# Sequencing Legal DNA

## NLP for Law and Political Economy

8. Convolutions, Recurrence, and Attention

# Reminder: Assignment 2 Due by April 15th

- Options:
  - Two reading response essays
  - one code replication exercise
  - problem set solution
  - one essay and solutions to 4 problems
  - COVID-19 task submission
- Upload to assignment dropbox (see syllabus).

# Reminder: Assignment 2 Due by April 15th

- Options:
  - Two reading response essays
  - one code replication exercise
  - problem set solution
  - one essay and solutions to 4 problems
  - COVID-19 task submission
- Upload to assignment dropbox (see syllabus).
- Essays will be uploaded to the forum on April 16th/17th:
  - vote on some of your classsmates' essays, and comment on at least one, by Monday April 27th.

# Outline

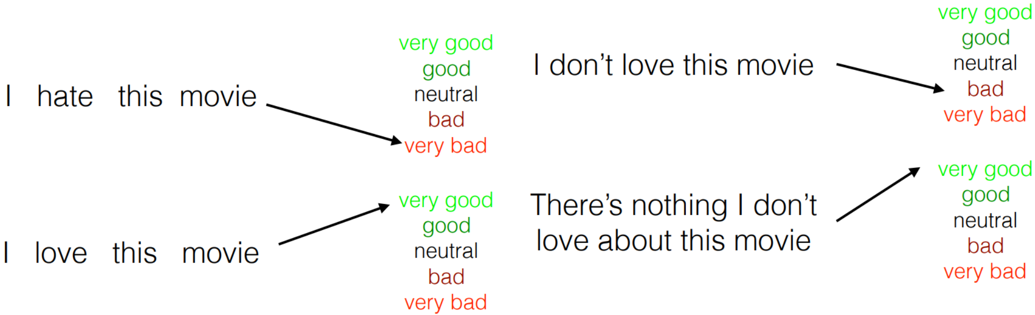
Convolutional Neural Nets

Recurrent Neural Nets

Rationales (Lei et al 2016)

Attention / Tranformers

# The Classic Sentence Classification Problem




Source: Graham Neubig slides.

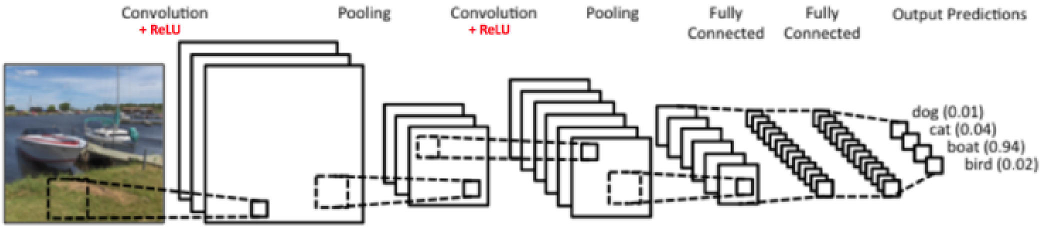
- (continuous) bag of words models (even with hidden layers) won't capture the importance of "don't love" or "nothing I don't love".

# What we have done so far: N-Grams

# What we have done so far: N-Grams

- Problems with n-grams models:
  - explosion in feature space
  - no sharing of information/weights across similar words/n-grams.

# Convolutional neural nets

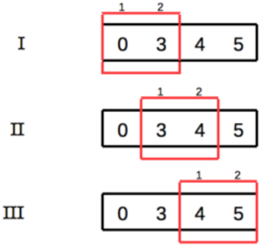



- A neural net architecture that constructs **filters** to extract **local structure** in data.
  - especially effective at image classification.

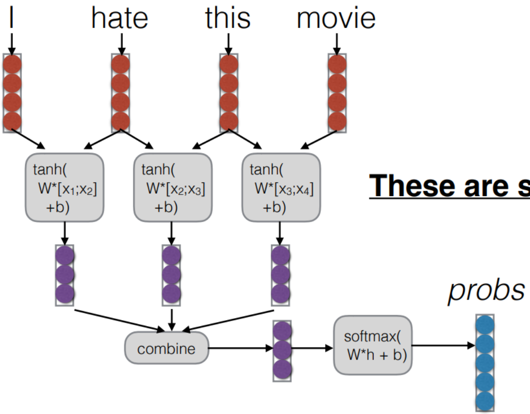
# Sequence Convolution

- CNN's generate filters, such as the $\{\phi_1, \phi_2) = \{1,2\}$ here, and slides the filters across the input sequence.

- At each window, take the dot product:
  - `[0 3] · [1 2] = 6, [3 4] · [1 2] = 11, [4 5] · [1 2] = 14`
  - output $= \{6, 11, 14\}$




- CNN learns the weights for the filter $\{\phi_1, \phi_2\}$, to try to match the output
  - complicated CNNs have more filters, and different filter window sizes.




**These are soft 2-grams!**

Graham Neubig slides.

# Convolutional Neural Nets ↔ N-gram Detectors

- More generally, can learn $n_c$-grams given a filter length $n_c$:

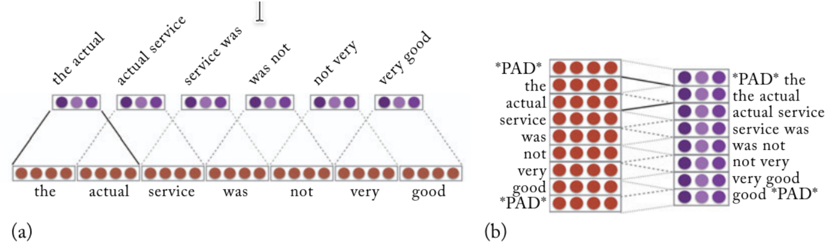



**Figure 13.1:** The inputs and outputs of a narrow and a wide convolution in the vector-concatenation and the vector-stacking notations. (a) A *narrow* convolution with a window of size $k = 2$ and 3-dimensional output ($\ell = 3$), in the vector-concatenation notation. (b) A *wide* convolution with a window of size $k = 2$, a 3-dimensional output ($\ell = 3$), in the vector-stacking notation.

- Let $w_{1:n_i}$ be a sequence of $n_i$ words in document $i$, each with a $n_E$-dimensional embedding vector $\boldsymbol{w}_l$ contained in embedding matrix $\boldsymbol{E}$.
  - for CNN's, documents have to be the same length (can add padding tokens to short documents to achieve this.)

- Let $w_{1:n_i}$ be a sequence of $n_i$ words in document $i$, each with a $n_E$-dimensional embedding vector $\boldsymbol{w}_l$ contained in embedding matrix $\boldsymbol{E}$.
  - for CNN's, documents have to be the same length (can add padding tokens to short documents to achieve this.)

- A 1D convolution of width $n_c$ moves a **sliding window** of size $n_c$ over the document:
  - Let $\boldsymbol{x}_l = [\boldsymbol{w}_l; \boldsymbol{w}_{l+1}; ...; \boldsymbol{w}_{l+n_c-1}]$ be the concatenated vectors for the $n_c$ words in the window starting at index $l \in \{1, ..., n_i - n_c + 1\}$.

- Let $w_{1:n_i}$ be a sequence of $n_i$ words in document $i$, each with a $n_E$-dimensional embedding vector $\boldsymbol{w}_l$ contained in embedding matrix $\boldsymbol{E}$.
  - for CNN's, documents have to be the same length (can add padding tokens to short documents to achieve this.)

- A 1D convolution of width $n_c$ moves a **sliding window** of size $n_c$ over the document:
  - Let $\boldsymbol{x}_l = [\boldsymbol{w}_l; \boldsymbol{w}_{l+1}; ...; \boldsymbol{w}_{l+n_c-1}]$ be the concatenated vectors for the $n_c$ words in the window starting at index $l \in \{1, ..., n_i - n_c + 1\}$.

- The same **filter** is applied at each window:
  - a filter is a dot product with weight $\phi$, usually followed by activation function $g(\cdot)$.
  - Formally:

$$h_l = g(\boldsymbol{x}_l \cdot \phi)$$

$$h_l \in \mathbb{R}, \boldsymbol{x}_l \in \mathbb{R}^{n_c n_E}, \phi \in \mathbb{R}^{n_c n_E}$$

- Let $w_{1:n_i}$ be a sequence of $n_i$ words in document $i$, each with a $n_E$-dimensional embedding vector $\boldsymbol{w}_l$ contained in embedding matrix $\boldsymbol{E}$.
    - for CNN's, documents have to be the same length (can add padding tokens to short documents to achieve this.)
- A 1D convolution of width $n_c$ moves a **sliding window** of size $n_c$ over the document:
    - Let $\boldsymbol{x}_l = [\boldsymbol{w}_l; \boldsymbol{w}_{l+1}; ...; \boldsymbol{w}_{l+n_c-1}]$ be the concatenated vectors for the $n_c$ words in the window starting at index $l \in \{1, ..., n_i - n_c + 1\}$.
- The same **filter** is applied at each window:
    - a filter is a dot product with weight $\phi$, usually followed by activation function $g(\cdot)$.
    - Formally:

$$h_l = g(\boldsymbol{x}_l \cdot \phi)$$

$$h_l \in \mathbb{R}, \boldsymbol{x}_l \in \mathbb{R}^{n_c n_E}, \phi \in \mathbb{R}^{n_c n_E}$$

- Applied at each index $l$, generating a vector $\boldsymbol{h}$ with $n_i - n_c + 1$ values.
    - standard practice is to add special padding tokens at beginning and end of document, such that $\boldsymbol{h} \in \mathbb{R}^{n_i}$, the length of the document.

# Convolutional layers have many filters

- Let $n_\phi$ be the number of filters, contained in a matrix $\phi$.
- Then we have, for each token index $l$, a $n_\phi$-vector

$$\boldsymbol{h}_l = \boldsymbol{g}(\boldsymbol{x}_l \cdot \phi)$$

$$\boldsymbol{h}_l \in \mathbb{R}^{n_\phi}, \boldsymbol{x}_l \in \mathbb{R}^{n_c n_E}, \phi \in \mathbb{R}^{n_c n_E \times n_\phi}$$

- Each dimension in $\phi$ (each filter) captures a different pattern – i.e., a predictive n-gram.

# Convolutional layers have many filters

- Let $n_\phi$ be the number of filters, contained in a matrix $\phi$.
- Then we have, for each token index $l$, a $n_\phi$-vector

$$\boldsymbol{h}_l = \boldsymbol{g}(\boldsymbol{x}_l \cdot \phi)$$

$$\boldsymbol{h}_l \in \mathbb{R}^{n_\phi}, \boldsymbol{x}_l \in \mathbb{R}^{n_c n_E}, \phi \in \mathbb{R}^{n_c n_E \times n_\phi}$$

- Each dimension in $\phi$ (each filter) captures a different pattern – i.e., a predictive n-gram.
  - filters are initialized randomly; they shift toward different patterns (specialize) based on initial positions.

# Convolutional layers have many filters

- Let $n_\phi$ be the number of filters, contained in a matrix $\phi$.
- Then we have, for each token index $l$, a $n_\phi$-vector

$$\boldsymbol{h}_l = \boldsymbol{g}(\boldsymbol{x}_l \cdot \phi)$$

$$\boldsymbol{h}_l \in \mathbb{R}^{n_\phi}, \boldsymbol{x}_l \in \mathbb{R}^{n_c n_E}, \phi \in \mathbb{R}^{n_c n_E \times n_\phi}$$

- Each dimension in $\phi$ (each filter) captures a different pattern – i.e., a predictive n-gram.
  - filters are initialized randomly; they shift toward different patterns (specialize) based on initial positions.
- Applied at each token index, the convolutional layer outputs a matrix $\boldsymbol{h} \in \mathbb{R}^{n_i \times n_\phi}$.

# Pooling

- The convolutional layer outputs a matrix $\boldsymbol{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- That matrix is then reduced to an $n_\phi$-vector $\boldsymbol{s}_i$ for input to an MLP (e.g.) for regression or classification.
- This reduction is done with a **pooling layer**.

# Pooling

- The convolutional layer outputs a matrix $\boldsymbol{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- That matrix is then reduced to an $n_\phi$-vector $\boldsymbol{s}_i$ for input to an MLP (e.g.) for regression or classification.
- This reduction is done with a **pooling layer**.
- The standard is "max pooling":
  - for each filter dimension $m \in \{1, ..., n_\phi\}$, look up the maximum value observed for that "feature" in the document:

$$\boldsymbol{s}_{[m]} = \max_{l \in \{1,...,n_i\}} \boldsymbol{h}_{[l,m]}, \forall m \in \{1, ..., n_\phi\}$$

    that is, across all token indexes $l$.

# Pooling

- The convolutional layer outputs a matrix $\boldsymbol{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- That matrix is then reduced to an $n_\phi$-vector $\boldsymbol{s}_i$ for input to an MLP (e.g.) for regression or classification.
- This reduction is done with a **pooling layer**.
- The standard is "max pooling":
  - for each filter dimension $m \in \{1, ..., n_\phi\}$, look up the maximum value observed for that "feature" in the document:

    $$\boldsymbol{s}_{[m]} = \max_{l \in \{1,...,n_i\}} \boldsymbol{h}_{[l,m]}, \forall m \in \{1, ..., n_\phi\}$$

    that is, across all token indexes $l$.

- can also take the average, but this usually gets worse performance.

# Pooling

- The convolutional layer outputs a matrix $\boldsymbol{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- That matrix is then reduced to an $n_\phi$-vector $\boldsymbol{s}_i$ for input to an MLP (e.g.) for regression or classification.
- This reduction is done with a **pooling layer**.
- The standard is "max pooling":
  - for each filter dimension $m \in \{1, ..., n_\phi\}$, look up the maximum value observed for that "feature" in the document:

    $$\boldsymbol{s}_{[m]} = \max_{l \in \{1,...,n_i\}} \boldsymbol{h}_{[l,m]}, \forall m \in \{1, ..., n_\phi\}$$

    that is, across all token indexes $l$.
- can also take the average, but this usually gets worse performance.
- Note that the vector $\boldsymbol{s}_i$ is a document embedding – documents with similar predictive information will have similar vectors $\boldsymbol{s}_i$.

## Stride

- Can skip some token indexes, e.g., apply convolution to every other index.

## Stride

- Can skip some token indexes, e.g., apply convolution to every other index.

## Multiple Channels

- visual CNNs have multiple channels for different color axes: Red, Green, Blue.
  - can do the same thing in NLP: for example, one channel can have the words, a second could have the POS tags, and a third could have the dependency tags.

## Stride

- Can skip some token indexes, e.g., apply convolution to every other index.

## Multiple Channels

- visual CNNs have multiple channels for different color axes: Red, Green, Blue.
    - can do the same thing in NLP: for example, one channel can have the words, a second could have the POS tags, and a third could have the dependency tags.

## Multiple Convolutions

- can have multiple parallel convolutional layers with different filter lengths $n_c$

## Stride

- Can skip some token indexes, e.g., apply convolution to every other index.

## Multiple Channels

- visual CNNs have multiple channels for different color axes: Red, Green, Blue.
  - can do the same thing in NLP: for example, one channel can have the words, a second could have the POS tags, and a third could have the dependency tags.

## Multiple Convolutions

- can have multiple parallel convolutional layers with different filter lengths $n_c$

## Dynamic Pooling

- Can do separate convolutions and/or pooling for different segments of the document.

## Stride

- Can skip some token indexes, e.g., apply convolution to every other index.

## Multiple Channels

- visual CNNs have multiple channels for different color axes: Red, Green, Blue.
  - can do the same thing in NLP: for example, one channel can have the words, a second could have the POS tags, and a third could have the dependency tags.

## Multiple Convolutions

- can have multiple parallel convolutional layers with different filter lengths $n_c$

## Dynamic Pooling

- Can do separate convolutions and/or pooling for different segments of the document.

## Convolutions on the Parse Tree

- can follow parse tree branches, rather than use linear word order.

Instead of pooling and outputting to MLP, can output $\boldsymbol{h}_1$ to second convolutional layer.

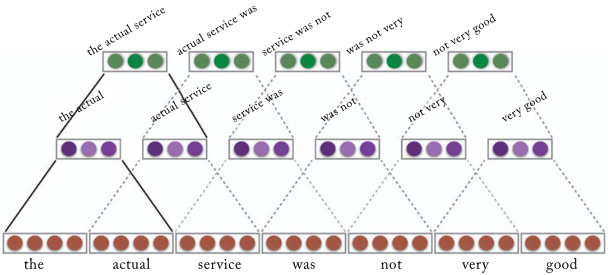



**Figure 13.3:** Two-layer hierarchical convolution with $k$=2.

Instead of pooling and outputting to MLP, can output $\boldsymbol{h}_1$ to second convolutional layer.

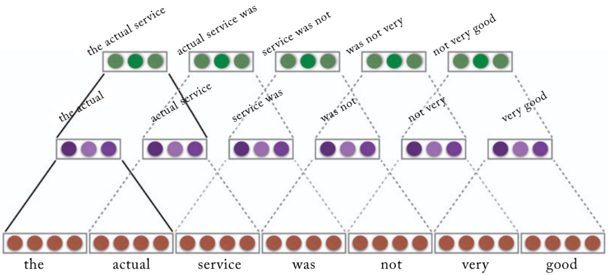



**Figure 13.3:** Two-layer hierarchical convolution with $k$=2.

- Let CONV$(x; c, m)$ be a 1D convolutional layer with $m$ filters of window size $c$, applied to input (sequence of vectors) $x$.
- Can define arbitrarily deep stack of $L$ convolutional layers:

$$
\begin{aligned}
\boldsymbol{h}_1 &= \text{CONV}(\boldsymbol{w}_{1:n_i}; c_1, m_1) \\
\boldsymbol{h}_2 &= \text{CONV}(\boldsymbol{h}_1; c_2, m_2) \\
&\ \ \ldots \\
\boldsymbol{h}_L &= \text{CONV}(\boldsymbol{h}_{L-1}; c_{L-1}, m_{L-1} \\
\boldsymbol{s} &= \text{POOL}(\boldsymbol{h}_L; m_{L-1})
\end{aligned}
$$

# Johnson and Zhang 2015

- Use CNN for classifying documents by topic and sentiment.
- Interesting part is showing what CNN used for the prediction.

# Johnson and Zhang 2015

- Use CNN for classifying documents by topic and sentiment.
- Interesting part is showing what CNN used for the prediction.

- Linear model (SVM) relied on single words and just a few n-grams:
  - poor, useless, returned, not worth, return, worse, disappointed, terrible, worst, horrible
  - great, excellent, perfect, love, easy, amazing, awesome, no problems, perfectly, beat

# Johnson and Zhang 2015

- Use CNN for classifying documents by topic and sentiment.
- Interesting part is showing what CNN used for the prediction.

- Linear model (SVM) relied on single words and just a few n-grams:
  - poor, useless, returned, not worth, return, worse, disappointed, terrible, worst, horrible
  - great, excellent, perfect, love, easy, amazing, awesome, no problems, perfectly, beat
- CNN recovers longer, more interesting phrases:

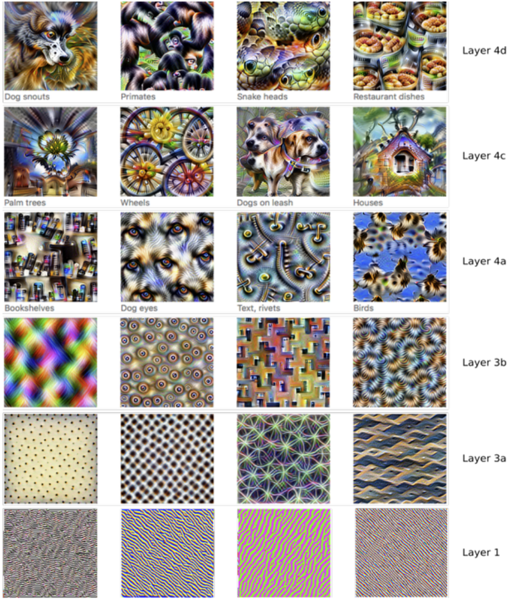
| | |
|---|---|
| N1 | completely useless ., return policy . |
| N2 | it won't even, but doesn't work |
| N3 | product is defective, very disappointing ! |
| N4 | is totally unacceptable, is so bad |
| N5 | was very poor, it has failed |
| P1 | works perfectly !, love this product |
| P2 | very pleased !, super easy to, i am pleased |
| P3 | 'm so happy, it works perfect, is awesome ! |
| P4 | highly recommend it, highly recommended ! |
| P5 | am extremely satisfied, is super fast |

Table 5: Examples of predictive text regions in the training set.

| |
|---|
| were unacceptably bad, is abysmally bad, were universally poor, was hugely disappointed, was enormously disappointed, is monumentally frustrating, are endlessly frustrating |
| best concept ever, best ideas ever, best hub ever, am wholly satisfied, am entirely satisfied, am incredicbly satisfied, 'm overall impressed, am awfully pleased, am exceptionally pleased, 'm entirely happy, are acoustically good, is blindingly fast, |

Table 6: Examples of text regions that contribute to prediction. They are from the *test set*, and they did *not* appear in the training set, either entirely or partially as bi-grams.

# How CNNs see the world



# How CNNs see the world

# Alternative: Embedded Hashed N-Grams

# Alternative: Embedded Hashed N-Grams

Goldberg (2017) proposes the hashing vectorizer as an efficient alternative to CNN's:

- Allocate $n_w \approx 1$ million rows to an embedding matrix $\boldsymbol{E}$
- Assign n-grams to embedding indexes with hashing function
- train MLP on top of embedding layer.

# Alternative: Embedded Hashed N-Grams

Goldberg (2017) proposes the hashing vectorizer as an efficient alternative to CNN's:

- Allocate $n_w \approx 1$ million rows to an embedding matrix $\boldsymbol{E}$
- Assign n-grams to embedding indexes with hashing function
- train MLP on top of embedding layer.
- Captures the local predictive power of n-grams without building vocabulary or costly training of CNN.

# Outline

Convolutional Neural Nets

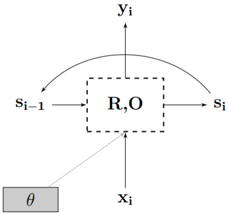
Recurrent Neural Nets

Rationales (Lei et al 2016)

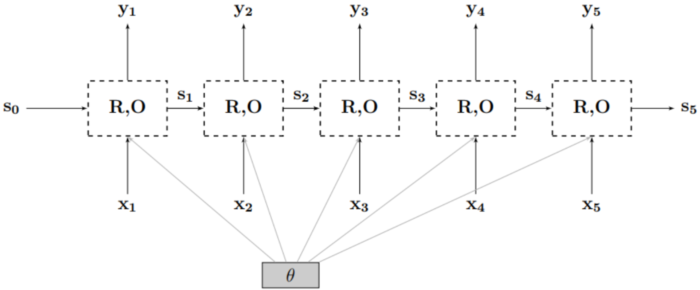
Attention / Tranformers

# From vectors to sequences

- The models we have looked at so far took inputs of fixed dimensions across rows.
- RNNs can work with sequences of arbitrary length.
  - they are useful for language tasks such as translation.



$\mathbf{y_i}$
$\mathbf{s_{i-1}}$
R,O
$\mathbf{s_i}$
$\theta$
$\mathbf{x_i}$

$\mathbf{y_1}$ $\mathbf{y_2}$ $\mathbf{y_3}$ $\mathbf{y_4}$ $\mathbf{y_5}$
$\mathbf{s_0}$ R,O $\mathbf{s_1}$ R,O $\mathbf{s_2}$ R,O $\mathbf{s_3}$ R,O $\mathbf{s_4}$ R,O $\mathbf{s_5}$
$\mathbf{x_1}$ $\mathbf{x_2}$ $\mathbf{x_3}$ $\mathbf{x_4}$ $\mathbf{x_5}$
$\theta$

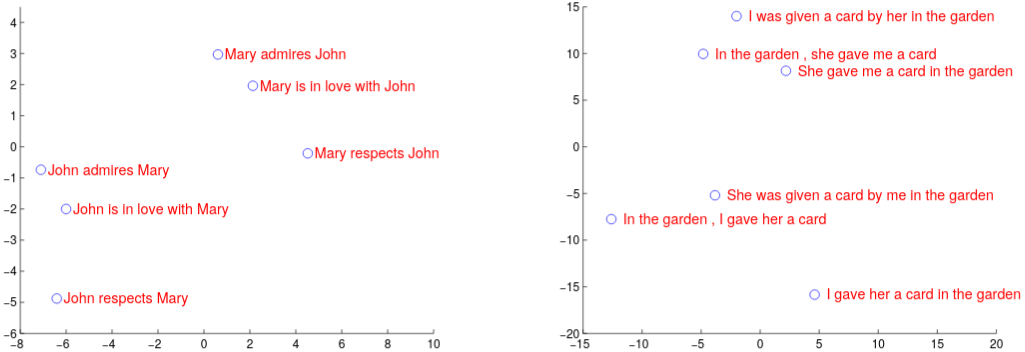
# RNN Implementation

- LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) refer to popular RNN implementations for natural language analysis.
  - Can predict the next word in a sequence, for example.
  - they don't apply naturally to document-level outcomes – e.g. predicting the citations for a document.

# RNN Implementation

- LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) refer to popular RNN implementations for natural language analysis.
  - Can predict the next word in a sequence, for example.
  - they don't apply naturally to document-level outcomes – e.g. predicting the citations for a document.
- Can perform sequence to sequence predictions – used for translations.

# Geometry of Embedded Sequences




Sutskever, Vinyals, and Le, "Sequence to sequence learning with neural networks."

# RNN's for predicting partisanship

Iyyer, Enns, Boyd-Graber, and Resnik (2014)

| n | Most conservative n-grams | Most liberal n-grams |
|---|---|---|
| 1 | Salt, Mexico, housework, speculated, consensus, lawyer, pharmaceuticals, ruthless, deadly, Clinton, redistribution | rich, antipsychotic, malaria, biodiversity, richest, gene, pesticides, desertification, Net, wealthiest, labor, fertilizer, nuclear, HIV |
| 3 | prize individual liberty, original liberal idiots, stock market crash, God gives freedom, federal government interference, federal oppression nullification, respect individual liberty, Tea Party patriots, radical Sunni Islamists, Obama stimulus programs | rich and poor,"corporate greed", super rich pay, carrying the rich, corporate interest groups, young women workers, the very rich, for the rich, by the rich, soaking the rich, getting rich often, great and rich, the working poor, corporate income tax, the poor migrants |
| 5 | spending on popular government programs, bailouts and unfunded government promises, North America from external threats, government regulations place on businesses, strong Church of Christ convictions, radical Islamism and other threats | the rich are really rich, effective forms of worker participation, the pensions of the poor, tax cuts for the rich, the ecological services of biodiversity, poor children and pregnant women, vacation time for overtime pay |
| 7 | government intervention helped make the Depression Great, by God in His image and likeness, producing wealth instead of stunting capital creation, the traditional American values of limited government, trillions of dollars to overseas oil producers, its troubled assets to federal sugar daddies, Obama and his party as racialist fanatics | African Americans and other disproportionately poor groups; the growing gap between rich and poor; the Bush tax cuts for the rich; public outrage at corporate and societal greed; sexually transmitted diseases , most notably AIDS; organize unions or fight for better conditions, the biggest hope for health care reform |

Table 2: Highest probability n-grams for conservative and liberal ideologies, as predicted by the **RNN2-(W2V)** model.

# Recurrent Autoencoders

For text (or other sequential) data, you can use a recurrent autoencoder:

1. encode a sequence using a sequence-to-vector RNN, to compress the sequence to a single vector
2. decode the vector using a vector-to-sequence RNN to do the opposite.

# Recurrent Autoencoders

For text (or other sequential) data, you can use a recurrent autoencoder:

1. encode a sequence using a sequence-to-vector RNN, to compress the sequence to a single vector
2. decode the vector using a vector-to-sequence RNN to do the opposite.

- Reconstructions are usually quite bad.

# Outline

Convolutional Neural Nets

Recurrent Neural Nets

Rationales (Lei et al 2016)

Attention / Tranformers

# Rationalizing Neural Predictions

- Lei, Barzilay, and Jaakola (2016) provide a text-based prediction model which learns to extract snippets of text to serve as justifications – rationales.

# Rationalizing Neural Predictions

- Lei, Barzilay, and Jaakola (2016) provide a text-based prediction model which learns to extract snippets of text to serve as justifications – rationales.

*Review*

the beer was n't what i expected, and i'm not sure it's "true to style", but i thought it was delicious. **a very pleasant ruby red-amber color** with a relatively brilliant finish, but a limited amount of carbonation, from the look of it. aroma is what i think an amber ale should be - a nice blend of caramel and happiness bound together.

*Ratings* *Look:* 5 stars *Smell:* 4 stars

**Figure 1:** An example of a review with ranking in two categories. The rationale for Look prediction is shown in bold.

# Rationalizing Neural Predictions

- Lei, Barzilay, and Jaakola (2016) provide a text-based prediction model which learns to extract snippets of text to serve as justifications – rationales.
- Key idea: find minimal span(s) of text that can (by themselves) explain the prediction
- Generator (x) outputs a probability distribution of each word being the rationale
- Encoder (x) predicts the output using the snippet of text x
- Regularization to support contiguous and minimal spans

# Outline

Convolutional Neural Nets

Recurrent Neural Nets

Rationales (Lei et al 2016)

Attention / Tranformers