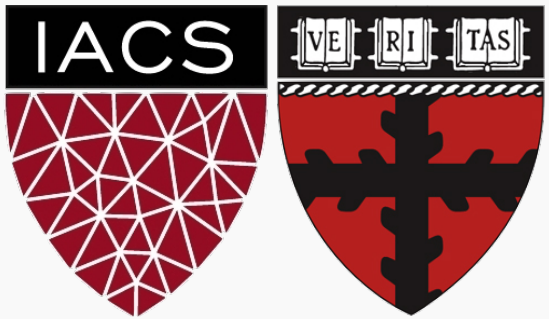
An Introduction to Natural Language Processing for Sentiment Analysis

ComputeFest
Cambridge, MA
Winter 2019



Workshop Outline

Overview of NLP

Sentiment Analysis: 1st Pass

2nd Pass: Neural Networks

3rd Pass: Everything Fancier

References

Overview of NLP

NLP Tasks and Applications

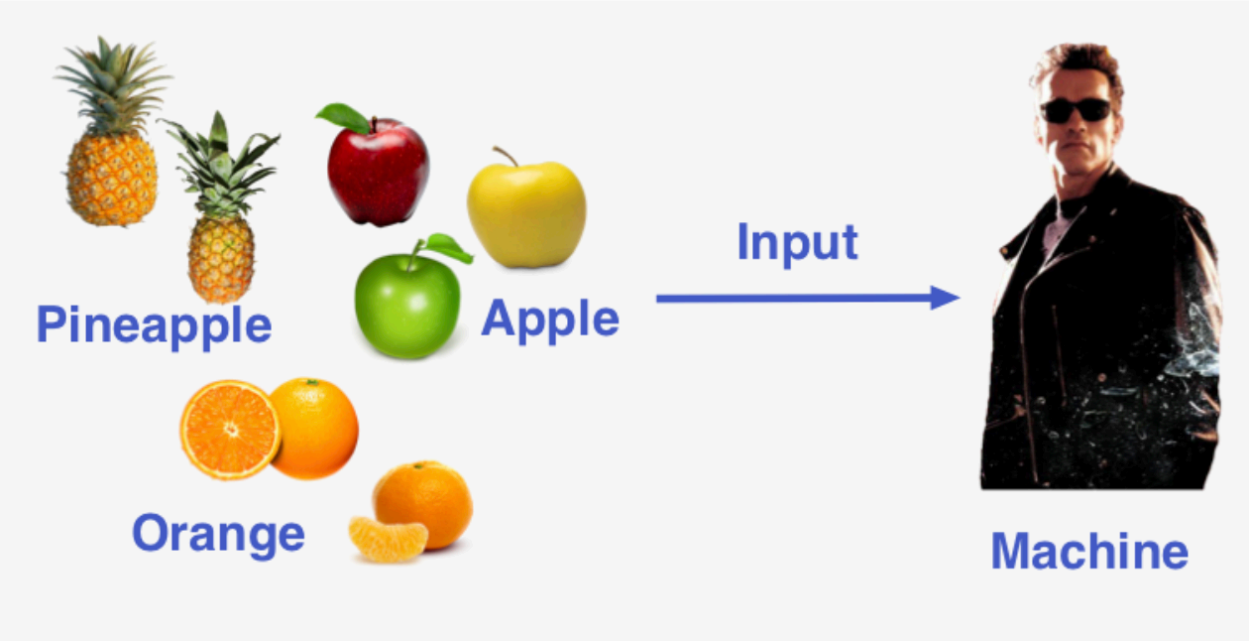
Tasks: do this…

1. Parts of speech tagging
2. Named entity recognition
3. Parsing/stemming

Applications: in order to…

1. Document classification: spam, sentiment, etc
2. Machine translation
3. Conversational agents

Levels of Linguistic Knowledge



Main Challenges

1. Ambiguity all all levels: ‘I made her duck’, ‘I went to the bank…’
2. Language changes through time, across domains
3. Many rare words

An Overview of Approaches

Sentiment Analysis: 1st Pass

Representing Textual Data

Comparing the content of the following two sentences is easy for an English speaking human (clearly both are discussing the same topic, but with different emotional undertone):

1. Linear R3gr3ssion is very very cool!
2. What don't I like it a single bit? Linear regressing!

Representing Textual Data

Comparing the content of the following two sentences is easy for an English speaking human (clearly both are discussing the same topic, but with different emotional undertone):

1. Linear R3gr3ssion is very very cool!
2. What don't I like it a single bit? Linear regressing!

But a computer doesn't understand

- which words are nouns, verbs etc (grammar)
- how to find the topic (word ordering)
- feeling expressed in each sentence (sentiment)

We need to represent the sentences in formats that a computer can easily process and manipulate.

Preprocessing

If we're interested in the topics/content of text, we may find many components of English sentences to be uninformative.

1. Word ordering
2. Punctuation
3. Conjugation of verbs (go vs going), declension of nouns (chair vs chairs)
4. Capitalization
5. Words with mostly grammatical functions: prepositions (before, under), articles (the, a, an) etc
6. Pronouns?

These uninformative features of text will only confuse and distract a machine and should be removed.

NPL Pipeline

1. Language detection
2. Text cleanup (boilerplate removal / normalization / OCR-error correction, etc)
3. Sentence segmentation
4. Tokenization
5. Morphological processing: stemming
6. POS tagging
7. Morphological processing: lemmatization
8. Word-level (lexical) semantics
9. Syntactic processing: parsing
10. Sentence-level semantics
11. Discourse semantics and pragmatics

Representing Documents: Bag Of Words

After preprocessing our sentences:

1. ($\mathbf{S_1}$) linear regression is very very cool
2. ($\mathbf{S_2}$) what don't like single bit linear regression

We represent text in the format that is most accessible to a computer: numeric. We simply make a vector of the **counts** of the words in each sentence.

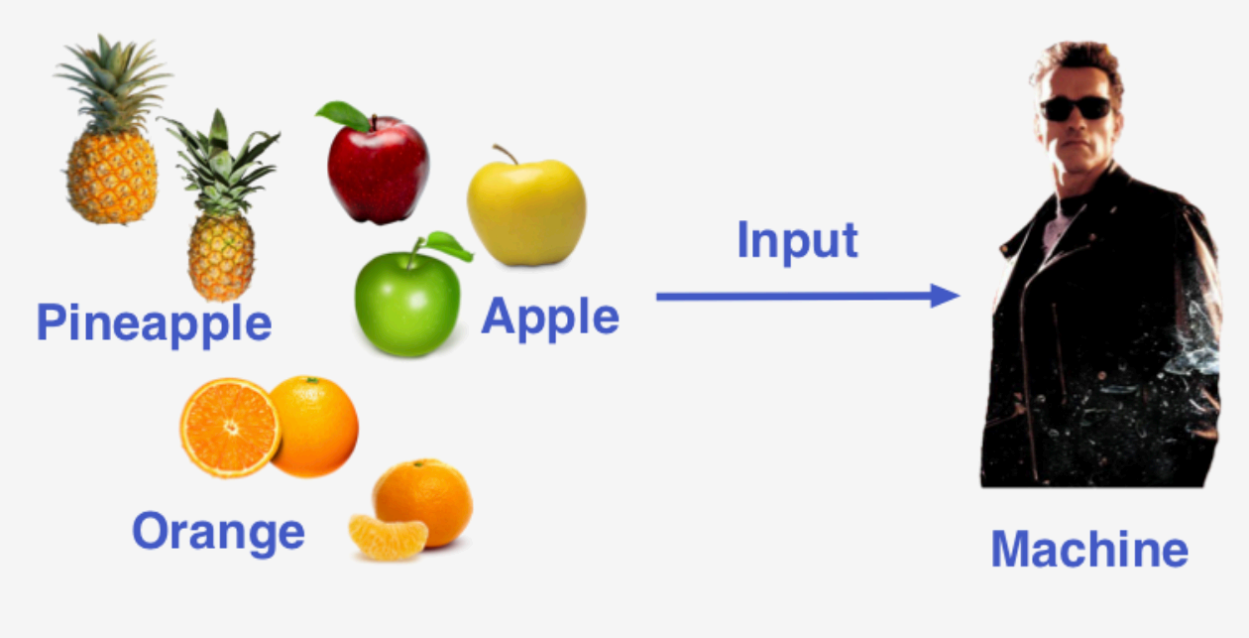
	linear	regression	is	very	cool	what	don't	like	single	bit
S_1	1	1	1	2	1	0	0	0	0	0
S_2	1	1	0	0	0	1	1	1	1	1

Turning a piece of text into a vector of word counts is called **Bag of Words**.

We are now ready to feed our vectorized **corpus** of texts into a machine learning model for **classification**.

Machine Learning Background

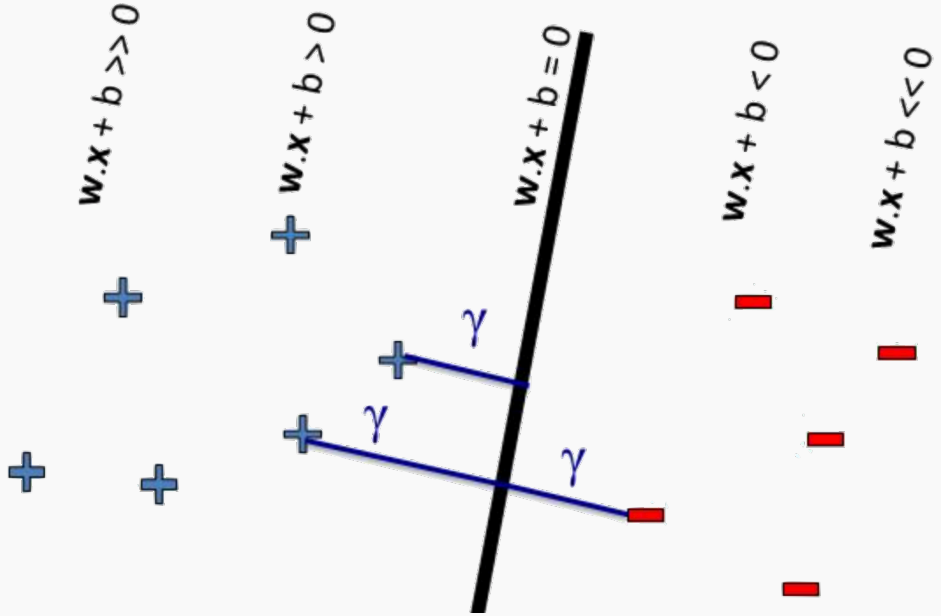
Supervised learning: inferring functions from labeled data



Given a set of values of predictors $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $x_n \in \mathbb{R}^D$ and corresponding responses $\mathbf{y} = \{y_1, \ldots, y_N\}$, learn function $f_w : \mathbb{R}^D \to \mathbb{R}$, with parameters w.

Classification: Logistic Regression

Model: Model the probability of an input being a positive instance as a function of its distance from the hyperplane



For example,

$$p(y = 1|\mathbf{w}, \mathbf{x}) = \text{sigm}(\mathbf{w}^\top \mathbf{x})$$

This model is ***logistic regression***.

Putting it Together

Supervised Learning: given a **training dataset** of N labeled docs

$$\mathbf{X} \in \mathbb{R}^{N \times V}, \ \mathbf{y} \in \{0, 1\}^N$$

find values for the parameters, **w**, of the model that maximizes a ***model fitness metric***.

In this case, the ***likelihood***, $\mathcal{L}$, of the training data

$$\mathbf{w}_{\text{MLE}} = \underset{\mathbf{w}}{\text{argmax}} \ \mathcal{L}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \underset{\mathbf{w}}{\text{argmax}} \prod_n p(y_n | \mathbf{w}, \mathbf{x}_n)$$

This optimization problem is convex and typically solved via ***gradient descent***.

Putting it Together

Inference: After learning the model, we can infer the labels of new images by computing

$$p(y_{\text{test}} = 1 | \mathbf{w}_{\text{MLE}}, \mathbf{x}_{\text{test}})$$

We can decide to label the the instance **x** as 1 if, say

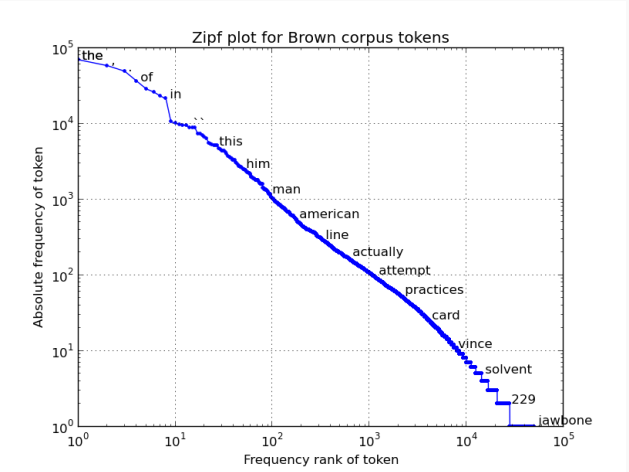
$$p(y_{\text{test}} = 1 | \mathbf{w}_{\text{MLE}}, \mathbf{x}_{\text{test}}) > 0.5$$

Exercise

Try classifying a corpus of tweets as positive or negative using BoW and Logistic Regression.

Draw Backs of the BoW Representation

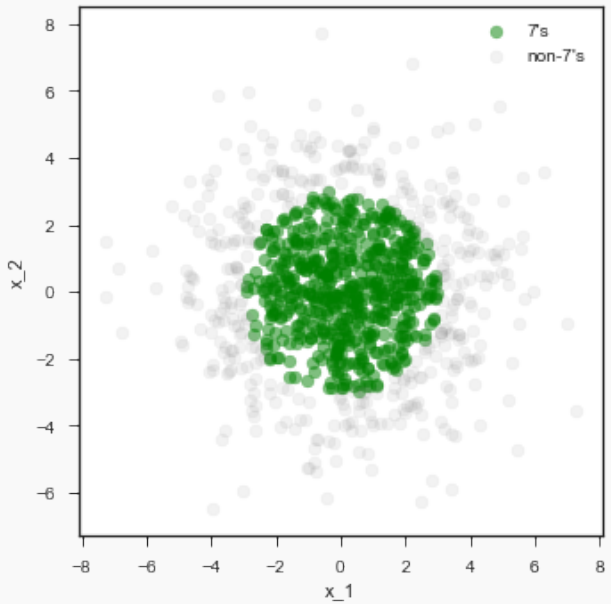
1. Count vector representation doesn't capture sequential information (e.g. context). What if we use n-grams (chunks of n-consecutive words)?
2. The vector will be inherently sparse and high dimensional



3. The numerical representation captures no semantic meaning

Draw Backs of Logistic Regression

Assume that the data is ***not*** linearly separable in the input space,



In fact, the probability of an document being a positive instance is a non-linear function, g, of the input

$$p(y = 1|\mathbf{W}, \mathbf{x}) = g(\mathbf{W}, \mathbf{x})$$

Draw Backs of Logistic Regression

Given an exact parametrization, we could learn g directly, just as before. However, assuming an exact for for g is restrictive.

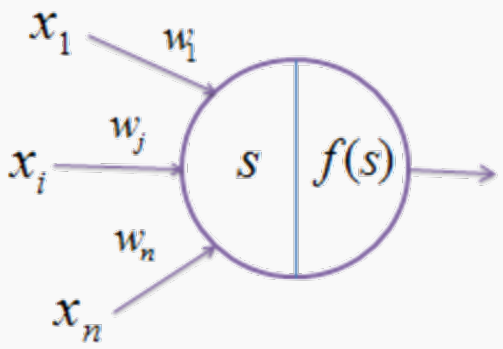
Rather, we can build increasingly good approximations, $\hat{g}$, of g by composing simple functions.

2nd Pass: Neural Networks

What is a Neural Network?

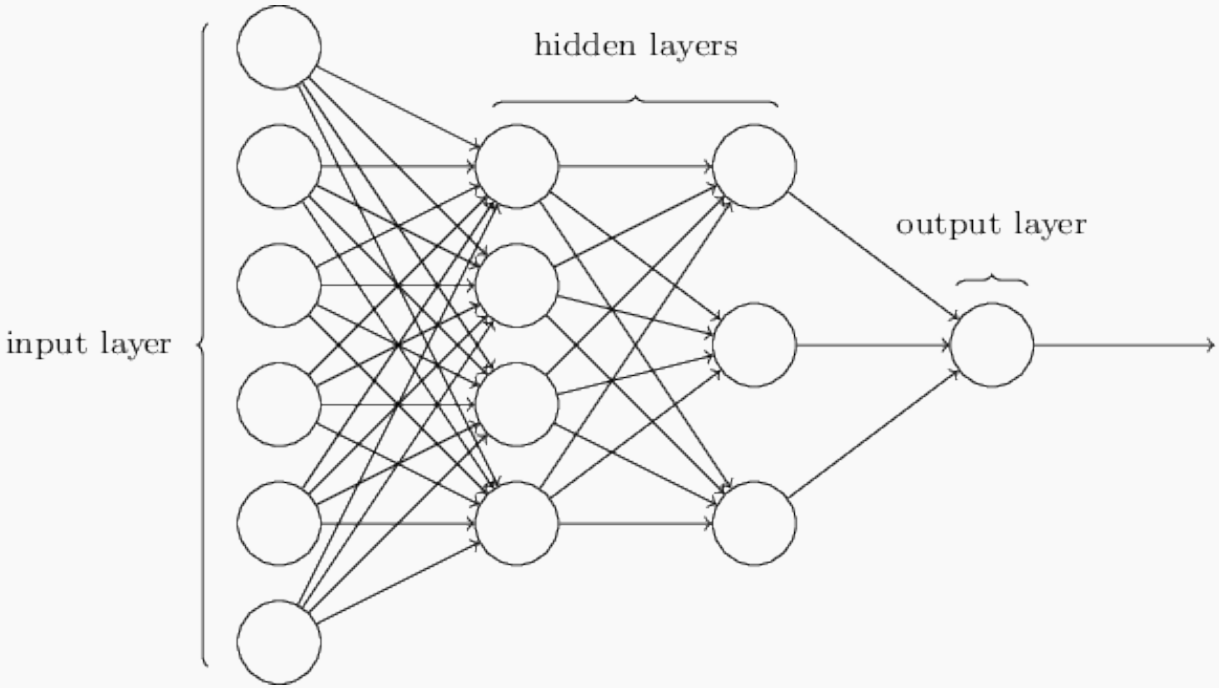
Goal: build an approximation $\hat{g}$ of a function g by composing simple functions.

For example, let the following picture represents $f\left(\sum_i w_i x_i\right)$, where f is a non-linear transform



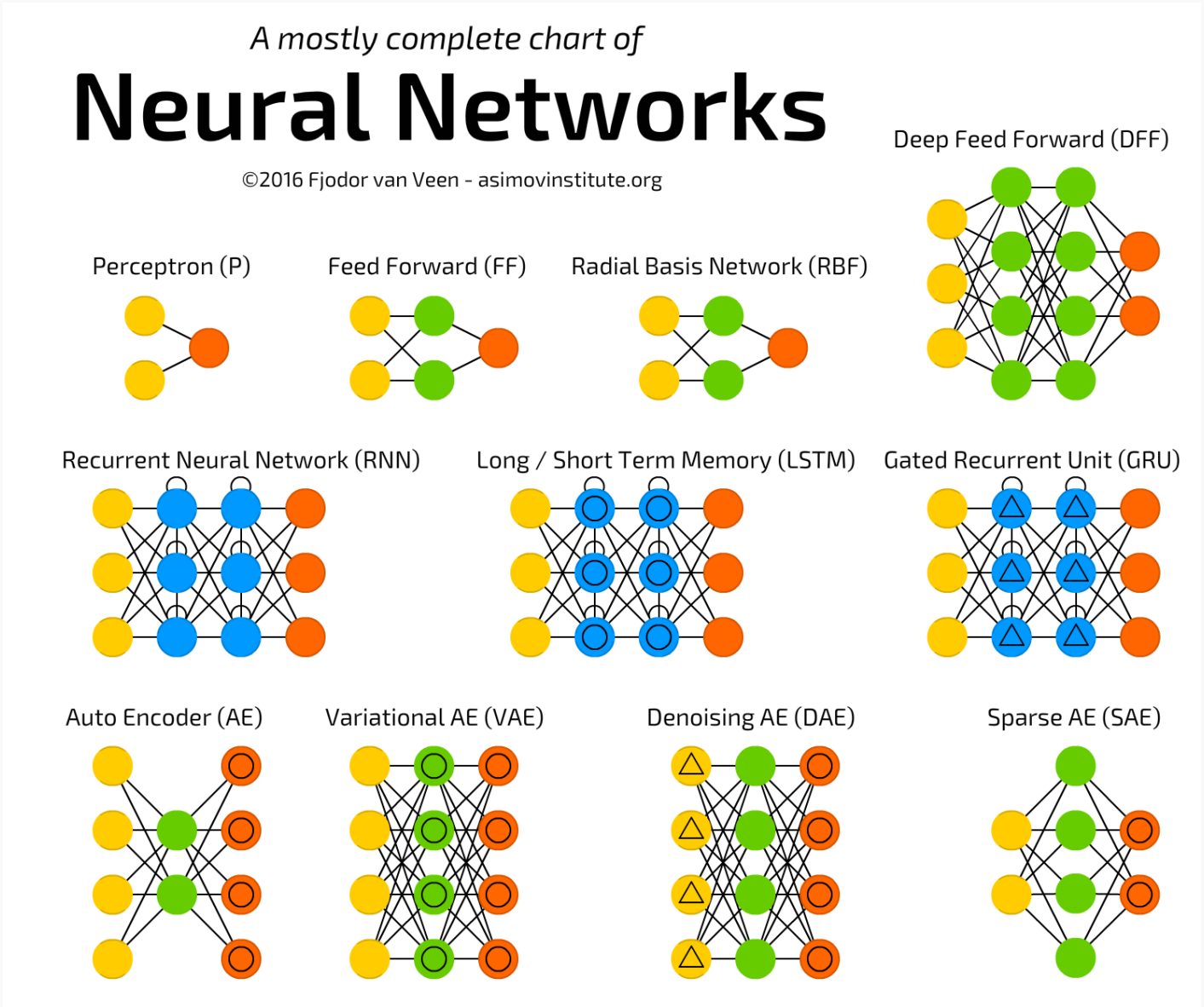
What is a Neural Network?

Then we can define $\hat{g}$ with a graphical schema representing a complex series of compositions and sums of the form, $f\left(\sum_i w_i x_i\right)$



This is a ***neural network***. We denote the weights of the neural network collectively by **W**.

Flexible Framework for Function Approximation



Neural Network for Classification

Supervised Learning: given a **training dataset** of N labeled images

$$\mathbf{X} \in \mathbb{R}^{N \times V}, \ \mathbf{y} \in \{0, 1\}^N$$

find values for the parameters, **W**, of the NN that maximizes a ***model fitness metric***.

In this case, we can again choose the likelihood on the training data

$$\mathbf{W}_{\text{MLE}} = \underset{\mathbf{w}}{\text{argmax}} \ \mathcal{L}(\mathbf{W}, \mathbf{X}, \mathbf{y}) = \underset{\mathbf{w}}{\text{argmax}} \prod_n p(y_n | \hat{g}(\mathbf{W}, \mathbf{X})$$

A method of 'gradient descent' called ***back-propagation*** can be used.

BackProp: An Intuition

The intuition behind various flavours of gradient descent is as follows:

BackProp: An Intuition

The intuition behind various flavours of gradient descent is as follows:

1. start at random place: $W_0 \leftarrow$ **random**
2. until (stopping condition satisfied):
   - 2.1 compute gradient:

     gradient = ∇ loss_function(W_t)
   - 2.2 take a step in the negative gradient direction:

     $W_{t+1} \leftarrow W_t$ - eta * gradient

Word Embeddings

Goal: dense numerical representations of words that capture semantic meaning.

Formally, a **word embedding** is a function, with parameters θ,

$$E_\theta : \mathbb{R}^V \to \mathbb{R}^K$$

mapping one-hot encoded words to a much smaller space of dimension K.

Neural Network for Word Embeddings

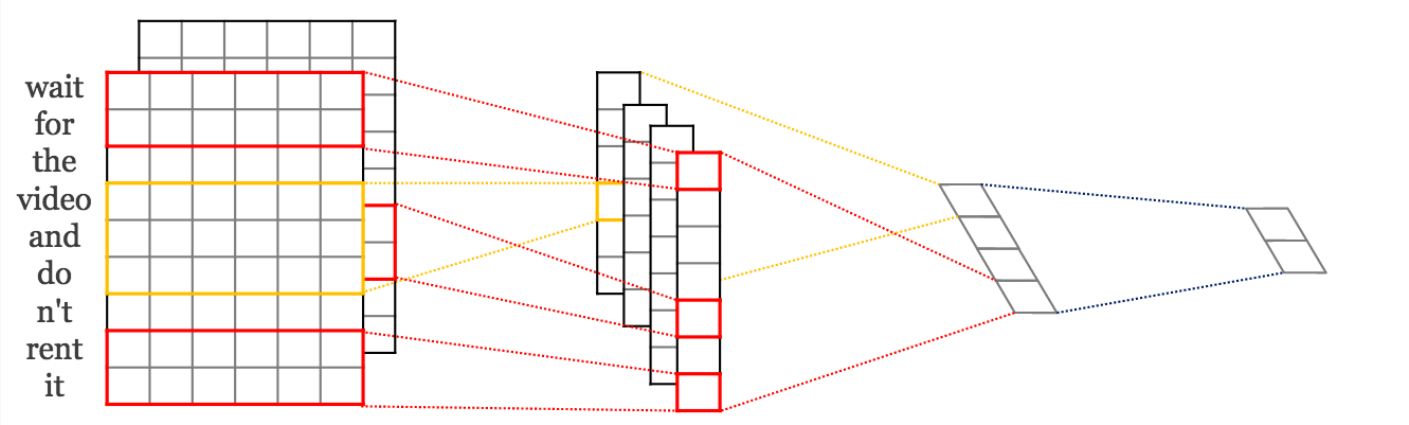
We can (but don't have to) represent the embedding E_θ using a neural network and find its optimal parameters θ using gradient descent.

Question: But what should the training objective be?

Answer: Formalize your specific task as an loss function.

Putting it Together

A full pipeline for sentiment classification using neural networks:



Exercise

Build a word embedding and a classifier for the tweet corpus using neural networks.

Not All Embeddings are Equal

How do we evaluate the quality of a word embedding?

- similar words should have ‘similar’ representations?
- good for a down stream task?
- relationship between words or relationship between data?
- rare words?

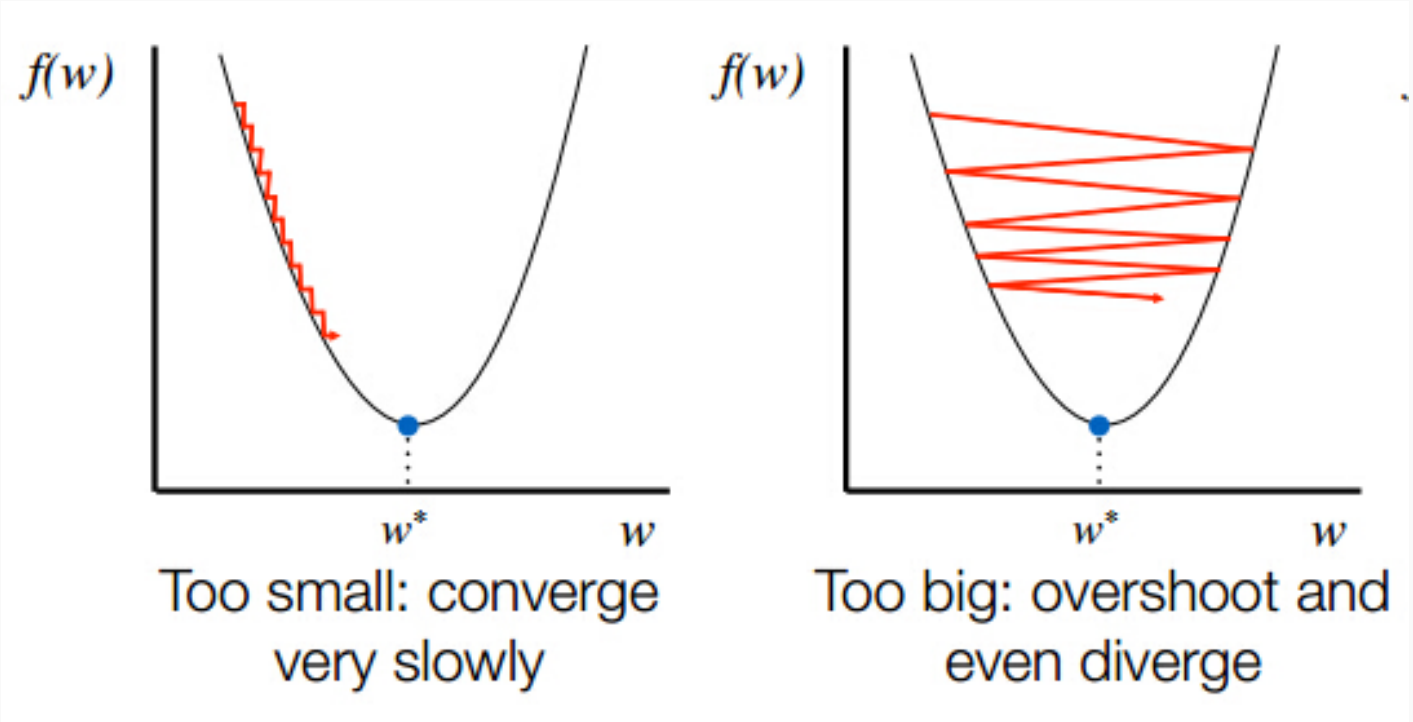
Expressive Models Can Be Hard to Train

Your training algorithms for NN's do not always end up with the same solution. Stochasticity is introduced in the learning process in two places:

1. the optimization is potentially not exact (we use stochastic gradients)
2. the objective is generally non-convex (i.e. there are possibly many 'pretty good' solutions). The solution you find will depend on initialization.

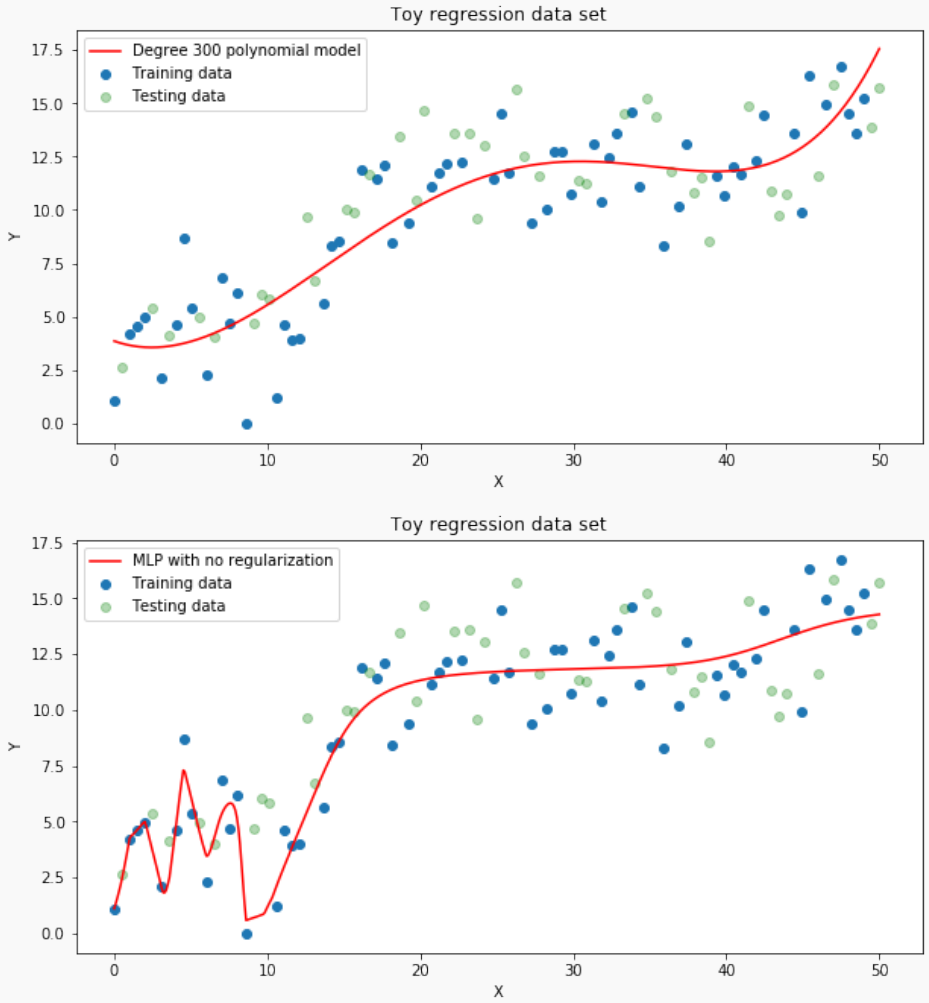
Expressive Models Can Be Hard to Train

Tuning the settings in your gradient descent method is important:



Expressive Models Can Be Hard to Train

Complex models can overfit:



Expressive Models Can Be Hard to Train

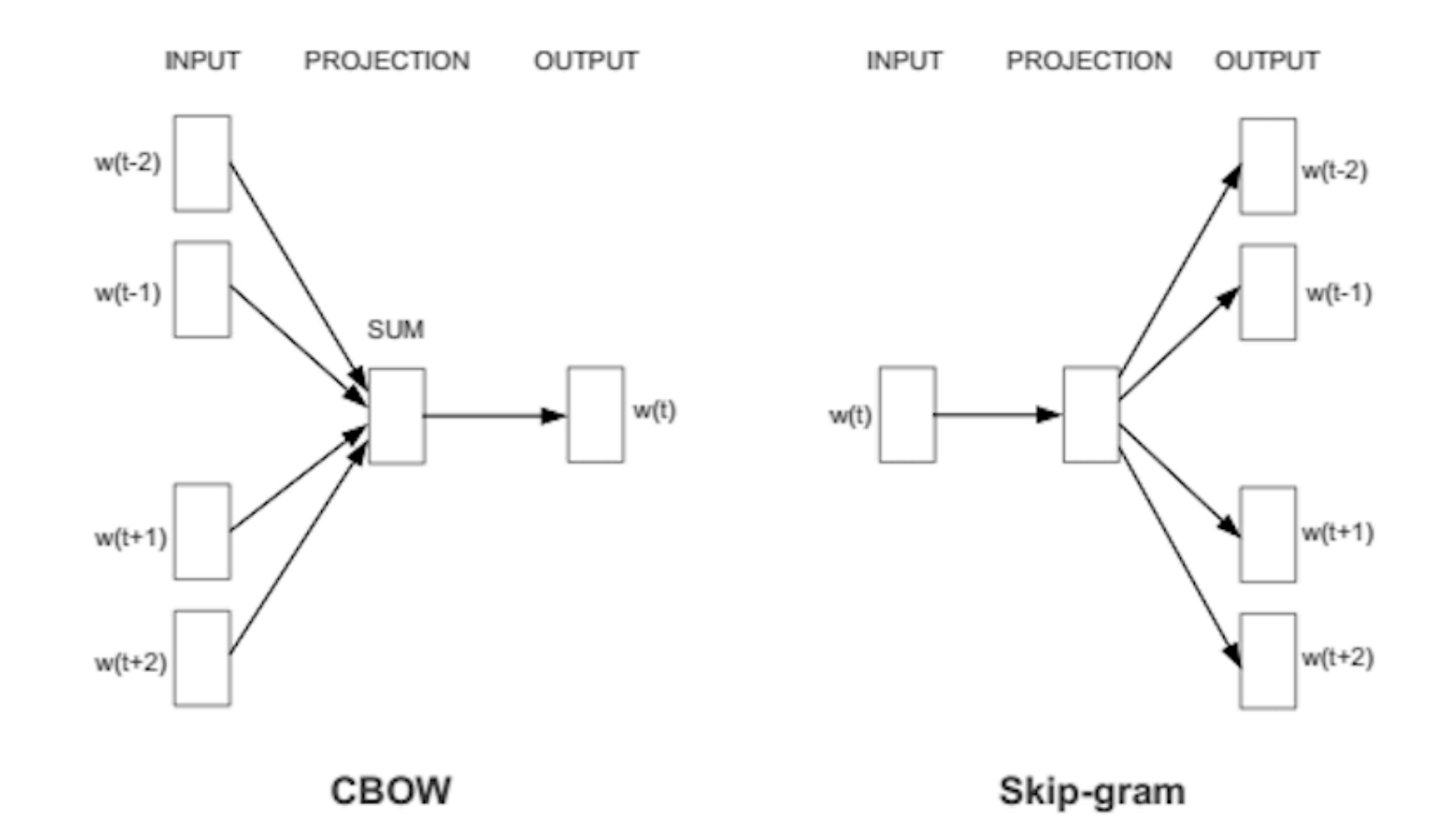
Regularization:

1. ℓ_p regularization
2. Dropout, Swapout
3. Ensemble
4. Bayesian Neural Networks

3rd Pass: Everything Fancier

Word2Vec

Intuition: Train embedding using context to predict a target word, or using a word to predict a target context.



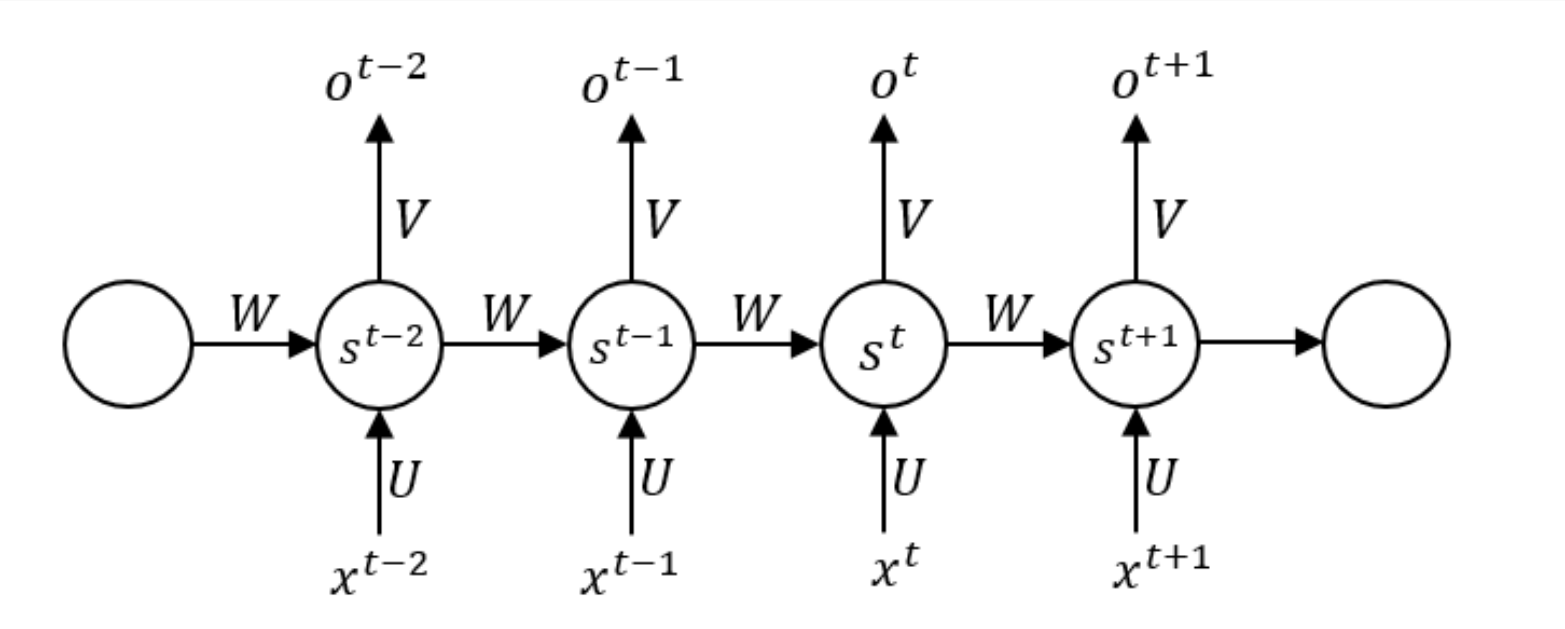
GloVe

Intuition: Ratios of word-word co-occurrence probabilities can encode meaning.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

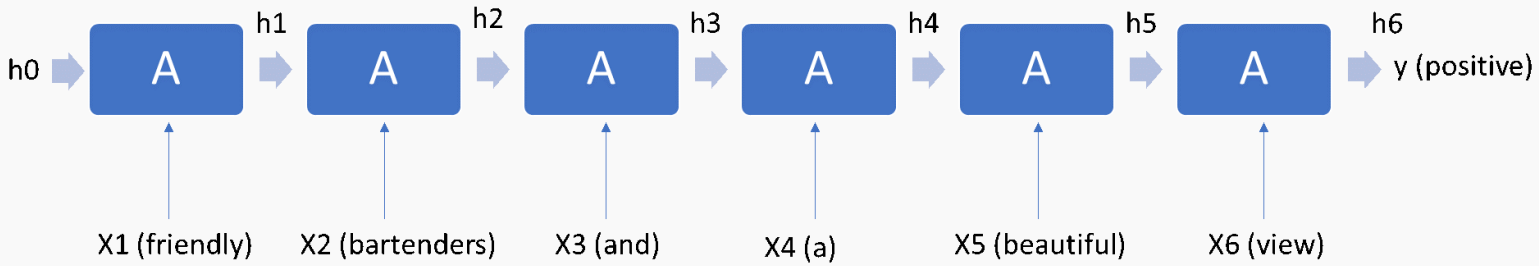
Learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence (matrix factorization). This associates the log ratios of co-occurrence probabilities with vector differences in the word vector space.

Recurrent Neural Networks



Recurrent Neural Networks

An RNN can analyze varying length input:



Problems with RNN

Observation: a recurrent neural network is just a very deep regular (feed-forward) neural network!

A Problem: for most activation functions, the gradients at each activation is bounded by 1, thus the application of the chain rule during BackProp produces the vanishing gradient problem.

RNN's don't have 'long term memory'.

References

Papers

1. A Survey of the Usages of Deep Learning in Natural Language Processing
2. Advances in natural language processing
3. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings
4. GloVe: Global Vectors for Word Representation
5. Natural Language Processing - Jacob Eisenstein