

Lecture 7

Lecture 7

Convolutional Neural Networks (Convent / CNN)

- Intro
- Challenge
- 2-D Convolution
- 3-D Convolution
- Convolution Layer
- Connection to fully-connected network
- Pooling Layer
- A breakthrough result

Sequence Prediction and Markov Models

- Sequential prediction & Language modeling
- n-gram language model
- Markov model
- Learning Markov models: MLE
- Learning Markov models: Another perspective
- Higher-order Markov models
- Generating text with a n-gram Language Model

Recurrent Neural Networks

- The problem with fixed-window
- Word embeddings / vectors
- A fixed-window neural language model
- RNN
- Training an RNN language model
- Summary

Convolutional Neural Networks (Convent / CNN)

Intro

Image classification: a core task in computer vision



This image by Nikita is
licensed under CC-BY 2.0

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

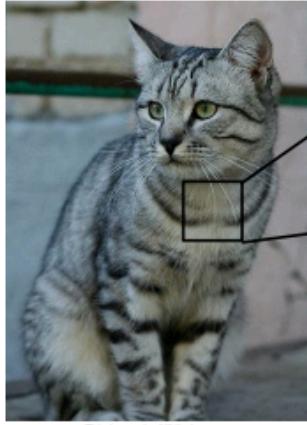
multiclass classification problem

cat

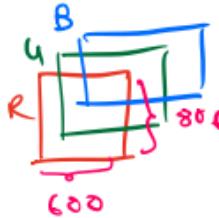
input : How is this represented ?

The problem: semantic gap

The Problem: Semantic Gap



This image by Nikita is licensed under CC-BY 2.0



[1] [105 112 100 131 144 99 106 98 96 103 112 119 184 97 93 83]
[99 98 102 106 126 184 79 98 103 99 105 123 116 118 105 94 95 83]
[76 93 98 105 126 105 87 96 95 99 115 120 102 107 93 95 83]
[99 81 93 123 131 127 100 95 99 102 98 99 93 101 94]
[106 93 61 64 93 91 88 99 85 101 107 109 75 80 94 96 93]
[104 98 100 105 107 102 100 95 99 101 107 109 75 80 94 96 93]
[133 137 147 183 85 85 88 85 52 54 74 74 84 102 93 85 82]
[128 137 144 146 105 95 70 70 62 65 63 63 72 86 101 93]
[125 133 148 137 151 121 117 94 65 79 68 65 54 64 72 96]
[127 123 131 147 137 123 126 131 113 98 89 75 61 64 72 94]
[124 128 132 131 137 131 123 126 113 98 89 75 61 64 72 94]
[89 93 98 102 106 126 131 131 113 114 119 109 106 95 77 83]
[81 97 106 108 126 131 131 113 114 119 109 106 95 77 83]
[83 77 86 91 77 79 102 122 117 113 117 125 123 138 115 87]
[63 65 82 89 76 78 102 124 120 119 118 101 107 101 131 139]
[83 65 75 78 88 76 72 62 83 128 139 135 185 81 98 100 118 106]
[85 65 75 78 88 76 72 62 83 128 139 135 185 81 98 100 118 106]
[156 97 82 106 157 123 116 64 41 95 95 93 89 99 105 187]
[184 149 112 116 82 128 124 124 184 76 48 45 86 99 101 182 189]
[157 174 158 123 93 98 116 132 112 97 60 55 78 70 62 94 94]
[139 124 134 161 159 170 158 116 123 121 130 114 85 53 69 70 94]
[132 127 137 147 147 147 147 147 147 147 147 147 147 147 147 147 147]
[123 107 96 66 83 113 153 149 122 109 184 75 98 107 112 99]
[121 122 182 86 82 88 94 57 145 145 153 182 78 98 79 102 187]
[122 164 148 183 75 78 78 83 93 101 109 139 102 61 69 84]

What the computer sees

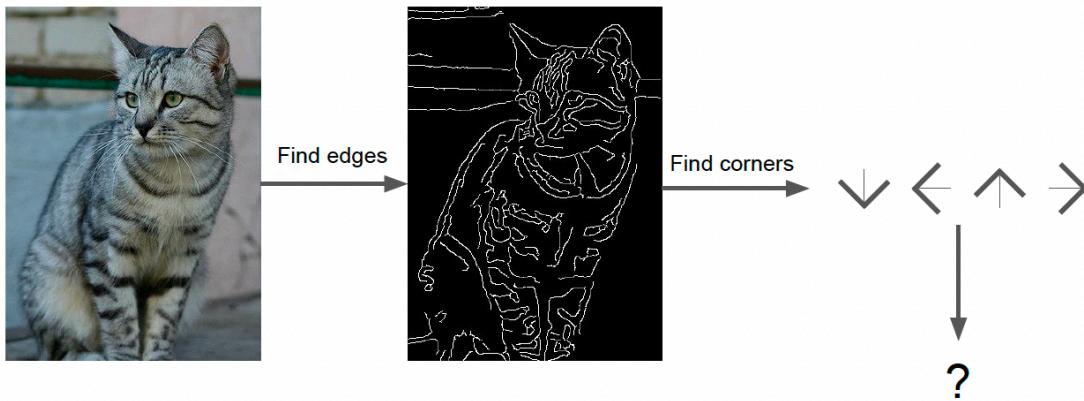
An image is just a big grid of numbers between [0, 255]:

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

Challenges: viewpoint variation, illumination, deformation, occlusion, background clutter, intraclass variation.

Unlike sorting a list of numbers, no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Attempts:



Data-Driven Approach

- Collect a dataset of images and labels
 - Use machine learning to train a classifier
 - Evaluate the classifier on new images

Challenge

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

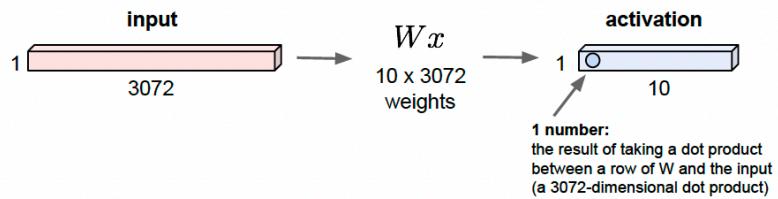
The ingredients:

- 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

$32 \times 32 \times 3$ image $\rightarrow 3072 \times 1$



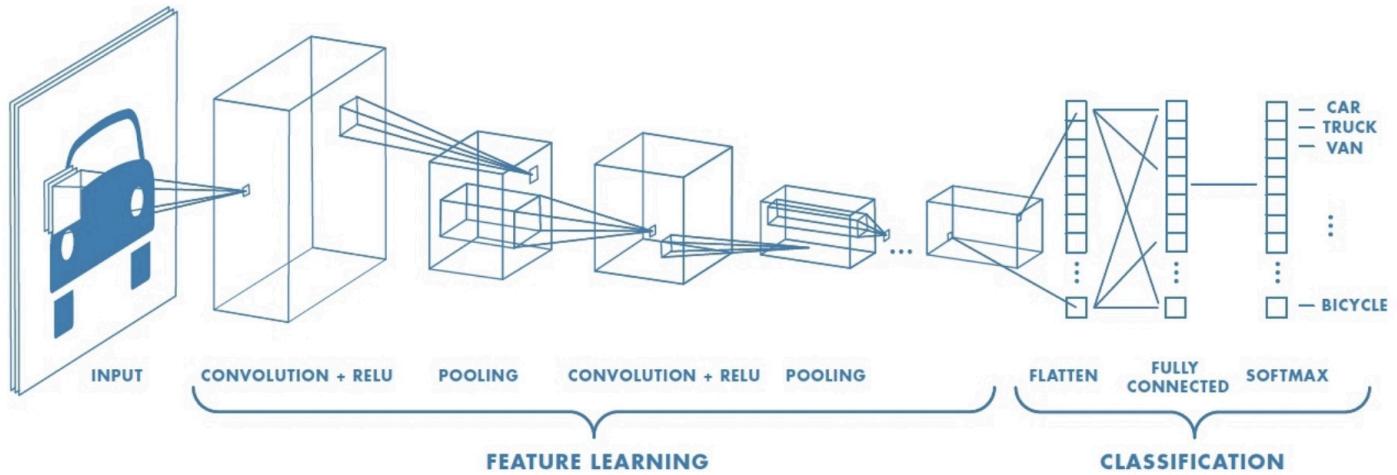
Completely loses out on spatial structure.

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.



2-D Convolution

* operation is convolution

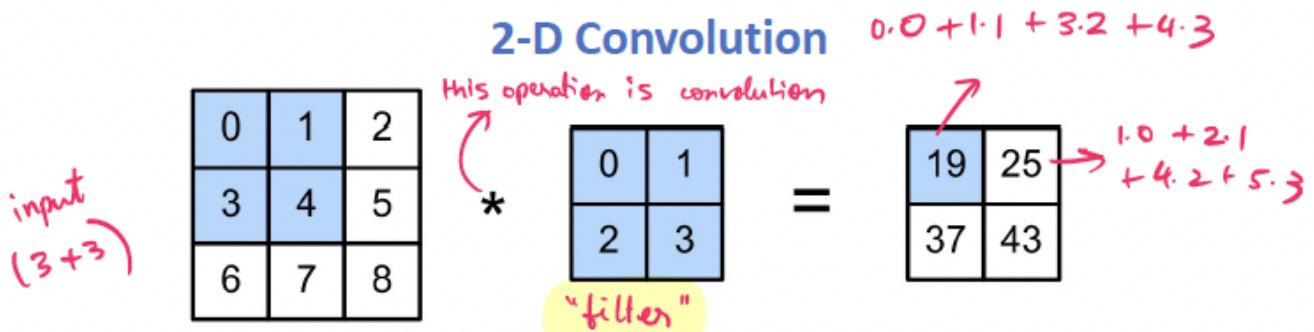
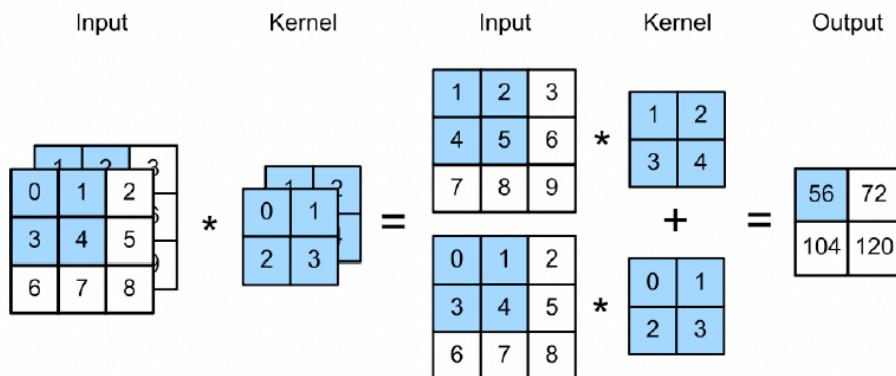


Figure 14.5: Illustration of 2d cross correlation. Generated by `conv2d_jax.ipynb`. Adapted from Figure 6.2.1 of [Zha+20].

3-D Convolution

The input is $3 \times 3 \times 2$

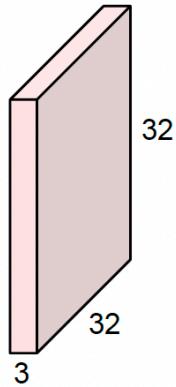


Add up the result for the two channels

Convolution Layer

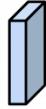
Convolution Layer

32x32x3 image



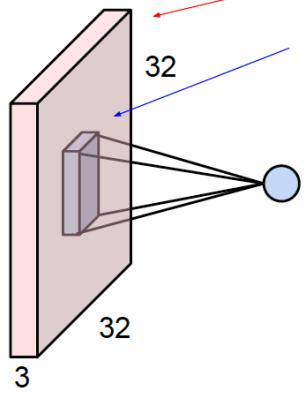
Filters always extend the full depth of the input volume

5x5x3 filter



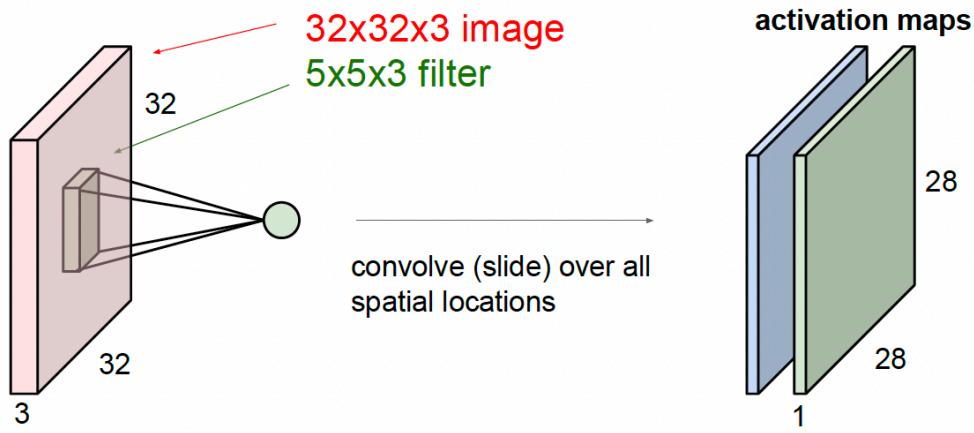
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

32x32x3 image
5x5x3 filter w

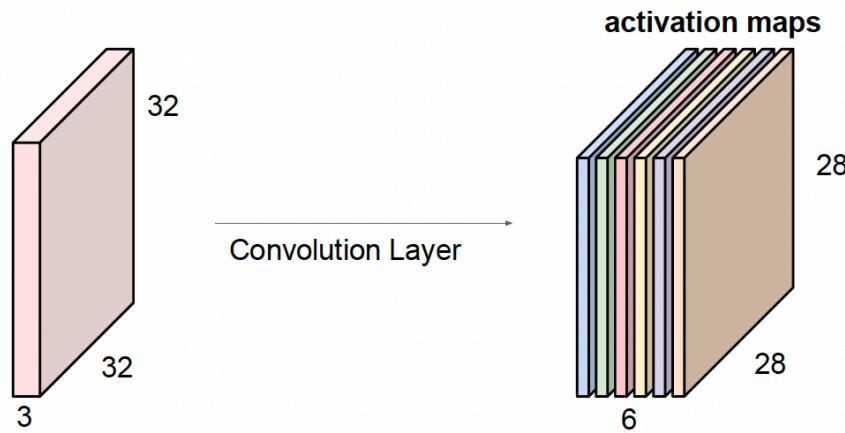


1 number:
the result of taking a dot product between the
filter and a small 5x5x3 chunk of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

$$w^T x + b$$

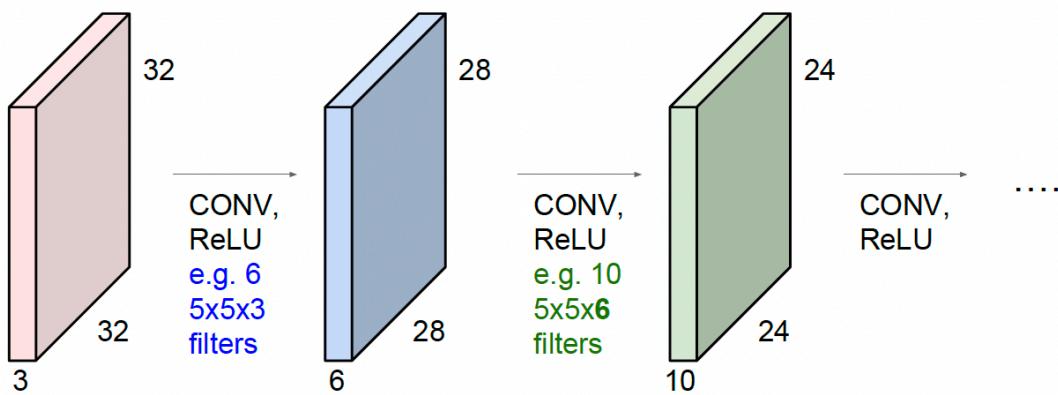


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

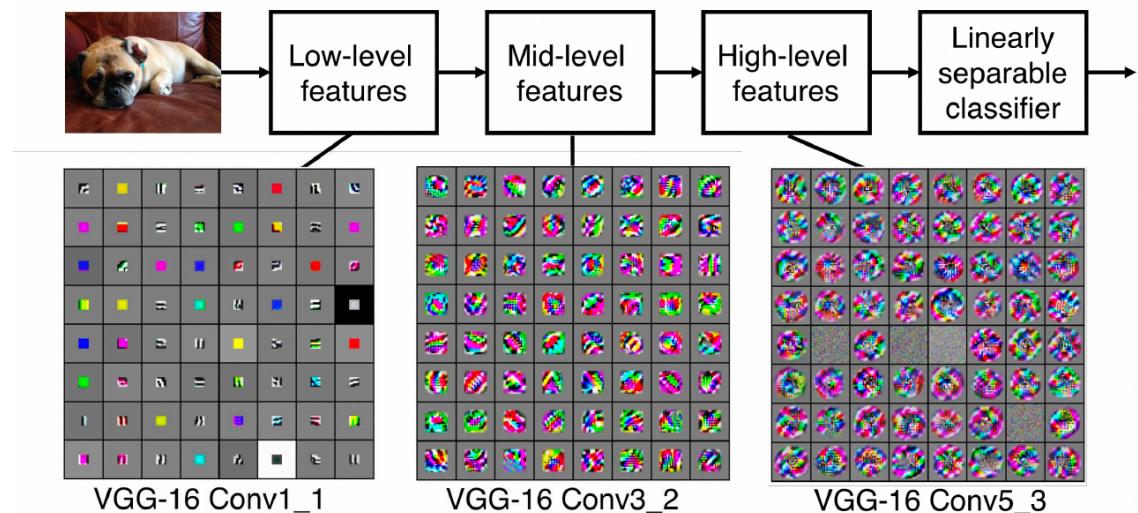
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



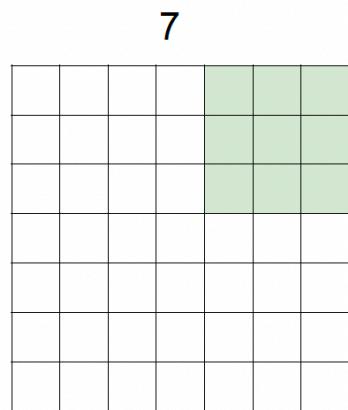
Preview

[Zeiler and Fergus 2013]

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

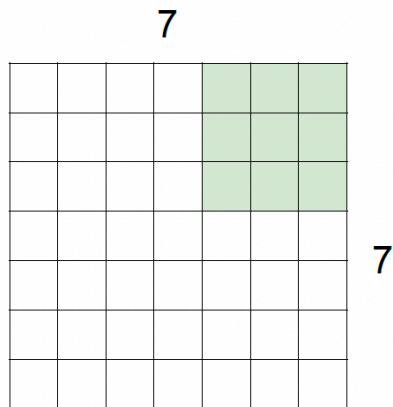


Calculation:

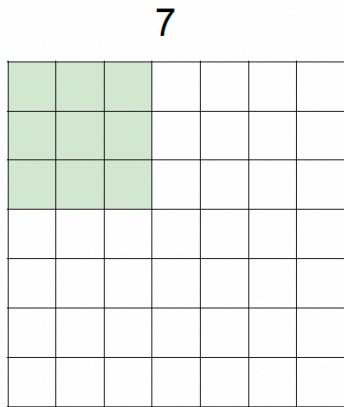


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

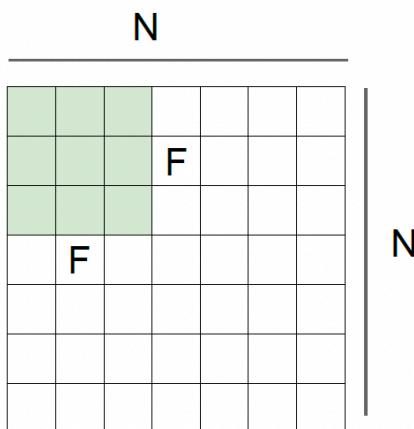


7x7 input (spatially)
assume 3x3 filter
applied with **stride 2**
=> 3x3 output!



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:
 $(N - F) / \text{stride} + 1$

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

In practice: Common to zero pad the border.

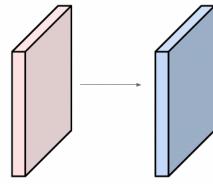
0	0	0	0	0	0		
0							
0							
0							
0							

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 $F \times F$, and zero-padding with
 $(F-1)/2$. (will preserve size spatially)
e.g. $F = 3 \Rightarrow$ zero pad with 1
 $F = 5 \Rightarrow$ zero pad with 2
 $F = 7 \Rightarrow$ zero pad with 3

$$\frac{N + 2P - F}{\text{stride}} + 1$$

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

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

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

$$W_2 = \frac{W_1 + 2P - F}{S} + 1$$

$$H_2 = \frac{H_1 + 2P - F}{S} + 1$$

$$D_2 = K$$

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

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

Exp: <https://poloclub.github.io/cnn-explainer/>

Connection to fully-connected network

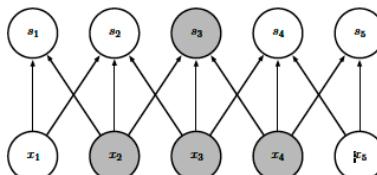
A convolutional layer is a special case of a fully connected layer:

Filter = weights with sparse connection and parameter sharing.

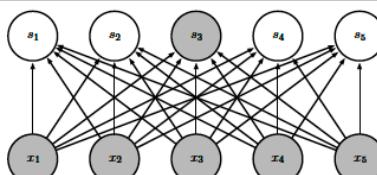
Local Receptive Field Leads to Sparse Connectivity (affects less),

Sparse connectivity: being affected by less

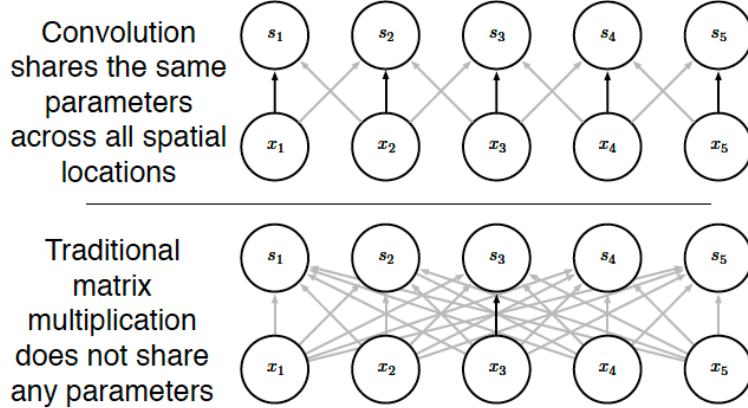
Sparse connections due to small convolution kernel



Dense connections



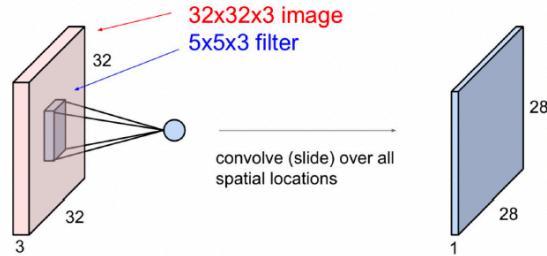
Parameter sharing



Much fewer parameters: (Ex ignoring bias terms)

FC layer: $(32 \times 32 \times 3) \times (28 \times 28) \approx 2.4M$

Conv layer: $5 \times 5 \times 3 = 75$



Pooling Layer

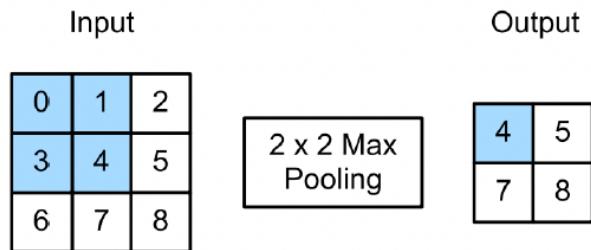
Makes the representations smaller and more manageable

Operates over each activation map independently.

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.



Shrink the feature map.

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

Hyperparameters:

- filters of size $F \times F$

- Stride S

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

$$W_2 = \frac{W_1 - F}{S} + 1$$

$$H_2 = \frac{H_1 - F}{S} + 1$$

$$D_2 = D_1$$

#parameters: 0 .

Input \rightarrow $[[Conv \rightarrow ReLU] * N \rightarrow Pool?] * M \rightarrow [FC \rightarrow ReLU] * Q \rightarrow FC$

Common choices: $N \leq 5, Q \leq 2, M$ is large. (# parameters here is really large)

How do we learn the filters/weights?

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

A breakthrough result

AlexNet

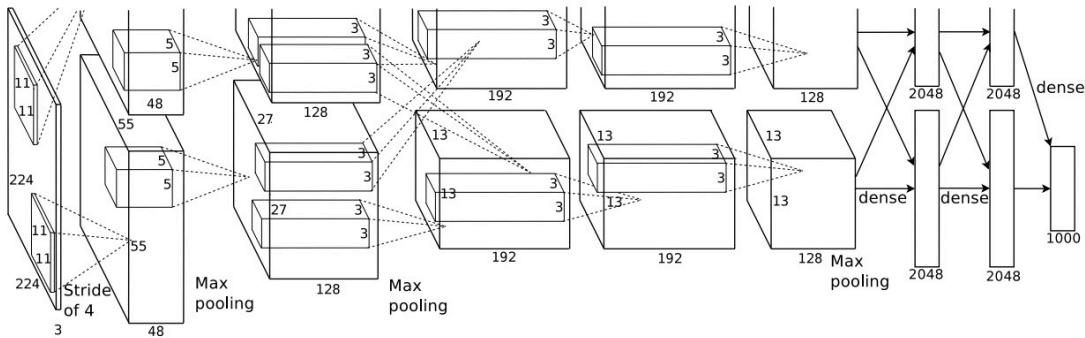


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input 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.

Sequence Prediction and Markov Models

Sequential prediction & Language modeling

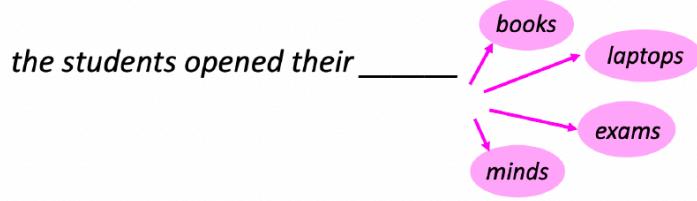
Given observations x_1, x_2, \dots, x_{t-1} (input), what is x_t (output).

Examples:

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

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

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



More formally, let X_i (r.v. over the randomness in the sentence, context, etc.) 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

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

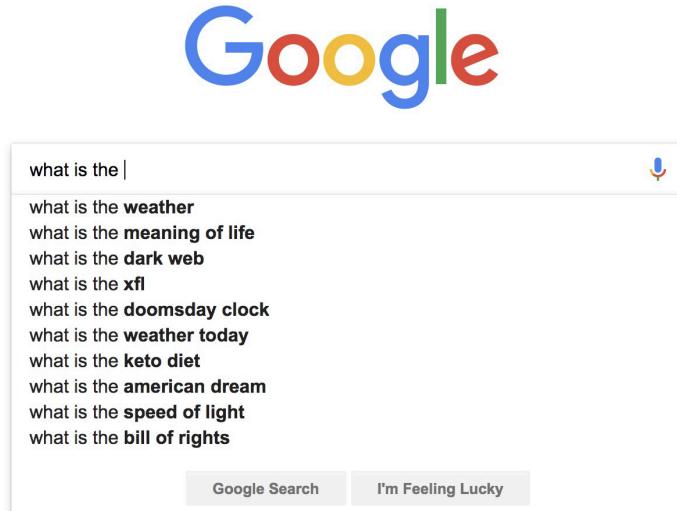
A system that does this is known as language model.

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, \dots, x_T , then the probability of this text (according to the Language Model) is:

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

You use Language Models every day!



n-gram language model

n-gram language model is a type of Markov model.

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.

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, \dots , s.t.

$$P(X_{t+1}|X_1 : t) = P(X_{t+1}|X_t)$$

i.e. the current state only depends on the most recent state (notation $X_{1:t}$ denotes the sequence X_1, \dots, X_t). This is a bigram(2-gram) model!

We will consider the following setting:

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

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

- Example 1 (Language model)

States $[S]$ represent a dictionary of words,

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

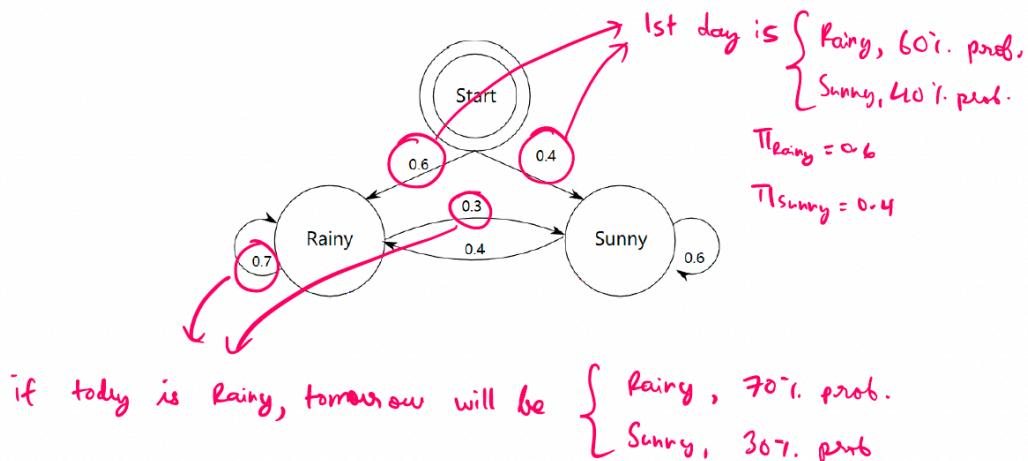
is an example of the transition probability.

- Example 2 (Weather)

States $[S]$ represent weather at each day

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

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



Learning Markov models: MLE

Now suppose we have observed n sequences of examples:

- $x_{1,1}, \dots, x_{1,T}$
- \dots
- $x_{i,1}, \dots, x_{i,T}$
- \dots
- $x_{n,1}, \dots, x_{n,T}$

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 (π, A) ?

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

$$\begin{aligned}
 & \ln P(X_{1:T} = x_{1:T}) \\
 &= \sum_{t=1}^T \ln P(X_t = x_t | X_{1:t-1} = x_{1:t-1}) \quad (\text{always true}) \\
 &= \sum_{t=1}^T \ln P(X_t = x_t | X_{t-1} = x_{t-1}) \quad (\text{Markov property}) \\
 &= \ln \pi_{x_1} + \sum_{t=2}^T \ln a_{x_{t-1}, x_t} \quad (\ln \pi_{x_1} \text{ means } P(X_1 = x_1) = \pi_{x_1}, \sum_{t=2}^T \ln a_{x_{t-1}, x_t} \text{ is the prob of transitioning from } x_{t-1} \rightarrow x_t) \\
 &= \sum_s \mathbb{I}[x_1 = s] \ln \pi_s + \sum_{s,s'} \left(\sum_{t=2}^T \mathbb{I}[x_{t-1} = s, x_t = s'] \right) \ln a_{s,s'}
 \end{aligned}$$

This is just over one sequence, we apply it to sum all sequences.

So MLE is

$$\arg \max_{\pi, A} \sum_s (\#\text{initial states with value } s) \ln \pi_s + \sum_{s,s'} (\#\text{transitions from } s \text{ to } s') \ln a_{s,s'}$$

If $\sum_s \#\text{initial states with value } s$ is large for some s , then π_s should be large for that s .

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

The solution is:

$$\begin{aligned}
 \pi_s &= \frac{\#\text{initial states with value } s}{\#\text{initial states}} \\
 a_{s,s'} &= \frac{\#\text{transitions from } s \text{ to } s'}{\#\text{transitions from } s \text{ to any state}}
 \end{aligned}$$

Learning Markov models: Another perspective

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

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

Using the definition of conditional probability

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

We can estimate this using data

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

We don't use the last state, since it won't transfer to any state

The initial state distribution follows similarly

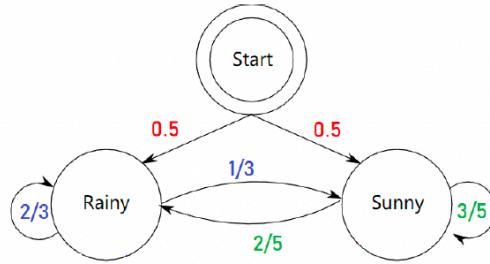
$$P(X_1 = s) \approx \frac{\# \text{times } s \text{ is first state}}{\# \text{sequences}}$$

This is just like estimating bias of a coin / dice.

Exp:

Suppose we observed the following 2 sequences of length 5

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



Sunny occurs 5, rainy occurs 3 (We don't use the last state, since it won't transfer to any state).

For example:

$$P(X_{t+1} = \text{rainy} | X_t = \text{sunny}) = \frac{\text{times } (\text{sunny, rainy}) \text{ occurs}}{\text{times } (\text{sunny}) \text{ occurs (and not the last state)}} = \frac{2}{5}$$

$$P(X_1 = \text{sunny}) = \frac{\text{times sunny is first state}}{\text{number of sequences}} = \frac{1}{2}$$

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. the second-order Markov assumption

$$P(X_{t+1} | X_t, \dots, X_1) = P(X_{t+1} | X_t, X_{t-1})$$

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):

$$P(X_{t+1} | X_t, \dots, X_1) = P(X_{t+1} | X_t, X_{t-1}, \dots, X_{t-n+1})$$

This is n -th order Markov assumption, $X_t, X_{t-1}, \dots, X_{t-n+1}$ is the previous n observations.

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

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

N-gram language models: example

Suppose we are learning a 4-gram Language Model.

~~as the proctor started the clock, the students opened their~~ _____
discard
condition on this

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

For example, suppose that in the corpus:

- "students opened their" occurred 1000 times
 - "students opened their books" occurred 400 times
 - $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
 - "students opened their exams" occurred 100 times
 - $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$
- } Should we have discarded the "proctor" context?

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

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

today the _____

Business and financial news

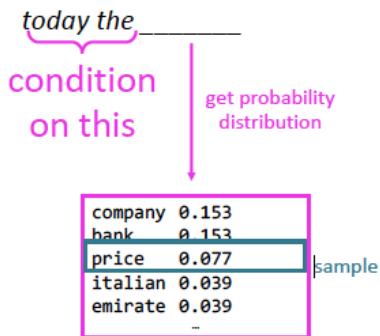
get probability distribution

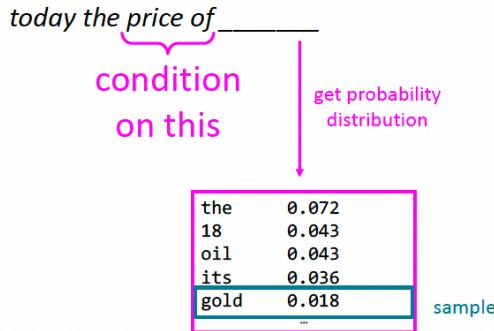
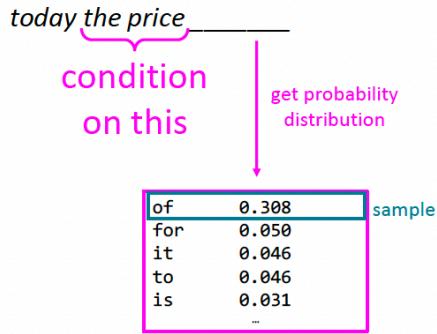
company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
...	

Notice that there isn't that much granularity in the distribution,
because "today the" doesn't appear too often in corpus.
Most two-grams won't appear too often.

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

Recurrent Neural Networks

The problem with fixed-window

Recall the language modeling task:

- Input: sequence of words $x^{(1)}, \dots, x^{(t)}$. (changing notation, $x^{(1)}$ is overloaded to refer to both random value and its value)
- Output: prob list of the next word $P(x^{(t+1)}|x^{(t)}, \dots, x^{(1)})$.

How about a window-based neural model?



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

$$\begin{aligned}'the' &= [1, 0, \dots, 0] \rightarrow s \text{ dim vectors} \\ 'students' &= [0, 1, \dots, 0] \rightarrow s \text{ dim vectors}\end{aligned}$$

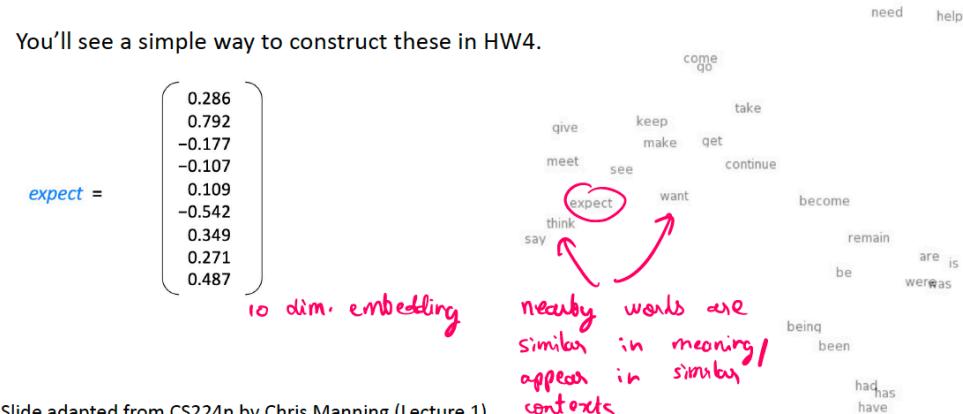
It's high dimensioned, and each presentation is orthogonal, even similar words have representation which are far away.

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.



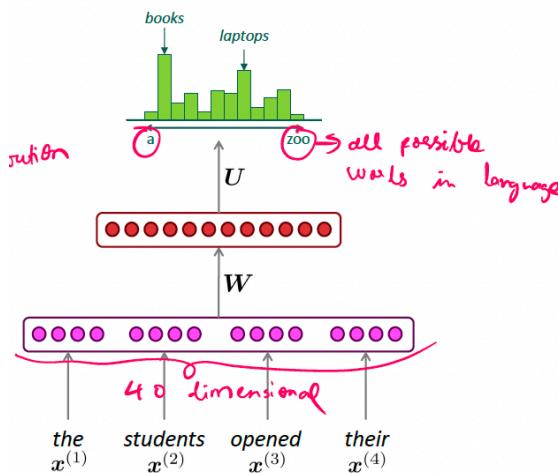
Slide adapted from CS224n by Chris Manning (Lecture 1)

Nearby words are similar meanings / appear in similar contexts.

A fixed-window neural language model

Same architecture as neural networks in HW3.

- Output distribution: $\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$.
- Hidden layer: $h = f(We + b_1)$, f is non-linearity (like ReLU).
- Concatenated word embeddings: $e = [e^{(1)}, e^{(2)}, e^{(3)}, e^{(4)}]$, supposed each is 10 dimensional.
- Words / one-hot vectors: $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$.



Problem with this architecture:

- Uses a fixed window, which can be too small.

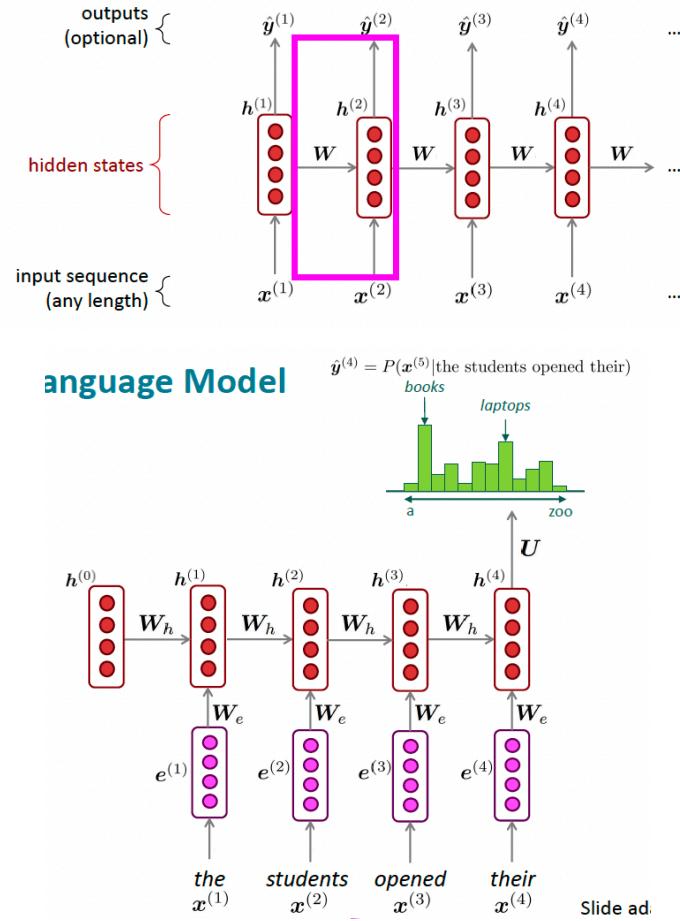
- Enlarging this window will enlarge the size of the weight matrix W .
- The inputs $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W .
- No symmetry in how inputs are processed!

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?

RNN

Core idea: apply the same weight W repeatedly. (Similar to what we did with filters in CNNs)



Note: this input sequence could be much longer now.

- Output distribution: $\hat{y}^{(t)} = \text{softmax}(Uh^{(t)} + b_2) \in \mathbb{R}^{|V|}$.
- Hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

σ is activation functions (ReLU).

$h^{(0)}$ is the initial hidden state. b is the bias.

- Word embeddings: $e^{(t)}$ for word $x^{(t)}$.

Training an RNN language model

Get a big corpus of text which is a sequence of words $x^{(1)}, \dots, x^{(T)}$

Feed into RNN-LM; compute output distribution $\hat{y}^{(t)}$ for every step t .

i.e. predict probability dist of every word, given words so far

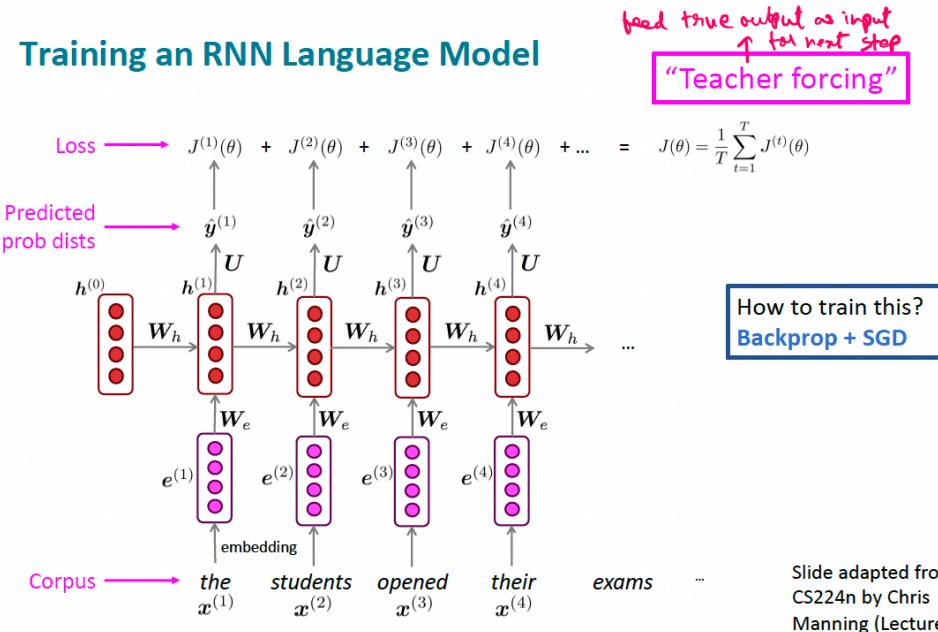
Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$):

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

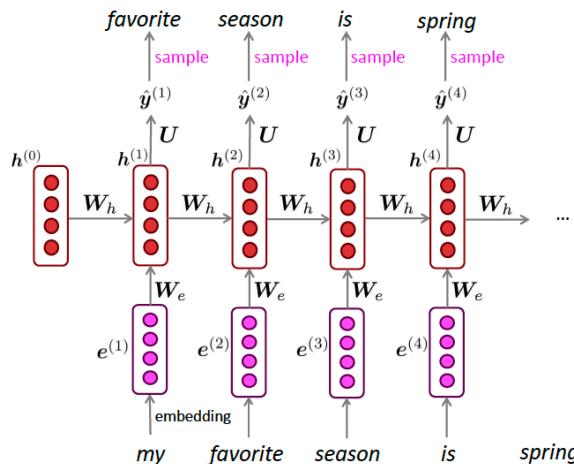
This is same as multi-class classification

Average this to get overall loss for entire training set:

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



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.



Summary

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

Ilia 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: