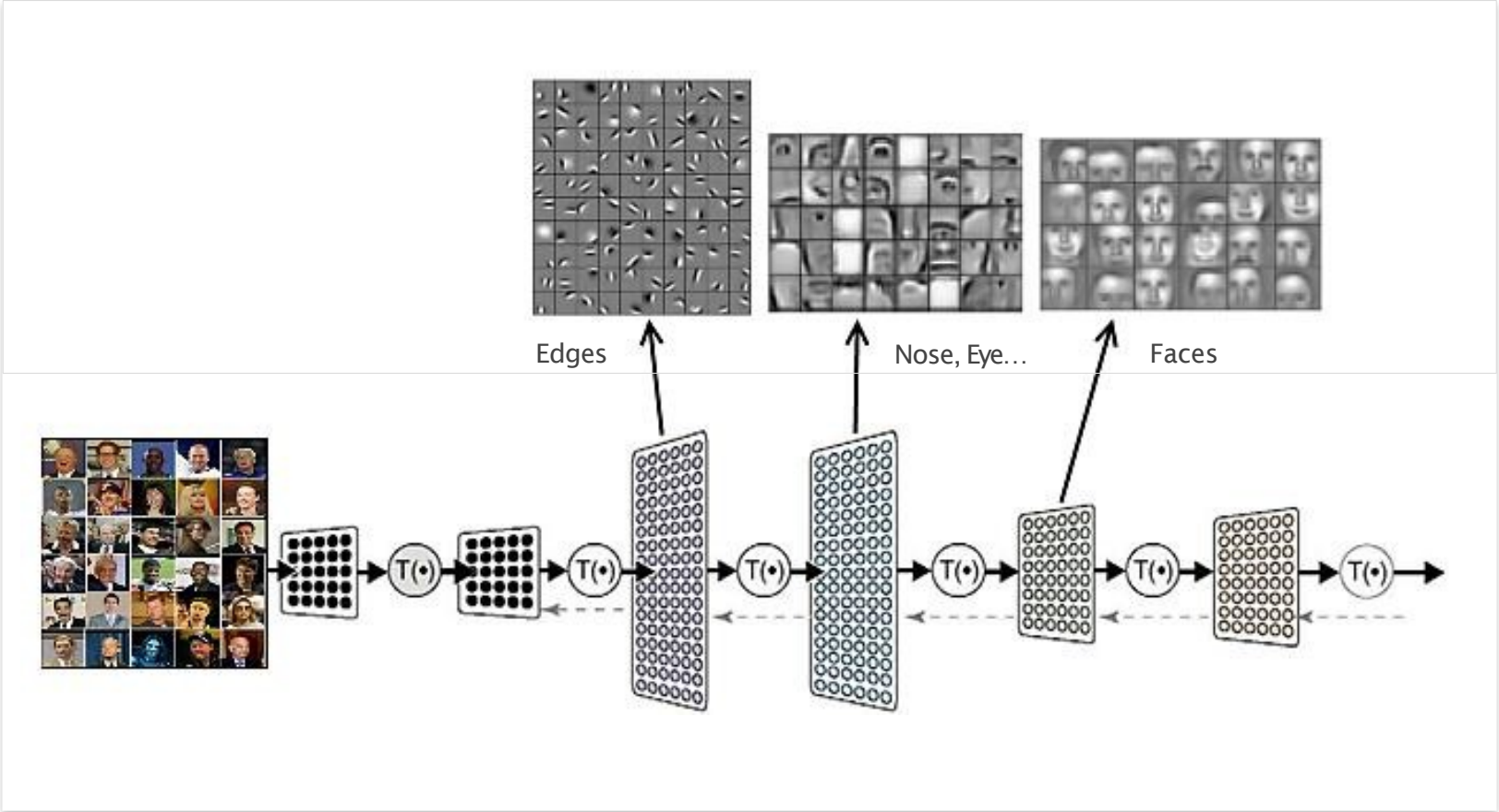
No more feature engineering



Deep Learning – Basics (cont.)

What did it learn?

A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations (e.g. edge -> nose -> face). The output layer combines those features to make predictions.

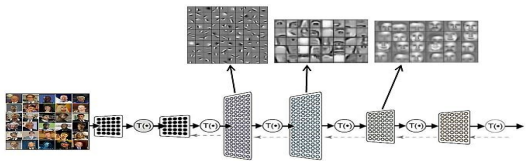


Deep Learning Application

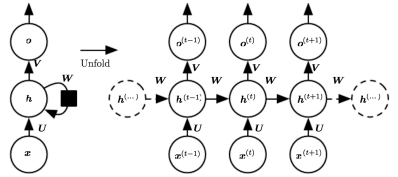
Speech Recognition

Computer Vision

Natural Language Processing

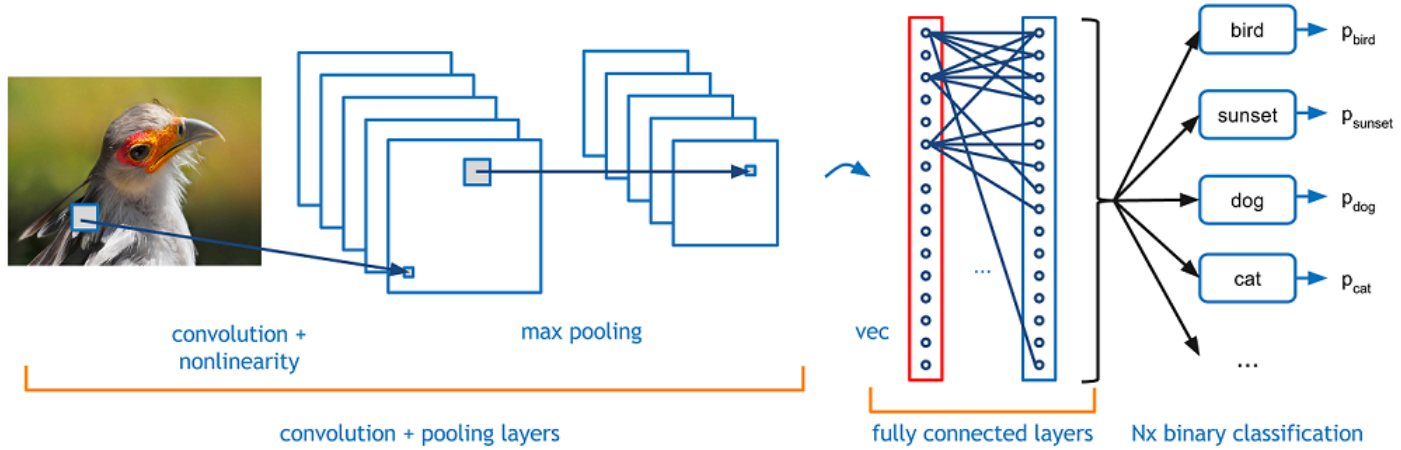


CNN



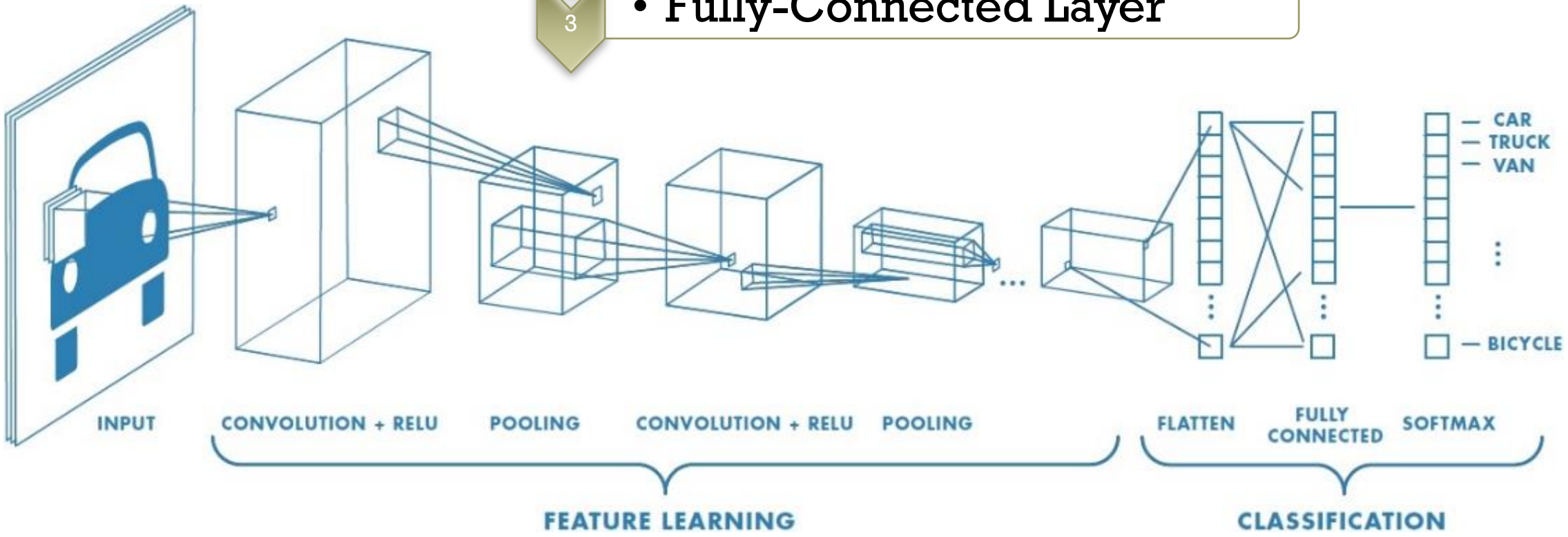
RNN (LSTM)

Convolutional Neural Network (CNN)

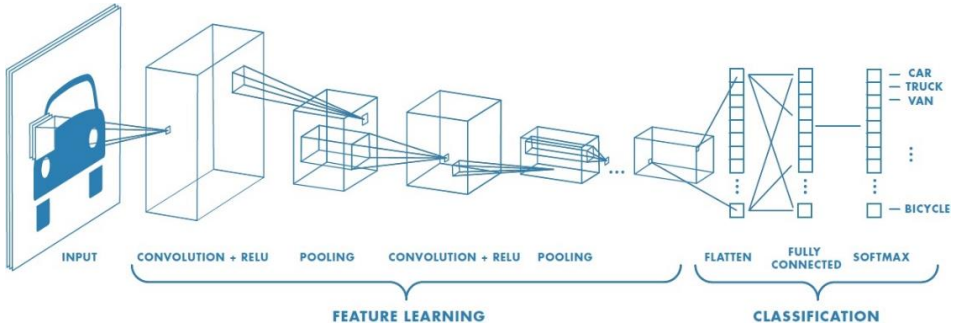


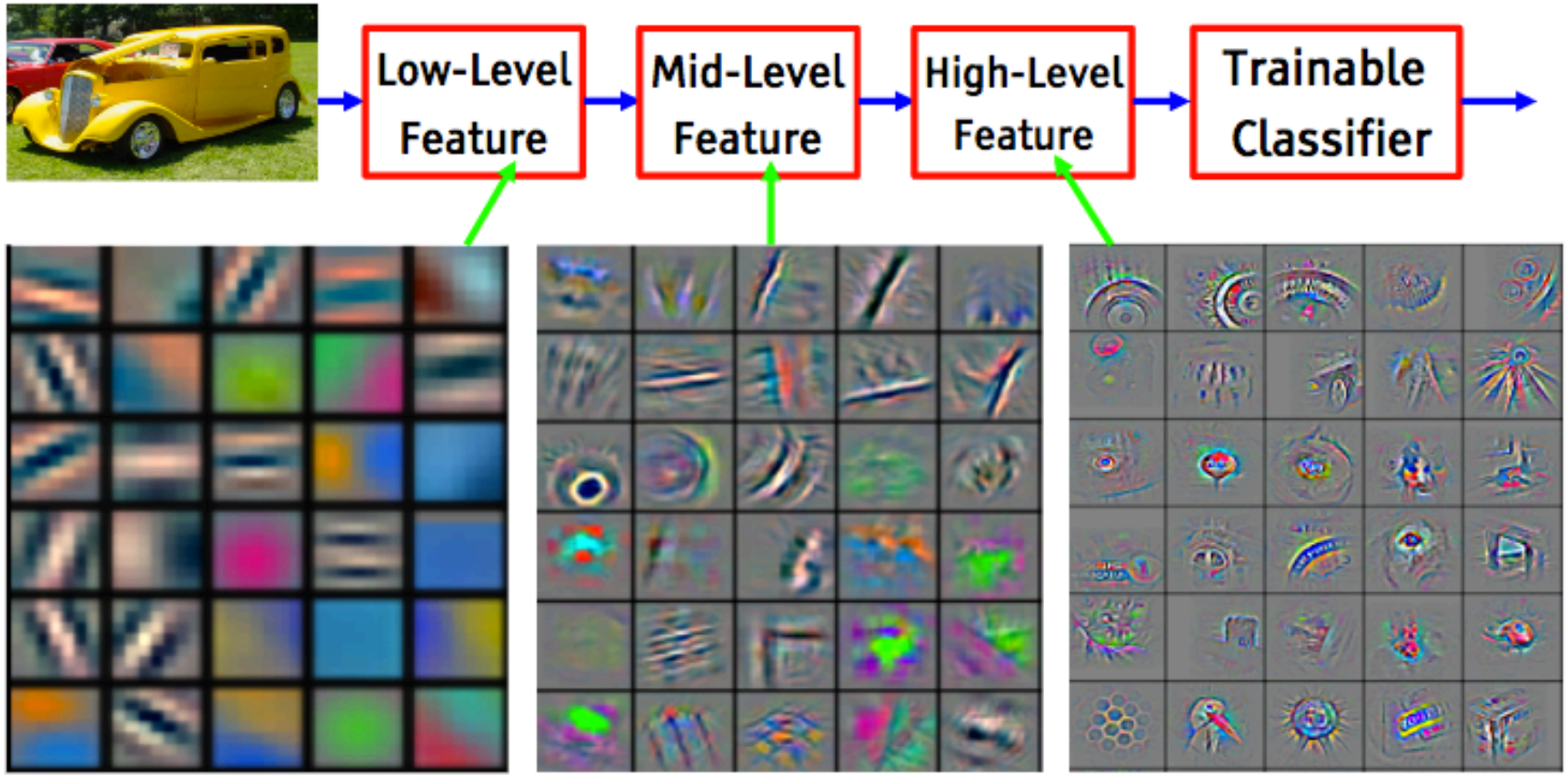
Deep Learning – CNN (Overview)

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer



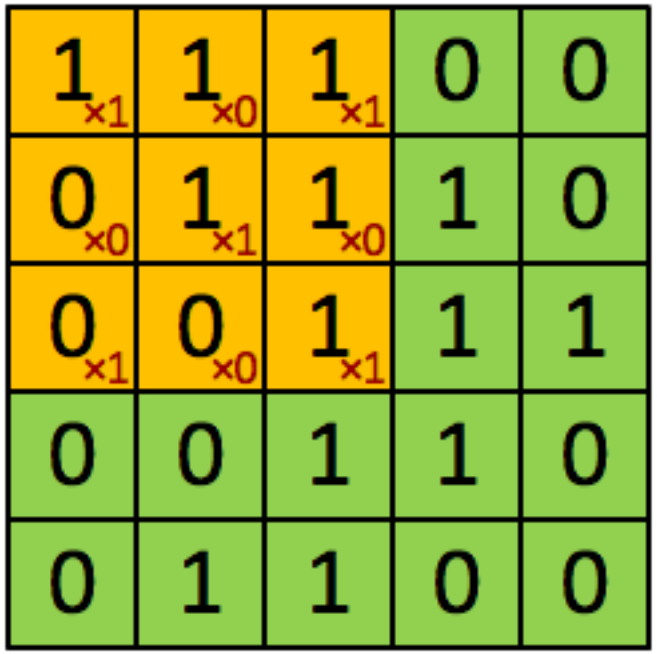
Deep Learning – CNN (Overview)



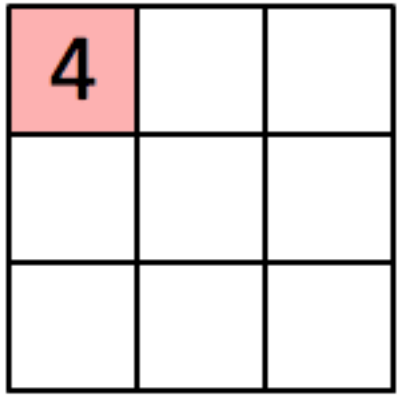


Layer 1: Convolutional Layer

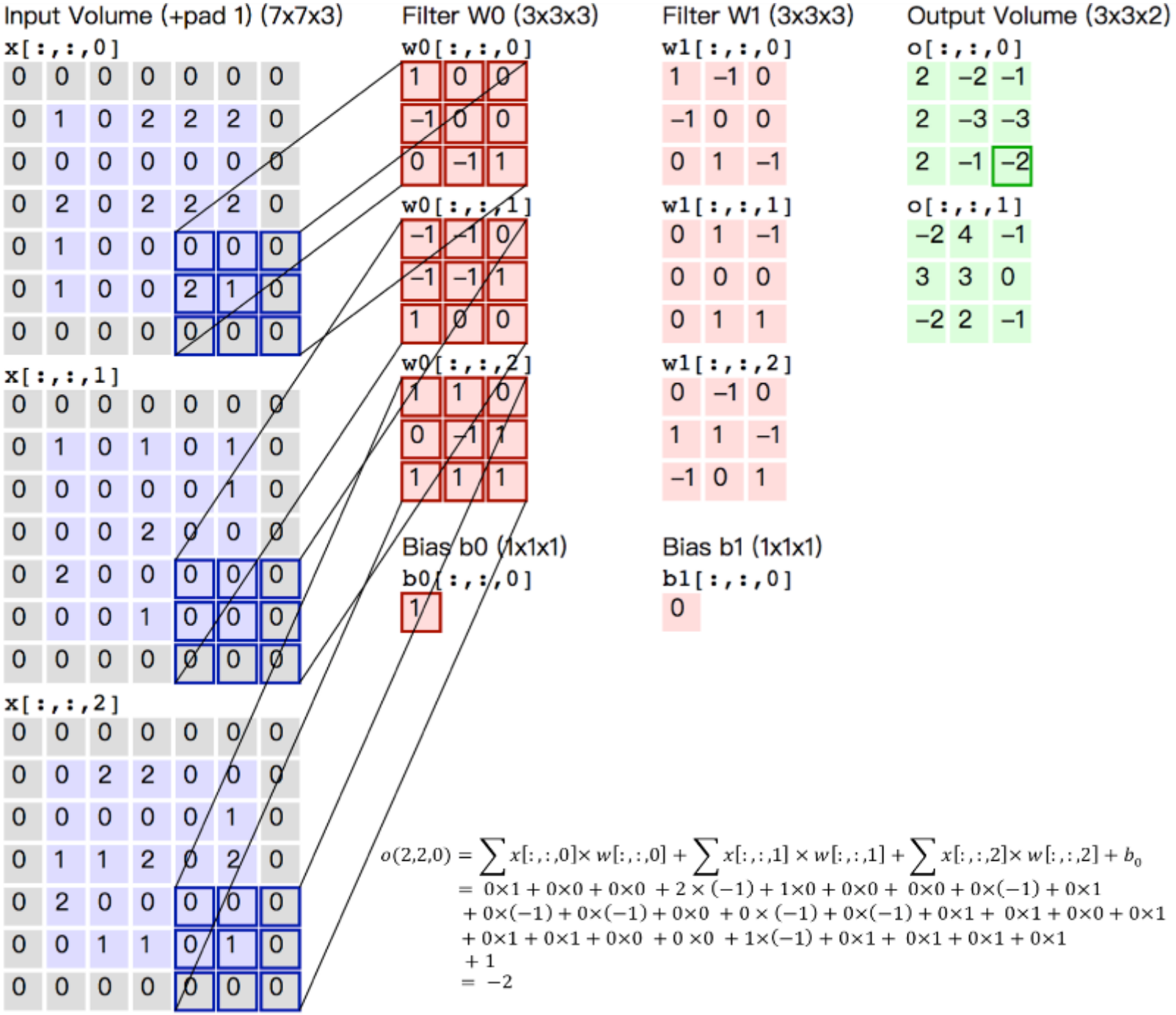
1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer



Image



Convolved Feature

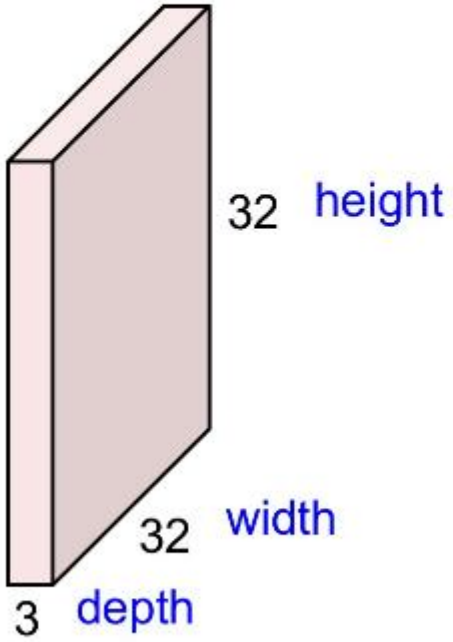


$$
\begin{aligned}
o(2,2,0) &= \sum x[:,:,0]\times w[:,:,0] + \sum x[:,:,1]\times w[:,:,1] + \sum x[:,:,2]\times w[:,:,2] + b_0 \\
&= 0\times1 + 0\times0 + 0\times0 \;+ 2\times(-1) + 1\times0 + 0\times0 + \;0\times0 + 0\times(-1) + 0\times1 \\
&\quad + 0\times(-1) + 0\times(-1) + 0\times0 \;+ 0\times(-1) + 0\times(-1) + 0\times1 + \;0\times1 + 0\times0 + 0\times1 \\
&\quad + 0\times1 + 0\times1 + 0\times0 \;+ 0\times0 \;+ 1\times(-1) + 0\times1 + \;0\times1 + 0\times1 + 0\times1 \\
&\quad + 1 \\
&= -2
\end{aligned}
$$

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer

Convolution Layer

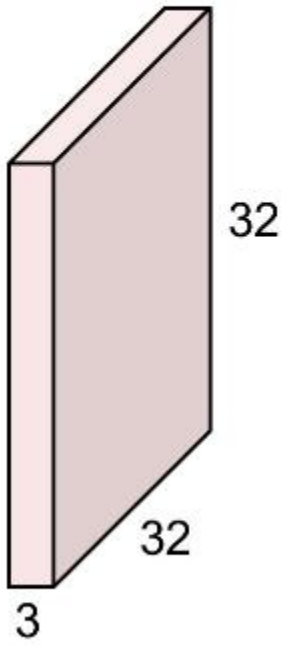
32x32x3 image -> preserve spatial structure



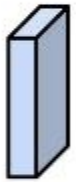
Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Convolution Layer

32x32x3 image



5x5x3 filter



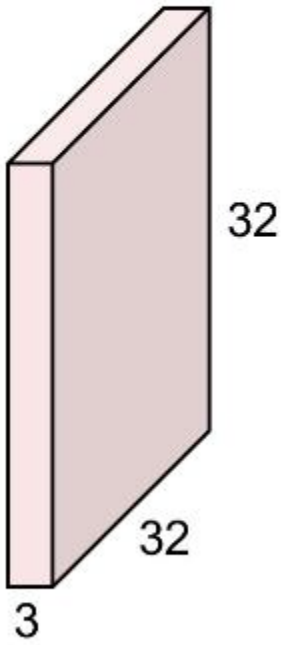
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

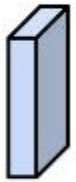
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



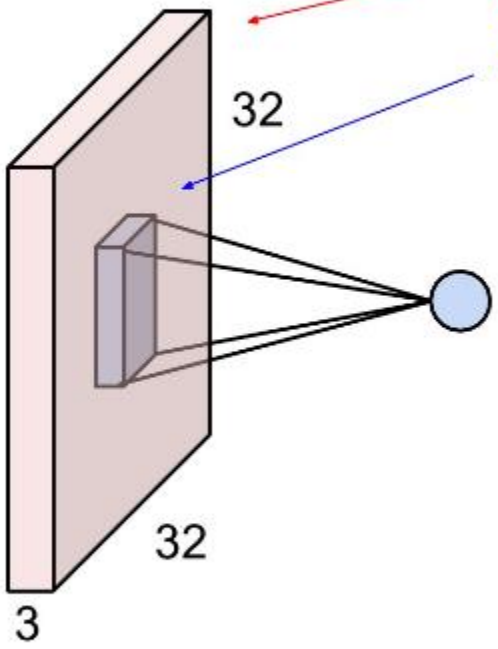
5x5x3 filter



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

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Convolution Layer

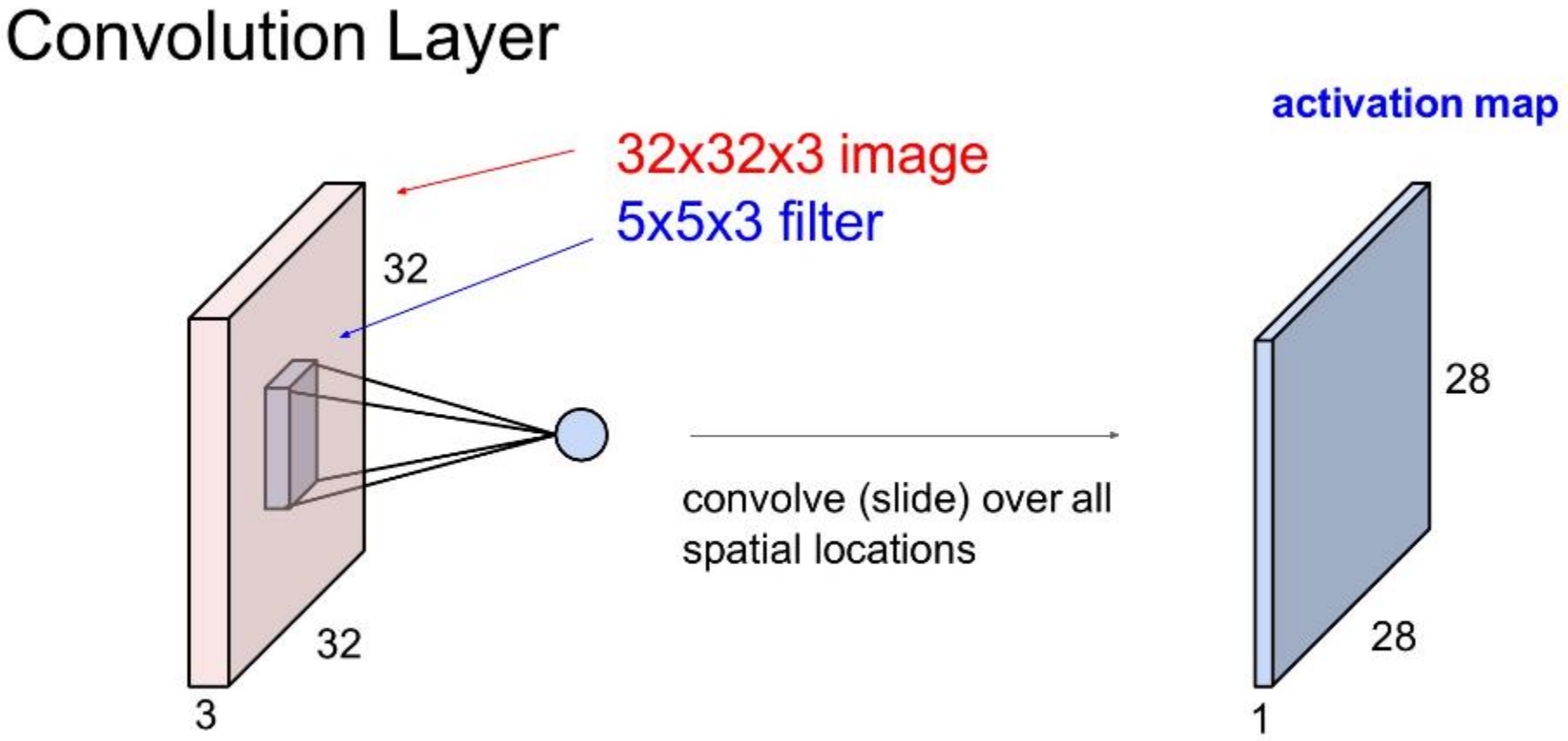


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

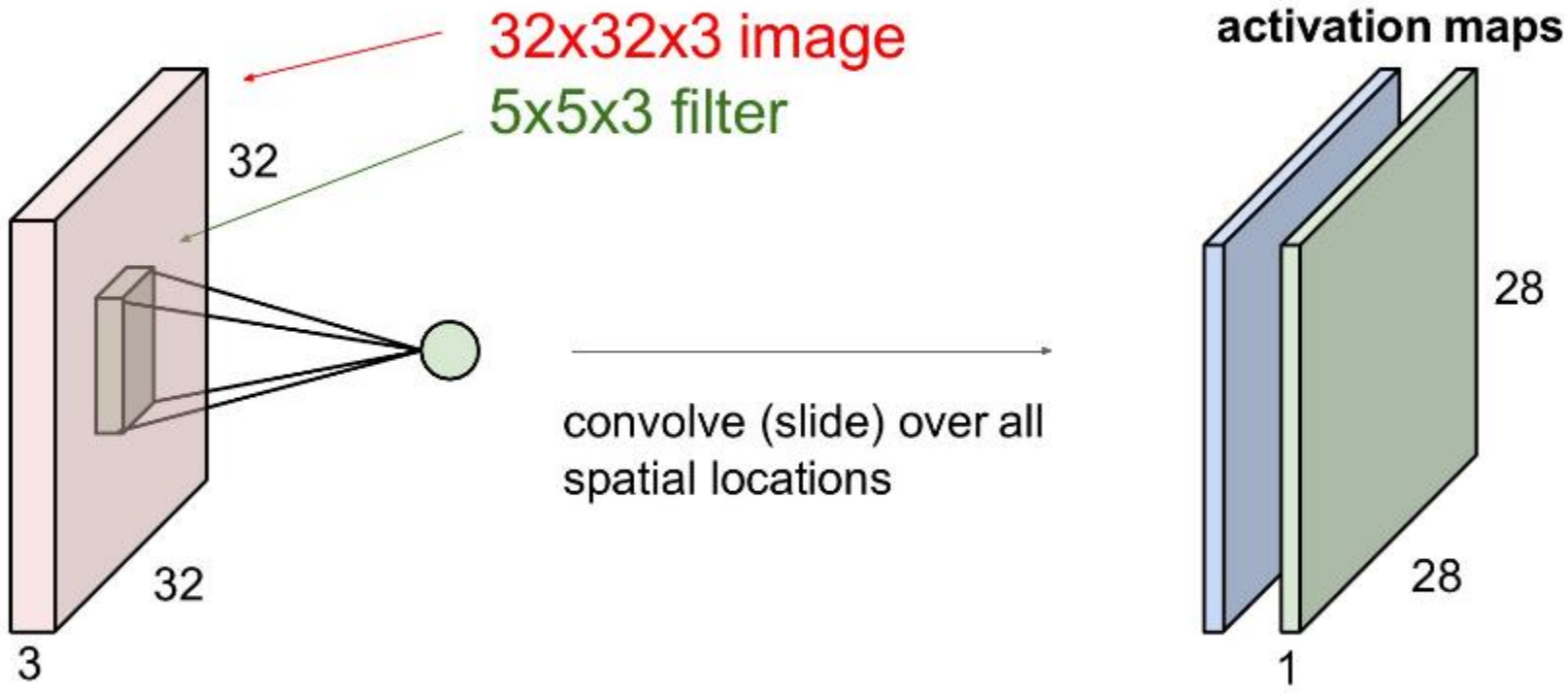
One convolutional filter generates one feature map (activation map).



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

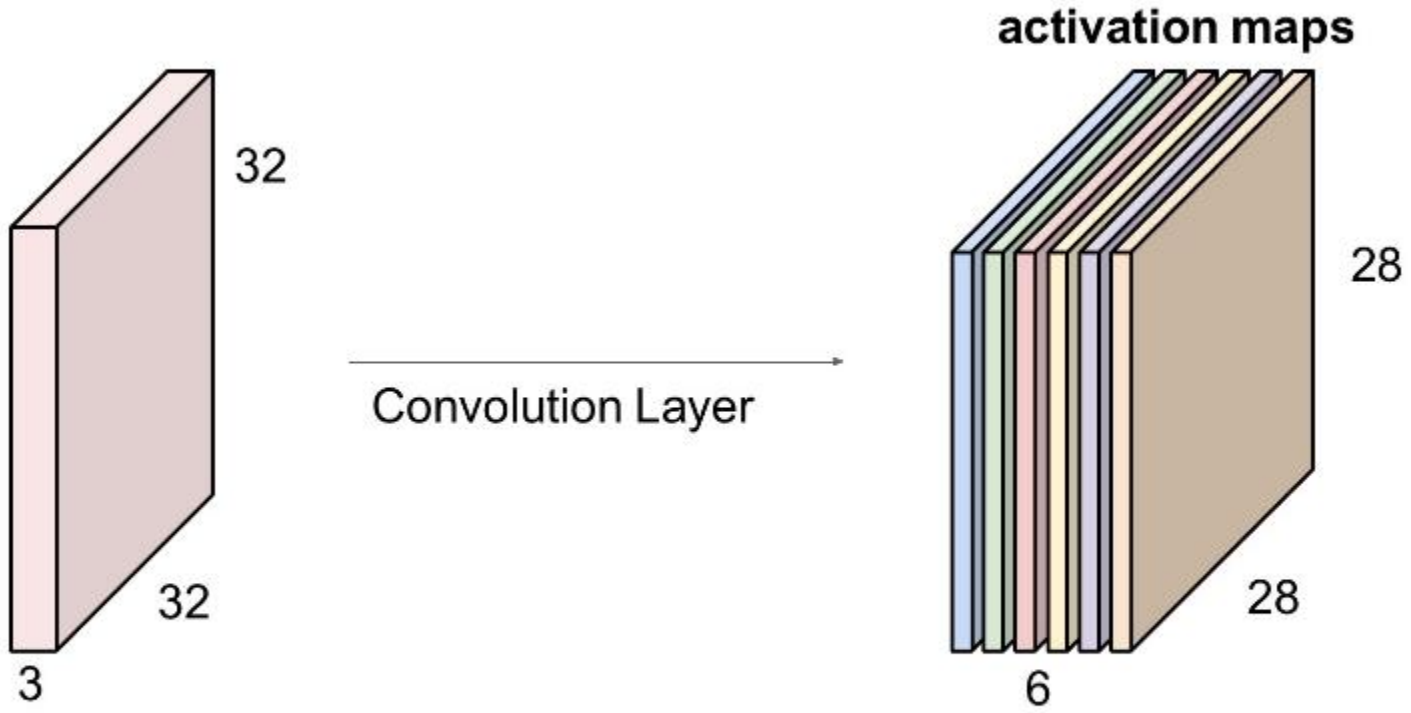
Convolution Layer

consider a second, green filter



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

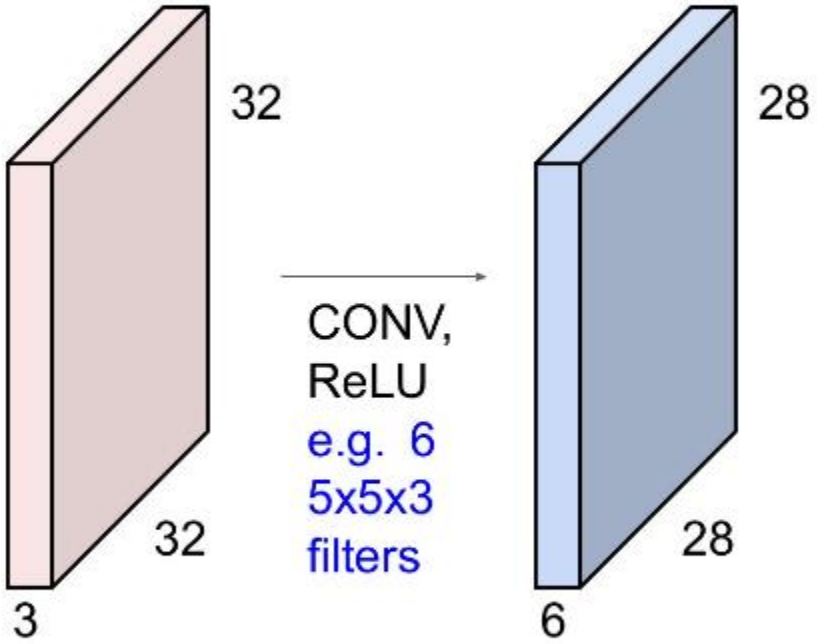


We stack these up to get a "new image" of size 28x28x6!

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

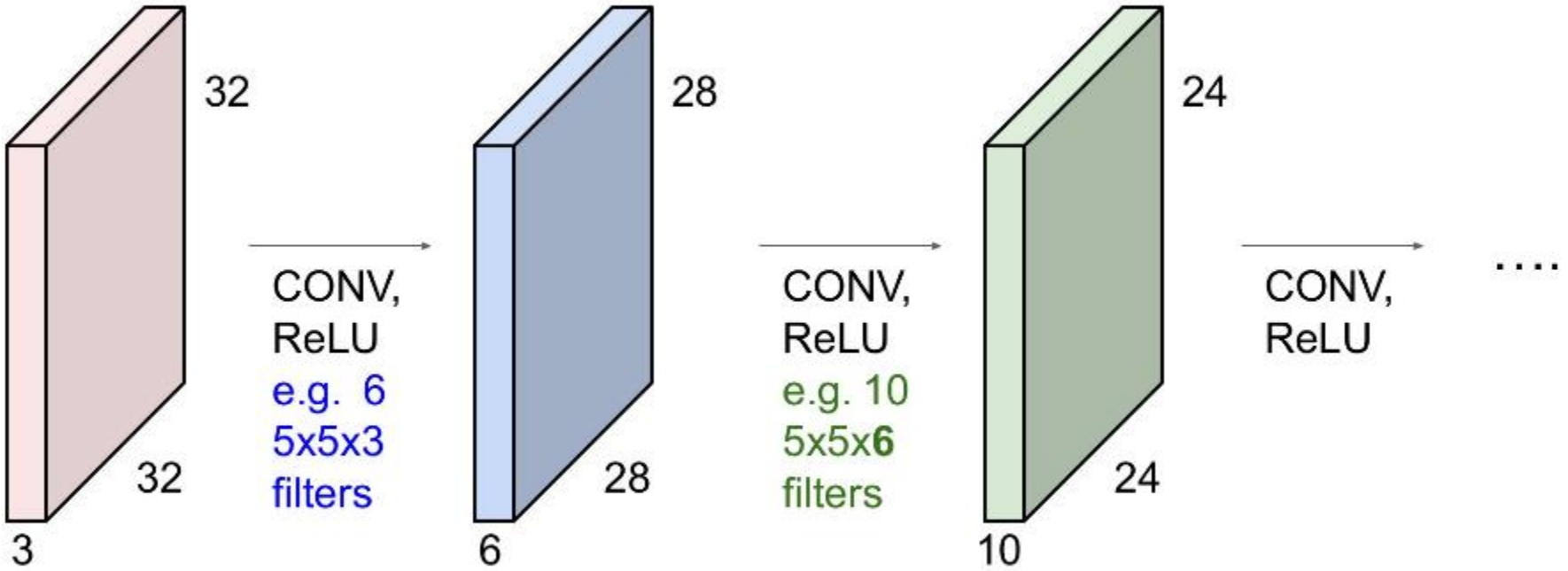
Each feature map must transformed using "activation function" (ReLU).

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



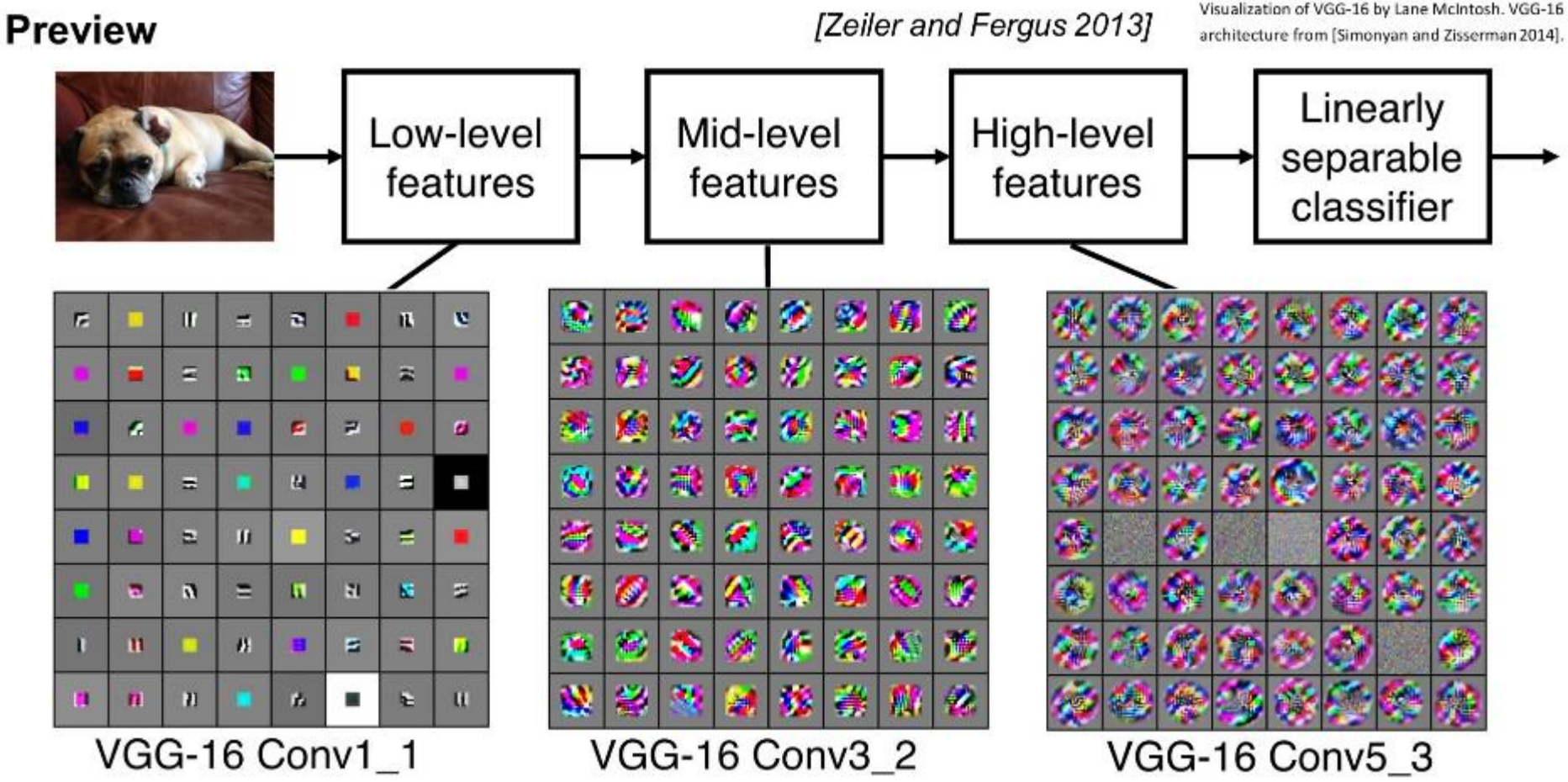
Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

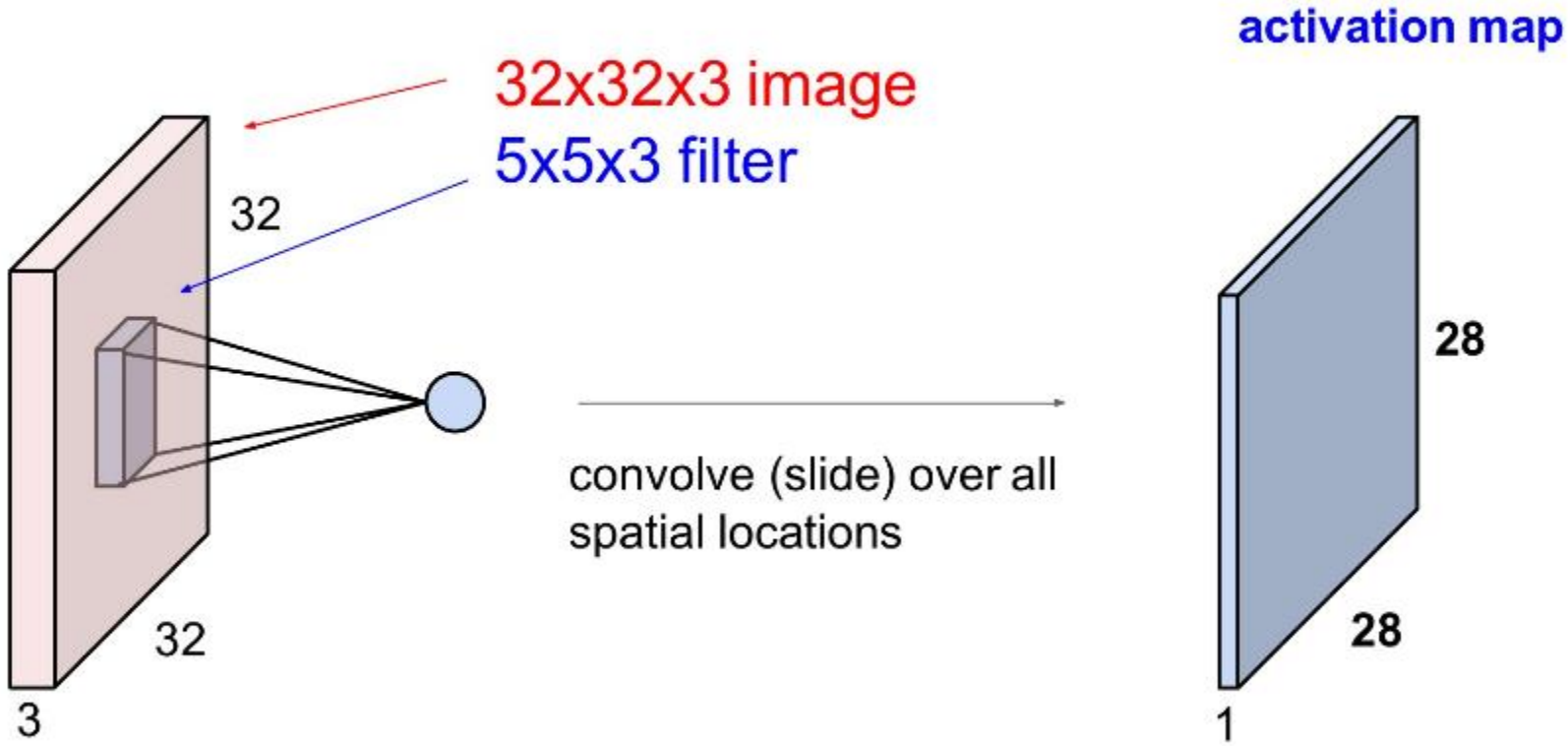
Pre-trained weights from VGG-16 (CNN with 16 layers)



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

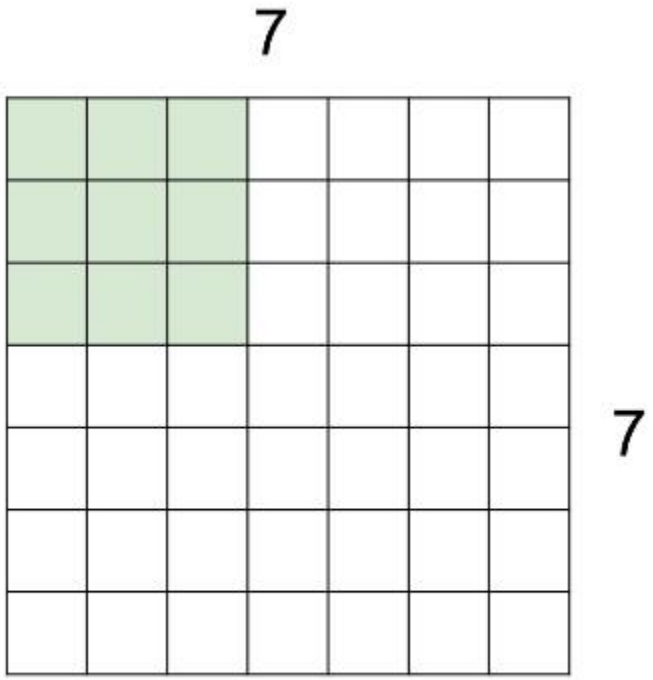
Consider the size of feature maps (original image & filter)

A closer look at spatial dimensions:



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

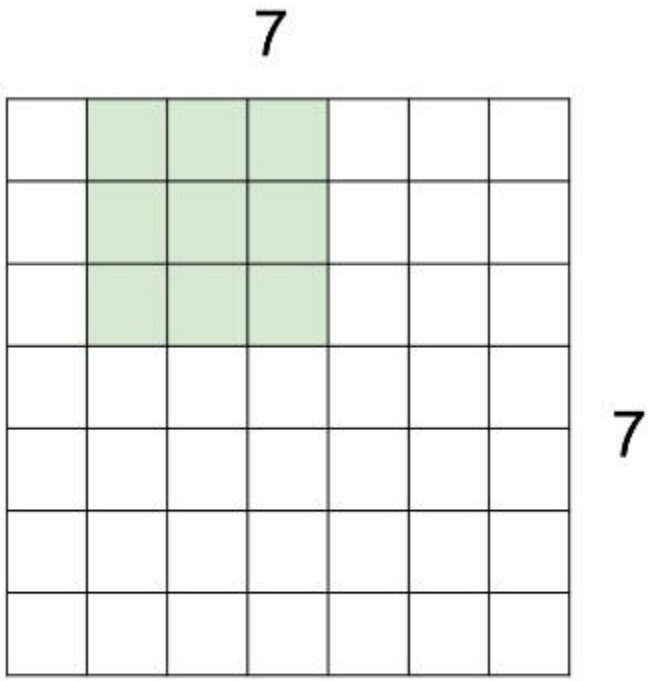
A closer look at spatial dimensions: Stride=1



7x7 input (spatially)
assume 3x3 filter

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

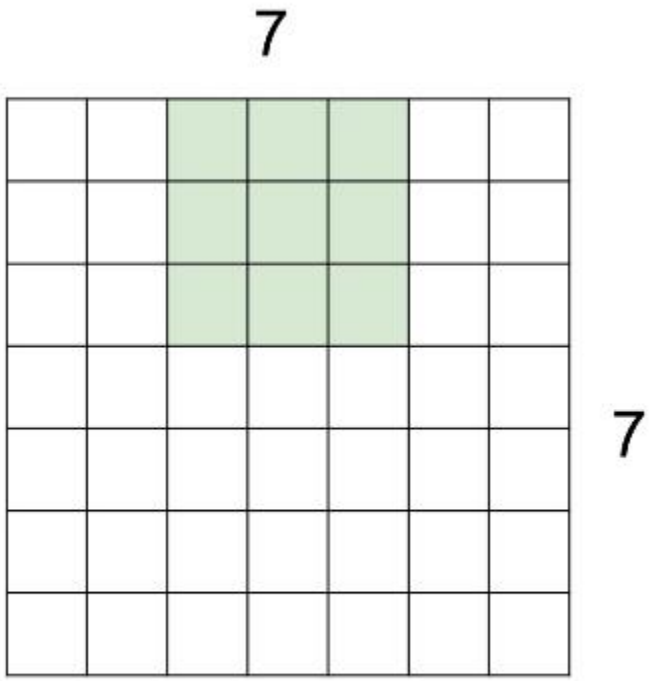
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

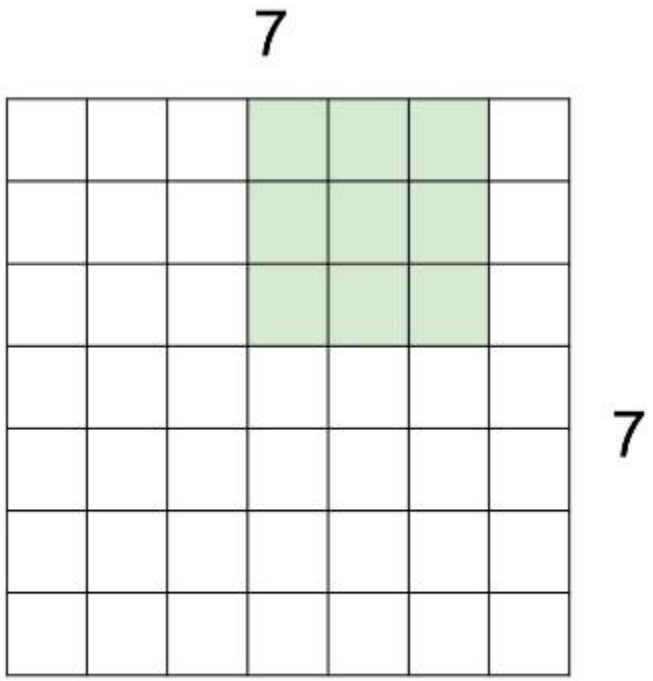
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

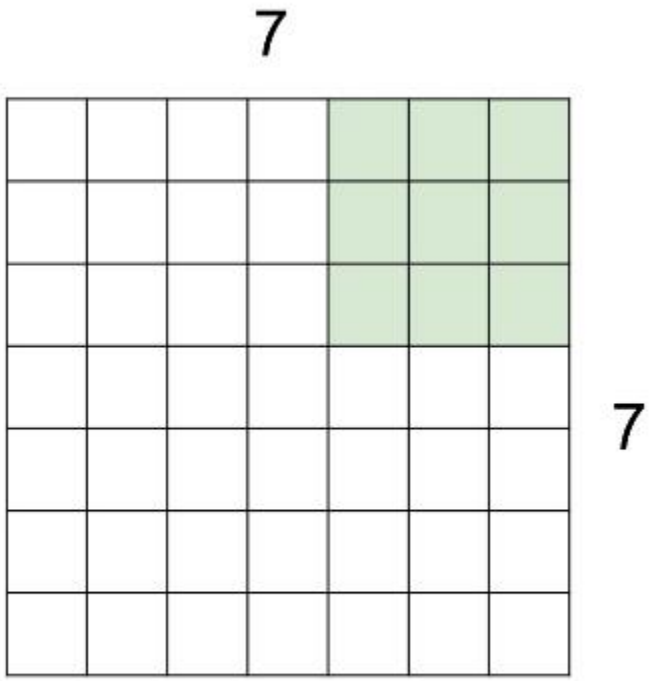
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

A closer look at spatial dimensions:

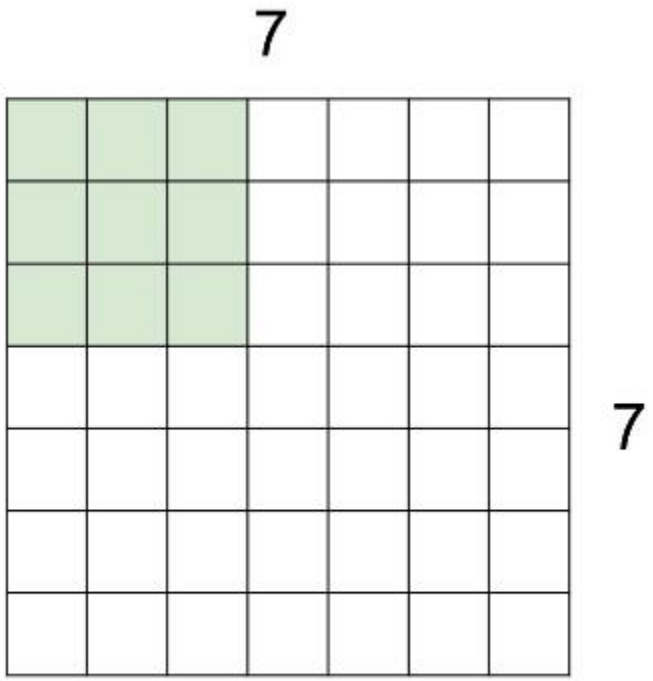


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

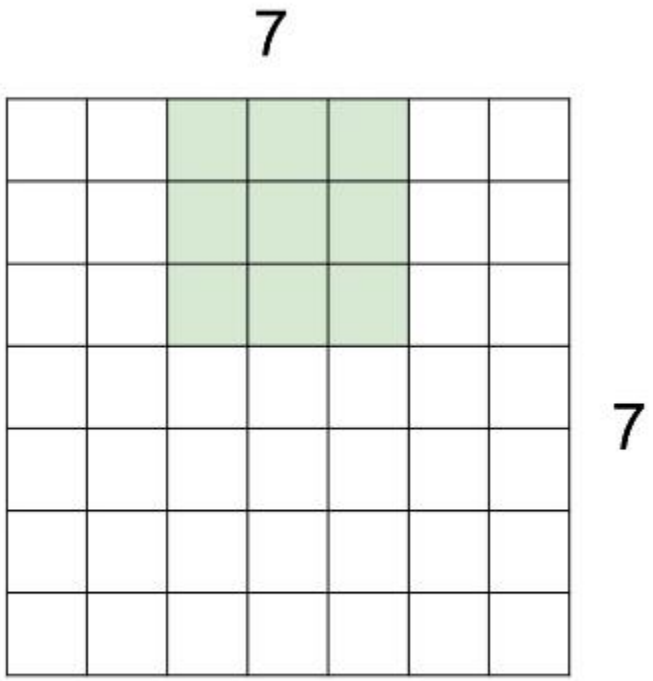
A closer look at spatial dimensions: Stride=2



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

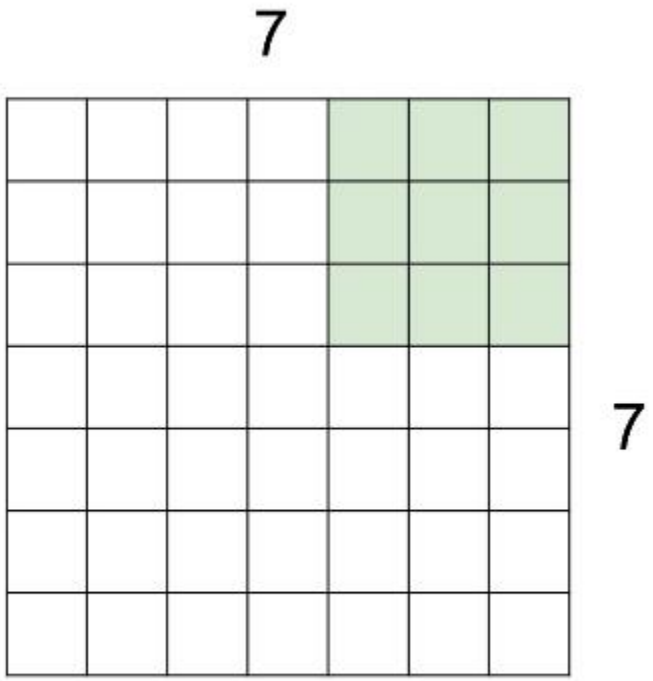
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

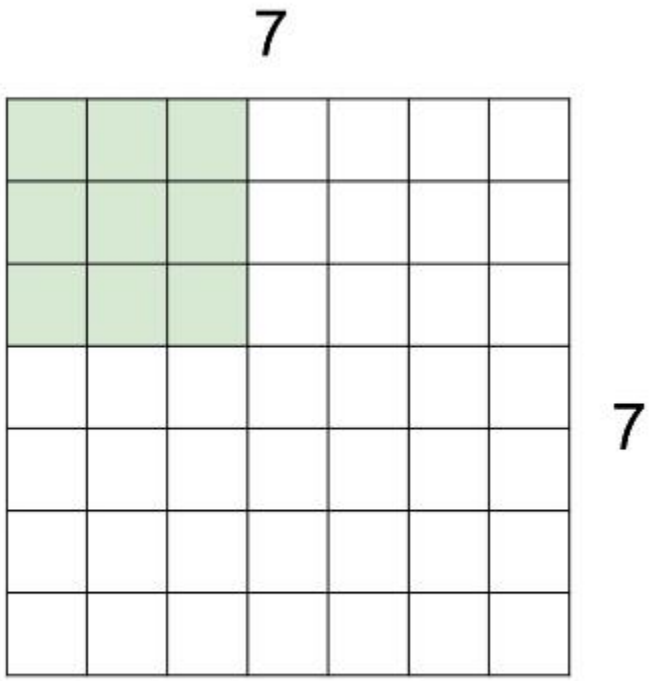
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

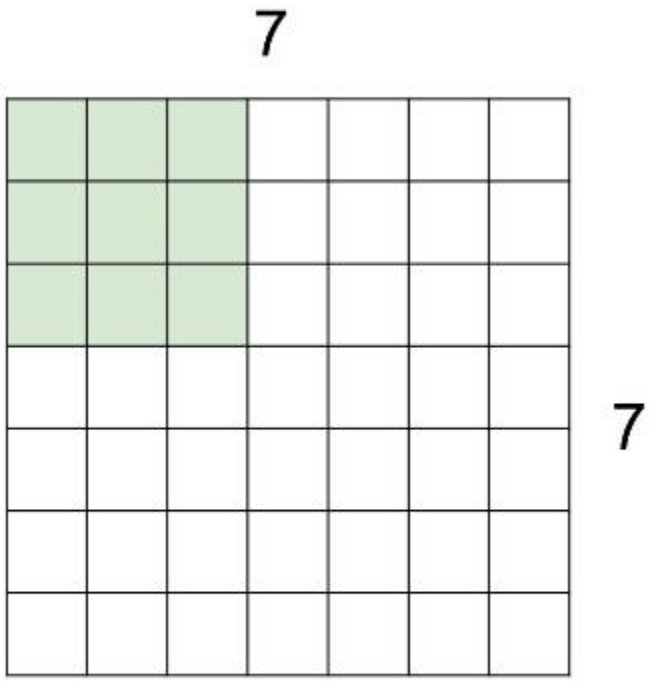
A closer look at spatial dimensions: Stride=3



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

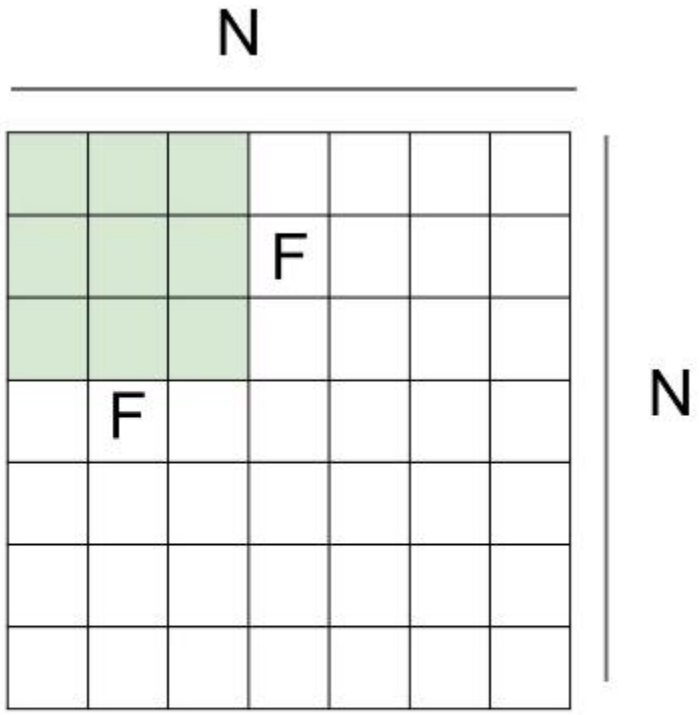
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung



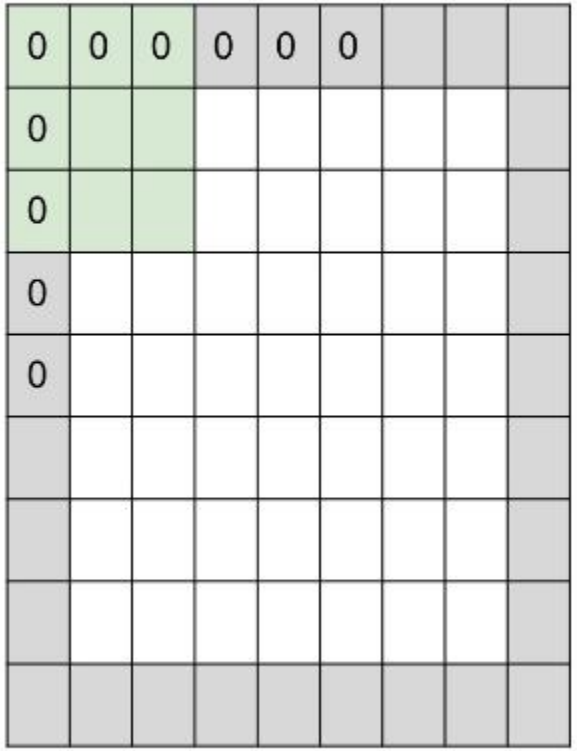
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Zero padding

In practice: Common to zero pad the border

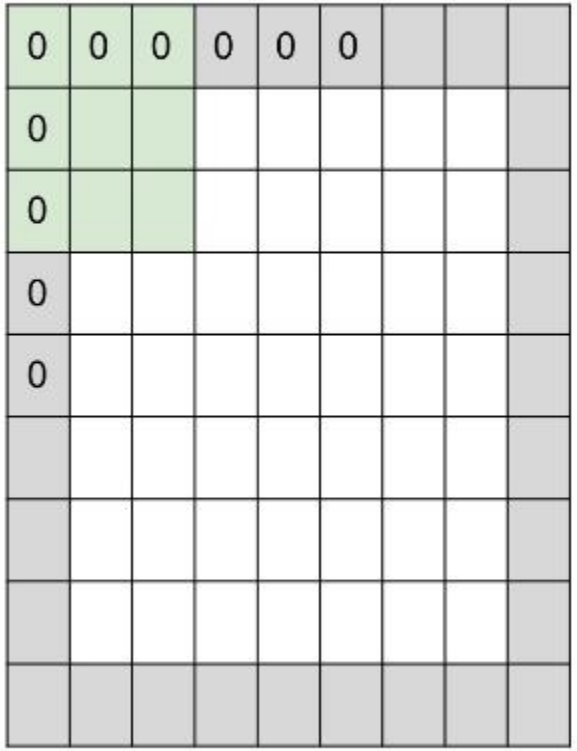


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

In practice: Common to zero pad the border

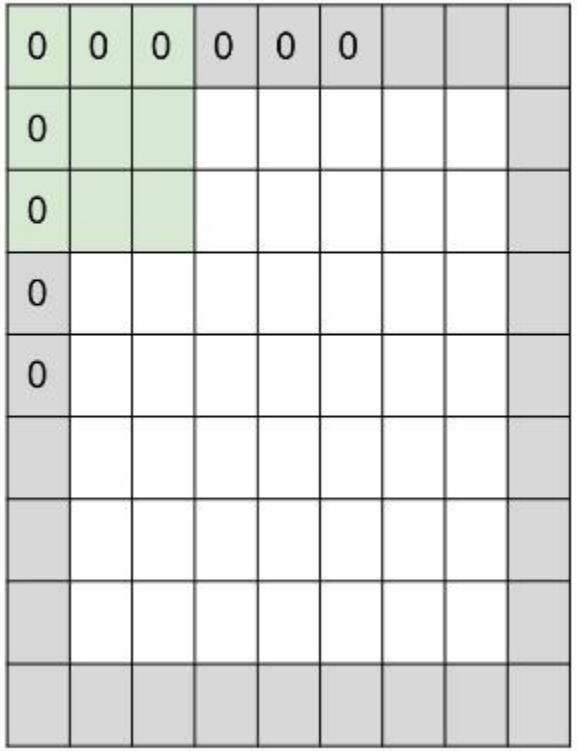


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

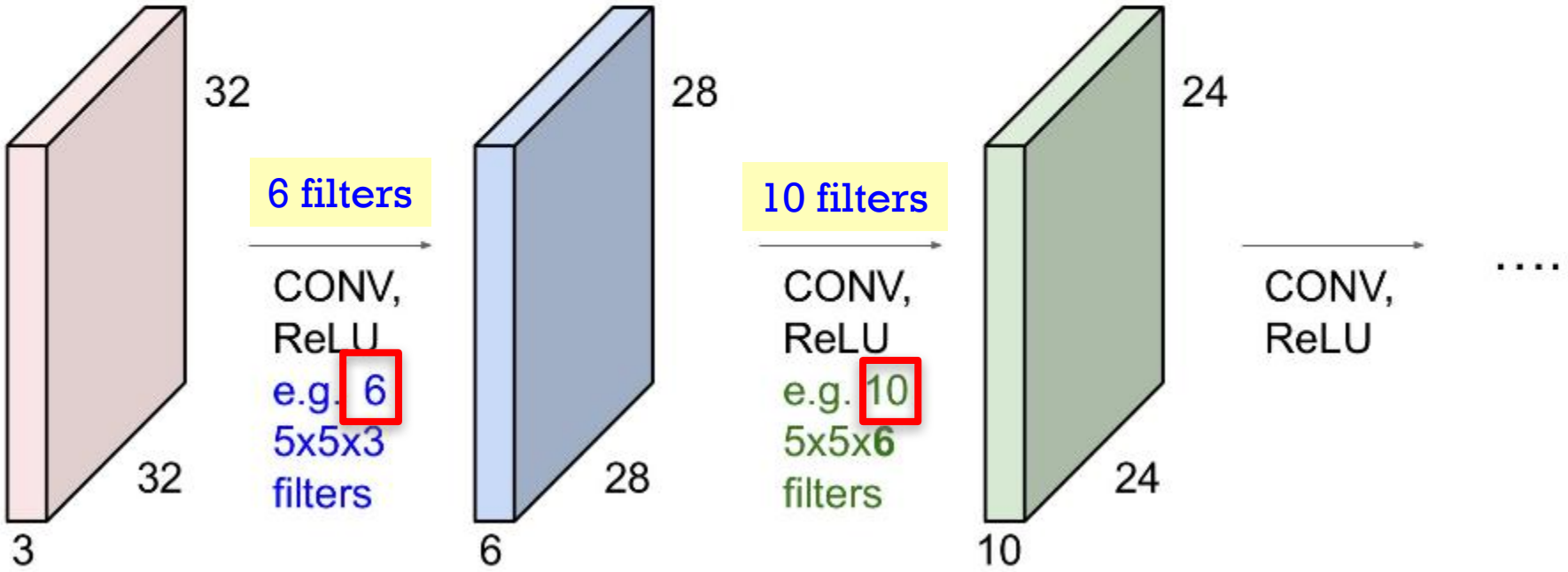
Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Each feature map must transformed using "activation function" (ReLU).

Remember back to…

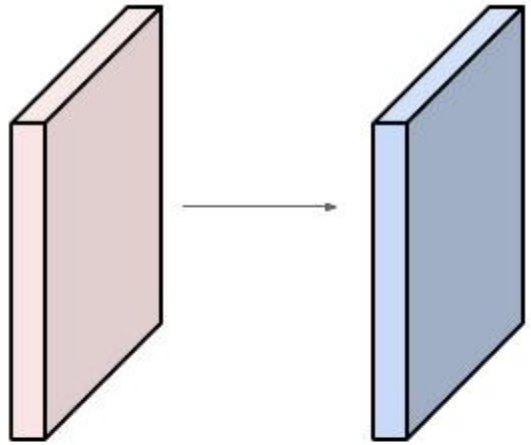
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

32 → 32 → 32 …



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Examples time:

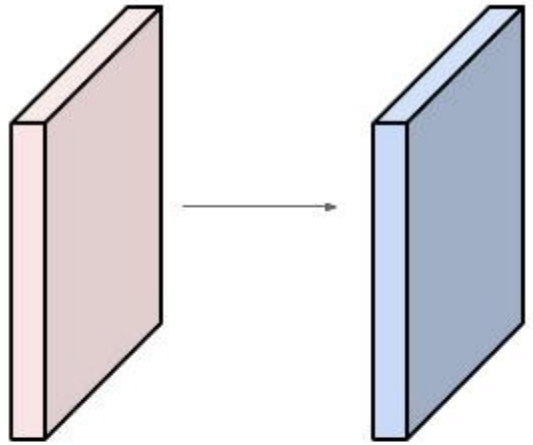


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Examples time:



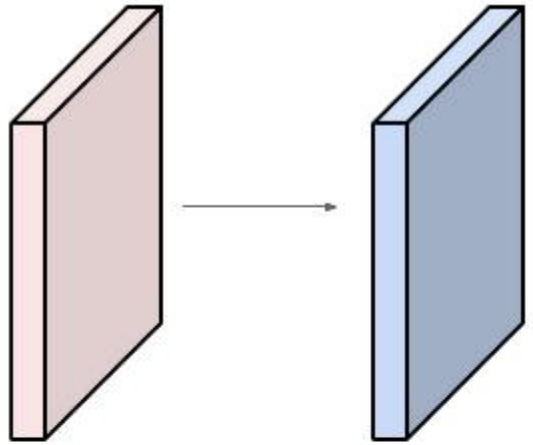
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

32 → 32 → 32 …

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Examples time:

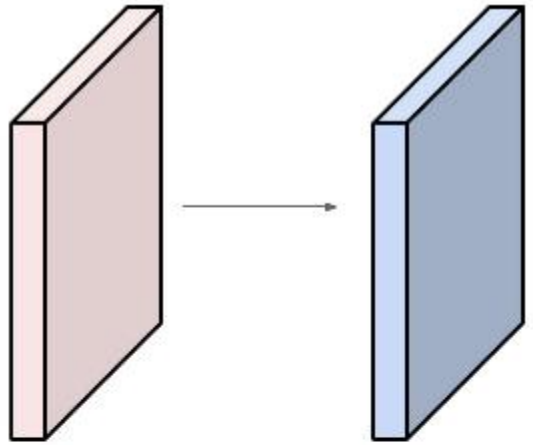


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Summary: **4 parameters** for convolutional layer

- Filter (size, n), Stride (step), Padding (n)

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

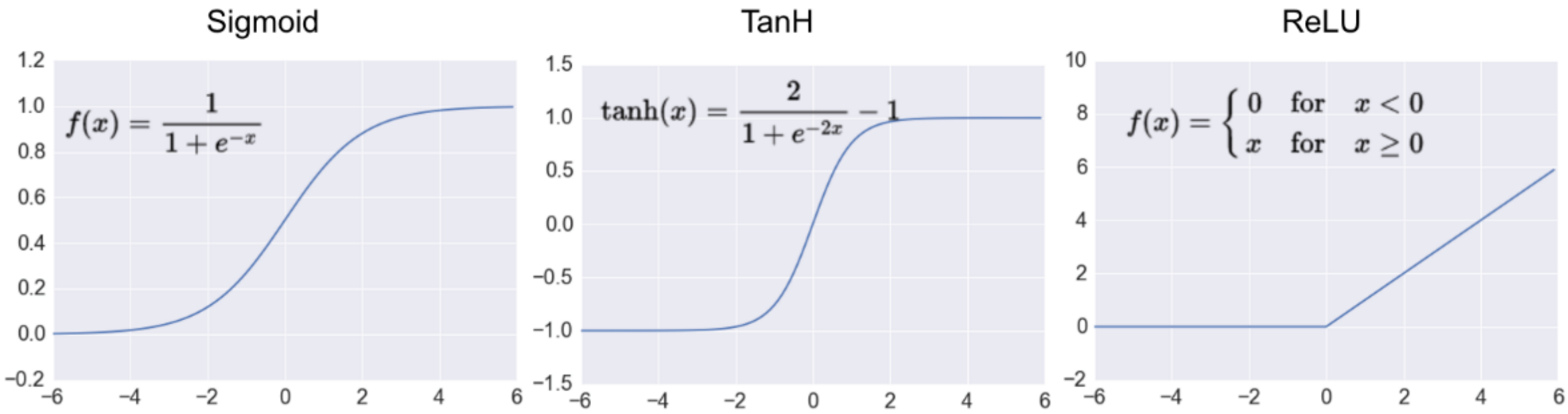
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

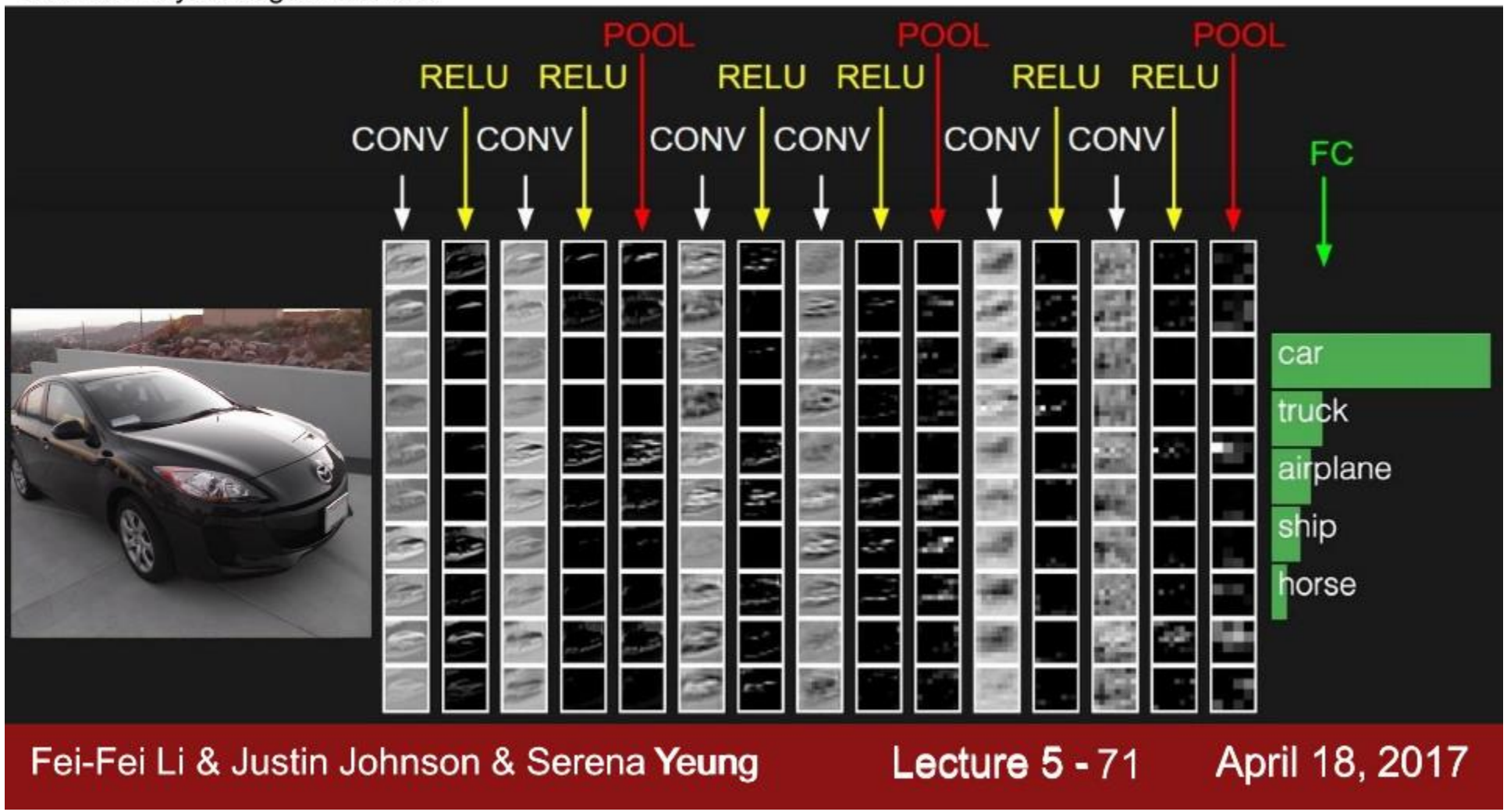
Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Deep Learning: Non-Linear Activation Function

Convolutional Neural Network

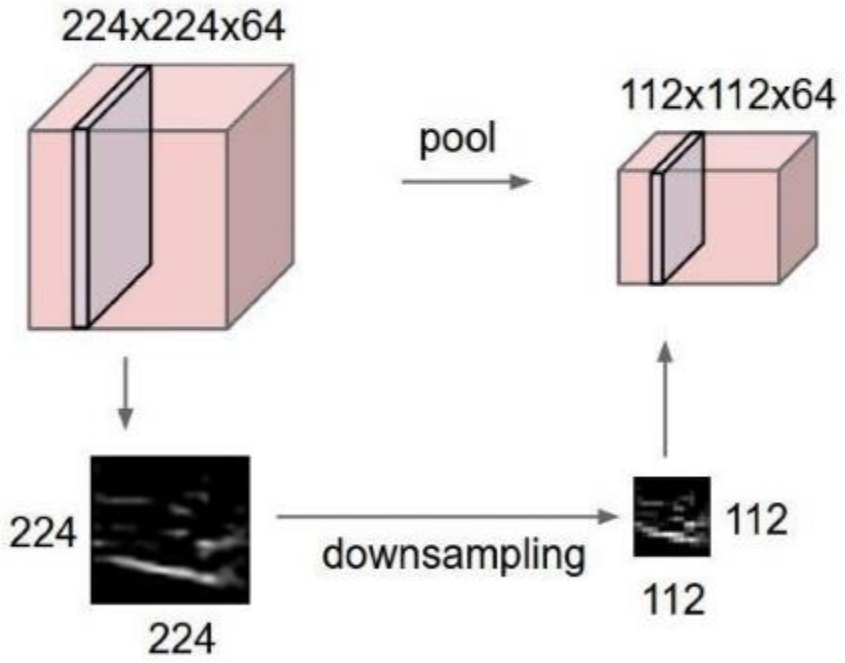


two more layers to go: POOL/FC



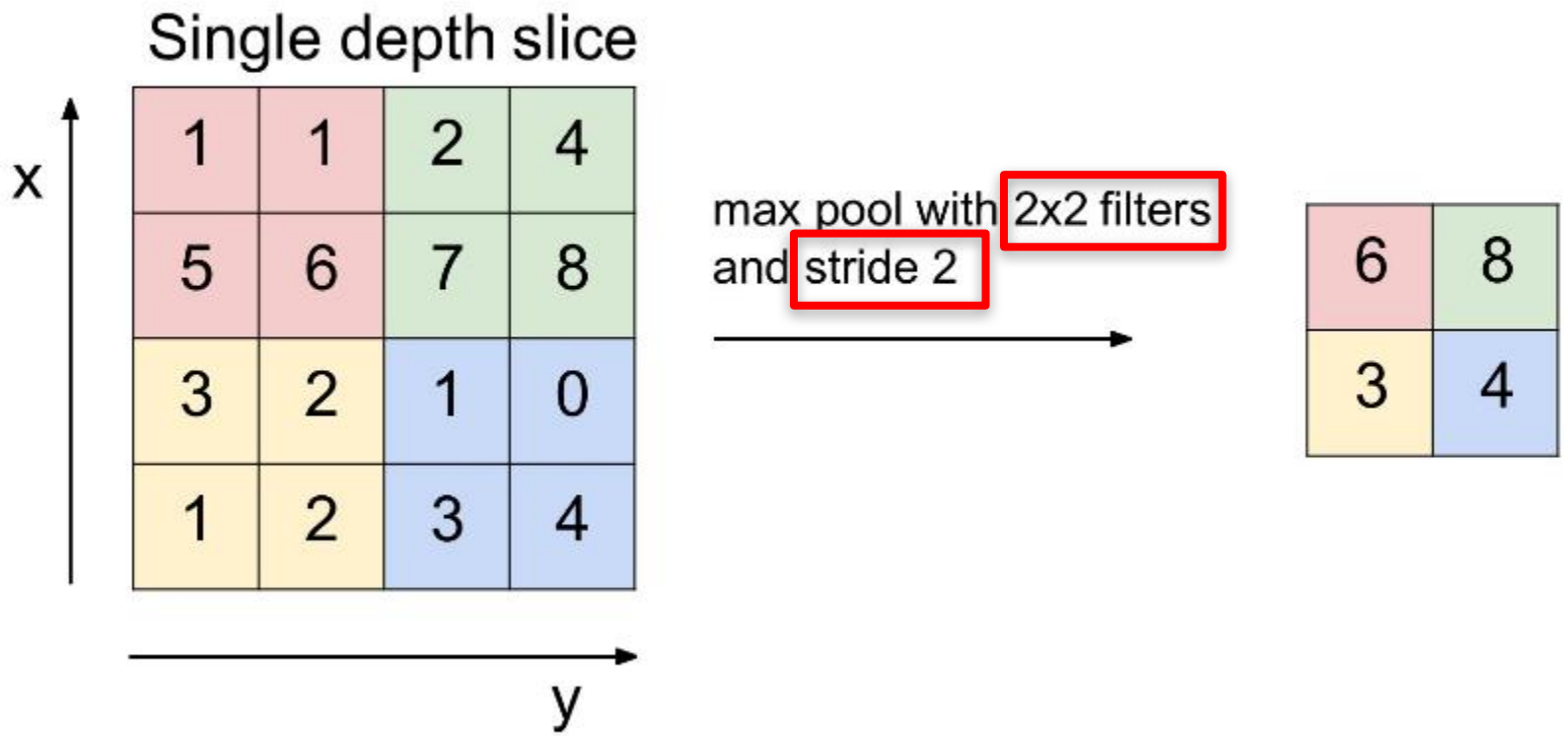
Layer2: Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

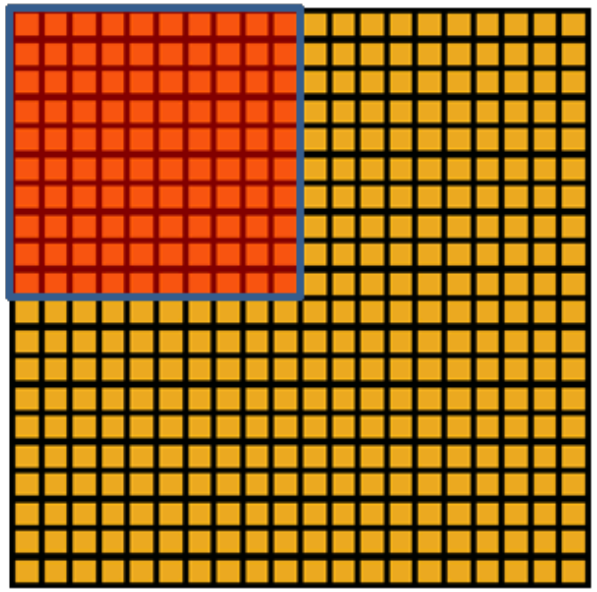
MAX POOLING

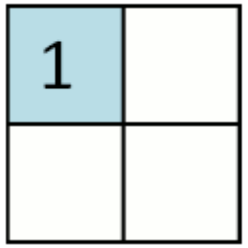


Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Deep Learning – CNN (Pooling Layer)

Convolutional Neural Network





Summing or Averaging

Convolved feature

Pooled feature

Summary: **2 parameters** for pooling layer

- Filter (size), Stride (step)

Common settings:

F = 2, S = 2
F = 3, S = 2

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

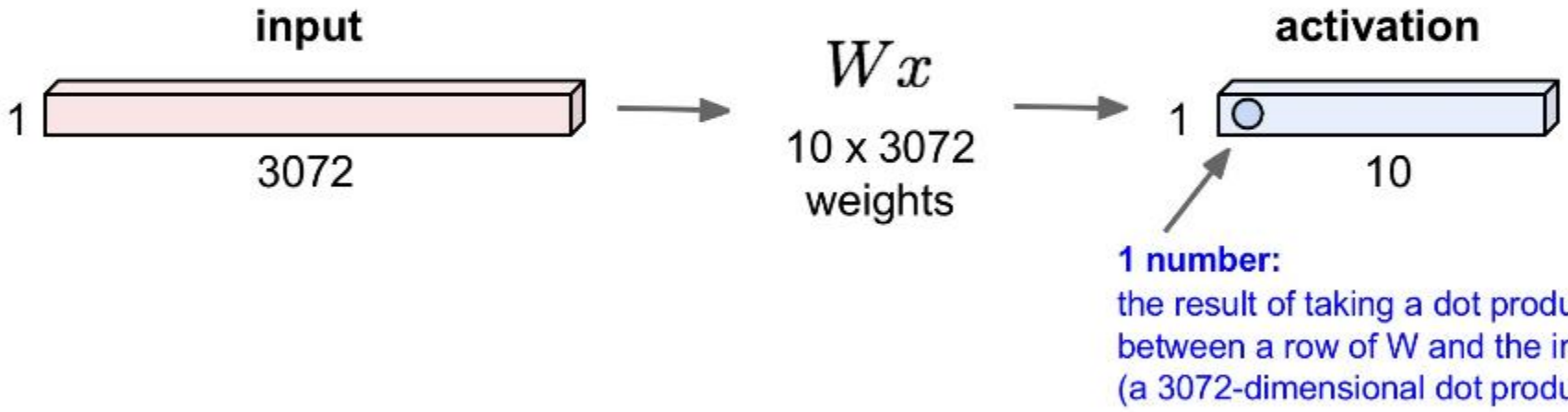
Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Layer3: Fully-Connected Layer (Classification Layer)

1. Convolutional Layer
2. Pooling Layer
3. Fully-Connected Layer

Each neuron looks at the full input volume

32x32x3 image -> stretch to 3072 x 1



Assume there are 10 output classes,
the final result is transformed using "softmax"

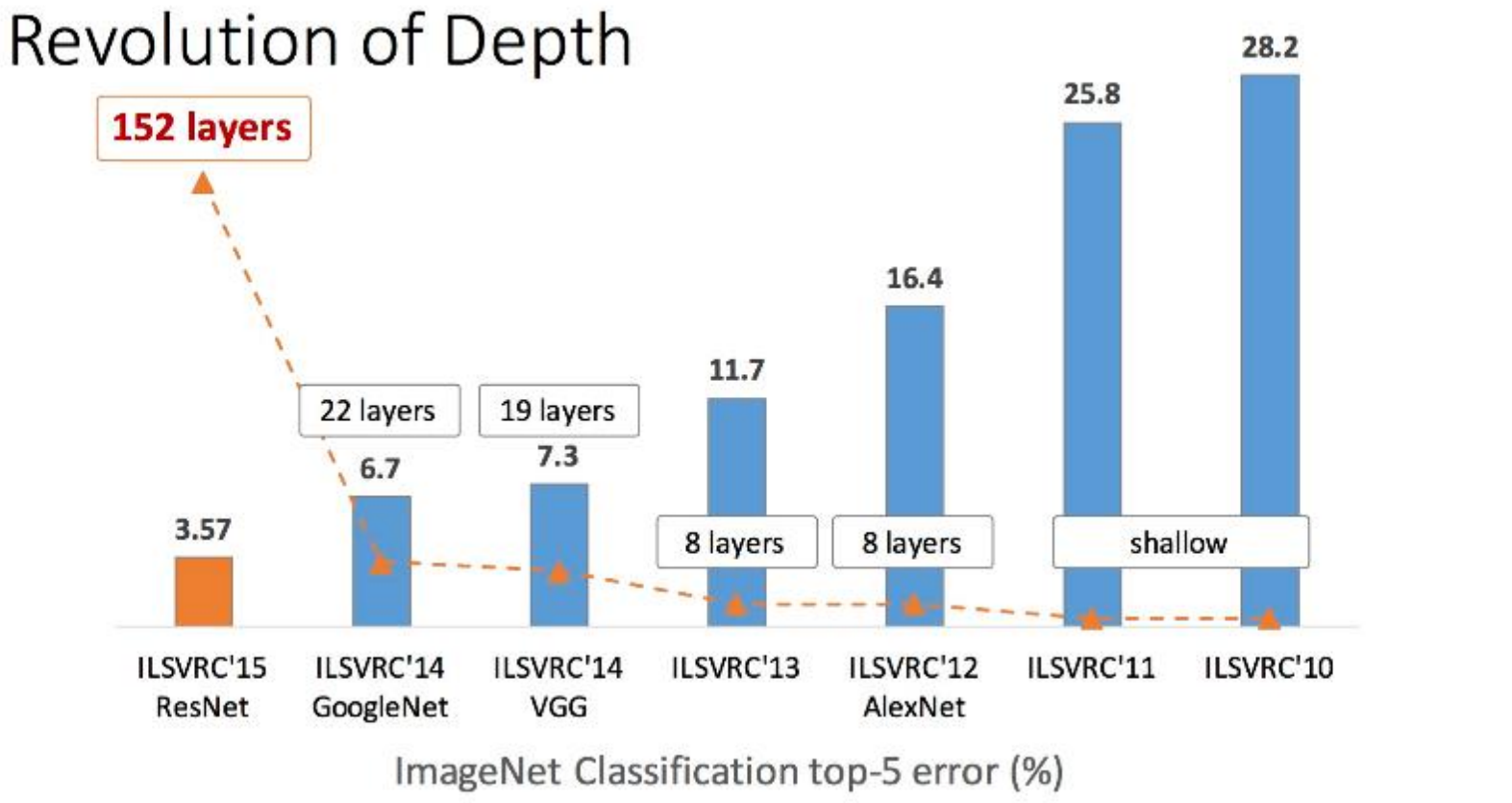
Reference: CNN Lecture 2017, Fei-Fei Li & Justin Johnson & Serena Yeung

Summary: **2 parameters** for FC layer (NN)

- #input units, #output units

Deep Learning – CNN (Increasing Depth)

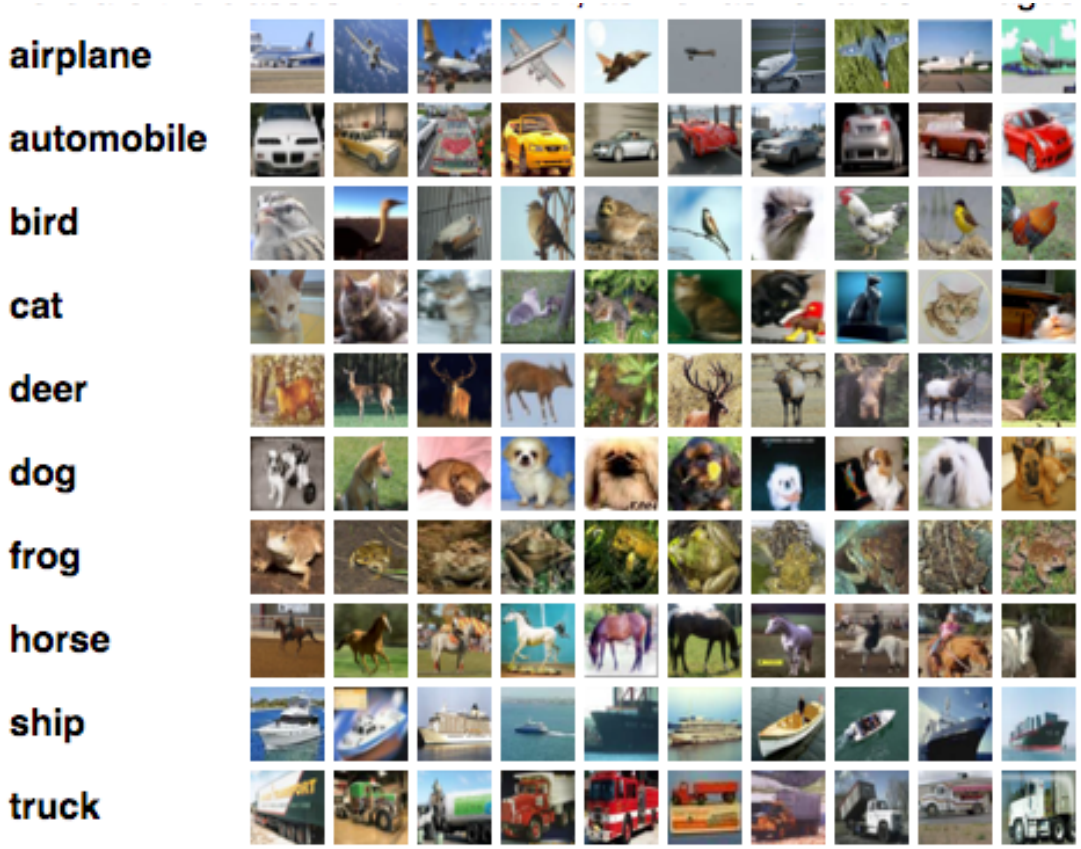
Convolutional Neural Network



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR

Deep Learning – CNN (Lab - Practice Problem)

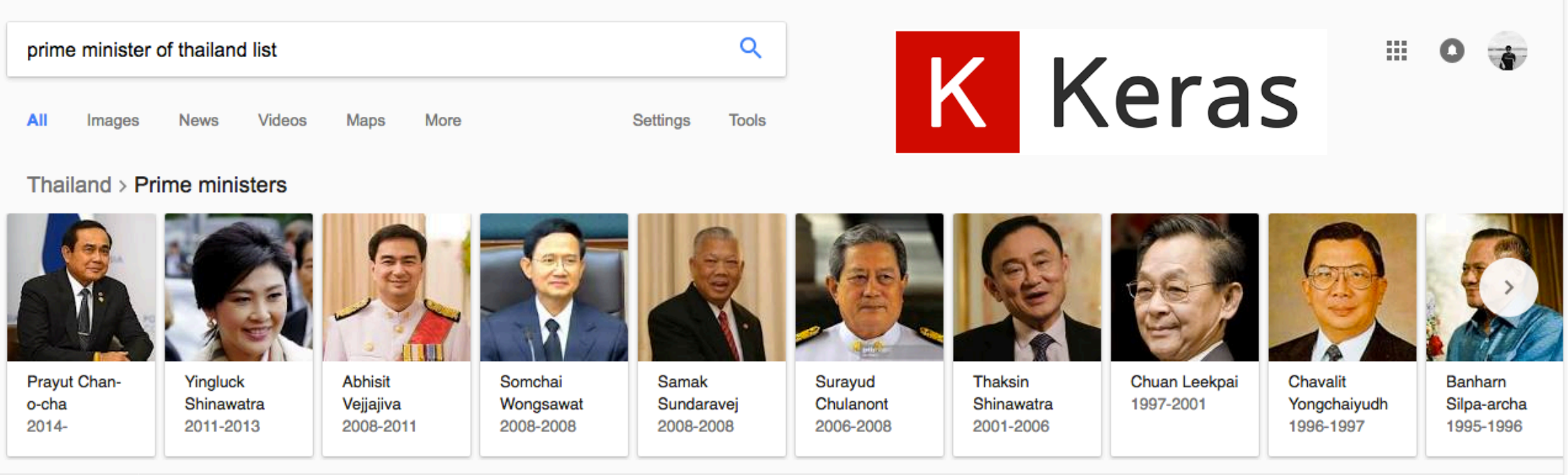
The CIFAR-10 dataset



Keras

- The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class.
- There are 50000 training images and 10000 test images.

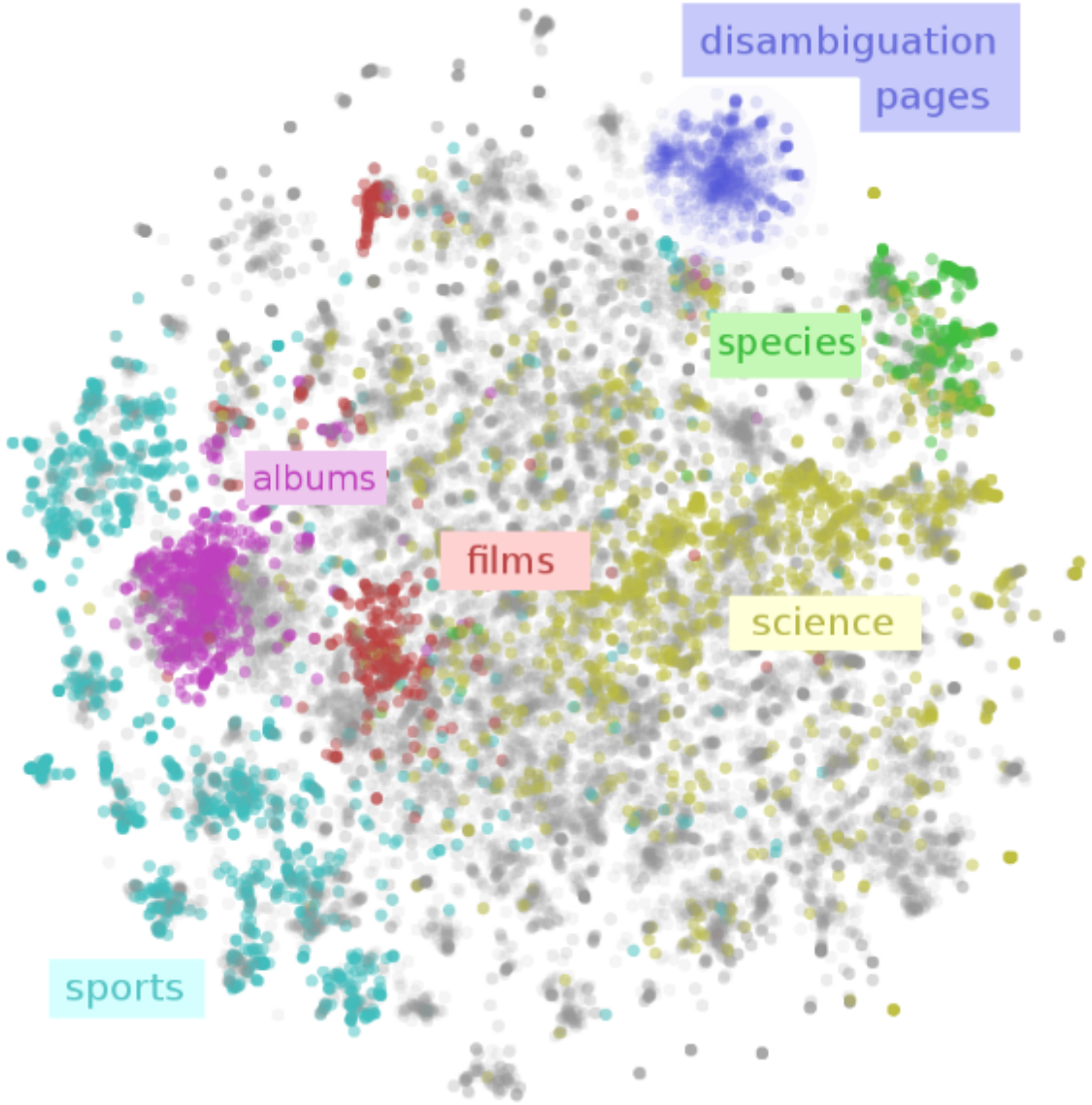
Deep Learning – CNN (LeNet5 + DropOut) (Lab - Practice Problem) - DEMO



Keras

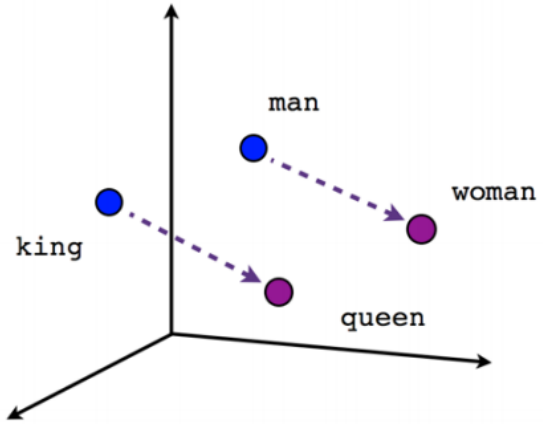
- Training set of 3,500 examples
- Testing set of 1,500 examples
- Image size (75x90) and target have 10 classes

Word Embedding

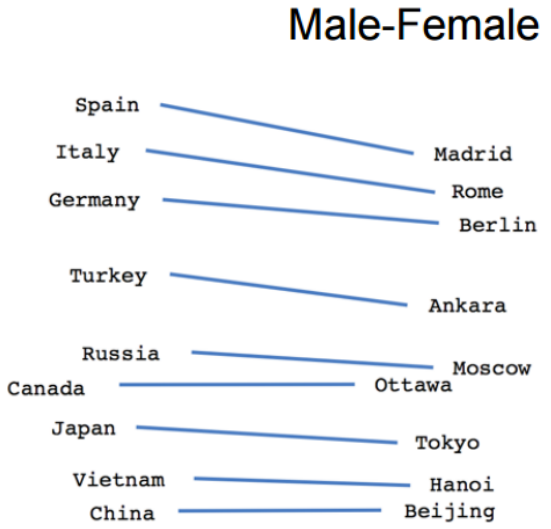


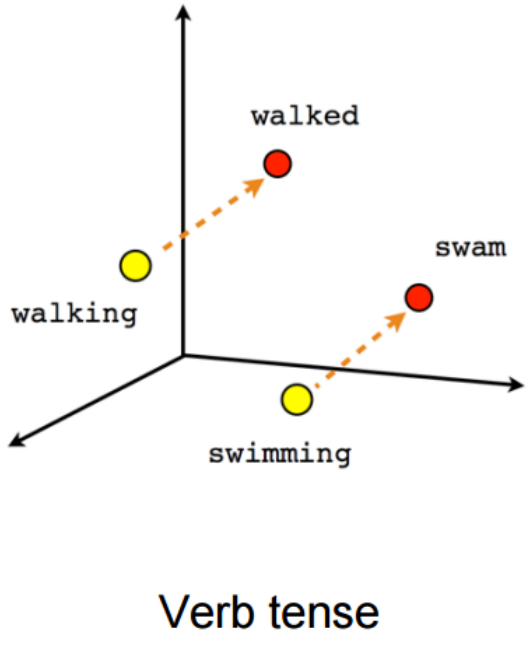
Credit : Christopher Olah, http://colah.github.io/

What's Word Embedding?



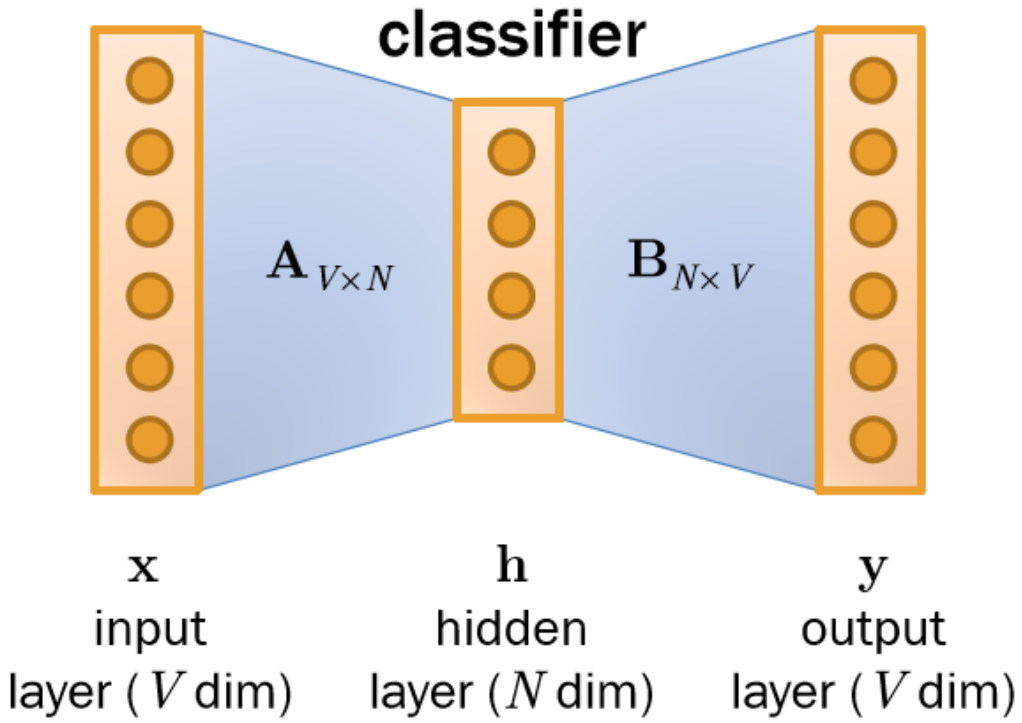
- Vector representation of words w.r.t. their contexts (tasks) (i.e. word = point) (Rumelhart+, 1986)
  - Old fashion are based on "a bag of words" or "a sequence of one-hot vectors"
- The dot product of two word vectors yields their spatial distance (i.e. word similarity)
- Word Embedding enables deep learning for NLP because it reduces the dimension of the input via vectorization, while preserving word meaning





Credit :Piotr Migda, http://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html, 2017

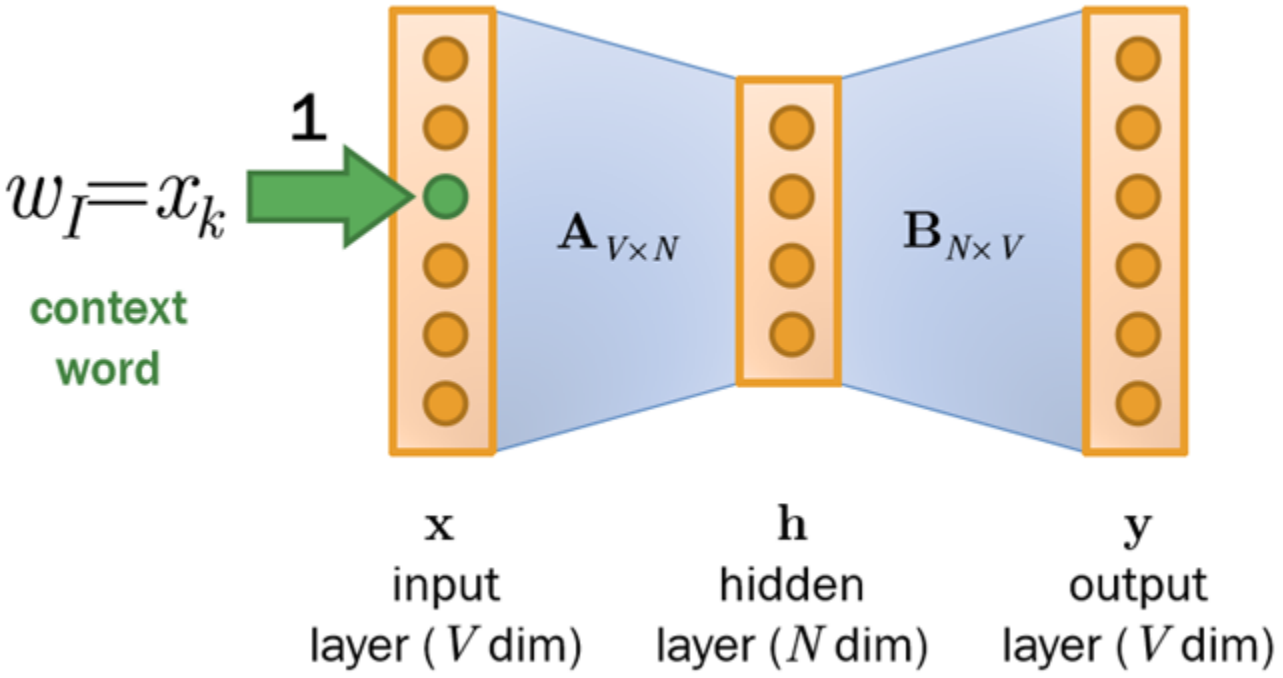
Word Embedding Model:
w.r.t. predicting next word (context; tasks)



- Word Embedding is the first hidden layer that act likes a mapping between a word and its vector form
- The hidden layer acts as a classifier, because it reduces the input layer's dimensionality to N

Credit : Prachya Boonkwan, Word2Vec - When language meets number crunching, 2016

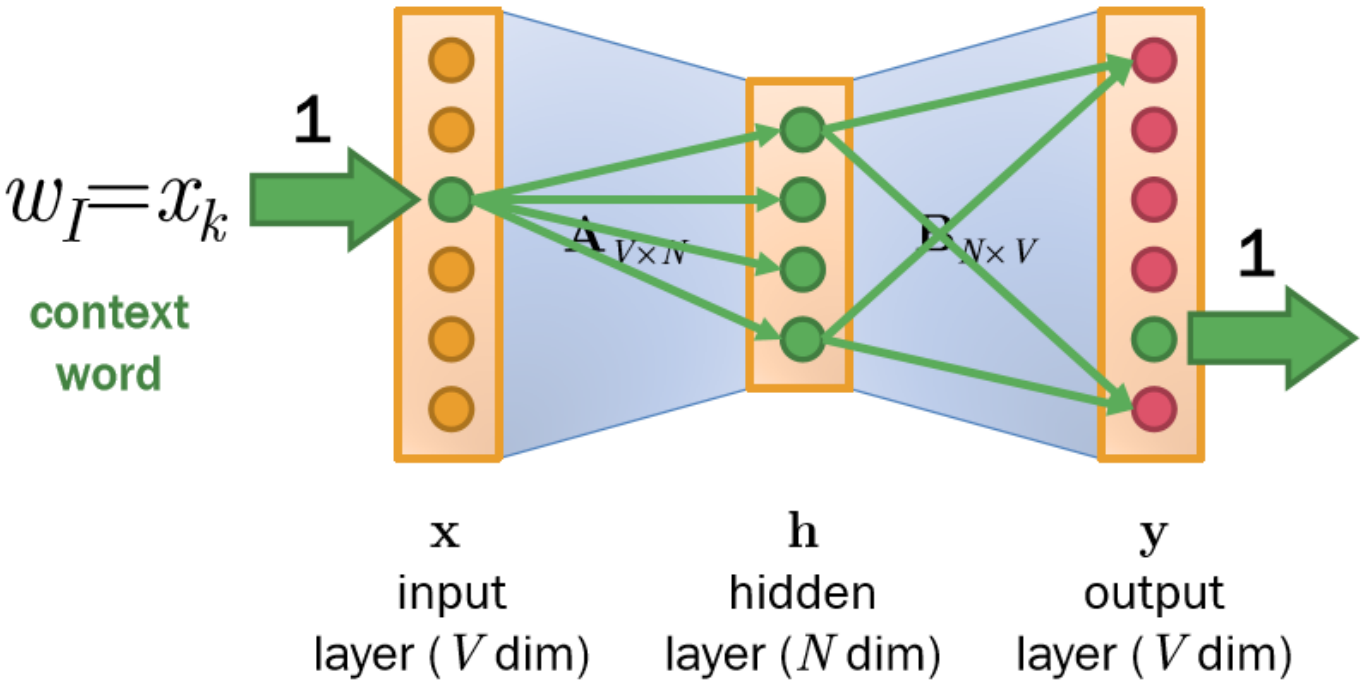
Word Embedding Model (cont.)



- Word Embedding is the first hidden layer that act likes a mapping between a word and its vector form
- The hidden layer acts as a classifier, because it reduces the input layer's dimensionality to N
- The input is usually a one-hot vector

Credit : Prachya Boonkwan, Word2Vec - When language meets number crunching, 2016

Word Embedding Model (cont.)

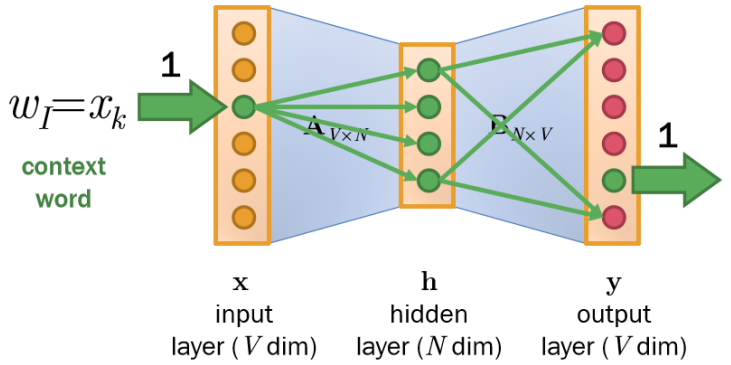


- Word Embedding is the first hidden layer that act likes a mapping between a word and its vector form
- The hidden layer acts as a classifier, because it reduces the input layer's dimensionality to N
- The input is usually a one-hot vector
- The next word is predicted by the softmax distribution over the output layer

Credit : Prachya Boonkwan, Word2Vec - When language meets number crunching, 2016

Word2Vec

- Word2Vec is a pre-trained word embedding for any domains
  - The Word2Vec vector will be an initial weight for word embedding.
- Word2Vec's goal is to produce word embeddings that encode general semantic relationships
- Notably Word2Vec model
  - Google : Word2Vec (Mikolov+, 2013)
  - **Google : Glove (Pennington+, 2014) [more training data]**
  - Facebook : FastText (Bojanowski+, 2016)

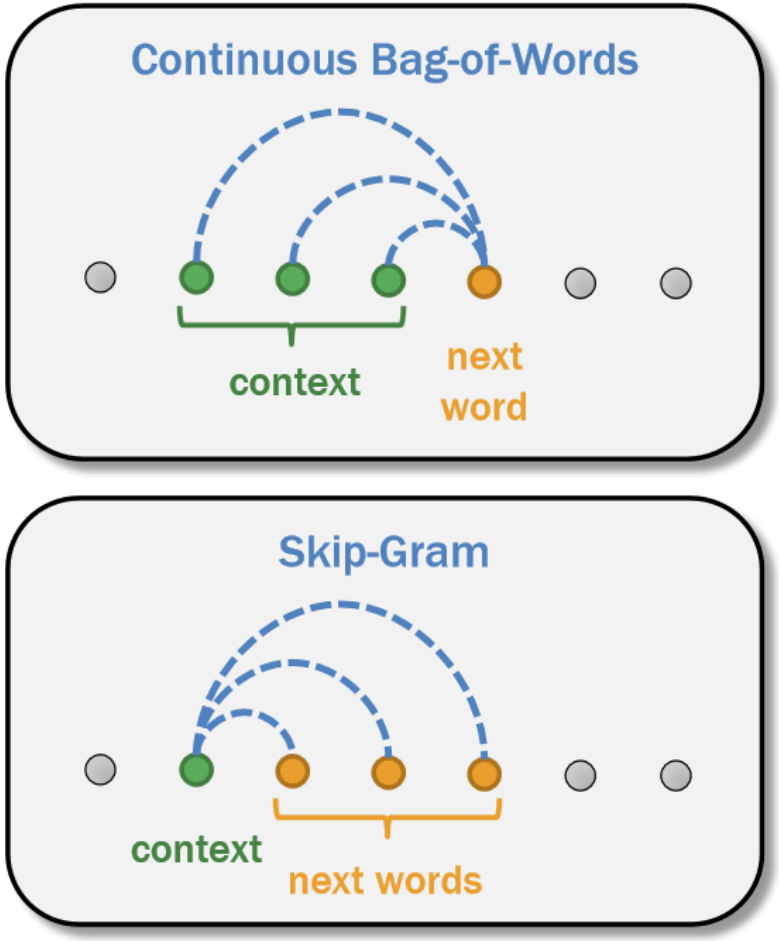


Google

facebook

Word2Vec (cont.)

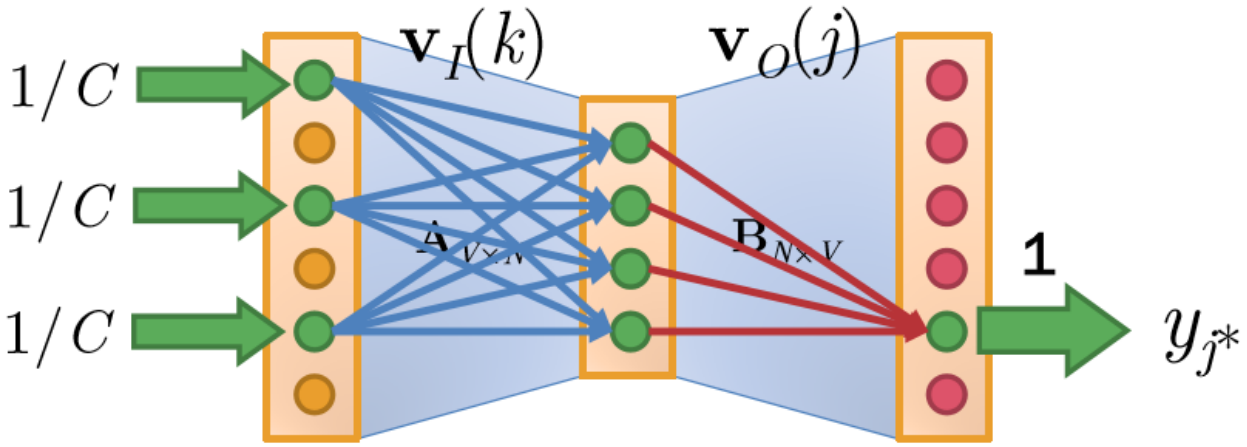
- Word2Vec is built using the concept of word co-occurrence
- The popular model use either:
  - Continuous Bag-of-Words (Harris, 1954): Many context words generate the next word
  - Skip-Gram (Mikolov+, 2013): One context word generates many next words

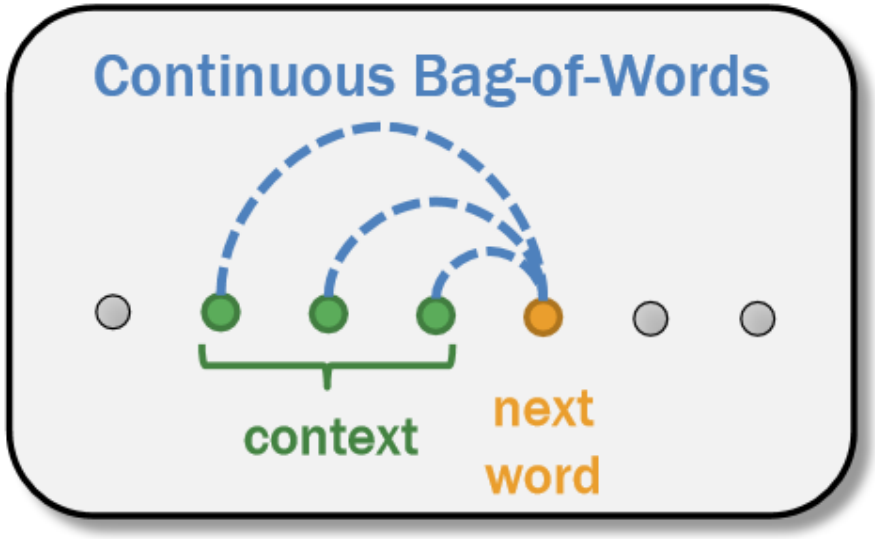


Credit : Prachya Boonkwan, Word2Vec - When language meets number crunching, 2016

Word2Vec (cont.)

- Continuous Bag-of-Words
  - We simultaneously input several context words (C) and predict one output word

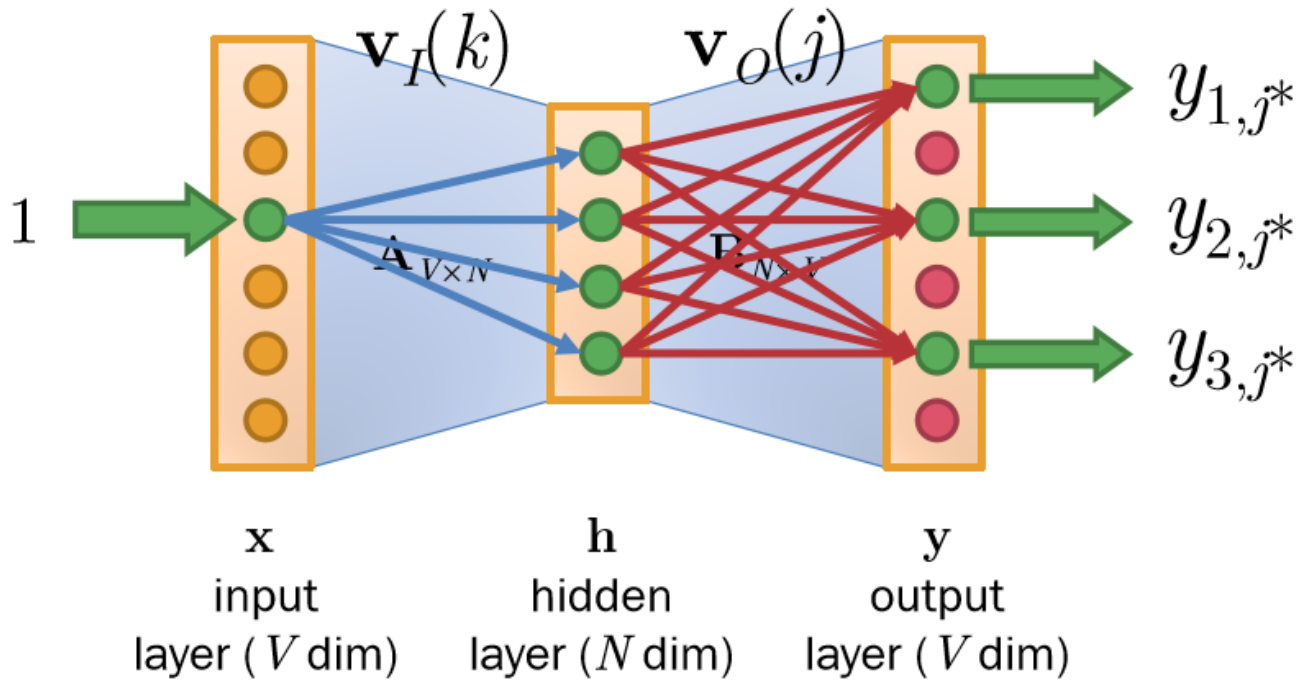




Credit : Prachya Boonkwan, Word2Vec - When language meets number crunching, 2016

Word2Vec (cont.)

- Skip-Gram
  - Several output words (C) are predicted independently from one context word



Credit : Prachya Boonkwan, Word2Vec - When language meets number crunching, 2016

Word2Vec (cont.)

- Benefit of pre-trained word embedding
  - Faster training time
  - Improve performance on rare words
- Applications
  - Text Classification, Sentiment Classification, Machine Translation, etc.
  - Or any neural network that has word as an input, and want general semantic relationships

Long-Short Term Memory (LSTM)

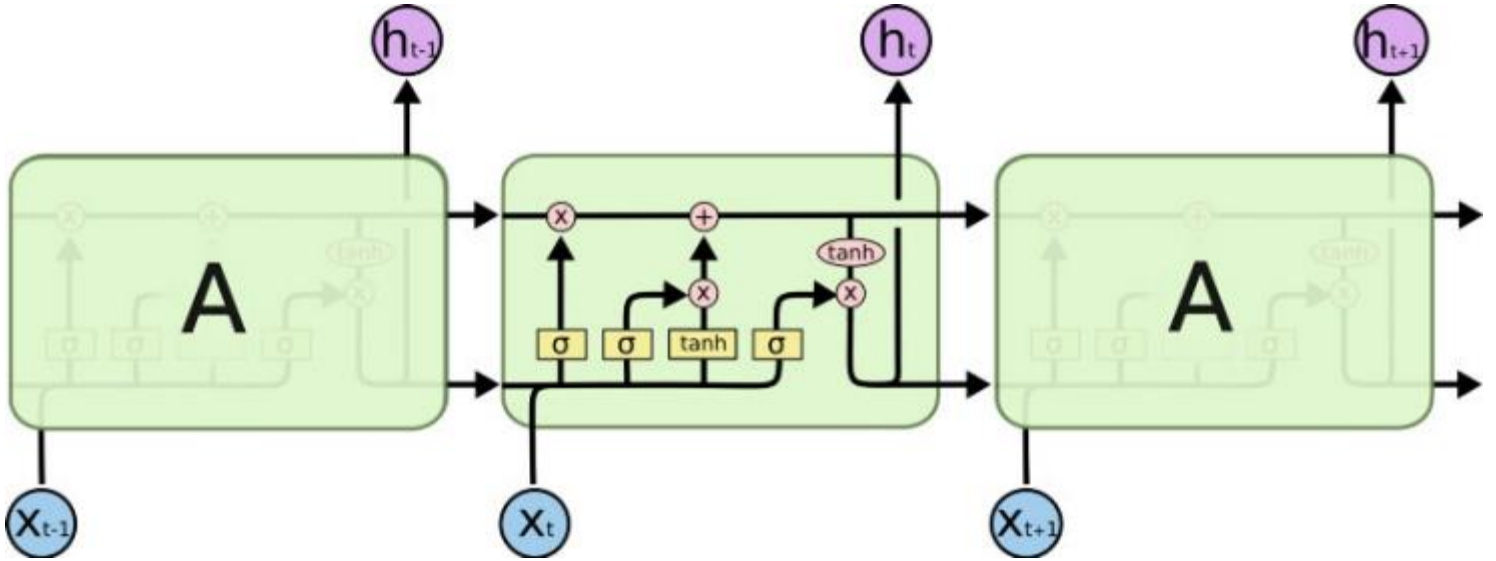
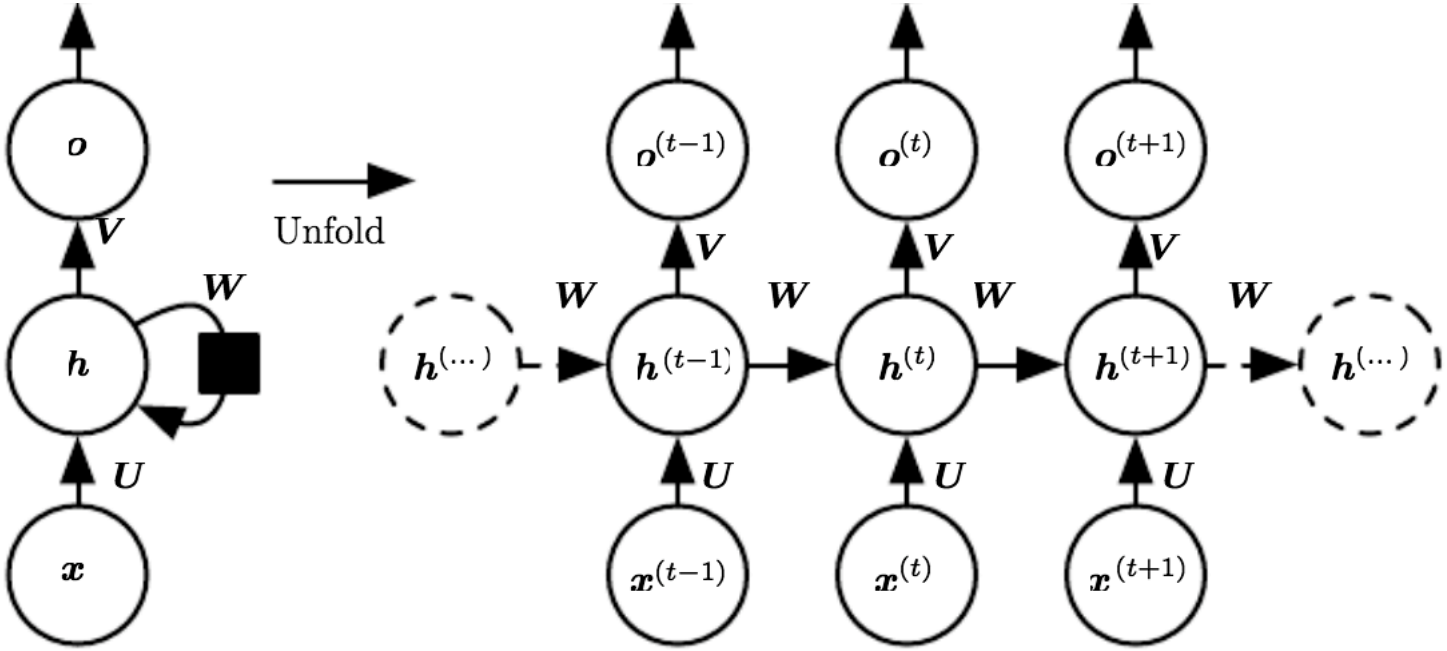


Image reference:http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network (RNN)

- RNN uses the same set of U, V, and W weights at every timestep/word
- The current output of the model is based on all previous timesteps and the current word

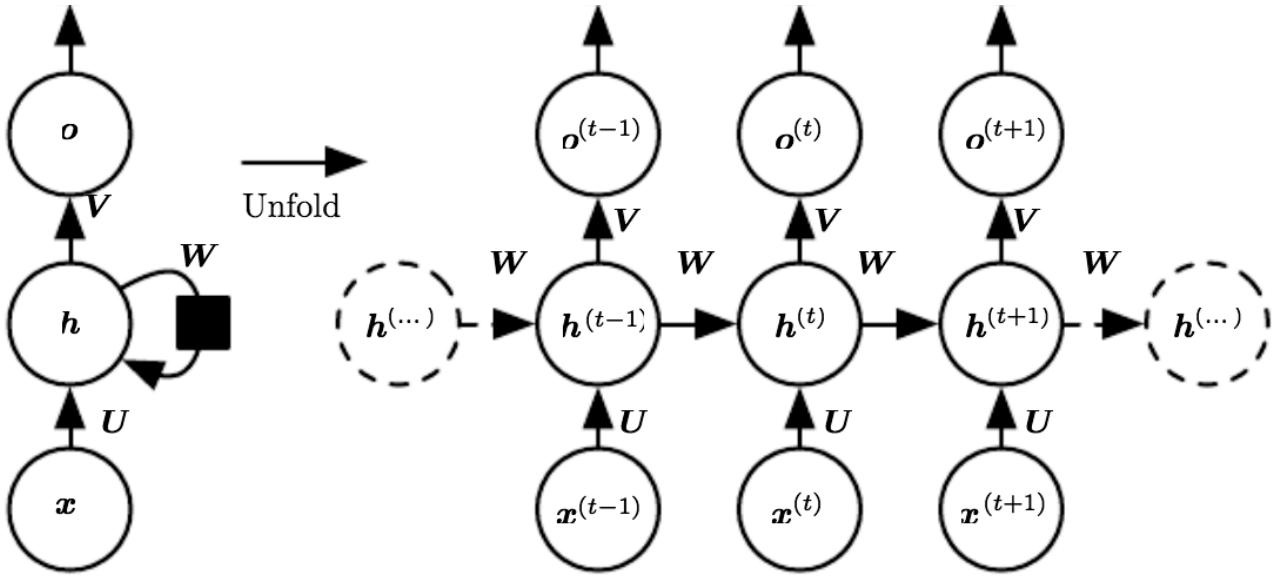


Reference: Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. MIT Press.
http://www.deeplearningbook.org

Recurrent Neural Network (RNN) (cont.)

current state | Activation function | previous state | current input

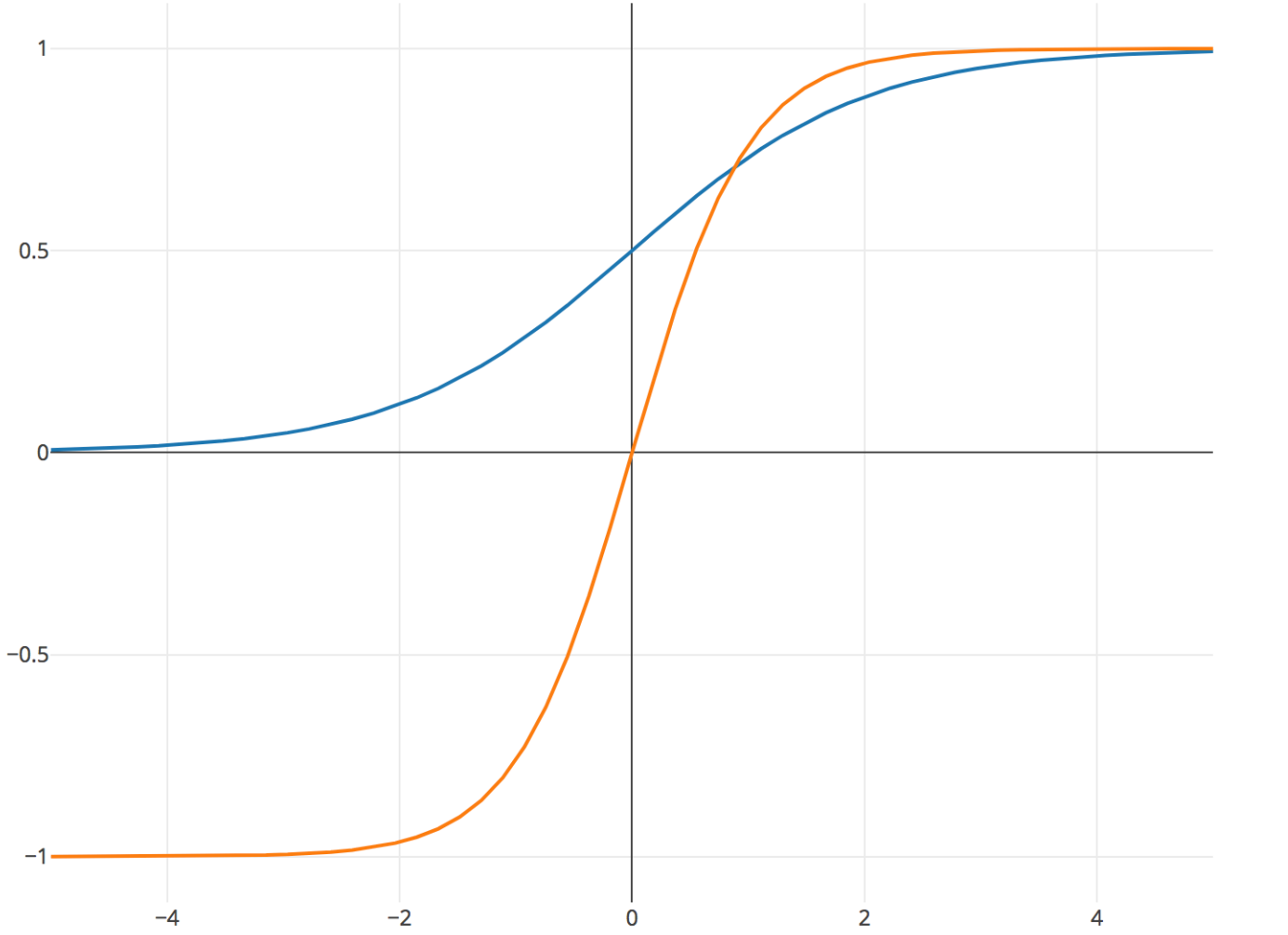
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)})$$



Recurrent Neural Network (RNN) (cont.)

Activation Function:

- Sigmoid
- Tanh



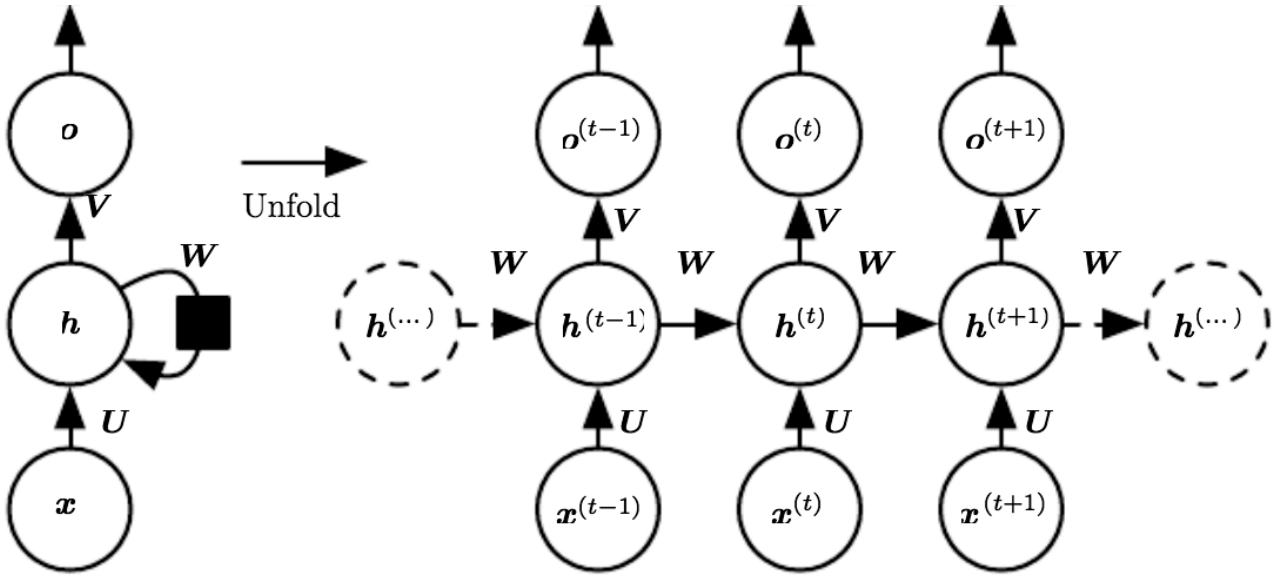
Recurrent Neural Network (RNN) (cont.)

current output

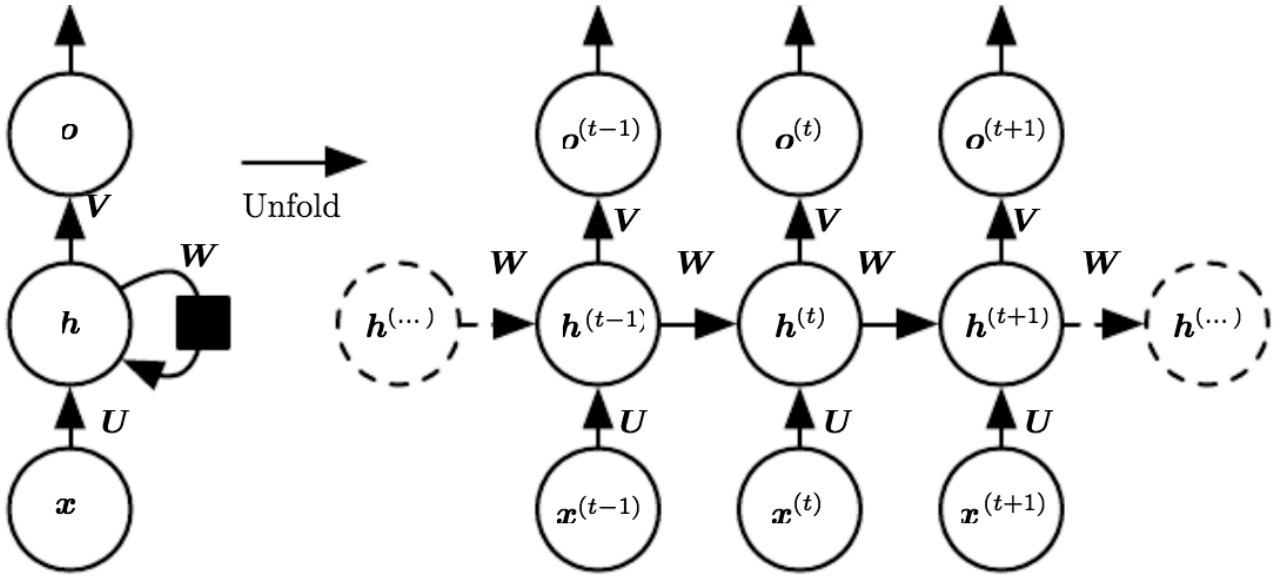
current state

bias

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$



RNN's Limitations



- One major problem with Recurrence Neural Network:
- It loses information from the timestep that is very far away
  - Long-term dependency
- Therefore we need a mechanism that allows information to flow through timesteps

Image reference:http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long-Short Term Memory (LSTM)

- Long Short-Term Memory (LSTM) has that mechanism (the cell state)

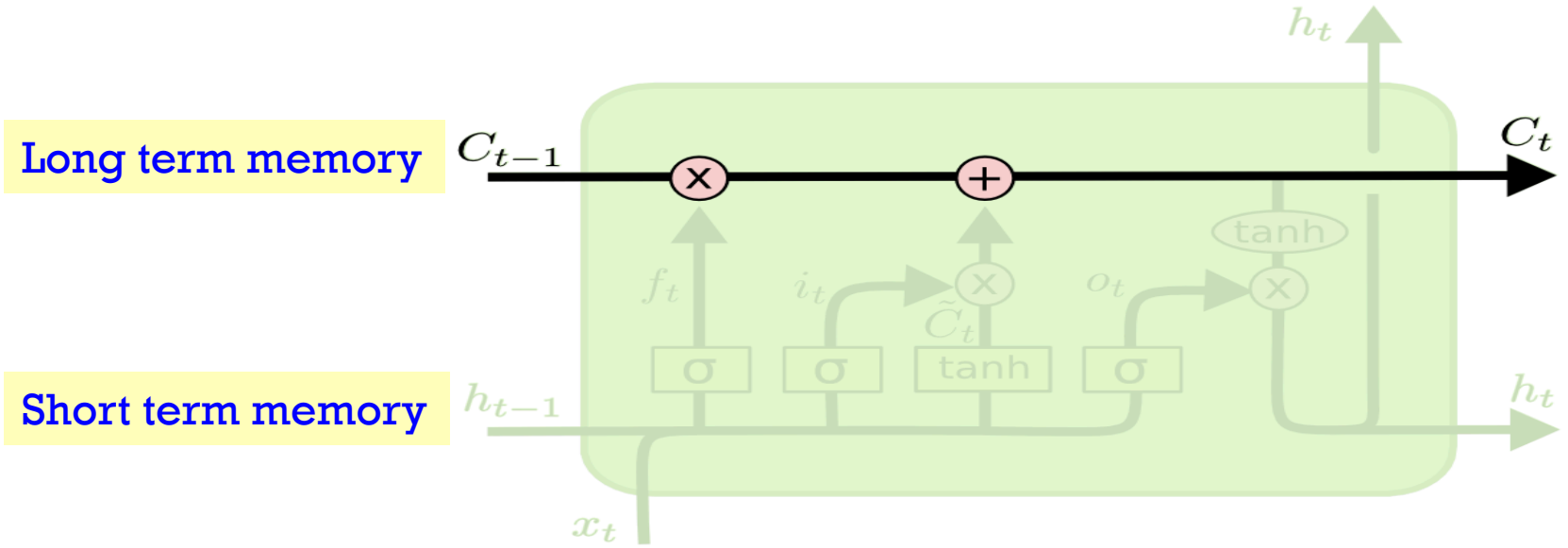
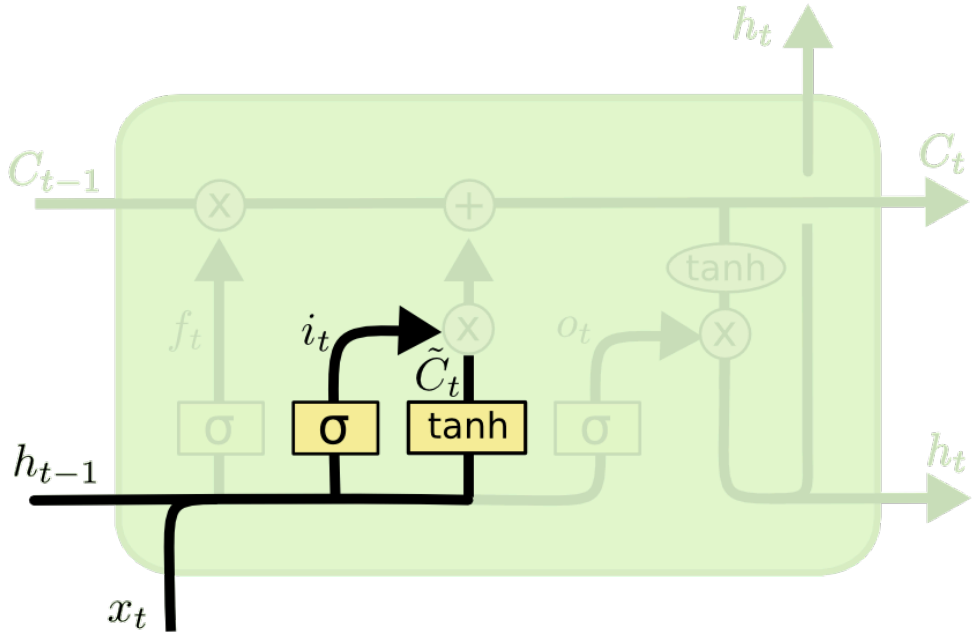


Image reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long-Short Term Memory (LSTM) (cont.)

- **Input gate**: determine how much the current input matters

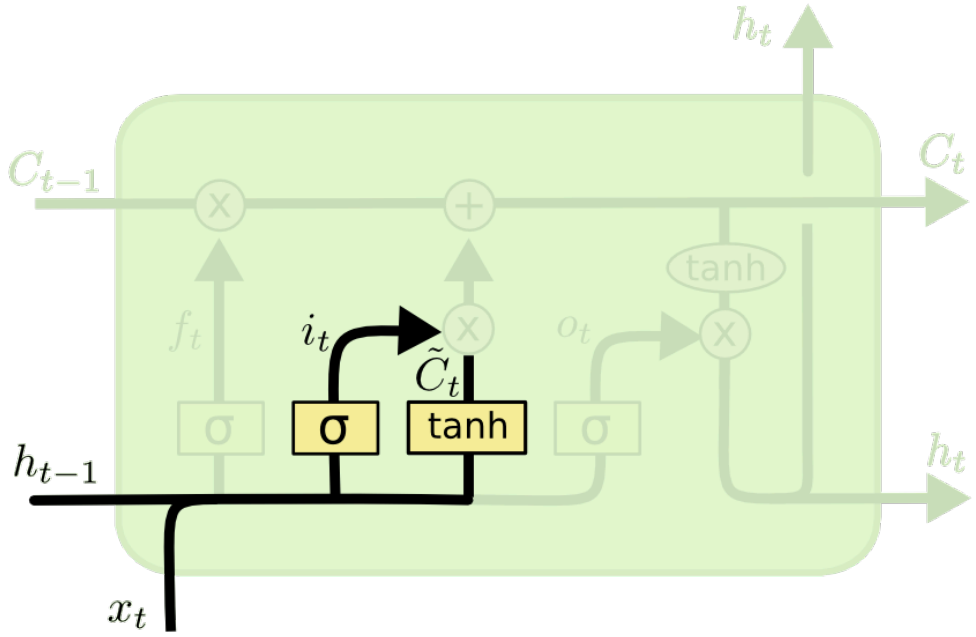


Input gate — current input — previous hidden state

$$i_t = \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi})$$

Long-Short Term Memory (LSTM) (cont.)

- **New memory cell**: new candidate values that "could be" added to the memory cell



New memory cell

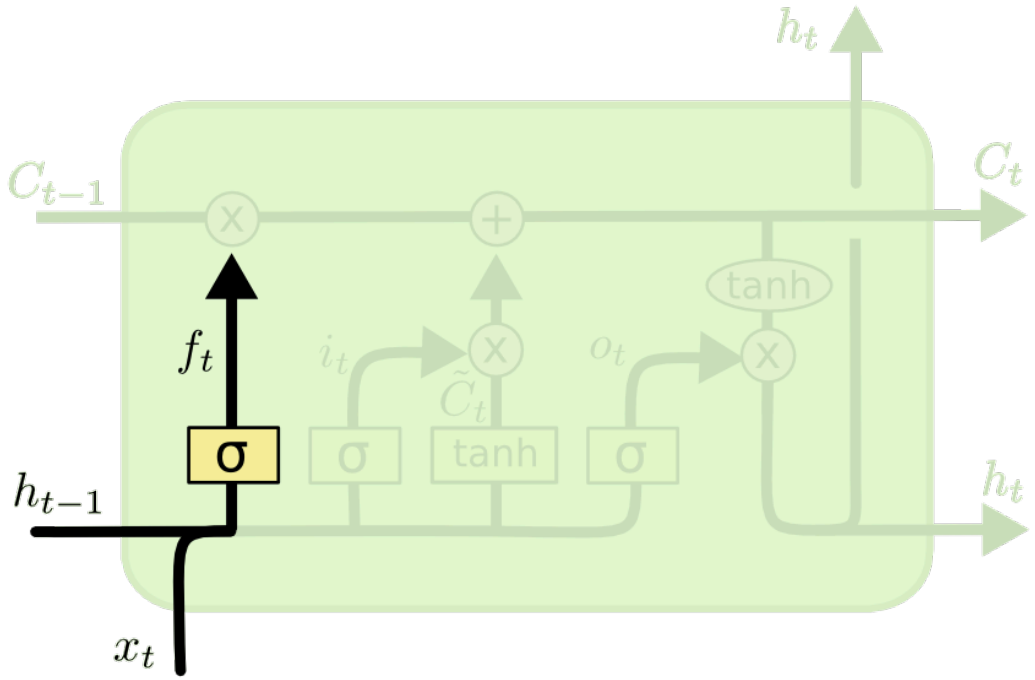
current input

previous hidden state

$$\widetilde{c}_t = \tanh(W_{ic}x_t + b_{ic} + W_{hc}h_{(t-1)} + b_{hc})$$

Long-Short Term Memory (LSTM) (cont.)

- **Forget Gate**: determine if the LSTM should forget the past memory cell



forget gate

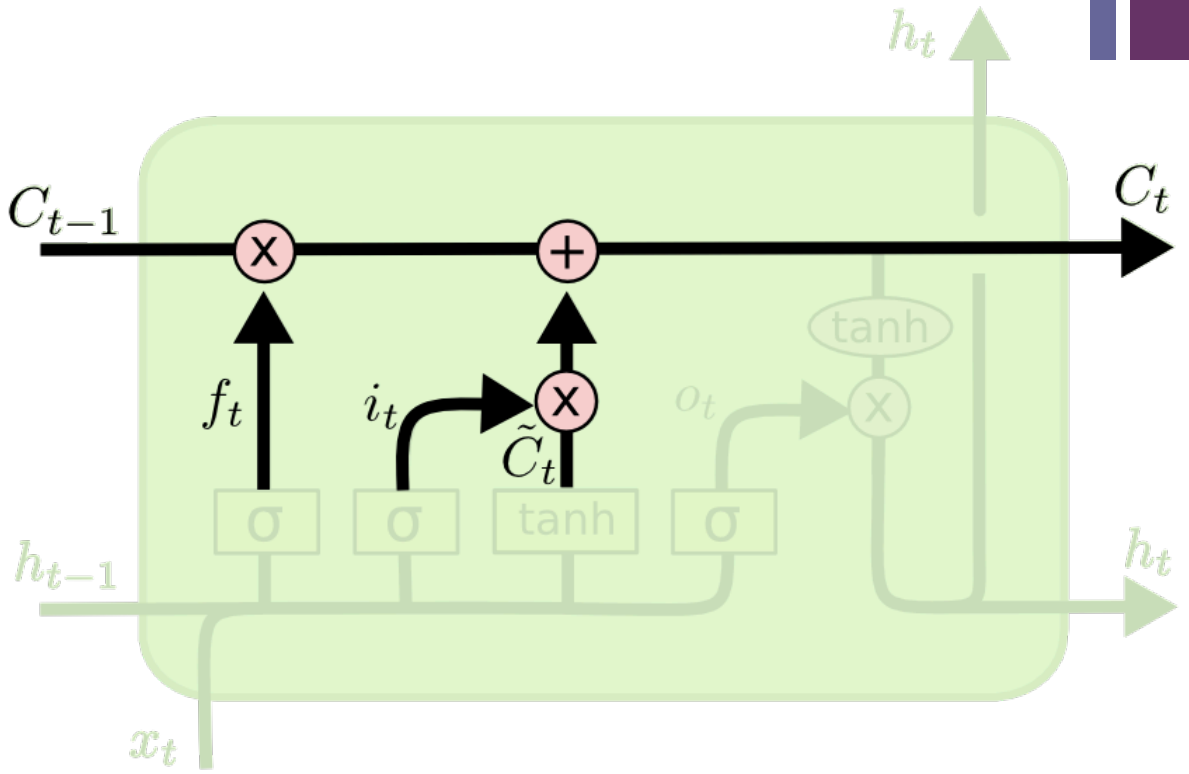
current input

previous hidden state

$$f_t = \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf})$$

Long-Short Term Memory (LSTM) (cont.)

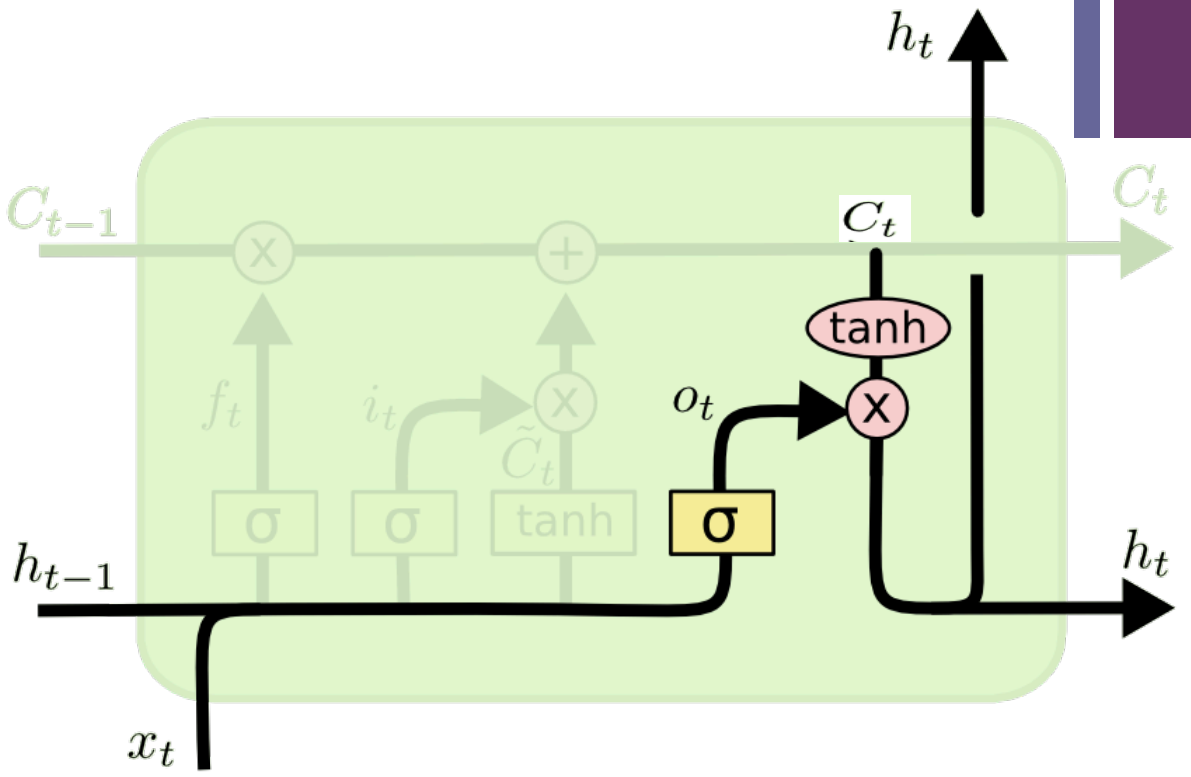
- **Final Memory Cell State:**
  - Final memory cell forgets the previous memory according to the forget gate
  - Final memory cell remembers the new memory according to the input gate



$$c_t = f_t * c_{(t-1)} + i_t * \tilde{c}_t$$

Long-Short Term Memory (LSTM) (cont.)

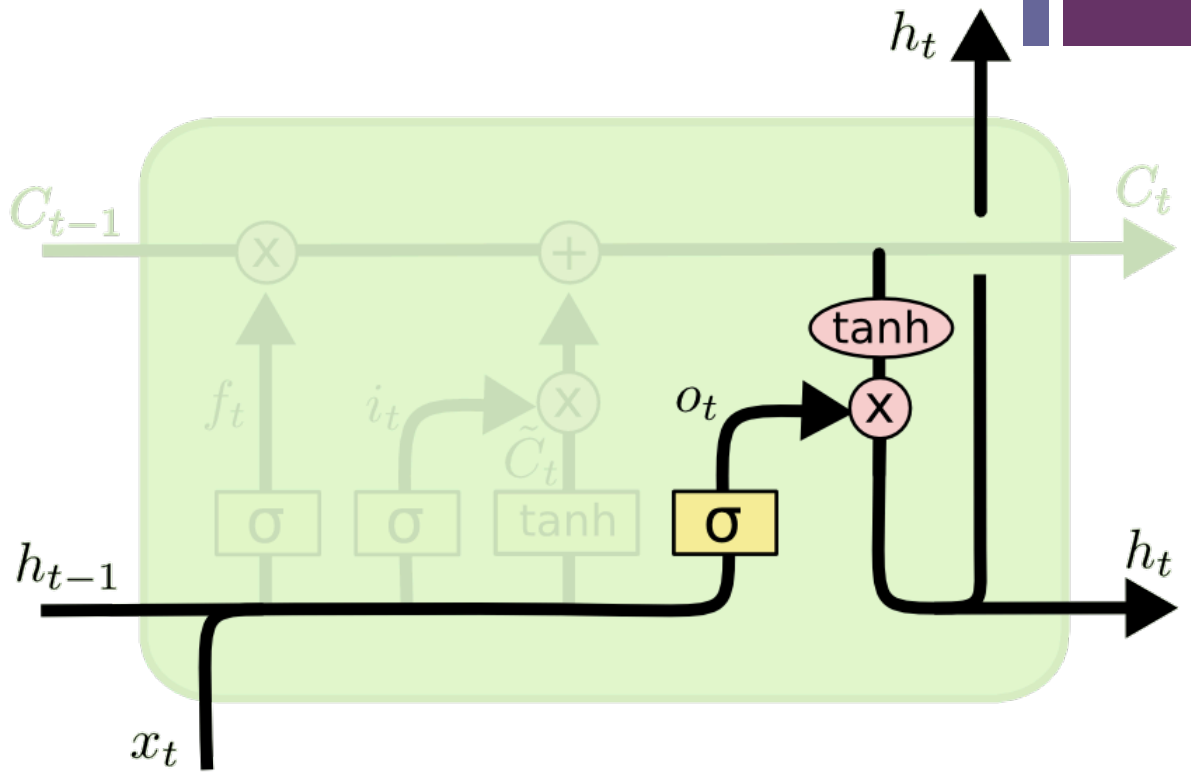
- **Output Gate:** determine how much the final memory should it pass to the next time step according to previous hidden state and the current input



$$o_t = \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho})$$

Long-Short Term Memory (LSTM) (cont.)

- **Final hidden state** captures the information from the final memory cell according to the output gate



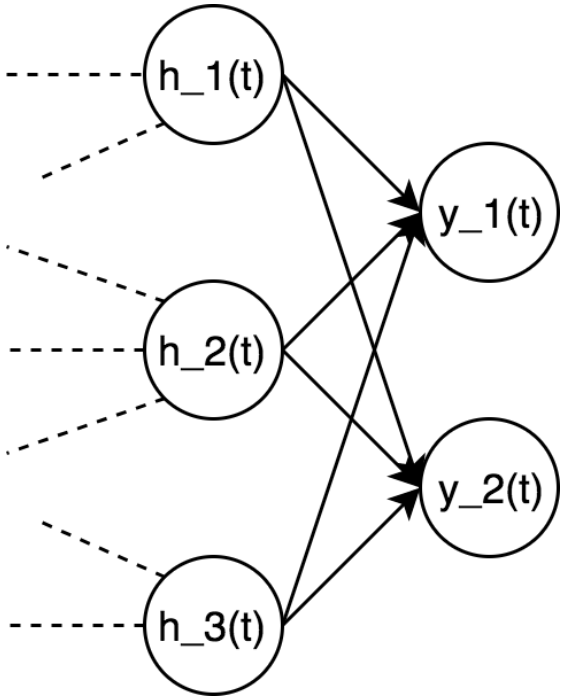
$$h_t = o_t * \tanh(c_t)$$

Long-Short Term Memory (LSTM) (cont.)

- Final hidden state to output:
  - In order to get an output, we need to pass the final hidden state through another linear layer to shrink down the number of dimensions to the number of possible categories.
  - Example:

$$\begin{bmatrix} w_1(1) & w_1(2) & w_1(3) \\ w_2(1) & w_2(2) & w_2(3) \end{bmatrix} * \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

[2×3] [3×1] [2×1]



Long-Short Term Memory (LSTM) (cont.)

- Final hidden state to output:
  - Then we pass the outputs to softmax function to convert the scores to probabilities
  - Example:

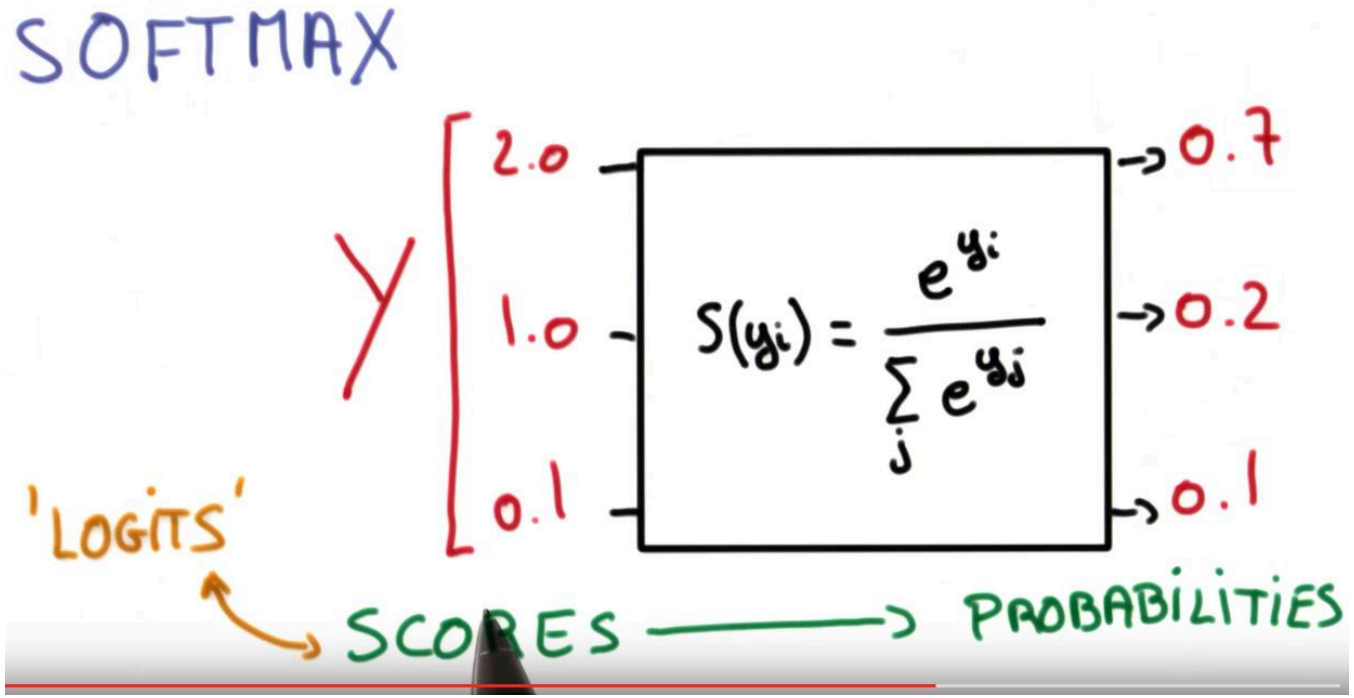


Image reference: https://becominghuman.ai/softmax-part-1-d9625ed1f9c6

Lab2: Sentiment Analysis with LSTM

- Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative).
- Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers).
- For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data.
- This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               53200
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 213,301
Trainable params: 213,301
Non-trainable params: 0
_________________________________________________________________
None
```

Lab3: Try to implement your LSTM architecture (Exercise)

- **First GOP Debate Twitter Sentiment**

- We looked through tens of thousands of tweets about the early August GOP debate in Ohio and asked contributors to do both sentiment analysis and data categorization.

- Contributors were asked if the tweet was relevant, which candidate was mentioned, what subject was mentioned, and then what the sentiment was for a given tweet.

- We've removed the non-relevant messages from the uploaded dataset.