Finishing Computer Vision; Word Vectors & word2vec

Robin Jia
USC CSCI 467, Spring 2024
February 22, 2024

With a lot borrowed from Jurafsky & Martin, "Speech and Language Processing"
https://web.stanford.edu/~jurafsky/slp3/

Review: Convolutions

-1	2	-1
-1	2	-1
-1	2	-1

Kernel
(K=3)

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

Input
(5 x 6)

input[1:4,2:5]

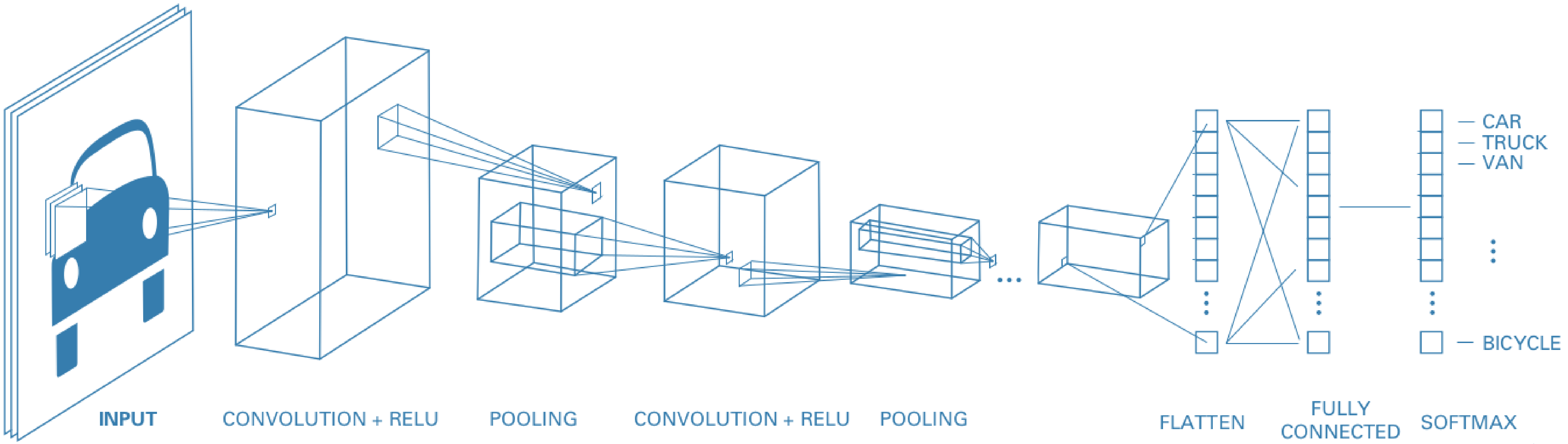
3	-1	0	0
5	-2	0	0
3	-1	0	0

Output
(5-3+1 x 6-3+1)
=(3 x 4)

(1, 2)-th element

- **Convolutional Layer**
  - Extract 1 feature for each window of input by applying kernel
  - Output is computed as a dot product (linear operation)
- **Local Receptive Field**: Each output cell is computed based on a small window of the input image
- **Weight Sharing**: Same kernel used to process each window of the input image
  - The kernel defines a classifier (e.g., is there a moose here?) that gets applied to every window of the image

Review: Convolutional Neural Networks

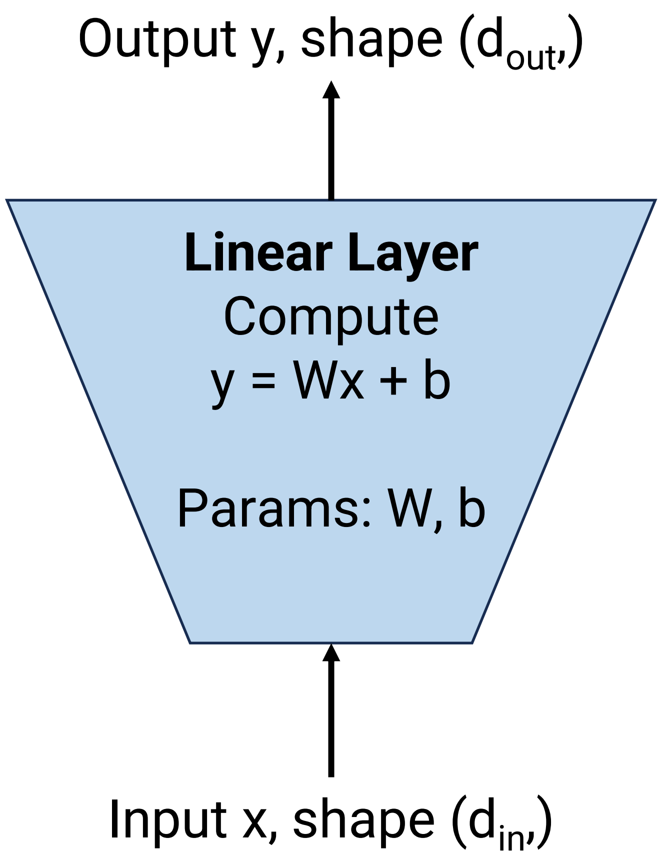


- Input -> Conv+ReLU + Pool -> Fully connected layer -> Output
  - Convolutions at beginning to understand each small window of image
  - Fully connected layer at end to make overall prediction

Review: The Basic “Building Blocks”

(1) Linear Layer

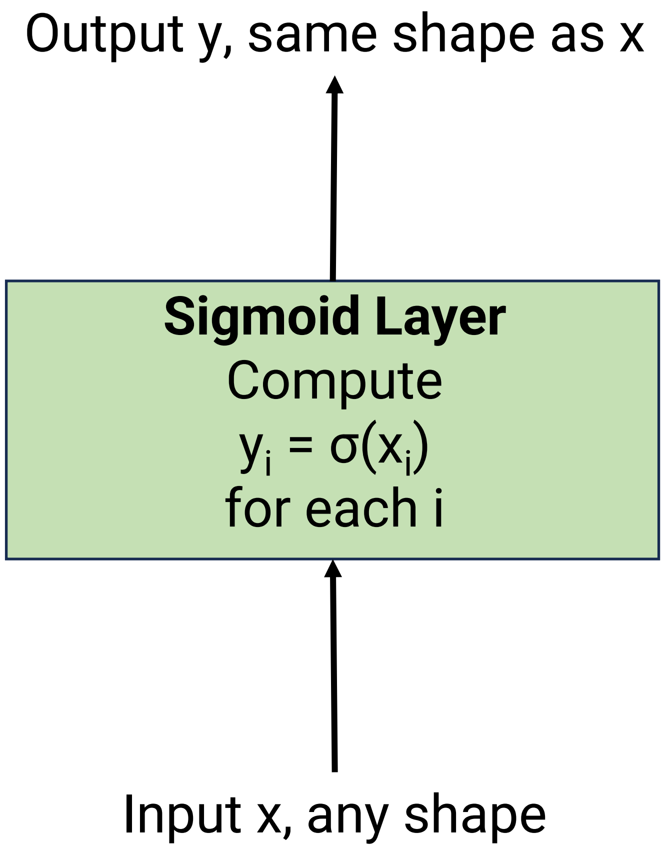
- Input x: Vector of dimension d_{in}
- Output y: Vector of dimension d_{out}
- Formula: $y = Wx + b$
- Parameters
  - W: d_{out} x d_{in} matrix
  - b: d_{out} vector
- In pytorch: nn.Linear()



Review: The Basic "Building Blocks"

(2) Non-linearity Layer

- Input x: Any number/vector/matrix
- Output y: Number/vector/matrix of same shape
- Possible formulas:
  - Sigmoid: y = σ(x), elementwise
  - Tanh: y = tanh(x), elementwise
  - Relu: y = max(x, 0), elementwise
- Parameters: None
- In pytorch: torch.sigmoid(), nn.functional.relu(), etc.



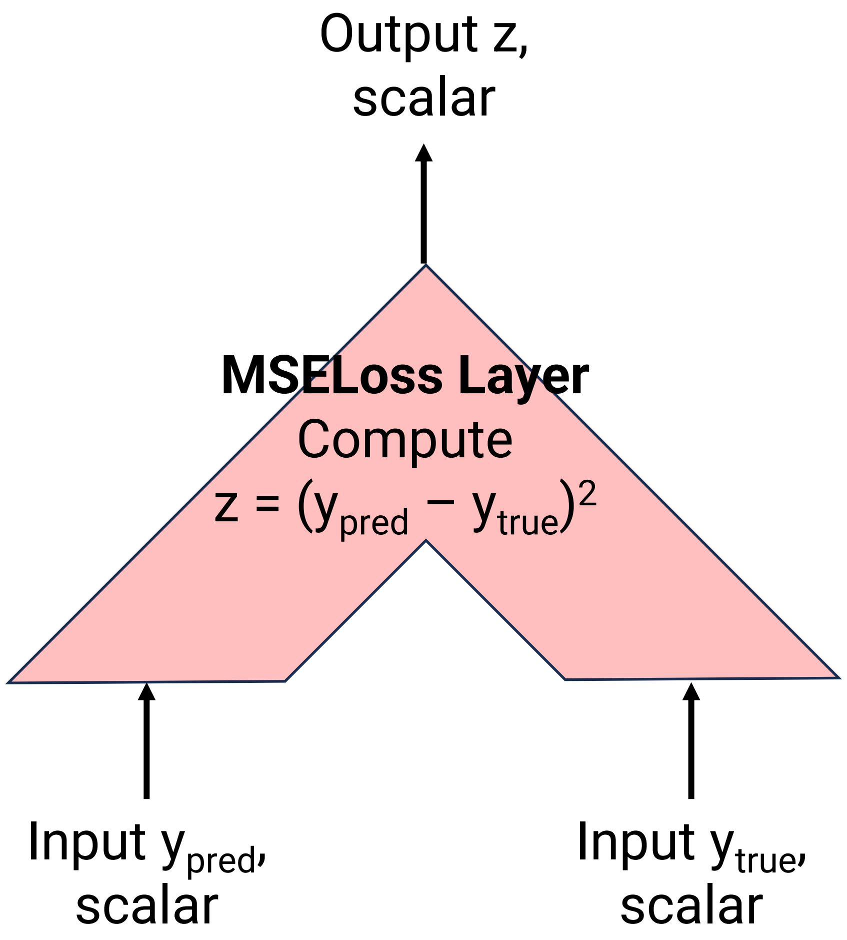
Review: The Basic "Building Blocks"

(3) Loss Layer

- Inputs:
  - y_{pred}: shape depends on task
  - y_{true}: scalar (e.g., correct regression value or class index)
- Output z: scalar
- Possible formulas:
  - Squared loss: y_{pred} is scalar, $z = (y_{pred} - y_{true})^2$
  - Softmax regression loss: y_{pred} is vector of length C,

$$z = -\left(y_{\text{pred}}[y_{\text{true}}] - \log \sum_{i=1}^{C} \exp(y_{\text{pred}}[i]) \right)$$

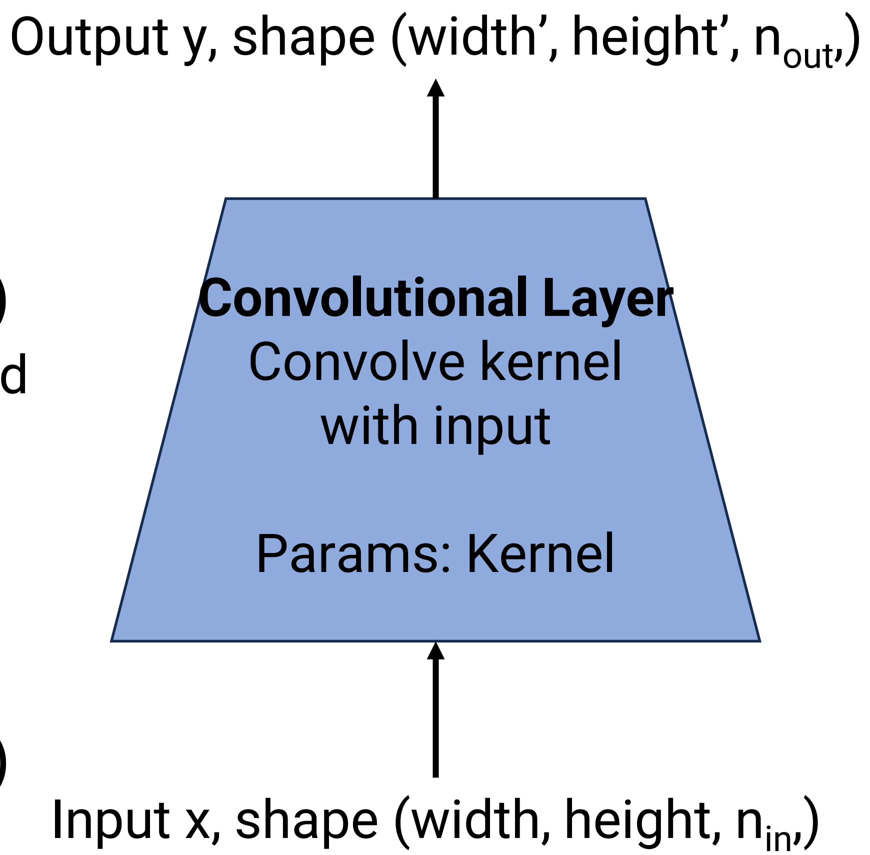
- Parameters: None
- In pytorch: nn.MSELoss(), nn.CrossEntropyLoss(), etc.



CNN "Building Blocks"

(4) Convolutional Layer

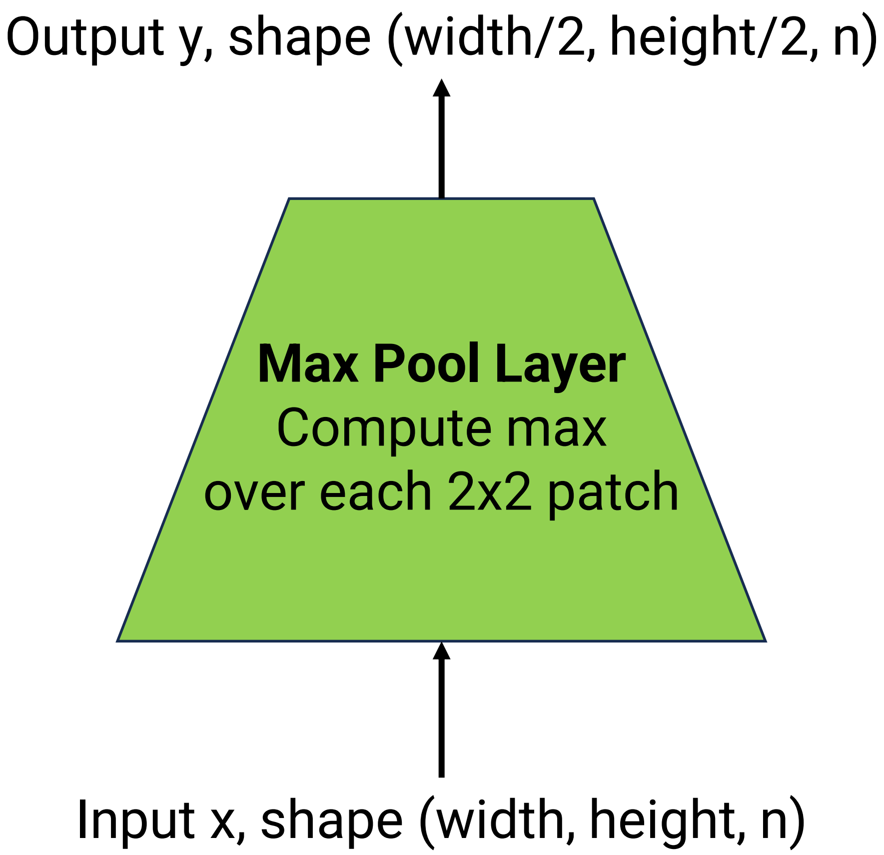
- Input x: Tensor of dimension (width, height, n_{in})
  - n_{in}: Number of input channels (e.g. 3 for RGB images)
- Output y: Tensor of dimension (width', height', n_{out})
  - width', height': New width & height, depends on stride and padding
  - n_{out}: Number of output channels
- Formula: Convolve input with kernel
  - Recall: This is in fact a linear operation
- Parameters: Kernel params of shape (K, K, n_{in}, n_{out})
- In pytorch: nn.Conv2d()



CNN "Building Blocks"

(5) Max Pooling layer

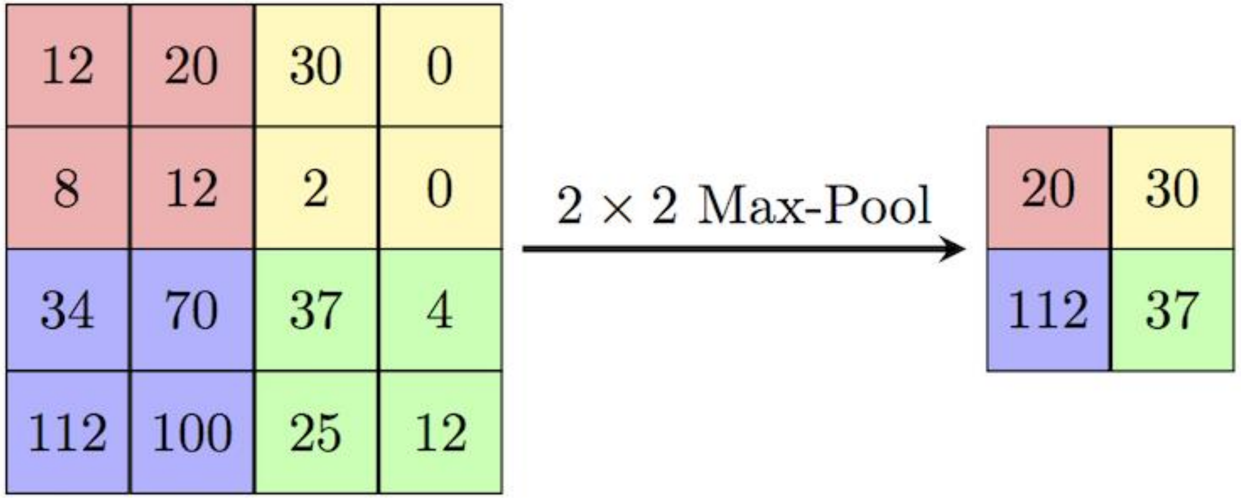
- Input x: Tensor of dimension (width, height, n)
  - n: Number of channels
- Output y: Tensor of dimension (width/2, height/2, n)
- Formula: In each 2x2 patch, compute max
- Parameters: None
- In pytorch: nn.MaxPool2d()

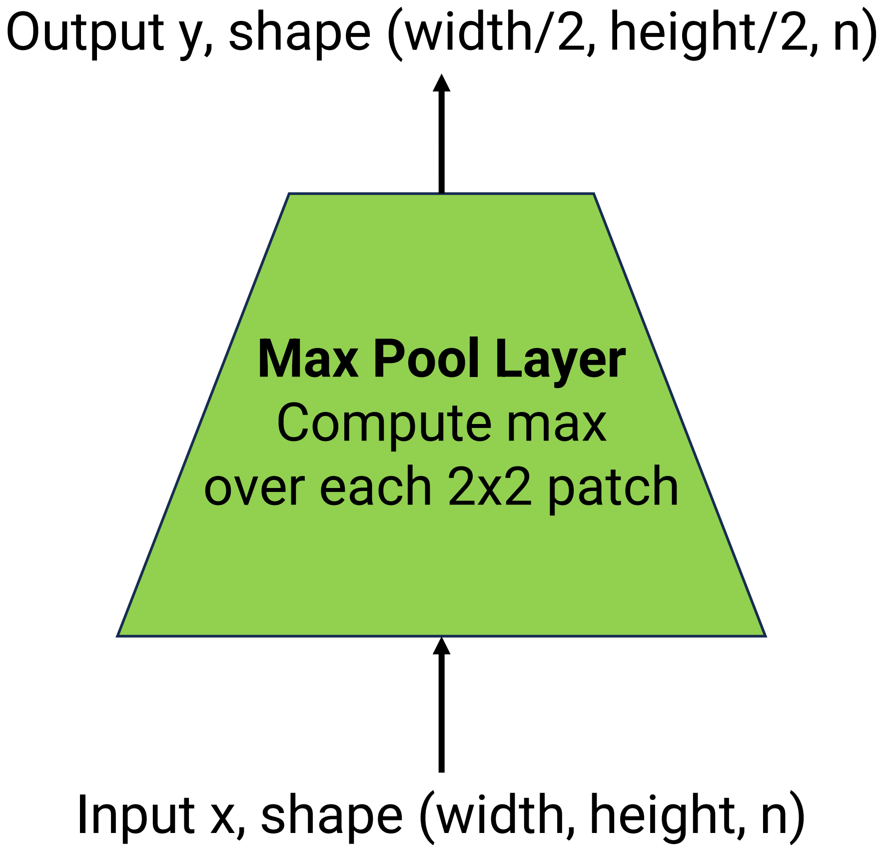


CNN "Building Blocks"

(5) Max Pooling layer

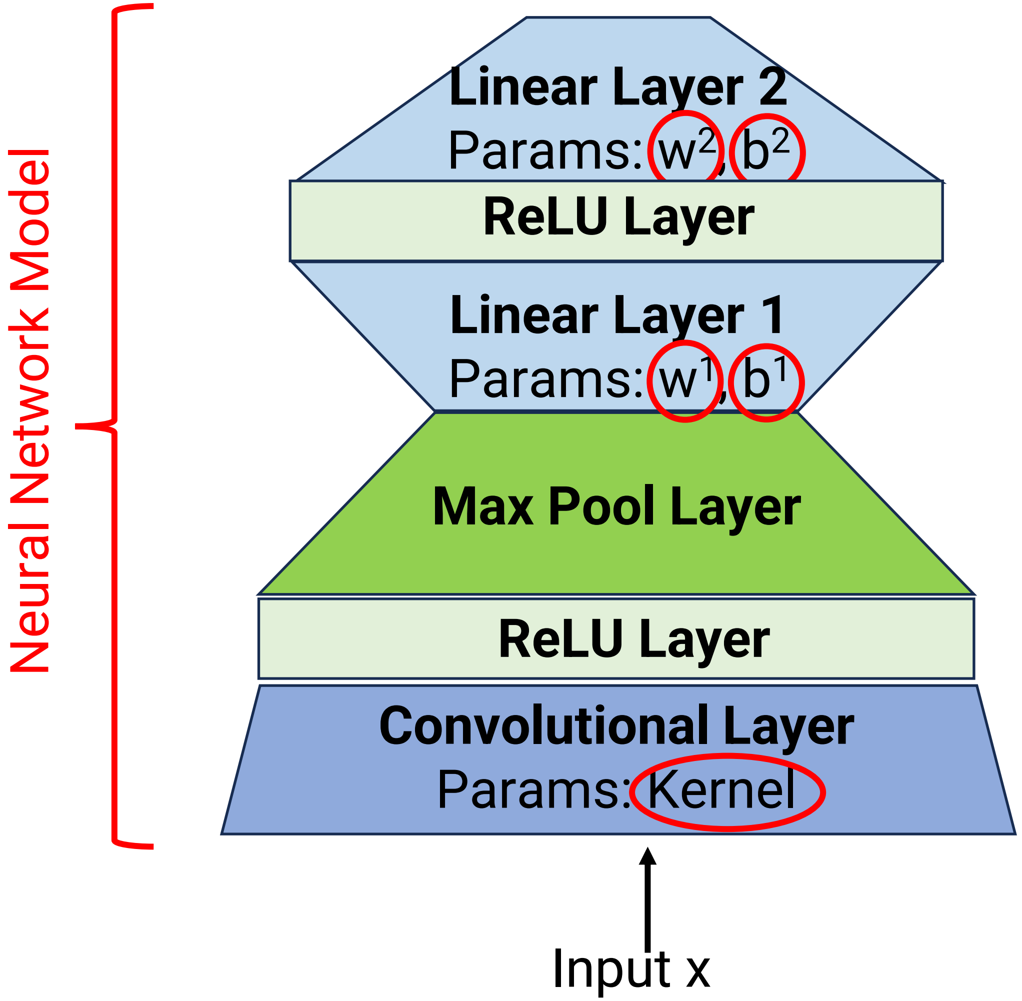
- Input x: Tensor of dimension (width, height, n)
  - n: Number of channels
- Output y: Tensor of dimension (width/2, height/2, n)





Building a CNN Model

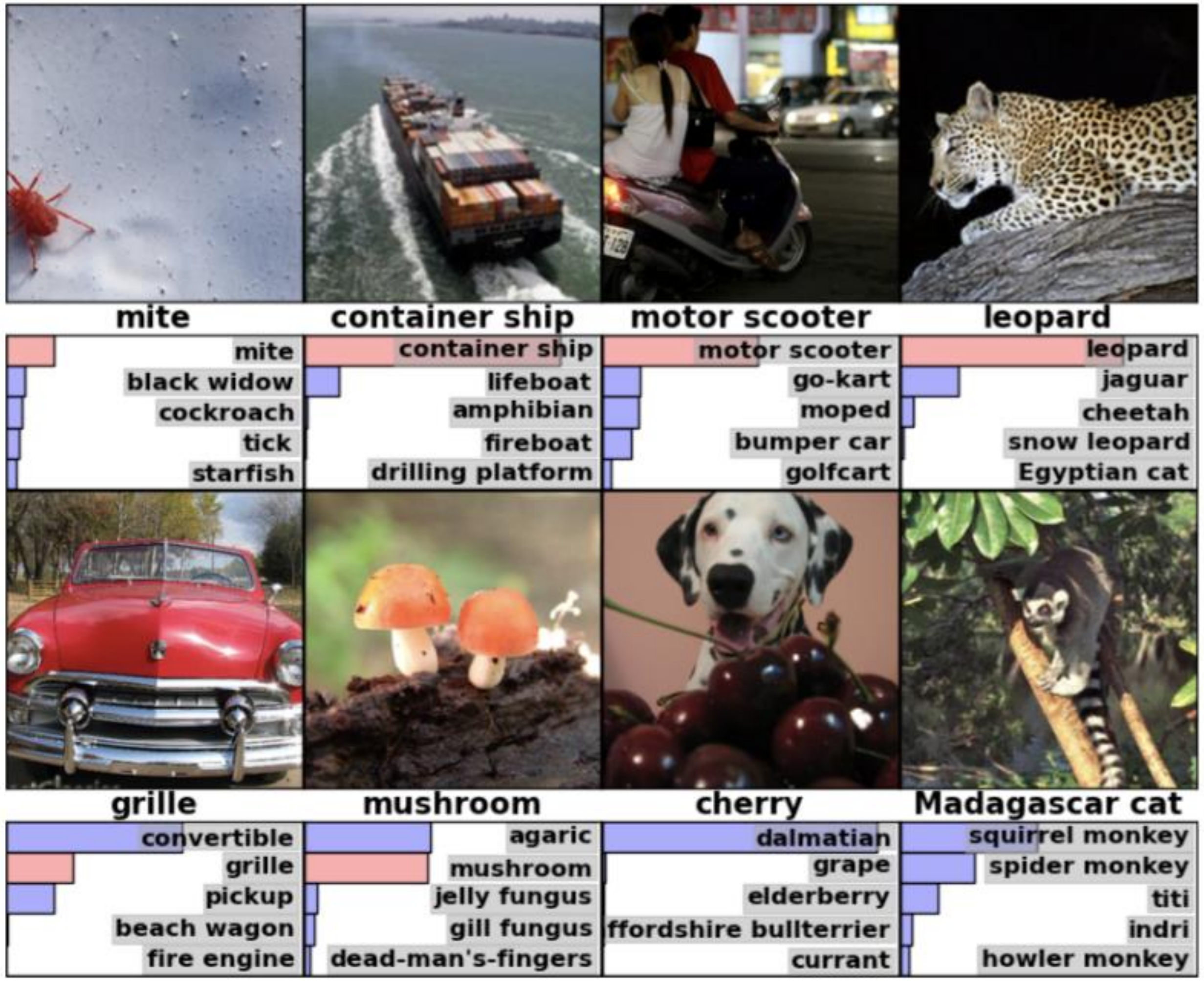
- A generic CNN architecture
  - First use conv + relu + pool to extract features
  - Then use MLP to make final prediction
- Basic steps are still all the same
  - Backpropagation still works
- Gradient descent needed to update all parameters



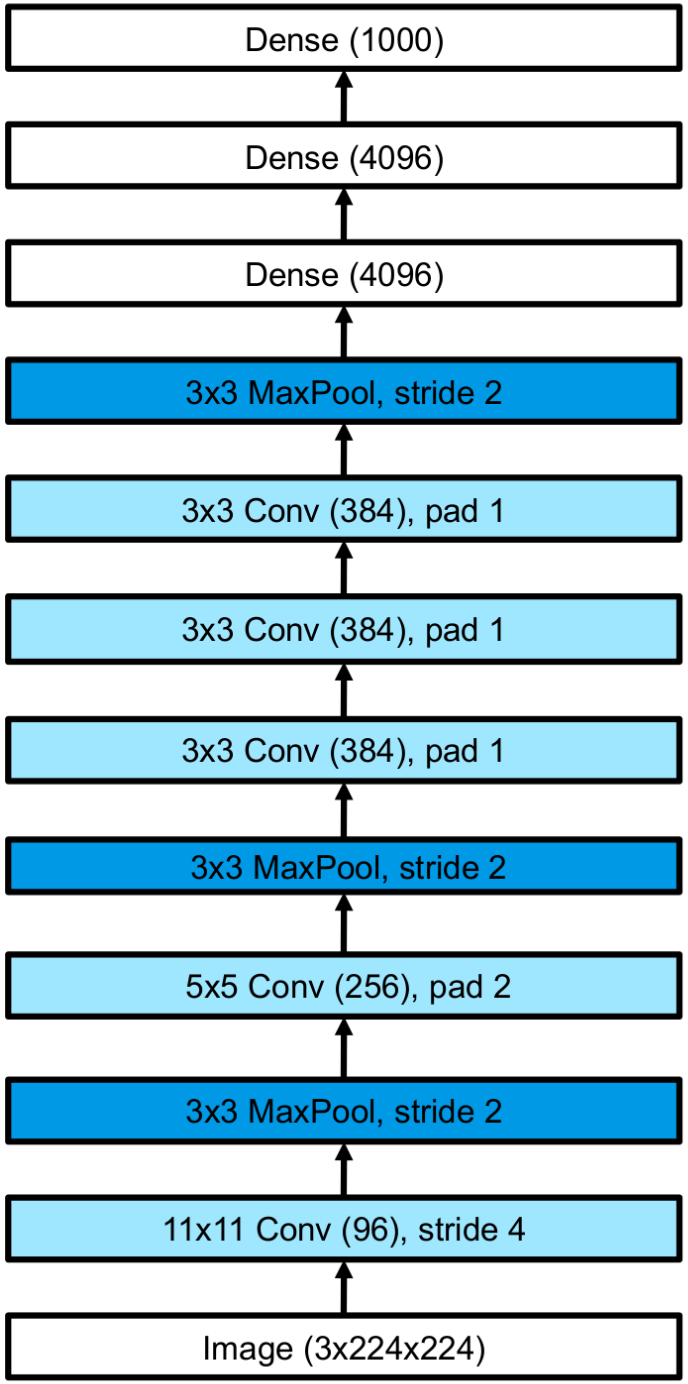
Outline

- Computer vision tasks
- Word vectors
    - What do we want?
    - word2vec
    - Solving analogies
    - Bias in word vectors

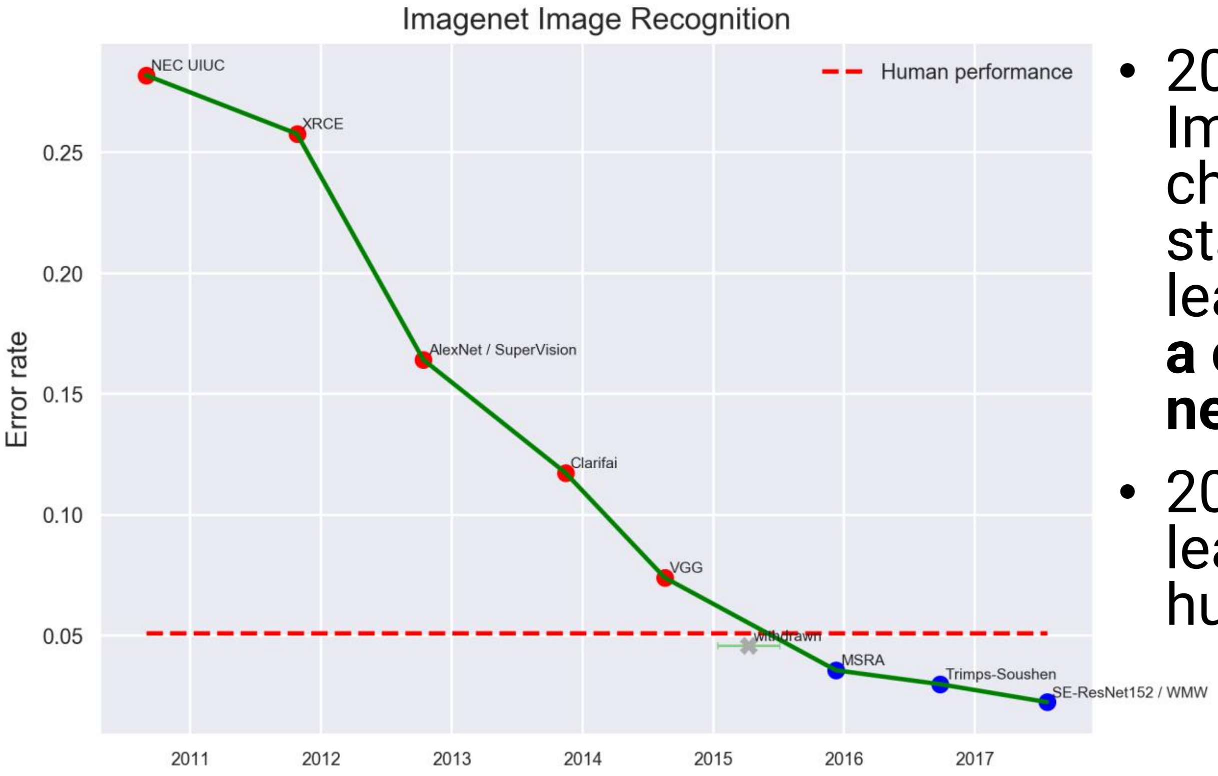
Image Classification



- ImageNet dataset: 14 million images, 1000 labels
- **CNNs do very well at these tasks!**



Progress on ImageNet



- 2012: AlexNet wins ImageNet challenge, marks start of deep learning era **(and is a convolutional neural network)**
- 2016: Machine learning surpasses human accuracy

Source: https://www.eff.org/files/AI-progress-metrics.html

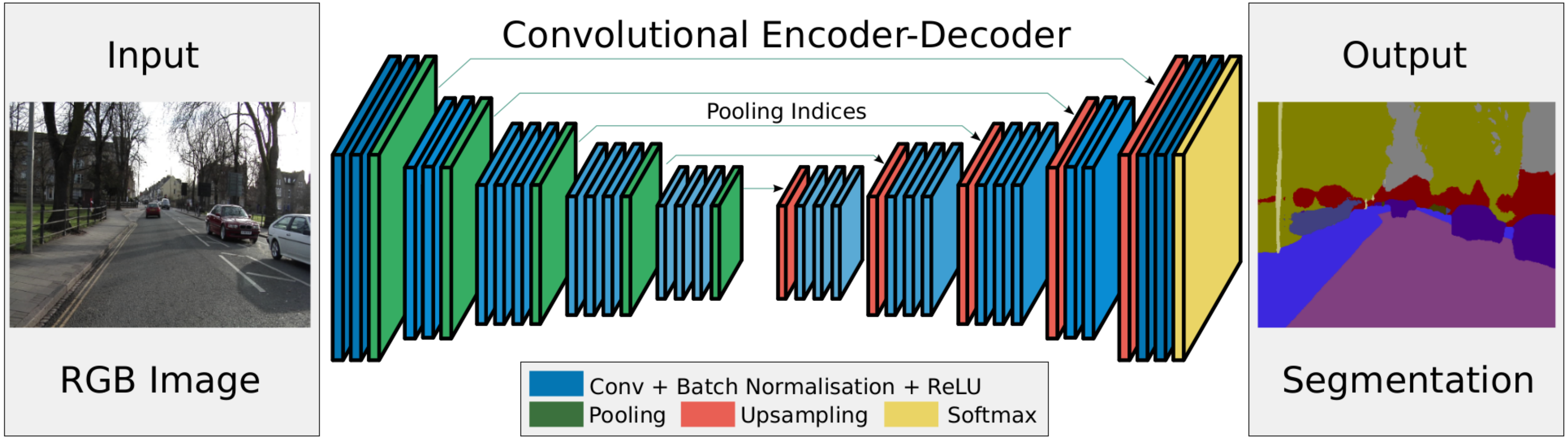
Object Detection

- Task: Identify objects, provide bounding boxes, and label them
- One strategy: Propose candidate bounding boxes, then classify each box (possibly as nothing)

Semantic Segmentation

- Task: Predict a class label for each pixel

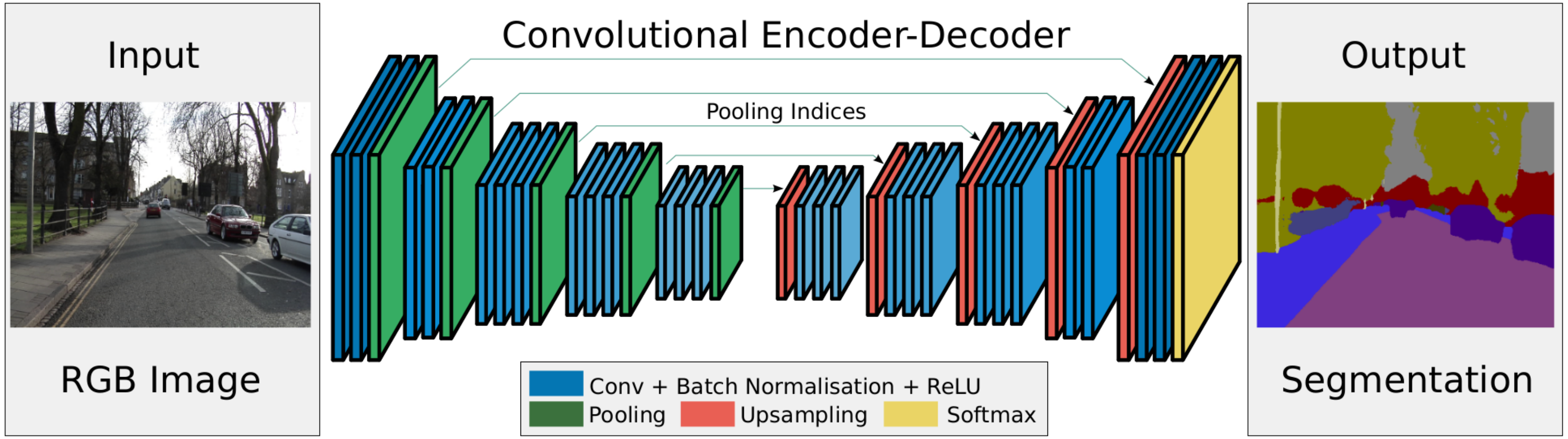
Semantic Segmentation



- One strategy: Encoder-Decoder ("U-net")
  - First do conv + ReLU + pooling as before
  - Then do upsampling + conv + ReLU to generate an output of original size

Image Generation

- Segmentation: "generates" a 2-D grid of predictions
  - This is almost like generating an image
- Can we use CNNs to generate new images?

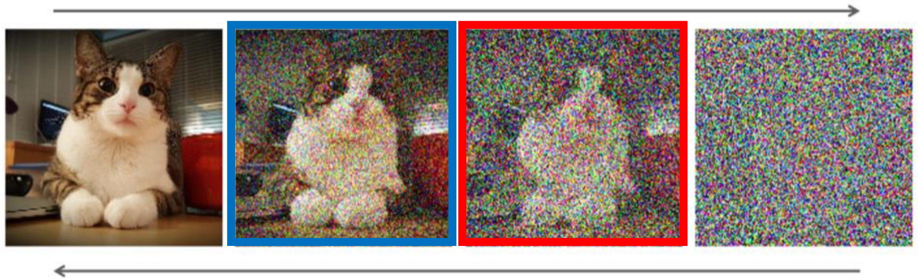


Diffusion Models

- Training: Add noise to good images, train neural network to undo the noise
  - **Input**: Noisy image
  - **Output**: Less noisy image
  - Architecture: Can also use U-Net
  - Objective: Per-pixel regression loss

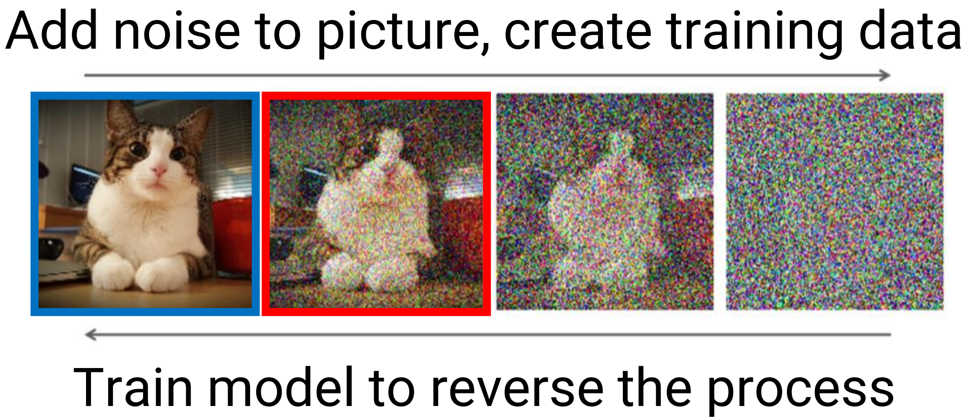
Diffusion Models

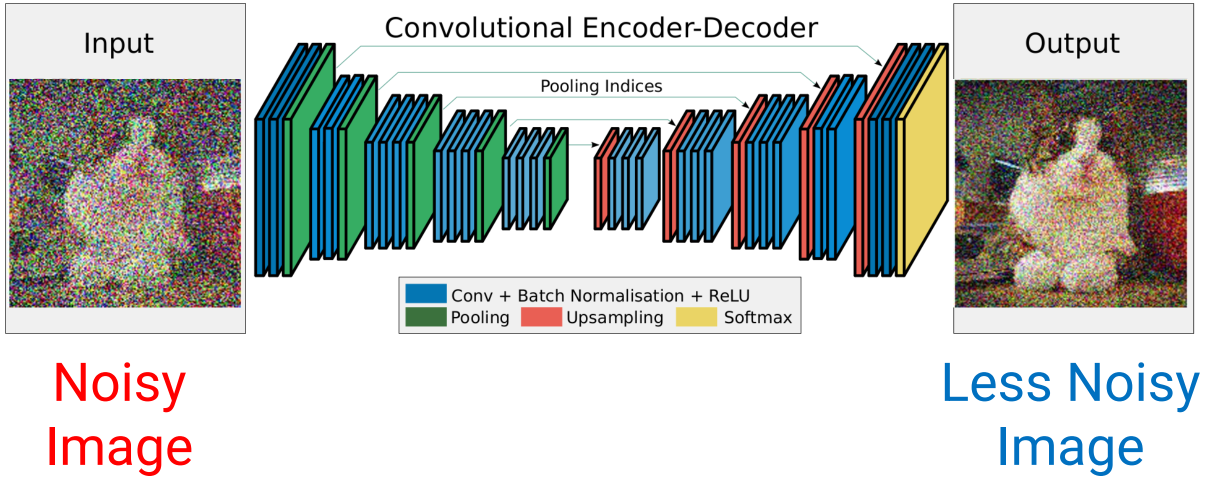
- Training: Add noise to good images, train neural network to undo the noise
  - **Input**: Noisy image
  - **Output**: Less noisy image
  - Architecture: Can also use U-Net
  - Objective: Per-pixel regression loss



Diffusion Models

- Training: Add noise to good images, train neural network to undo the noise
  - **Input**: Noisy image
  - **Output**: Less noisy image
  - Architecture: Can also use U-Net
  - Objective: Per-pixel regression loss

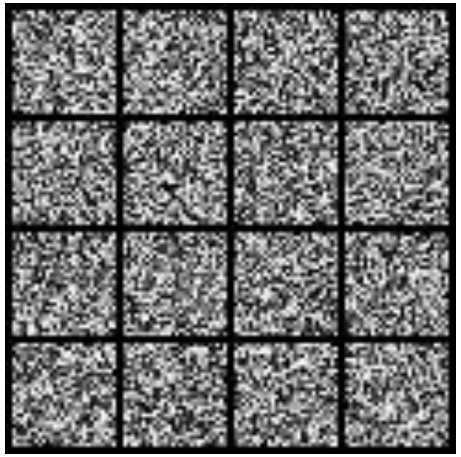




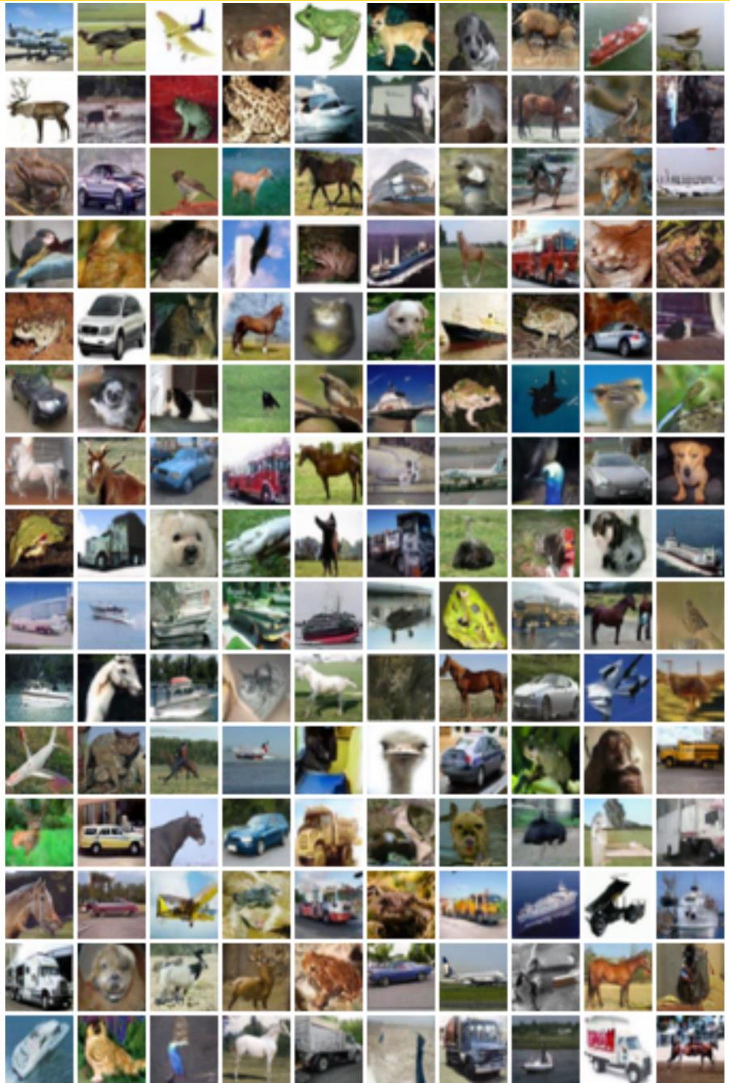
Diffusion Models

- Training: Add noise to good images, train neural network to undo the noise
  - **Input**: Noisy image
  - **Output**: Less noisy image
  - Architecture: Can also use U-Net
  - Objective: Per-pixel regression loss
- Test-time: Start from pure noise, apply the neural network many times to create an image!
- How to input a caption? More on this later…

Test time: Model converts noise to images over many iterations



Diffusion Model Generated Images



Denoising Diffusion Probabilistic Models. Jonathan Ho, Ajay Jain, and Pieter Abbeel. NeurIPS 2020.

Announcements

- HW1 Regrades: Open until next Tuesday, February 27
- HW2 Due Thursday, February 29
- Midterm exam Thursday, March 7
  - In-class, 80 minutes in SLH 100
  - Allowed one double-sided 8.5x11 sheet of notes
  - Practice Exams from past 2 semester will be released soon
- Section tomorrow: Sci-kit learn

Outline

- Computer vision tasks
- Word vectors
    - What do we want?
    - word2vec
    - Solving analogies
    - Bias in word vectors

Word vectors

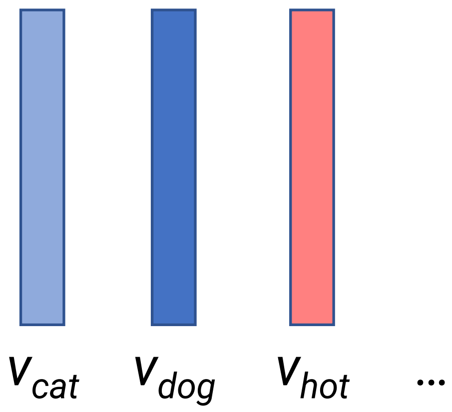
- Goal: For each word *w*, learn vector v_w that represents word's meaning
  - Similar words should have similar vectors
  - Different components of the vector may represent different properties of a word
- Why?
  - Neural networks take vectors as inputs. To feed them sentences, need to represent each word as a vector
  - Independently interesting to understand relationships between words

Word *w*	Vector v_w
A	[-0.4, 1.4, -1.2]
Aardvark	[2.2, -1.8, 0.6]
Airport	[0.7, 0.3, 3.1]
…	
Elephant	[2.1, -1.3, 0.3
…	
Zoo	[2.1, -1.4 3.2]

Related to animals? Is a place

Lexical Semantics

- Word vectors should capture *lexical semantics*
  - Lexical = word-level
  - Semantics = meaning
- What do we want to represent?
  - Synonymy (*car*/*automobile*) or antonymy (*cold*/*hot*)
  - Hypernymy/Hyponymy (*animal*/*dog*)
  - Similarity (*cat*/*dog*, *coffee*/*cup*, *waiter*/*menu*)
  - Various features
    - Sentiment (positive/negative)
    - Formality
    - All sorts of properties (Is a city? Is an action that a person can do?)



The Distributional Hypothesis

- You hear a new word, ***ongchoi***
  - ***Ongchoi*** *is delicious sauteed with garlic.*
  - ***Ongchoi*** *is superb over rice.*
  - *...**ongchoi** leaves with salty sauces...*
- Compare with similar contexts:
  - *...**spinach** sauteed with garlic over rice...*
  - *...**chard** stems and leaves are delicious...*
  - *...**collard greens** and other salty leafy greens*

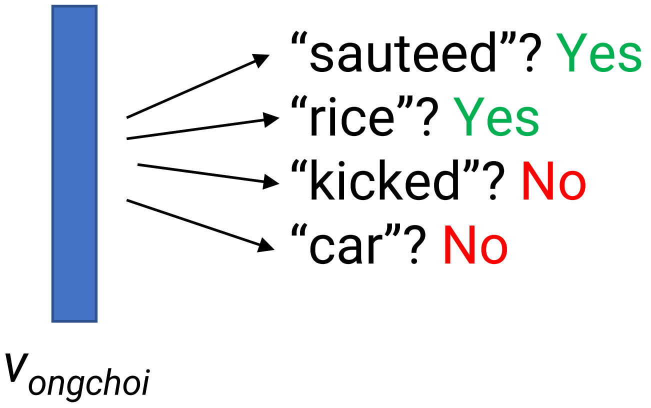
- Conclusion: **ongchoi** is probably a leafy green similar to spinach, chard, and collard greens
- Distributional Hypothesis: Words appearing in similar contexts have similar meanings!
- Firth 1957: "You Shall Know a Word by the Company It Keeps"

Outline

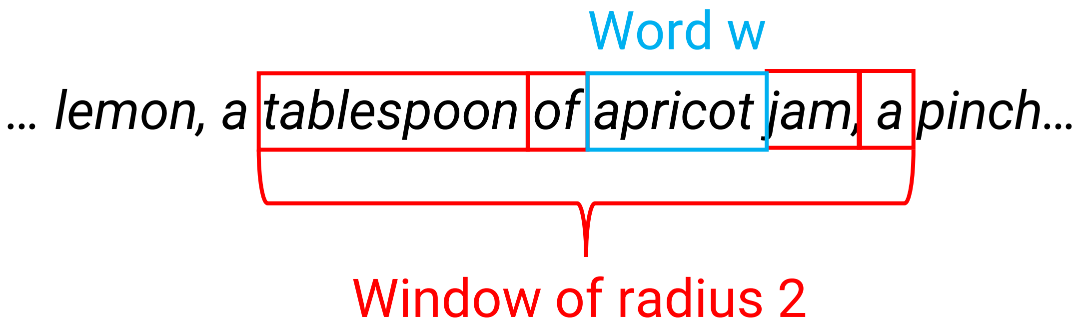
- Computer vision tasks
- Word vectors
  - What do we want?
  - word2vec
  - Solving analogies
  - Bias in word vectors

Word vectors as a learning problem

- Want to learn vector v_w for each word *w*
- What makes a vector good?
- Idea: v_w should help you predict which words co-occur with w
  - Captures **distribution** of context words for *w*
  - Think of it as N binary classification problems, where N is size of vocabulary



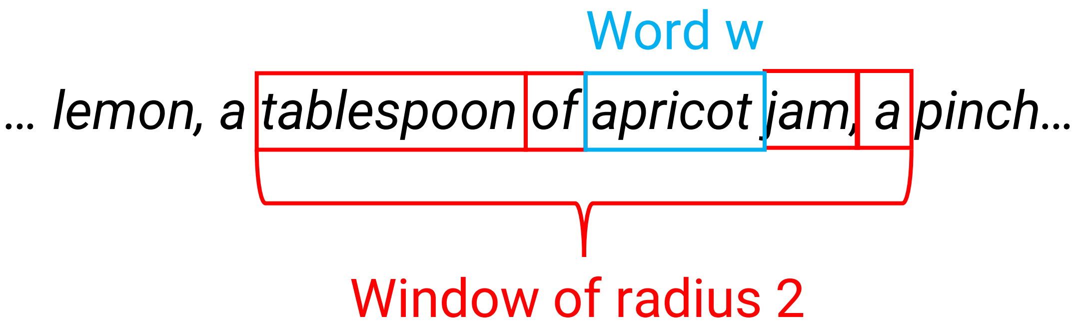
Creating a dataset



Word *w* ("input")	Context *w'* ("task")	y (label)
apricot	tablespoon	+1
apricot	of	+1
apricot	jam	+1
apricot	a	+1

- Given: Raw dataset of text (unsupervised)
- We will create N "fake" supervised learning problems!
  - We don't really care about these supervised learning problems
  - We just care that we learn good vectors
- Task i: Did word *w* co-occur with the i-th word?
  - Positive examples: Real co-occurrences within sliding window
  - Negative examples: Random samples

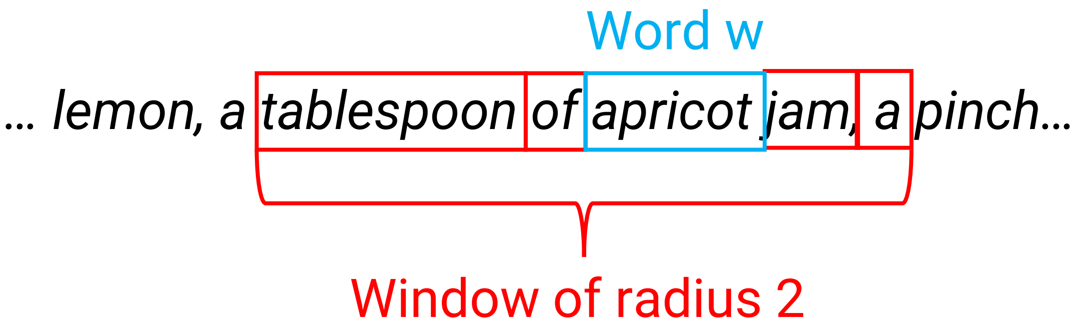
Creating a dataset



Word *w* (“input”)	Context *w’* (“task”)	y (label)
apricot	tablespoon	+1
apricot	of	+1
apricot	jam	+1
apricot	a	+1
apricot	seven	-1
apricot	forever	-1
apricot	dear	-1
apricot	if	-1

- Given: Raw dataset of text (unsupervised)
- We will create N “fake” supervised learning problems!
  - We don’t really care about these supervised learning problems
  - We just care that we learn good vectors
- Task i: Did word *w* co-occur with the i-th word?
  - Positive examples: Real co-occurrences within sliding window
  - Negative examples: Random samples

How to sample negatives?



Word *w* ("input")	Context *w'* ("task")	y (label)
apricot	tablespoon	+1
apricot	of	+1
apricot	jam	+1
apricot	a	+1
apricot	seven	-1
apricot	forever	-1
apricot	dear	-1
apricot	if	-1

- Choose a fixed ratio of negative:positive (e.g. 2)
- Baseline: Sample according to frequency of word p(w) in the data
  - Not ideal because very common words ("the") get sampled a lot
- Improvement: Sample according to α-weighted frequency

$$p_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w' \in V} \text{count}(w')^\alpha}$$

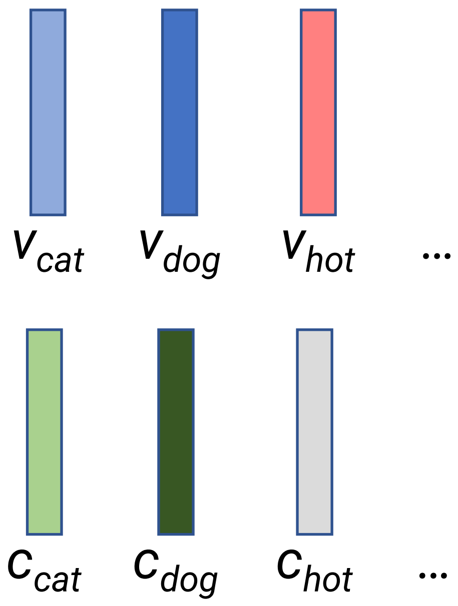
  - For α < 1, high-frequency words get down-weighted
  - Typically choose around α=.75

word2vec model

- Parameters (all of dimension d):
  - Word vector v_w for each word ("features"—the actual word vectors)
  - Context vector c_w for each word ("classifier weights" for task corresponding to w as context)
- Goal: v_w can be used by **linear classifier** to do **any** of the N "was this a context word" tasks
- Objective looks just like logistic regression:

$$L(v, c) = \sum_{(w, w', y)} -\log \sigma(y \cdot v_w^\top c_{w'})$$

word context "features" for word "weight" for context



Training word2vec

- Strategy: Gradient descent
- Gradient updates essentially same as logistic regression
  - Gradient w.r.t. *c* holds *v* fixed, so it's like *v* are fixed features

$$\nabla_{c_u} L(v,c) = \sum_{(w,w',y):w'=u} -\sigma(y \cdot v_w^\top c_u) \cdot y \cdot v_w$$

Examples where w' = u

Same as logistic regression where v_w is the input x

  - Gradient w.r.t. *v* is symmetrical

$$\nabla_{v_u} L(v,c) = \sum_{(w,w',y):w=u} -\sigma(y \cdot v_u^\top c_{w'}) \cdot y \cdot c_{w'}$$

Examples where w = u

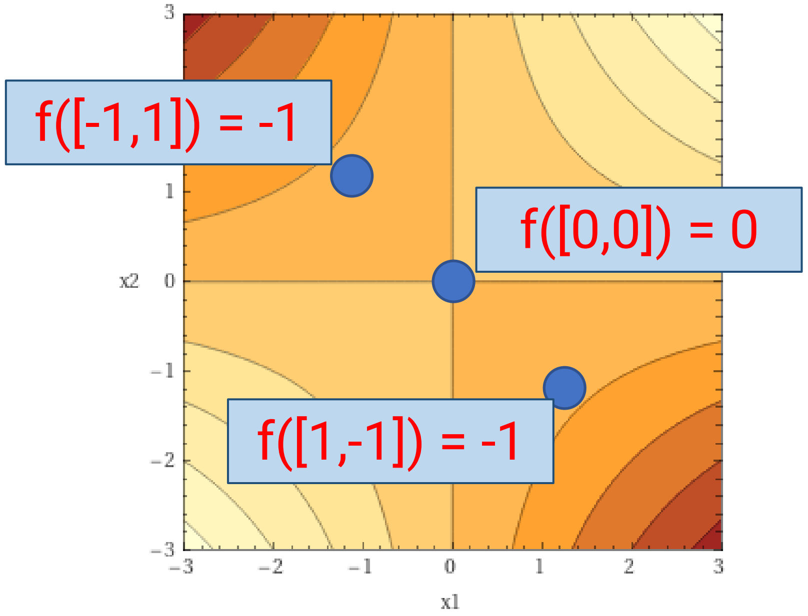
Same as logistic regression where $c_{w'}$ is the input x

Is this a convex problem?

- Looks a lot like logistic regression…
- But it's not convex!
- Why?
  - In logistic regression, we only optimize w.r.t. weights, features are constant
  - Now we optimize both at the same time!
- Fact to remember: f(x) = $x_1 * x_2$ is not convex
  - Consider points [-1, 1] and [1, -1]
  - f(x) = -1 at both points
  - But at the midpoint [0, 0], f(x) = 0
- Corollary: We need to randomly initialize
  - Must break symmetry, as in neural networks

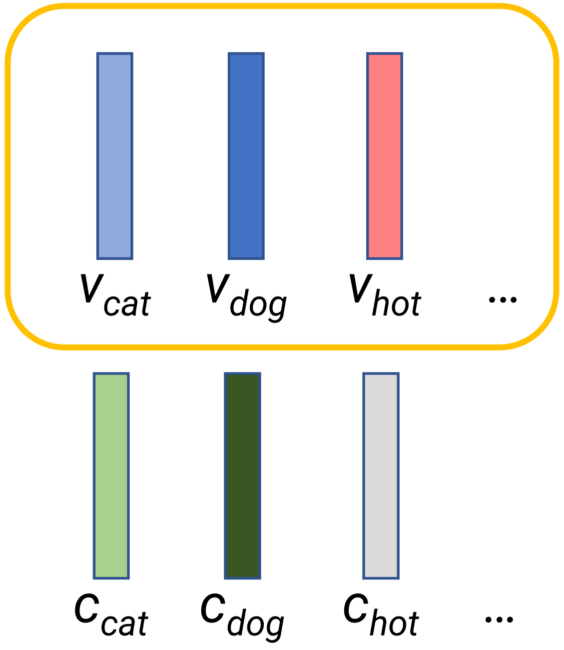
$$L(v, c) = \sum_{(w,w',y)} -\log \sigma(y \cdot v_w^\top c_{w'})$$

Both are optimization variables



word2vec overview

- Acquire large unsupervised text corpus
- Create positive examples for every word by using sliding window
- Create negative examples by randomly sampling context word from weighted word frequency
- Randomly initialize all *v* and *c* vectors
- Train on logistic regression-like loss with gradient descent
- Return *v* vectors
  - c vectors not needed—just helpers



Outline

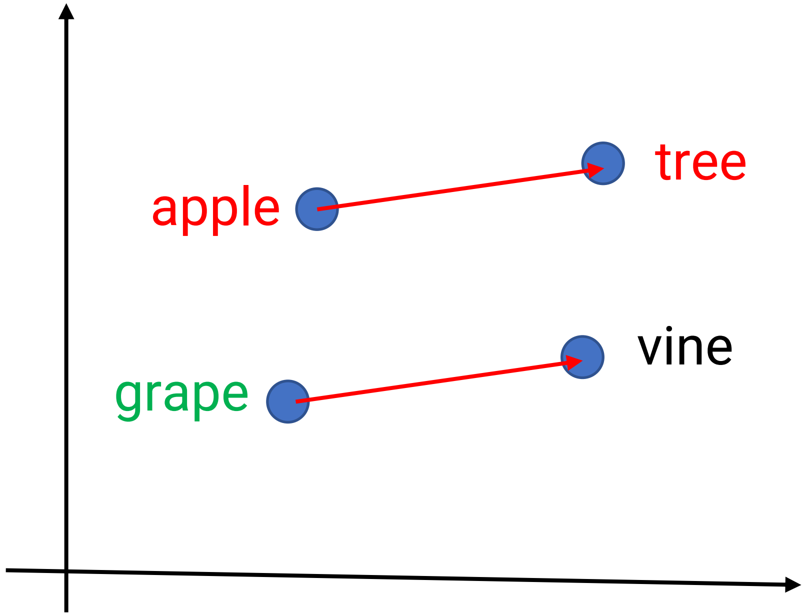
- Computer vision tasks
- Word vectors
  - What do we want?
  - word2vec
  - Solving analogies
  - Bias in word vectors

Analogies in vector space

- *Apple is to tree as grape is to…*
- In vector space, resembles a parallelogram
  - Same relationship between apple and tree holds between grape and vine
- $v_{vine} \approx \underbrace{v_{tree} - v_{apple}}_{\text{Represents the “grows on” relation}} + \underbrace{v_{grape}}_{\text{Query word}}$

Represents the “grows on” relation

Query word

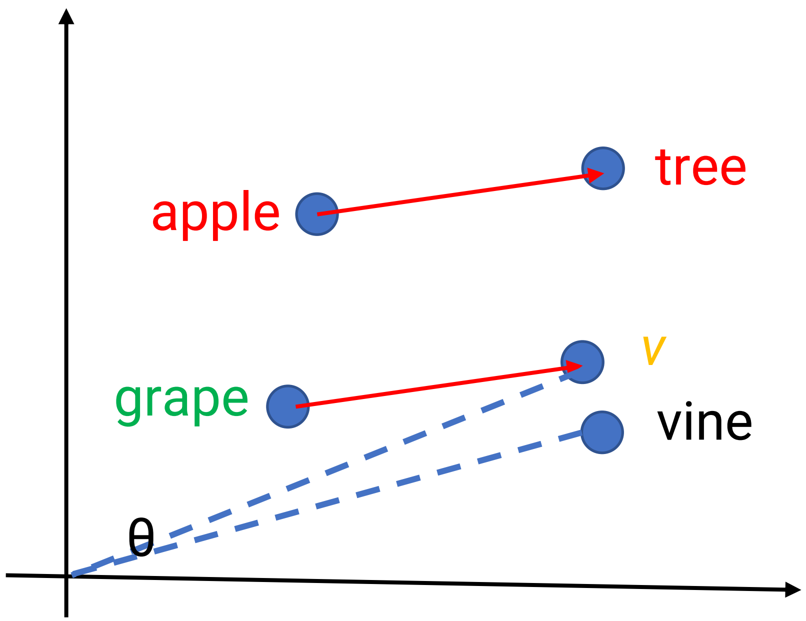


Answering analogy queries

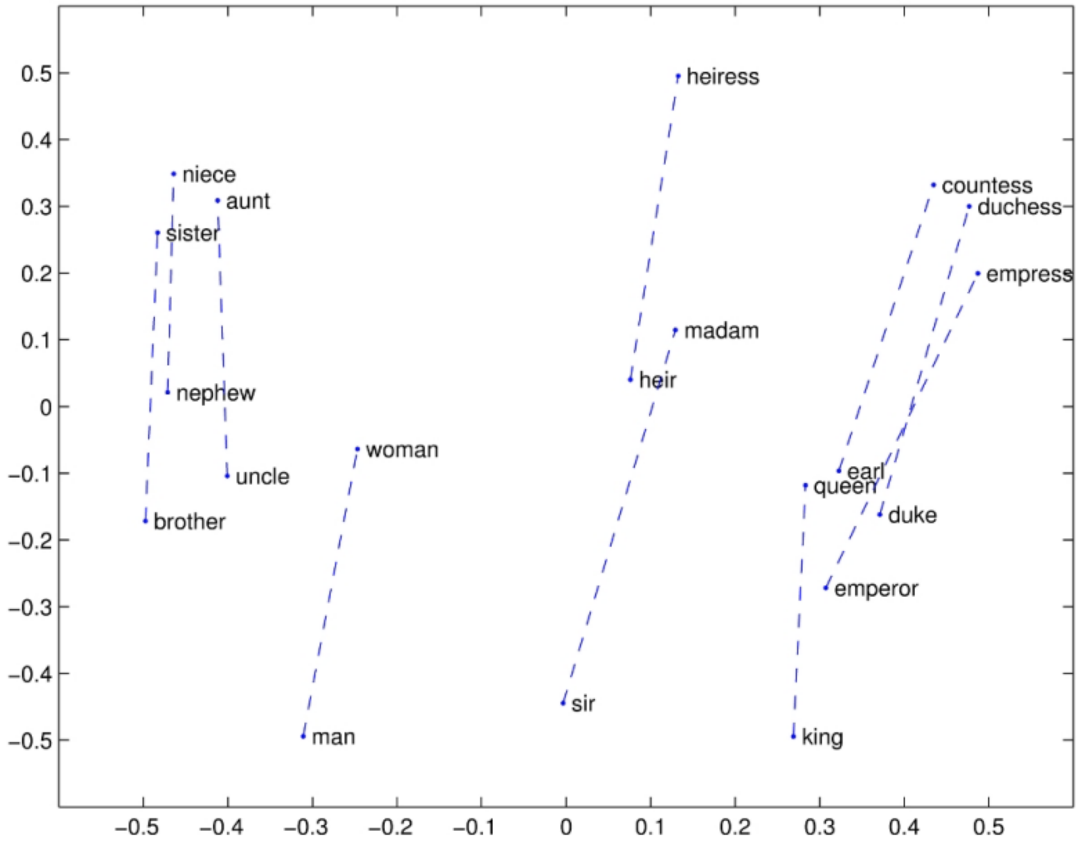
- Compute $v = v_{tree} - v_{apple} + v_{grape}$
- Find word w in vocabulary whose v_w is most similar to v
    - Common choice: Cosine similarity

$$\mathrm{cossim}(x, y) = \frac{x^\top y}{\|x\| \|y\|}$$

(= cosine of angle between x and y)
    - Typically need to exclude words very similar to the query word (e.g. "*grapes*")

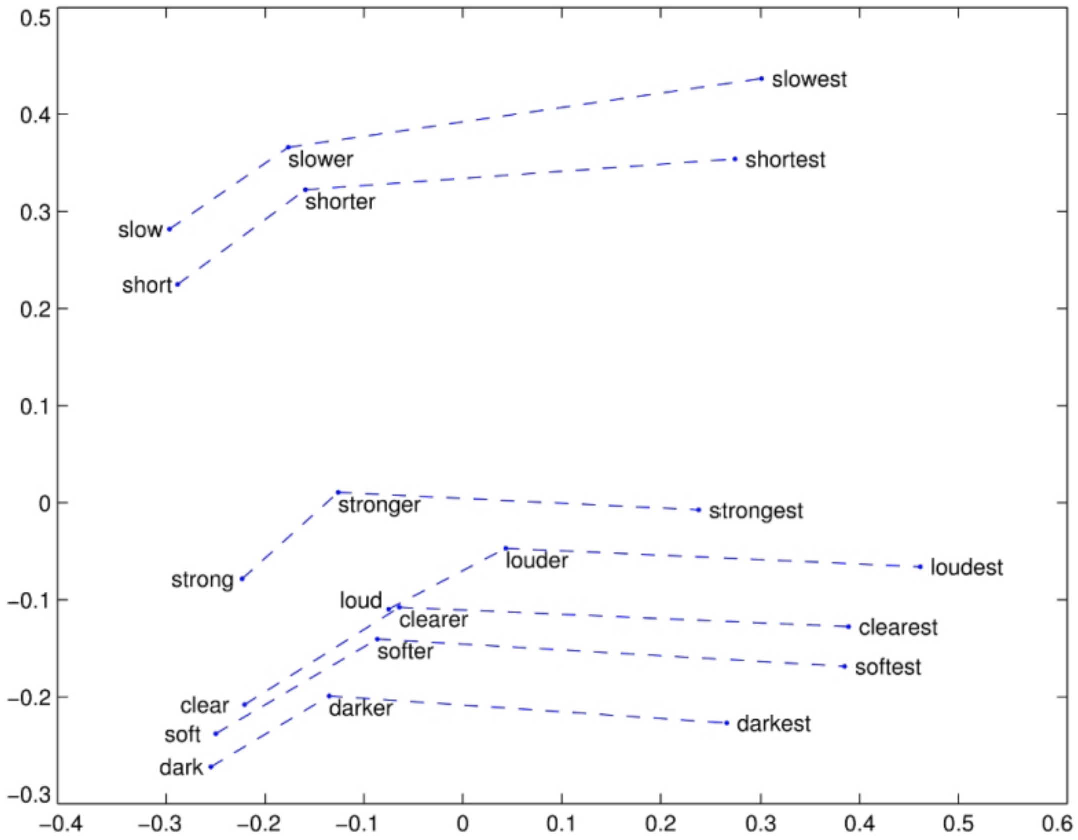


Visualizing Analogies



- Figure: *Dimensionality reduction* to 2D, then plot words with known relationship
  - We'll talk about dimensionality reduction later!
- Roughly same difference between male/female versions of the same word

Visualizing Analogies



- Figure: *Dimensionality reduction* to 2D, then plot words with known relationship
  - We'll talk about dimensionality reduction later!
- Roughly same difference between base, comparative, and superlative forms of adjectives

Outline

- Computer vision tasks
- Word vectors
  - What do we want?
  - word2vec
  - Solving analogies
  - Bias in word vectors

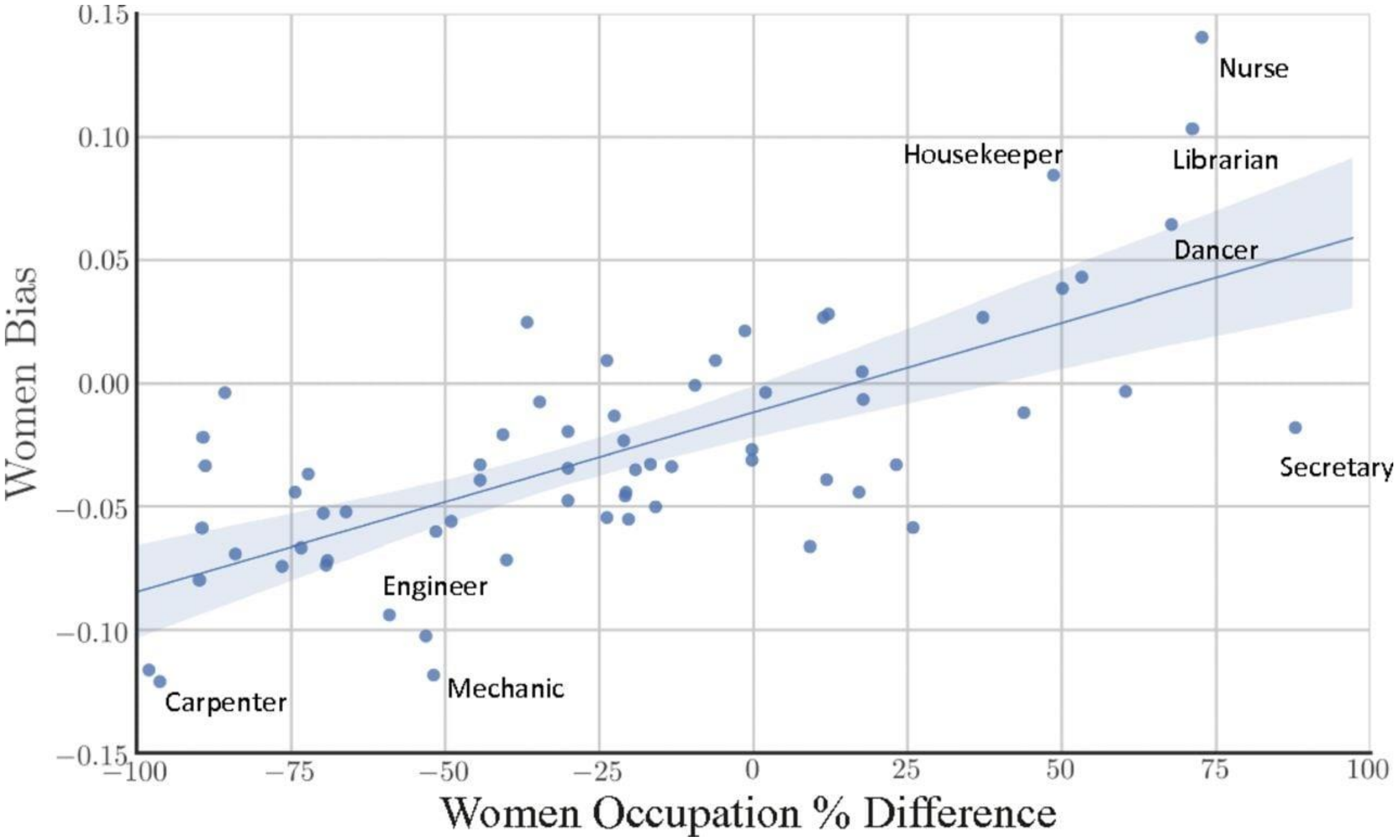
Machine learning is a tornado

- …it picks up everything in its path
- Data has all sorts of associations we may not want to model

What word associations are out there?

- What is *programmer* – *man* + *woman*?
  - According to word vectors trained on news data, it’s *homemaker*
  - Existing data has tons of correlations between occupation and gender
- word2vec doesn’t know what is a semantic relationship and what is a historical correlation
  - “*queen*” is more related to “*she*” than “*he*” semantically
  - “*nurse*” may co-occur more with “*she*” than “*he*” in available data but not a semantic relationship!

Word vectors quantify gender stereotypes



- X-axis: Real percentage difference in workforce between women & men
- Y-axis: Embedding bias = difference of distance from male-related words and female-related words
- Strong correlation!

Conclusion

- Distributional hypothesis: Words that appear in similar contexts have similar meanings
- word2vec: Learn vectors by inventing a prediction problem (did this word-context pair really occur in the text?)
- Vector arithmetic lets us complete relations
- Vectors capture both lexical semantics and historical biases
- Next time: Word vectors as a component of neural networks for processing text