CS 4650/7650:
Natural Language Processing

Neural Text Classification

Diyi Yang

Some slides borrowed from Jacob Eisenstein (was at GT) and Danqi Chen & Karthik Narasimhan (Princeton)

Homework and Project Schedule

- First half of the semester: homework
- Mid: midterm
- Second half of the semester: project

This Lecture

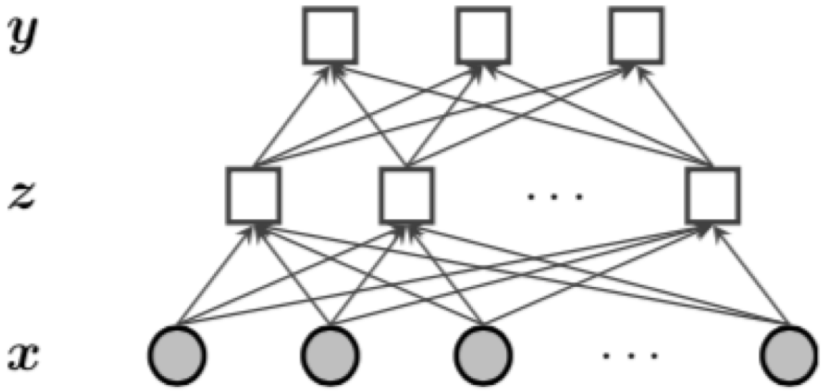
- Feedforward neural network
- Learning neural networks
- Text classification applications
- Evaluating text classifier

A Simple Feedforward Architecture

Suppose we want to label stories as $y \in \{Bad, Good, Okay\}$

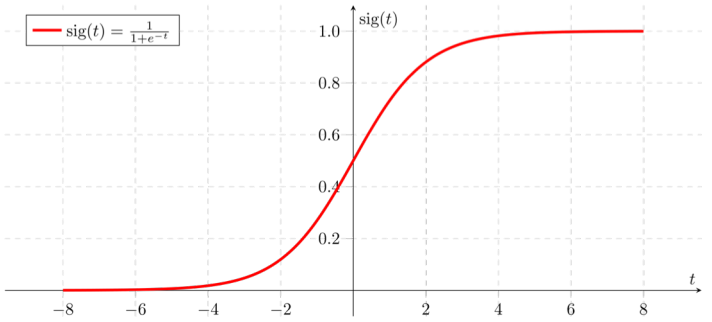
- What makes a good story?
- Let's call this vector of features $\boldsymbol{z}$
- If $\boldsymbol{z}$ is well-chosen, it will be easy to predict from $\boldsymbol{x}$, and it will make it easy to predict y (the label)

A Simple Feedforward Architecture

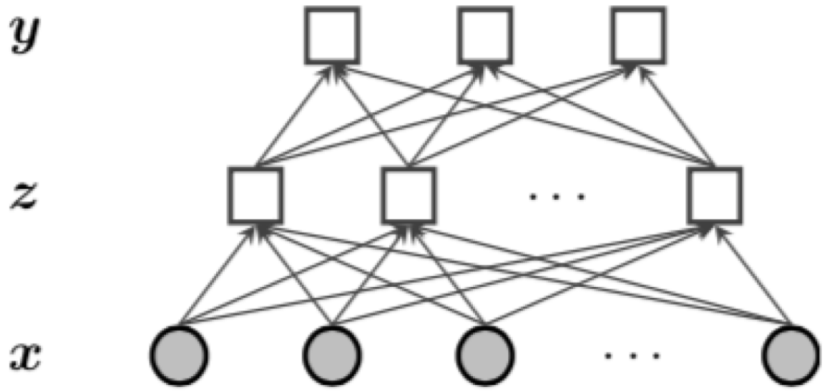


Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$
$$= (1 + \exp(-\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}))^{-1}$$



A Simple Feedforward Architecture

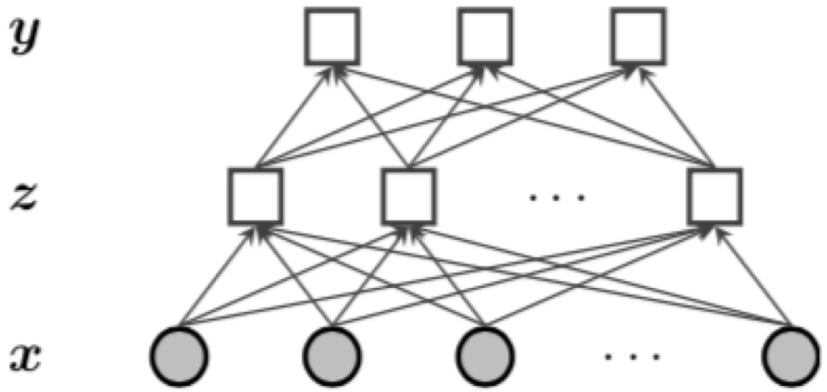


Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\begin{aligned}\Pr(z_k = 1 \mid \boldsymbol{x}) &= \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}) \\ &= (1 + \exp(-\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}))^{-1}\end{aligned}$$

$$\boldsymbol{\Theta}^{(x \to z)} = [\boldsymbol{\theta}_1^{(x \to z)}, \boldsymbol{\theta}_2^{(x \to z)}, \ldots, \boldsymbol{\theta}_{K_z}^{(x \to z)}]^\top$$

A Simple Feedforward Architecture



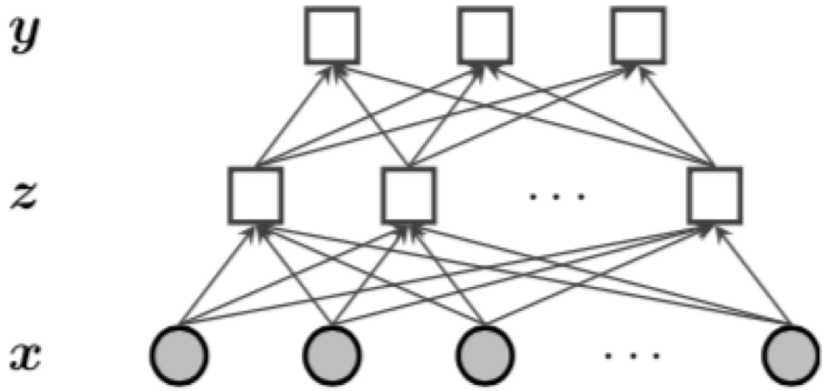
Next predict y from $\boldsymbol{z}$, again via logistic regression:

$$\Pr(y = j \mid z) = \frac{\exp(\boldsymbol{\theta}_j^{(z \to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\boldsymbol{\theta}_{j'}^{(z \to y)} \cdot \boldsymbol{z} + b_{j'})}$$

where each b_j is an offset. This is denoted:

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \text{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

Feedforward Neural Network



To summarize:

$$\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$$

$$\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{z}) = \mathrm{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

- In reality, we never observe $\boldsymbol{z}$, it is a hidden layer. We compute $\boldsymbol{z}$ directly from $\boldsymbol{x}$.
- This makes $\mathrm{p}(\mathrm{y}|\boldsymbol{x})$ a nonlinear function of $\boldsymbol{x}$

Designing Feedforward Neural Network

1. Activation Functions

Sigmoid Function

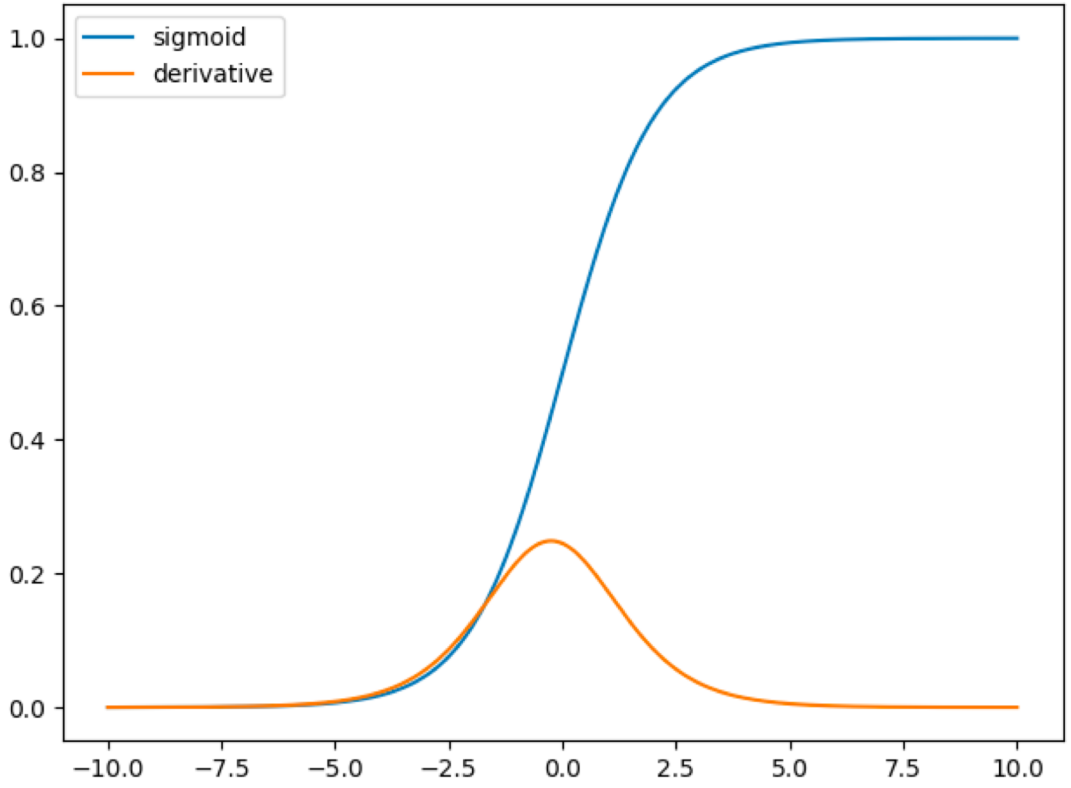
The **sigmoid** in $\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$

is an **activation function**

In general, we write $\boldsymbol{z} = f(\boldsymbol{\Theta}^{(x \to z)}\boldsymbol{x})$ to

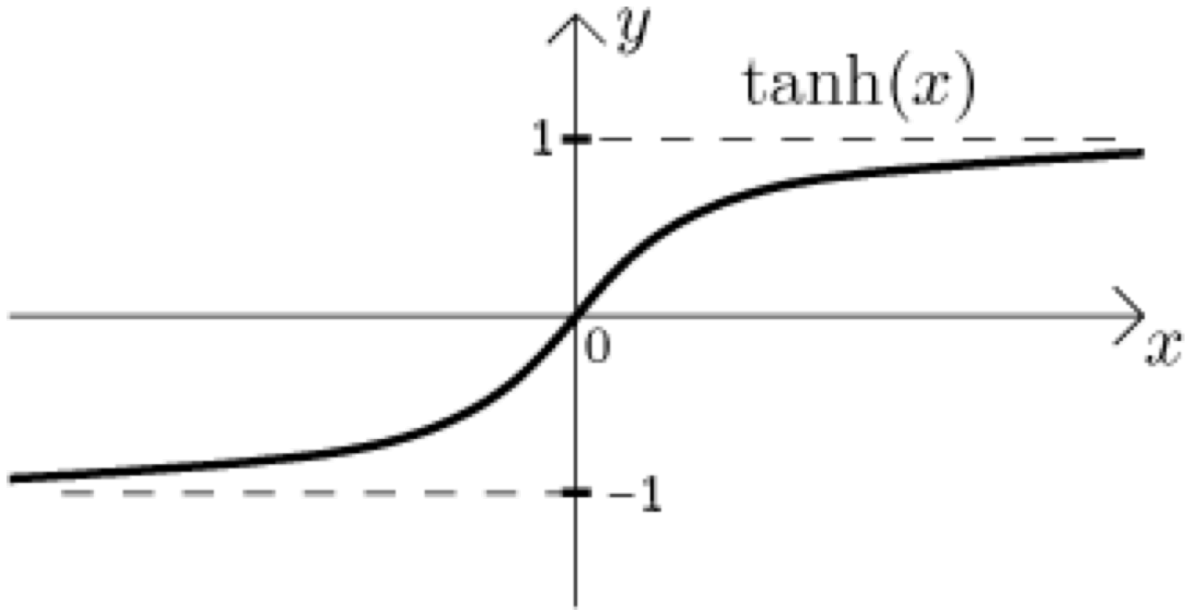
indicate an arbitrary activation function.

$$\frac{\partial}{\partial a}\sigma(a) = \sigma(a)(1 - \sigma(a))$$



Tanh Function

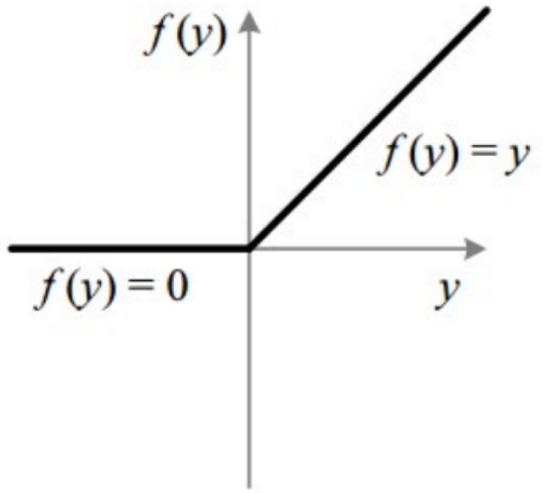
- Hyperbolic Tangent
- Range: (-1, 1)
- $\frac{\partial}{\partial a}\tanh(a) = 1 - \tanh(a)^2$

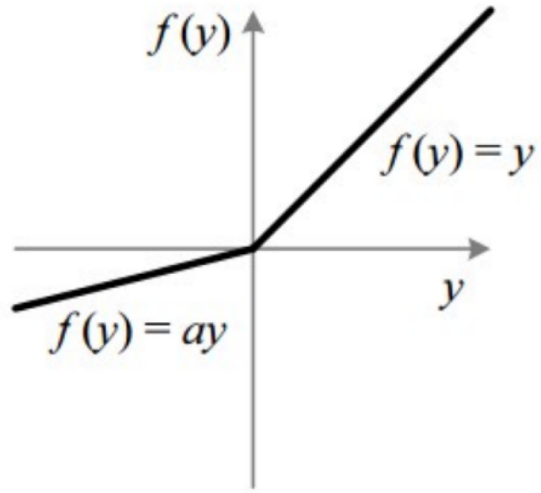


ReLU Function

- Rectified Linear Unit

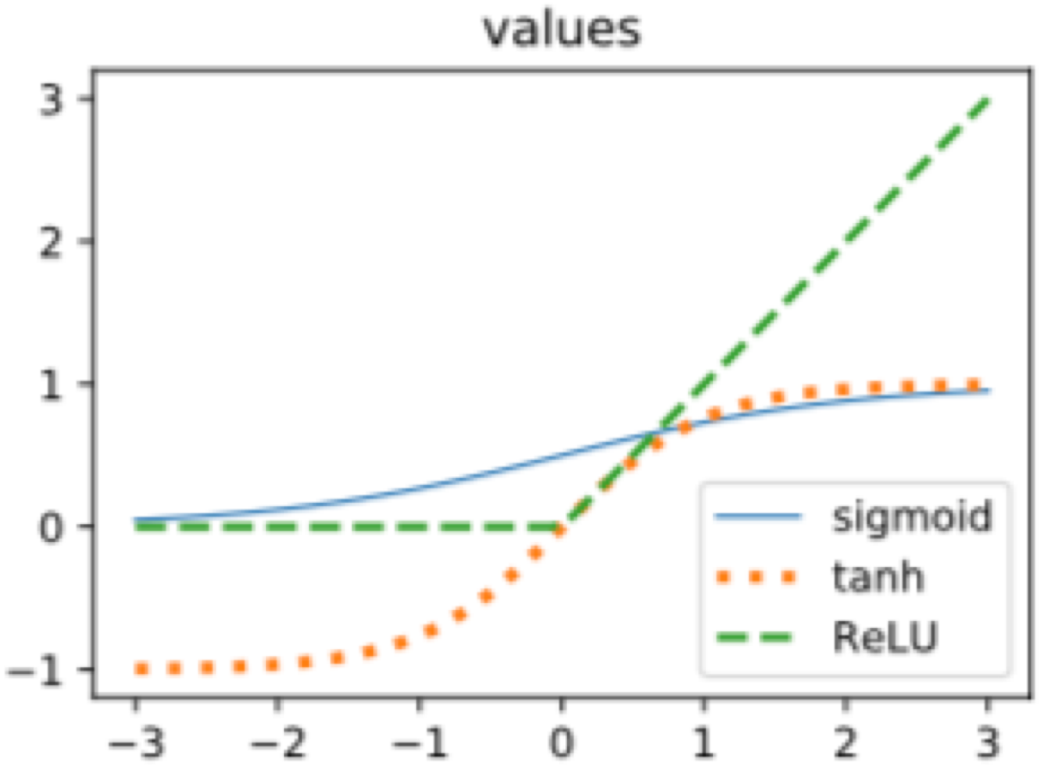
$$\text{ReLU}(a) = \begin{cases} a, & a \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$

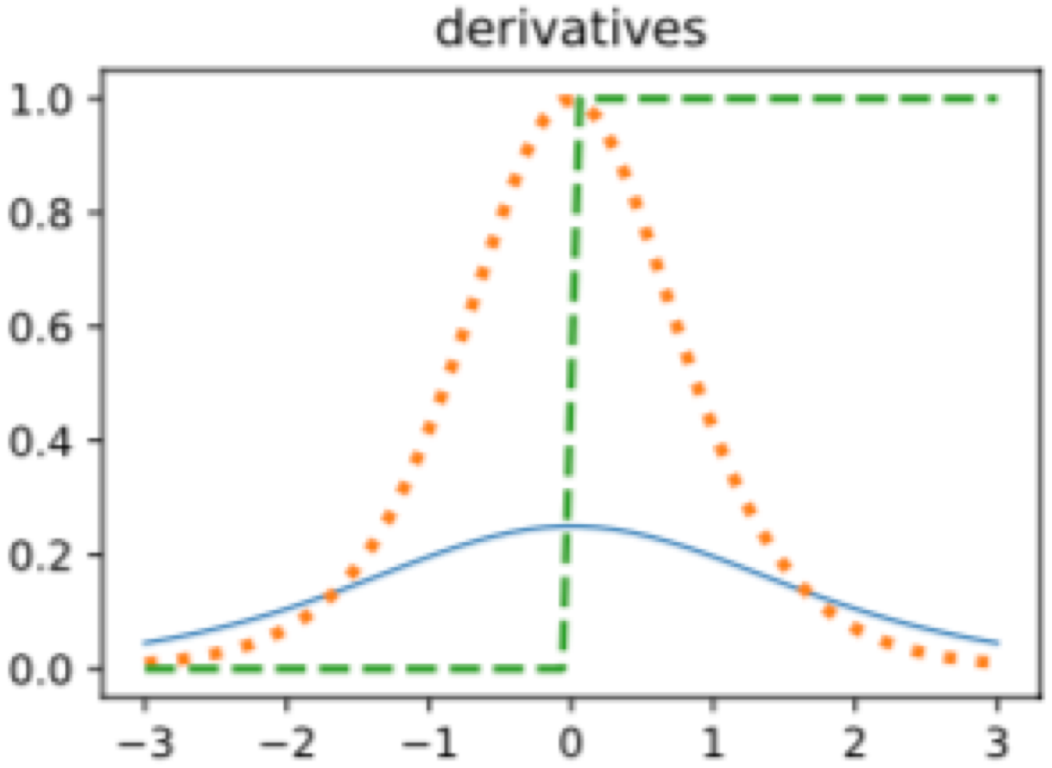




- Leaky ReLU

Activation Functions





Designing Feedforward Neural Network

2. Outputs and Loss Function

Outputs and Loss Functions

- The **softmax** output activation is used in combination with the negative log-likelihood loss, like logistic regression.
- In deep learning, this loss is called the cross-entropy:

$$\tilde{\boldsymbol{y}} = \text{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

$$-\mathcal{L} = -\sum_{i=1}^{N} \boldsymbol{e}_{y^{(i)}} \cdot \log \tilde{\boldsymbol{y}},$$

where $\boldsymbol{e}_{y^{(i)}} = [0, 0, \ldots, 0, \underbrace{1}_{y^{(i)}}, 0, \ldots, 0]$, **a one-hot vector**

Designing Feedforward Neural Network

3. Input and Lookup Layers

Designing Feedforward Neural Network

4. Learning Neural Networks

Gradient Descent in Neural Networks

Neural networks are often learned by gradient descent, typically with minibatches

$$\boldsymbol{\theta}_k^{(z \to y)} \leftarrow \boldsymbol{\theta}_k^{(z \to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$$

$\eta^{(t)}$ is the learning rate at update t

$\ell^{(i)}$ is the loss on instance (minibatch) i

$\nabla_{\boldsymbol{\theta}_k^{(z \to y)}} \ell^{(i)}$ is the gradient of the loss wrt the column vector of output weights $\boldsymbol{\theta}_k^{(z \to y)}$

Gradient Descent in Neural Networks

Neural networks are often learned by gradient descent, typically with minibatches

$$\boldsymbol{\theta}_k^{(z\to y)} \leftarrow \boldsymbol{\theta}_k^{(z\to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)}$$

$\eta^{(t)}$ is the learning rate at update t

$\ell^{(i)}$ is the loss on instance (minibatch) i

$\nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)}$ is the gradient of the loss wrt the column vector of output weights $\boldsymbol{\theta}_k^{(z\to y)}$

$$\nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)} = \left[\frac{\partial \ell^{(i)}}{\partial \theta_{k,1}^{(z\to y)}}, \frac{\partial \ell^{(i)}}{\partial \theta_{k,2}^{(z\to y)}}, \ldots, \frac{\partial \ell^{(i)}}{\partial \theta_{k,K_y}^{(z\to y)}} \right]^\top$$

Gradient Descent for Simple Feedforward Neural Net

Feedforward Network

$$\boldsymbol{z} \leftarrow f\left(\boldsymbol{\Theta}^{(x \rightarrow z)} \boldsymbol{x}^{(i)}\right)$$

$$\tilde{\boldsymbol{y}} \leftarrow \text{SoftMax}\left(\boldsymbol{\Theta}^{(z \rightarrow y)} \boldsymbol{z} + \boldsymbol{b}\right)$$

$$\ell^{(i)} \leftarrow -\boldsymbol{e}_{y^{(i)}} \cdot \log \tilde{\boldsymbol{y}},$$

Update Rule

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta^{(t)} \nabla_{\boldsymbol{b}} \ell^{(i)}$$

$$\boldsymbol{\theta}_k^{(z \rightarrow y)} \leftarrow \boldsymbol{\theta}_k^{(z \rightarrow y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z \rightarrow y)}} \ell^{(i)}$$

$$\boldsymbol{\theta}_n^{(x \rightarrow z)} \leftarrow \boldsymbol{\theta}_n^{(x \rightarrow z)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_n^{(x \rightarrow z)}} \ell^{(i)},$$

Backpropagation

If we don't observe $\boldsymbol{z}$,

how can we learn the weight $\Theta^{(x \to z)}$?

Backpropagation

Compute loss on y & apply **chain rule** of calculus to compute gradient on all parameters

$$\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \to z)}} = \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial z_k}{\partial \theta_{n,k}^{(x \to z)}}$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial f(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})}{\partial \theta_{n,k}^{(x \to z)}}$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \times f'(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}) \times x_n$$

A Working Example:

Deriving Gradients for Simple Neural Network

Input: $\mathbf{x}$ $\quad \mathbf{x} \in \mathbb{R}^d$

$$\mathbf{h}_1 = \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{W}_1 \in \mathbb{R}^{d_1 \times d} \quad \mathbf{b}_1 \in \mathbb{R}^{d_1}$$

$$\mathbf{h}_2 = \tanh(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \qquad \mathbf{W}_2 \in \mathbb{R}^{d_2 \times d_1} \quad \mathbf{b}_2 \in \mathbb{R}^{d_2}$$

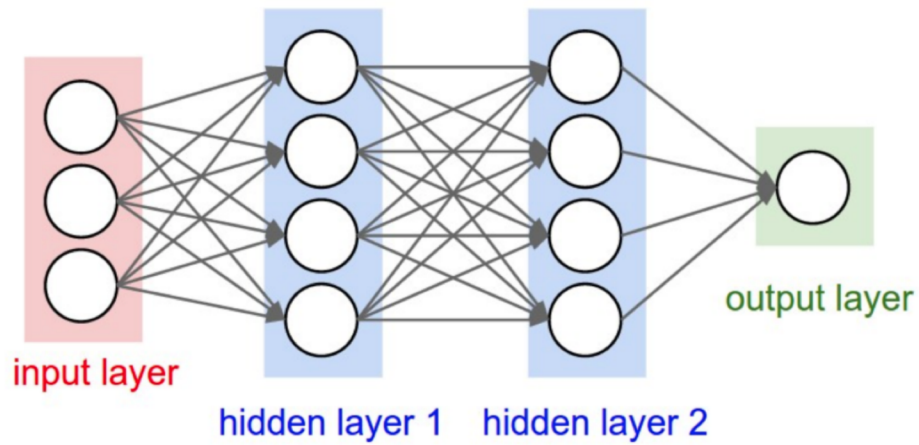
$$y = \sigma(\mathbf{w}^\intercal \mathbf{h_2} + b) \qquad \mathbf{w} \in \mathbb{R}^{d_2}$$

$$\mathcal{L}(y, y^*) = -y^* \log y - (1 - y^*) \log (1 - y)$$

$$\frac{\partial L}{\partial \mathbf{w}} = ? \qquad \frac{\partial L}{\partial b} = ?$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = ? \qquad \frac{\partial L}{\partial \mathbf{b}_2} = ?$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = ? \qquad \frac{\partial L}{\partial \mathbf{b}_1} = ?$$

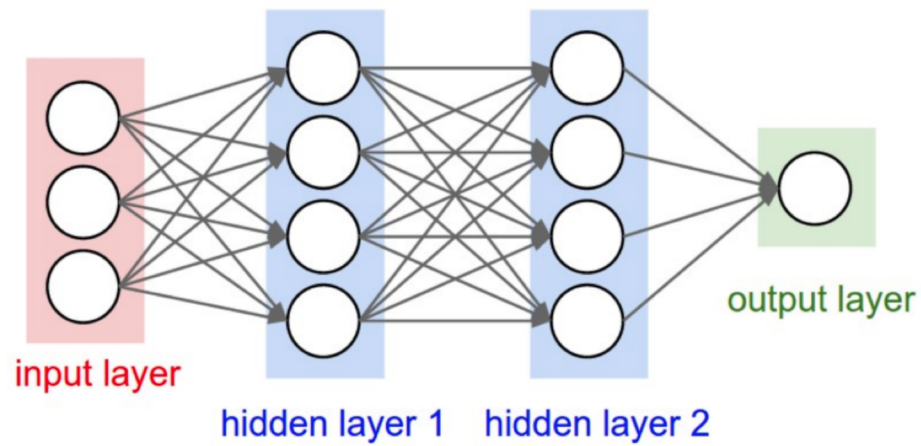


$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \qquad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$

$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \qquad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$

$$y = \sigma(\mathbf{w}^\intercal \mathbf{h_2} + b)$$

Forward Propagation



$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$
$$y = \sigma(\mathbf{w}^\intercal \mathbf{h_2} + b)$$

↓ Forward Propagation

↓ **Backward Propagation**

$$L(y, y^*) = -y^* \log y - (1-y^*)\log(1-y)$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial w} = \left(-y^*\frac{1}{y} + (1-y^*)\frac{1}{1-y}\right)\frac{\partial y}{\partial w}$$

$$= \left(\frac{-y^*(1-y)+(1-y^*)y}{y(1-y)}\right)\cdot y(1-y)\cdot h_2$$

$$= \left(-y^* + yy^* + y - yy^*\right)h_2 = (y-y^*)h_2$$

$$\frac{\partial L}{\partial b} = (y-y^*)$$

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$
$$y = \sigma(\mathbf{w}^\intercal\mathbf{h_2} + b)$$

Forward Propagation

$$\frac{\partial \mathcal{L}}{\partial b} = y - y^* \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = (y - y^*)\mathbf{h}_2 \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_2} = (y - y^*)\mathbf{w}$$

Backward Propagation

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$
$$y = \sigma(\mathbf{w}^\intercal\mathbf{h_2} + b)$$

Forward Propagation

$$\frac{\partial \mathcal{L}}{\partial b} = y - y^* \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = (y - y^*)\mathbf{h}_2 \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_2} = (y - y^*)\mathbf{w}$$

Backward Propagation

$$\frac{\partial L}{\partial \mathbf{z}_2} = (1 - \mathbf{h}_2^2) \circ \frac{\partial L}{\partial \mathbf{h}_2}$$

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$
$$y = \sigma(\mathbf{w}^\intercal\mathbf{h_2} + b)$$

Forward Propagation

$$\frac{\partial \mathcal{L}}{\partial b} = y - y^* \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = (y - y^*)\mathbf{h}_2 \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_2} = (y - y^*)\mathbf{w}$$

Backward Propagation

$$\frac{\partial L}{\partial \mathbf{z}_2} = (1 - \mathbf{h}_2^2) \circ \frac{\partial L}{\partial \mathbf{h}_2}$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{z}_2}\mathbf{h}_1^\intercal \qquad \frac{\partial L}{\partial \mathbf{b}_2} = \frac{\partial L}{\partial \mathbf{z}_2}$$

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$
$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \quad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$
$$y = \sigma(\mathbf{w}^\intercal\mathbf{h_2} + b)$$

Forward Propagation

$$\frac{\partial \mathcal{L}}{\partial b} = y - y^* \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = (y - y^*)\mathbf{h}_2 \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_2} = (y - y^*)\mathbf{w}$$

Backward Propagation

$$\frac{\partial L}{\partial \mathbf{z}_2} = (1 - \mathbf{h}_2^2) \circ \frac{\partial L}{\partial \mathbf{h}_2}$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{z}_2}\mathbf{h}_1^\intercal \qquad \frac{\partial L}{\partial \mathbf{b}_2} = \frac{\partial L}{\partial \mathbf{z}_2} \qquad \frac{\partial L}{\partial \mathbf{h}_1} = \mathbf{W}_2^\intercal \frac{\partial L}{\partial \mathbf{z}_2}$$

$$\frac{\partial L}{\partial \mathbf{z}_1} = (1 - \mathbf{h}_1^2) \circ \frac{\partial L}{\partial \mathbf{h}_1} \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{z_1}}\mathbf{x}^\intercal \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{z_1}}$$

Backpropagation as an algorithm

Forward propagation:

- Visit nodes in topological sort order
- Compute value of node given predecessors

Backward propagation:

- Visit nodes in reverse order
- Compute gradient wrt each node using gradient wrt successors

Backpropagation

Re-use derivatives computed for higher layers in computing derivatives for lower layers so as to minimize computation

 Good news is that modern automatic differentiation tools did all for you!

Implementing backprop by hand is like programming in assembly language.

“Tricks” for Better Performance

- Preventing overfitting with regularization and dropout
- Smart initialization
- Online learning

"Tricks": Regularization and Dropout

Because neural networks are powerful learners, overfitting is a potential problem.

- Regularization works similarly for neural nets as it does in linear classifiers: penalize the weights by $\lambda||\boldsymbol{\theta}||_2^2$.
- Dropout prevents overfitting by randomly deleting weights or nodes during training. This prevents the model from relying too much on individual features or connections.
- Dropout rates are usually between 0.1 and 0.5, tuned on validate data.

“Tricks”: Initialization

Unlike linear classifiers, initialization in neural networks can affect the outcome.

- If the initial weights are too large, activations may saturate the activation function (for sigmoid or tanh, small gradients) or overflow (for ReLU activation).
- If they are too small, learning may take too many iterations to converge.

Other "Tricks"

Stochastic gradient descent is the simplest learning algorithm for neutral networks, but there are many other choices:

- Use adaptive learning rates for each parameter
- In practice, most implementations clip gradient to some maximum magnitude before making updates

Early stopping: check performance on a development set, and stop training when performances starts to get worst.

Neural Architectures for Sequence Data

Text is naturally viewed as a sequence of tokens $w_1, w_2, \ldots, w_M$

- Context is lost when this sequence is converted to a bag-of-words
- Instead, a lookup layer can compute embeddings for each token, resulting in a matrix $\mathbf{X}^{(0)} = \mathbf{\Theta}^{(x \to z)}[\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \ldots, \boldsymbol{e}_{w_M}]$, where $\mathbf{X}^{(0)} \in \mathbb{R}^{K_e \times M}$
- Higher-order representations $\mathbf{X}^{(d>0)}$ can then be computed from $\mathbf{X}^{(0)}$

Convolutional Neural Networks

Convolutional neural networks compute successively higher representations $\mathbf{X}^{(d)}$

by convolving $\mathbf{X}^{(d-1)}$ with a set of local filter matrices $\boldsymbol{C}$

$$\mathbf{X}^{(1)} = f(\boldsymbol{b} + \mathbf{C} * \mathbf{X}^{(0)}) \quad \Longrightarrow \quad x_{k,m}^{(1)} = f\left(b_k + \sum_{k'=1}^{K_e} \sum_{n=1}^{h} c_{k',n}^{(k)} \times x_{k',m+n-1}^{(0)}\right)$$

f is a non-linear activation function

h is the filter size, K_e is the size of word embedding

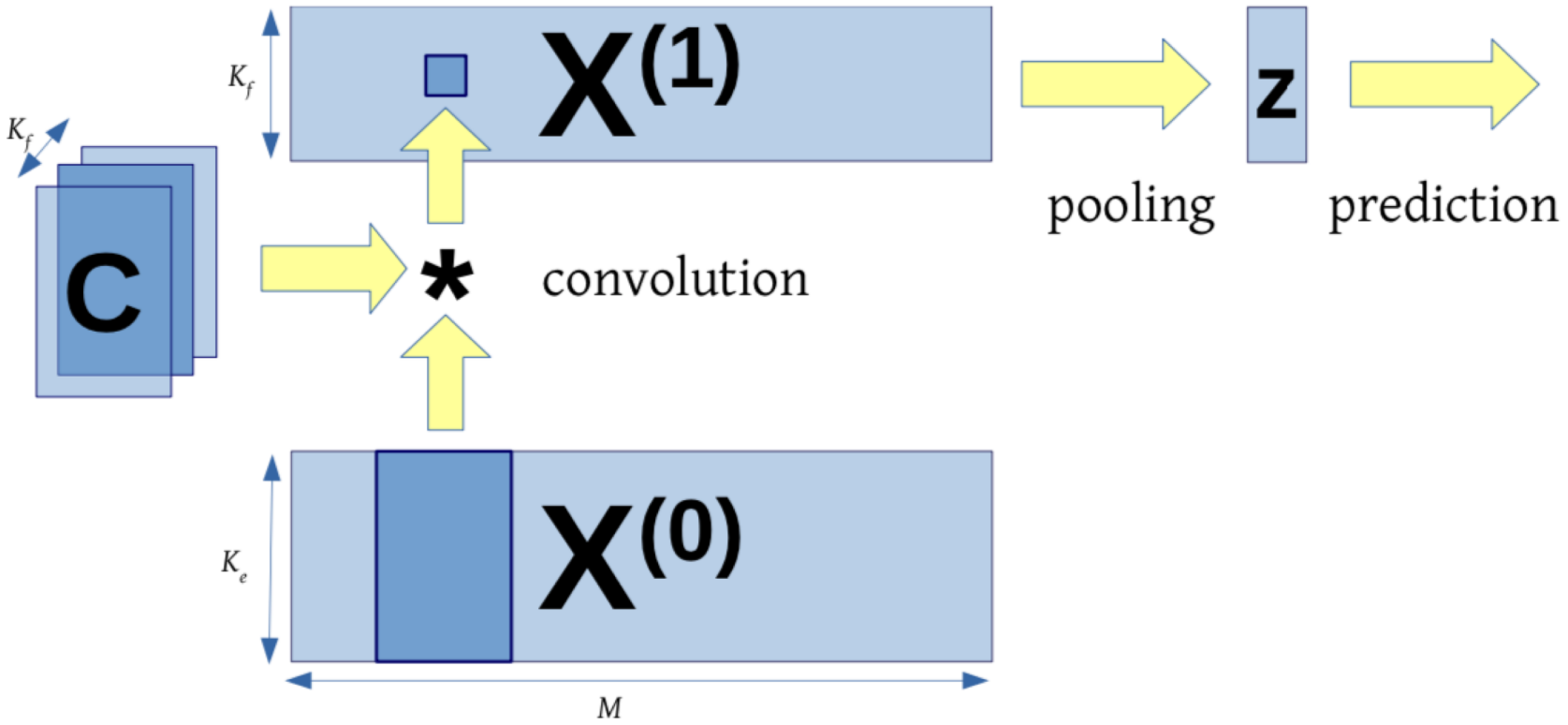
the filter parameters c are learned from data

Convolutional Neural Networks

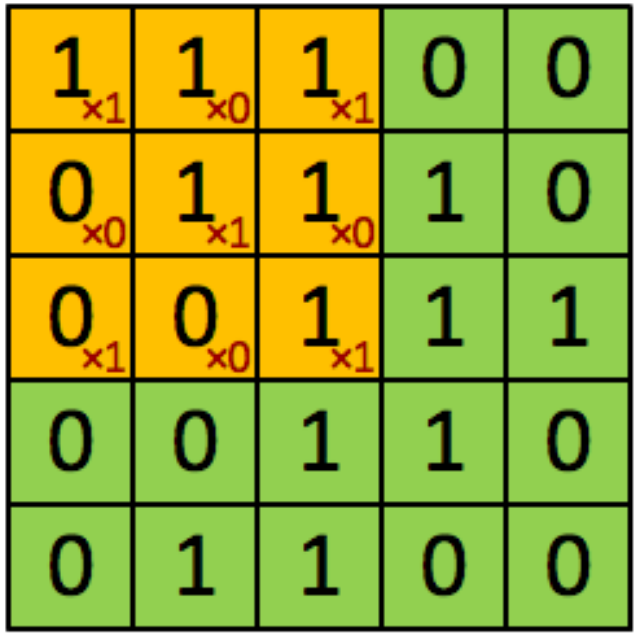
Convolutional neural networks compute successively higher representations $\mathbf{X}^{(d)}$ by convolving $\mathbf{X}^{(d-1)}$ with a set of local filter matrices $\boldsymbol{C}$

In this way, each $x_{k,m}^{(d)}$ is **a function of locally adjacent features at the previous level.**

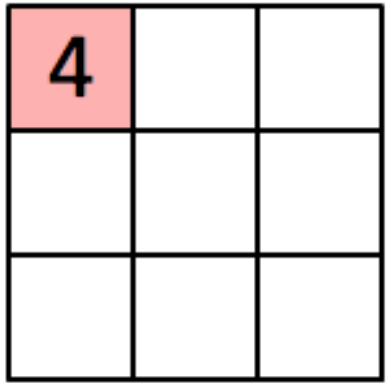
Convolutional Neural Networks



Convolutional Neural Networks



Input Feature



Convoluted Feature

Pooling in CNN

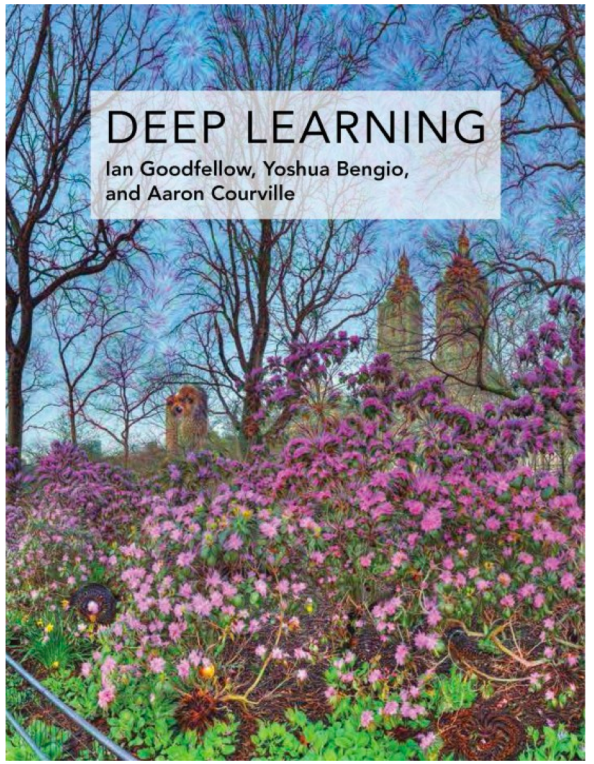
$$\boldsymbol{z} = \text{MaxPool}(\mathbf{X}^{(D)}) \quad \Longrightarrow \quad z_k = \max\left(x_{k,1}^{(D)}, x_{k,2}^{(D)}, \ldots x_{k,M}^{(D)}\right)$$

$$\boldsymbol{z} = \text{AvgPool}(\mathbf{X}^{(D)}) \quad \Longrightarrow \quad z_k = \frac{1}{M}\sum_{m=1}^{M} x_{k,m}^{(D)}.$$

Additional Resources on CNN

Text Understanding from Scratch

Xiang Zhang XIANG@CS.NYU.EDU
Yann LeCun YANN@CS.NYU.EDU
Computer Science Department, Courant Institute of Mathematical Sciences, New York University



Other Neural Architecture

- CNN are sensitive to local dependencies between words
- In **Recurrent Neural Networks**
  - A model of context is constructed while processing the text from left-to-right. Those networks are theoretically sensitive to global dependencies
  - LSTM, Bi-LSTM, GRU, Bi-GRU, …

Text Classification Applications & Evaluation

Text Classification Applications

- Classical Applications of Text Classification
  - Sentiment and opinion analysis
  - Word sense disambiguation
- Design decisions in text classification
- Evaluation

Sentiment Analysis

- The **sentiment** expressed in a text refers to the author's subjective or emotional attitude towards the central topic of the text.
- Sentiment analysis is a classical application of text classification, and is typically approached with a bag-of-words classifier.

Beyond the Bag-of-words

Some linguistic phenomena require going beyond the bag-of-words:

- *That's not bad for the first day*
- *This is not the worst thing that can happen*
- *It would be nice if you acted like you understood*
- *This film should be brilliant. The actors are first grade. Stallone plays a happy, wonderful man. His sweet wife is beautiful and adores him. He has a fascinating gift for living life fully. It sounds like a great plot, however, the film is a failure.*

Related Classification Problems

Subjectivity: Does the text convey factual or subjective content?

Related Classification Problems

Subjectivity: Does the text convey factual or subjective content?

Stance Classification: Given a set of possible positions, or stances, which is being taken by the author?

Targeted Sentiment Analysis: What is the author's attitude towards several different entities?

- *The vodka was good, but the meat was rotten.*

Emotion Classification: Given a set of possible emotional states, which are expressed by the text?

Word Sense Disambiguation

Consider the following headlines:

- Iraqi **head** seeks arms
- Drunk gets nine years in violin **case**

Word Senses

Many words have multiple senses, or meanings. For example, the verb appeal has the following senses:

Appeal	take a court case to a higher court for review
Appeal, invoke	request earnestly (something from somebody)
Attract, appeal	be attractive to

http://wordnetweb.princeton.edu/perl/webwn?s=appeal

Word Senses

Many words have multiple senses, or meanings.

- Word sense disambiguation is the problem of identifying the intended word sense in a given context.
- More formally, senses are properties of lemmas (uninflected word forms), and are grouped into synsets (synonym sets). Those synsets are collected in WORDNET.

Word Sense Disambiguation as Classification

How can we tell living *plants* from manufacturing plants? Context.

- *Town officials are hoping to attract new manufacturing plants through weakened environmental regulations.*
- *The endangered plants play an important role in the local ecosystem.*

Applying Text Classification

- The “raw” form of text is usually a sequence of characters
- Converting this into a meaningful feature vector x requires a series of design decisions, such as tokenization, normalization, and filtering

Tokenization

- Tokenization is the task of splitting the input into discrete tokens
- This may seem easy for Latin script languages like English, but there are some tricky parts. How many tokens do you see in this example?
- *O'Neill's prize-winning pit bull isn't really a "bull".*

Four English Tokenizers

- Input: *Isn't Ahab, Ahab? ;)*

Whitespace	Isn't	Ahab,	Ahab?	;)				
Treebank	Is	n't	Ahab	,	Ahab	?	;	)
Tweet	Isn't	Ahab	,	Ahab	?	;)		
TokTok	Isn	'	t	Ahab	,	Ahab	?	;	)

Tokenization in Other Scripts

- Some languages are written in scripts that do not include whitespace. Chinese is a prominent example.
- Tokenization can usually be solved by matching character sequences against a dictionary, but some sequences have multiple possible segmentations.

(1) 日文 章魚 怎麼 説?
Japanese octopus how say
How to say octopus in Japanese?

(2) 日 文章 魚 怎麼 説?
Japan essay fish how say

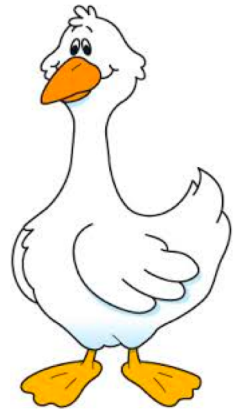
Normalization

- Distinctions with a difference?
  - *apple* vs *apples*
  - *apple* vs *Apple*
  - *1,000* vs *1000* vs *one thousand*
  - *soooooooooooo* vs *so*
  - *Aug 20* vs *August 20* vs *8/20* vs *20 August*

Normalization

- Distinctions with a difference?
  - *apple* vs *apples*
  - *apple* vs *Apple*
  - *1,000* vs *1000* vs *one thousand*
  - *soooooooooooo* vs *so*
  - *Aug 20* vs *August 20* vs *8/20* vs *20 August*

- More aggressive ways to group words:
  - **Stemming:** removing inflectional affixes, *whales* → *whale*
  - **Lemmatization:** converting to a base form, *geese* → *goose*.



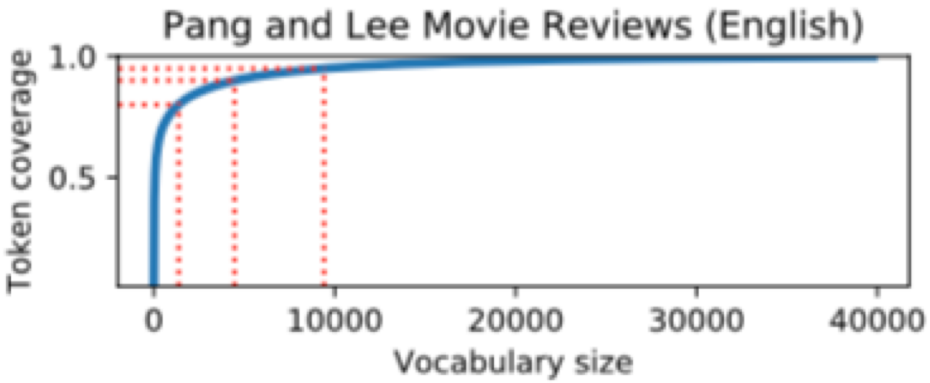
Three English Stemmers

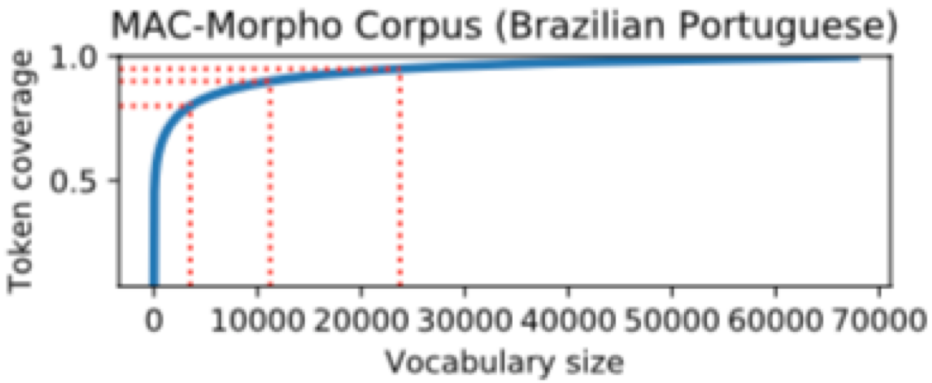
Original	The	Williams	sisters	are	leaving	this	tennis	centre
Porter	the	william	sister	are	leav	thi	tenni	centr
Lancaster	the	william	sist	ar	leav	thi	ten	cent
WordNet	The	Williams	sister	are	leaving	this	tennis	centre

- Stemming and lemmatization rarely help supervised classification, but can be useful for string matching and unsupervised learning.

Vocabulary Size Filtering

A small number of word **types** accounts for the majority of word **tokens**.





- The number of parameters in a classifier usually grows linearly with the size of the vocabulary
- It can be useful to limit the vocabulary, e.g., to word types appearing at least x times, or in at least y% of documents.

Evaluating Your Classifier

Goal is to predict **future** performance, on unseen data.

- It is hard to predict the future.
- Do not evaluate on data that was already used …
  - For training
  - For hyperparameter selection
  - For selecting the classification model or model structure
  - For making preprocessing decisions, such as vocabulary selection.

Evaluating Your Classifier

Goal is to predict **future** performance, on unseen data.

- Even if you follow all those rules, you will still probably overestimate your classifier's performance, because real future data will differ from your test set in ways that you cannot anticipate.

Accuracy

Most basic metric is **accuracy**: how often is the classifier right?

$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N} \sum_{i=1}^{N} \delta(y^{(i)} = \hat{y})$$

Accuracy

Most basic metric is **accuracy**: how often is the classifier right?

$$\text{acc}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{i=1}^{N}\delta(y^{(i)} = \hat{y})$$

The problem with accuracy is **rare labels**.

- Consider a system for detecting tweets written in Telugu.
- 0.3% of Tweets are written in Telugu.
- A system that always says "Not Telugu" is 99.7% accurate.

Beyond Right and Wrong

For any label, there are two ways to be wrong:

- False positive: the system incorrectly predicts the label
- False negative: the system incorrectly fails to predict the label.

Similarly, there are two ways to be right:

- True positive: the system correctly predicts the label
- True negative: the system correctly predicts that the label does not apply to it.

Recall

$$r = \frac{TP}{TP+FN}$$

- Recall is the fraction of positive instances which were correctly classified.
- The “never Telugu” classifier has zero recall.
- An “always Telugu” classifier would have perfect recall.

Precision

$$p = \frac{TP}{TP+FP}$$

- Precision is the fraction of positive **predictions** that were correct.
- The "never Telugu" classifier has precision $\frac{0}{0}$.
- An "always Telugu" classifier would have precision p=0.003, which is the rate of Telugu tweets in the dataset.

Combining Recall and Precision

- In binary classification, there is an inherent tradeoff between recall and precision.
- The correct navigation of this tradeoff is problem-specific!
  - For a preliminary medical diagnosis, we might prefer high recall. False positives can be screened out later.
  - The “beyond a reasonable doubt” standard of U.S. criminal law implies a preference for high precision.

Combining Recall and Precision

- In binary classification, there is an inherent tradeoff between recall and precision.
- The correct navigation of this tradeoff is problem-specific!

- If recall and precision are weighted equally, they can be combined into a single number called F-measure

$$F = \frac{2 \times r \times p}{r + p}$$

Evaluating Multi-Class Classification

- Recall and precision imply binary classification: each instance is either positive or negative.
- In multi-class classification, each instance is positive for one class, and negative for all other classes.

Evaluating Multi-Class Classification

- Two ways to combine performance across classes:
  - **Macro F-measure:** compute the F-measure per class, and average across all classes. This treats all classes equally, regardless of their frequency.
  - **Micro F-measure:** compute the total number of true positives, false positives, and false negatives across all classes, and compute a single F-measure. This emphasizes performance on high-frequency classes.

Comparing Classifiers

- Suppose you and your friend build classifiers to solve a problem:
  - *You classifier C_1 get 82% accuracy*
  - *You friend's classifier C_2 get 73% accuracy*

- Will C_1 be more accurate in the **future**?

Comparing Classifiers

- Suppose you and your friend build classifiers to solve a problem:
  - *You classifier C_1 get 82% accuracy*
  - *You friend's classifier C_2 get 73% accuracy*

- Will C_1 be more accurate in the **future**?
  - What is the test set had 10000 examples?
  - What is the test set had 11 examples?

Getting Labels

Text classification relies on large datasets of labeled examples. There are two main ways to get labels:

- **Metadata** sometimes tell us exactly what we want to know: Did the Senator vote for a bill? How many stars did the reviewer give? Was the request for free pizza accepted?
- Other times, the labels must be **annotated**, either by experts or by "crow-workers"

Labeling Data

1. Determine what to annotate
2. Design or select a software tool to support the annotation effort
3. Formalize the instructions for the annotation task
4. Prepare for a pilot annotation
5. Annotate the data
6. Compute and report inter-annotator agreement
7. Release the data

Next Class

Language Modeling