Neural Network based LMs

Char-RNN

Announcement: Parts 4+5 of HW5 will now be part of HW6. Parts 1-3 of HW5 are due on Wednesday.

Goal of language modeling:

- Learn a function that returns the joint probability of a sequence of words

Primary difficulties:

1. There are too many parameters to accurately estimate. This is sometimes called the "curse of dimensionality"
2. In n-gram based models, we fail to generalize to related words / word sequences that we have observed.

1. Curse of dimensionality / sparse statistics

Suppose we want a joint distribution over 10 words

" have a vocab of size 100,000

$100{,}000^{10} = 10^{50}$ parameters — too high to estimate from data

In LMs we use chain rule to get the conditional probability of next word in sequence given all of the previous words

$$p(w_1\ w_2\ w_3 \dots w_T) = \prod_{t=1}^{T} p(w_t \mid w_1 \dots w_{t-1})$$

What assumption do we make in ngram LMs to simplify this? Probability of the next word only depends on previous N-1 words ← get an estimate

to simplify this? Probability of the next word only depends on previous N-1 words ← get an estimate for small N from data

We construct tables to look up the probability of seeing a word given a history

history (N-1)	W_T	$p(W_T \mid W_{T-N} \dots W_{T-1})$
curse of	dimensionality	—
	azure	—
	oak	—
	knowledge	—

"STUPID" multinomial distributions == Table lookup.

What happens when we have a new (unseen) combination of N words. The table only stores observed sequences

1. Back-off
2. Smoothing / interpolation

We are basically just stitching together short sequences of observed words.

Alternate idea

Let's try generalizing.

Intuition: take a sentence like

The cat is walking in the bedroom

and use it when we assign probabilities to a similar sentence like

The dog is running around a room.

sim(cat, dog)

Bengio et al 2003 NIPS paper "A Neural Probabilistic Language model".

$M = 30, 60, 100$

① use a vector space model where words are vectors w/ real values $\mathbb{R}^M$. Gives them a way to compute word similarity

prob. LM

② Define a fn that returns a joint probability of words in a sequence based on a sequence of these vectors

③ Simultaneously learn the word representations and the probability fn from data.

Seeing one of the cat/dog sentences allows them to increase the probability for that sentence and its combinatorial # of "neighbor" sentences in vector space.

A neural network architecture for prob. LMs.

Given: a training set $w_1 \dots w_T$ where $w_t \in V$

Learn: $f(w_1 \dots w_t) = \hat{P}(w_t \mid w_1 \dots w_{t-1})$

s.t. giving a high probability to an unseen test/dev set (e.g. minimize the perplexity)

constraint: create a proper probability distribution (e.g. sums to 1) so that we can take the product of conditional probs to get the joint prob of a sentence.

Model

① Create a mapping function C from any word in V onto $\mathbb{R}^M$. $M = 30, 60, 100$

Model

① Create a mapping function C from any word in V onto $\mathbb{R}^M$. $M = 30, 60, 100$

Store this in a V-by-M matrix.
Initalize it w/ SVD

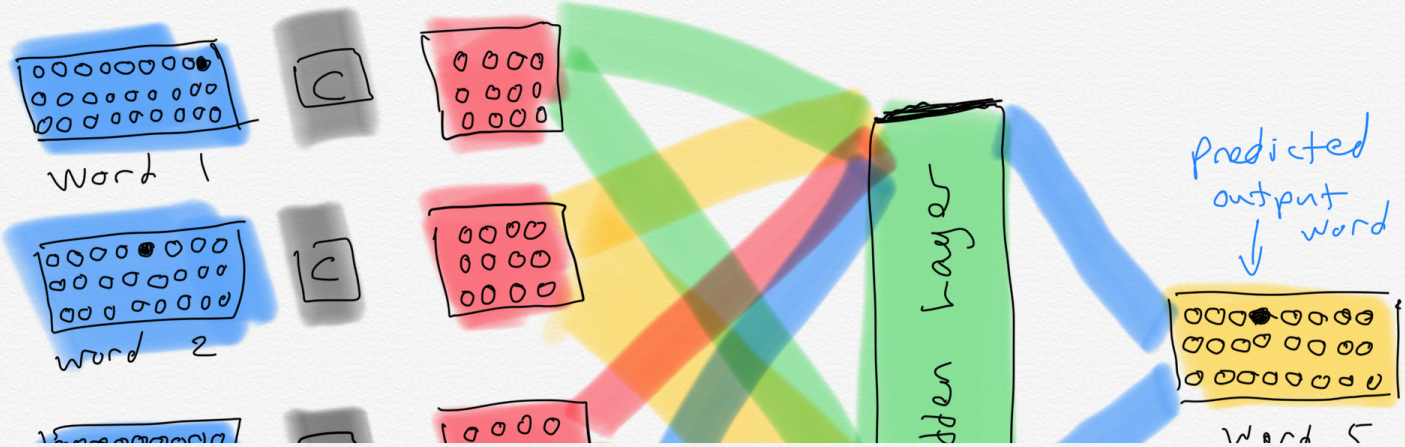
② The neural architecture:
a fn g maps seq of word vectors onto a prob. distribution over words vocab V.

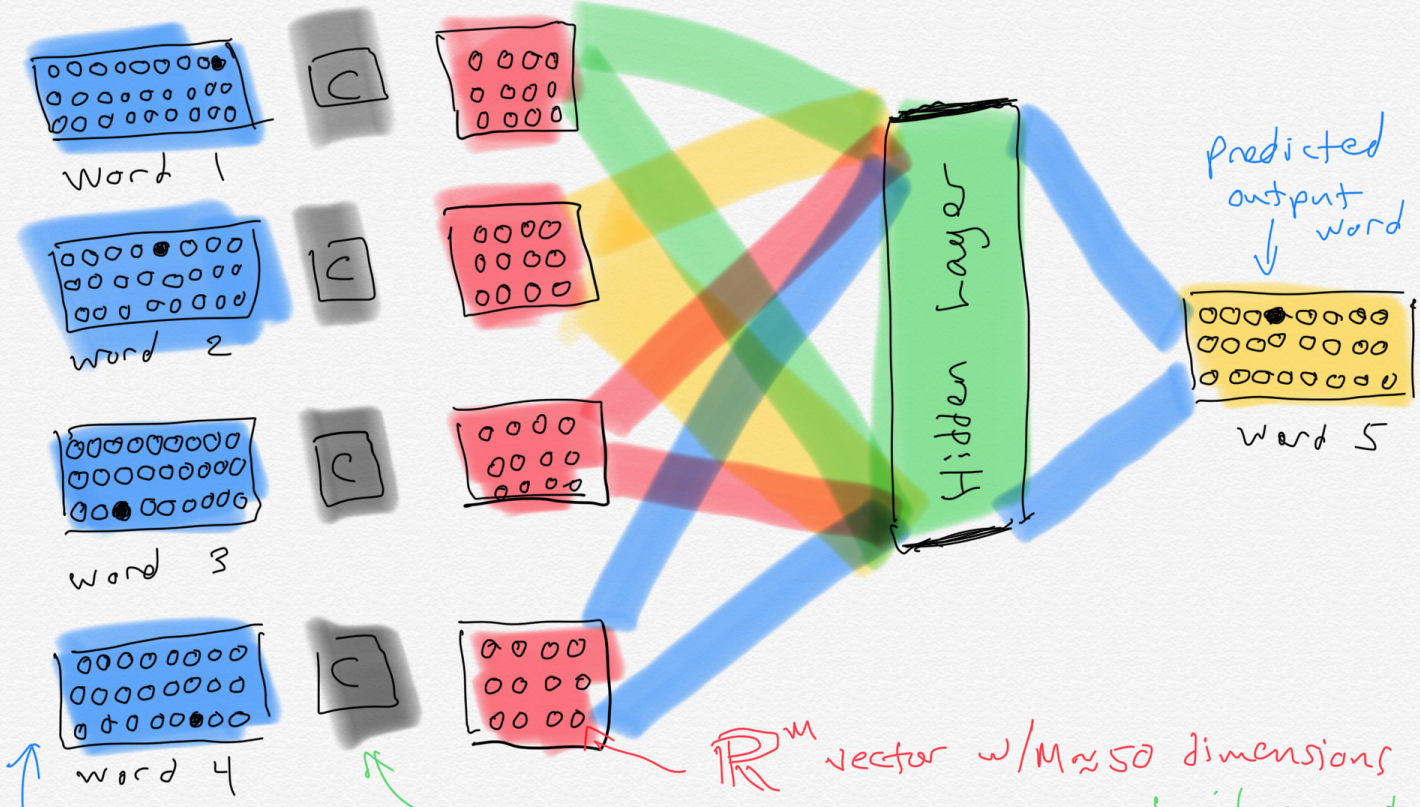
$$g(C(w_{t-N}) \cdots C(w_{t-1})) = \hat{P}(w_t \mid w_{t-N} \cdots w_{t-1})$$

g is implemented w/ a feed forward or recurrent neural network.
trained w/ stochastic gradient descent
They use "softmax" in output layer

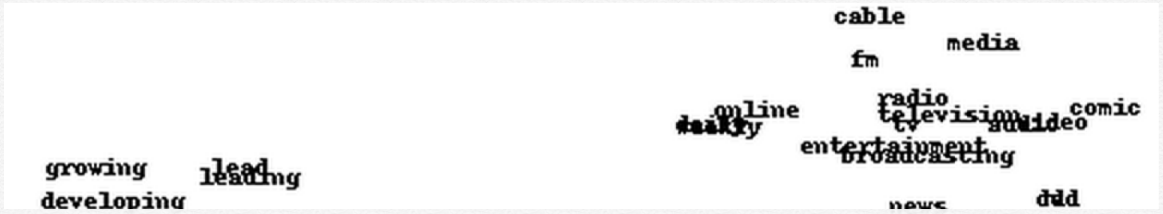
Claim: NN LM is better than the n-gram LM
Verify. = Evaluate

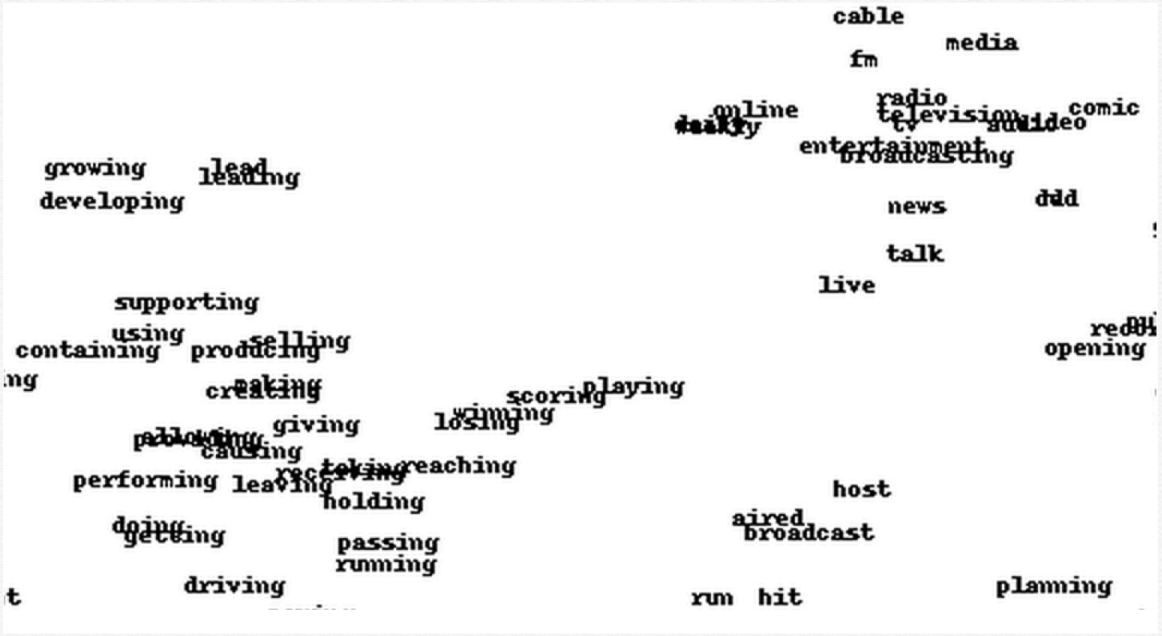
- Train on same training data
- Intrinsic evaluation or extrinsic eval
  Perplexity on previously unseen data (test set)





A by-product of training a neural network based language models are dense word embeddings. This is the way that word embeddings that you downloaded for the advanced vector space models HW were created. E.g. word2vec, GLOVE.





When the ~50 dimensional vectors that result from training a Neural LM are projected down to 2-dimensions, we see a lot of words that are intuitively similar to each other are close together.

For more information about Neural Networks, check out chapter 8 of the textbook.

It covers

- Neural network training
- activation functions
- Feed forward NNs
- Training w/loss fns and stocastic gradient descent.