1 Neural Networks

1.1 Why Neural Networks?
- Most current machine learning works well because of human designed representations and input features.
- But handcrafting features is time-consuming.
- The features are often both over-specified and incomplete.
- The work has to be done again for each task/domain.
- Deep learning provides a way of doing this by automatically learning good features and representations.

1.2 Demystifying Neural Networks
- In neural networks, a neuron gets weighted sum of inputs.
- It applies a non-linear activation function on that weighted sum of inputs and produces output.

1.2.1 From Maxent to Neural Nets
A Maxent classifier can be written in vector form as:

\[ P(c, d|\lambda) = \frac{\exp(\lambda^T f(d, c))}{\sum_c \exp(\lambda^T f(d, c))} \]

where \( f(d, c) \) is feature vector given data and a class \( c \).
Assuming we have just two classes $c_1$ and $c_2$. Then maxent classifier for class $c_1$ can be written as:

$$P(c_1, d|\lambda) = \frac{\exp(\lambda^T f(d, c_1))}{\exp(\lambda^T f(d, c_1)) + \exp(\lambda^T f(d, c_2))}$$

The above expression can also be written as follows:

$$P(c_1, d|\lambda) = \frac{1}{1 + \exp(-\lambda^T x)}$$

which is sigmoid function

$$P(c_1, d|\lambda) = \frac{1}{1 + \exp(-\lambda^T (f(d,c_2) - f(d,c_1)))}$$

This is exactly what an artificial 'neuron' computes, as can be seen in following figure.

A neural network is running several logistic regressions at the same time, as seen in following figure.

1.2.2 Neural Networks Summary

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs. But we dont have to decide ahead of time what variables these logistic regressions are trying to predict.
We can also add additional layers by feeding the sum of weighted output of one layer into another logistic regression function. Training criteria is setup to adjust the weights in such that we predict the final output accurately. We need non-linearity in neural networks because without non-linearities, deep neural networks can do nothing more than a linear transform. Extra layers could just be compiled down into a single linear transform. Apart from sigmoid function other non-linearities, such as tanh, ReLU can also be used.

2 Part of Speech Tagging

Def: Tagging is a task of labelling each word in a sentence with its appropriate part of speech. eg: noun, adjective, verb etc.

Parts of speech can be divided into two broad categories:

- Closed class types: Closed classes are those that have relatively closed class fixed membership. eg: prepositions are a closed class because there is a fixed set of them in English.
- Open class types: By contrast nouns and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages. eg: verbs and nouns

Common Parts of speech in English language are:

There are a small number of popular tagsets for English, many of which evolved from the 87-tag tagset used for the Brown corpus (Francis, 1979; Francis and Kucera, 1982).

Three of the most commonly used tagsets are:

1. The small 45-tag Penn Treebank tagset (Marcus et al., 1993). Some of the tags from this tag-set are shown in figure:1.

2. The medium-sized 61 tag C5 tagset used by the Lancaster UCREL projects CLAWS (the Constituent Likelihood Automatic Word-tagging System) tagger to tag the British National Corpus (BNC) (Garside et al., 1997)

3. The larger 146-tag C7 tagset (Leech et al., 1994)
The part of speech for a word gives a significant amount of information about the word and its neighbors.

Knowing part of speech for a word can tell us what words are likely to occur in its vicinity. This can be useful in a language model for speech recognition.

Some words are pronounced differently based on which part of speech they belong to. Thus knowing the part of speech can produce more natural pronunciations in a speech synthesis system.

It can also be helpful in case of rare words, even though we haven’t seen the word before we can understand the behavior of the word.

Parts of speech can also be used in stemming for informational retrieval (IR), since knowing a words part of speech can help tell us which morphological affixes it can take.

They can also help an IR application by helping select out nouns or other important words from a document.

Useful as features to downstream systems. eg: Given a Premise: **Bob is trying to sell his house.**

Our task is to predict which of the given hypothesis is correct.

**hypothesis-1:** Bob sold his house.

**hypothesis-2:** Bob wants to sell his house.

Neural networks might have hard time at this task and might predict hypothesis-1 to be correct but POS-tagging will predict hypothesis-2 to be correct.
2.2 Part-of-speech tagging algorithms

2.2.1 Rule-based part-of-speech tagging

This approach is a baseline approach for assigning part-of-speech based on a two-stage architecture. The first stage uses a dictionary to assign each word a list of potential parts of speech. The second stage used large lists of hand-written disambiguation rules to window down this list to a single part-of-speech for each word.

This approach has a serious drawback that it cannot handle unknown words.

2.2.2 Classifier-based part-of-speech tagging

Another approach can be to build a classifier that maps words to part of speech tags based on 1-grams. This approach also has drawbacks because words can be mapped to various parts of speech and the model won’t work too well. eg: fire can be both a noun and a verb. We would also need surrounding words (context) as features to correctly map a word to its corresponding part of speech.

2.2.3 Sequence-based part-of-speech tagging

This section describes a particular stochastic tagging algorithm generally known as the Hidden Markov Model or HMM tagger. The intuition behind all stochastic taggers is a simple generalization of pick the most-likely tag for this word approach based on the Bayesian framework.

For a given sentence or word sequence, HMM taggers choose the most probable sequence of tags.

\[ T^* = \arg \max_T P(T|W) \]

\[ P(T|W) = P(W|T)P(T) \]

\[ P(T|W) = \prod_i P(w_i|w_{i-1} t_{i-1} w_{i-2} t_{i-2} \ldots w_1 t_1) P(t_i|t_{i-1} t_{i-2} \ldots t_1) \]

Now we make few assumptions for modeling the probability of word sequences. We will use trigram model, so we will assume that the tag history can be approximated by the most recent two tags. We will make another simplifying assumption that the probability of a word is dependent only its tag. We can keep increasing the order of the model but there is a trade-off, higher order model means higher complexity.

\[ P(T|W) = \prod_i P(w_i|t_i) P(t_i|t_{i-1} t_{i-2}) \]

The first part of the above equation is called emission matrix, and second part is called transition matrix.

emission matrix = \( P(w_i|t_i) \)

transition matrix = \( P(t_i|t_{i-1} t_{i-2}) \)

Calculating Emission and transition matrix from data

We can use maximum likelihood estimation from relative frequencies to estimate emission and transition matrix from the data.

\[ \text{transition matrix} = P(t_i|t_{i-1} t_{i-2}) = \frac{\text{count}(t_i|t_{i-1} t_{i-2})}{\text{count}(t_{i-1} t_{i-2})} \]
emission matrix = \( P(w_i|t_i) = \frac{\text{count}(w_i, t_i)}{\text{count}(t_i)} \)

### Estimating Transitions

We can use linear interpolation to smooth \( P(S_i|S_{i-1}, S_{i-2}) \)

\[
P(s_i|s_{i-1}, s_{i-2}) = \lambda_2 P(s_i|s_{i-1}, s_{i-2}) + \lambda_1 P(s_i|s_{i-1}) + (1 - \lambda_1 - \lambda_2) P(s_i)
\]

For smoothing of Transition matrix it doesn’t help much to use fancier smoothing techniques (e.g. KN smoothing) or use higher orders, because Tag distribution is quite different from word distribution.

### Estimating Emissions

For smoothing in case of unknown words we can’t back off, eg: We can’t write \( P(w_i|S_i) \approx P(w_i) \). Because in this case the bottle neck is the rare word and not the tag. **Thus to predict emissions we use smoothing at character level.**

Given an unknown word which is combination of 7 characters \( c_1c_2c_3c_4c_5c_6c_7 \).

We first compute \( P_{sm}(s|c_1c_2...c_7) \) using interpolation.

\[
P_{sm}(s|c_1c_2...c_7) = \hat{P}(s|c_1c_2...c_7) + \kappa P_{sm}(s|c_2c_3...c_7)
\]

Then using bayes rule we compute \( P(c_1c_2...c_7|s) \)

\[
P(c_1c_2...c_7|s) = \frac{P(s|c_1c_2...c_7) P(c_1c_2...c_7)}{P(s)}
\]

While interpolating we start removing characters from beginning of the word because suffixes give more information about the word as compared to prefixes eg: **adverbs**: hastily, quickly, slowly, angrily **verbs**: running, flowing, living, jumping

### 2.3 Decoding with HMMs

Problem Statement: Find the most likely sequence of tags that maximizes the following model

\[
s^* = \arg\max_s \prod_i P(s_i|s_{i-1}) P(w_i|s_i)
\]

#### 2.3.1 Brute Force Algorithm

The naive, brute force method would be to simply enumerate all possible tag sequences \( s_1...s_n \), score them under the function \( P(s_1...s_n|w_1...w_n) \), and take the highest scoring sequence. But this algorithm cannot be used in practice because of its of exponential time complexity. Assuming that the set of possible tags is \( K = \{D, N, V\} \), then for an input sentence of length \( n \), there are \( |K|^n \) possible tag sequences.
2.3.2 Beam Search Algorithm

In practice it is often too expensive to consider all possible sequences of states, thus low-probability paths are pruned at each step and are not considered while calculating next state in the sequence. Beam search is a greedy heuristic which often works well in practice. The algorithm maintains a short list of high-probability sequences of states whose path probabilities are within some percentage (beam width) of the most probable path. Instead of maintaining sequences that are within some factor of the most probable path, algorithm can also maintain top K sequences and discard the rest. eg: when the value of K is chosen to be 1, the algorithm will maintain just one sequence of states whose path probability is the highest. Because the algorithm is maintaining just one sequence of states, at each step while calculating state of next word, the state will be picked such that it maximizes the path probability of this particular sequence of states.

2.3.3 Viterbi algorithm

We can efficiently find the highest probability tag sequence, using a dynamic programming algorithm that is often called the Viterbi algorithm. We will discuss the following two parts in depth:

Recursion

The problem statement is: Given a sequence of observed states(words) find the most probable sequence of hidden states(tags). For a given sequence of n words, while calculating state at $i$-th position, we will list all probable states for that position. Then for each of those probable states the we will evaluate following expression:

$$\psi_{i-1} = \arg\max_{s_i \cdots s_{i-2}} P(s_{i-1} | s_{i-2}) P(w_{i-1} | s_{i-1}) P(s_{i-2} | s_{i-3}) P(w_{i-2} | s_{i-2}) \cdots P(s_1) P(w_1 | s_1)$$

We will also evaluate a similar expression for $i$-th position

$$\psi_i = \arg\max_{s_{i-1} \cdots s_{i-2} s_{i-1}} P(s_i | s_{i-1}) P(w_i | s_i) P(s_{i-1} | s_{i-2}) P(w_{i-1} | s_{i-2}) \cdots P(s_1) P(w_1 | s_1)$$

As can be clearly seen from above two equations:

$$\psi_i = \arg\max_{s_{i-1}} P(s_i | s_{i-1}) P(w_i | s_i) \psi_{i-1}$$

We can take advantage of this recursive equation and use dynamic programming to calculate the sequence of most probable states as explained in next section.

Dynamic Programming

For $i$-th position we will evaluate following expression:

$$\delta(s_i) = \max_{s_{i-1} \cdots s_{i-1}} P(s_i | s_{i-1}) P(w_i | s_i) P(s_{i-1} | s_{i-2}) P(w_{i-1} | s_{i-2}) \cdots P(s_1) P(w_1 | s_1)$$

Using the recursive equation derived in the above section we can write:

$$\delta(s_i) = \max_{s_{i-1}} P(s_i | s_{i-1}) P(w_i | s_i) \ast \delta(s_{i-1})$$
Where our Basecase will be as follows:

\[ \delta(s_1) = P(w_1|s_1) \ P(s_1) \]

To calculate final score we need to calculate:

\[ \text{final score} = \max_{s_n} \delta(s_n) \]

We can also store the backtrace which maximizes the score. That will give us the sequence of most probable tags. At any position \( i \) for a sequence of states ending at \( s_i \) we can find the \( i-1 \)th most probable tag with the help of following equation given that we have already computed first \( i-2 \) most probable states in the sequence:

\[ s = \arg\max_{s_{i-1}} P(s_i|s_{i-1}) \ P(w_i|s_i) \ \delta(s_{i-1}) \]

### 2.4 Accuracies with above models

<table>
<thead>
<tr>
<th>Known Words</th>
<th>Unknown Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary Lookup - Most frequent tag</td>
<td>0.90</td>
</tr>
<tr>
<td>Features only based on word in question</td>
<td>0.937</td>
</tr>
<tr>
<td>Trigram HMM</td>
<td>0.95</td>
</tr>
<tr>
<td>TNT (Trigram HMM++)</td>
<td>0.97</td>
</tr>
</tbody>
</table>

### 2.5 Maximum-entropy Markov model

A maximum-entropy Markov model (MEMM), or conditional Markov model (CMM), is a graphical model for sequence labeling that combines features of hidden Markov models (HMMs) and maximum entropy (MaxEnt) models. An MEMM is a discriminative model that extends a standard maximum entropy classifier by assuming that the unknown values to be learnt are connected in a Markov chain rather than being conditionally independent of each other.

\[ P(S|W) = \prod_{i}^{ME} P(s_i|s_{i-2}..s_1, W) \]

As seen from the above equation, while computing hidden states, the classifier makes single decision at a time which is conditioned on both observations and the previous decisions. Thus while computing hidden state of \( i \)th observed state we are allowed to use features of all the observed states and also the features from \( i-1 \) hidden states(which the classifier has already computed). But in practice we use features from only one or two previous hidden states. Thus we pick a sequence of states that maximizes following equation.

\[ \psi = \arg\max_{s_1..s_n} \prod_i P(s_i|s_{i-1}, w_i) \]

As we can see above equation is similar to the one we saw in HMM, thus we can use Beam Search or Viterbi algorithm to find the sequence of states that maximizes the above probability.

**Recursive equation to find sequence of states for MEMM that maximize the score**

\[ \psi(s_i) = \arg\max_{s_{i-1}} P(s_i|s_{i-1}, w_i) \ \delta(s_{i-1}) \]

where \( \delta(s_i) = \max_{s_{i-1}} P(s_i|s_{i-1}, w_i) \ \delta(s_{i-1}) \)

As compared to HMM, Maximum entropy Markov model is difficult to train since we would need to run gradient decent over entire data. But on the other hand we can include much expressive features(possibly from all the observed states and all previous, computed hidden states)