Mining Unstructured Data

9. Word Classification

Javier Ferrando

Recap
Sparse Word Vectors

Rome = [1, 0, 0, 0, 0, 0, ..., 0]
Paris = [0, 1, 0, 0, 0, 0, ..., 0]
Italy = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]

One-hot encoding

<table>
<thead>
<tr>
<th></th>
<th>As You Like It</th>
<th>Twelfth Night</th>
<th>Julius Caesar</th>
<th>Henry V</th>
</tr>
</thead>
<tbody>
<tr>
<td>battle</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>good</td>
<td>114</td>
<td>80</td>
<td>62</td>
<td>89</td>
</tr>
<tr>
<td>fool</td>
<td>36</td>
<td>58</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>wit</td>
<td>20</td>
<td>15</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Term-document matrix

<table>
<thead>
<tr>
<th></th>
<th>aardvark</th>
<th>...</th>
<th>computer</th>
<th>data</th>
<th>result</th>
<th>pie</th>
<th>sugar</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>cherry</td>
<td>0</td>
<td>...</td>
<td>2</td>
<td>8</td>
<td>9</td>
<td>442</td>
<td>25</td>
<td>...</td>
</tr>
<tr>
<td>strawberry</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>60</td>
<td>19</td>
<td>...</td>
</tr>
<tr>
<td>digital</td>
<td>0</td>
<td>...</td>
<td>1670</td>
<td>1683</td>
<td>85</td>
<td>5</td>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>information</td>
<td>0</td>
<td>...</td>
<td>3325</td>
<td>3982</td>
<td>378</td>
<td>5</td>
<td>13</td>
<td>...</td>
</tr>
</tbody>
</table>

Term-term matrix
Sparse Word Vectors

- Improve word vectors from term-document matrix with tf-idf:
  \[ w_{i,d} = tf_{i,d} \times idf_t \]  
  Penalizing frequent words

- Improve word vectors from term-term matrix with PPMI:
  \[ \text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \]  
  how much more the two words co-occur in our corpus than we would have a priori expected them to appear by chance.
Dense Word Vectors

- Low-dimensional floating-point vectors (100, 200 components)
- Geometric relationship between two word vectors reflects the semantic relationship between these words
- Synonyms are embedded into similar word vectors
Word2Vec

\[ P(w_{t-2} \mid w_t) \]
\[ P(w_{t-1} \mid w_t) \]
\[ P(w_{t+1} \mid w_t) \]
\[ P(w_{t+2} \mid w_t) \]

... problems turning into banking crises as ...

outside context words in window of size 2
center word at position t
outside context words in window of size 2

Skip-gram

CBOB is a group of related models that are used to produce word embeddings

CBOB
Word2Vec
Index

- Classification
  - Binary Logistic Regression
  - Multimodal Logistic Regression

- Neural Networks
  - Neuron
  - Neural Networks
  - Non-linearities
  - Training with softmax and cross-entropy loss

- Named Entity Recognition (NER)
  - NER with window classification and NN
  - Training
  - Key Concepts in Deep Learning
Classification
Classification setup and notation

- Generally we have a training dataset consisting of samples:
  \[ \{ x_i, y_i \}^N_{i=1} \]

- \( x_i \) 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

- Tradition ML approach: assume $x_i$ are fixed, train softmax/logistic regression weights to determine a decision boundary
Binary Logistic Regression

- Classify each data point $x$ as one of two classes ($c_0, c_1$)

- We need to model the conditional probability of a class given the input data point $x$: $p(c_1|x)$

- We use the sigmoid function to do so:
  $$p(c_1|x; \theta) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- The linear classifier $f(x)$ predicts the class that is more likely:
  $$f(x) = \mathbb{I}(p(c_1|x) > p(c_2|x))$$
Binary Logistic Regression

- Logits are computed by the dot product between the weight vector $\mathbf{w}$ and the feature vector $\mathbf{x}$:

$$ z = \mathbf{w}^\top \mathbf{x} + b $$

$$ p(c_1 | \mathbf{x}; \theta) = \sigma(\mathbf{w}^\top \mathbf{x} + b) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x} + b}} $$

- The decision boundary is defined by:

$$ \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x} + b}} = 0.5 $$

$$ \mathbf{w}^\top \mathbf{x} + b = 0 $$

hyperplane
Binary Logistic Regression

- If we can perfectly separate the training examples by such a linear boundary we say the data is **linearly separable**

- Most of the data is not linearly separable, but we can transform it:
  - In the pre-processing
  - By letting the model learn the transformation

That’s what neural networks do!
Multinomial Logistic Regression

- Multinomial Logistic Regression lets us classify between more than 2 classes (multi-class classification)

\[ p(c_k|x; \theta) = \frac{e^{z_k}}{\sum_j e^{z_j}} \]

- \(a\) is \(C\)-dimensional vector of logits
- \(W\) is a \(C\times D\)
- We get rid of the bias term by appending a 1 in \(x\) and adding \(b\) to the last column of \(W\)
Neural Networks
Neural Computation
Perceptron (Rosenblatt, 1958)

- Heaviside step function
- Non-differentiable function -> no gradient-based optimization -> perceptron learning algorithm

\[ f(x) = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \]
Neuron

\[ p(c_1 | x; \theta) = \sigma(w^T x + b) = \frac{1}{1 + e^{-w^T x + b}} \]

Binary logistic regression!
Neural Network = several logistic regression 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 ...

\[ x_1, x_2, x_3, x_4, +1 \rightarrow z_1, z_2, z_3 \]

\[ W \in \mathbb{R}^{C \times D} \]

\[ z \in \mathbb{R}^C \]

\[ x \in \mathbb{R}^D \]
Neural Network = several logistic regression at the same time

... which we can feed into a softmax function:

\[ p(c_k | x; \theta) = \frac{e^{z_k}}{\sum_j e^{z_j}} \]
Neural Network = several logistic regression at the same time

Multi-layer perceptron (MLP)

\[
p(c_1 | x; \theta) = \sigma(z) = \frac{1}{1 + e^{-z}}
\]

z is no longer lineal
Neural Network = several logistic regression at the same time

Multi-layer perceptron (MLP)

\[
p(c_k | x; \theta) = \frac{e^{z_k}}{\sum_j e^{z_j}}
\]
Neural Network = several logistic regression at the same time

Multi-layer perceptron (MLP)

\[ h_1^2 = f(W_{11}^1 x_1 + W_{12}^1 x_2 + W_{13}^1 x_3 + b_1^1) \]
\[ h_2^2 = f(W_{21}^1 x_1 + W_{22}^1 x_2 + W_{23}^1 x_3 + b_2^1) \]

Activation function \( f \) is applied element-wise

\[ f([z_1^2, z_2^2, z_3^2]) = [f(z_1^2), f(z_2^2), f(z_3^2)] \]
Non-linearities: why do we need them

- 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: \( \mathbf{W}^{1} \mathbf{W}^{2} \mathbf{x} = \mathbf{W} \mathbf{x} \)
- With more layers, they can approximate more complex functions.
## Non-linearities: why do we need them

<table>
<thead>
<tr>
<th>Function</th>
<th>Sigmoid</th>
<th>Tanh</th>
<th>ReLU</th>
<th>Leaky ReLU</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g(z)$</td>
<td>$\frac{1}{1+e^{-z}}$</td>
<td>$\frac{e^z - e^{-z}}{e^z + e^{-z}}$</td>
<td>$\max(0, z)$</td>
<td>$\max(\epsilon z, z)$ with $\epsilon \ll 1$</td>
</tr>
</tbody>
</table>

**Graphs:**
- Sigmoid: A smooth S-shaped curve.
- Tanh: A hyperbolic tangent function.
- ReLU: A step function.
- Leaky ReLU: A modified ReLU with a small leaky part.
Check it yourself

Tensorflow Playground
Training with softmax and cross-entropy loss
Training with softmax and cross-entropy loss

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

\[
p(y|x; \theta) = \frac{e^{z_y}}{\sum_j e^{z_j}} = \frac{e^{W_{y,x}}}{\sum_j e^{W_{j,x}}}
\]

- This is equivalent to minimizing the negative log probability of that class:

\[
- \log p(y|x; \theta) = - \log \left( \frac{e^{W_{y,x}}}{\sum_j e^{W_{j,x}}} \right)
\]

- Using log probability converts our objective function to sums, which is easier to work with on paper and in implementation.
Training with softmax and cross-entropy loss

- 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 loss 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: \( p = [0, \ldots, 0, 1, 0, \ldots, 0] \)
- 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 \( \{x_i, y_i\}_{i=1}^{N} \)

\[
J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{y_i x_i}}{\sum_{j=1}^{C} e^{x_j}} \right) \quad \theta = \begin{bmatrix} W_1. \\ \vdots \\ W_C. \end{bmatrix}
\]

- We only update decision boundary via

\[
\nabla J(\theta) = \begin{bmatrix} \nabla W_1. \\ \vdots \\ \nabla W_C. \end{bmatrix}
\]
Word Classification

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
  - Words vectors are moved around in an intermediate layer vector space—for easy classification with a (linear) softmax classifier

\[
\nabla_\theta J(\theta) = \begin{bmatrix}
\nabla W_{.1} \\
\vdots \\
\nabla W_{.d} \\
\nabla x_{aardvark} \\
\vdots \\
\nabla x_{zebra}
\end{bmatrix}
\]
Named Entity Recognition (NER): example of Word Classification task
A named entity is anything that can be referred to with a proper name: a person, a location, an organization.

The task of named entity recognition (NER) is to find spans of text that constitute proper names and tag the type of the entity.

- **PER** (person), **LOC** (location), **ORG** (organization)
Named Entity Recognition (NER)

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
  - In sentiment analysis we might want to know a consumer’s sentiment toward a particular entity.
Named Entity Recognition (NER)

- Idea: classify each word in its context window of neighboring words

- 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 the concatenation of words surrounding in a window.

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

  ... museums in Paris are amazing ... .

  \[
  \mathbf{x}_{\text{window}} = [ x_{\text{museums}} \ x_{\text{in}} \ x_{\text{Paris}} \ x_{\text{are}} \ x_{\text{amazing}} ]^T
  \]

- Resulting vector \( \mathbf{x}_{\text{window}} \in \mathbb{R}^{5d} \), a column vector!
Window classification: softmax

- With $\mathbf{x} = \mathbf{x}_{\text{window}}$ we can use the softmax classifier

$$p(y | \mathbf{x}; \theta) = \frac{e^{z_y}}{\sum_j e^{z_j}} = \frac{e^{\mathbf{W}_y \mathbf{x}}}{\sum_j e^{\mathbf{W}_j \mathbf{x}}}$$

- With cross entropy error:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{z_{y_i}}}{\sum_{j=1}^{C} e^{z_j}} \right)$$

- How do you update the word vectors?
  - Short answer: Just take derivatives and optimize
Window classification using binary logistic classifier

- Train logistic classifier on hand-labeled data to classify center word \{yes/no\} for each class based on a concatenation of word vectors in a window.

- Example: Classify “Paris” as +/- location in context of sentence with window length 2:

\[
\begin{align*}
\text{... museums in Paris are amazing ...} \\
\hline
\text{... museums in Paris are amazing ...} \\
X_{\text{window}} &= \begin{bmatrix}
x_{\text{museums}} & x_{\text{in}} & x_{\text{Paris}} & x_{\text{are}} & x_{\text{amazing}}
\end{bmatrix}^T
\end{align*}
\]
Window classification using binary logistic classifier

- We do supervised training and want high score if it's a location

\[ J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}} \]

\[ s = w^T h \]

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

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

\[ x = [x_{\text{museums}}, x_{\text{in}}, x_{\text{Paris}}, x_{\text{are}}, x_{\text{amazing}}] \]
Window classification using binary logistic classifier

- We do supervised training and want high score if it's a location
Stochastic Gradient Descent

Update equation gradient descent:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_{\theta} J(\theta)$$

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{z_{y_{i}}}}{\sum_{j=1}^{C} e^{z_{j}}} \right)$$

Update equation stochastic gradient descent (SGD):

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_{\theta} J_{i}(\theta; x_{i}, y_{i})$$

1. Randomly shuffle dataset
2. For every training sample (i) in the dataset -> apply the update rule

We can also update the parameter every minibatch, which means a few number of samples.

How we compute $\nabla_{\theta} J_{i}(\theta; x_{i}, y_{i})$?
Stochastic Gradient Descent

\[ \theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J_i(\theta; x_i, y_i) \]
Gradients

- Given a function with 1 output and \( n \) inputs

\[ f(x) = f(x_1, x_2, \ldots, x_n) \]

- Its gradient is a vector of partial derivatives with respect to each input

\[ \frac{\partial f}{\partial x} = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right] \]
Jacobian Matrix: Generalization of the Gradient

- Given a function with m outputs and n inputs

\[ f(x) = [f_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)] \]

- It’s Jacobian is an m x n matrix of partial derivatives

\[
\frac{\partial f}{\partial x} = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]
Gradients in our Neural Network

- Let’s find $\frac{\partial s}{\partial b}$

  - Really, we care about the gradient of the loss $J_i$ but we will compute the gradient of the score for simplicity

  $$s = u^T h$$

  $$h = f(Wx + b)$$

  $x$ (input) $\rightarrow$ $x = [x_{\text{museums}}, x_{\text{in}}, x_{\text{Paris}}, x_{\text{are}}, x_{\text{amazing}}]$
Gradients in our Neural Network

\[ s = u^T h \]

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

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

\[ s = u^T h \]

\[ h = f(z) \]

\[ z = Wx + b \]

\[ x \quad \text{(input)} \]
Apply chain rule

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]

\[
\frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b}
\]
Apply chain rule

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \quad \text{(input)} \]
Apply chain rule

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]

\[ \frac{\partial s}{\partial b} = \frac{\partial s}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial b} \]
Computational Graph

- Software represents our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

Forward-pass

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \text{ (input)} \]
Backpropagation

- Then go backwards along edges
  - Pass along gradients

\[ s = u^T h \]
\[ h = f(z) \]
\[ z = Wx + b \]
\[ x \quad \text{(input)} \]
Backpropagation: Single Node

- Node receives an "upstream gradient"
- Goal is to pass on the correct "downstream gradient"
- Each node has a local gradient
  - The gradient of its output with respect to its input

\[ h = f(z) \]
Regularization

- Regularization (largely) prevents overfitting when we have a lot of features (or later a very powerful/deep model, ++)
- L2 Regularization: penalize the growth of parameters
  \[
  J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{z_{y_i}}}{\sum_{j=1}^{C} e^{z_j}} \right) + \lambda \sum_{k} \theta_k^2
  \]
- Dropout: disconnect neurons with certain probability
Summary

- Neural Networks can be more powerful than logistic regression because they are able to capture non-linearities in hidden layers.

- Complex Classification Tasks can successfully be addressed with Neural Networks.

- Named Entity Recognition can be tackled as a Classification Task with a Neural Network.