# Optimal Transport in Machine Learner: Word Mover's Distance

James Nesbit

June 20, 2019

# Motivation

- Text is an increasingly popular input in empirical economic research
  - Stock market returns using financial news (Tetlock [07])
  - Political slant of media outlets (Groseclose and Milyo [05])
  - Understand macro policy using records of policy actions (Romer, Romer [04, 10])
- Text is high-dimensional data: need dimensionality reduction
  - FOMC transcripts 1987–2006: 46K+ docs, 5M+ words, 24k unique terms TODO

## Document Similarity

- A common question is how similar are two documents?
- Consider the following two 'documents'
  1. **Obama speaks** to the **media** in **Illinois**.
  2. The **President greets** the **press** in **Chicago**.
- Obviously these are two sentences convey very similar information, but they have no words in common
  - Let $V$ be the size of the vocabulary, e.g. in this context $V = 8$
  - Represent a document $d$ in the $V - 1$ simplex, where $d_i$ is the frequency of word $i$
  - $\mathbf{d}$ and $\mathbf{d}'$ are as far apart as possible
- Need a method that takes into account semantic information

# Word Embedding

- Popular method in Natural Language Processing (NLP) is a 'dense representation' of words, where a vocabulary is embedded in a low dimensional vector space $\mathbb{R}^N$, where $N \approx 200$
  - Also commonly known as a word embedding
- The idea is that words that are 'similar' to each other will be 'close' in this vector space
- We will use this vector space embedding of **words** to also come up with an idea of **documents** being close together
- Several popular word embedding algorithms
  - **word2vec**
  - GloVe
  - fastText

# word2vec

- We want word vectors where semantically 'similar' words are 'close'
- Idea is use 'context' words, words that occur in neighbourhood of a given word

$$\underbrace{\text{Obama}}_{w_1}\,\underbrace{\text{speaks}}_{w_2}\,\underbrace{\text{media}}_{w_3}\,\underbrace{\text{Illinois}}_{w_4}$$

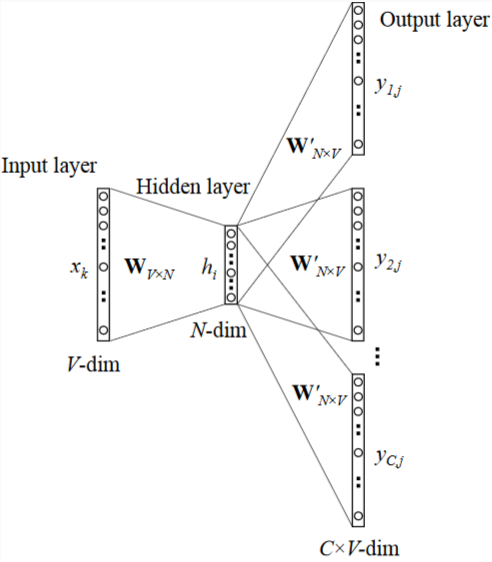
  - For example **speaks** and **Illinois** would be in the neighbourhood of **media**—$nb(3) = \{2, 4\}$,

## word2vec

- We want a model that predicts the context words given the input word
  - Skip-gram: $p(w_2, w_4|w_3)$
  - Continuous Bag of Words (CBOW): $p(w_3|w_2, w_4)$
- We want to maximize the log probability of neighbouring words in a corpus. Given a sequence of words $w_1, \ldots, w_T$

$$max_\theta \frac{1}{T} \sum_{t=1}^{T} \sum_{j \in nb(t)} \log p(w_j|w_t;\theta)$$

# According to ML: Neural Network Architecture



# According to ML: Neural Network Architecture

- Assume for simplicity we have only one word in context
  - In the output layer only worry about $y_{1,j}$
- $V$ terms in vocabulary, $N$ is the size of word embedding space, $T$ is the number of words

**Input layer–Hidden layer**

- If input word is term $k$, then $x_k = 1$ and $x_{k'} = 0$ for $k' \neq k$ (one-hot vector)
- Weights between input and hidden layer are $V \times N$ matrix $\mathbf{W}$.
- Therefore $\mathbf{h} = \mathbf{W}^\intercal \mathbf{x} = \mathbf{W}_{(k,\cdot)} = v_{W_I}^\intercal$
  - $v_{w_I}$ is the row of $\mathbf{W}$ corresponding to the input word $w_I$. We will call this the (input) word vector for $w_I$

# According to ML:Neural Network Architecture

**Hidden layer–Output layer**

- $N \times V$ weight matrix $\mathbf{W}'$, $u = \mathbf{W}'\mathbf{h}$, so $u_j = \mathbf{v}'_{w_j}\mathbf{h}$
  - $v'_{w_j}$ is the $j$th column of $\mathbf{W}'$. The (output) word vector for $w_I$

**Activation function**

- $p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})}$

# According to Econometricians: Neural Network Architecture

- $p(w_j|w_I) = \frac{\exp(\mathbf{v}'_{w_j}\mathbf{v}_{w_I})}{\sum_{j'=1}^{V}\exp(\mathbf{v}'_{w_{j'}}\mathbf{v}_{w_I})}$

# Training the NN: Backpropagation

According to ML

# Training the NN: Backpropagation

- For an individual observation with input word $w_I$ and context word $w_O$

$$
\begin{aligned}
\max p(w_O|w_I) &= \log y_{j^*} \\
&= u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) := -E
\end{aligned}
$$

  where $j^*$ is the index of $w_I$
- In standard ML fashion we are going to use Stochastic Gradient Descent, so let's solve for the derivatives of our parameters
  - Hidden layer–Output layer weights: $\mathbf{W}'$
  - Input layer–Hidden layer weights: $\mathbf{W}$

# Backpropagation—Hidden layer–Output layer weights

- Recall $E = \log \sum_{j'=1}^{V} \exp(u_{j'}) - u_{j^*}$

$$\frac{\partial E}{\partial u_j} = y_j - \mathbf{1}\{j = j^*\} := e_j$$

- And $u_j = \mathbf{v}'_{w_j} \mathbf{h}$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

- Using stochastic gradient descent

$$w'^{(new)}_{ij} = w'^{(old)}_{ij} - \eta e_j \cdot h_i$$

# Backpropagation—Hidden layer–Output layer weights

- Stacking over $i$

$$\mathbf{v}'^{(new)}_{w_j} = \mathbf{v}'^{(old)}_{w_j} - \eta e_j \cdot \mathbf{v}_{w_I}$$

- If $e_j = y_j - \mathbf{1}\{j = j^*\} > 0$, $\mathbf{v}'_{w_j}$ moves further from $\mathbf{v}_{w_I}$

# Backpropagation—Input layer–Hidden layer weights

- Recall $E = \log \sum_{j'=1}^{V} \exp(u_{j'}) - u_{j^*}$ and $u_j = \mathbf{v}'_{w_j}\mathbf{h}$

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := EH_i$$

- $EH$ is $N$ vector, sum of all the output vectors of all words in vocabulary weighted by their prediction errors
- Recall that $h_i = \sum_{k=1}^{V} x_k \cdot w_{ki}$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = EH_i x_k$$

## Backpropagation—Input layer–Hidden layer weights

- Stack across $i$, and noting that $x_k = 1$ only for the input word, so only the row of $\mathbf{W}$ corresponding to the input vector gets updated

$$\mathbf{v}_{w_I}^{(new)} = \mathbf{v}_{w_I}^{(old)} - \eta EH^{\mathsf{T}}$$

- If $e_j = y_j - \mathbf{1}\{j = j^*\} > 0$, $\mathbf{v}_{w_I}$ moves further from $\mathbf{v}_{w_j}$

# Training the NN: Backpropagation

According to Econometricians

## Training the NN: Backpropagation

According to Econometricians

Chain Rule

## Training the NN: Easier derivation

- Let's do the situation with $C$ context words. Now

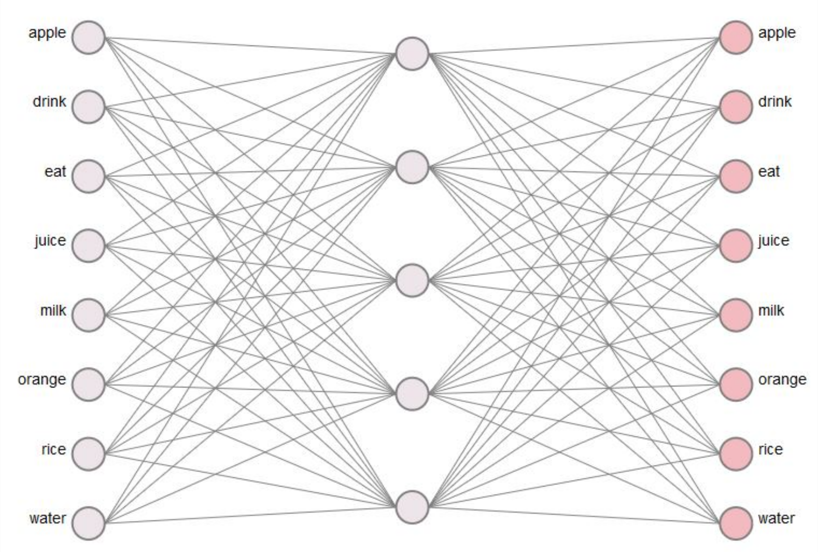
$$
\begin{aligned}
E &= -\log p(w_{O,1}, w_{O,2}, \dots, w_{O,C} | w_I) \\
&= C \cdot \log \sum_{j'=1}^{V} \exp(v'_{w_{j'}} v_{w_I}) - \sum_{c=1}^{V} v'_{w_{j_c^*}} v_{w_I} \\
\frac{\partial E}{\partial v'_{w_j}} &= \left( C \frac{\exp(v'_{w_j} v_{w_I})}{\sum_{j'=1}^{V} \exp(v'_{w_{j'}} v_{w_I})} - \sum_{c=1}^{C} \mathbf{1}\{j = j^*\} \right) v_{w_I} \\
&= v_{w_I} \sum_{c=1}^{V} (y_j - \mathbf{1}\{j_c = j_c^*\}) \\
&= v_{w_I} \sum_{c=1}^{V} e_{j_c}
\end{aligned}
$$

# Training the NN: Easier derivation

- Recall $E = C \cdot \log \sum_{j'=1}^{V} \exp(v'_{w_{j'}} v_{w_I}) - \sum_{c=1}^{V} v'_{w_{j_c^*}} v_{w_I}$

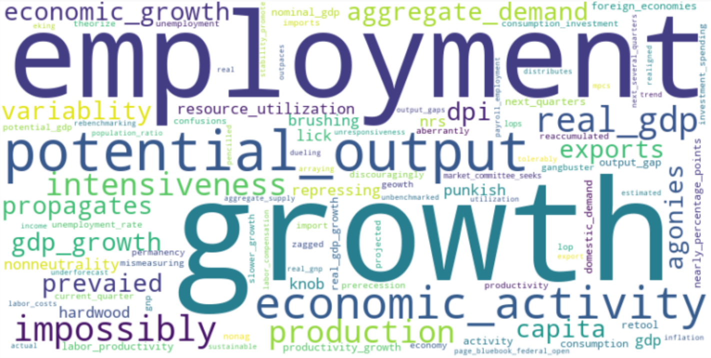
$$
\begin{aligned}
\frac{\partial E}{\partial v'_{w_I}} &= C \frac{\sum_{j=1}^{V} v'_{w_j} \exp(v'_{w_j} v_{w_I})}{\sum_{j'=1}^{V} \exp(v'_{w_{j'}} v_{w_I})} - \sum_{c=1}^{C} v'_{w_{j_c^*}} \\
&= \sum_{j=1}^{V} \sum_{c=1}^{C} \left( \frac{\exp(v'_{w_j} v_{w_I})}{\sum_{j'=1}^{V} \exp(v'_{w_{j'}} v_{w_I})} - \mathbf{1}\{j_c = j_c^*\} \right) v'_{w_j} \\
&= \sum_{j=1}^{V} \sum_{c=1}^{C} e_{j_c} v'_{w_j}
\end{aligned}
$$

# Visualizing Training



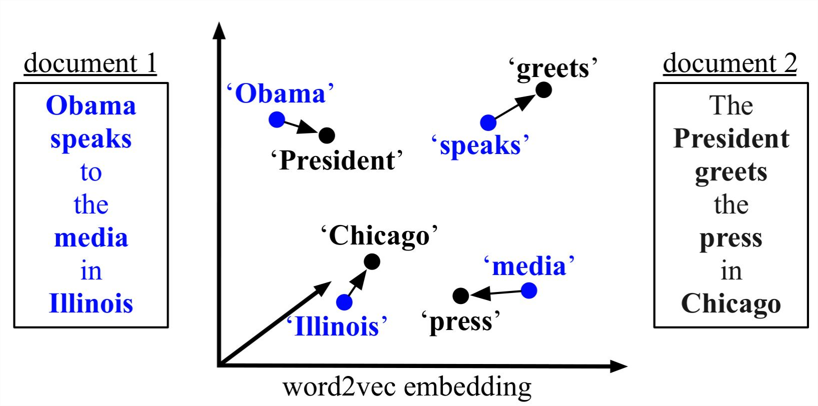
# Word Vectors for FOMC transcripts

- Transcripts for Federal Open Market Committee transcripts are available to the public at Fed's website
- Words most similar to output



# Using Word Vectors for Document Similarity

1. **Obama speaks** to the **media** in **Illinois**.
2. The **President greets** the **press** in **Chicago**.



## Word travel cost

- Key idea is to measure similarity by the 'cost' of travelling from document 1 to document 2

- The distance between two word vector $i$ and $j$ is

$$c(i,j) = ||v_i - v_j||_2$$

## Document Distance

- Let $T_{ij} \in \mathbb{R}^{V \times V}$ be a sparse flow matrix where $T_{ij} \geq 0$ denotes how much of word $i$ in $\mathbf{d}$ moves to word $j$ in $\mathbf{d}'$.
- To transform $\mathbf{d}$ entirely into $\mathbf{d}'$, we ensure that the entire outgoing flow from word $i$ equals $d_i$

$$\sum_j T_{ij} = d_i$$

- And the entire incoming flow to word $j$ equals $d'_j$

$$\sum_i T_{ij} = d'_j$$

# Transportation Problem

- The minimum cumulative cost of moving $\mathbf{d}$ to $\mathbf{d}'$ is

$$
\begin{aligned}
&\min_{\mathbf{T}\geq 0} \sum_{i,j=1}^{V} T_{ij}\, c(i,j) \\
\text{s.t.} \quad &\sum_{j=1}^{V} T_{ij} = d_i, \quad \forall i \in \{1,\cdots,V\} \\
&\sum_{i=1}^{V} T_{ij} = d'_j, \quad \forall j \in \{1,\cdots,V\}
\end{aligned}
$$

- The minimum distance is called the Word Mover's distance

# Example revisited