DEEP LEARNING

AAKASH GHOSH

Contents

Part I

Theory of Deep Learning

Chapter 1

Deep Learning

1.1 Traditional Machine Learning

Traditionally, solving a problem by machine

Part II

Natural Language Processing for Deep Learning

Chapter 2

Why is language a pain to model? And how do we model them.

2.1 Language and ideas

Language arose about $100,000 - 1,000,000$ years ago and that took us from Bronze Age to modern science. From a linguistic point of view, a word is the representation of an idea. Words representing old obsolete ideas are often forgotten, and new words representing new ideas.

2.2 Why is NLP hard?

Written as a person fluent in English, Bengali and Hindi
The short answer is when we speak, we leave out massive chunks of information. A lot of the information conveyed while writing assumes that the reader has prior context and experience to extract meaningful information from the text. It is even more difficult when looking at transcriptions of spoken text: While speaking we use our facial expressions and vocal tones as a part of communication. Some particular difficulties are described below:

1. Homonyms: When we say ”He/She is cool” we often mean that a person is calm and collected. We use our prior experience to infer ”cool” is not referring to feeling coldness. Ideally, we would like our model to understand such subtleties. More examples include:

   - **Iraqi Head Seeks Arms**: Here head is to be interpreted as leader and not as body part.
   - **Kids Make Nutritious Snacks**: Here make is to be understood as the process of creation and is not meant as a part of.
   - **Miners Refuse to Work after Death**: Here death refers to death of a person and not the state of being dead.

2. For complex sentences we use prior experiences for deciphering meaning. This process is made more complex by the fact that in English language, a single word can act as different parts of speech. For example,

   - **Enraged Cow Injures Farmer with Ax**: the with ax part is describing the farmer and not the cow.

  - **Hospitals are Sued by 7 Foot Doctors**: The phrase"7 foot" are two different ideas: 7 describes the number of doctors and foot describes their specialty of work.
  - **Students cook and serve grandparents**: Grandparents are served food, they are not the food

  More examples are available here:
  `https://www.plainlanguage.gov/resources/humor/funny-headlines/`

3. Figures of speech: We use the example of sarcasm. When we use sarcasm we mean the opposite of what we say. We again use prior experience to identify that the writer doesn't mean what they say. For example, consider a very simple case: "Great! Now my tires are flat". We use our prior knowledge that a flat tire (in most cases) is not a desirable thing and so the word "great" is not used in a positive sense. Things get tricker when this is used in more subtle sense.
4. Not all ideas are available in all languages: As a very artificial example, no 16^{th} century Indian language would have a word for potato simply because no one knew what a potato was! Some more examples are available here:
   `https://www.babbel.com/en/magazine/untranslatable-01`
5. As someone once said, language is a living thing. New words are always being made to represent new ideas and the internet has accelerated this process. Words like Poggers, KEKW, OC etc. were invented in the era of memes and twitch chat. Usage of unalive (which was not an actual word some time ago) is gaining popularly as the actual word "dead" is often flagged by online platforms.

2.3 Context and Lexical semantics

Each word represents an idea, and when we communicate we chain those ideas. This gives rise to the idea of context. We claim a word gets it's meaning from the words surrounding it. For example, if we have a new word **plompyskompy** and we say it is used as follows: "There is an apple **plompyskompy** on the table" you might (again, due to your experience) understand it represents same kind of idea the word "on" represents. You understand this due to the words surrounding it. This leads us to the idea of Lexical semantics(Ref: `https://en.wikipedia.org/wiki/Lexical_semantics`)

2.4 Language as a Markov process

This will now be a quick discussion. A word gets it's meaning from its surrounding words. The rest of the corpus doesn't matter. What came 100 words ago doesn't matter(unless we are considering what came 100 words ago to be a part of the context of the word). We look at the last few words and by our theory, it should be enough to predict what our next word is going to be. Therefore, if we are looking at a context window of n words then it should be modelled as an n^{th} order Markov chain. Ref:
`https://www.cs.princeton.edu/courses/archive/spr05/cos126/assignments/markov.html`

2.5 Language as a stochastic process

Languages can also be modelled as stochastic process, i.e. we drop the assumption that local context is all that is needed. And it should be noted that the memoryless property of Markov processes are counterintuitive to our claim that experience plays an important role in decoding ideas from sentences. Ref: `https://openreview.net/forum?id=pMQwKL1yctf`

2.6 Crude early models and their shortcomings

Traditionally, a computer understands a word by using a dictionary and looking up synonyms(word with similar meaning), hypernyms(expression of belonging to a more general category), hyponyms(expression of belonging to more specific subclass). A minimal example implemented in NLTK is shown below. Some arrays are truncated for brevity.

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synonyms('car')
[['auto', 'automobile', 'machine', 'motorcar'], ['railcar', 'railroad_car',
    'railway_car'], ['gondola'], ['elevator_car'], ['cable_car']]
>>> wn.synsets('car')[:2]
[Synset('car.n.01'), Synset('car.n.02')]
>>> wn.synset('car.n.01').definition()
'a motor vehicle with four wheels; usually propelled by an internal combustion
    engine'
>>> wn.synset('car.n.01').examples()
['he needs a car to get to work']
>>> wn.synset('car.n.01').hypernyms()
[Synset('motor_vehicle.n.01')]
>>> wn.synset('dog.n.01').hypernyms()
[Synset('canine.n.02'), Synset('domestic_animal.n.01')]
>>> wn.synset('car.n.01').hyponyms()[:5]
[Synset('ambulance.n.01'), Synset('beach_wagon.n.01'), Synset('bus.n.04'),
    Synset('cab.n.03'), Synset('compact.n.03')]
```

But this is not a very robust solution: dependency on context and nuances are not resolved(Ex: ”crimson” is a synonym of ”red” but ”the color of an apple is crimson” is a weird sounding sentence). Moreover, language is ever-changing and keeping track of an ever changing In traditional bag of words approach, words are given a one hot vector representation. For example:

$$\text{"hotel"} = [0, 0, 0, 0, 0, 1, 0, 0, 0]$$
$$\text{"motel"} = [0, 0, 0, 0, 0, 0, 0, 1, 0]$$

But this approach has its own limitations:

1. Vector size increase with increase in vocabulary
2. We understand that the words above are similar, but that similarity is not reflected in such one-hot encoding.

Chapter 3

Word Embeddings

3.1 Why vectors?

We would like to have a structure

1. that can be modelled using a small(<- this depends on what your idea of small is, but we would like to have a memory efficient approach) number of parameters
2. that has an inbuilt idea of similarity
3. that can be easily manipulated

As it turns out finite dimensional vector spaces are perfect for this task: the number of parameters is the number of dimensions, a metric can be used as notion of similarity, and we already have tools to manipulate them.

3.2 Data Preparation

We need to prepare the data or the next two models. We select a window size n. From the corpus, n consecutive words are sampled. The middle word is the target word and the rest is the context. Each such target word, context pair is a data point.

3.3 One-hot Representation of Words and Context

Once the data preparation is done we will have a set of data points of the form (S, W) where S is the context and W is the word. If the word is in the i^{th} place in the vocabulary V then we represent W the one-hot vector e_i (i^{th} element of the standard basis). If the context contains the words $W_1 = V[\alpha_1], W_2 = V[\alpha_2] \ldots W_n = V[\alpha_n]$ then we will represent the context S as $\sum_{i=1}^{n} e_{\alpha_i}$, i.e. the sum of the one-hot encoding of the individual words. For the rest of this discussion I will use the notation $w_i, V[i]$ and e_i interchangeably for words.

3.4 Continuous bag of words model(CBOW)

Assume we have a data point(we have taken size 5 for the example, but this should be adjusted as needed) as follows:

Word1 Word 2 Word Word3 Word 4

We model a classification problem where given context {Word1, Word2, Word3, Word4} we wish to classify which word this set provides context for.[Note, when we represent it as a set, we automatically let go the notion of a sequence- hence the name 'bag of words']. For this we shall use a feed forward network described below:

1. **Input** :For context $S \subset V$, we make an input vector v where:

$$v_S = \sum_{w_i \in S} e_i$$

2. **Hidden Layer 1**: This is obtained by multiplying v by a matrix W. We denote:

$$h_S = W v_S$$

   For a single word w_i, $h_i = W e_i = i^{th}$ column of W will represent the hidden representation of w_i. Therefore, if our context contains is $S = \{w_{\alpha_1}, w_{\alpha_2}, w_{\alpha_3}, w_{\alpha_4} \dots w_{\alpha_n}\}$ then our hidden representation of the context will be $h_S = W\left(\sum_{k=1}^{n} e_{\alpha_k}\right) = \sum_{k=1}^{n} h_{w_{\alpha_k}}$ (Note how our choice of using a linear space as structure of choice and setting the context as the linear combination of its constituents comes handy here)

3. **Hidden Layer 2**: We use another matrix W' and get

$$y = W' h_S$$

4. **Hidden Layer 3**: This is just a soft max to normalize things:

$$\hat{y} = \text{Softmax}(y)$$

We use cross-entropy as our loss function of choice to train the model. $\hat{y}$ is compared to the one-hot representation of the word.
We have:

$$\hat{y} = Softmax(W'(h_S))$$

But as h_S is independent of all h_w where $w \notin S$. Therefore, we don't compute Wv_s nor do we keep track of all the weights. We look only at those columns of W that correspond to words in the context and don't care about the rest.

3.4.a Interpretation on a two word window

Assume we have a two word window. We use a data point (w_α, w_β). Note that:

$$\mathcal{L}(w_\beta) = -\log \hat{y}_\beta = -\log\left(\frac{e^{y_\beta}}{\sum_r e^{y_r}}\right)$$

We do back propagation and calculate the gradients:

$$\frac{\partial \mathcal{L}}{\partial y_i} = \hat{y}_i - \delta(i, \beta)$$

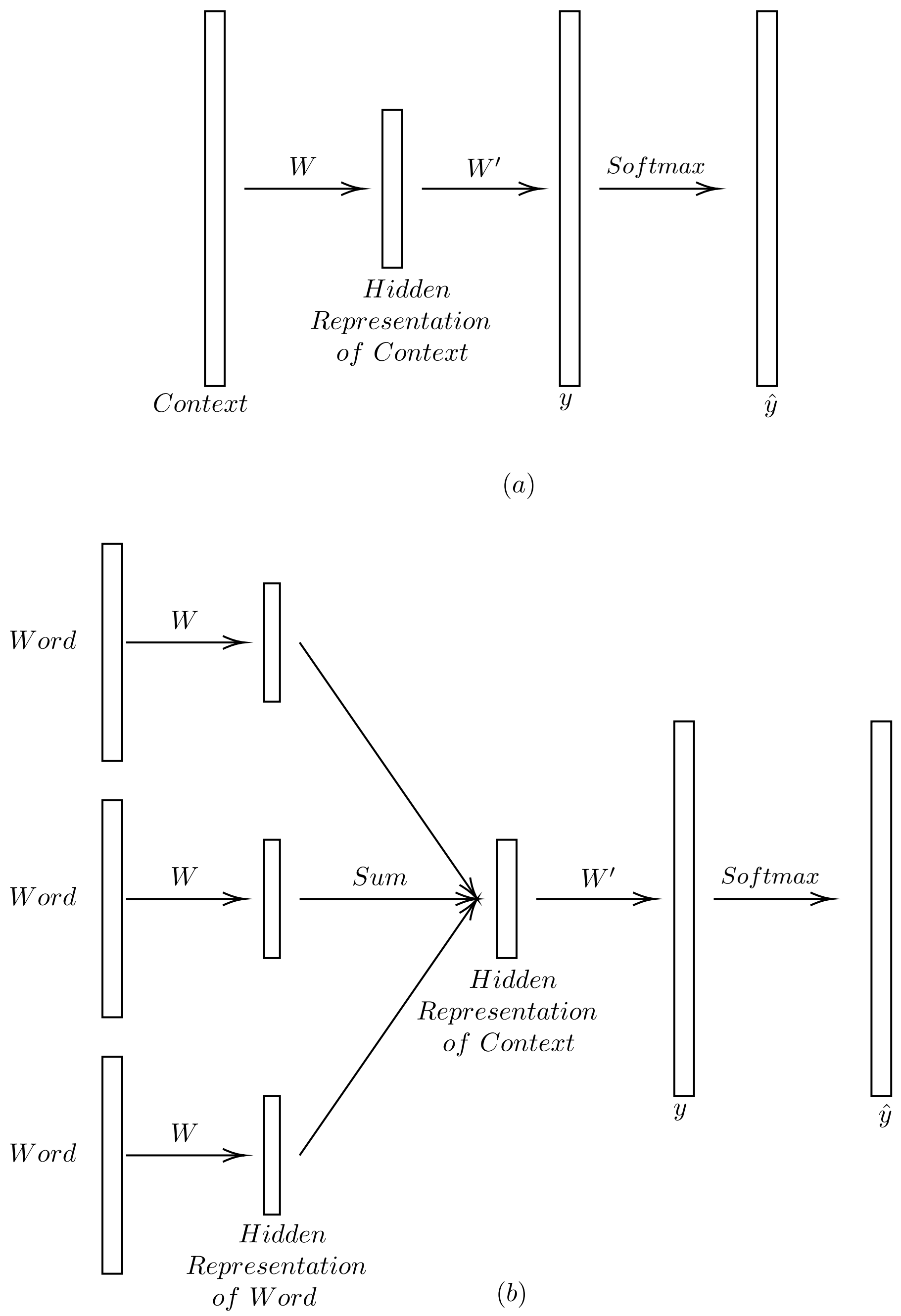

Figure 3.1: Equivalent expression of CBOW

Denote this as E_i and define $E = [E_1, E_2 \dots E_n]$. Note $E = \hat{y} -$Observed y

$$\frac{\partial \mathcal{L}}{\partial h_i} = \sum_j \frac{\partial \mathcal{L}}{\partial y_j} \frac{\partial y_j}{\partial h_i} = \sum_j E_j \frac{\partial \sum_k h_k W'_{k,j}}{\partial h_i} = \sum_j E_j W'_{ij}$$

Therefore the gradient with respect to h is

$$\nabla \mathcal{L} = W'^T E$$

This gives the update rules for h as

$$h := h - \eta \nabla \mathcal{L} = h - \eta W \hat{y} + \eta W \text{ Observed } y$$

Assume initial entries of W to be small. Then and therefore, entries of h_{Cat} and h_{Dog} will have similar starting values: almost 0. Look at the following sentences:

Cat sleeps
Dog sleeps

Note in both those cases, Observed y will be same and therefore h_{cat} and h_{dog} will be adjusted in the same direction when they appear in similar context. This idea makes sure that similar words will have similar cosine similarity due to a form of transitive action.

3.5 Skip-Gram

Skip-gram in a sense is CBOW trained in opposite direction. In this case we use a word to classify what context it comes from. The structure of the model is almost same as before.

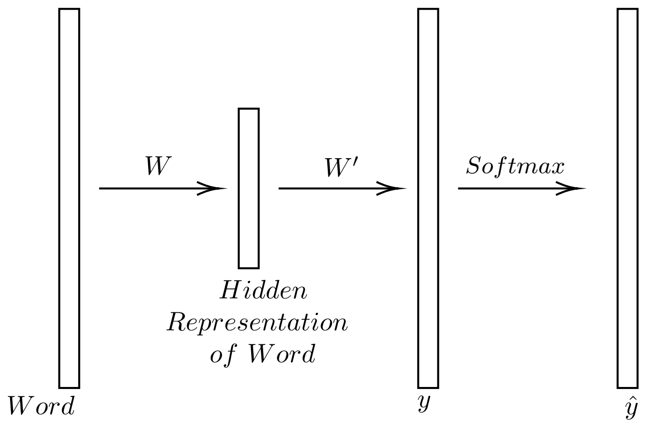

Figure 3.2: Schematic diagram of Skip gram

1. **Input :**For word w_i, the input is $v = e_i$
2. **Hidden Layer 1**: This is obtained by multiplying v by a matrix W. We denote:

$$h = Wv$$

3. **Hidden Layer 2**: We use another matrix W' and get

$$y = W'h$$

4. **Hidden Layer 3**: This is just a soft max to normalize things:

$$\hat{y} = \text{Softmax}(y)$$

The target output is the one-hot representation of the context. If we have a window size of n our loss function, cross-entropy is the sum of $n-1$ terms. For example, if the data point has context $w_\alpha, w_\beta, w_\gamma, w_\delta$ then the loss function will be:

$$\mathcal{L} = -\log \hat{y}_\alpha - \log \hat{y}_\beta - \log \hat{\hat{y}}_\gamma - \log \hat{y}_\delta$$

Note that gradients are summed. For each term, the update to h is the same as the one for CBOW. We again note that only the i^{th} column of W is updated: therefore we need not keep track of the rest of the weights. The interpretation of CBOW is carried over, this time with multi-word windows. For the sentences:

My pet Cat sleeps
My pet Dog sleeps

Since both gives the same Observed y, h_{Cat} and h_{Dog} are adjusted in same direction, making them similar.

3.6 Negative sampling

To improve performance, we might further want to make sure that words which are not similar are further apart. This gives rise to the idea of negative sampling. We make randomized context-word pairs which don't occur in the corpus. Therefore, we have two sets: $D = \{(S, W) \in \mathit{Corpus}\}, D' = \{(S, W) \notin \mathit{Corpus}\}$. Define a random variable Z as:

$$Z = \sigma\left(\hat{y}_S^T \cdot W\right) \text{ or } \sigma\left(S^T \cdot \hat{y}_W\right)$$

We would like to maximize:

$$P(Z = 1 | (S, W) \in D)$$
$$P(Z = 0 | (S, W) \in D')$$

We use now compute the log-likelihood function as follows:

$$\underset{\theta}{\text{maximize}} \prod_{(w,c)\in D} p(z=1 \mid w,c) \prod_{(w,r)\in D'} p(z=0 \mid w,r)$$

$$= \underset{\theta}{\text{maximize}} \prod_{(w,c)\in D} p(z=1 \mid w,c) \prod_{(w,r)\in D'} (1 - p(z=1 \mid w,r))$$

$$= \underset{\theta}{\text{maximize}} \sum_{(w,c)\in D} \log p(z=1 \mid w,c) + \sum_{(w,r)\in D'} \log(1 - p(z=1 \mid w,r))$$

$$= \underset{\theta}{\text{maximize}} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v_c^T v_w}} + \sum_{(w,r)\in D'} \log \frac{1}{1 + e^{v_r^T v_w}}$$

$$= \underset{\theta}{\text{maximize}} \sum_{(w,c)\in D} \log \sigma\left(v_c^T v_w\right) + \sum_{(w,r)\in D'} \log \sigma\left(-v_r^T v_w\right)$$

This leads to a better loss function. Ideally we would want k bad w, s' pairs in D' for each correct $w.s$ in D. Since the number of bad combinations is significantly more than the number of correct combinations, $k > 1$, where k is a hyperparameter to be tuned. The samples are picked from a modified unigram distribution with $p(r) \sim count(r)^p$. The original word2vec used $p = 0.75$, but it can also be trained as a hyperparameter. Unless we properly tune k and p, SVD gives better performance.

3.7 Contrastive Estimation

The problem is soft-max is expensive. So instead of using soft max, we give it a score. Assume for elements of D we get a total score of s and for elements of D' we get a total score of s'. We make our objective to maximise $max(0, s - s - m)$: We claim that if the difference is m, then we are happy with the contrast.

3.8 SVD to learn word representations

We consider two arrays: a corpus C and a vocabulary V. We assume that the context of a word is given by words at a distance d away from it. Define a co-occurance matrix $\mathcal{M}$ defined as:

$$\mathcal{M}[i][j] = \{\text{Number of occurrence of } w_i \text{ and } w_j \text{ with distance at most } d\}$$

Consider the SVD decomposition of $\mathcal{M}$ as $\mathcal{M} = U \sum V$. Represent the columns of U as U_is and the rows of V as V_i^T. The k^{th} rank approximation of $\mathcal{M}$ is given by:

$$\hat{\mathcal{M}}_k = \sum_{i=1}^{k} \sigma_i U_i V_i^T$$

This representation brings out latent relations between variables. Now we define the representation of the i^{th} word as the i^{th} row of $U \sum$. This reduces the size of representation while keeping the cosine similarity fixed.

3.9 GloVe Representations

We combine count based methods(like SVD) and Prediction based methods(CBOW, Skip gram). For words $V[i]$ and $V[j]$ we would like to have a representations h_i, h_j that is faithful to $\mathcal{M}$ i.e:

$$h_i \cdot h_j = \log(j|i) = \log \mathcal{M}[i][j] - \log X_i$$

$$h_j \cdot h_i = \log(i|j) = \log \mathcal{M}[j][i] - \log X_j$$

Where X_i, X_j are the counts of occurrences of X_i and X_j We combine to get:

$$h_i \cdot h_j = \mathcal{M}[i][j] - \frac{1}{2} (\log X_i + \log X_j)$$

But the problem with this is it weights all co-occurances equally. To circumvent this issue, We replace X_i, X_j as trainable weights b_i, b_j. Therefore, our loss function becomes:

$$\sum_{w_i, w_j} (\mathcal{M}[i][j] - b_i - b_j - h_i \cdot h_j))$$

3.10 Other embeddings

Suppose we have some unstructured data with some underlying notion of similarity(For example: Faces, Words, Sentences, Graphs etc.). We can generalise the above-mentioned process of embeddings to get an alternate representation of such data in a low dimensional vector space while retaining their inherent notion of similarity.

Part III

Project: Using Deep Learning To Predict and Trade derivatives

Chapter 4

Derivatives, Options and Futures

Chapter 5

Sequence Modelling

Part IV

Codes

Chapter 6

Tensorflow ANN Cookbook

6.1 Introduction

6.2 Include Script

We will be using TensorFlow for our works here. The `include` script looks like the following:

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib
```

6.3 Loading Data

For the examples we mentioned above we can simply use already available datasets:

```
fashion_mnist = tf.keras.datasets.fashion_mnist.load_data()
```

However, I have downloaded the data as external files, and will use them to try the more general approach.
Ref: `https://www.tensorflow.org/tutorials/load_data/pandas_dataframe`

```
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist
X_train, y_train = X_train_full[:-5000], y_train_full[:-5000]
X_valid, y_valid = X_train_full[-5000:], y_train_full[-5000:]
```

Now define the class labels:

```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat","Sandal",
    "Shirt", "Sneaker", "Bag", "Ankle boot"]
```

And we are ready to go. Things to note:

1. Inputs must be `numpy` arrays.