LANGUAGE MODELLING NOTES
SENTHIL KUMAR

Main sources:
- Wikipedia,
- Udemy Course on Deep NLP by Lazy Programmer,
- NLP Course on Coursera by National Research University Higher School of Economics

Agenda

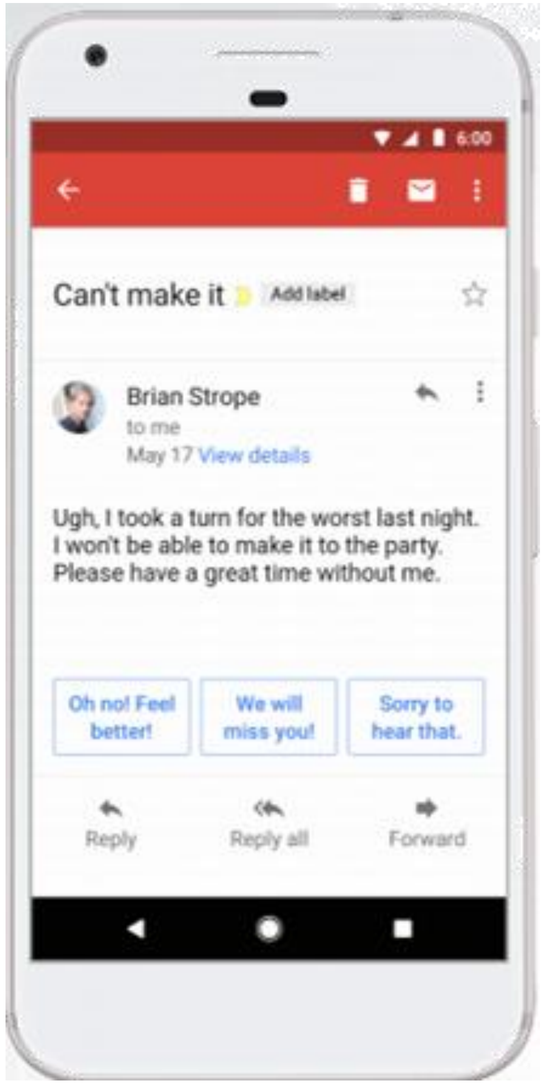
What is a Language Model? Where is it used?

Language Model: A model of the **probability of a sequence of words**

A language model **can assign probability to each possible next word**. And also, help in **assigning a probability to an entire sentence**.

Applications:
Predicting upcoming words or estimating probability of a phrase or sentence is useful in noisy, ambiguous text data sources

i) **Speech Recognition** - E.g.: P("recognize speech") >> P("wreck a nice beach")
ii) **Spelling Correction** – E.g.: P("I have a gub") << P("I have a gun")
iii) **Machine Translation** – E.g.: P("strong winds") > P("large winds")
iv) **Optical Character Recognition/ Handwriting Recognition**
v) **Autoreply Suggestions**
vi) **Text Classification**


Can't make it

Brian Strope
to me
May 17 View details

Ugh, I took a turn for the worst last night. I won't be able to make it to the party. Please have a great time without me.

What is a Bigram Model?

Sample Corpus:

This is **the house** that Jack built.

This is **the** malt

That lay in **the house** that Jack built.

This is the rat,

That ate **the** malt

That lay in **the house** that Jack built.

This is the cat,

That killed **the** rat,

That ate **the** malt

That lay in **the house** that Jack built.

P (house | the) = count (the house) / count (the)

Aside from the intuitive way we would calculate, this is an example of **Conditional Probability**!

P(B | A) = P(A , B) / P(A)

Technically, what we have computed above is **a Bigram Language Model**

$$Bigram\ model: \ p(w_t \mid w_{t-1})$$

How do ngrams help us calculate the prob of a sentence?

- Sentence, S = “A B”

  We know that, probability of this sentence “A B” as,

  P (S) = Prob (A ***before*** B) = P(A , B) = Joint Probability of A and B = P(B | A)* P(A)

- Let us assume a three word sentence, S = “A B C”

  P(S) = Prob (A ***before*** B ***before*** C) = P (A , B , C)= P(C | A , B) * P (A n B)

  = P (C| A , B) * P(B | A) * P (A)

- Even if there are more words, we could keep applying Conditional Probability and compute the prob of a sentence. The above rule is called “**Chain Rule of Probability**” given by (from wiki):

$$\mathrm{P}(A_n,\ldots,A_1) = \mathrm{P}(A_n|A_{n-1},\ldots,A_1)\cdot\mathrm{P}(A_{n-1},\ldots,A_1)$$

Repeating this process with each final term creates the product:

$$\mathrm{P}\left(\bigcap_{k=1}^{n} A_k\right) = \prod_{k=1}^{n}\mathrm{P}\left(A_k \,\middle|\, \bigcap_{j=1}^{k-1} A_j\right)$$

With four variables, the chain rule produces this product of conditional probabilities:

$$\mathrm{P}(A_4,A_3,A_2,A_1) = \mathrm{P}(A_4 \mid A_3,A_2,A_1)\cdot\mathrm{P}(A_3 \mid A_2,A_1)\cdot\mathrm{P}(A_2 \mid A_1)\cdot\mathrm{P}(A_1)$$

Wait, we have not calculated the P(S) for S="A B C" yet!

S = "A B C"

P(S) = P (C | A , B) * P(B | A) * P (A) → from previous slide

This is a <u>bigram</u>,

P(B | A) = Count (A , B) / Count (A), [since, P (house | the) = count (the house) / count (the)]

This is a <u>trigram</u> and this is a <u>unigram</u>. How will we find their probabilities. Again, by counting!

Unigram: P(A) = Count (A) / Length(Corpus)

Bigram: P(B | A) = P(A , B) / P(A)

Trigram: P(C | A , B) = Count (A , B, C) / Count (A , B)

What about a 5 word sentence like S = "A B C D E":

$$p(A,B,C,D,E) = p(E \mid A,B,C,D)p(D \mid A,B,C)p(C \mid A,B)p(B \mid A)p(A)$$

The Out-of-Vocabulary Problem

Original Corpus:

This is the house that Jack built.

This is the malt

That lay in the house that Jack built.

This is the rat,

That ate the malt

That lay in the house that Jack built.

This is the cat,

That killed the rat,

That ate the malt

That lay in the house that Jack built.

New Data 1 –

"This is the dog,

That scared the cat,

That killed the rat,

That ate the malt,

That lay in the house that Jack built."

New Data 2 -

"I am great"

What is the probability of the above new texts being part of the corpus?

For New Data 1, **by our chain rule of probability** where we keep multiplying the probabilities, we would encounter **P(dog | the)** = 0, hence the overall probability would be zero too

For New Data 2, none of the words exist, hence its probability of occurrence is also zero

One way to avoid, Smoothing!

MLE *: $$P(B|A) = \frac{\text{Count}(A, B)}{\text{Count}(B)}$$

Why is it called MLE?

Suppose a bigram “the dog” occurs 5 times in 100 documents. Given that we have this 100 document corpus (which the language model represents), the maximum likelihood of the bigram parameter “the dog” appearing in the text is 0.05

Add 1 (Laplace) Smoothing: $$P_{\text{smooth}}(B \mid A) = \frac{\text{Count}(A\ B) + 1}{\text{Count}(B) + |V|}$$

- We pretend that each unique bigram occurs once more than it actually did!
- Since we have added 1 to each bigram, we have added |V| bigrams in total. Hence normalizing by adding |V| to the denominator

Add- ∂ Smoothing: $$P_{\text{smooth}}(B \mid A) = \frac{\text{Count}(A\ B) + \partial * 1}{\text{Count}(B) + \partial * |V|}$$

- *∂ is any fraction such as 0.1, 0.05, etc.,*
- *Unlike Add one smoothing, which reduces drastically the prob of high occurring words, this solves the zero prob issue*

The Markov Assumption

For a 5-word sentence, S = “A B C D E”, the probability of the sentence occurring is:

$$p(A,B,C,D,E) = p(E \mid A,B,C,D)p(D \mid A,B,C)p(C \mid A,B)p(B \mid A)p(A)$$

What is Markov Assumption?

“What I see NOW depends only on what I saw in the PREVIOUS step”

$$p(w_t \mid w_{t-1}, w_{t-2}, \dots, w_1) = p(w_t \mid w_{t-1})$$

Hence the probability of occurrence of the 5-word sentence is:

$$p(A,B,C,D,E) = p(E \mid D)p(D \mid C)p(C \mid B)p(B \mid A)p(A)$$

What are its advantages?

- Probability of an entire sentence could be very low but individual phrases could be more probable.

  For e.g.:

  Actual Data: “*The quick fox jumps over the lazy* ***dog***”

  Probability of a new sentence: Prob(“*The quick fox jumps over the lazy* ***cat***”) = 0 (though probable)

  In Markov assumption (with additive smoothing), the above sentence will have a realistic probability

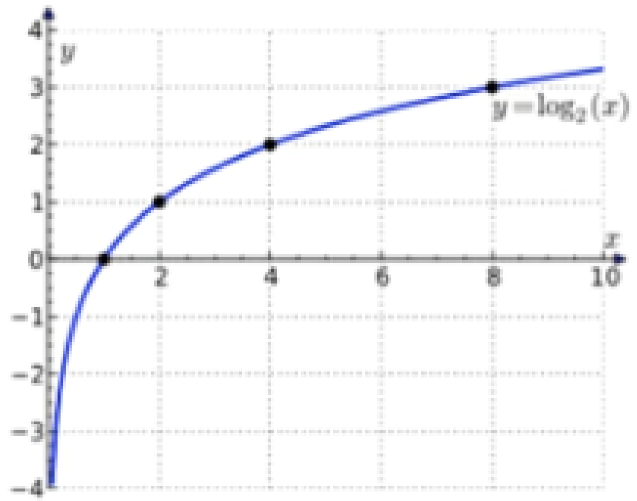
Evaluation – Log-probability

So, the probability of occurrence of the 5-word sentence, by Markov Assumption, is:

$$p(A,B,C,D,E) = p(E \mid D)p(D \mid C)p(C \mid B)p(B \mid A)p(A)$$

But multiply probabilities like these could end up giving you the result closer to zero.
For e.g.: P(E|D) = 0.1, P(D|C) = 0.07, P(C|B) = 0.1, P(B|A) = 0.05, P(A) = 0.2
P(A before B before C before D before E) = 0.000007 → too low or almost zero !

Logarithm to the rescue !



Logarithm is a monotonically increasing function !
Meaning: If P(E|D) > P(D|C), then log(P(E|D)) > log (P(D|C))
Also, log (A *B) = log (A) + log (B)

$$logp(w_1, \ \ldots, \ w_T) = logp(w_1) + \sum_{t=2}^{T} logp(w_t \mid w_{t-1})$$

where T is the number of words in the sentence.
Since log (probabilities) are always negative (see graph), shorter sentences will have higher probability of occurrence. To normalize it,

$$\frac{1}{T} logp(w_1, \ \ldots, \ w_T) = \frac{1}{T} \left[logp(w_1) + \sum_{t=2}^{T} logp(w_t \mid w_{t-1}) \right]$$

Evaluation – Perplexity (used in literature, a variation of the log-probability)

- **Perplexity** is a measurement of how well a probability model predicts a sample.
- It is used to compare probability models.
- A low perplexity indicates the probability distribution is good at predicting the sample

Definition:

Perplexity is the inverse probability of the **test** set, normalized by the number of words.

Perplexity of test data = PP(test data) $= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$ → ***the lower the perplexity, the better it is***

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

PP (test data) $= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$

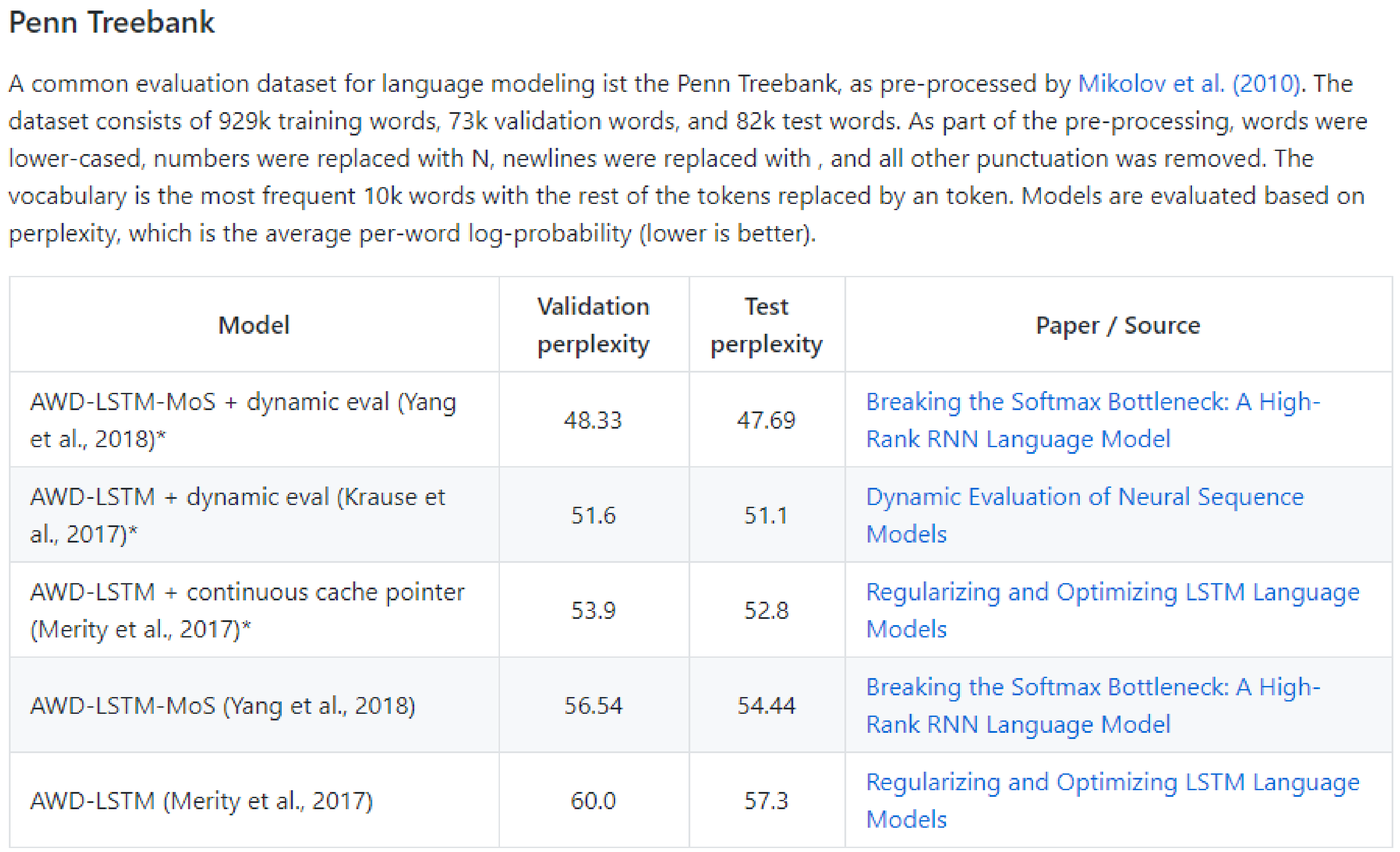
PP(test data for a bigram model) $= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$

Comparing with normalized log probability for a bigram model:

$$\frac{1}{T}\left[logp(w_1) + \sum_{t=2}^{T} logp(w_t \mid w_{t-1}) \right]$$

The higher the log-probability value, the better it is

Best-in-class Neural Network Language Models



Penn Treebank

A common evaluation dataset for language modeling ist the Penn Treebank, as pre-processed by Mikolov et al. (2010). The dataset consists of 929k training words, 73k validation words, and 82k test words. As part of the pre-processing, words were lower-cased, numbers were replaced with N, newlines were replaced with , and all other punctuation was removed. The vocabulary is the most frequent 10k words with the rest of the tokens replaced by an token. Models are evaluated based on perplexity, which is the average per-word log-probability (lower is better).

Model	Validation perplexity	Test perplexity	Paper / Source
AWD-LSTM-MoS + dynamic eval (Yang et al., 2018)*	48.33	47.69	Breaking the Softmax Bottleneck: A High-Rank RNN Language Model
AWD-LSTM + dynamic eval (Krause et al., 2017)*	51.6	51.1	Dynamic Evaluation of Neural Sequence Models
AWD-LSTM + continuous cache pointer (Merity et al., 2017)*	53.9	52.8	Regularizing and Optimizing LSTM Language Models
AWD-LSTM-MoS (Yang et al., 2018)	56.54	54.44	Breaking the Softmax Bottleneck: A High-Rank RNN Language Model
AWD-LSTM (Merity et al., 2017)	60.0	57.3	Regularizing and Optimizing LSTM Language Models

In Complexity (from left to right)

N-gram LM → Softmax based LM → Simple NN Model → … → AWD LSTM Models

APPENDIX

Logistic Regression modeled bigram probabilities

If x is the current word vector and y is the next word vector,
Prob of y being the next word given the current word is x is given by

$$p(y \mid x) = softmax(W^T x)$$

How do we find W?

- By doing **gradient descent on the cost function**
- What is a cost function then?

Cost function/ Log loss of a binary logistic regression is a cross-entropy,

$$\sum_{(x,y)\in D} -y\log(y') - (1-y)\log(1-y')$$

y is the label in a labeled example. Since this is logistic regression, every value of y must either be 0 or 1.

y' is the predicted value (somewhere between 0 and 1), given the set of features in x.

Cost function in our case is: $J = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{V} y_{n,k} log(p(y_{n,k}|x_n))$

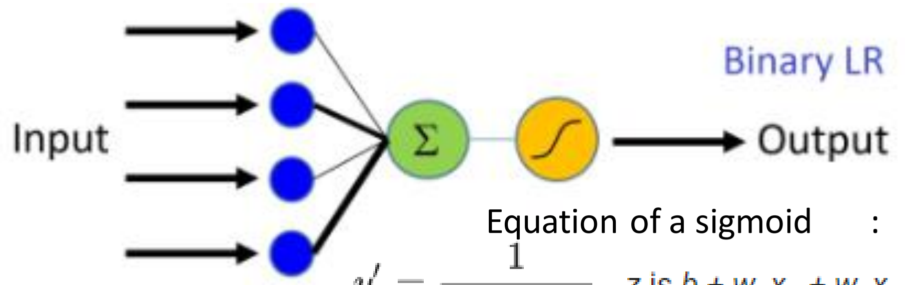
Initialize W to be a random weights matrix of size V x V, where V is the vocabulary size

Gradient Descent on J = Partial derivative of J with respect to W: $\nabla J = X^T(p(Y|X) - Y)$

Optimize W: $W \leftarrow W - \eta\nabla J,\ where\ \nabla J = X^T(p(Y|X) - Y)$

Where η is the learning rate

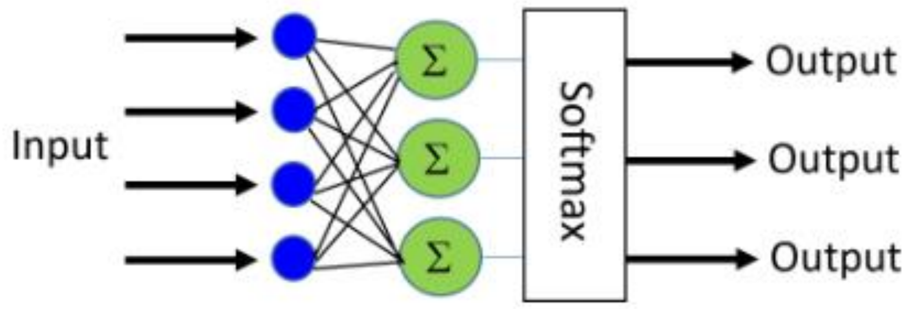
Binary LR



Equation of a sigmoid :

$$y' = \frac{1}{1+e^{-(z)}}$$ z is $b + w_1x_1 + w_2x_2 + \ldots w_Nx_N$

Multiclass LR



Equation of a softmax:

$$p(y=j|\mathbf{x}) = \frac{e^{(\mathbf{w}_j^T\mathbf{x}+b_j)}}{\sum_{k\in K} e^{(\mathbf{w}_k^T\mathbf{x}+b_k)}}$$

Where j is one of the K classes

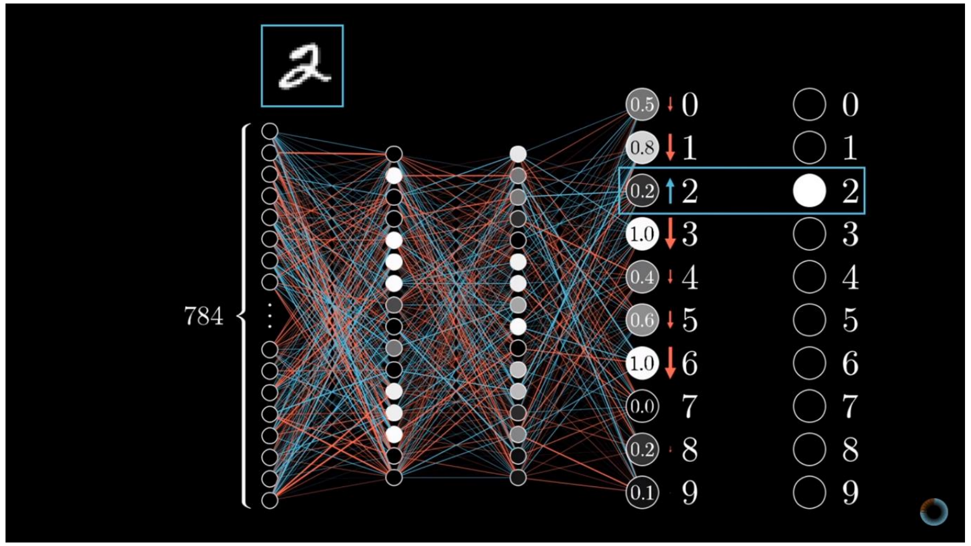
```
inputs[np.arange(n - 1), sentences[:n-1]] = 1
targets[np.arange(n - 1), sentences[1:]] = 1
# get output predictions
predictions = softmax(inputs.dot(W[every]))
# do a gradient descent step
W[every] = W[every] - lr * inputs.T.dot(predictions - targets)
```

Neural Network Language Model

Logistic Regression based model

$$p(y \mid x) = softmax(W^T x)$$

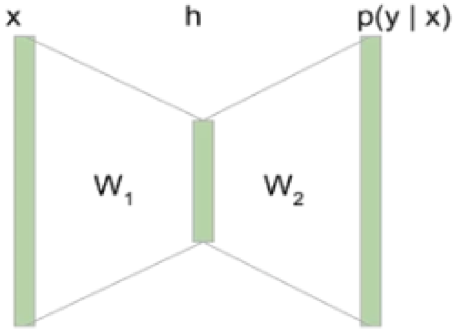
In a NN, Using a softmax activation function at the output layer is basically equivalent to a Logistic Regression over the features extracted from the layer before the final Fully Connected layer



Neural Network based Model

$$h = tanh(W_1^T x)$$

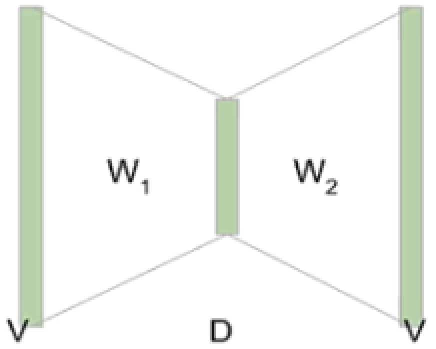
$$p(y \mid x) = softmax(W_2^T h)$$



Cost function is the same for the NN based model:

$$J = \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{V} y_{n,k} log(p(y_{n,k} | x_n))$$

shape(W_1) == V x D
shape(W_2) == D x V



But the gradient descent on the cost function has two weight matrices W1 and W2

$$W_2 \leftarrow W_2 - \eta \nabla_{W_2} J \quad \nabla_{W_2} J = H^T (p(Y \mid X) - Y)$$

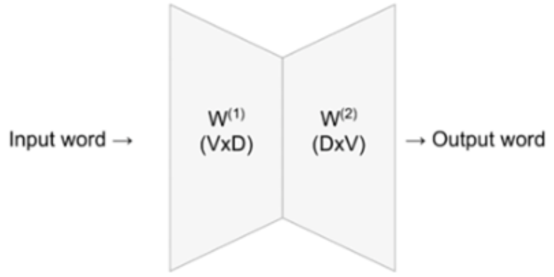
$$W_1 \leftarrow W_1 - \eta \nabla_{W_1} J \quad \nabla_{W_1} J = X^T \left[(p(Y \mid X) - Y) W_2^T \odot (1 - H^2) \right]$$

Is Bigram Language models a precursor to Word2Vec?

Bag of words vectors (TFIDF) → Language model → Word2Vec type embeddings

Typical Bigram model using softmax and any activation function f

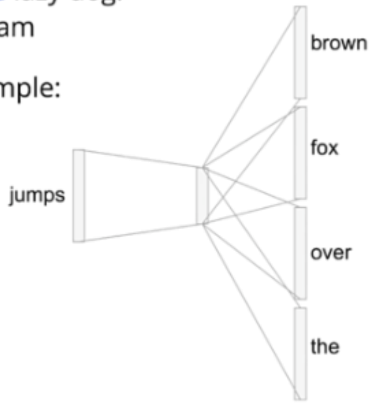
$$p(x_{t+1} \mid x_t) = softmax(W^{(2)T} f(W^{(1)T} x_t))$$



Given a window size of 2 words around a focus word, the **skip-gram model predicts the neighboring/context words given the current/focus word** (here – jumps).

Skipgram

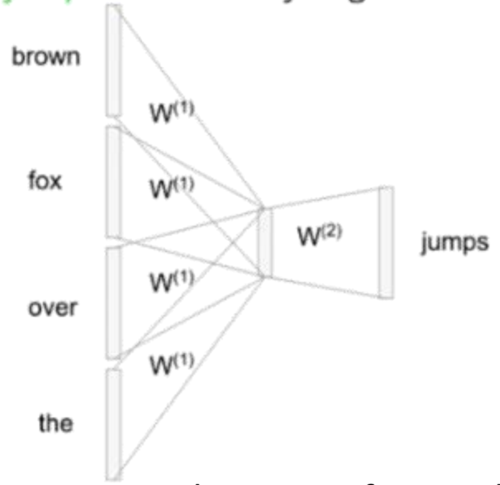
- "The quick brown fox jumps over the lazy dog."
- Helpful to think of it in terms of bigram
- Bigram model gives us 1 training sample:
  jumps → over
- Skipgram gives us 3 additional training samples:
  jumps → brown
  jumps → fox
  jumps → the



The 2 Word2Vec Models

CBOW - continuous bag of words

"The quick brown fox jumps over the lazy dog."



Given a window size of 2 words around a focus word - jumps, **the CBOW model predicts the current word *'jumps'*, given the neighboring words in the window**

How Language Models differ from Naïve Bayes Classifier?

A quick intro to NB Classifier:

$$\text{Posterior probability} = \frac{\text{Prior probability x Likelihood}}{\text{Evidence}}$$

$$\Pr(\, y \mid X) = \frac{\Pr(y) \times \Pr(X \mid y)}{\Pr(X)}$$

$$\Pr(y=CS|\text{"Python"}) = \frac{\Pr(y=CS) \times \Pr(\text{"Python"} \mid y=CS)}{\Pr(\text{"Python"})}$$

$$\Pr(y=Zoology|\text{"Python"}) = \frac{\Pr(y=Zoology) \times \Pr(\text{"Python"} \mid y=Zoology)}{\Pr(\text{"Python"})}$$

The above is Bayes rule

Naïve – because the word features are assumed to be independent of each other given the class label

For X = “python download”,

$$\Pr(y = CS \mid \text{"python download"}) = \frac{\Pr(y = CS) * \Pr(\text{"python"} \mid y = CS) * \Pr(\text{"download"} \mid y = CS)}{\Pr(\text{"python download"})}$$ *__generally ignored*

Naïve Bayes is also a probabilistic model but assumes no relationship between text features.
In our case, it **does not take word order into consideration**

N-gram LM assumes relationship between words and predicts the probability of next word given the current word

THANK YOU