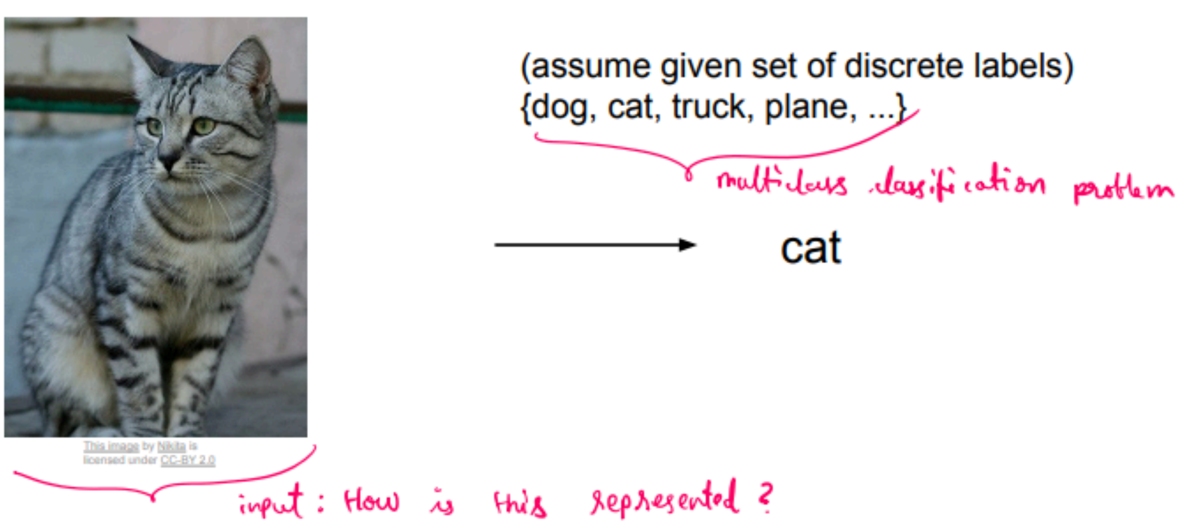
Lecture 7

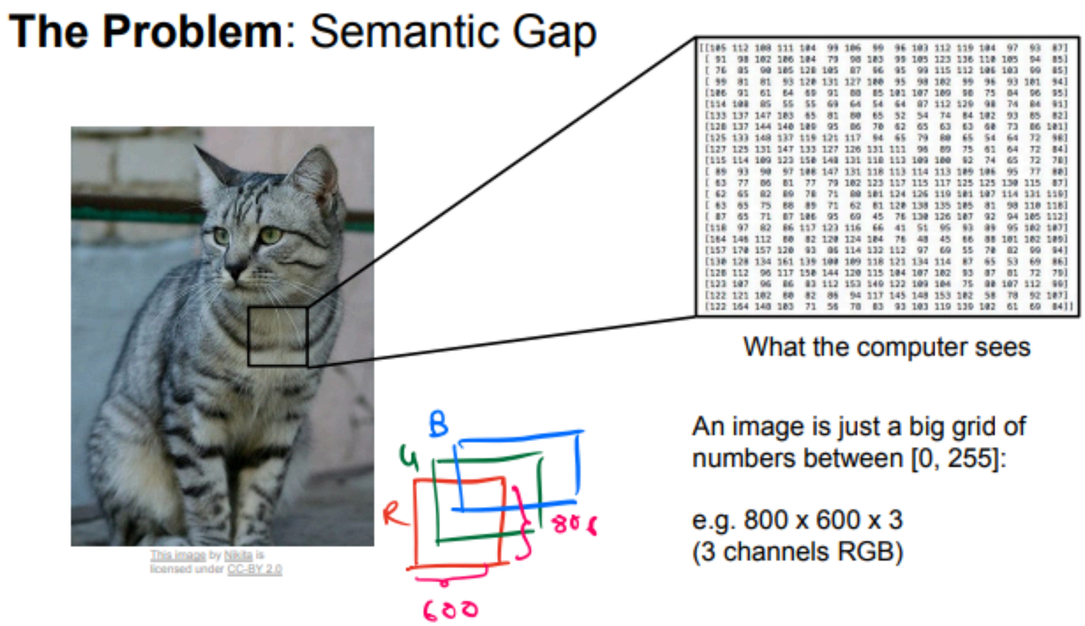
Convolutional Neural Networks (Convent / CNN)

Intro

Image classification: a core task in computer vision



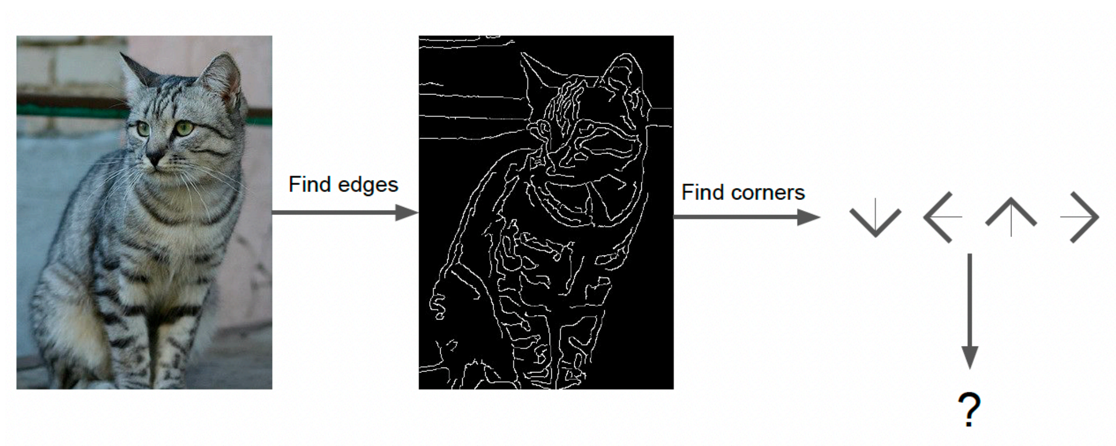
The problem: semantic gap



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?

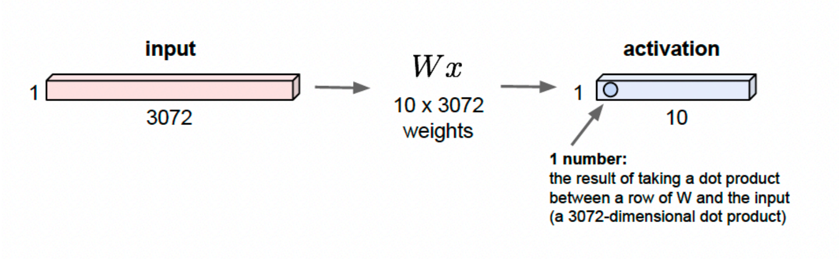
The ingredients:

- 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 -> 3072×1



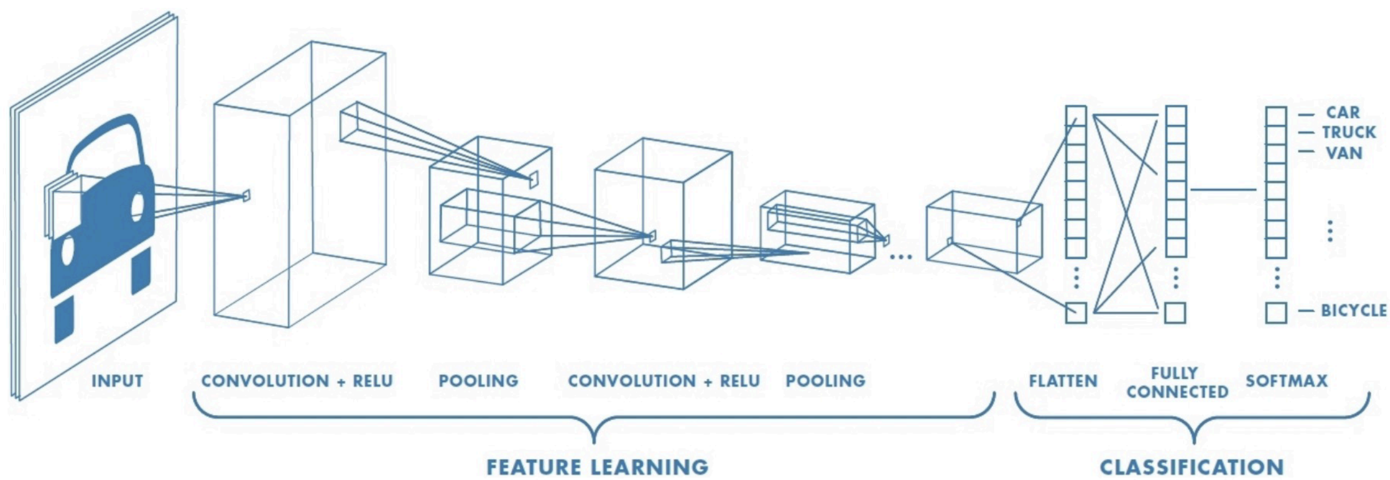
Completely loses out on spatial structure.

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

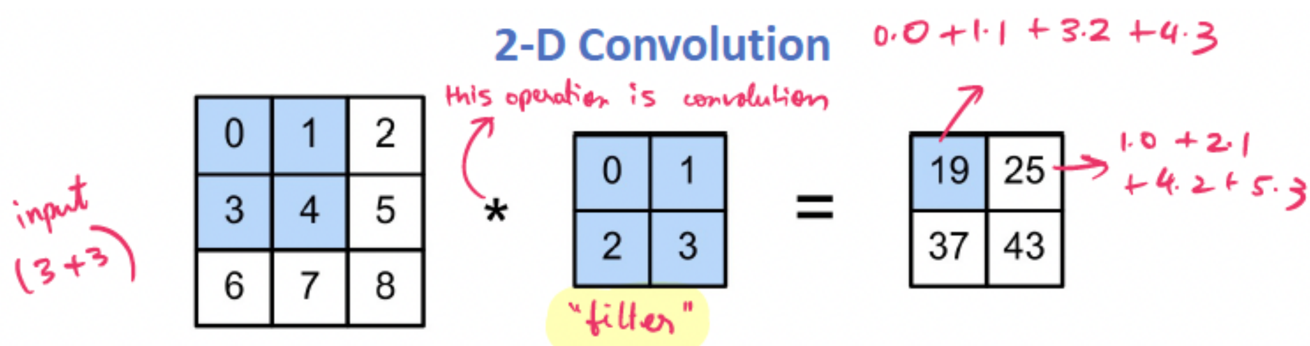
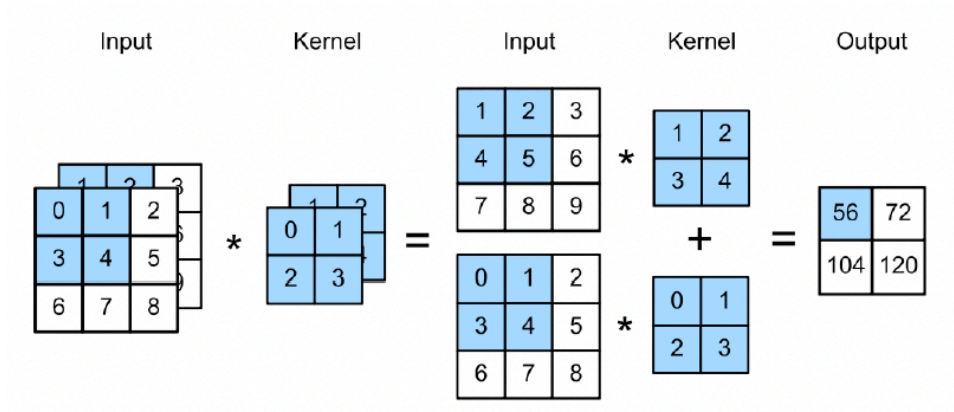


Figure 14.5: Illustration of 2d cross correlation. Generated by conv2d_jax.ipynb. Adapted from Figure 6.2.1 of [Zha+20].

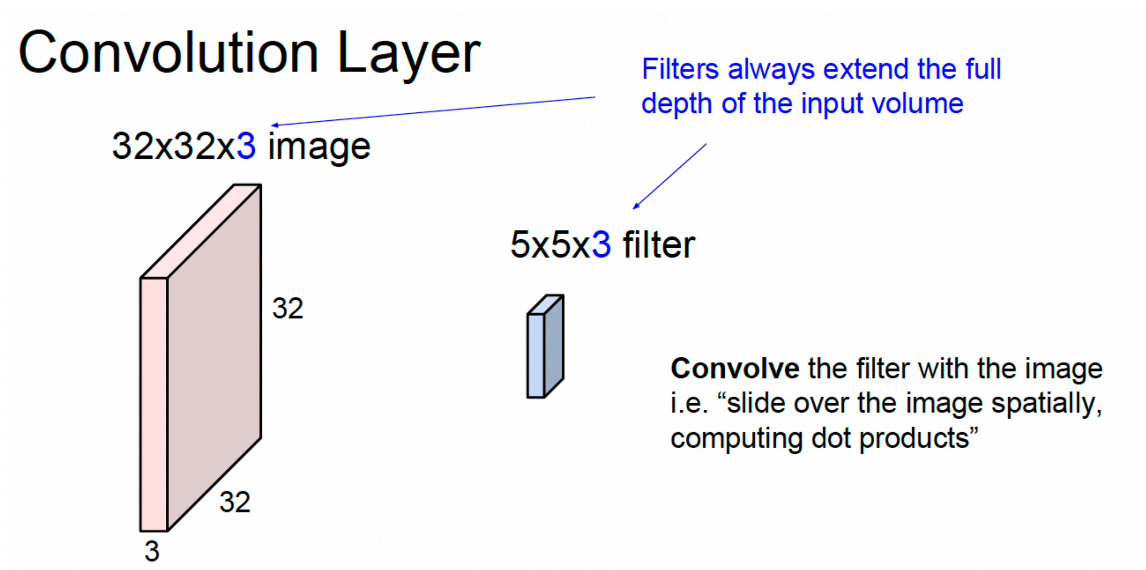
3-D Convolution

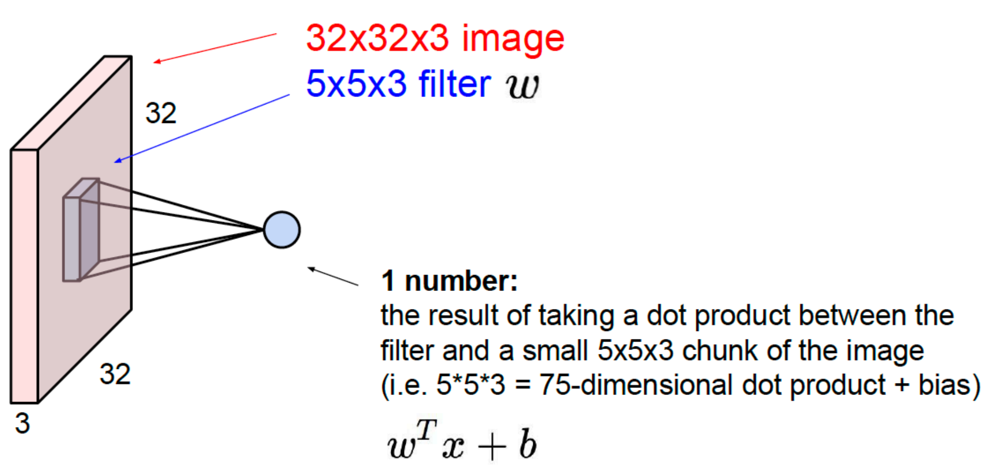
The input is $3 \times 3 \times 2$

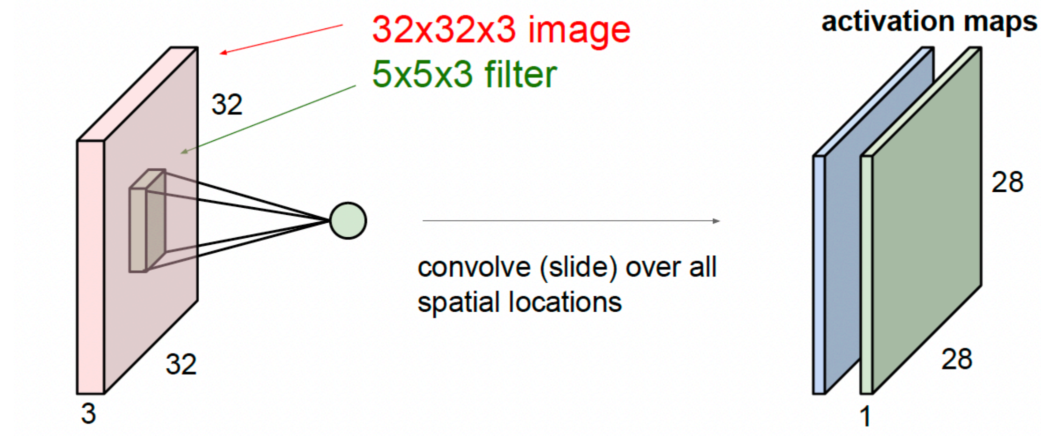


Add up the result for the two channels

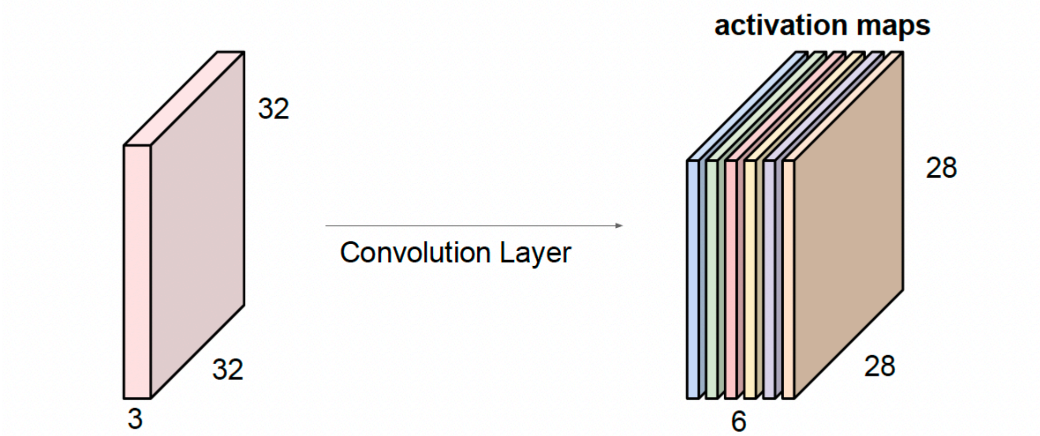
Convolution Layer





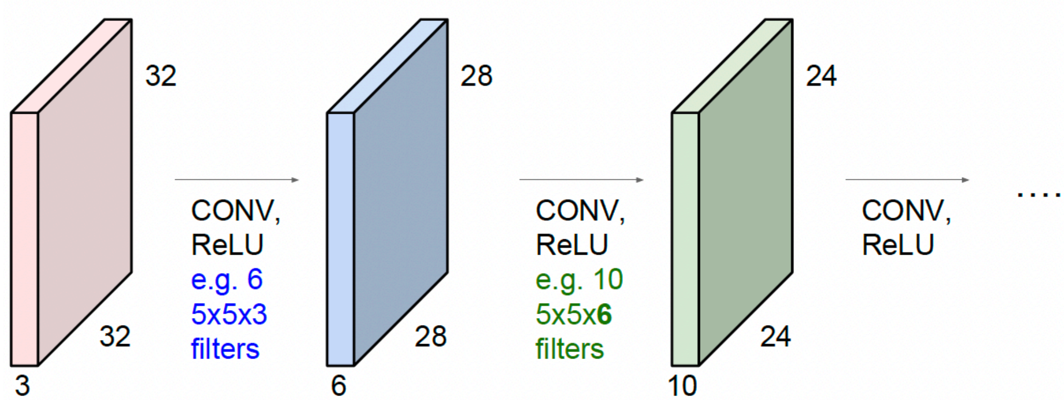


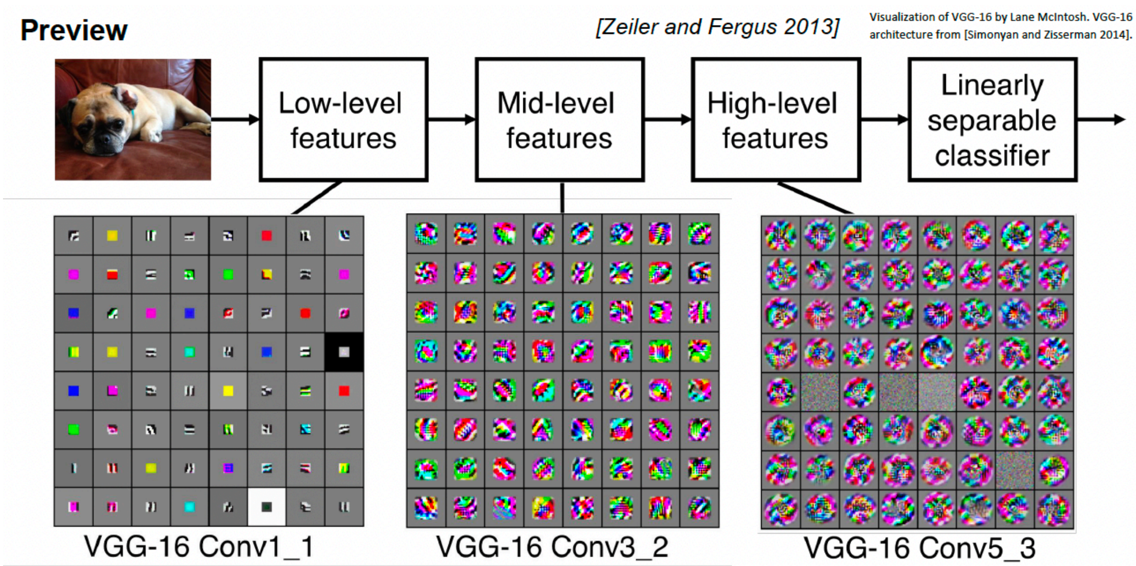
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



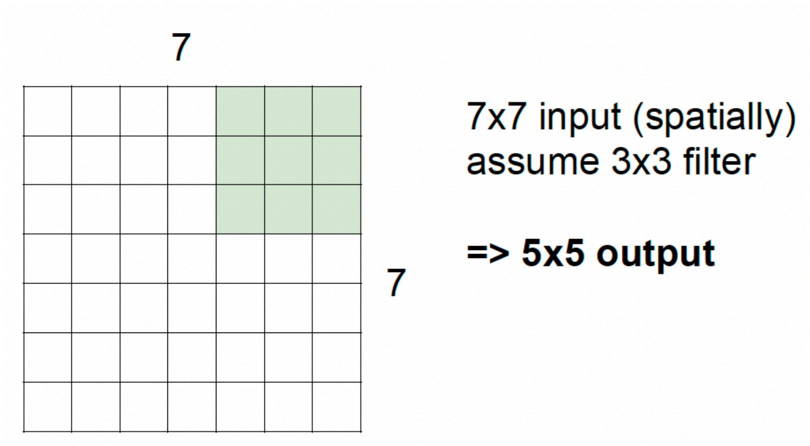
We stack these up to get a "new image" of size 28x28x6!

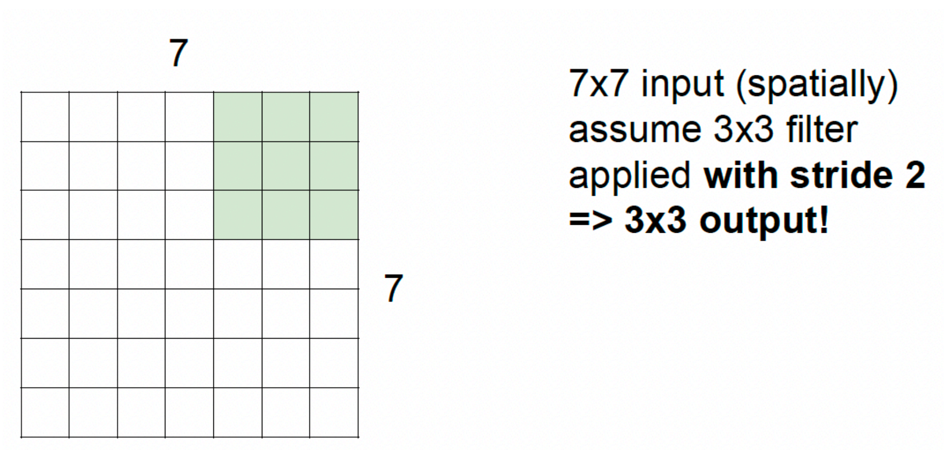
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

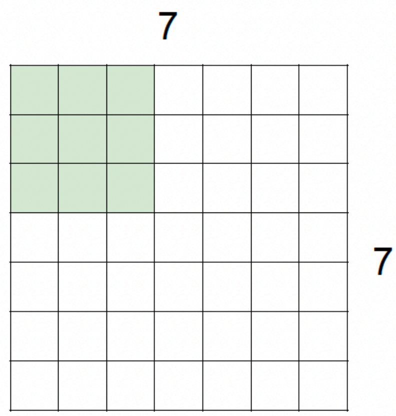




Calculation:

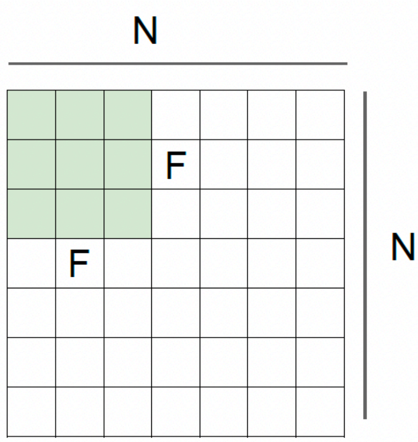






7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

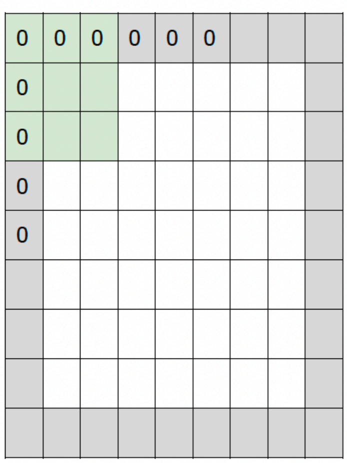
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

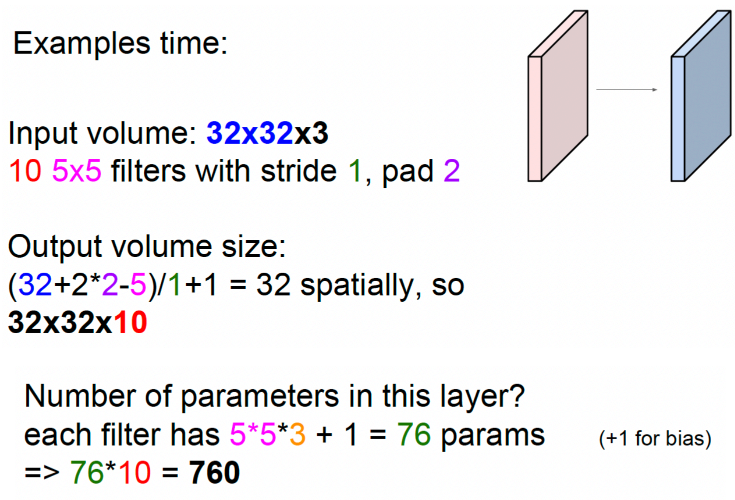
In practice: Common to zero pad the border.



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

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

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



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/

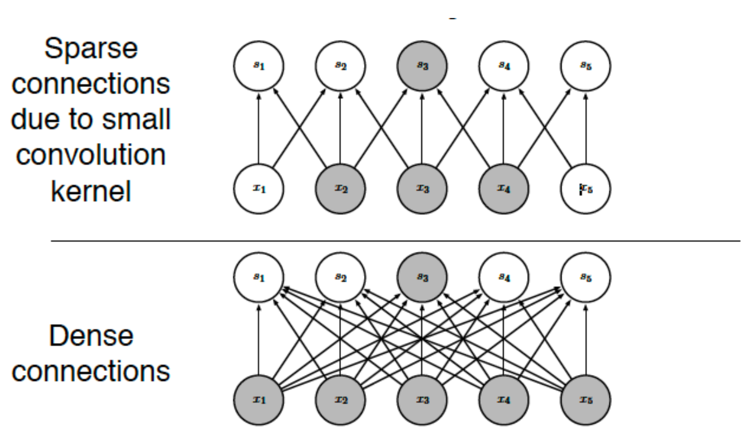
Connection to fully-connected network

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

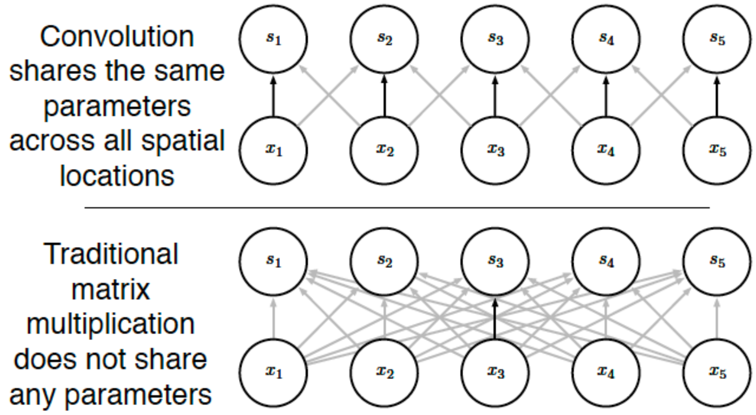
Filter = weights with sparse connection and parameter sharing.

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

Sparse connectivity: being affected by less

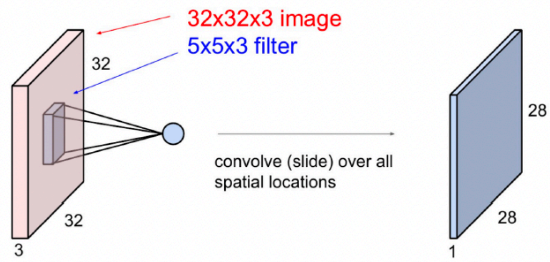


Parameter sharing



Much fewer parameters: (Ex ignoring bias terms)

FC layer: (32 × 32 × 3) × (28 × 28) ≈ 2.4M
Conv layer: 5 × 5 × 3 = 75



Pooling Layer

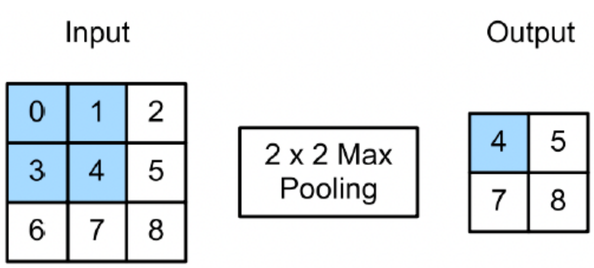
Makes the representations smaller and more manageable

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

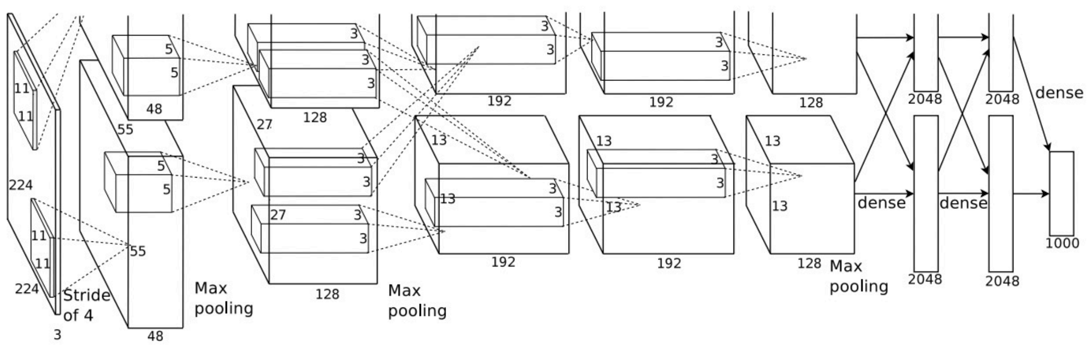


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input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

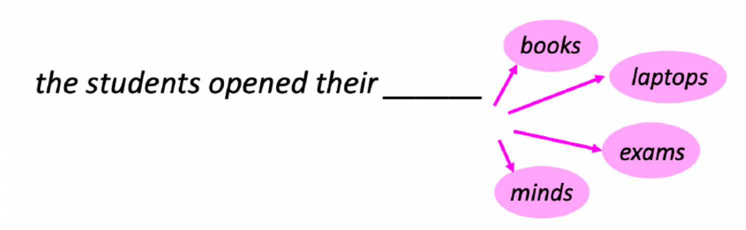
Given observations $x_1, x_2, \cdots, x_{t-1}$ (input), what is x_t (output).

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

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

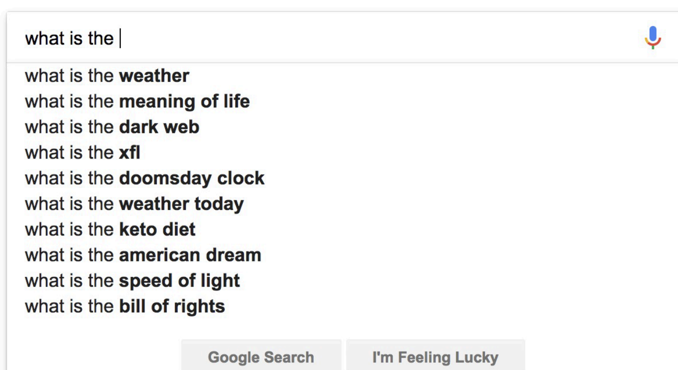
A system that does this is known as language model.

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 $x_1, \cdots, x_T$, then the probability of this text (according to the Language Model) is:

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

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 $X_1, X_2, \cdots$, 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 $X_1, \cdots, X_t$). This is a bigram(2-gram) model!

We will consider the following setting:

- All X_t'S take value from the same discrete set $\{1, \cdots, S\}$. 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.
- $(\{\pi_s\}, \{a_{sis'}\}) = (\pi, A)$ are parameters of the model. ($A \in \mathbb{R}^{R\times S}$ is the matrix where the entry corresponding to s, s' is $a_{s,s'}$).

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

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

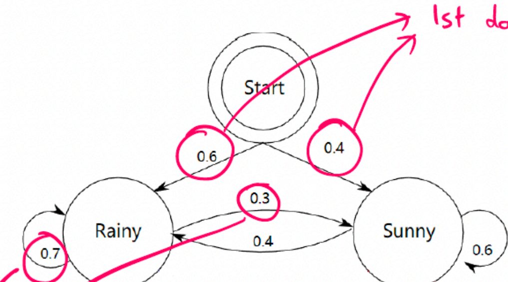
$$a_{\text{ice,cream}} = P(X_{t+1} = \text{cream} \mid X_t = \text{ice})$$

  is an example of the transition probability.

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

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

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: MLE

Now suppose we have observed n sequences of examples:

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

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 $x_1, \cdots, x_T$ is

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

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

So MLE is

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

If $\sum_s \#initial\ states\ with\ value\ s$ 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:

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

Learning Markov models: Another perspective

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

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

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

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

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

The initial state distribution follows similarly

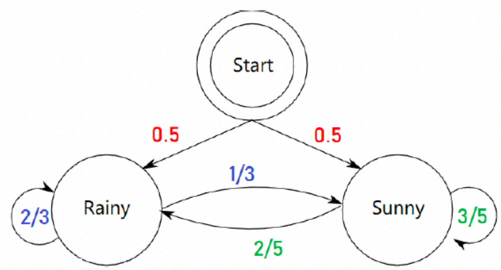
$$P(X_1=s)\approx\frac{\#times\ s\ is\ first\ state}{\#sequences}$$

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:

$$P(X_{t+1}=rainy|X_t=sunny)=\frac{times\ (sunny,rainy)\ occurs}{times\ (sunny)\ occurs(and\ not\ the\ last\ state)}=\frac{2}{5}$$
$$P(X_1=sunny)=\frac{times\ sunny\ is\ first\ state}{number\ of\ sequences}=\frac{1}{2}$$

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

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

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):

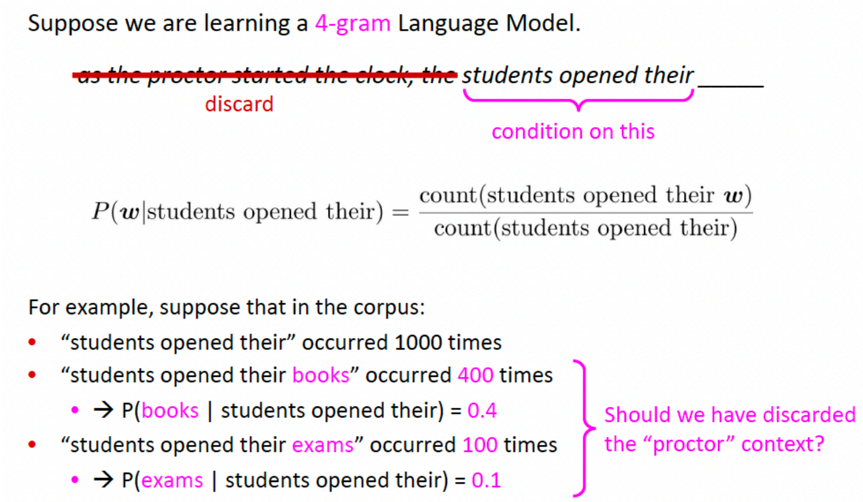
$$P(X_{t+1}|X_t,\cdots,X_1)=P(X_{t+1}|X_t,X_{t-1},\cdots,X_{t-n+1})$$

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

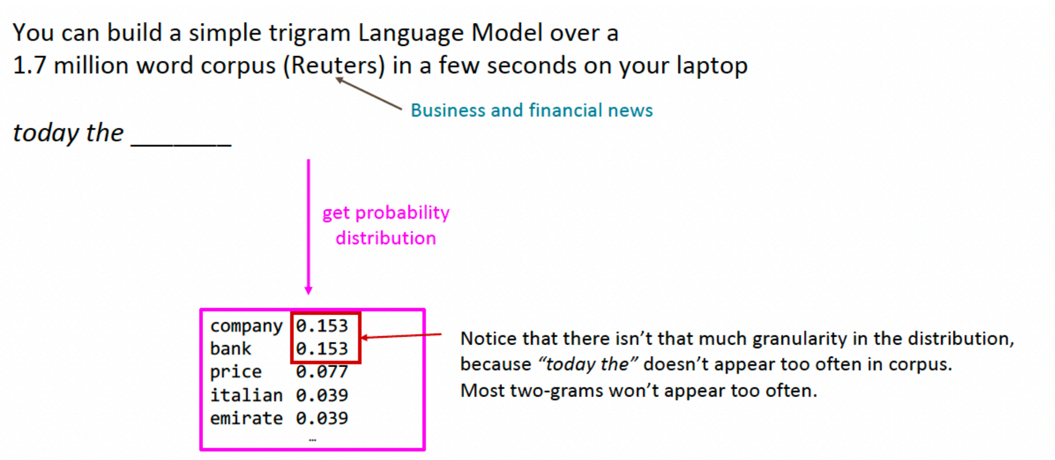
Learning higher order Markov chains is similar, but more expensive.

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

N-gram language models: example

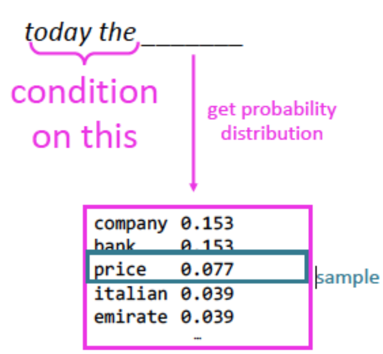


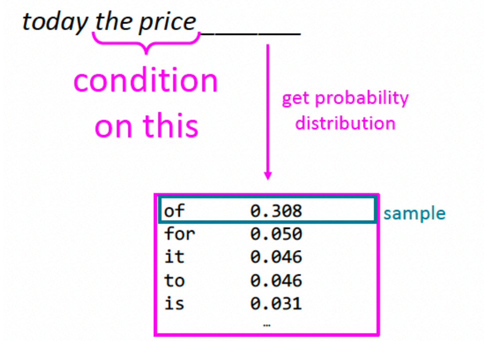
You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop.

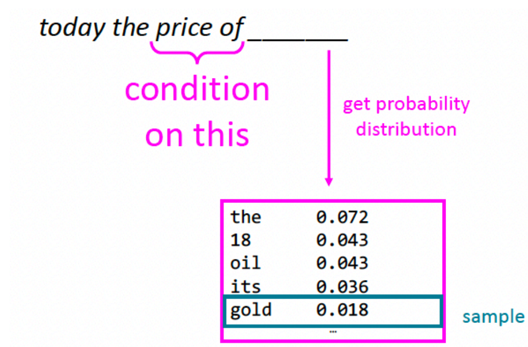


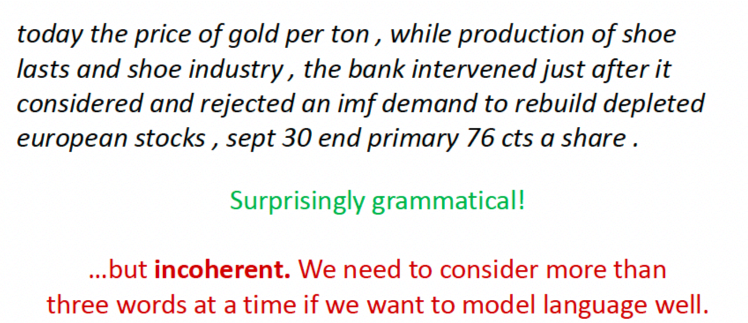
Generating text with a n-gram Language Model

You can also use a language model to generate text









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

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

$$'the' = [1, 0, \cdots, 0] \to s\ dim\ vectors$$
$$'students' = [0, 1, \cdots, 0] \to s\ dim\ vectors$$

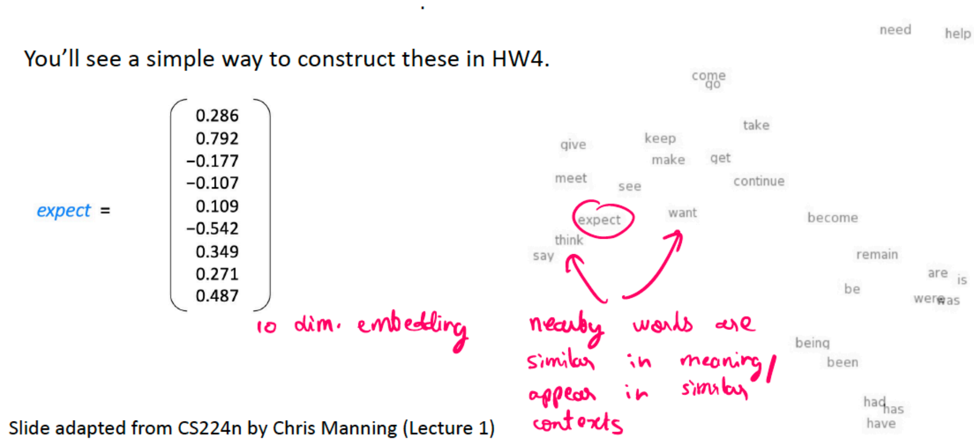
It's high dimensioned, and each presentation is orthogonal, even similar words have representation which are far away.

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.



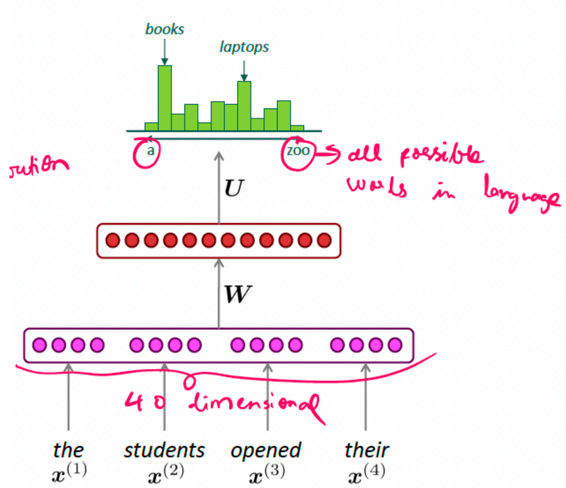
Slide adapted from CS224n by Chris Manning (Lecture 1)

Nearby words are similar meanings / appear in similar contacts.

A fixed-window neural language model

Same architecture as neural networks in HW3.

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

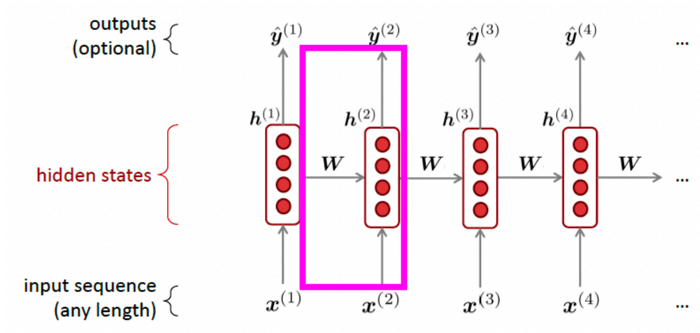
  No symmetry in how inputs are processed!

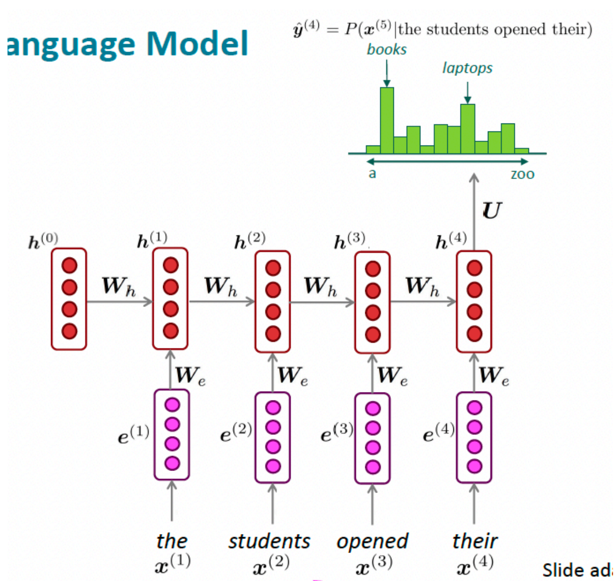
As with CNNs for images before, we need an architecture which has similar symmetries as the data.

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: $\hat{y}^{(t)} = softmax(Uh^{(t)} + b_2) \in \mathbb{R}^{|V|}$.
- 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 $x^{(1)}, \cdots, x^{(T)}$

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

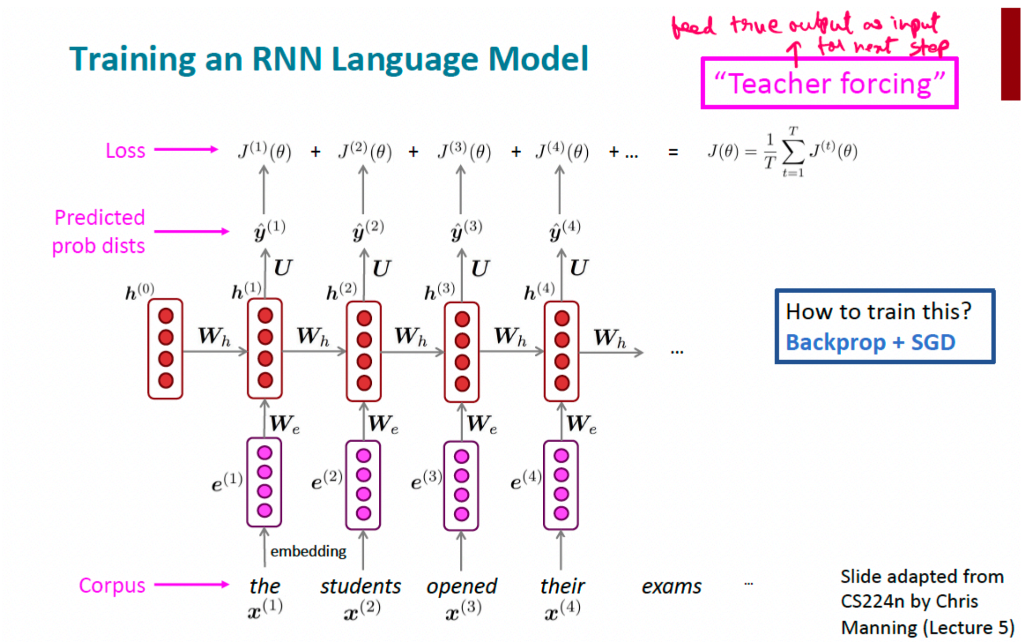
Loss function on step t is cross-entropy between predicted probability distribution $\hat{y}^{(t)}$, and the true next word $y^{(t)}$ (one-hot for $x^{(t+1)}$) :

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

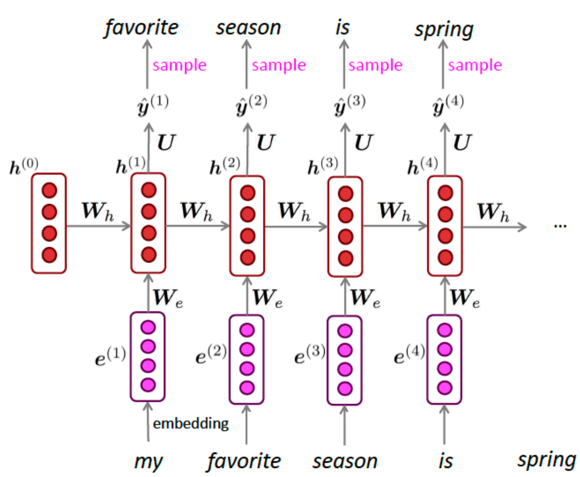
This is same as multi-class classification

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

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: