CSCI 567: Machine Learning

Vatsal Sharan

Fall 2022

Lecture 7, Oct 20

USC University of Southern California

Administrivia

- Quiz 1 grades will be released soon.
  - Linear algebra tip: Whenever you see or write a matrix-matrix or matrix-vector product, double check to make sure the dimensions match.

Make sure none of your linear-algebra operations are caught by the “matrix police”...

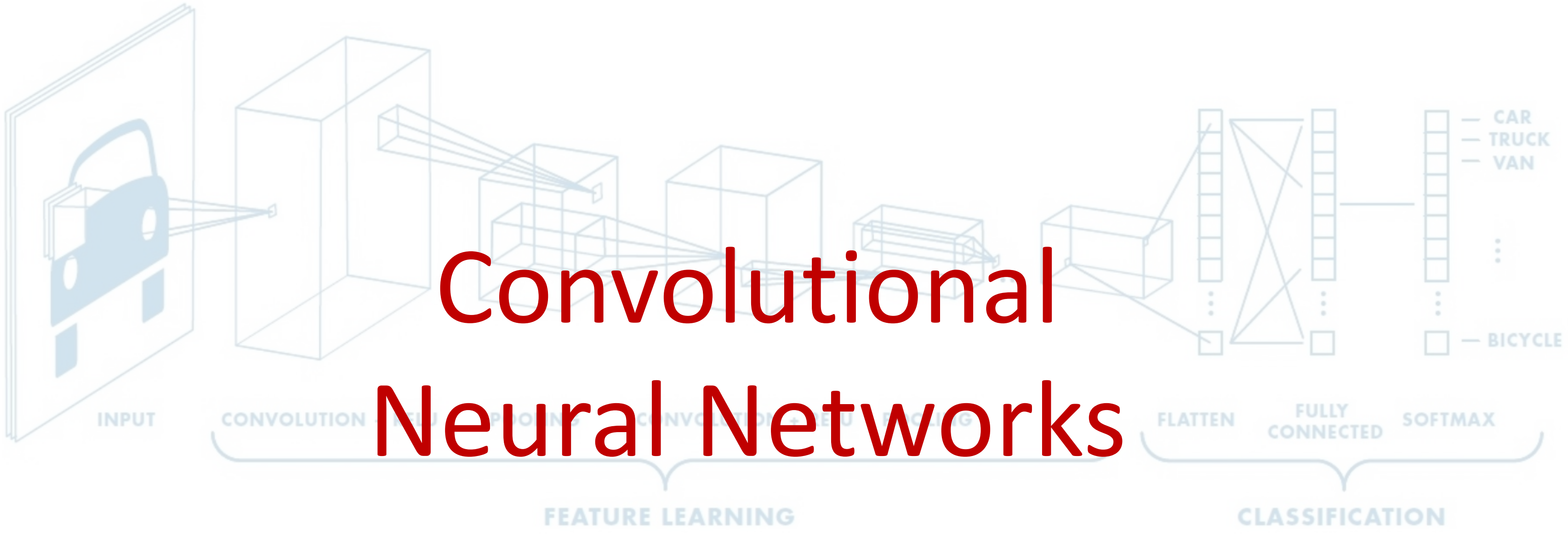
Administrivia

- Quiz 1 grades will be released soon.
  - Linear algebra tip: Whenever you see or write a matrix-matrix or matrix-vector product, double check to make sure the dimensions match.

- Project details will be released in 1-2 weeks (Kaggle competition).
- Groups of 4 (start forming groups)

- Today's plan:
  - Convolutional neural networks
  - Sequential prediction, Markov models and (a bit of) recurrent neural networks

Convolutional Neural Networks



Acknowledgements

Not much math in this part, but there'll be empirical intuition (and cat pictures ☺)

The materials in this part borrow heavily from the following sources:

- Stanford's CS231n: http://cs231n.stanford.edu/
- Deep learning book by Goodfellow, Bengio and Courville: http://deeplearningbook.org

Both website provides a lot of useful resources: notes, demos, videos, etc.

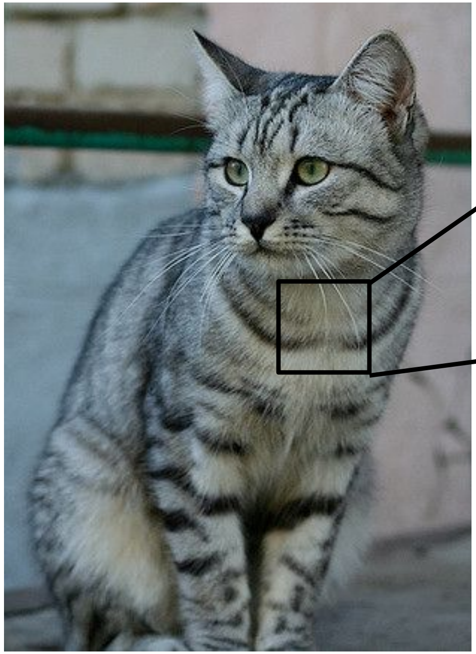
Image Classification: A core task in Computer Vision

input : How is this represented ?

(assume given set of discrete labels)
$\underbrace{\{\text{dog, cat, truck, plane, ...}\}}_{\text{multiclass classification problem}}$

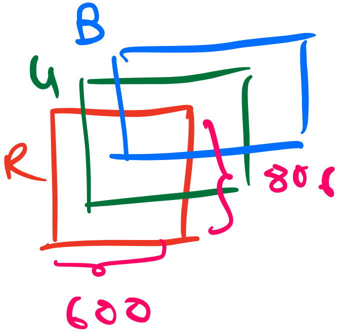
→ cat

The Problem: Semantic Gap




```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

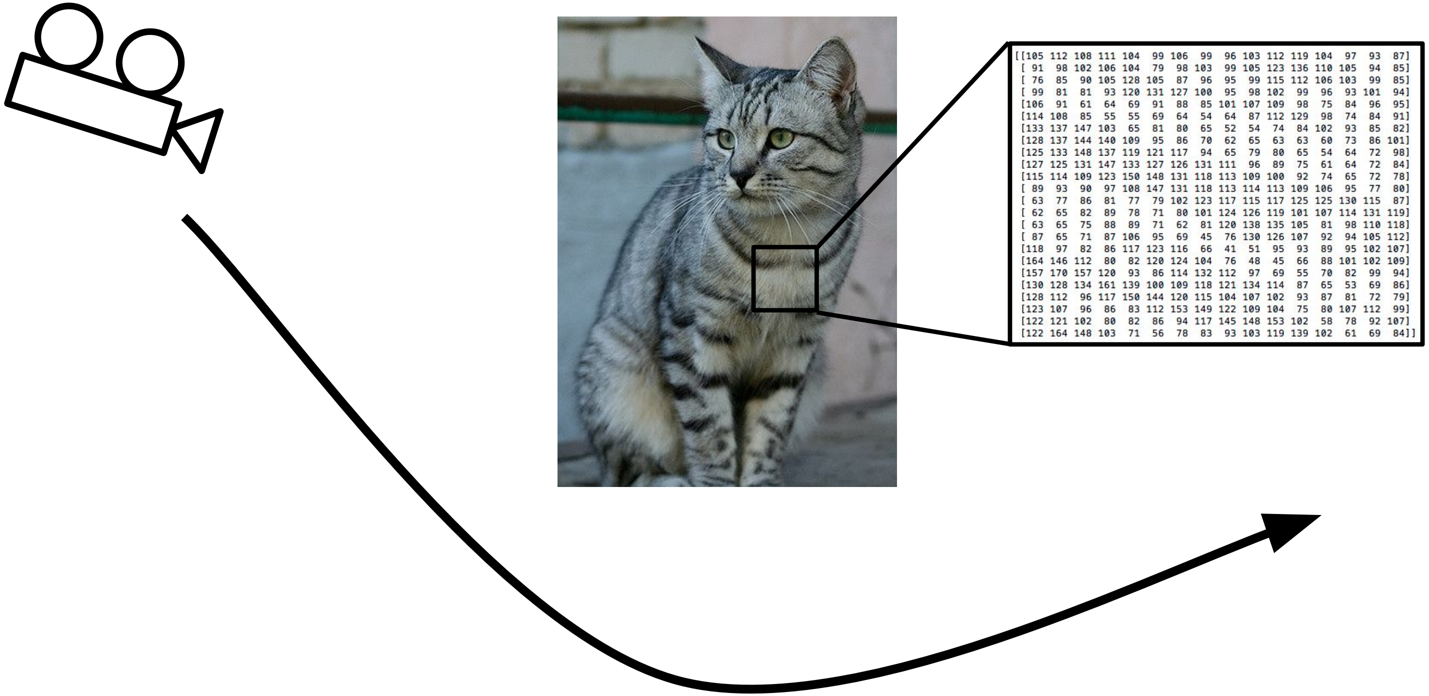
What the computer sees



An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation



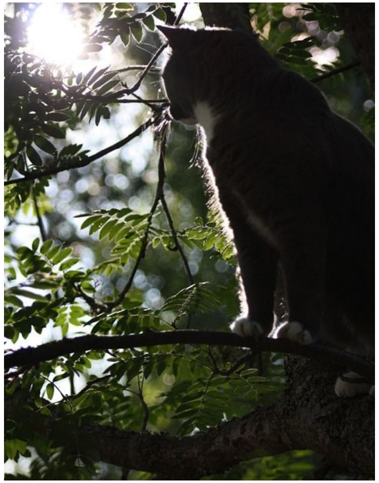
All pixels change when the camera moves!

Challenges: Illumination

This image is CC0 1.0 public domain

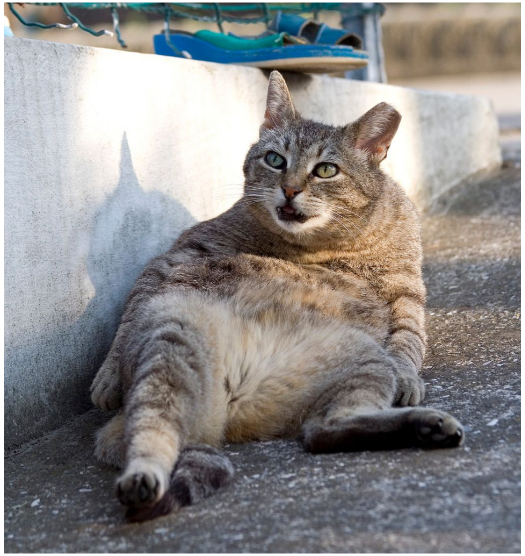
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

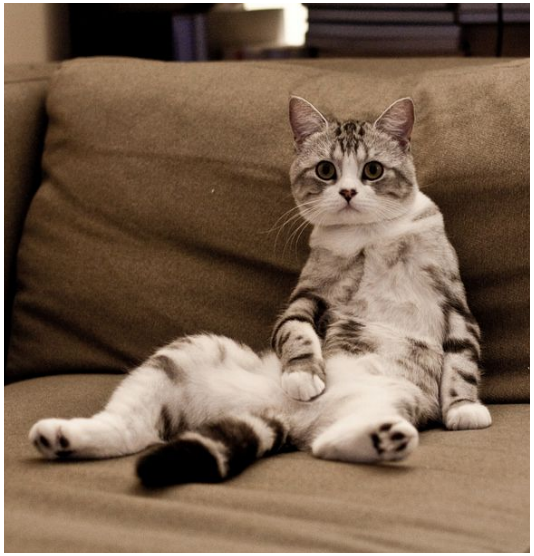
Challenges: Deformation



This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

Challenges: Background Clutter

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

Challenges: Intraclass variation

This image is CC0 1.0 public domain

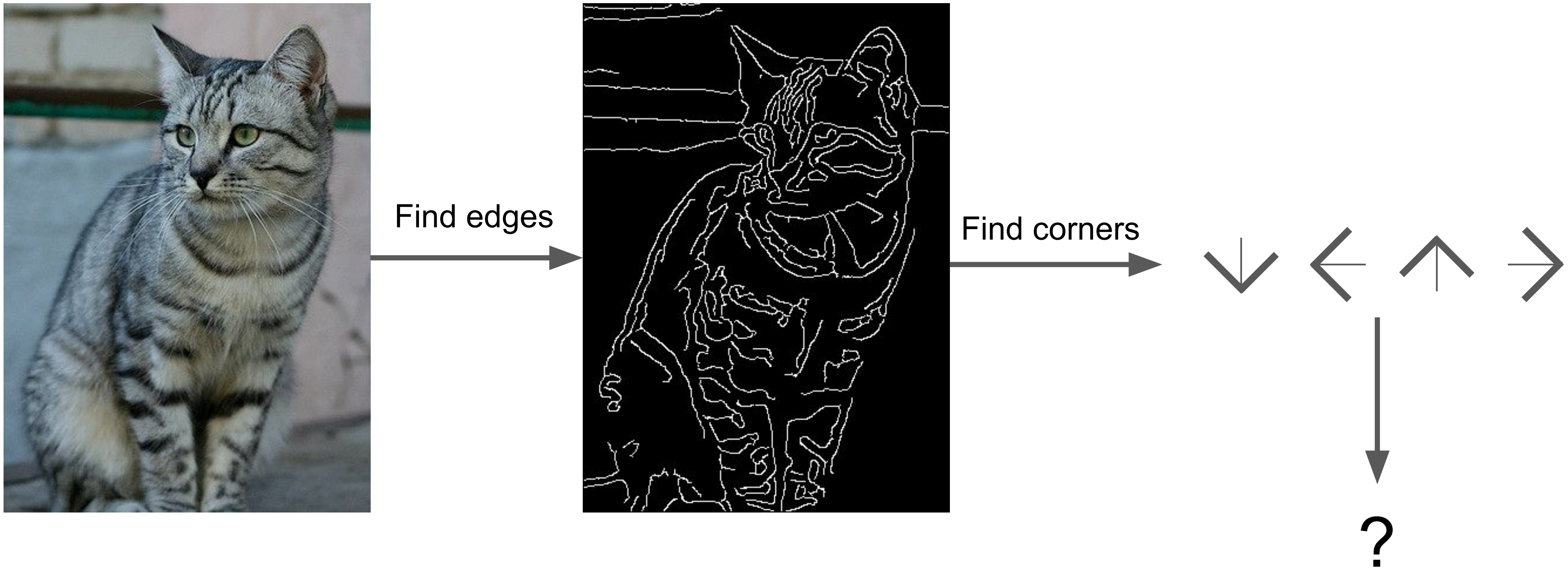
An image classifier

```python
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Attempts have been made



John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

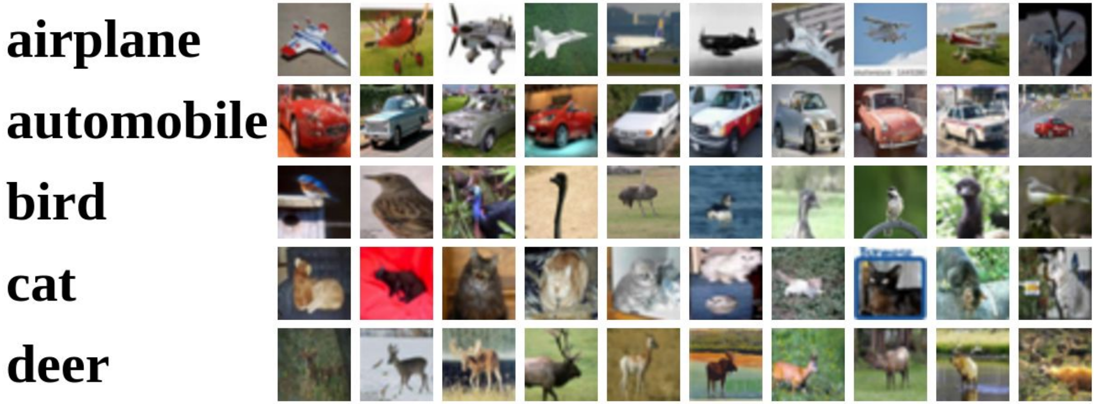
Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

Example training set



The challenge

How do we train a model that can do well despite all these variations?

The ingredients:

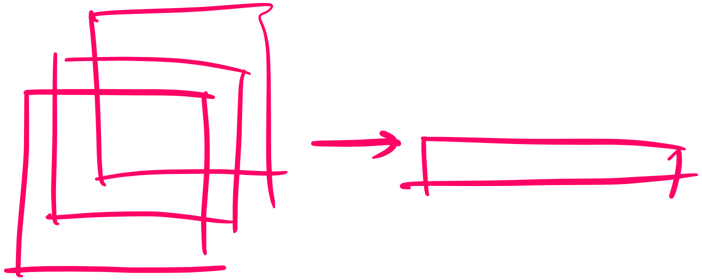
- ***A lot of data*** (so that these variations are observed).
- ***Huge models*** with the capacity to consume and learn from all this data (and the ***computational infrastructure*** to enable training)

What helps:

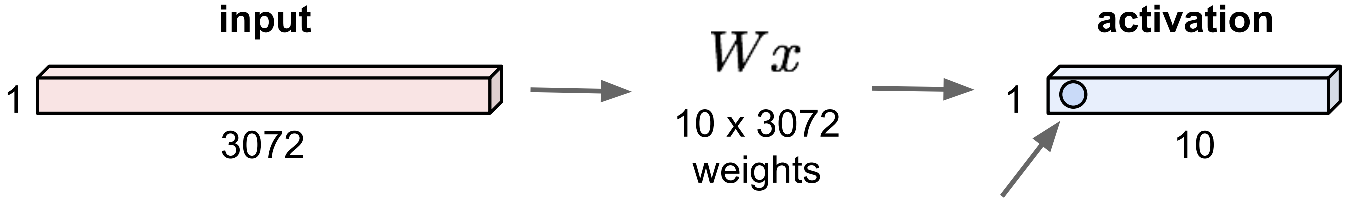
- Models with the right properties which makes the process easier (goes back to our discussion of *choosing the function class*).

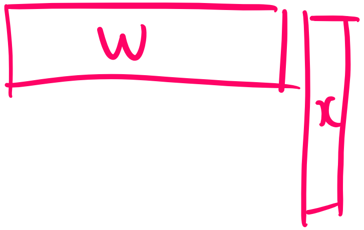
The problem with standard NN for image inputs

Fully Connected Layer



32x32x3 image -> stretch to 3072 x 1



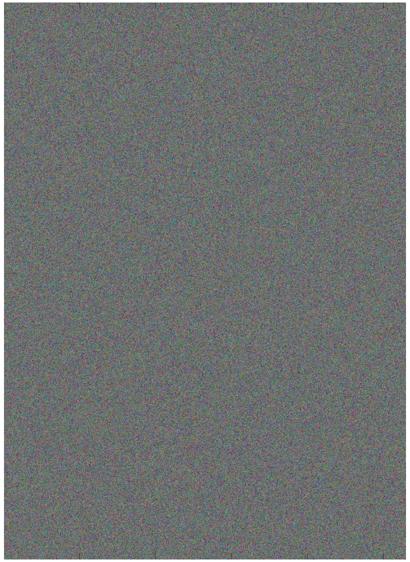


1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

Completely loses out on spatial structure

The task is as easy, or rather as difficult, for a fully-connected network even if I shuffle the pixels.
Is this okay?

A shuffling/ permutation
of the pixels



Solution: Convolutional Neural Net (ConvNet/CNN)

A special case of fully connected neural nets.

Usually consist of convolution layers, ReLU layers, pooling layers, and regular fully connected layers

Key idea: learning from low-level to high-level features

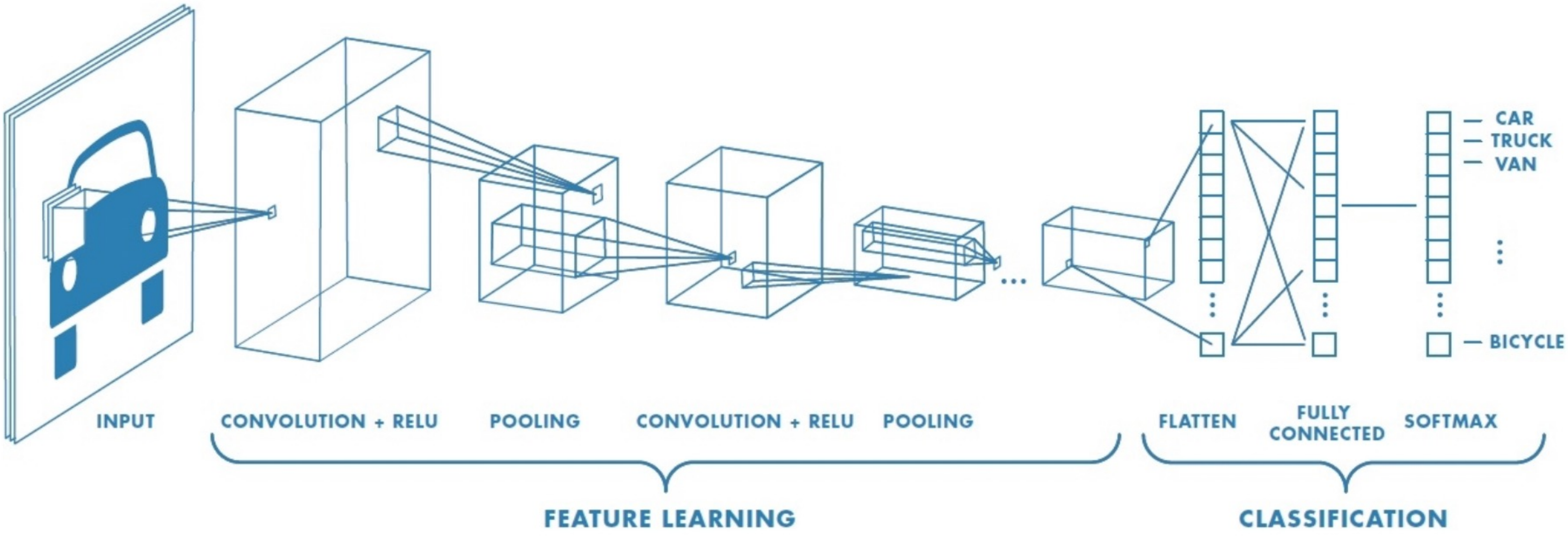


Figure from https://blog.floydhub.com/building-your-first-convnet/

2-D Convolution

$0 \cdot 0 + 1 \cdot 1 + 3 \cdot 2 + 4 \cdot 3$

this operation is convolution

input
(3×3)

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

"filter"

$1 \cdot 0 + 2 \cdot 1 + 4 \cdot 2 + 5 \cdot 3$

Figure 14.5: Illustration of 2d cross correlation. Generated by conv2d_jax.ipynb. Adapted from Figure 6.2.1 of [Zha+20].

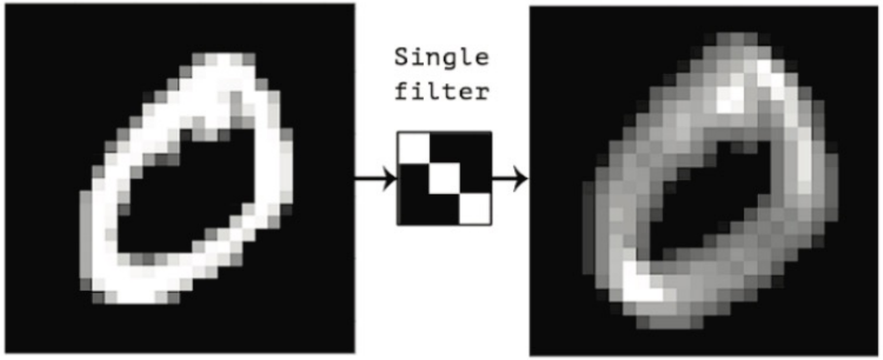


Figure 14.6: Convolving a 2d image (left) with a 3×3 filter (middle) produces a 2d response map (right). The bright spots of the response map correspond to locations in the image which contain diagonal lines sloping down and to the right. From Figure 5.3 of [Cho17]. Used with kind permission of Francois Chollet.

Figures from PML

3-D Convolution

The input is 3+3+2 (2 channels)

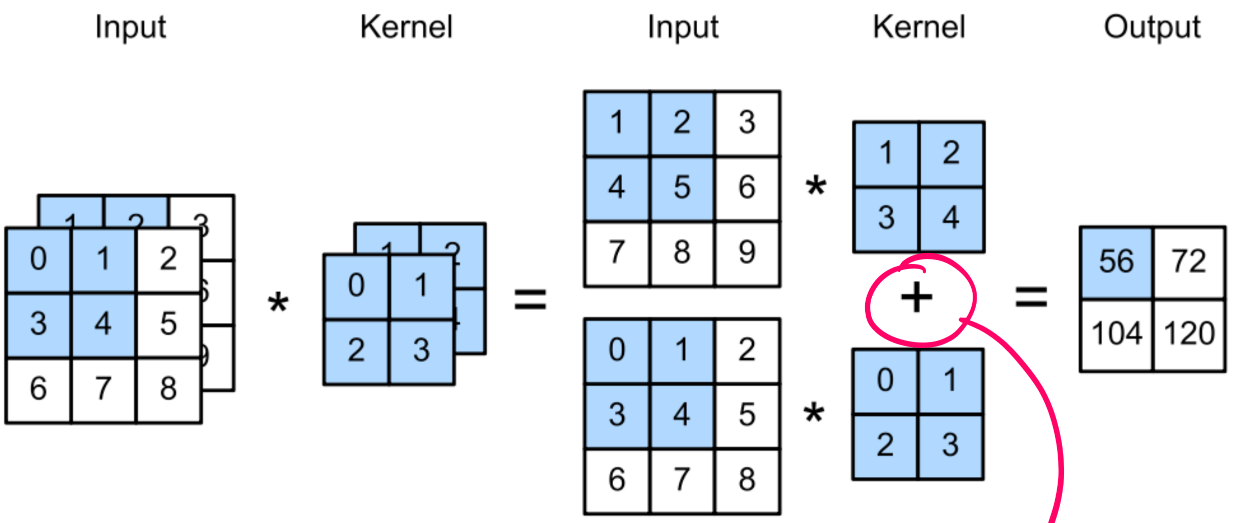


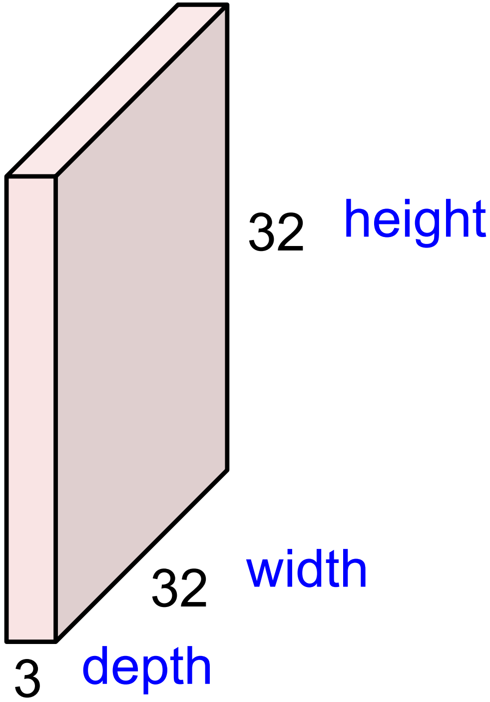
Figure 14.9: Illustration of 2d convolution applied to an input with 2 channels. Generated by conv2d_jax.ipynb. Adapted from Figure 6.4.1 of [Zha+20].

add up the result for the two channels

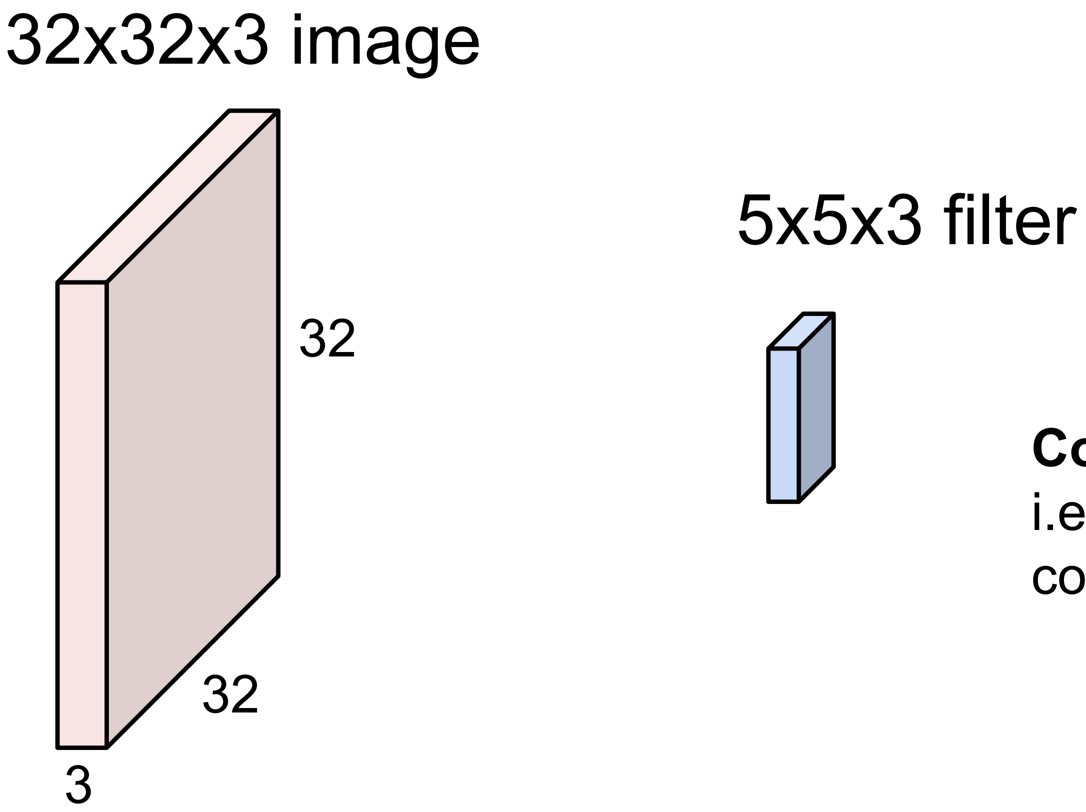
Figure from PML

Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

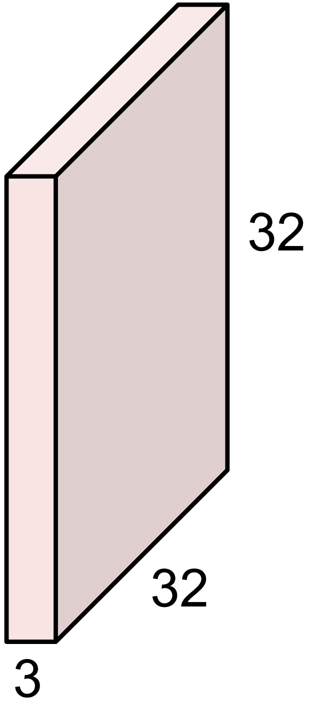


Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

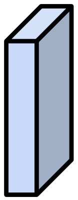
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

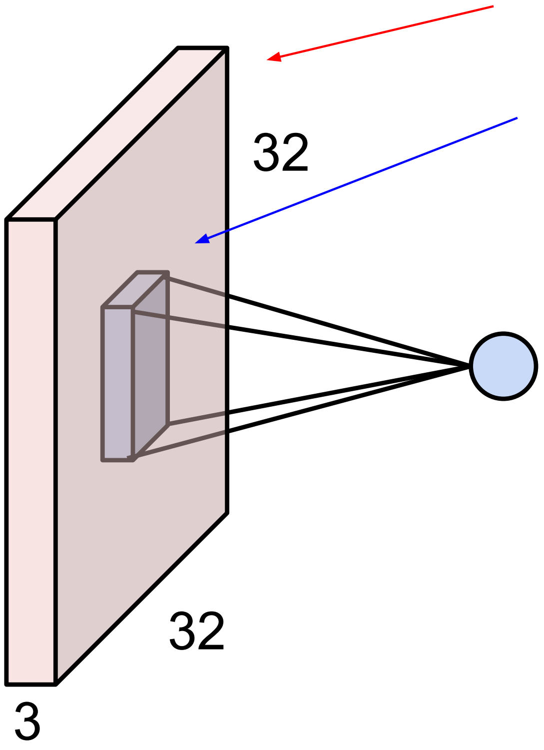


5x5x3 filter



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

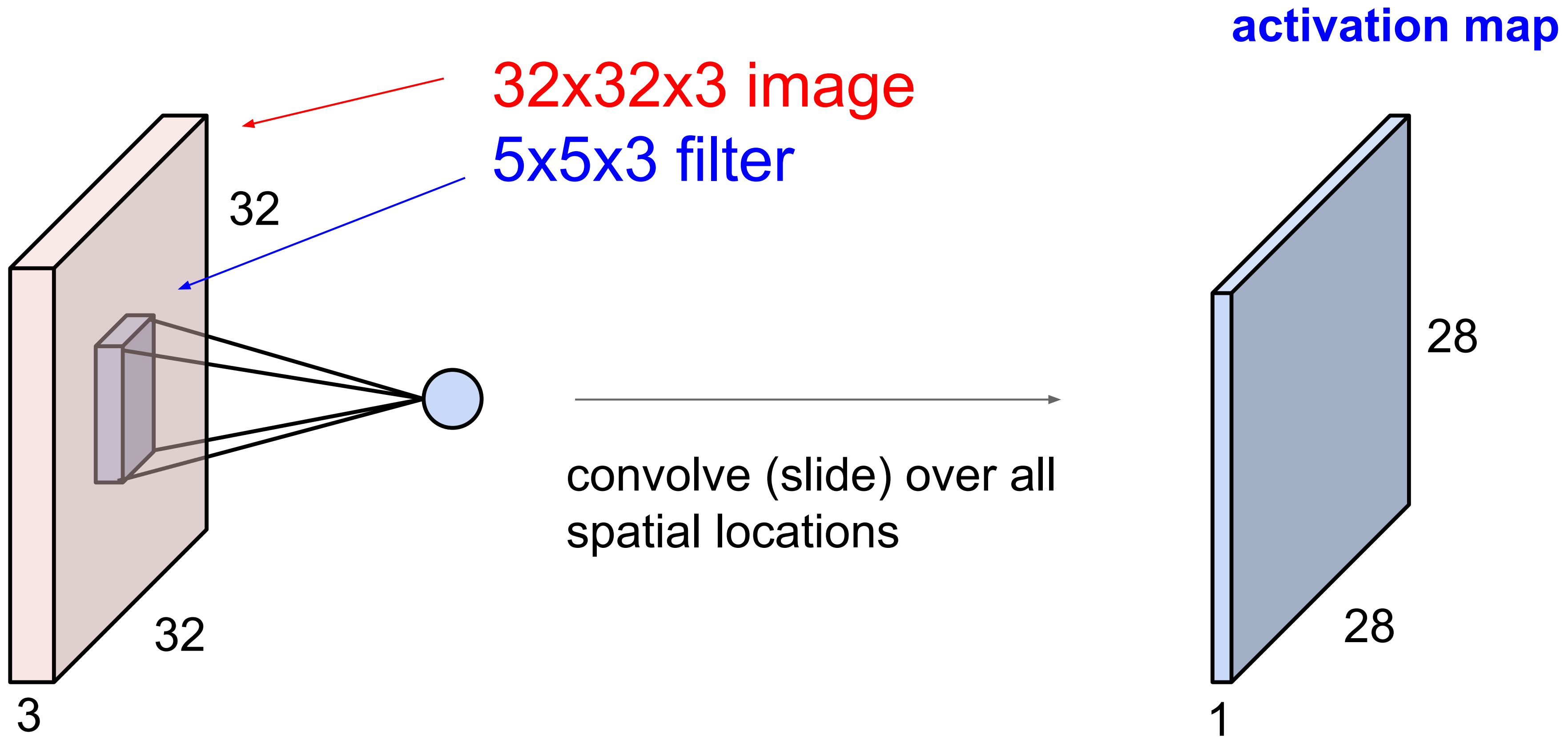
Convolution Layer



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)

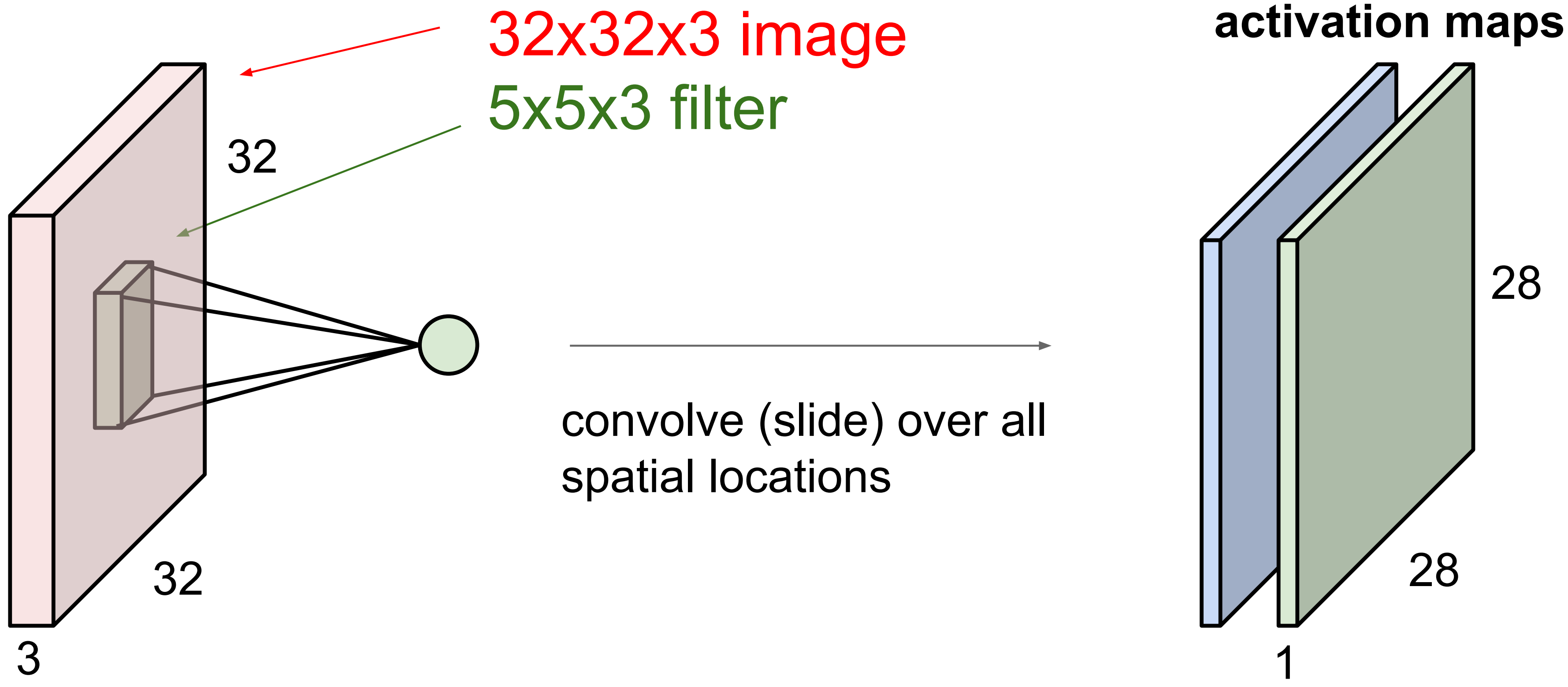
$$w^T x + b$$

Convolution Layer

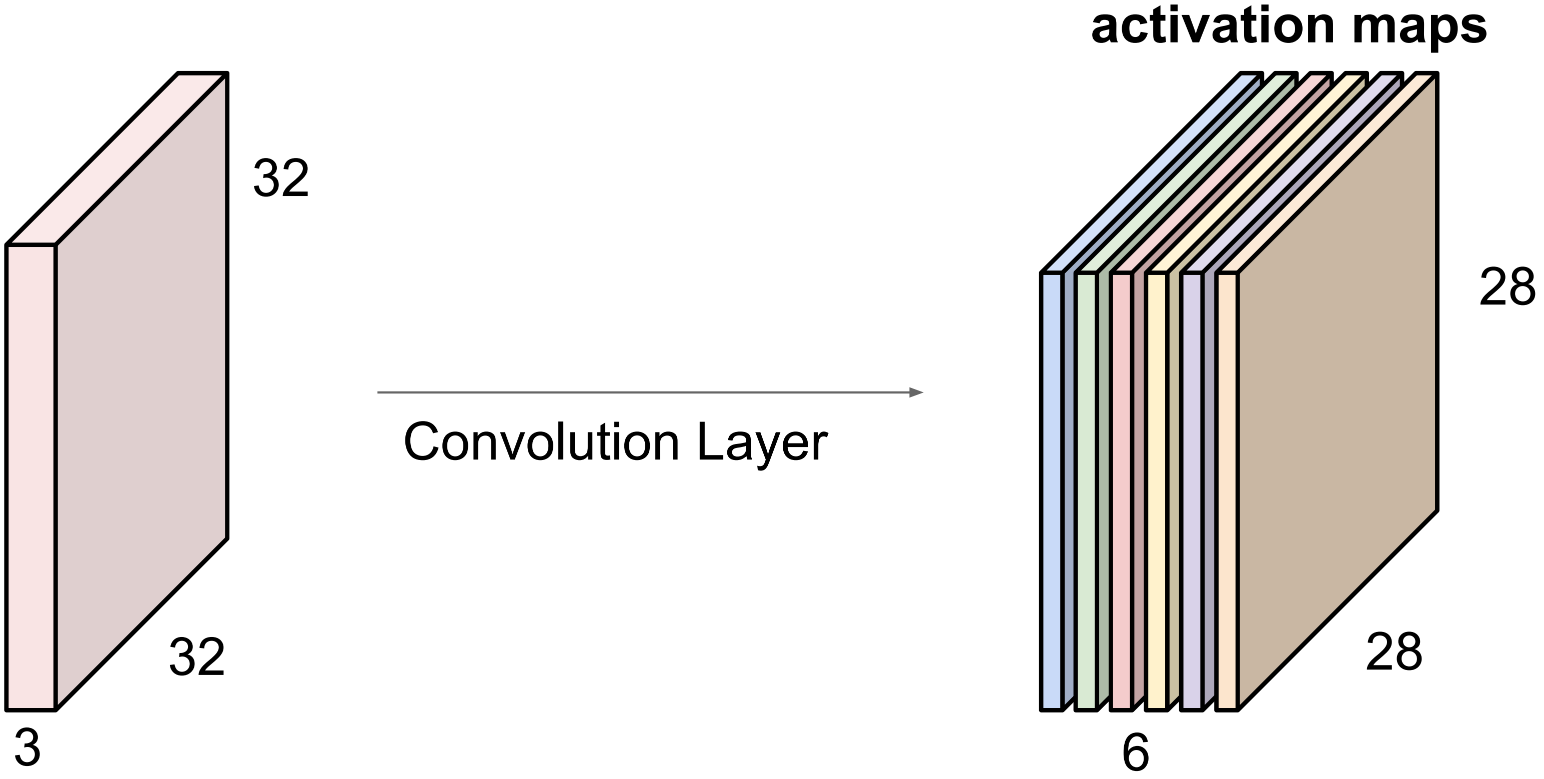


Convolution Layer

consider a second, green filter

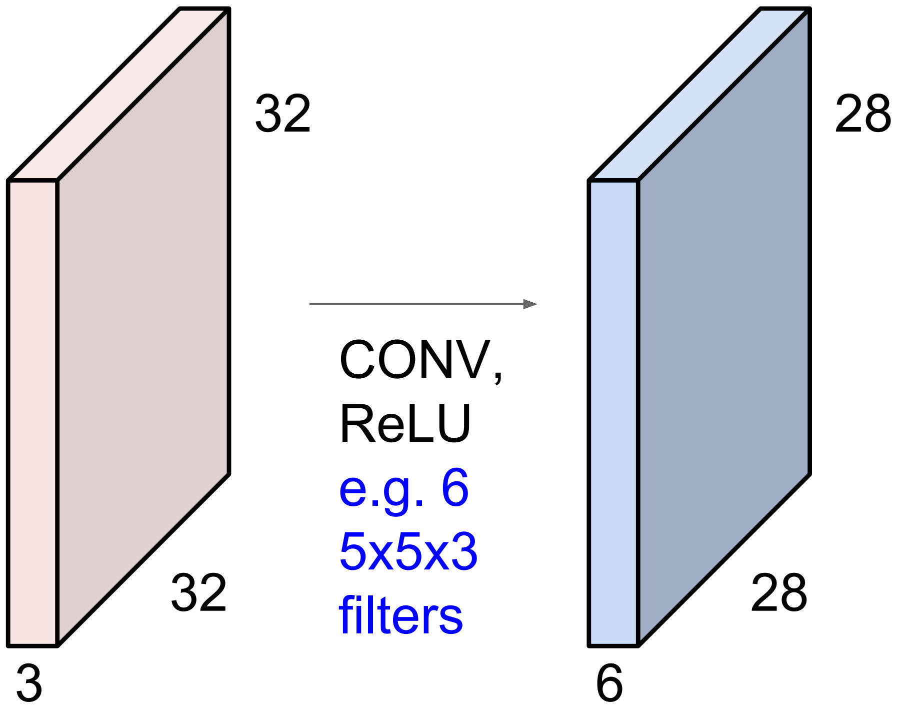


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

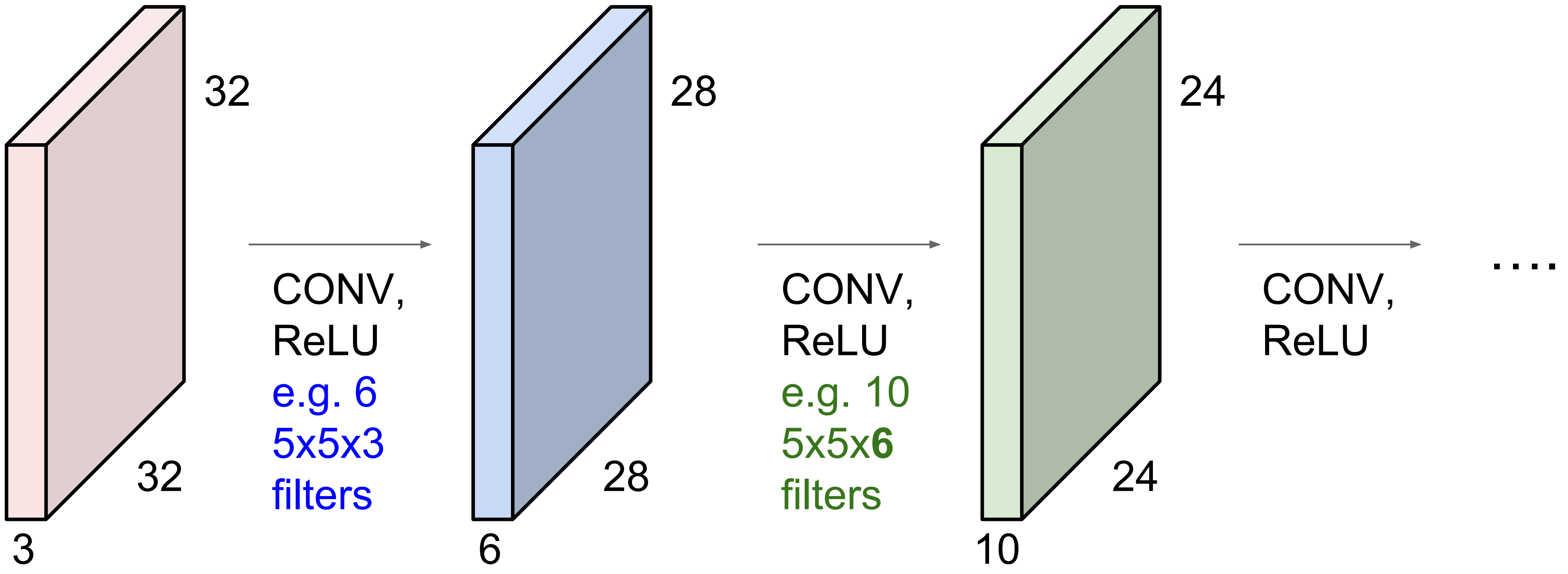


We stack these up to get a "new image" of size 28x28x6!

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



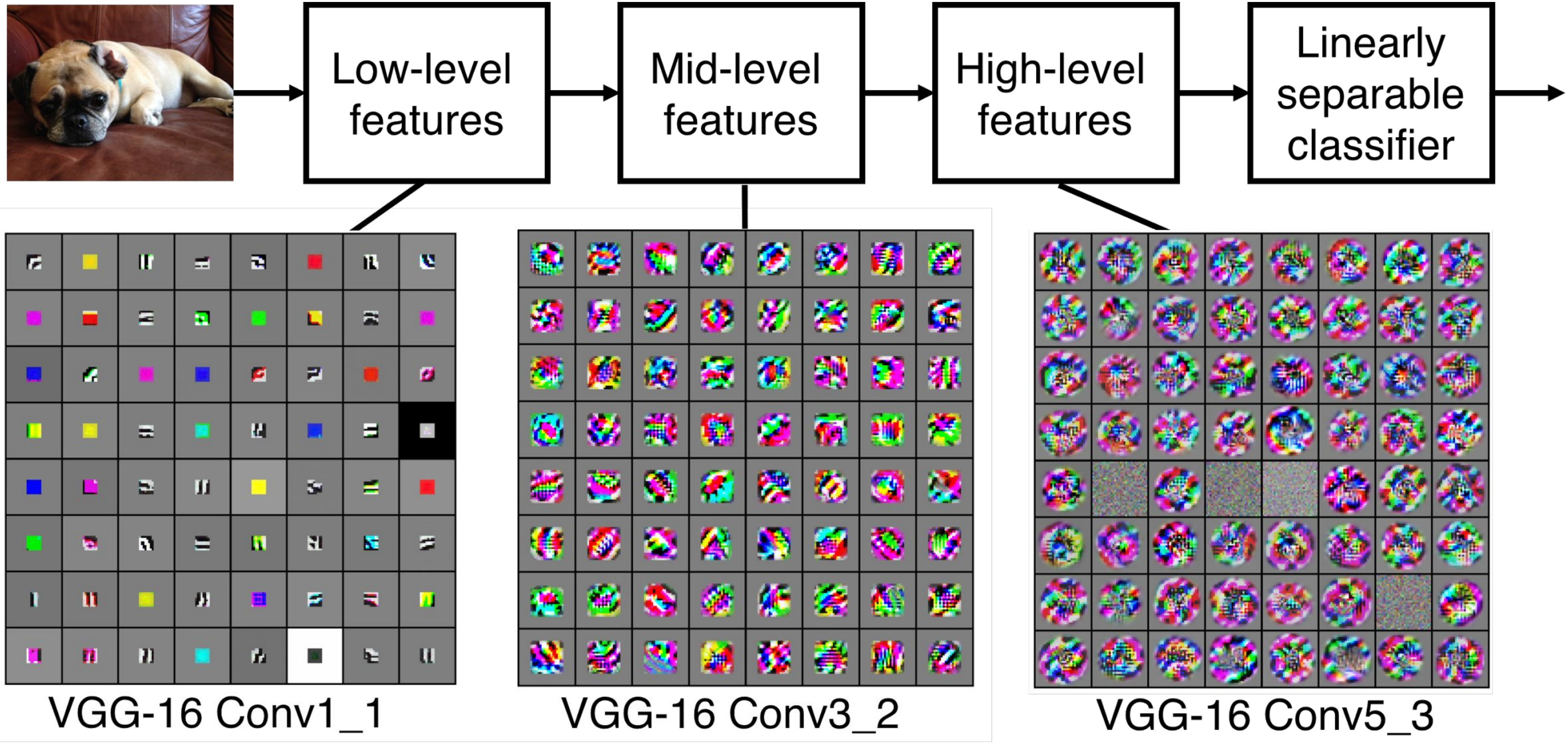
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Preview

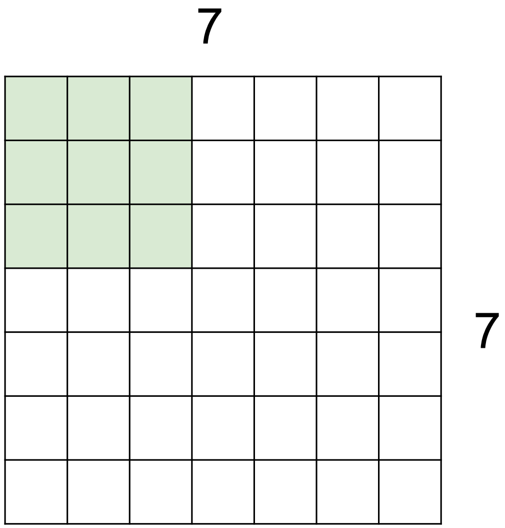
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



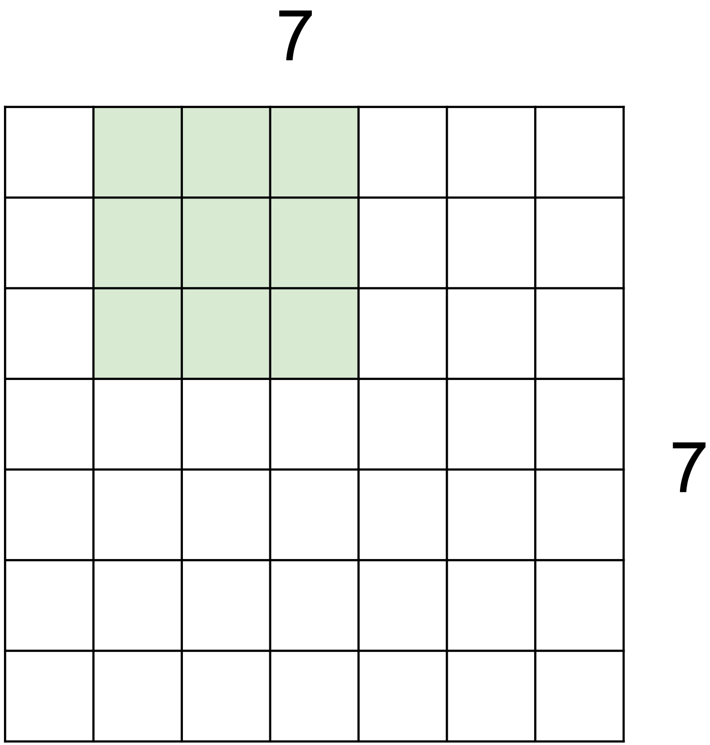
Understanding spatial dimensions of Conv layer

A closer look at spatial dimensions:



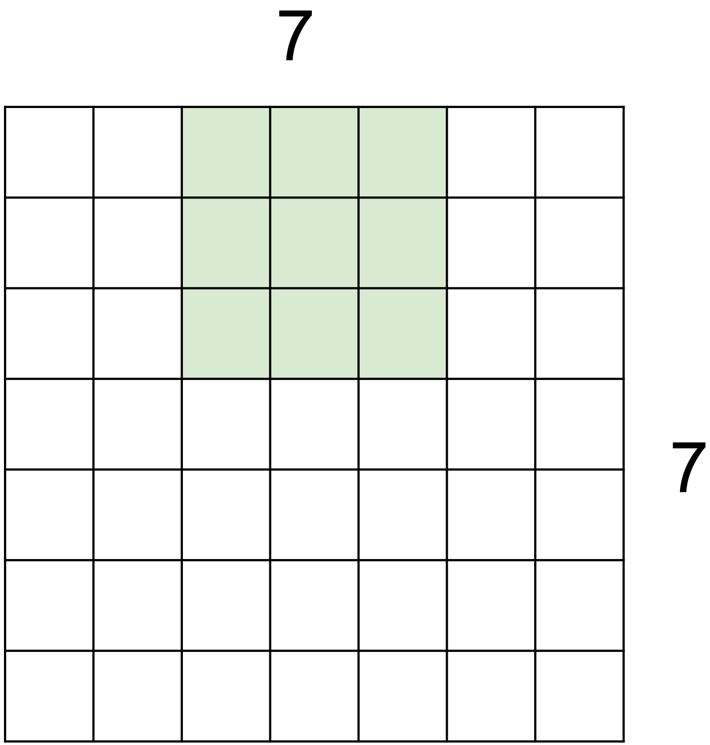
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



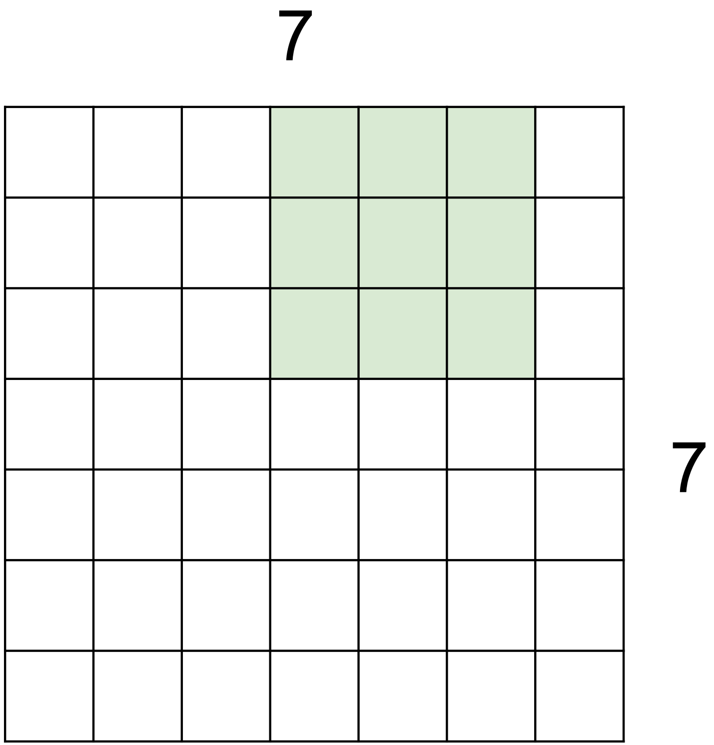
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



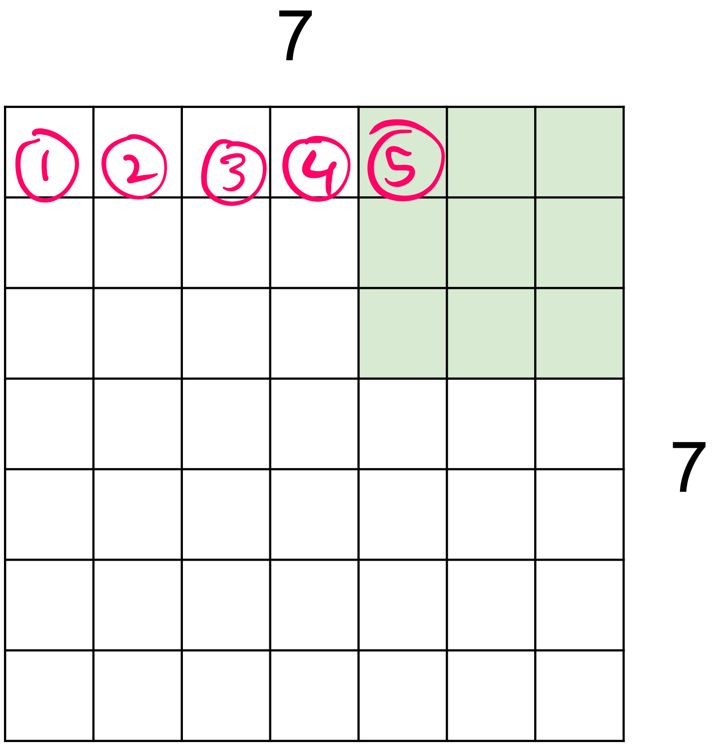
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

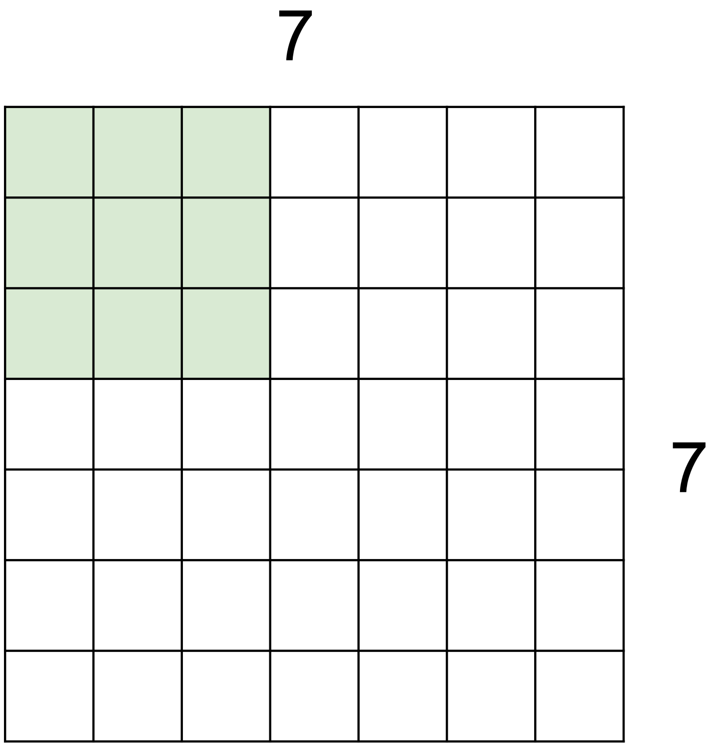
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

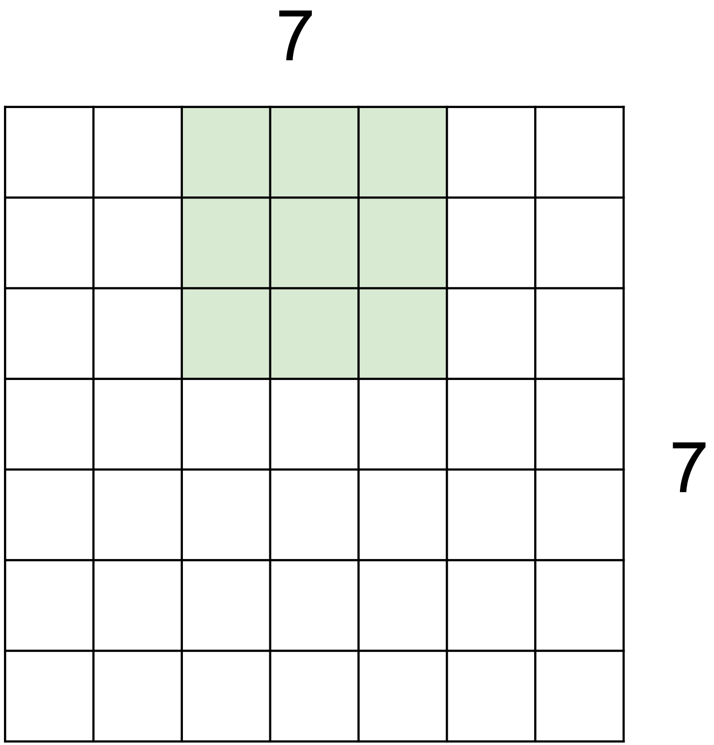
A closer look at spatial dimensions:



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

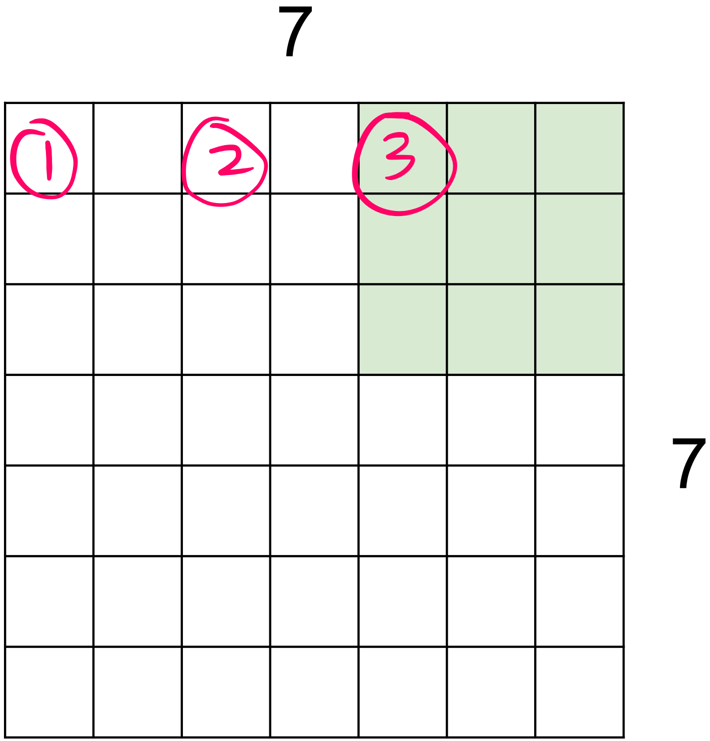
skip 1 pixel in between

A closer look at spatial dimensions:



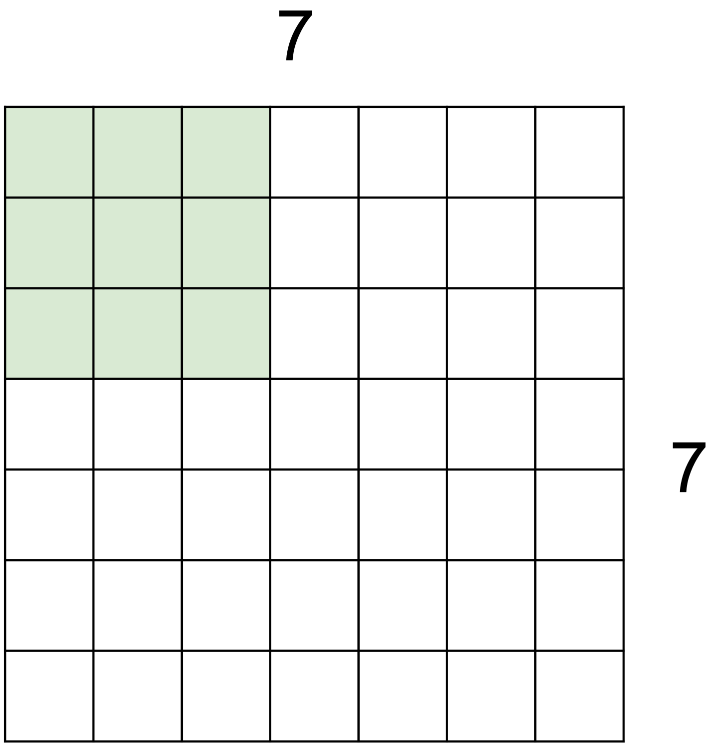
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



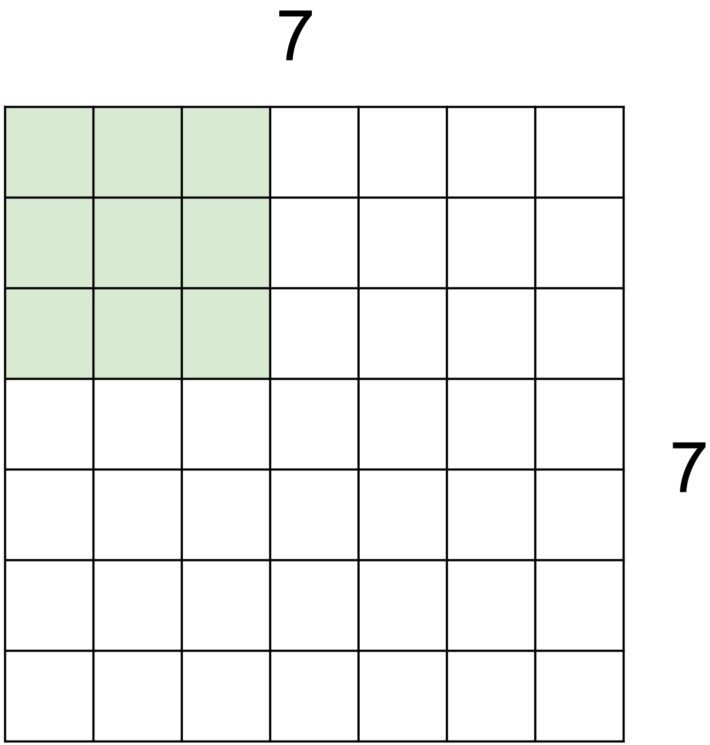
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



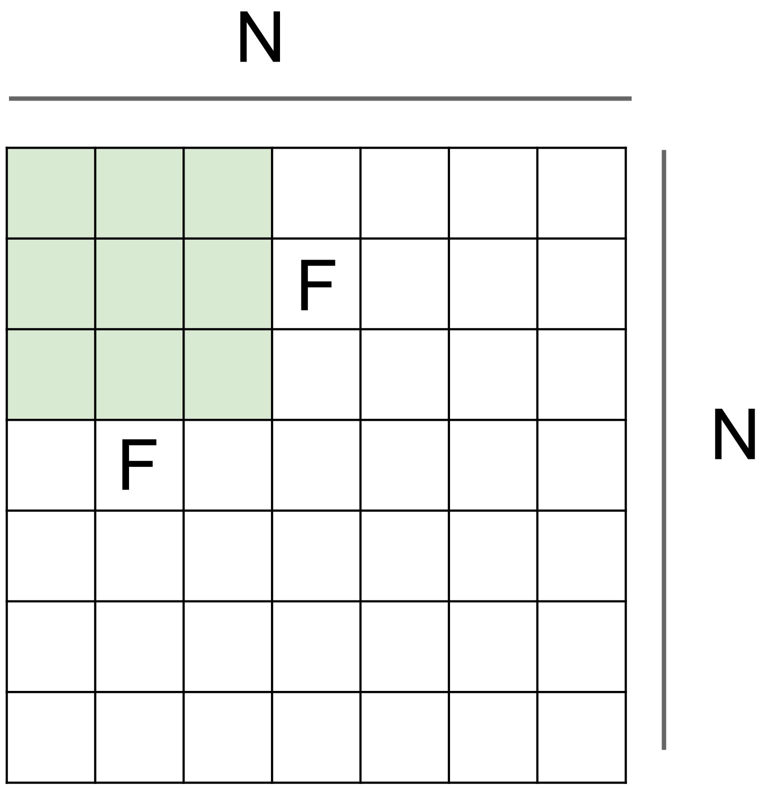
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:



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

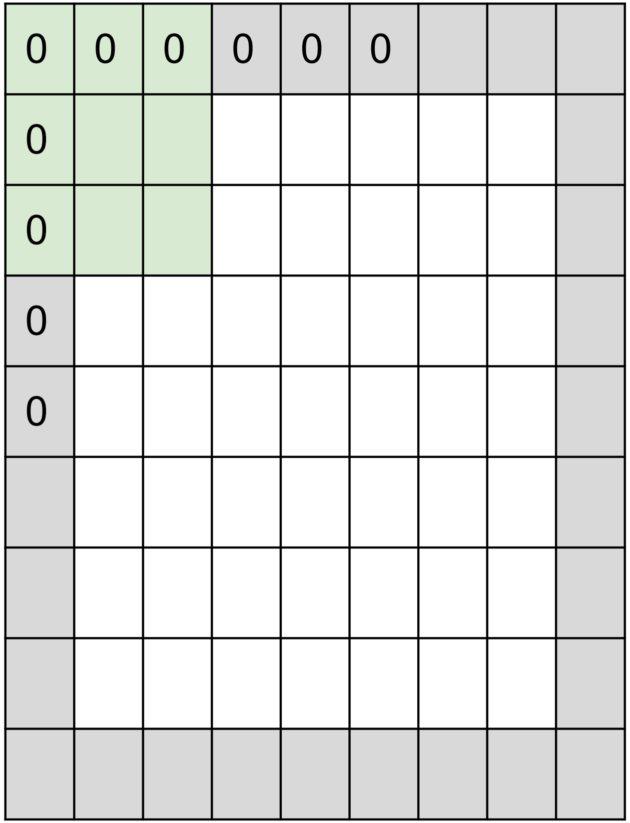
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

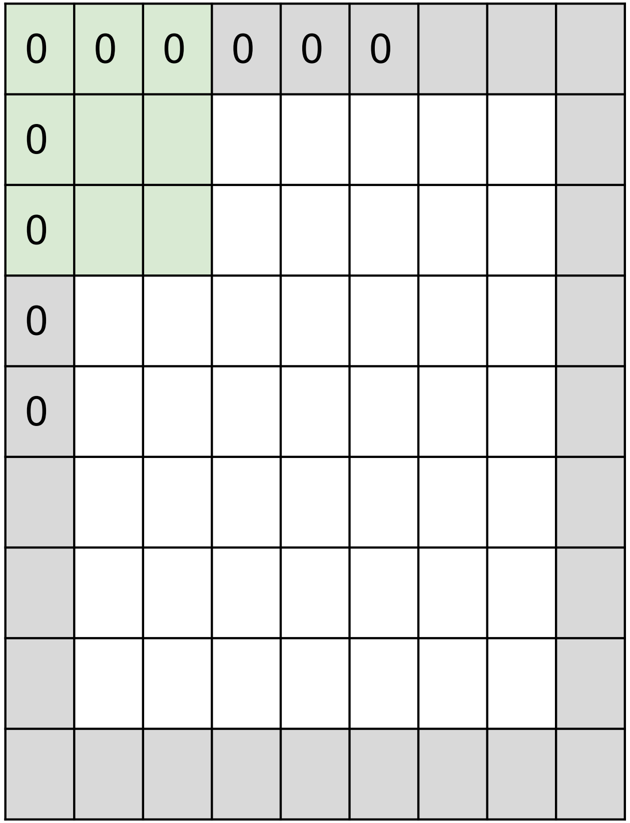
In practice: Common to zero pad the border



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

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

In practice: Common to zero pad the border



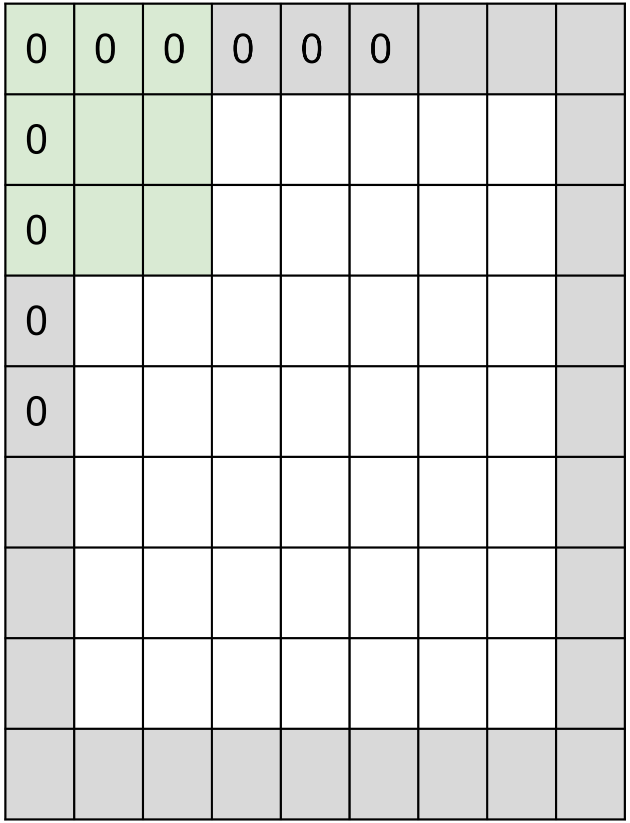
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

$(N - F + 1)$

$(9 - 3 + 1 = 6)$

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?

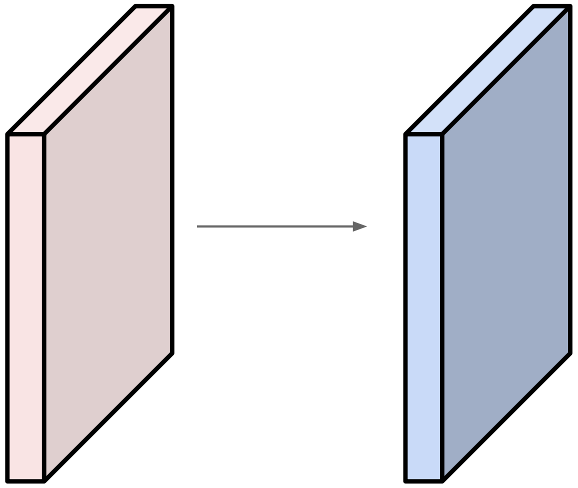
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

$(\underbrace{N+2P-F})/\text{stride}+1$

$N-1+1=N$

Examples time:

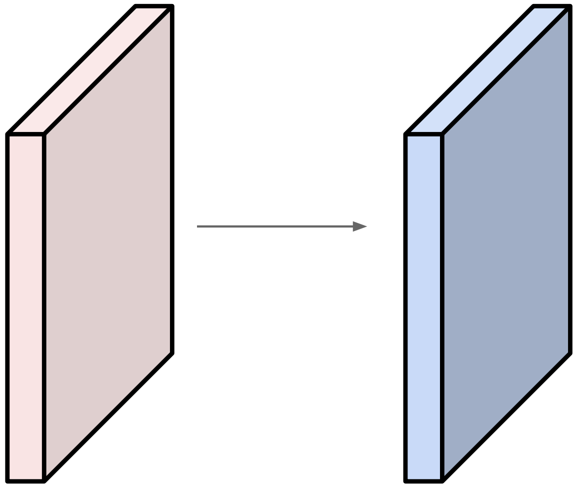


Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?

Examples time:



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

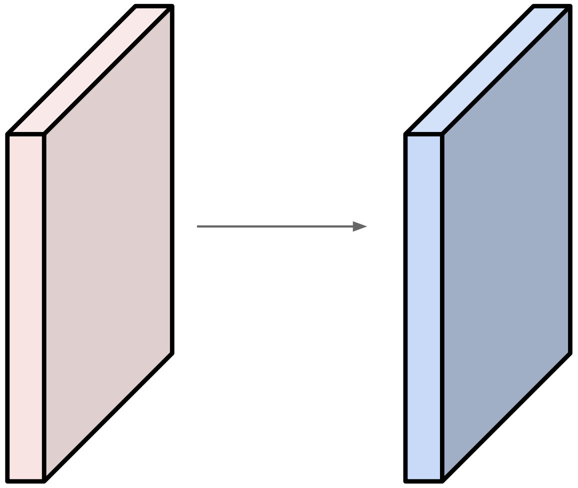
(N+2P-F)/stride +1

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

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**

Summary for convolutional layer

Input: a volume of size $W_1 \times H_1 \times D_1$

Hyperparameters:

- K filters of size $F \times F$
- stride S
- amount of zero padding P (for one side)

Output: a volume of size $W_2 \times H_2 \times D_2$ where

- $W_2 = (W_1 + 2P - F)/S + 1$
- $H_2 = (H_1 + 2P - F)/S + 1$
- $D_2 = K$

#parameters: $(F \times F \times D_1 + 1) \times K$ weights

Common setting: $F = 3, S = P = 1$

Demo time

What is a Convolutional Neural Network?

https://poloclub.github.io/cnn-explainer/

Connection to fully connected networks

A convolutional layer is a special case of a fully connected layer:
filter = weights with sparse connection

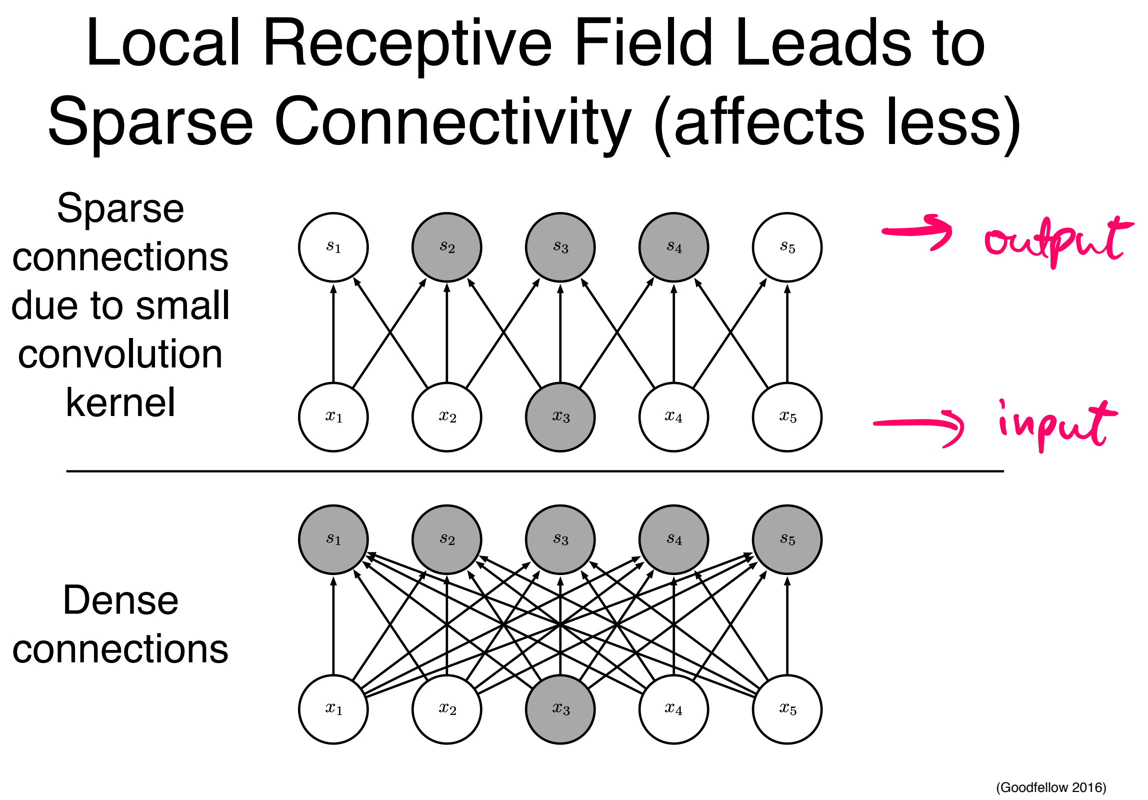


Figure from Goodfellow'16

Connection to fully connected networks

A convolutional layer is a special case of a fully connected layer:
filter = weights with sparse connection

Sparse connectivity: being affected by less

Sparse connections due to small convolution kernel

Dense connections

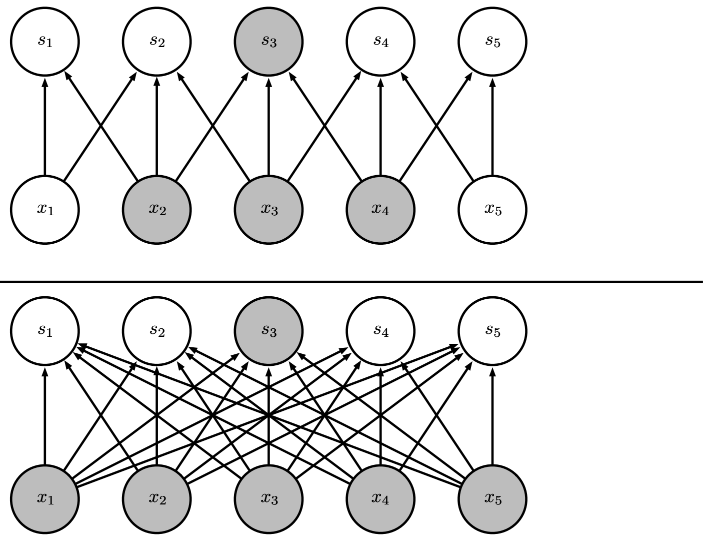


Figure 9.3

(Goodfellow 2016)

Figure from Goodfellow'16

Connection to fully connected networks

A convolutional layer is a special case of a fully connected layer:
filter = weights with sparse connection and parameter sharing

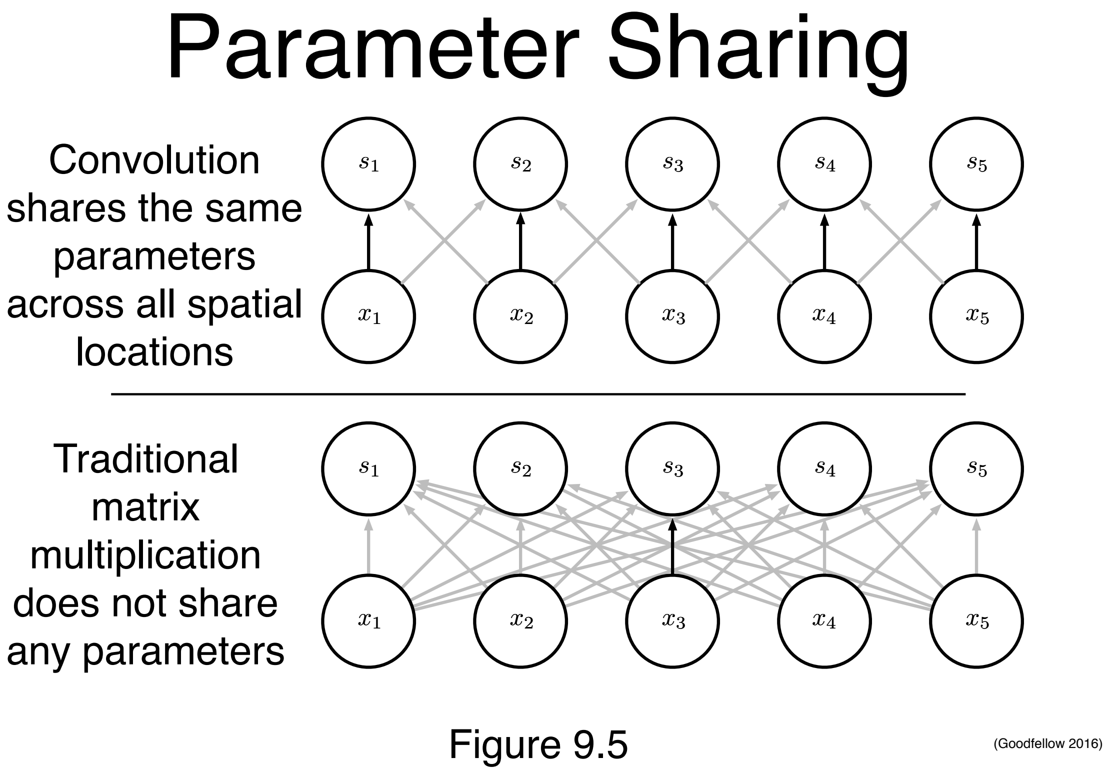


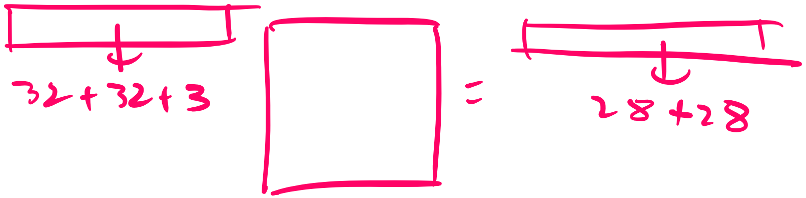
Figure from Goodfellow'16

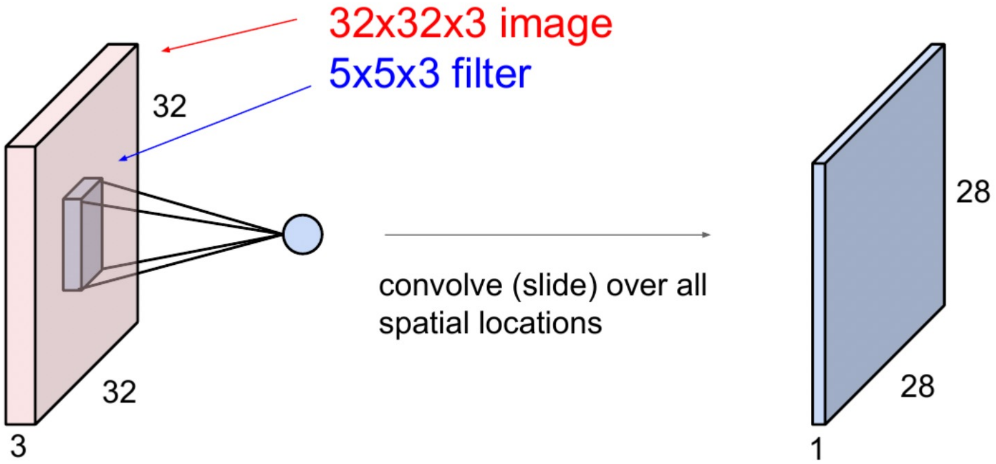
Connection to fully connected networks

A convolutional layer is a special case of a fully connected layer:
filter = weights with sparse connection and parameter sharing

Much fewer parameters! Example (ignoring bias terms):

FC layer: $(32 \times 32 \times 3) \times (28 \times 28) \approx 2.4M$
Conv layer: $5 \times 5 \times 3 = 75$

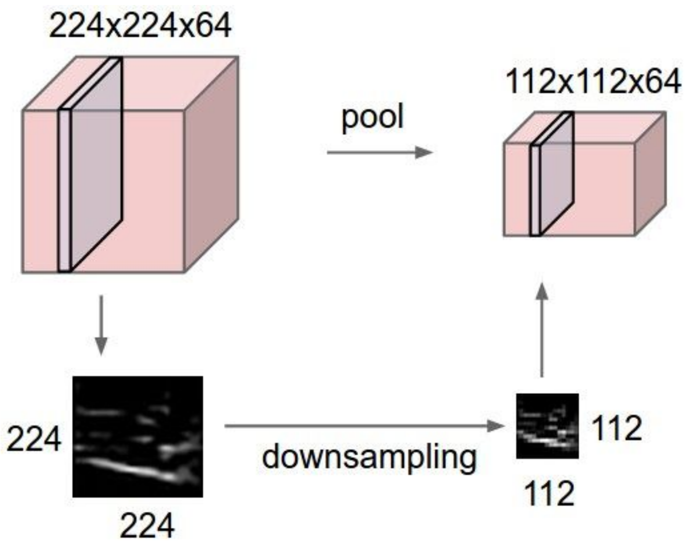




Another element: Pooling

Pooling layer

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



Another element: Pooling

Similar to a filter, except
- depth is always 1
- different operations: average, L2-norm, max
- no parameters to be learned

Max pooling with 2 × 2 filter and stride 2 is very common

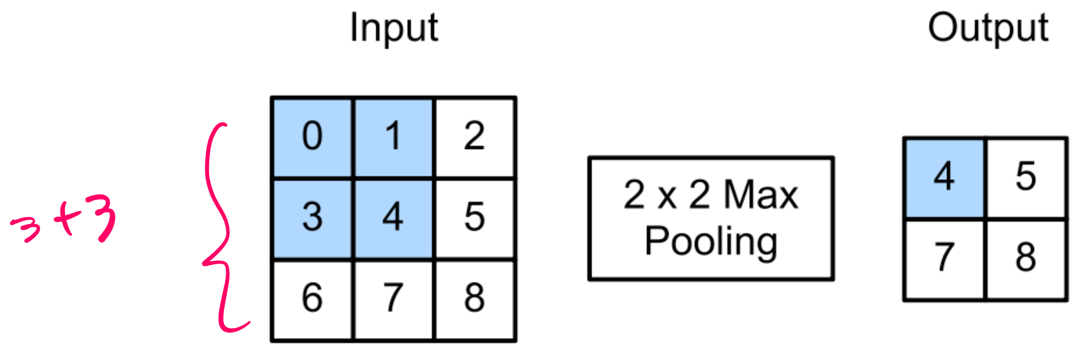


Figure 14.12: Illustration of maxpooling with a 2x2 filter and a stride of 1. Adapted from Figure 6.5.1 of [Zha+20].

Figure from PML

Finishing things up...

Typical architecture for CNNs:

Input → [[Conv → ReLU]*N → Pool?]*M → [FC → ReLU]*Q → FC

Common choices: $N \leq 5$, $Q \leq 2$, M is large

parameters here is very large

How do we learn the filters/weights?

Essentially the same as fully connected NNs: apply SGD/backpropagation

Demo time

What is a Convolutional Neural Network?

https://poloclub.github.io/cnn-explainer/

A breakthrough result

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

A breakthrough result

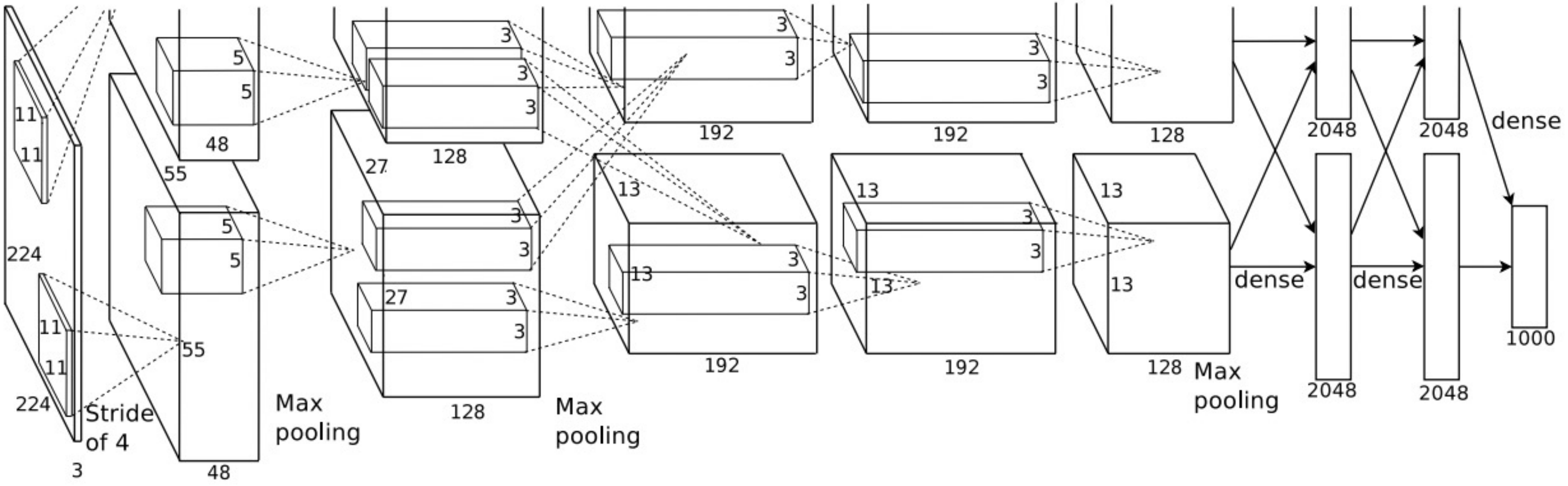
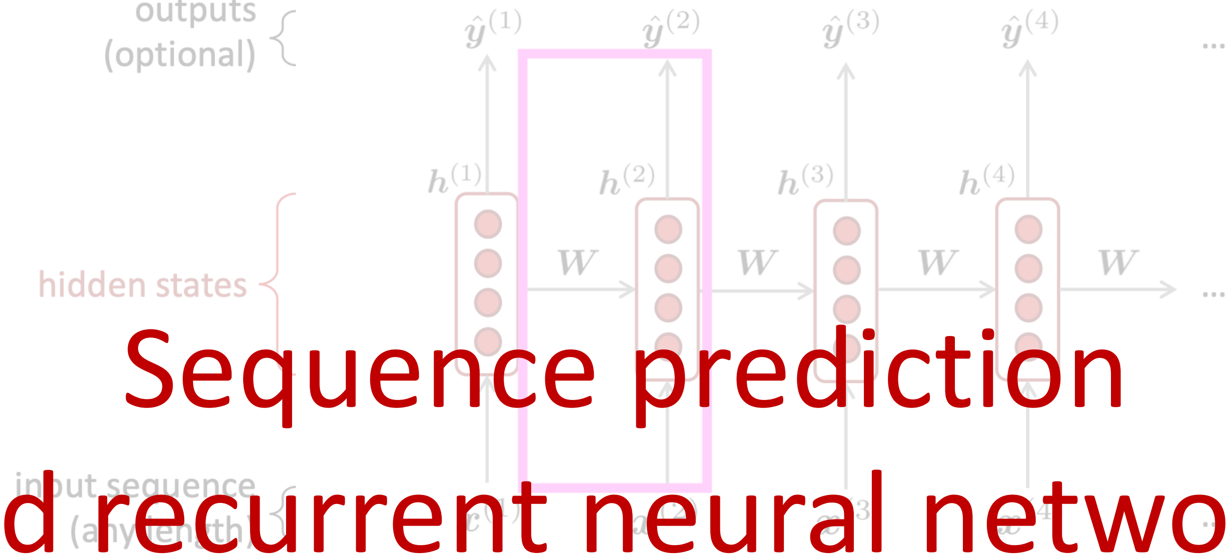


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure from Krizhevsky et al.'12



Sequence prediction and recurrent neural networks

Acknowledgements

A bit more math, and fewer cat pictures now ☹

We borrow heavily from:

- Stanford's CS224n: https://web.stanford.edu/class/cs224n/

Sequential prediction

Given observations $\overbrace{x_1, x_2, \dots, x_{t-1}}^{\text{input}}$ what is $\overbrace{x_t}^{\text{output}}$?

the input can be of different lengths

Examples:

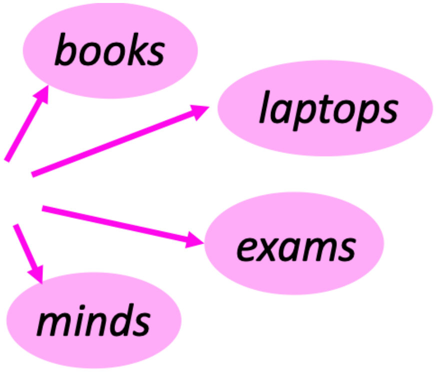
- text or speech data
- stock market data
- weather data
- …

In this lecture, we will mostly focus on text data (**language modelling**).

Language modelling

Language modelling is the task of predicting what word comes next:

the students opened their ______



r.v. over the randomness in the sentence, context etc.

More formally, let X_i be the random variable for the i-th word in the sentence, and let x_i be the value taken by the random variable. Then the goal is to compute

$$P(X_{t+1}|X_t = x_t, \ldots, X_1 = x_1).$$

A system that does this is known as a **Language Model.**

Slide adapted from CS224n by Chris Manning (Lecture 5)

Language modelling

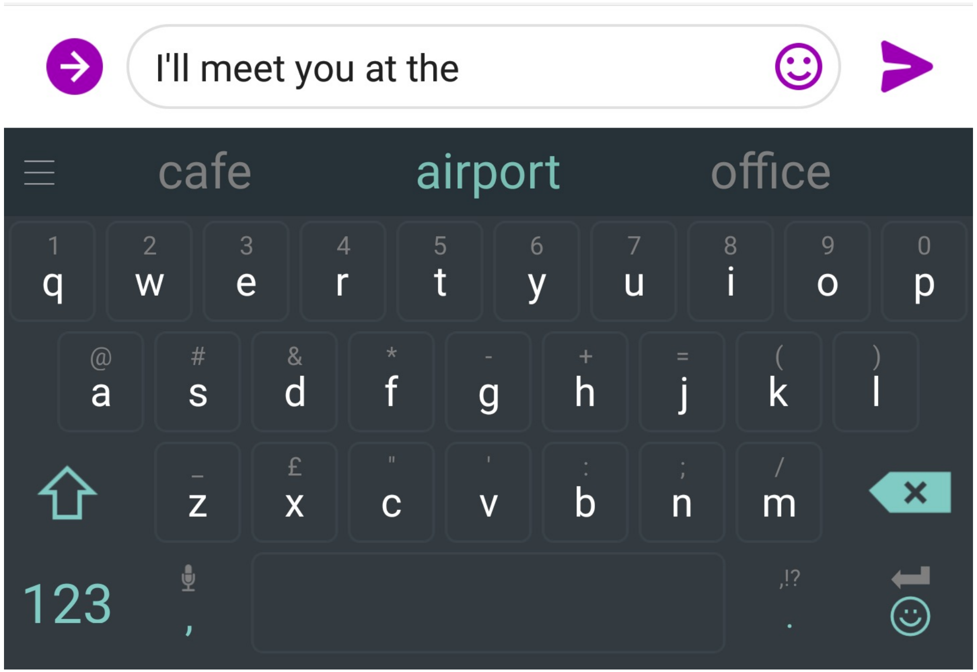
We can also think of a Language Model as a system that *assigns a probability to a piece of text.*

For example, if we have some text $x_1, \ldots, x_T$, then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(X_1 = x_1, \ldots, X_T = x_T) &= P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \\ &\times \cdots \times P(X_T = x_T | X_{T-1} = x_{T-1}, \ldots, X_1 = x_1) \\ &= \Pi_{t=1}^{T} P(X_t = x_t | X_{t-1} = x_{t-1}, \ldots, X_1 = x_1). \end{aligned}$$

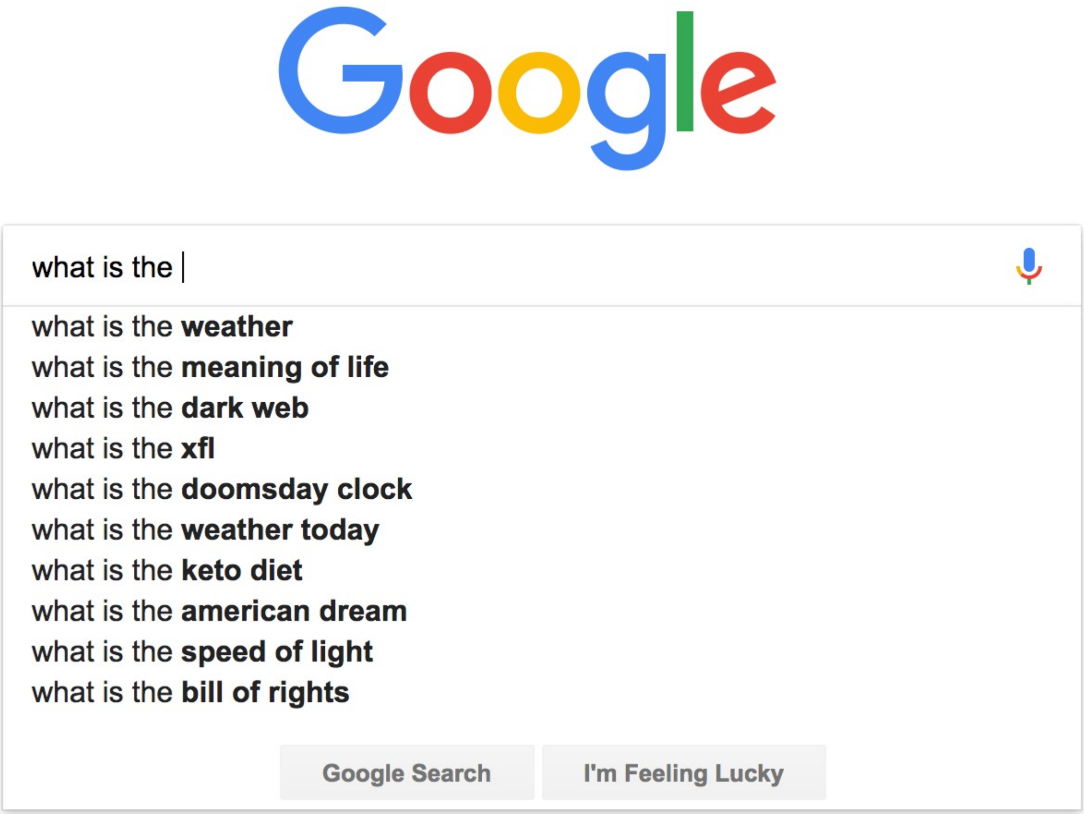
Slide adapted from CS224n by Chris Manning (Lecture 5)

You use Language Models every day!



Slide adapted from CS224n by Chris Manning (Lecture 5)

You use Language Models every day!



Slide adapted from CS224n by Chris Manning (Lecture 5)

n-gram Language Models

the students opened their ______

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!

- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - four-grams: “the students opened their”

- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

Slide adapted from CS224n by Chris Manning (Lecture 5)

n-gram language model: A type of Markov model

A **Markov model or Markov chain** is a sequence of random variables with the **Markov property**: a sequence of random variables $X_1, X_2, \cdots$ s.t.

$$P(X_{t+1} \mid X_{1:t}) = P(X_{t+1} \mid X_t) \qquad \text{(Markov property)}$$

i.e. *the current state only depends on the most recent state* (notation $X_{1:t}$ denotes the sequence $X_1, \ldots, X_t$). This is a *bigram model*.

We will consider the following setting:

- All X_t's take value from the same discrete set $\{1, \ldots, S\}$ the size of the dictionary of all possible words
- $P(X_{t+1} = s' \mid X_t = s) = a_{s,s'}$, known as transition probability
- $P(X_1 = s) = \pi_s \rightarrow$ initial probability
- $(\{\pi_s\}, \{a_{s,s'}\}) = (\boldsymbol{\pi}, \boldsymbol{A})$ are parameters of the model. ($\boldsymbol{A} \in \mathbb{R}^{S \times S}$ is the matrix where the entry corresponding to s, s' is $a_{s,s'}$.)

$P(X_1, \ldots, X_T) = P(X_1) \cdot P(X_2 \mid X_1) \cdot P(X_3 \mid X_2) \cdots P(X_T \mid X_{T-1})$

Markov model: examples

- Example 1 (**Language model**)
  States $[S]$ represent a dictionary of words,

$$a_{\text{ice,cream}} = P(X_{t+1} = \text{cream} \mid X_t = \text{ice})$$

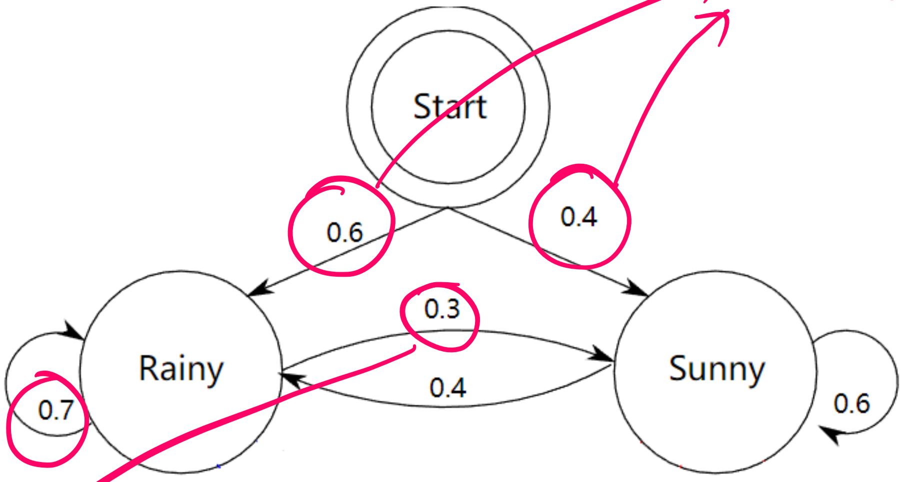
  is an example of the transition probability.

- Example 2 (**Weather**)
  States $[S]$ represent weather at each day

$$a_{\text{sunny,rainy}} = P(X_{t+1} = \text{rainy} \mid X_t = \text{sunny})$$

Markov model: Graphical representation

A Markov model is nicely represented as a **directed graph**



1st day is { Rainy, 60% prob.
Sunny, 40% prob.

$\pi_{Rainy} = 0.6$

$\pi_{Sunny} = 0.4$

if today is Rainy, tomorrow will be { Rainy, 70% prob.
Sunny, 30% prob

Learning Markov models

Now suppose we have observed n **sequences of examples**:

(rainy, sunny , ..., rainy)

(sunny, sunny, ... sunny)

- $x_{1,1}, \ldots, x_{1,T}$
- $\cdots$
- $x_{i,1}, \ldots, x_{i,T}$
- $\cdots$
- $x_{n,1}, \ldots, x_{n,T}$

where

- for simplicity we assume each sequence has the same length T
- lower case $x_{i,t}$ represents the value of the random variable $X_{i,t}$

From these observations how do we *learn the model parameters* $(\boldsymbol{\pi}, \boldsymbol{A})$?

Learning Markov models: MLE

Same story, find the **MLE**. The log-likelihood of a sequence $x_1, \ldots, x_T$ is

$$\ln P(X_{1:T} = x_{1:T})$$

$$= \sum_{t=1}^{T} \ln P(X_t = x_t \mid X_{1:t-1} = x_{1:t-1}) \qquad \text{(always true)}$$

$$= \sum_{t=1}^{T} \ln P(X_t = x_t \mid X_{t-1} = x_{t-1}) \qquad \text{(Markov property)}$$

$$= \ln \pi_{x_1} + \sum_{t=2}^{T} \ln a_{x_{t-1}, x_t}$$

$P(X_1 = x_1) = \pi x_1$

Prob. of transitioning from $x_{t-1} \to x_t$

$$= \sum_{s} \mathbb{I}[x_1 = s] \ln \pi_s + \sum_{s,s'} \left(\sum_{t=2}^{T} \mathbb{I}[x_{t-1} = s, x_t = s'] \right) \ln a_{s,s'}$$

This is over one sequence, can sum over all.

Learning Markov models: MLE

So MLE is

if this is large for some s $\Rightarrow$ this should be large for that s

$$\underset{\boldsymbol{\pi}, \boldsymbol{A}}{\operatorname{argmax}} \sum_s (\textbf{\#initial states with value } s) \ln \pi_s + \sum_{s,s'} (\textbf{\#transitions from } s \textbf{ to } s') \ln a_{s,s'}$$

This is an optimization problem, and can be solved by hand (though we'll skip in class).
The solution is:

$$\pi_s = \frac{\text{\#initial states with value } s}{\text{\#initial states}}$$

$$a_{s,s'} = \frac{\text{\#transitions from } s \text{ to } s'}{\text{\#transitions from } s \text{ to any state}}$$

Learning Markov models: Another perspective

Let's first look at the transition probabilities. By the Markov assumption,

$$P(X_{t+1} = x_{t+1} \mid X_t = x_t, \ldots, X_1 = x_1) = P(X_{t+1} = x_{t+1} \mid X_t = x_t)$$

Using the definition of conditional probability,

$$P(X_{t+1} = x_{t+1} \mid X_t = x_t) = \frac{P(X_{t+1} = x_{t+1}, X_t = x_t)}{P(X_t = x_t)}$$

We can estimate this using data,

$$\frac{P(X_{t+1} = x_{t+1}, X_t = x_t)}{P(X_t = x_t)} \approx \frac{\text{\#times } (x_t, x_{t+1}) \text{ appears}}{\text{\# times } (x_t) \text{ appears (and is not the last state)}}$$

The initial state distribution follows similarly,

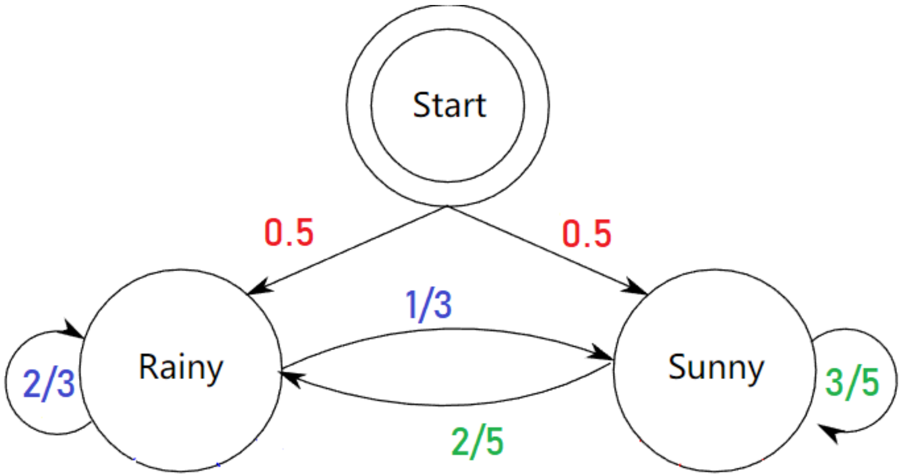
$$P(X_1 = s) \approx \frac{\text{\#times } s \text{ is first state}}{\text{\#sequences}}$$

Just like estimating bias of a coin / dice.

Learning Markov models: Example

Suppose we observed the following 2 sequences of length 5

- sunny, sunny, rainy, rainy, rainy
- rainy, sunny, sunny, sunny, rainy



Higher-order Markov models

Is the Markov assumption reasonable? Not so in many cases, such as for language modeling.

Higher order Markov chains make it a bit more reasonable, e.g.

$$P(X_{t+1} \mid X_t, \dots, X_1) = P(X_{t+1} \mid X_t, X_{t-1}) \qquad \textbf{(second-order Markov assumption)}$$

i.e. the current word only depends on the last two words. This is a *trigram model*, since we need statistics of three words at a time to learn. In general, we can consider a n-th Markov model (or a *$(n+1)$-gram model*):

$$P(X_{t+1} \mid X_t, \dots, X_1) = P(X_{t+1} \mid \overbrace{X_t, X_{t-1}, \dots, X_{t-n+1}}^{\text{previous } n \text{ observations}}) \quad \textbf{(}n\textbf{-th order Markov assumption)}$$

Learning higher order Markov chains is similar, but more expensive.

$$\begin{aligned}
P(X_{t+1} = x_{t+1} \mid X_t = x_t, \dots, X_1 = x_1) &= P(X_{t+1} = x_{t+1} \mid X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_{t-n+1} = x_{t-n+1}) \\
&= \frac{P(X_{t+1} = x_{t+1}, X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_{t-n+1} = x_{t-n+1})}{P(X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_{t-n+1} = x_{t-n+1})} \\
&\approx \frac{\text{count}(x_{t-n+1}, \dots, x_{t-1}, x_t, x_{t+1}) \text{ in the data}}{\text{count}(x_{t-n+1}, \dots, x_{t-1}, x_t) \text{ in the data}}
\end{aligned}$$

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

$$\cancel{\textit{as the proctor started the clock, the}}_{\text{discard}} \; \underbrace{\textit{students opened their}}_{\text{condition on this}} \_\_\_\_\_\_$$

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \boldsymbol{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

Slide adapted from CS224n by Chris Manning (Lecture 5)

n-gram Language Models in practice

- You can build a simple trigram Language Model over a
  1.7 million word corpus (Reuters) in a few seconds on your laptop

Business and financial news

today the _______

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the ________

condition on this

get probability distribution

```
company  0.153
bank     0.153
price    0.077
italian  0.039
emirate  0.039
         …
```

sample

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price ________

condition on this

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
…	

sample

Generating text with a n-gram Language Model

You can also use a Language Model to generate text

today the price of ________

condition on this

get probability distribution

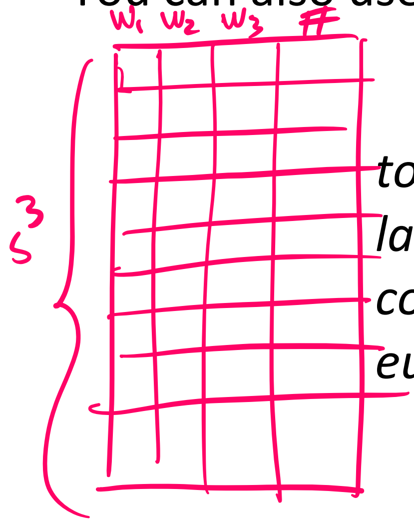
the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
…	

sample

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with a n-gram Language Model

You can also use a Language Model to generate text



today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

Surprisingly grammatical!

…but **incoherent.** We need to consider more than three words at a time if we want to model language well.

However, larger n increases model size and requires too much data to learn

Slide adapted from CS224n by Chris Manning (Lecture 5)

How to build a *neural* Language Model?

Changing notation, $x^{(1)}$ is overloaded to refer to both r.v. & its value

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob dist of the next word $P(\boldsymbol{x}^{(t+1)} | \ \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

- How about a window-based neural model?

Slide adapted from CS224n by Chris Manning (Lecture 5)

A fixed-window neural Language Model

~~*as the proctor started the clock*~~ *the students opened their* ______

discard

$\underbrace{\textit{the students opened their}}_{\text{fixed window}}$

Use a fixed window of previous words, and train a vanilla fully-connected neural network to predict the next word? → This is a standard supervised learning task

Neural networks take vectors as inputs, how to give a word as input?

Approach 1: one-hot (sparse) encoding

Suppose vocabulary is of size S

'the' = [1, 0, ... 0] → S dim. vector

'students' = [0, 1, ... 0] → S dim. vector

1 high dimensional

2 each representation is orthogonal, even similar words have representations which are far away.

Approach 2: **word embeddings/word vectors**

Word embeddings/vectors

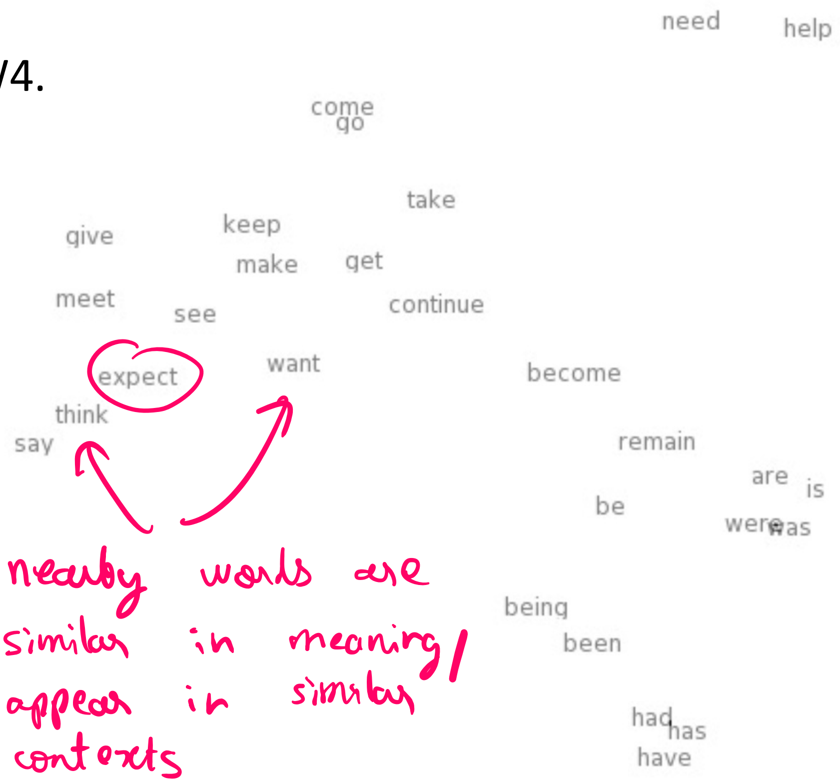
A word embedding is a (dense) mapping from words, to vector representations of the words.

Ideally, this mapping has the property that words similar in meaning have representations which are close to each other in the vector space.

You'll see a simple way to construct these in HW4.

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

10 dim. embedding



nearby words are similar in meaning / appear in similar contexts

Slide adapted from CS224n by Chris Manning (Lecture 1)

A fixed-window neural Language Model

Same architecture as network in HW3

output distribution

$$\hat{\boldsymbol{y}} = \underbrace{\mathrm{softmax}}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

using softmax to get distribution

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

f: non-linearity (ReLU)

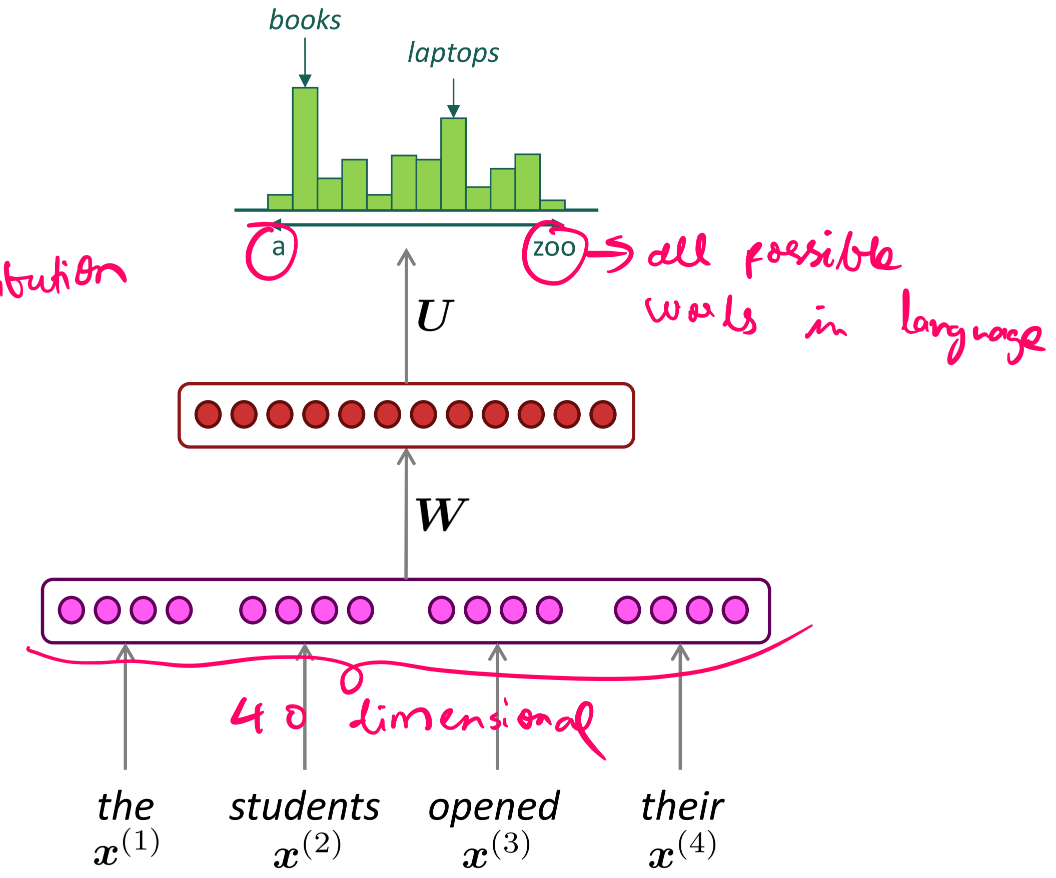
concatenated word embeddings

$$\boldsymbol{e} = [\underbrace{\boldsymbol{e}^{(1)}}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

suppose each is 10-dimensional

words / one-hot vectors

$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$



Slide adapted from CS224n by Chris Manning (Lecture 5)

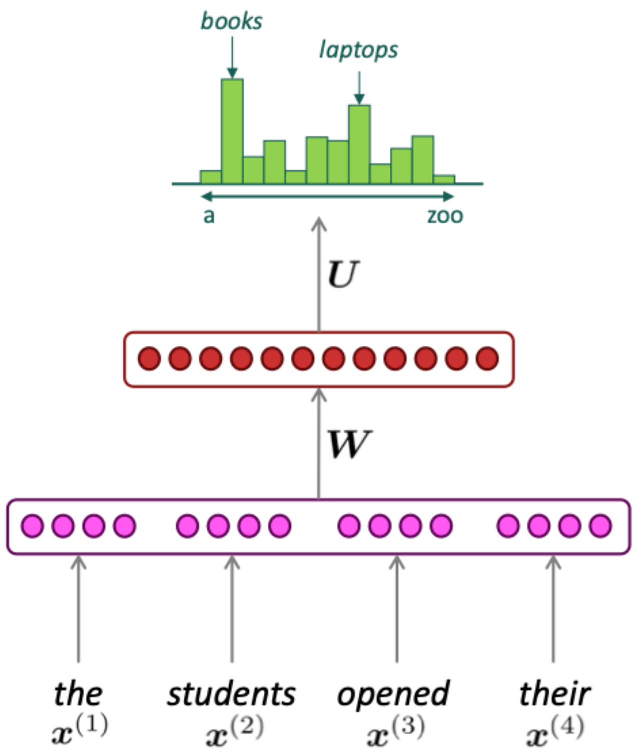
The problem with this architecture

- Uses a fixed window, which can be too small.
- Enlarging this window will enlarge the size of the weight matrix $\boldsymbol{W}$.
- The inputs $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ are multiplied by completely different weights in $\boldsymbol{W}$.

  No symmetry in how inputs are processed!

As with CNNs for images before, we need an architecture which has similar symmetries as the data.

In this case, ***can we have an architecture that can process any input length?***



Slide adapted from CS224n by Chris Manning (Lecture 5)

Recurrent Neural Networks (RNN)

A family of neural architectures

It’s okay if you don’t *fully* understand the next few slides on RNNs, but you should get the main ideas...

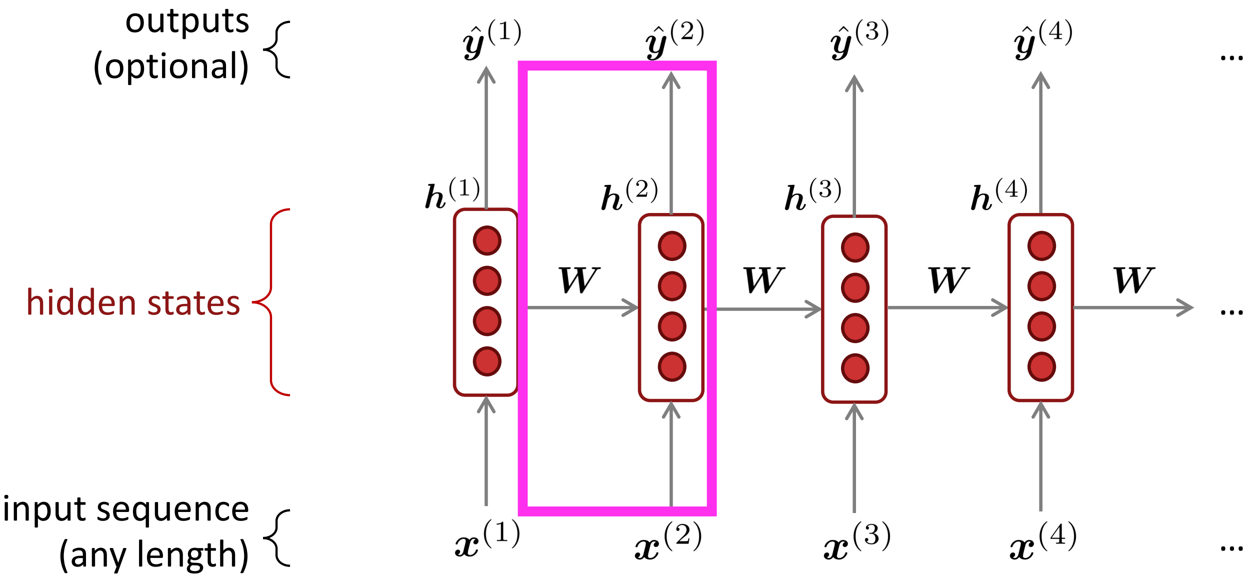
Slide adapted from CS224n by Chris Manning (Lecture 5)

Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights W *repeatedly*

similar to what we did with filters in CNNs



Slide adapted from CS224n by Chris Manning (Lecture 5)

A Simple RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\underbrace{\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2}_{\text{linear + softmax}}\right) \in \mathbb{R}^{|V|}$$

bias

hidden states

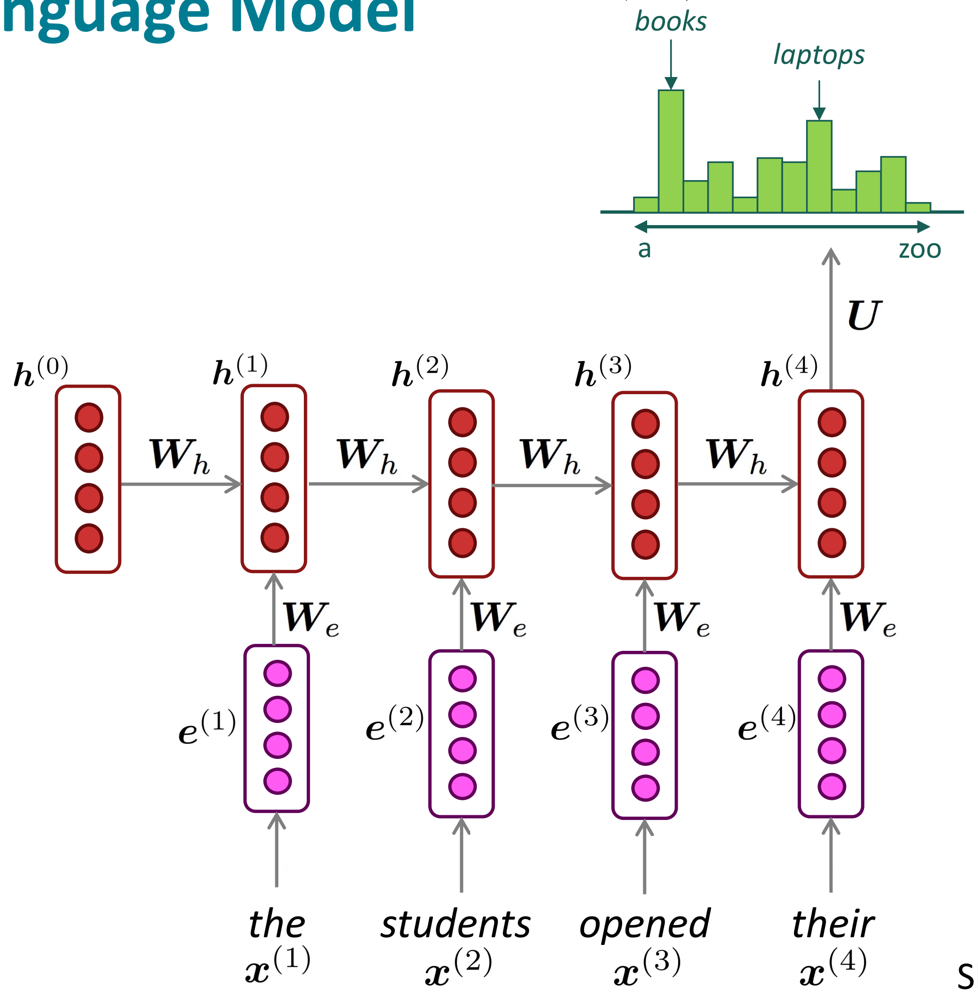
$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

σ : Activation (ReLu

word embeddings

$\boldsymbol{e}^{(t)}$ for word $\boldsymbol{x}^{(t)}$



***Note**: this input sequence could be much longer now!*

Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ *for every step t.*
  - i.e. predict probability dist of *every word*, given words so far

- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

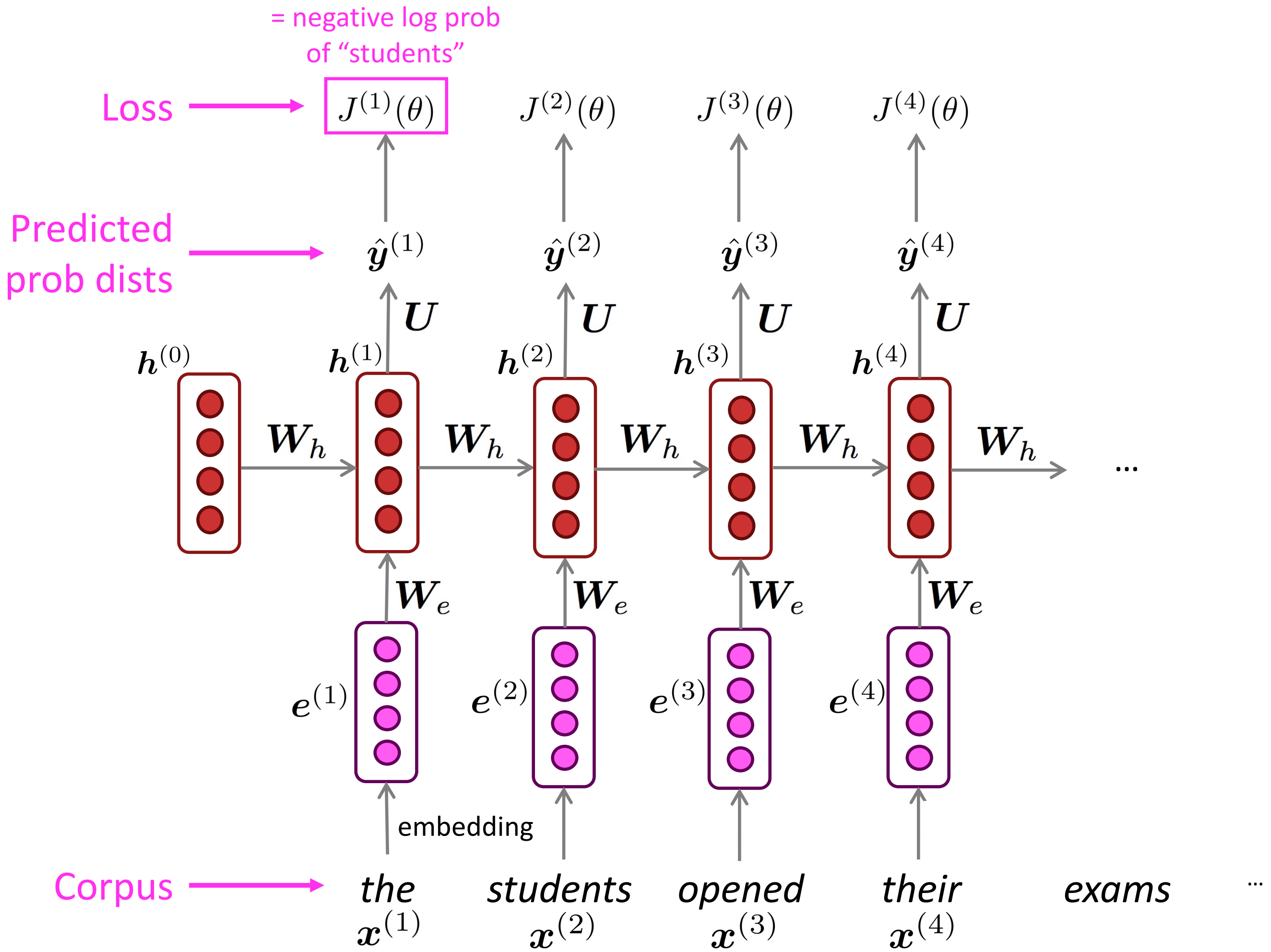
multi-class classification

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

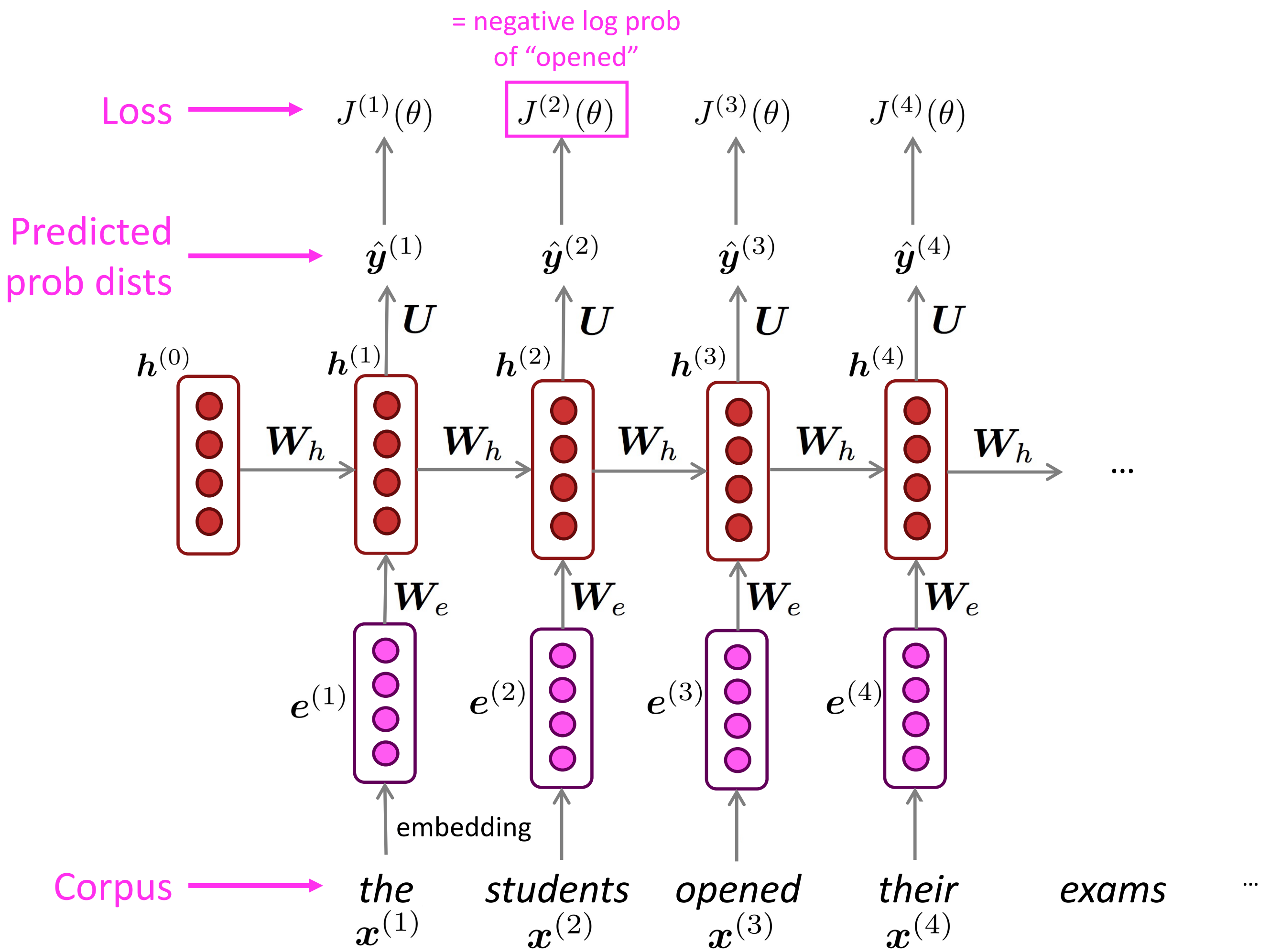
Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model



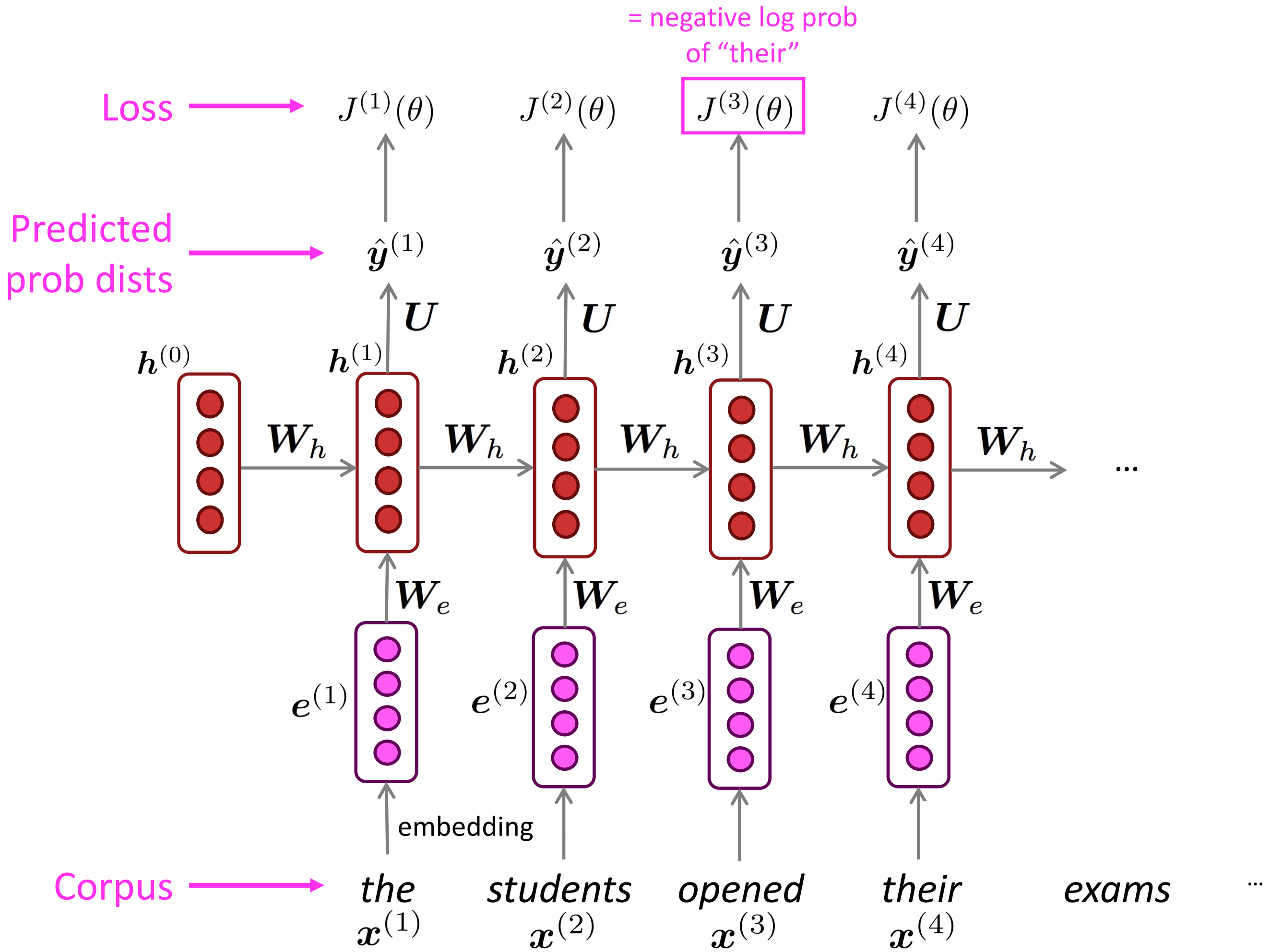
Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model



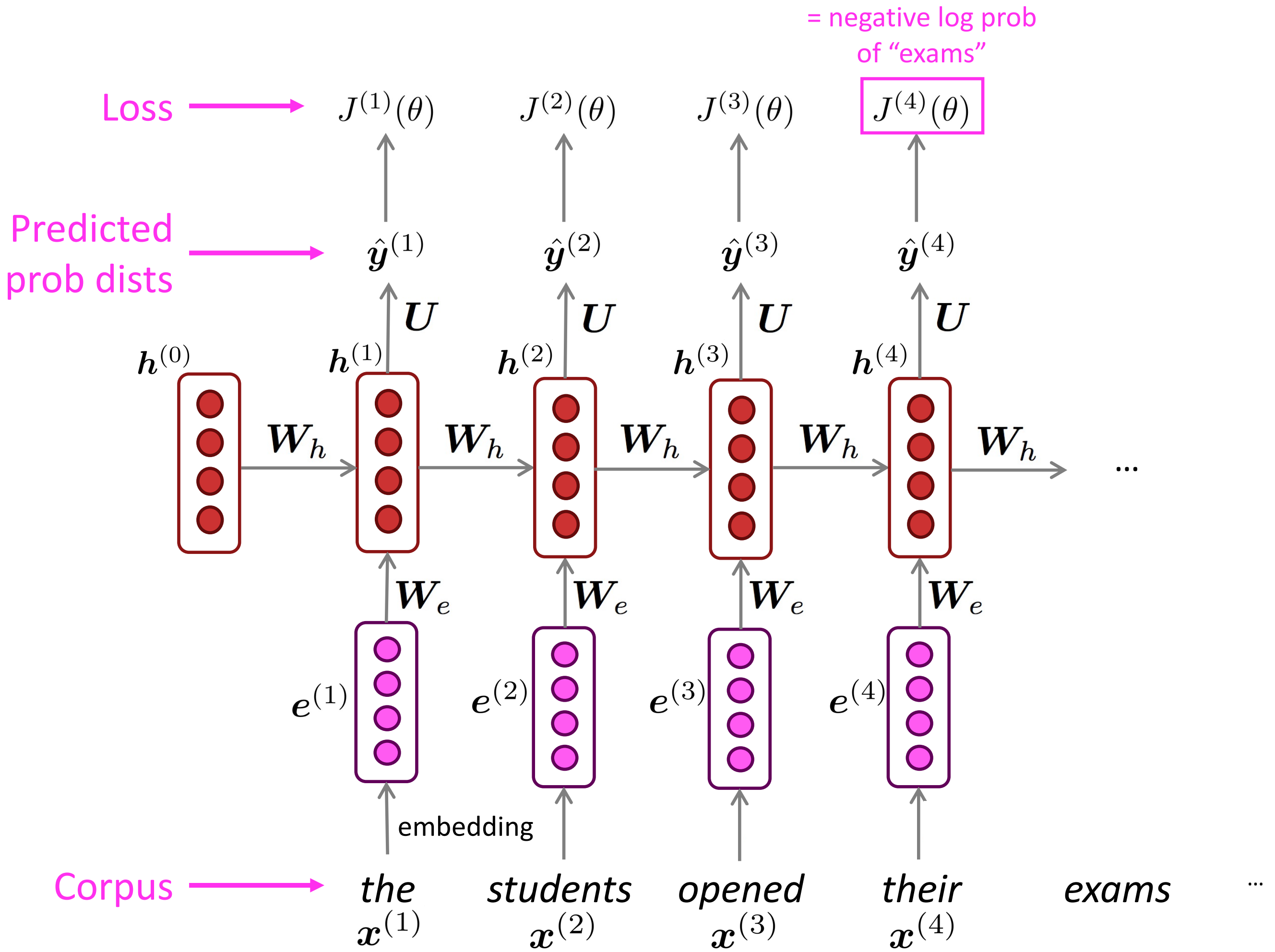
Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model



Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model

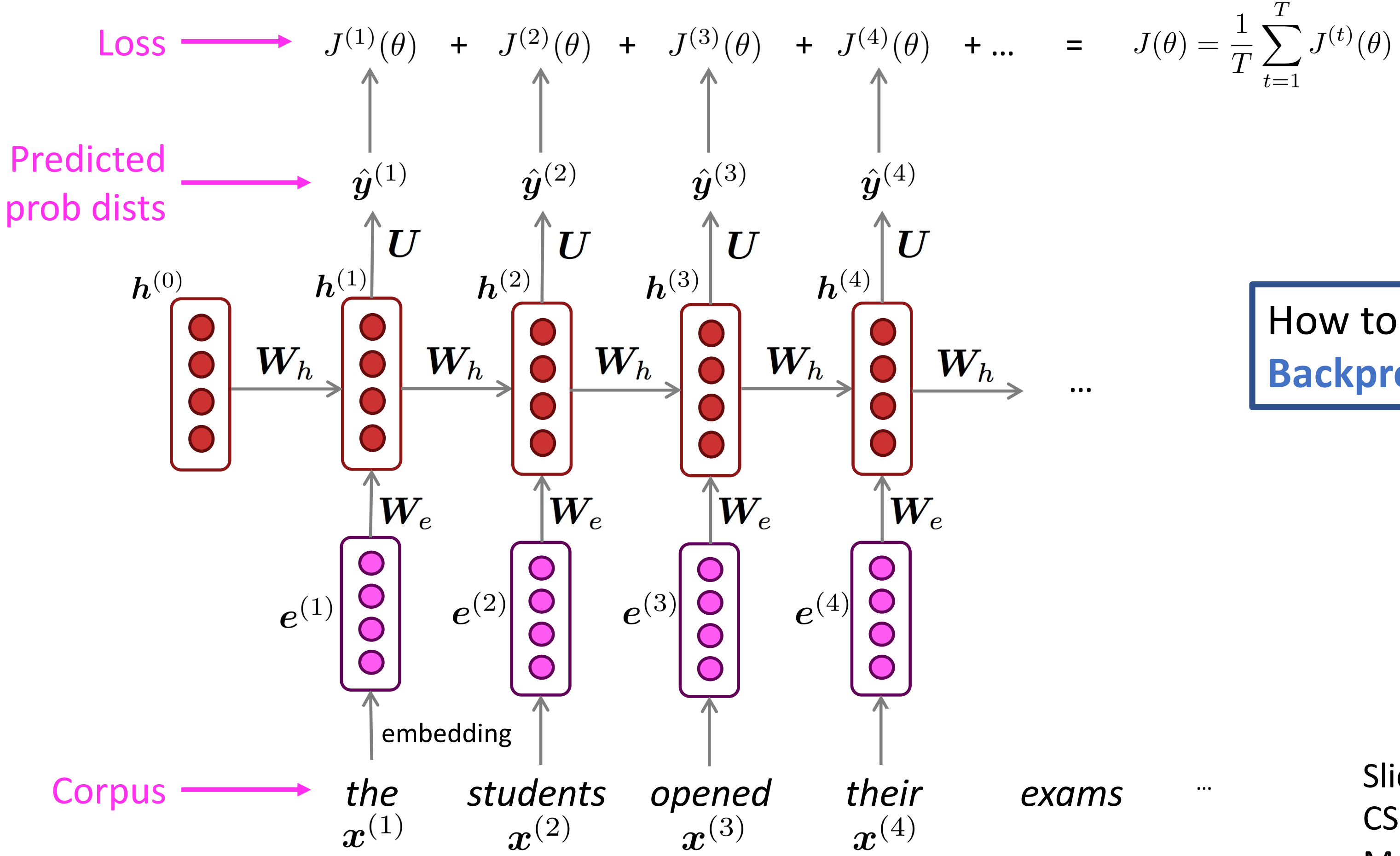


Slide adapted from CS224n by Chris Manning (Lecture 5)

Training an RNN Language Model

feed true output as input for next step

"Teacher forcing"

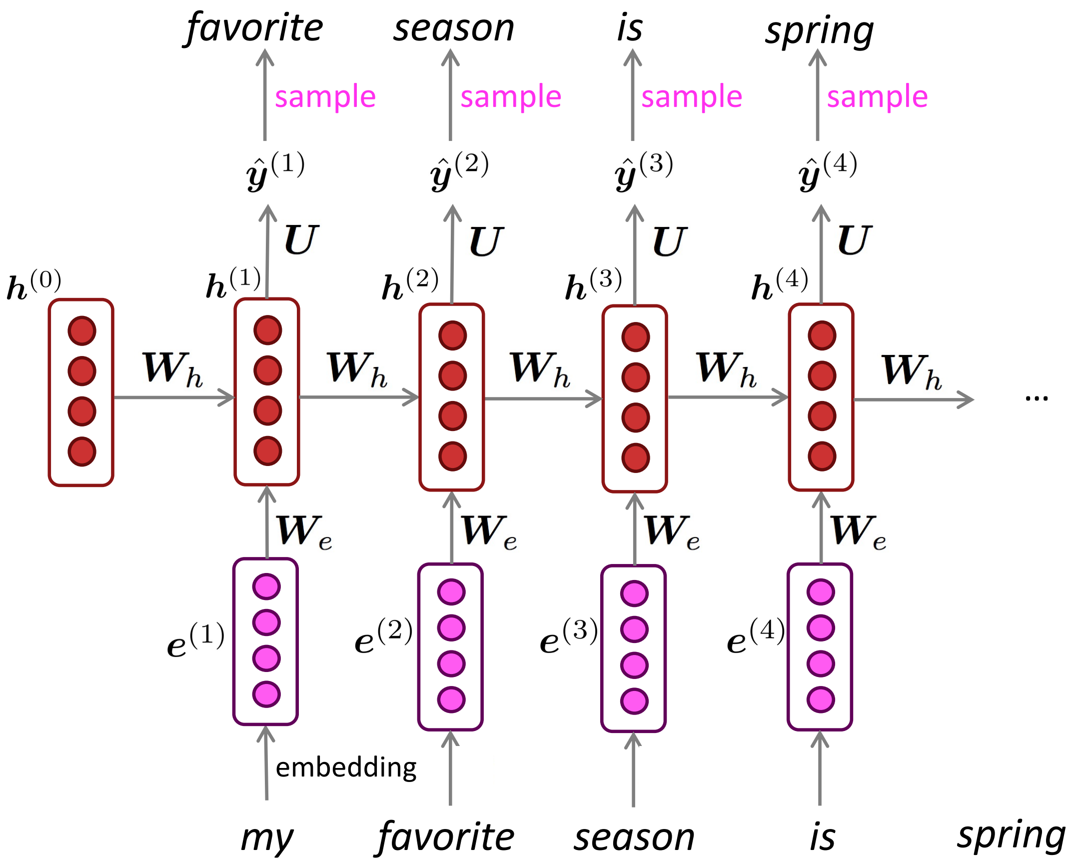


How to train this?
Backprop + SGD

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with a RNN Language Model

Just like a n-gram Language Model, you can use a RNN Language Model to generate text by repeated sampling. Sampled output becomes next step's input.



Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with an RNN Language Model

Let's have some fun!

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Slide adapted from CS224n by Chris Manning (Lecture 5)

Generating text with an RNN Language Model

Let's have some fun!

- You can train an RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Slide adapted from CS224n by Chris Manning (Lecture 5)

More recent models improve drastically on RNNs.
A particularly important model: The Transformer

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[* †]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[* ‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Why should we care about Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Predictive typing
  - Speech recognition
  - Handwriting recognition
  - Spelling/grammar correction
  - Authorship identification
  - Machine translation
  - Summarization
  - Dialogue
  - etc.
- Language Modeling has been extended to cover everything else in NLP

Slide adapted from CS224n by Chris Manning (Lecture 5)

Why should we care about Language Modeling?

- Language M
  measure ou
- Language M
  generating t

Text generation: GPT-3

The New York Times

Account

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.

I gave the following prompt to a much simpler model:
Today 250 students at the University of Southern California begin their machine learning class. What all will they learn??

Today 250 students at the University of Southern California begin their machine learning class. What all will they learn??

Machine learning is a fascinating subject that has been at the forefront of computer science for quite some time now. We're also seeing some new interest from those within the field. Here is one of our students here who has been working on machine learning for a number of years.

This is Dr. William Smith. He works on machine learning at the University of Southern California. He is a professor in computer science and the founder of the University of Southern California. It's a fantastic position to be part of the research community.

At the University of Southern California, you have a wide range of research interests. So, what's the most interesting research area to you?

GPT-2-simple response using https://watt-ai.github.io/

- Language M

Remember GPT-3 from Lecture 1?