Natural Language Processing with Deep Learning

Word Window Classification with Neural Networks
Classification setup and notation

• Generally we have a training dataset consisting of samples

\[ \{x_i, y_i\}_{i=1}^{N} \]

• \( x_i \) are inputs, e.g. words (indices or vectors!), sentences, documents, etc.
  • Dimension \( d \)

• \( y_i \) are labels (one of \( C \) classes) we try to predict, for example:
  • classes: sentiment, named entities, buy/sell decision
  • other words
  • later: multi-word sequences
Classification intuition

• Training data: \( \{x_i, y_i\}_{i=1}^{N} \)

• Simple illustration case:
  - Fixed 2D word vectors to classify
  - Using softmax/logistic regression
  - Linear decision boundary

• Traditional ML/Stats approach: assume \( x_i \) are fixed, train (i.e., set) softmax/logistic regression weights \( W \in \mathbb{R}^{C \times d} \) to determine a decision boundary (hyperplane) as in the picture

• Method: For each \( x \), predict:

\[
p(y|x) = \frac{\exp(W_{y,x})}{\sum_{c=1}^{C} \exp(W_{c,x})}
\]
Details of the softmax classifier

\[ p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)} \]

We can tease apart the prediction function into two steps:

1. Take the \( y^{th} \) row of \( W \) and multiply that row with \( x \):

\[ W_y.x = \sum_{i=1}^{d} W_{yi} x_i = f_y \]

Compute all \( f_c \) for \( c = 1, ..., C \)

2. Apply softmax function to get normalized probability:

\[ p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} = \text{softmax}(f_y) \]
Training with softmax and cross-entropy loss

- For each training example $(x, y)$, our objective is to maximize the probability of the correct class $y$.

- Or we can minimize the negative log probability of that class:

\[
- \log p(y|x) = - \log \left( \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} \right)
\]
Background: What is “cross entropy” loss/error?

- Concept of “cross entropy” is from information theory
- Let the true probability distribution be $p$
- Let our computed model probability be $q$
- The cross entropy is:

$$H(p, q) = - \sum_{c=1}^{C} p(c) \log q(c)$$

- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: $p = [0,...,0,1,0,...0]$ then:
- Because of one-hot $p$, the only term left is the negative log probability of the true class
Classification over a full dataset

- Cross entropy loss function over full dataset \( \{x_i, y_i\}_{i=1}^{N} \)

\[
J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_{c}}} \right)
\]

- Instead of

\[
f_y = f_y(x) = W_y.x = \sum_{j=1}^{d} W_{yj}x_j
\]

We will write \( f \) in matrix notation:

\[
f = Wx
\]
Traditional ML optimization

• For general machine learning $\theta$ usually only consists of columns of $W$:

$$\theta = \begin{bmatrix} W_1 \\ \vdots \\ W_d \end{bmatrix} = W(:,) \in \mathbb{R}^{Cd}$$

• So we only update the decision boundary via

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla W_1 \\ \vdots \\ \nabla W_d \end{bmatrix} \in \mathbb{R}^{Cd}$$
Neural Network Classifiers

• Softmax (≈ logistic regression) alone not very powerful
• Softmax gives only linear decision boundaries

- This can be quite limiting
- Unhelpful when a problem is complex
- Wouldn’t it be cool to get these correct?
Neural Nets for the Win!

- Neural networks can learn much more complex functions and nonlinear decision boundaries!
  - In original space
Classification difference with word vectors

- Commonly in NLP deep learning:
  - We learn **both** $W$ and word vectors $x$
  - We learn **both** conventional parameters and representations
  - The word vectors re-represent one-hot vectors—move them around in an intermediate layer vector space—for easy classification with a (linear) softmax classifier via layer $x = Le$

\[
\nabla_{\theta} J(\theta) = \begin{bmatrix}
  \nabla W_1 \\
  \vdots \\
  \nabla W_d \\
  \nabla x_{aardvark} \\
  \vdots \\
  \nabla x_{zebra}
\end{bmatrix} \in \mathbb{R}^{Cd+Vd}
\]

Very large number of parameters!
An artificial neuron

- Neural networks come with their own terminological baggage
- But if you understand how softmax models work, then you can easily understand the operation of a neuron!
A neuron can be a binary logistic regression unit

\[ f = \text{nonlinear activation fct. (e.g. sigmoid)}, \ w = \text{weights}, \ b = \text{bias}, \ h = \text{hidden}, \ x = \text{inputs} \]

\[ h_{w,b}(x) = f(w^T x + b) \]

\[ f(z) = \frac{1}{1 + e^{-z}} \]

\( b \): We can have an “always on” feature, which gives a class prior, or separate it out, as a bias term

\( w, b \) are the parameters of this neuron i.e., this logistic regression model
A neural network = running several logistic regressions at the same time

If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs ...

But we don’t have to decide ahead of time what variables these logistic regressions are trying to predict!
A neural network = running several logistic regressions at the same time

... which we can feed into another logistic regression function

> It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.
Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

Activation $f$ is applied element-wise:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$
Non-linearities (aka “f”): Why they’re needed

- Example: function approximation, e.g., regression or classification
  - Without non-linearities, deep neural networks can’t do anything more than a linear transform
  - Extra layers could just be compiled down into a single linear transform: $W_1 W_2 x = Wx$
  - With more layers, they can approximate more complex functions!
The European Commission [ORG] said on Thursday it disagreed with German [MISC] advice.

Only France [LOC] and Britain [LOC] backed Fischler [PER] 's proposal .

“What we have to be extremely careful of is how other countries are going to take Germany 's lead”, Welsh National Farmers ' Union [ORG] ( NFU [ORG] ) chairman John Lloyd Jones [PER] said on BBC [ORG] radio .

Possible purposes:

• Tracking mentions of particular entities in documents
• For question answering, answers are usually named entities
• A lot of wanted information is really associations between named entities
• The same techniques can be extended to other slot-filling classifications
• Often followed by Named Entity Linking/Canonicalization into Knowledge Base
Named Entity Recognition on word sequences

We predict entities by classifying words in context and then extracting entities as word subsequences

Foreign    ORG    }       B-ORG
Ministry   ORG    }       I-ORG
spokesman  O       }       O
Shen       PER    }       B-PER
Guofang    PER    }       I-PER
told       O       }       O
Reuters    ORG    }       B-ORG
that       O       }       O

👍 BIO encoding
Why might NER be hard?

• Hard to work out boundaries of entity

First National Bank Donates 2 Vans To Future School Of Fort Smith

Is the first entity “First National Bank” or “National Bank”

• Hard to know if something is an entity

Is there a school called “Future School” or is it a future school?

• Hard to know class of unknown/novel entity:

To find out more about Zig Ziglar and read features by other Creators Syndicate writers and

What class is “Zig Ziglar”? (A person.)

• Entity class is ambiguous and depends on context

“Charles Schwab” is PER not ORG here! 👉

where Larry Ellison and Charles Schwab can live discreetly amongst wooded estates. And
Binary word window classification

• In general, classifying single words is rarely done

• Interesting problems like ambiguity arise in context!

• Example: auto-antonyms:
  • "To sanction" can mean "to permit" or "to punish"
  • "To seed" can mean "to place seeds" or "to remove seeds"

• Example: resolving linking of ambiguous named entities:
  • Paris → Paris, France vs. Paris Hilton vs. Paris, Texas
  • Hathaway → Berkshire Hathaway vs. Anne Hathaway
Window classification

• **Idea**: classify a word in its context window of neighboring words.

• For example, **Named Entity Classification** of a word in context:
  • Person, Location, Organization, None

• A simple way to classify a word in context might be to average the word vectors in a window and to classify the average vector
  • Problem: that would lose position information
Window classification: Softmax

- Train softmax classifier to classify a center word by taking concatenation of word vectors surrounding it in a window.

- **Example**: Classify “Paris” in the context of this sentence with window length 2:

  \[
  X_{\text{window}} = \begin{bmatrix}
  x_{\text{museums}} & x_{\text{in}} & x_{\text{Paris}} & x_{\text{are}} & x_{\text{amazing}}
  \end{bmatrix}^T
  \]

- Resulting vector \( x_{\text{window}} = x \in \mathbb{R}^{5d} \), a column vector!
Simplest window classifier: Softmax

- With $x = x_{\text{window}}$ we can use the same softmax classifier as before

$$\hat{y}_y = p(y|x) = \frac{\exp(W_{y,x})}{\sum_{c=1}^{C} \exp(W_{c,x})}$$

- With cross entropy error as before:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}} \right)$$

- How do you update the word vectors?
- Short answer: Just take derivatives like last week and optimize
Binary classification with unnormalized scores

Method used by Collobert & Weston (2008, 2011)

• Just recently won ICML 2018 Test of time award

• For our previous example:

\[ X_{\text{window}} = [x_{\text{museums}} \quad x_{\text{in}} \quad x_{\text{Paris}} \quad x_{\text{are}} \quad x_{\text{amazing}}] \]

• Assume we want to classify whether the center word is a Location

• Similar to word2vec, we will go over all positions in a corpus. But this time, it will be supervised and only some positions should get a high score.

• E.g., the positions that have an actual NER Location in their center are “true” positions and get a high score
Binary classification for NER Location

- Example: Not all museums in Paris are amazing.
- Here: one true window, the one with Paris in its center and all other windows are “corrupt” in terms of not having a named entity location in their center.

  museums in Paris are amazing

- “Corrupt“ windows are easy to find and there are many: Any window whose center word isn’t specifically labeled as NER location in our corpus

  Not all museums in Paris
Neural Network Feed-forward Computation

Use neural activation $a$ simply to give an unnormalized score

$$\text{score}(x) = U^T a \in \mathbb{R}$$

We compute a window’s score with a 3-layer neural net:

- $s = \text{score}("museums in Paris are amazing")$

$$s = U^T f(Wx + b)$$

$x \in \mathbb{R}^{20 \times 1}, W \in \mathbb{R}^{8 \times 20}, U \in \mathbb{R}^{8 \times 1}$

$$x_{\text{window}} = [ x_{\text{museums}}, x_{\text{in}}, x_{\text{Paris}}, x_{\text{are}}, x_{\text{amazing}} ]$$
Main intuition for extra layer

The middle layer learns non-linear interactions between the input word vectors.

Example: only if “museums” is first vector should it matter that “in” is in the second position

\[ X_{\text{window}} = [ x_{\text{museums}} \ x_{\text{in}} \ x_{\text{Paris}} \ x_{\text{are}} \ x_{\text{amazing}} ] \]
The max-margin loss

- **Idea for training objective:** Make true window’s score larger and corrupt window’s score lower (until they’re good enough)
- $s = \text{score(} \text{museums in Paris are amazing}\text{)}$
- $s_c = \text{score(} \text{Not all museums in Paris}\text{)}$
- **Minimize**

\[
J = \max(0, 1 - s + s_c)
\]

- This is not differentiable but it is continuous $\Rightarrow$ we can use SGD.
Max-margin loss

- Objective for a single window:

\[ J = \max(0, 1 - s + s_c) \]

- Each window with an NER location at its center should have a score +1 higher than any window without a location at its center

- For full objective function: Sample several corrupt windows per true one. Sum over all training windows.

- Similar to negative sampling in word2vec
Simple net for score

\[ s = u^T h \]

\[ h = f(Wx + b) \]

\[ x \quad (\text{input}) \]

\[ x = [ x_{\text{museums}} \ x_{\text{in}} \ x_{\text{paris}} \ x_{\text{are}} \ x_{\text{amazing}} ] \]
Remember: Stochastic Gradient Descent

- Update equation:
  \[ \theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_{\theta} J(\theta) \]
  \( \alpha = \text{step size or learning rate} \)

- How do we compute \( \nabla_{\theta} J(\theta) \)?
  - By hand
  - Algorithmically: the backpropagation algorithm