Text Classification

CS 288 Spring 2026

UC Berkeley

cal-cs288.github.io/sp26

Berkeley EECS BAIR

Today's Question:

How to Train a Text Classifier?

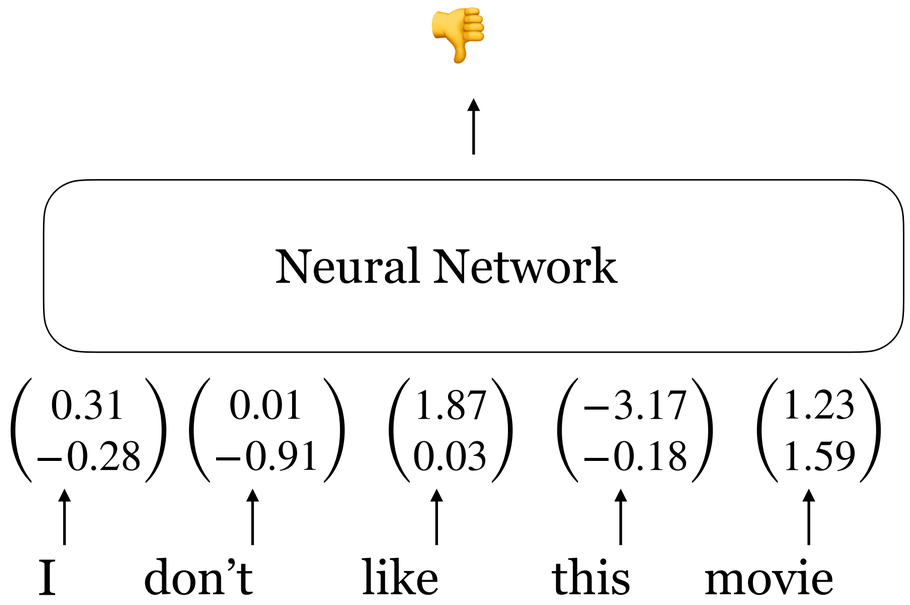
- Last lecture, we learned about word representations.
- Once we have word representations, how does a model combine them to make decisions?
- We'll look into this through a task of **text classification**

Another theme:

Word embeddings to neural networks

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

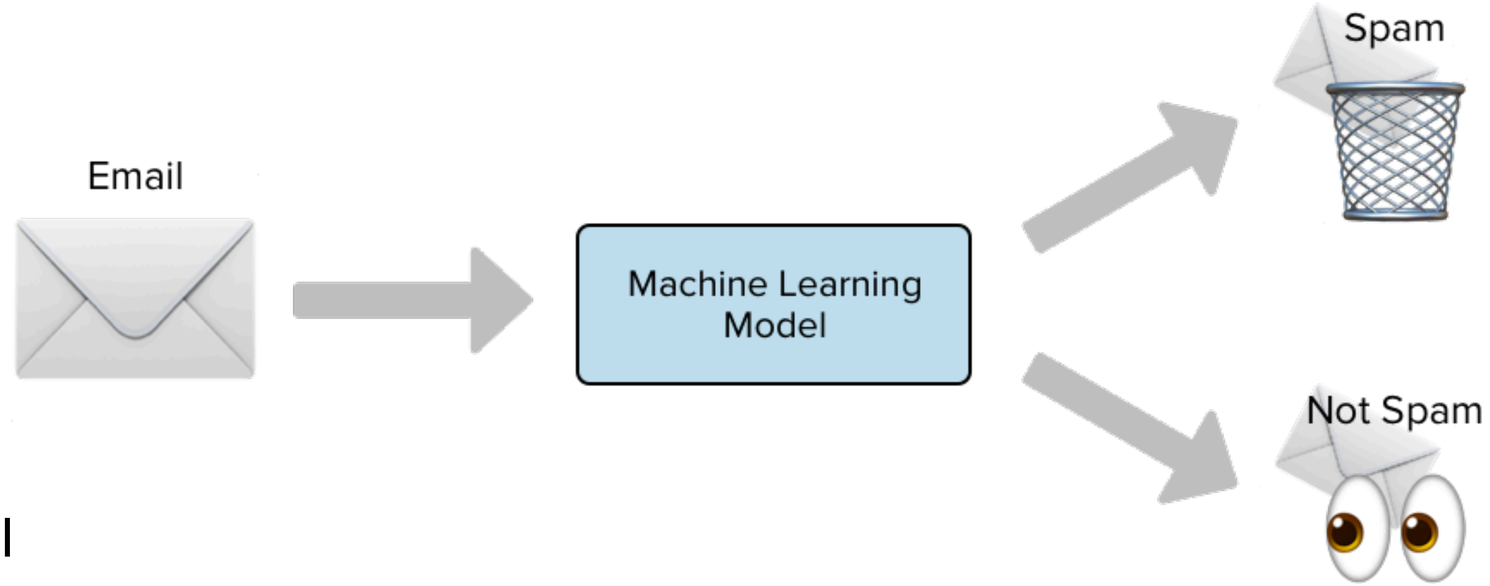
$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$



Text Classification

- One of the most basic NLP tasks
- Input: a text
- Output: a label from a predefined set
- Learning problem: estimate the parameters of a function that maps a text to its label

Example 1: Spam vs. Ham



- Input: email
- Output: spam/ham
- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham”
  - Note: someone has to hand label all this data!
  - Goal: learn to predict labels of new, future emails

Example 2: Topic Classification

Goal: classify documents into broad semantic topics (e.g., politics, sports, business, technology)

> Obama is hoping to rally support for his $825 billion stimulus package on the eve of a crucial House vote. Republicans have expressed reservations about the proposal, calling for more tax cuts and less spending. GOP

> California will open the 2009 season at home against Maryland Sept. 5 and will play a total of six games in Memorial Stadium in the final football schedule announced by the Pacific-10 Conference Friday.

- Which one is the POLITICS document? Did this require a deep analysis?

Example 3: Sentiment Analysis

Goal: detect the overall sentiment of the text

This movie was great! Will watch again 

Not bad at all! but not a masterpiece 

Could never enjoy, even with closed eyes 

- Did this require more reasoning compared to categorization?

Example 3: Sentiment Analysis

Goal: detect the overall sentiment of the text

This movie was great! Will watch again 

Not bad at all! but not a masterpiece 

Could never enjoy, even with closed eyes 

- Did this require more reasoning compared to categorization?
- Just spotting individual words is not enough

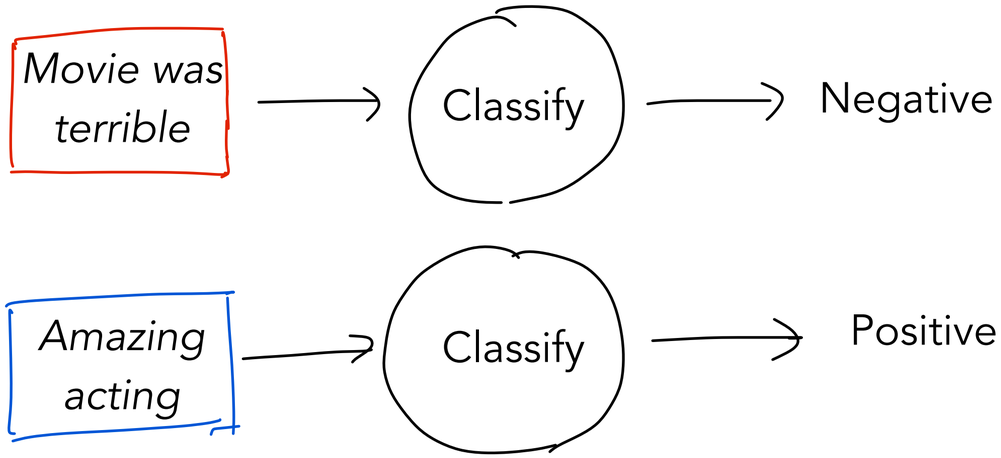
Text classification

Inputs:

- A document d
- A set of classes C (m classes)

Output:

- Predicted class $c \in C$ for document d



Rule-based text classification

IF there exists word w in document d such that w in [good, great, extra-ordinary, …],
THEN output Positive
IF email address ends in [ithelpdesk.com, makemoney.com, spinthewheel.com, …]
THEN output SPAM

+ Can be very accurate (if rules carefully refined by expert, especially in narrow domains)

+ Interpretable

- Rules may be hard to define (and some even unknown to us!)

- Expensive

- Not easily generalizable

VADER-Sentiment-Analysis

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is *specifically attuned to sentiments expressed in social media*. It is fully open-sourced under the [MIT License]

https://github.com/cjhutto/vaderSentiment

Supervised training: Let's use statistics!

Let the machine figure out the best patterns using data

Inputs:

- Set of m classes C
- Set of n 'labeled' documents: $\{(d_1, c_1), (d_2, c_2), \ldots, (d_n, c_n)\}$, $d_i \in \mathscr{D}, c_i \in C$

Output:

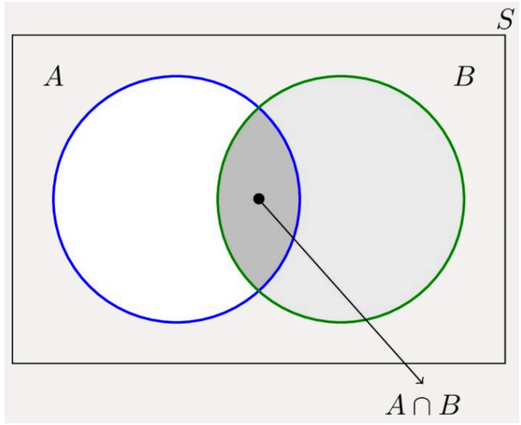
- Trained classifier, $F : \mathscr{D} \rightarrow C$

Key questions:

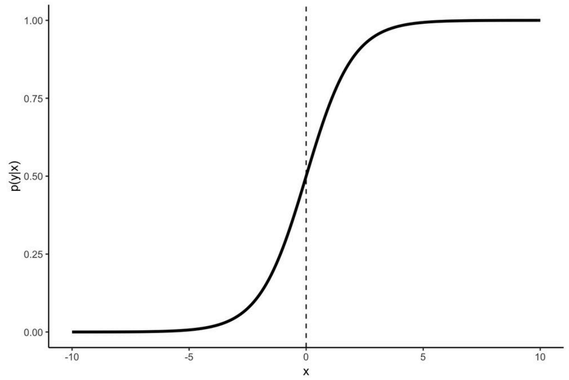
a) What is the form of F?

b) How do we learn F?

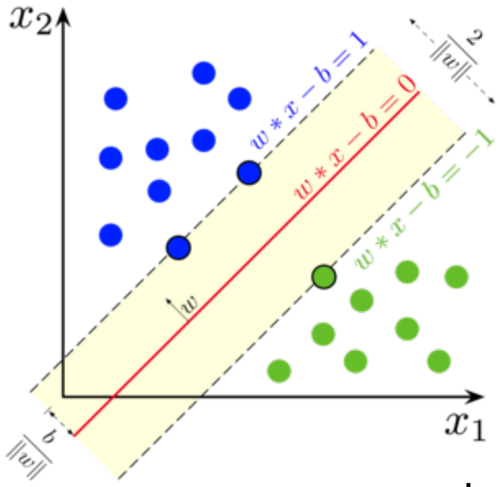
Types of supervised classifiers



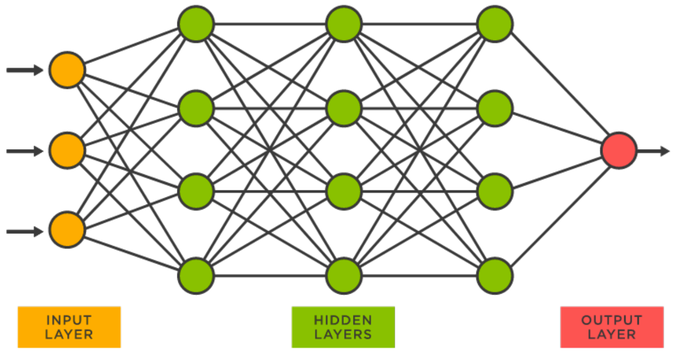
Naive Bayes



Logistic regression



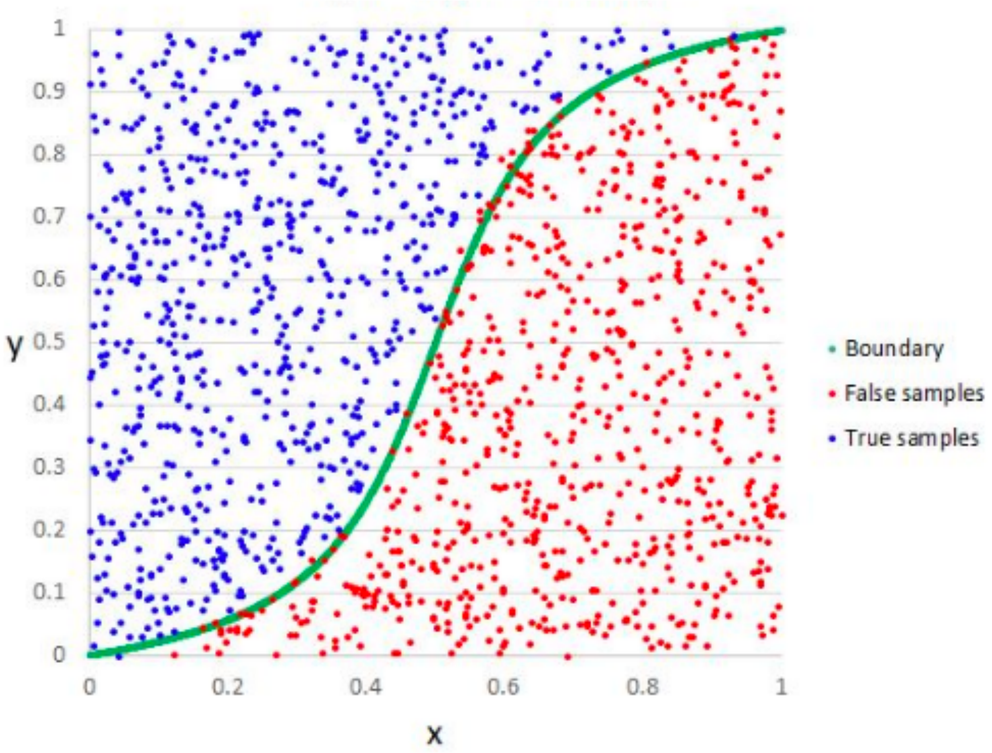
Support vector machines



neural networks

Logistic regression

Logistic regression



- Powerful supervised model
- Baseline approach for many NLP tasks
- It's not a regression model (!)
- Binary (two classes) or multinomial (>2 classes)
- Foundation of neural networks

https://machine-learning.paperspace.com/wiki/logistic-regression

Generative vs. discriminative models

- Naive Bayes is a ***generative*** model
- Logistic regression is a ***discriminative*** model

$$\text{argmax}_{c \in C} P(c \mid d)$$

Suppose we're distinguishing cat from dog images

imagenet

imagenet

Generative classifiers

- Build a model of what is in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image - how cat-y is this image?

- Also build a model for dog images

- Now given a new image:
  - **Run both models and see which one fits better**

Discriminative classifiers

Just try to distinguish dogs from cats

Oh look, dogs have collars!
Let's ignore everything else

Overall process: Discriminative classifiers

Input: a set of labeled documents $\{(d_i, y_i)\}_{i=1}^{n}$

$y_i = 0$ or 1 (binary)

$y_i = 1, \ldots, m$ (multinomial)

- Training phrase:
  1. Convert d_i into a **vector representation** x_i
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
  3. **Loss function** for learning e.g., cross-entropy
  4. Optimization **algorithm** to minimize loss function e.g., stochastic gradient descent
- Test phase: Apply **parameters** to predict class given a new input x (feature representation of testing document d)

Overall process: Discriminative classifiers

Input: a set of labeled documents $\{(d_i, y_i)\}_{i=1}^{n}$

$y_i = 0$ or 1 (binary)

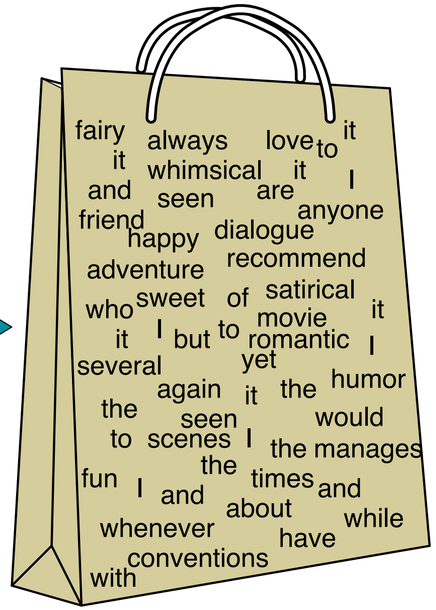
$y_i = 1, \ldots, m$ (multinomial)

- Training phrase:
  1. Convert d_i into a **vector representation** x_i
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
  3. **Loss function** for learning e.g., cross-entropy
  4. Optimization **algorithm** to minimize loss function e.g., stochastic gradient descent
- Test phase: Apply **parameters** to predict class given a new input x (feature representation of testing document d)

Naive option: Bag of words

Bag of words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1

$$\mathbf{x} = [x_1, x_2, \ldots, x_k]$$

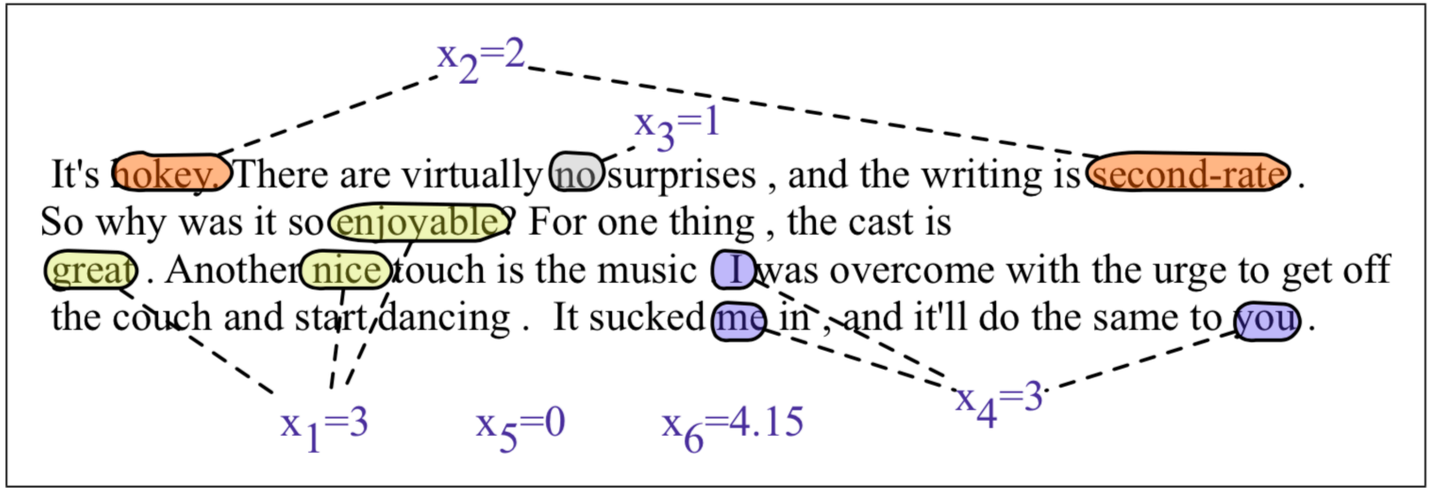
x_1: how many times the word “love” appears in d

In BoW representations, $k = |V|$ and the vector has many zeros.

Hand-crafted features

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Example: Sentence classification



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Remember that the values make up the feature vector!

Overall process: Discriminative classifiers

Input: a set of labeled documents $\{(d_i, y_i)\}_{i=1}^{n}$

$y_i = 0$ or 1 (binary)

$y_i = 1, \ldots, m$ (multinomial)

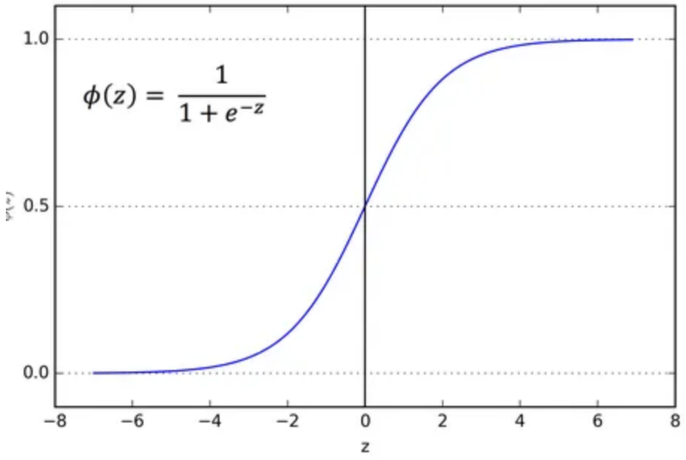
- Training phrase:
  1. Convert d_i into a **vector representation** x_i
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
  3. **Loss function** for learning e.g., cross-entropy
  4. Optimization **algorithm** to minimize loss function e.g., stochastic gradient descent
- Test phase: Apply **parameters** to predict class given a new input x (feature representation of testing document d)

Classification function

- *Given*: Input feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_k]$
- *Output*: $P(y = 1 \mid \mathbf{x})$ and $P(y = 0 \mid \mathbf{x})$ *(binary classification)*

Weight vector $\mathbf{w} = [w_1, w_2, \ldots, w_k]$ bias

- Given input features $\mathbf{x}$: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Therefore, $\hat{y} = P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$
- Decision boundary: $= \begin{cases} 1 & \text{if } \hat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases}$



Example: Sentence classification

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if ``no'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if ``!'' } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $\mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Overall process: Discriminative classifiers

Input: a set of labeled documents $\{(d_i, y_i)\}_{i=1}^{n}$

$y_i = 0$ or 1 (binary)

$y_i = 1, \ldots, m$ (multinomial)

- Training phrase:
  1. Convert d_i into a **vector representation** x_i
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
  3. **Loss function** for learning e.g., cross-entropy
  4. Optimization **algorithm** to minimize loss function e.g., stochastic gradient descent
- Test phase: Apply **parameters** to predict class given a new input x (feature representation of testing document d)

Loss function

- For n data points (x_i, y_i), y_i = 0 or 1, $\hat{y}_i = P(y_i = 1 \mid x_i)$
- Classifier probability: $\Pi_{i=1}^{n} P(y_i \mid x_i) = \Pi_{i=1}^{n} \hat{y}_i^{y_i}(1 - \hat{y}_i)^{1-y_i}$
- Loss: $-\log \prod_{i=1}^{n} P(y_i \mid x_i) = -\sum_{i=1}^{n} \log P(y_i \mid x_i)$

$$L_{CE} = -\sum_{i=1}^{n} [y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$$

Properties of CE loss

$$L_{CE} = -\sum_{i=1}^{n} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

What values can this loss take?

A) 0 to ∞ B) $-\infty$ to ∞ C) $-\infty$ to 0 D) 1 to ∞

- The answer is A) - Ranges from 0 (perfect predictions) to ∞

Overall process: Discriminative classifiers

Input: a set of labeled documents $\{(d_i, y_i)\}_{i=1}^{n}$

$y_i = 0$ or 1 (binary)

$y_i = 1, \ldots, m$ (multinomial)

- Training phrase:
  1. Convert d_i into a **vector representation** x_i
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
  3. **Loss function** for learning e.g., cross-entropy
  4. Optimization **algorithm** to minimize loss function e.g., stochastic gradient descent
- Test phase: Apply **parameters** to predict class given a new input x (feature representation of testing document d)

Optimization

- We have our **classification function** and **loss function** - how do we find the best w and b?
- Cross entropy loss for logistic regression is **convex** (i.e. has only one global minimum) so gradient descent is guaranteed to find the minimum.
- Stochastic gradient descent: Use a mini-batch of training examples!

Regularization

Training objective: $\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(y_i \mid x_i)$

This might fit the training set too well! (including noisy features), and lead to poor generalization to the unseen test set — **Overfitting**

L2 regularization:

$$\hat{\theta} = \arg\max_{\theta} \left[\sum_{i=1}^{n} \log P(y_i \mid x_i) \; - \; \alpha \sum_{j=1}^{d} \theta_j^2 \right]$$

Multinomial logistic regression

- What if we have more than 2 classes?
- Need to model $P(y = c \,|\, x) \quad \forall c \in \{1, \ldots, m\}$
- Recall: Binary logistic regression

$$P(y = 1 \,|\, x) = \sigma(z) \text{ where } z = \mathbf{w} \cdot \mathbf{x} + b$$

$$P(y = 0 \,|\, x) = 1 - P(y = 1 \,|\, x)$$

- Multinomial logistic regression

$$P(y = c \,|\, x) = \frac{e^{z_c}}{\sum_{j=1}^{m} e^{z_j}} \text{ where } z_i = \mathbf{w}_i \cdot \mathbf{x} + b_i$$

- Multinomial CE loss

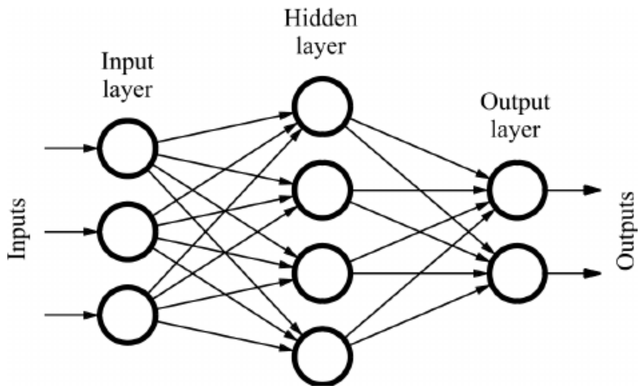
$$L_{CE}(\hat{y}, y) = -\sum_{c=1}^{m} 1\{y = c\} \log P(y = c \,|\, x)$$

Multilayer Perceptron (MLP)
(Neural Networks!)

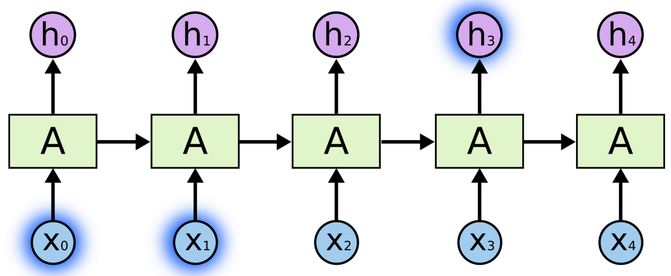
A little about Neural Networks for NLP

Neural networks in NLP

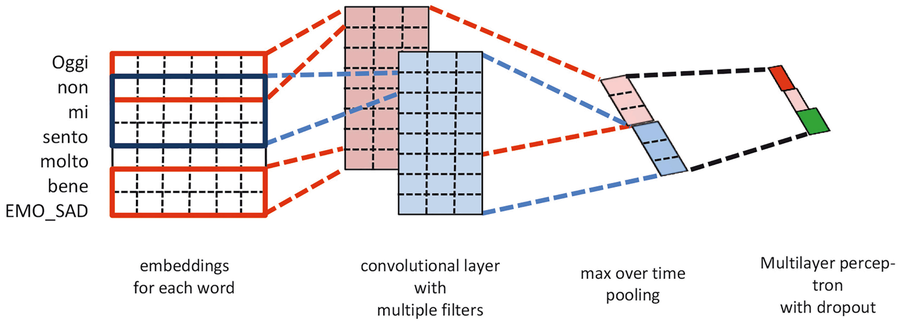
Feed-forward NNs



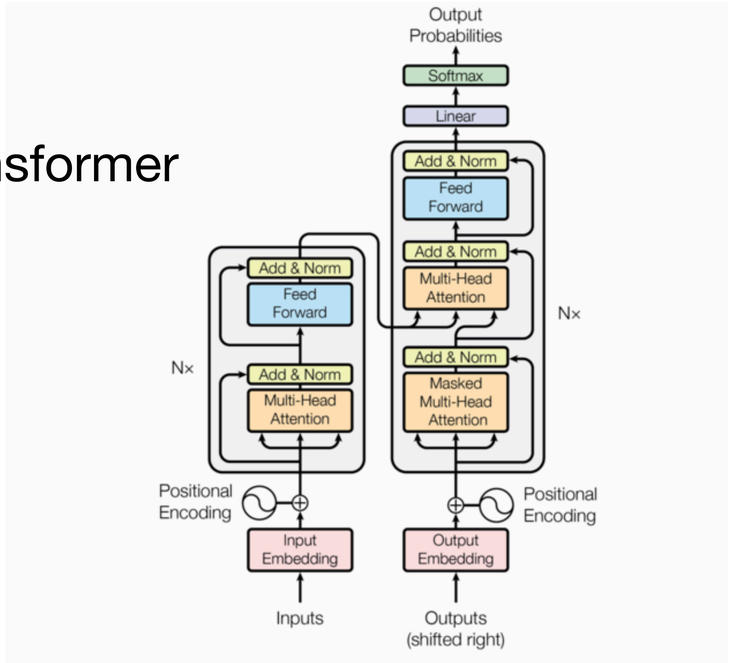
Recurrent NNs



Convolutional NNs

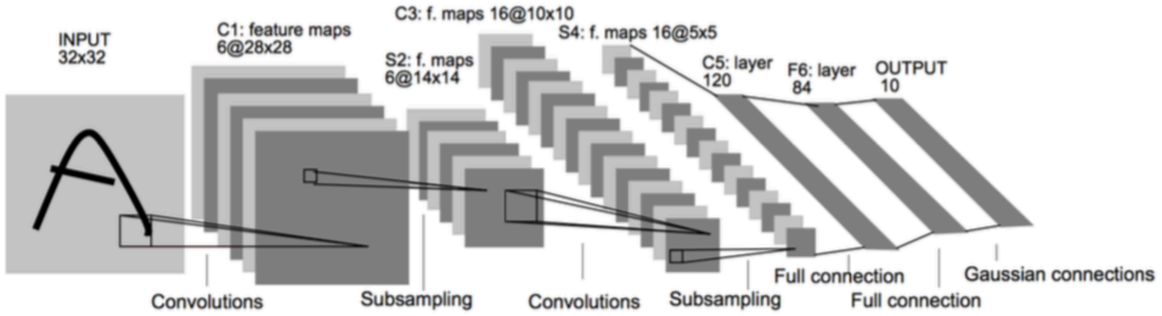


Transformer

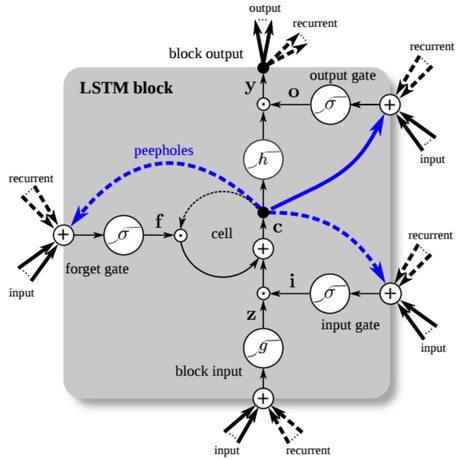


NN "dark ages"

- Neural network algorithms date from the 80s
- ConvNets: applied to MNIST by LeCun in 1998

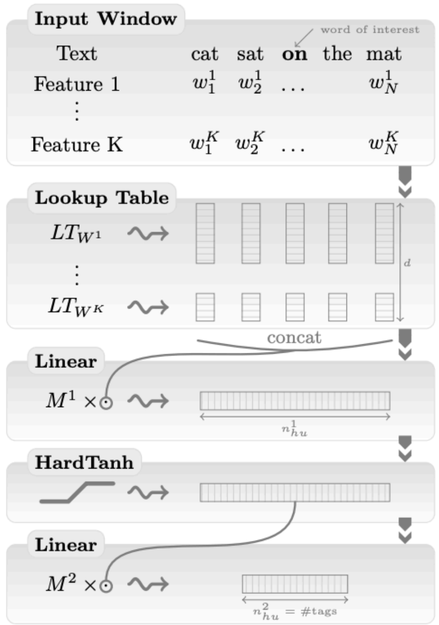


- Long Short-term Memory Networks (LSTMs): Hochreiter and Schmidhuber 1997
- Henderson 2003: neural shift-reduce parser, not SOTA



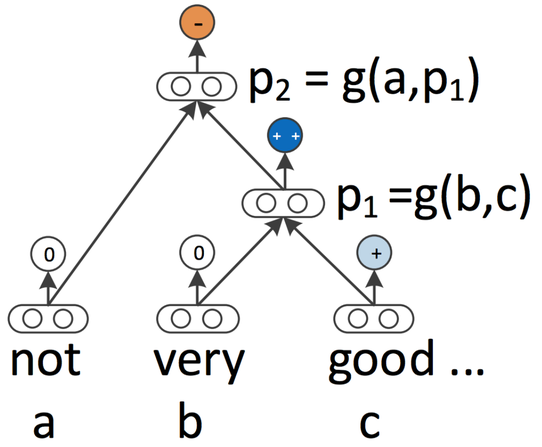
Slide credit: Greg Durrett

2008—2013: A glimmer of light



- Collobert and Weston 2011: "**NLP (almost) from Scratch**"
  - Feedforward NNs can replace "feature engineering"
  - 2008 version was marred by bad experiments, claimed SOTA but wasn't, 2011 version tied SOTA

- Krizhevskey et al, 2012: AlexNet for ImageNet Classification
- Socher 2011-2014: tree-structured RNNs working okay



Slide credit: Greg Durrett

2014: Stuff starts working

- Kim (2014) + Kalchbrenner et al, 2014: sentence classification
  - ConvNets work for NLP!
- Sutskever et al, 2014: sequence-to-sequence for neural MT
  - LSTMs work for NLP!
- Chen and Manning 2014: dependency parsing
  - Even feedforward networks work well for NLP!

Convolutional Neural Networks for Sentence Classification

Yoon Kim
New York University
yhk255@nyu.edu

A Fast and Accurate Dependency Parser using Neural Networks

Danqi Chen
Computer Science Department
Stanford University
danqi@cs.stanford.edu

Christopher D. Manning
Computer Science Department
Stanford University
manning@stanford.edu

- 2015: explosion of neural networks for everything under the sun
- 2018-2019: NLP has entered the era of pre-trained models (ELMo, GPT, BERT)
- 2020+: the emergence of large language models (GPT-3, ChatGPT, OpenAI o1/DeepSeek R1)

Why didn't they work before?

- **Datasets too small**: for machine translation, not really better until you have 1M+ parallel sentences (and really need a lot more)

- **Optimization not well understood**: good initialization, per-feature scaling + momentum (Adagrad/Adam) work best out-of-the-box
  - Regularization: dropout is pretty helpful
  - Computers not big enough: can't run for enough iterations

- Inputs: need **word embeddings** to have the right continuous semantics

Slide credit: Greg Durrett

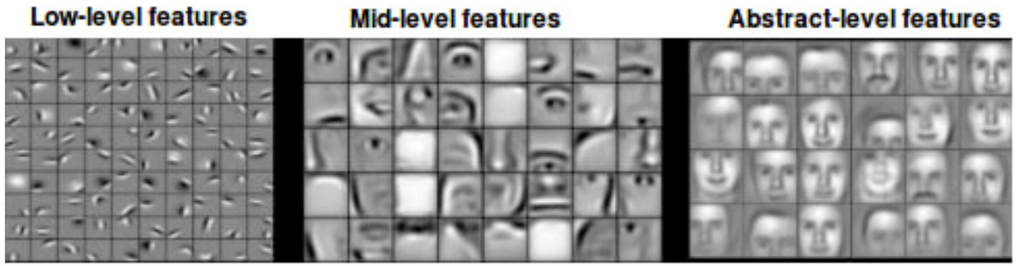
The "promise" of deep learning

- Most NLP works in the past focused on human-designed representations and input features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- **Representation learning** attempts to automatically learn good features and representations

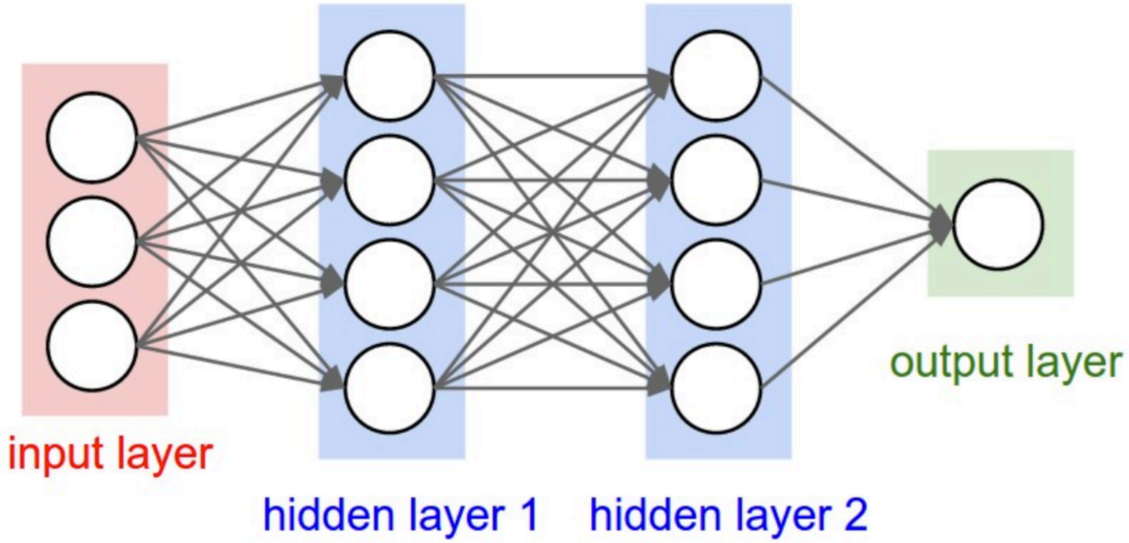
- **Deep learning** attempts to learn multiple levels of representations on increasing complexity/abstraction



Multilayer Perceptron (MLP)
(Neural Networks!)

MLP

- The units are connected with no cycles
- The outputs from units in each layer are passed to units in the next higher layer
- No outputs are passed back to lower layers

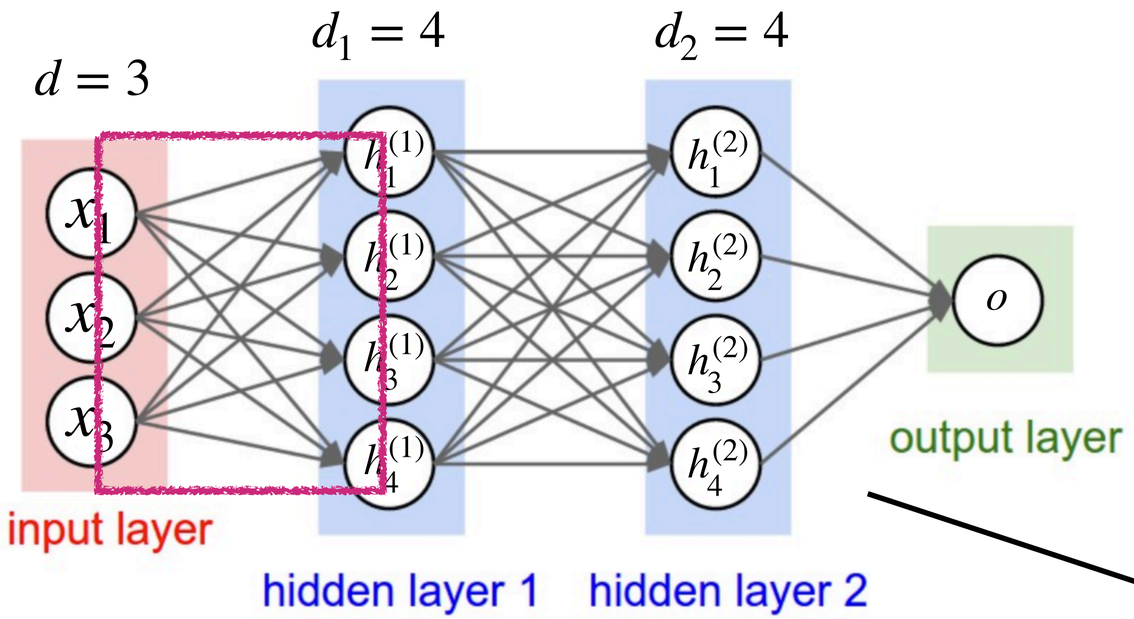


Fully-connected (FC) layers:

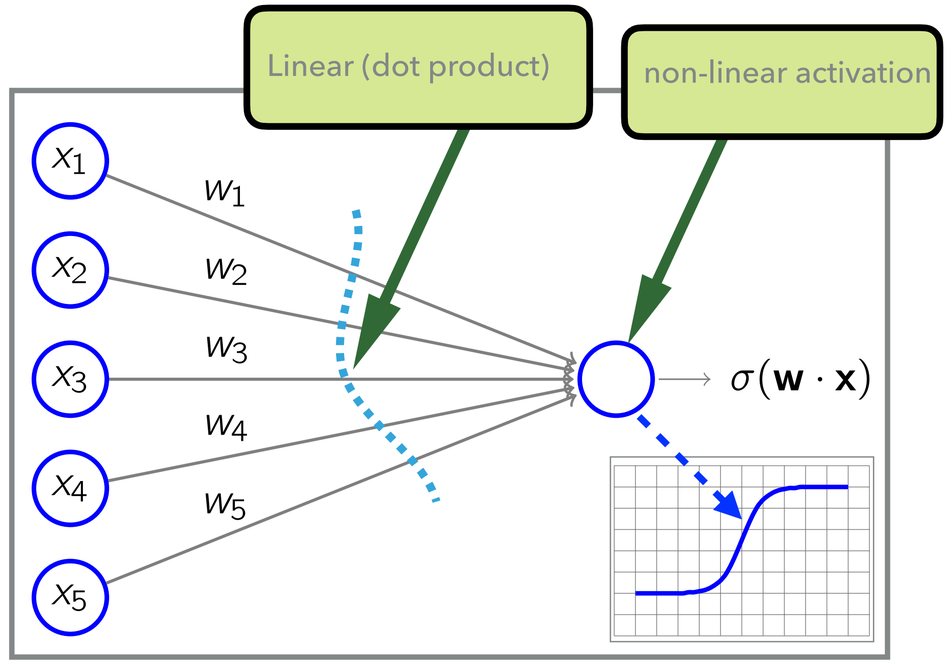
All the units from one layer are fully connected to every unit of the next layer.

1. We'll assume we already have the input vector $\mathbf{x} \in \mathbb{R}^d$ with fixed d (here, $d = 3$), and learn how MLP works.
2. After that, we'll learn how to get $\mathbf{x} \in \mathbb{R}^d$ for the text input.

MLP

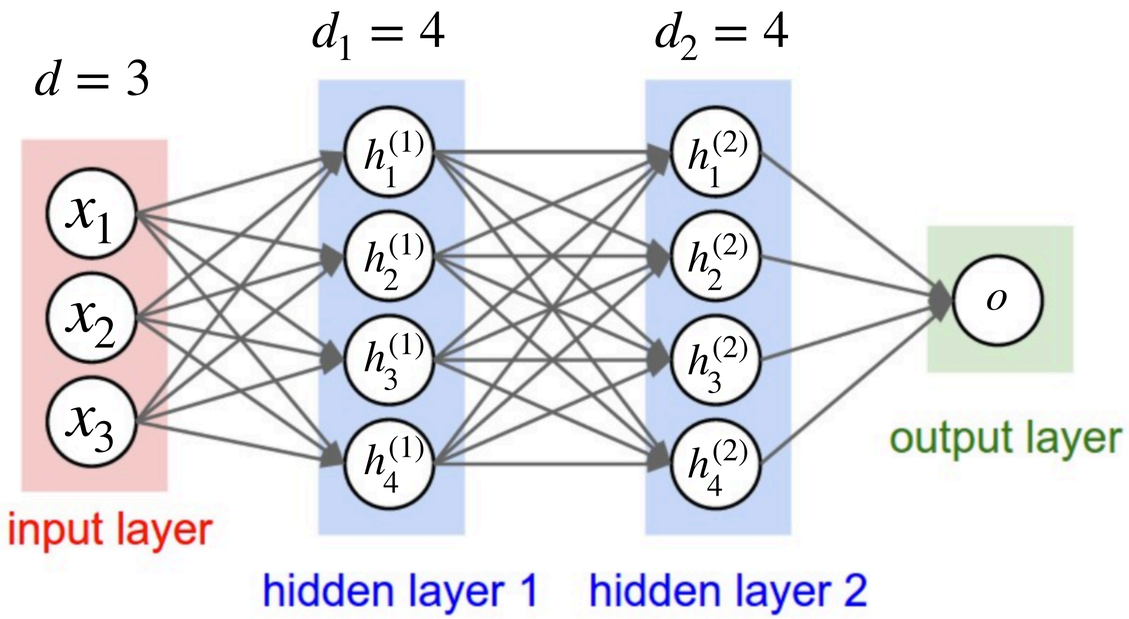


non-linearity f: σ, $\tanh$ *or ReLU.*



Bias term omitted for simplicity

MLP



Quick quiz: For $\mathbf{h} = \mathbf{W}^{(1)}\mathbf{x}$, what is the size of $\mathbf{W}^{(1)}$?

(a) $d \times d_1$ (b) $d_1 \times d$

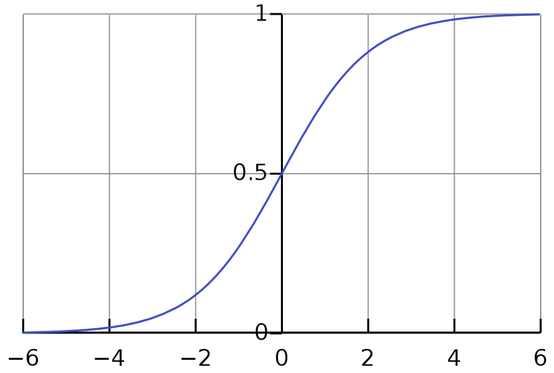
Answer is (b).

Bias term omitted for simplicity

Activation functions

sigmoid

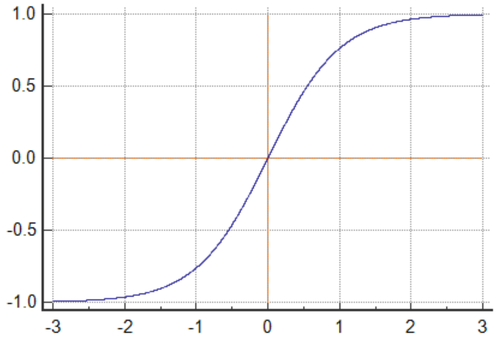
$$f(z) = \frac{1}{1 + e^{-z}}$$



$$f'(z) = f(z) \times (1 - f(z))$$

tanh

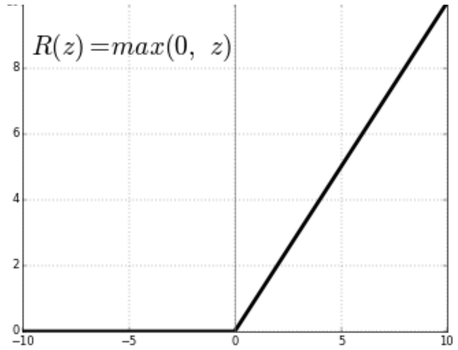
$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$



$$f'(z) = 1 - f(z)^2$$

ReLU
(rectified linear unit)

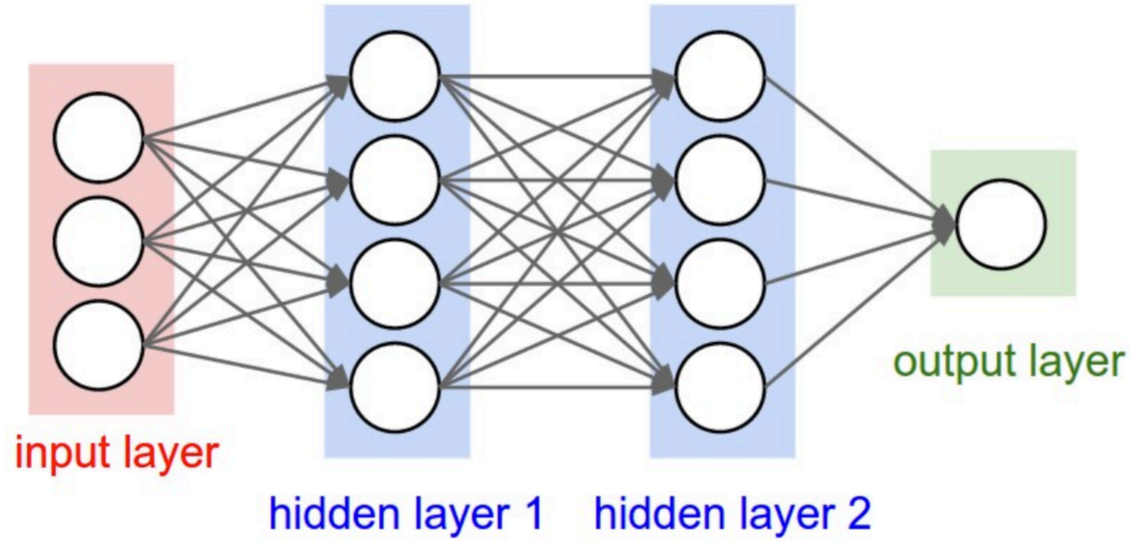
$$f(z) = \max(0, z)$$



$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

Activation function f is applied element-wise: $f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$

Matrix notations



d: input dimension, d_1, d_2: hidden dimensions

C: number of classes

- Input layer: $\mathbf{x}$
- Hidden layer 1:

$$\mathbf{h}_1 = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \in \mathbb{R}^{d_1}$$

$$\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$$

- Hidden layer 2:

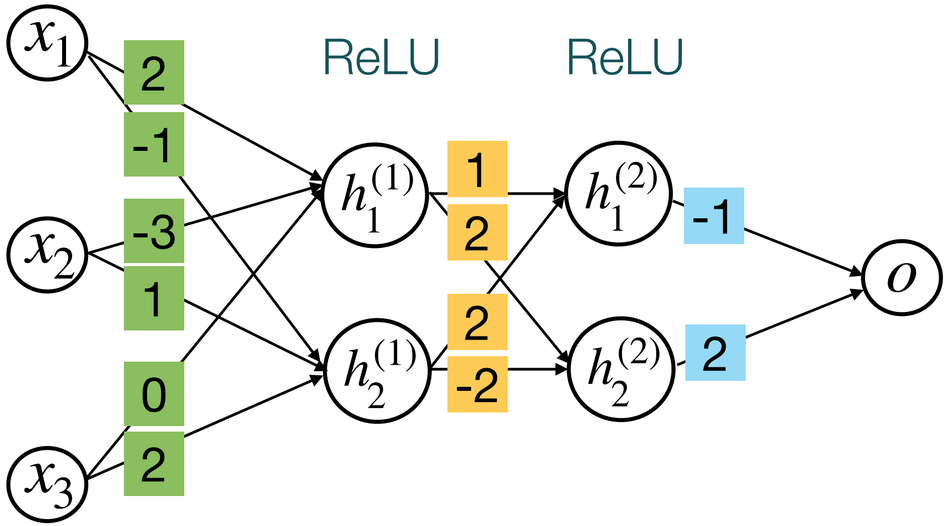
$$\mathbf{h}_2 = f(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}) \in \mathbb{R}^{d_2}$$

$$\mathbf{W}^{(2)} \in \mathbb{R}^{d_2 \times d_1}, \mathbf{b}^{(2)} \in \mathbb{R}^{d_2}$$

- Output layer:

$$\mathbf{y} = \mathbf{W}^{(o)}\mathbf{h}_2, \mathbf{W}^{(o)} \in \mathbb{R}^{C \times d_2}$$

Quick quiz



(Bias terms omitted in the next few slides)

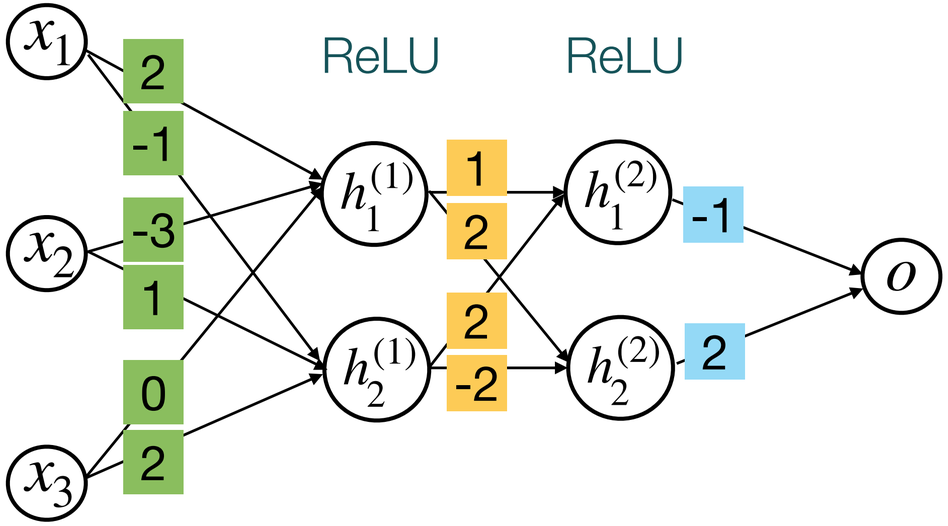
For $x_1 = x_2 = x_3 = 1$, what is the value of $h_1^{(1)}$?

(a) 0 (b) -1 (c) 1 (d) 2

Correct: (a), because of the RELU:

max(2 x 1 + (-3) x 1 + 0 x 1, 0) = max(-1, 0) = 0

Quick quiz



(Bias terms omitted in the next few slides)

$$\mathbf{h}^{(1)} = \mathrm{ReLU}(\mathbf{W}^{(1)}\mathbf{x})$$ What is the matrix $\mathbf{W}^{(1)}$?

(a) $\begin{bmatrix} 2 & -1 \\ -3 & 1 \\ 0 & 2 \end{bmatrix}$ (b) $\begin{bmatrix} 2 & -3 & 0 \\ -1 & 1 & 2 \end{bmatrix}$ (c) $\begin{bmatrix} 1 & 2 \\ 2 & -2 \end{bmatrix}$

Correct: (b). $\mathbf{W}^{(1)}$ is a 2 x 3 matrix.

MLP Objective

$$\mathbf{y} = \mathbf{W}^{(o)}\mathbf{h}_2$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{y}) \qquad \text{softmax}(\mathbf{y})_k = \frac{\exp(y_k)}{\sum_{j=1}^{C}\exp(y_j)} \qquad \mathbf{y} = [y_1, y_2, \ldots, y_C]$$

Training loss:

$$\min_{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(o)}} \; -\sum_{(\mathbf{x},y)\in D} \log \hat{\mathbf{y}}_y$$

Training feedforward NNs:
stochastic gradient descent!

Neural networks are difficult to optimize.
SGD can only converge to local minimum.
Initializations and optimizers matter a lot!

Backpropagation

Forward propagation:
from input to output layer

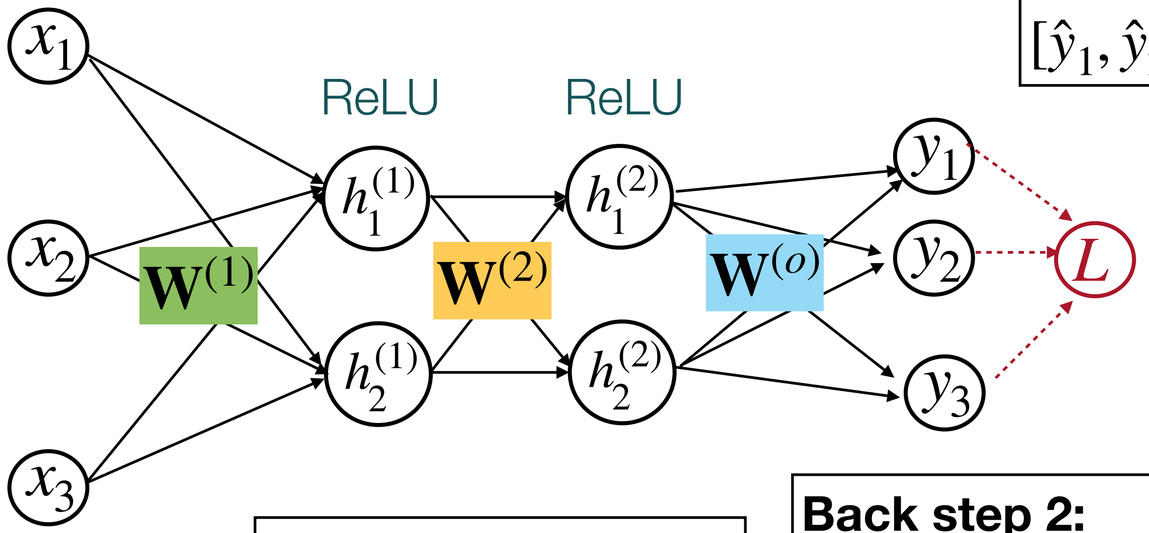
Given: x_1, x_2, x_3 and the class label y (a single training example)

Forward step 1:
Compute $h_1^{(1)}, h_2^{(1)}$

Forward step 2:
Compute $h_1^{(2)}, h_2^{(2)}$

Forward step 3:
Compute y_1, y_2, y_3 and
$[\hat{y}_1, \hat{y}_2, \hat{y}_3] = \text{softmax}[y_1, y_2, y_3]$

Forward step 4:
Compute loss
$L = -\log \hat{\mathbf{y}}_y$



Goal:
$\frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial W^{(2)}}, \frac{\partial L}{\partial W^{(o)}}$

Back step 4:
Compute
$\frac{\partial L}{\partial W^{(1)}}$

Back step 3:
Compute
$\frac{\partial L}{\partial h_1^{(1)}}, \frac{\partial L}{\partial h_2^{(1)}}, \frac{\partial L}{\partial W^{(2)}}$

Back step 2:
Compute
$\frac{\partial L}{\partial h_1^{(2)}}, \frac{\partial L}{\partial h_2^{(2)}}, \frac{\partial L}{\partial W^{(o)}}$

Back propagation:
from output to input layer

Back step 1:
Compute
$\frac{\partial L}{\partial y_1}, \frac{\partial L}{\partial y_2}, \frac{\partial L}{\partial y_3}$

Back-propagation in PyTorch

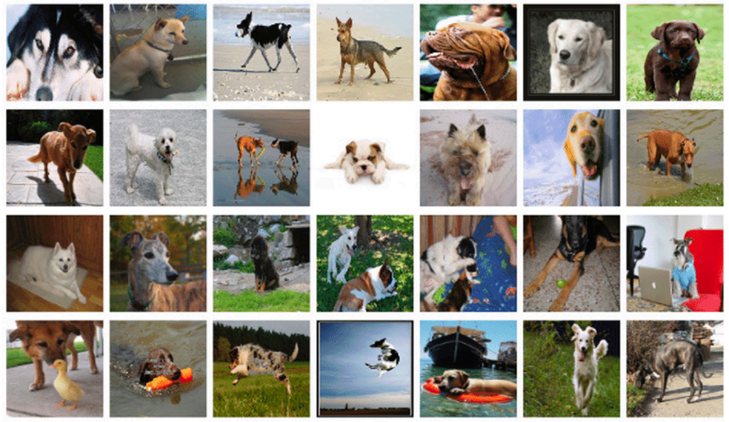
```python
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

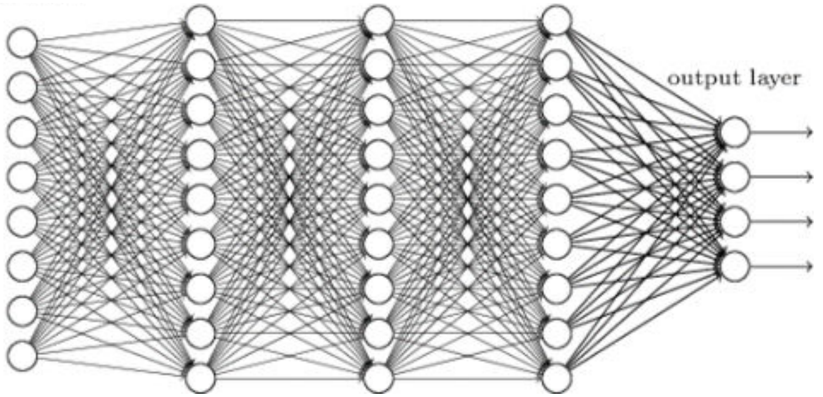
```python
import torch.optim as optim

net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```python
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

PyTorch did back-propagation for you in this one line of code!

Comparison: Image vs. text inputs





label = “dog”

a sometimes tedious film

i had to look away - this was god awful .

a gorgeous , witty , seductive movie .

label = positive

- Images: fixed-size input, continuous values
- Text: **variable-length** input, discrete words

How do we get $\mathbf{x} \in \mathbb{R}^d$ for the text input? → We’ll look into this next!

Jan 29 lecture starts from here

CS 288 Advanced Natural Language Processing

Course website: cal-cs288.github.io/sp26

Ed: edstem.org/us/join/XvztdK

- Class starts at 15:40!
- Questions about A1: Ed or GSI office hours!
- Lecture plan: Finish text classification (15min) → sequence models (65min)

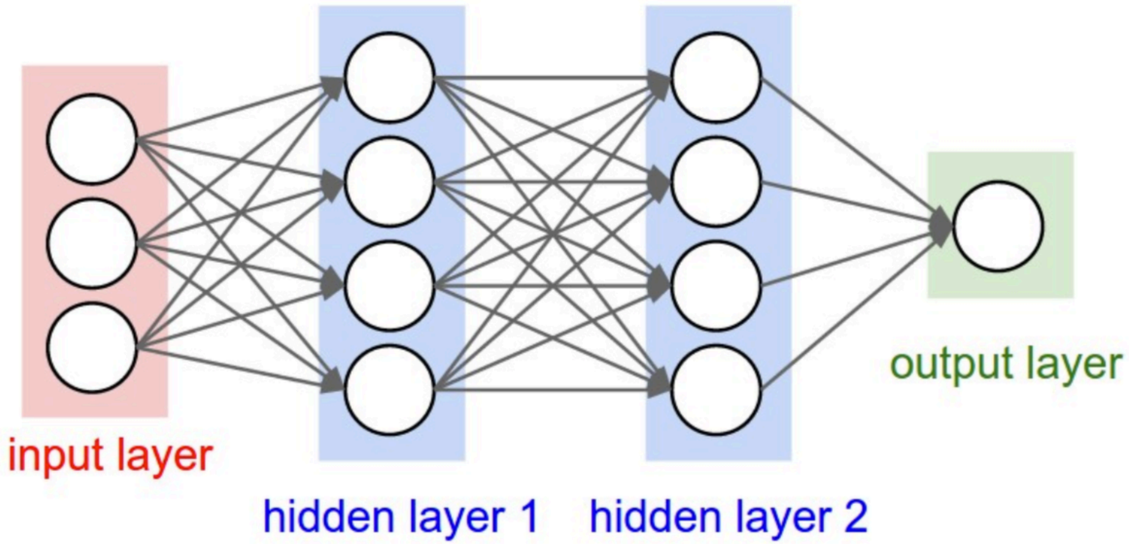
Recap: Text classification (1/2)

- Text classification: One of the basic NLP tasks!
  - Input: a pice of text; Output: a class label $y \in C$
  - Examples: spam vs. ham, topic classification, sentiment analysis
- Traditionally rule-based systems → No! Let's use statistics!
- Logistic regression
  - *Given*: Input vector $\mathbf{x} = [x_1, x_2, \ldots, x_k]$ (we'll talk about how to get $\mathbf{x}$ in a sec)

$$\hat{y} = P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

Recap: Text classification (2/2)

- Multilayer perceptron — Neural networks!
  - *Given*: Input vector $\mathbf{x} = [x_1, x_2, \ldots, x_k]$



- Hidden layer 1:

$$\mathbf{h}_1 = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \in \mathbb{R}^{d_1}$$

$$\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$$

- Hidden layer 2:

$$\mathbf{h}_2 = f(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}) \in \mathbb{R}^{d_2}$$

$$\mathbf{W}^{(2)} \in \mathbb{R}^{d_2 \times d_1}, \mathbf{b}^{(2)} \in \mathbb{R}^{d_2}$$

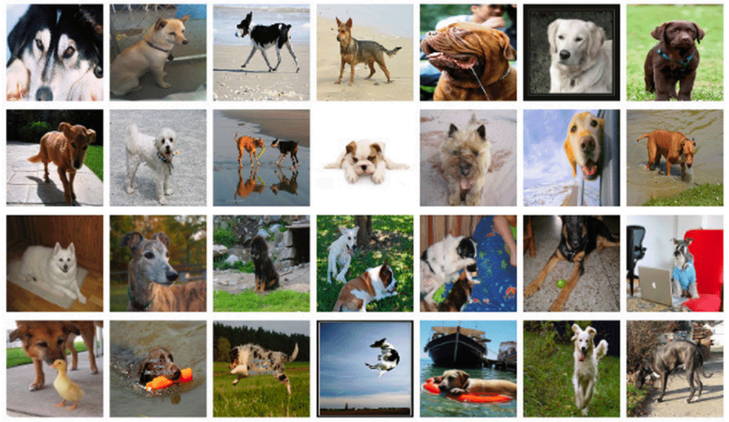
- Output layer:

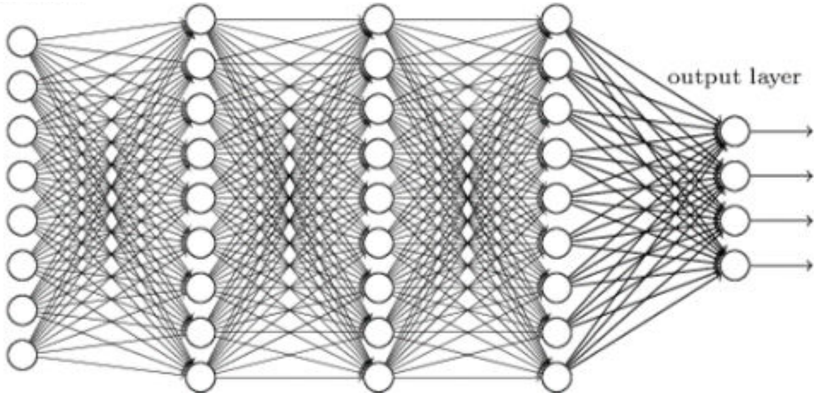
$$\mathbf{y} = \mathbf{W}^{(o)}\mathbf{h}_2, \mathbf{W}^{(o)} \in \mathbb{R}^{C \times d_2}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{y})$$

Remaining question: How to get $\mathbf{x}$?

Comparison: Image vs. text inputs





label = “dog”

a sometimes tedious film

i had to look away - this was god awful .

a gorgeous , witty , seductive movie .

label = positive

- Images: fixed-size input, continuous values
- Text: **variable-length** input, **discrete** words

One reason progress was slower in NLP than vision!

Neural networks for text classification

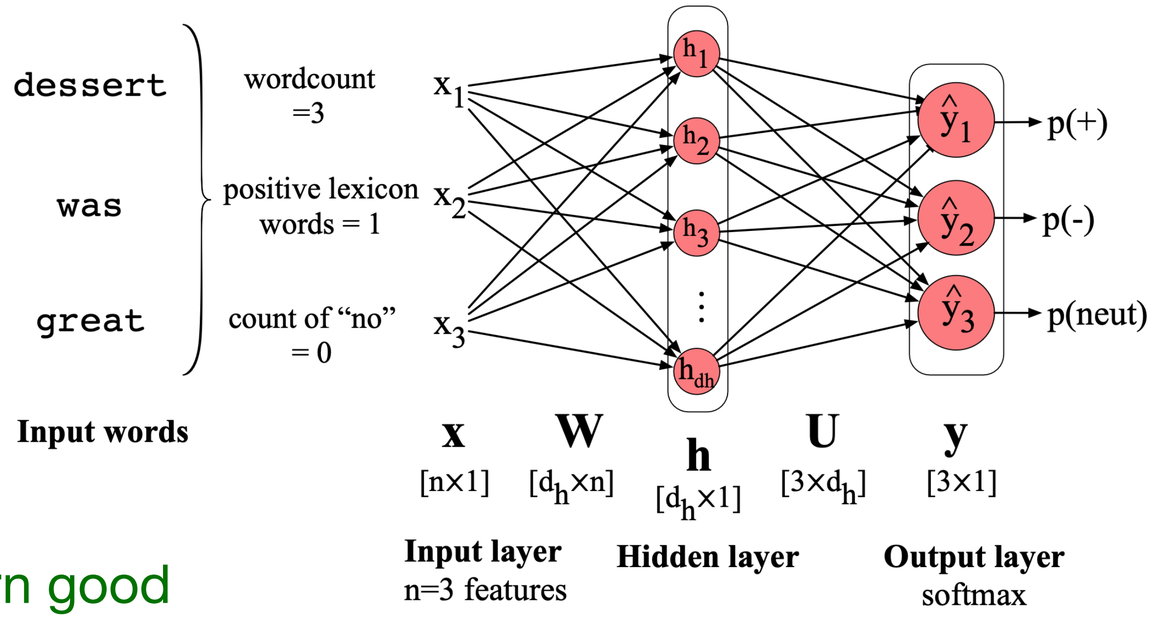
- Input: $w_1, w_2, \ldots, w_K \in V$
- Output: $y \in C$

- Input: dessert was great
- Output: positive C = {positive, negative, neutral}

Solution #1: You can construct a feature vector $\mathbf{x}$ from the input and simply feed the vector to a **neural network**!

x_1 = word count
x_2 = # of positive lexicon words
x_3 = Count of "no"

$$\mathbf{x} = [x_1, x_2, x_3]$$



Deep learning has the promise to learn good features automatically..

Important note: each input has a different K

Neural networks for text classification

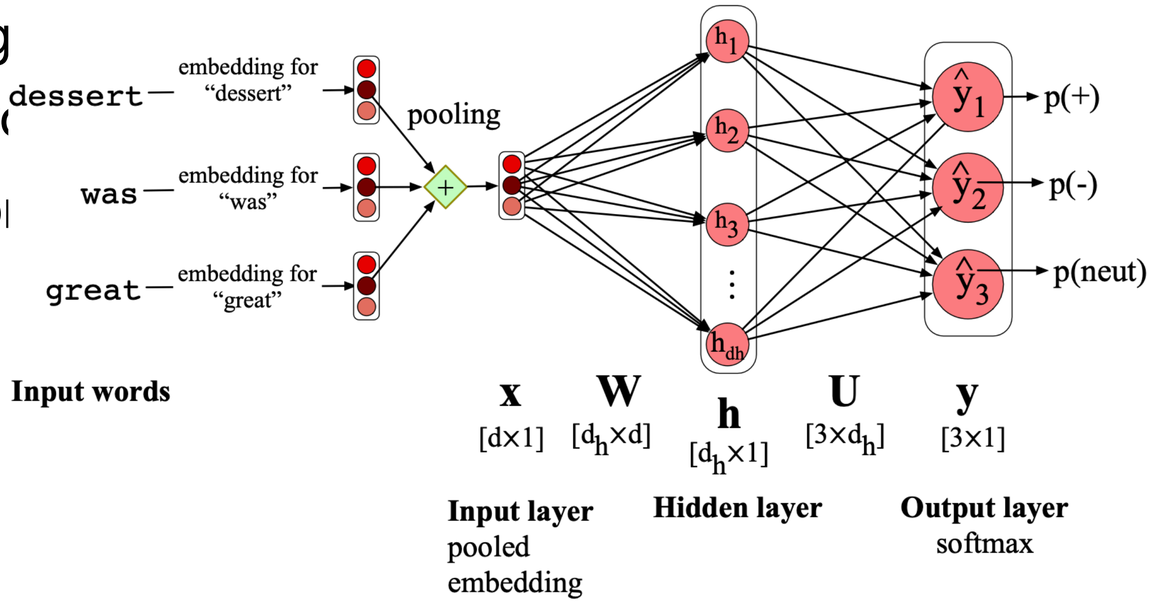
- Input: $w_1, w_2, \ldots, w_K \in V$
- Output: $y \in C$

- Input: dessert was great
- Output: positive C = {positive, negative, neutral}

Solution #2: Use **word embeddings**!

- First, look up all the word embedding
  $E(w_1), E(w_2), \ldots, E(w_K) \in \mathbb{R}^d$ (d: w
- Use **pooling**: sum, mean, or max. Fo

$$\mathbf{x} = \frac{1}{K} \sum_{i=1}^{K} E(w_i) \in \mathbb{R}^d$$

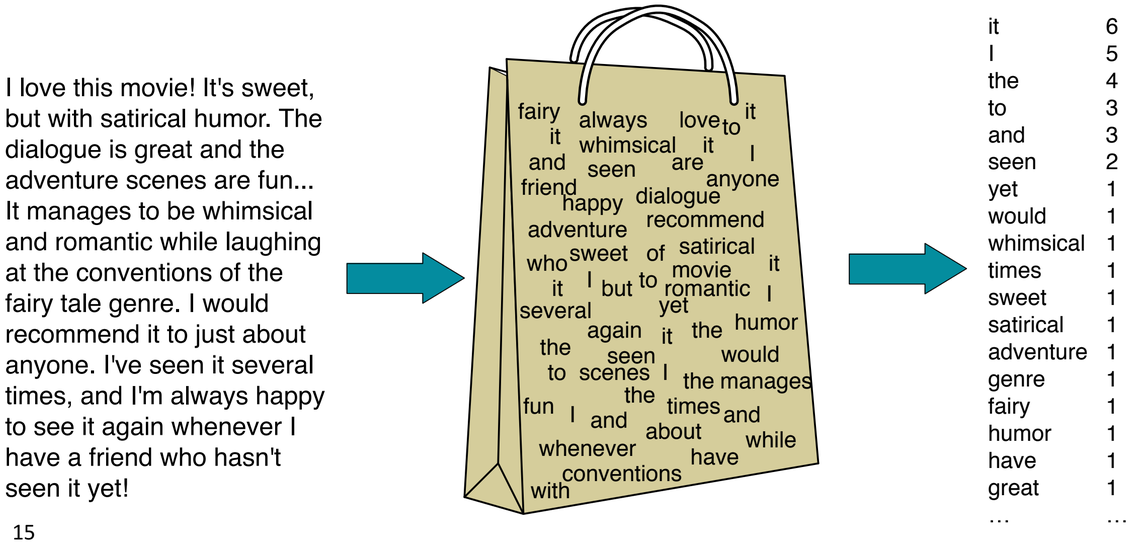


Important note: each input has a different K

Neural networks for text classification

- (+): This provides a simple and flexible way to handle variable-length input
- (+): It learns feature representations automatically from the data
- (+): It can generalize to similar inputs through word embeddings
- (-): The model throws away any sequential information of the text

neural bag-of-words model (NBOW)



When training, how to handle E?

- Word embeddings can be treated as parameters too!

$$\mathbf{E} \in \mathbb{R}^{|V| \times d}$$

- Common practice: initialize $\mathbf{E}$ using word embeddings (e.g. word2vec), and optimize them jointly with other parameters, using SGD!

- When the training data is small, don't treat $\mathbf{E}$ as parameters!

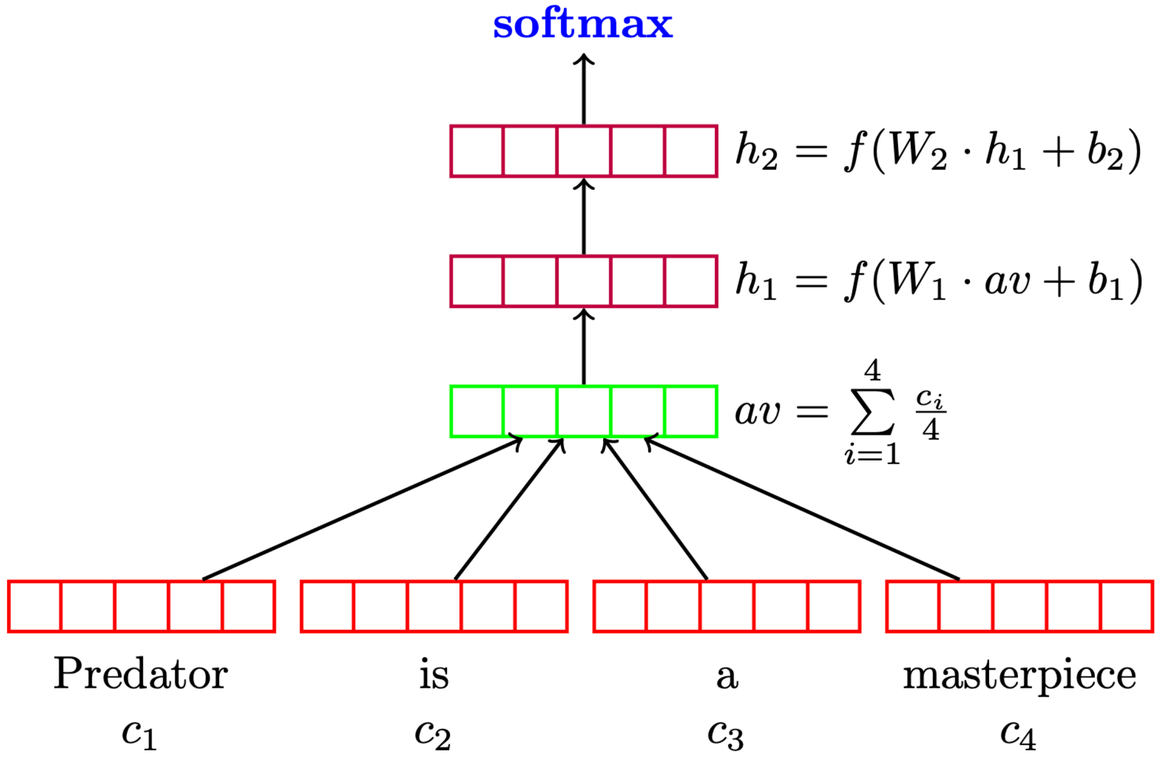
- When the training data is very large (e.g., language modeling), initialization doesn't matter much either (= can use random initialization)

Deep Averaging Networks (DAN)

(Iyyer et al., 2015)

Deep Unordered Composition Rivals Syntactic Methods for Text Classification

DAN



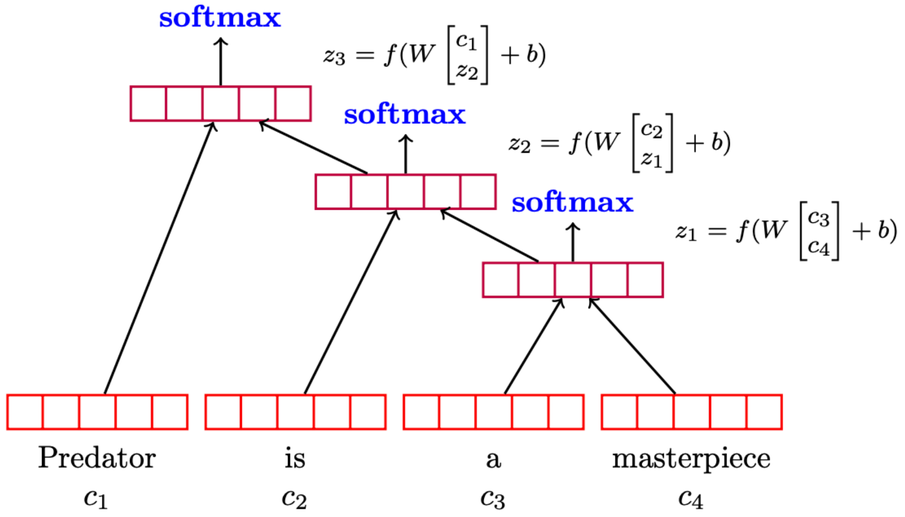
Basically the same as NBOW but neural network is deeper!

f: non-linearity

Deep Averaging Networks (DAN)

DAN-RAND: no initialization from GloVe

RecNN



Model	RT	SST fine	SST bin	IMDB	Time (s)
DAN-ROOT	—	46.9	85.7	—	**31**
DAN-RAND	77.3	45.4	83.2	88.8	136
DAN	80.3	47.7	86.3	89.4	136
NBOW-RAND	76.2	42.3	81.4	88.9	91
NBOW	79.0	43.6	83.6	89.0	91
BiNB	—	41.9	83.1	—	—
NBSVM-bi	79.4	—	—	91.2	—
RecNN*	77.7	43.2	82.4	—	—
RecNTN*	—	45.7	85.4	—	—
DRecNN	—	49.8	86.6	—	431
TreeLSTM	—	**50.6**	86.9	—	—
DCNN*	—	48.5	86.9	89.4	—
PVEC*	—	48.7	87.8	**92.6**	—
CNN-MC	**81.1**	47.4	**88.1**	—	2,452
WRRBM*	—	—	—	89.2	—

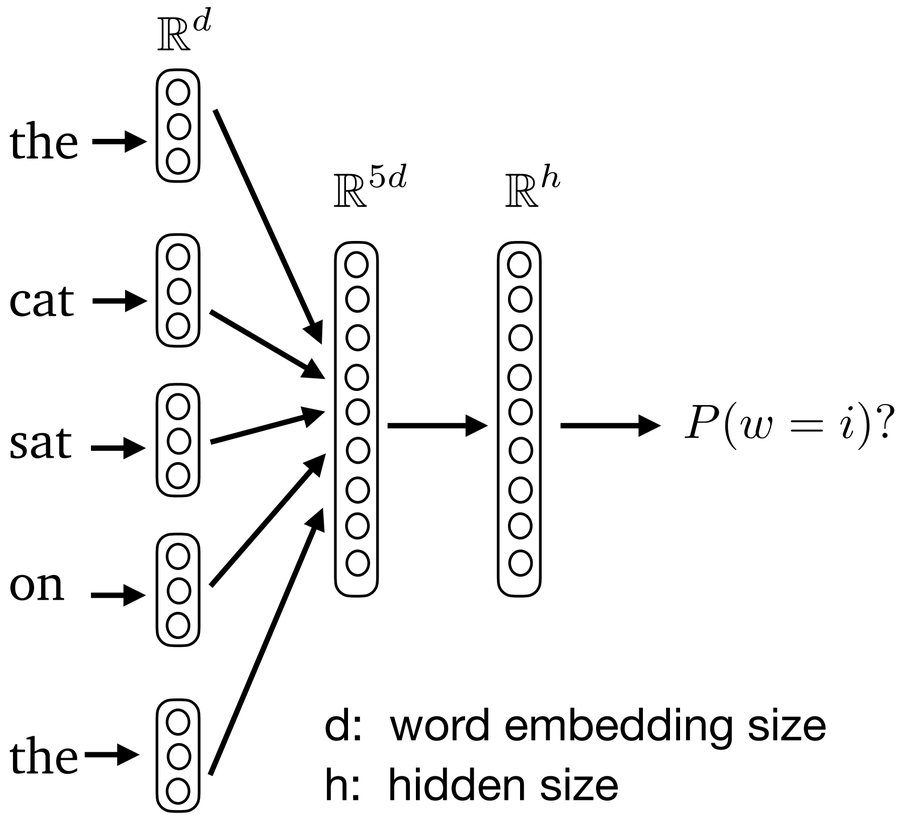
Bonus slide: Neural Language Models

$$P(x_1, x_2, \ldots, x_n) \approx \prod_{i=1}^{n} P(x_i \mid x_{i-m+1}, \ldots, x_{i-1})$$

- Can be considered as $|V|$-way classification!
- In particular, feedforward neural language models approximate the probability based on the previous *m* (e.g., 5) words

Why concatenate, not mean-pool?

Why not concatenate in text classification?



Questions?

Acknowledgement

Princeton COS 484 by Danqi Chen, Tri Dao, Vikram Ramaswamy

Cornell LM-class by Yoav Artzi

CMU Advanced NLP by Graham Neubig & Sean Welleck