

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 classmates' essays, and comment on at least one, by Monday April 27th.

Sequence Data

- ▶ This lecture focuses on deep learning approaches to NLP that treat language as a sequence of words.
- ▶ “Traditional” architectures:
 - ▶ Convolutional neural nets (CNNs)
 - ▶ Recurrent Neural Nets (RNNs)

Sequence Data

- ▶ This lecture focuses on deep learning approaches to NLP that treat language as a sequence of words.
- ▶ “Traditional” architectures:
 - ▶ Convolutional neural nets (CNNs)
 - ▶ Recurrent Neural Nets (RNNs)
- ▶ As of April 2020, CNNs and RNNs (as currently implemented) consistently get worse performance than attentional neural nets (transformers).

Outline

Convolutional Neural Nets

Recurrent Neural Nets

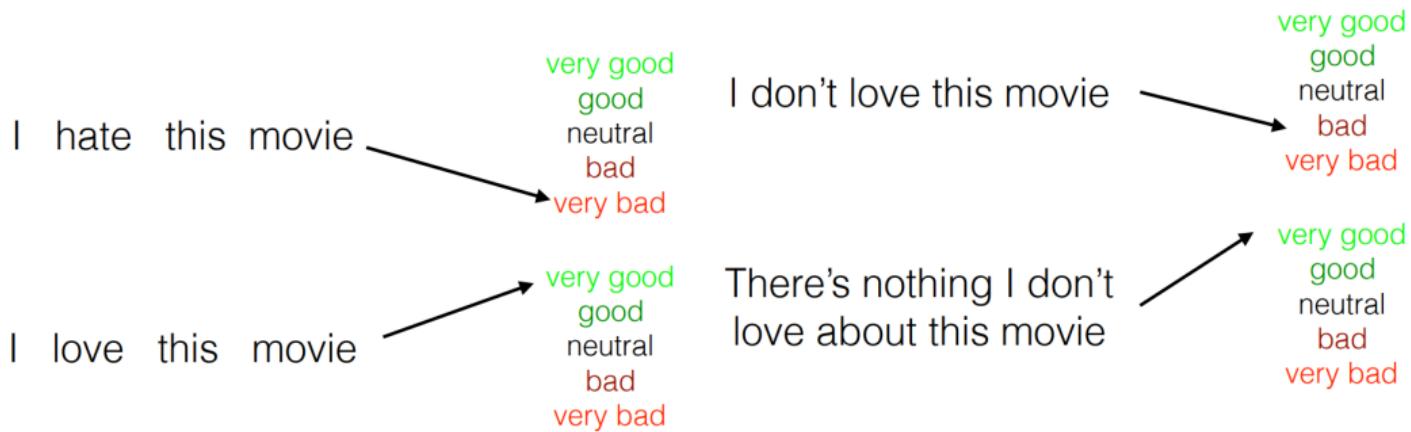
Rationales

Lei et al (2016)

Sanford, Roberts, and Li (2019)

Attention / Tranformers

The Classic Sentence Classification Problem



Source: Graham Neubig slides.

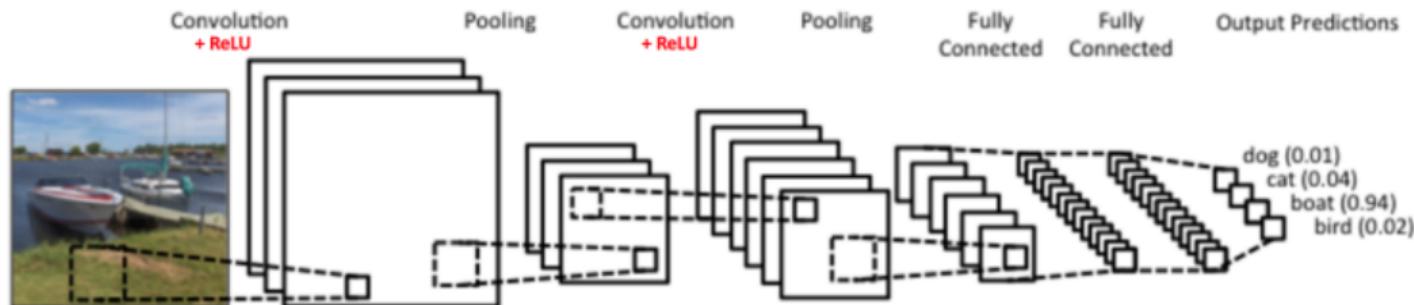
- ▶ (continuous) bag of words models (even with hidden layers) won't capture the importance of "don't love" or "nothing I don't love".

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

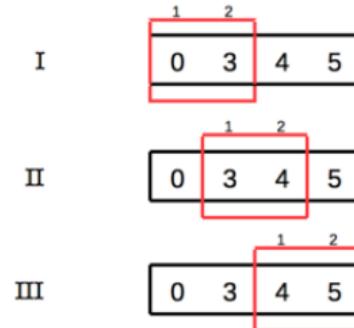


- ▶ A neural net architecture that constructs **filters** to extract **local structure** in data.
 - ▶ especially effective at image classification.

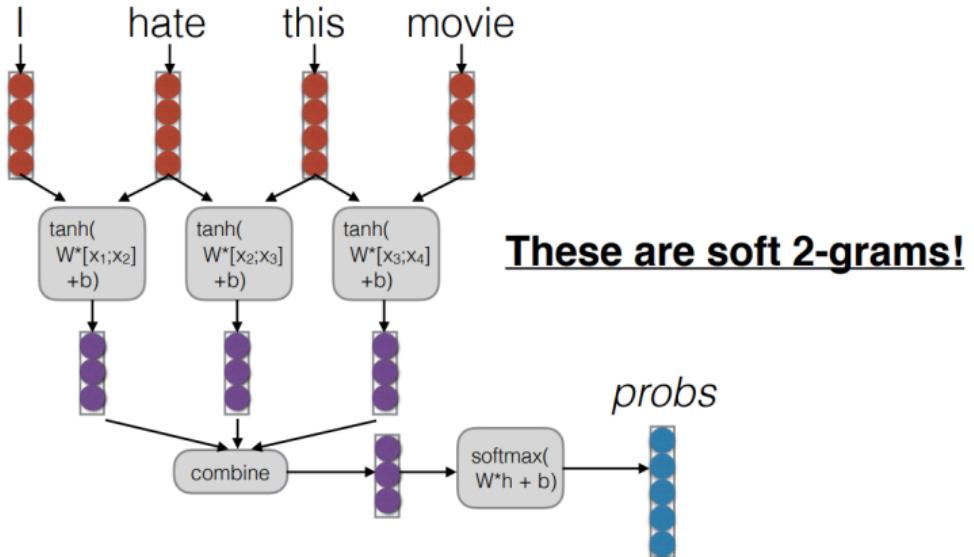
Sequence Convolution

- ▶ 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:

- ▶ $[0 \ 3] \cdot [1 \ 2] = 6, [3 \ 4] \cdot [1 \ 2] = 11, [4 \ 5] \cdot [1 \ 2] = 14$
- ▶ 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.



Graham Neubig slides.

Convolutional Neural Nets \leftrightarrow N-gram Detectors

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

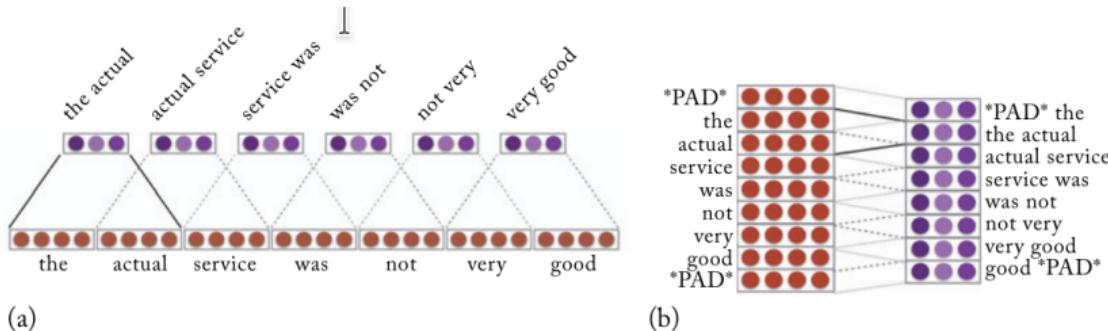


Figure 13.1: The inputs and outputs of a narrow and a wide convolution in the vector-concatenation and the vector-stacking notations. (a) A *narrow* convolution with a window of size $k = 2$ and 3-dimensional output ($\ell = 3$), in the vector-concatenation notation. (b) A *wide* convolution with a window 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 \mathbf{w}_i contained in embedding matrix \mathbf{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 \mathbf{w}_i contained in embedding matrix \mathbf{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 $x_l = [\mathbf{w}_l; \mathbf{w}_{l+1}; \dots; \mathbf{w}_{l+n_c-1}]$ be the concatenated vectors for the n_c words in the window starting at index $l \in \{1, \dots, n_i - n_c + 1\}$.

- ▶ Let $w_{1:n_i}$ be a sequence of n_i words in document i , each with a n_E -dimensional embedding vector \mathbf{w}_i contained in embedding matrix \mathbf{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 $\mathbf{x}_l = [\mathbf{w}_l; \mathbf{w}_{l+1}; \dots; \mathbf{w}_{l+n_c-1}]$ be the concatenated vectors for the n_c words in the window starting at index $l \in \{1, \dots, n_i - n_c + 1\}$.
- ▶ The same **filter** is applied at each window:
 - ▶ a filter is a dot product with weight ϕ , usually followed by activation function $g(\cdot)$.
 - ▶ Formally:

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

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

- ▶ Let $w_{1:n_i}$ be a sequence of n_i words in document i , each with a n_E -dimensional embedding vector \mathbf{w}_i contained in embedding matrix \mathbf{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 $\mathbf{x}_l = [\mathbf{w}_l; \mathbf{w}_{l+1}; \dots; \mathbf{w}_{l+n_c-1}]$ be the concatenated vectors for the n_c words in the window starting at index $l \in \{1, \dots, n_i - n_c + 1\}$.
- ▶ The same **filter** is applied at each window:
 - ▶ a filter is a dot product with weight ϕ , usually followed by activation function $g(\cdot)$.
 - ▶ Formally:
$$h_l = g(\mathbf{x}_l \cdot \phi)$$

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

Convolutional layers have many filters

- ▶ Let n_ϕ be the number of filters, contained in a matrix ϕ .
- ▶ Then we have, for each token index I , a n_ϕ -vector

$$\mathbf{h}_I = \mathbf{g}(\mathbf{x}_I \cdot \phi)$$

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

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

Convolutional layers have many filters

- ▶ Let n_ϕ be the number of filters, contained in a matrix ϕ .
- ▶ Then we have, for each token index I , a n_ϕ -vector

$$\mathbf{h}_I = \mathbf{g}(\mathbf{x}_I \cdot \phi)$$

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

- ▶ Each dimension in ϕ (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_ϕ be the number of filters, contained in a matrix ϕ .
- ▶ Then we have, for each token index I , a n_ϕ -vector

$$\mathbf{h}_I = \mathbf{g}(\mathbf{x}_I \cdot \phi)$$

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

- ▶ Each dimension in ϕ (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 $\mathbf{h} \in \mathbb{R}^{n_i \times n_\phi}$.

Pooling

- ▶ The convolutional layer outputs a matrix $\mathbf{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- ▶ That matrix is then reduced to an n_ϕ -vector \mathbf{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 $\mathbf{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- ▶ That matrix is then reduced to an n_ϕ -vector \mathbf{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 $m \in \{1, \dots, n_\phi\}$, look up the maximum value observed for that “feature” in the document:

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

that is, across all token indexes l .

Pooling

- ▶ The convolutional layer outputs a matrix $\mathbf{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- ▶ That matrix is then reduced to an n_ϕ -vector \mathbf{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 $m \in \{1, \dots, n_\phi\}$, look up the maximum value observed for that “feature” in the document:

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

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 $\mathbf{h} \in \mathbb{R}^{n_i \times n_\phi}$.
- ▶ That matrix is then reduced to an n_ϕ -vector \mathbf{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 $m \in \{1, \dots, n_\phi\}$, look up the maximum value observed for that “feature” in the document:

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

that is, across all token indexes l .

- ▶ can also take the average, but this usually gets worse performance.
- ▶ Note that the vector \mathbf{s}_i is a document embedding – documents with similar predictive information will have similar vectors \mathbf{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 h_1 to second convolutional layer.

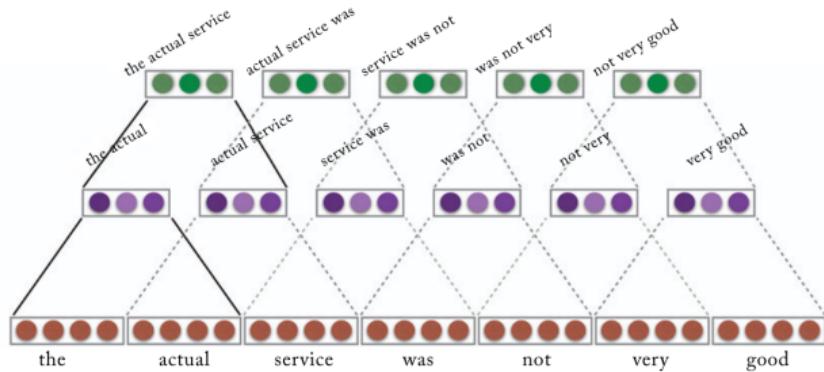


Figure 13.3: Two-layer hierarchical convolution with $k=2$.

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

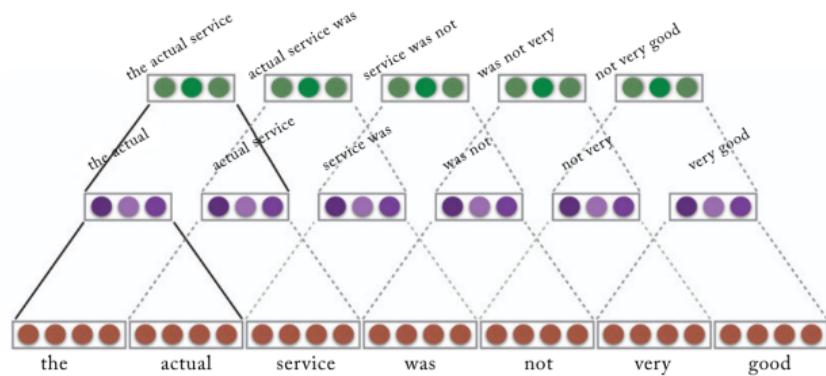


Figure 13.3: Two-layer hierarchical convolution with $k=2$.

► Let $\text{CONV}(x; c, m)$ 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:

$$\mathbf{h}_1 = \text{CONV}(\mathbf{w}_{1:n_i}; c_1, m_1)$$

$$\mathbf{h}_2 = \text{CONV}(\mathbf{h}_1; c_2, m_2)$$

...

$$\mathbf{h}_L = \text{CONV}(\mathbf{h}_{L-1}; c_{L-1}, m_{L-1})$$

$$\mathbf{s} = \text{POOL}(\mathbf{h}_L; m_{L-1})$$

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

- ▶ 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 .	were unacceptably bad, is abysmally bad, were universally poor, was hugely disappointed, was enormously disappointed, is monumentally frustrating, are endlessly frustrating
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	best concept ever, best ideas ever, best hub ever,
P2	very pleased !, super easy to, i am pleased	am wholly satisfied, am entirely satisfied, am incredibly satisfied, 'm overall impressed, am awfully pleased, am exceptionally pleased, 'm entirely happy,
P3	'm so happy, it works perfect, is awesome !	are acoustically good, is blindingly fast,
P4	highly recommend it, highly recommended !	
P5	am extremely satisfied, is super fast	

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

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.

WaveNet

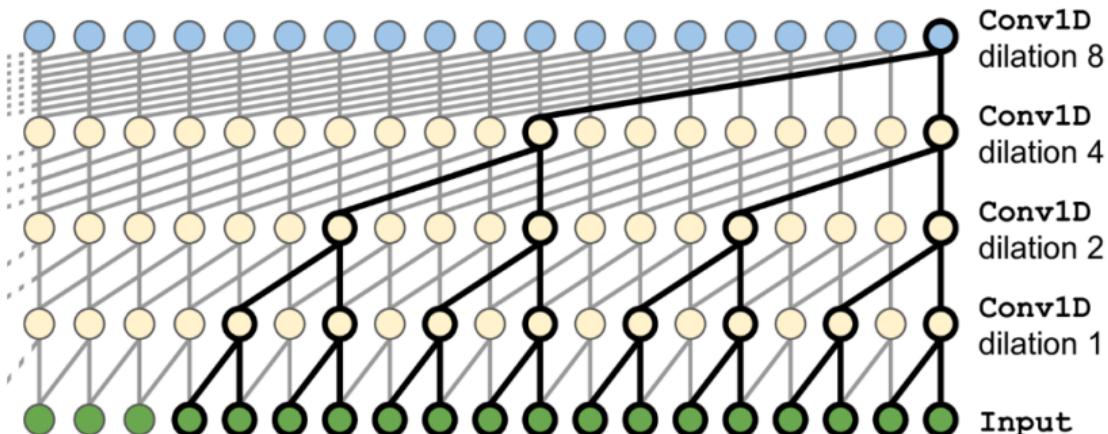


Figure 15-11. WaveNet architecture

- ▶ WaveNet is a 1-D CNN:
 - ▶ stacked convolution layers, doubling the dilation rate at every layer.

WaveNet

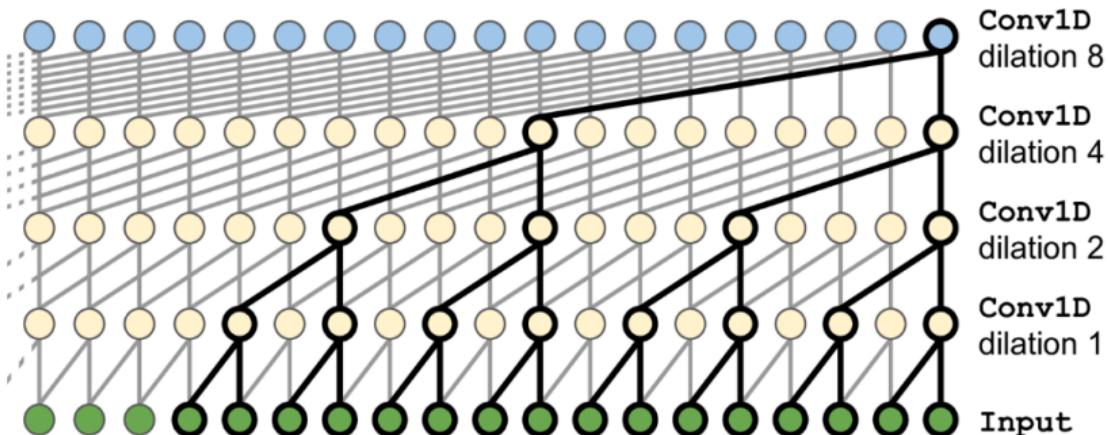
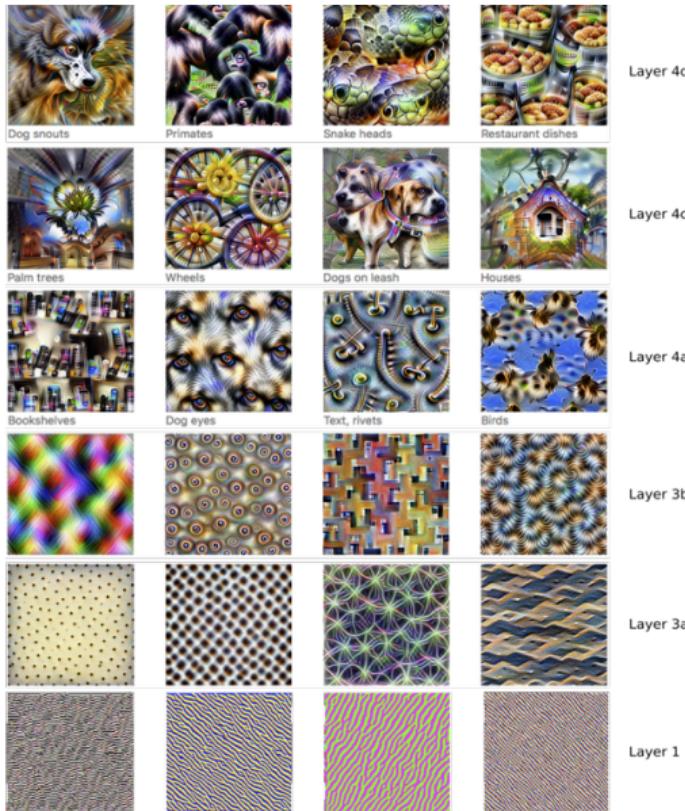


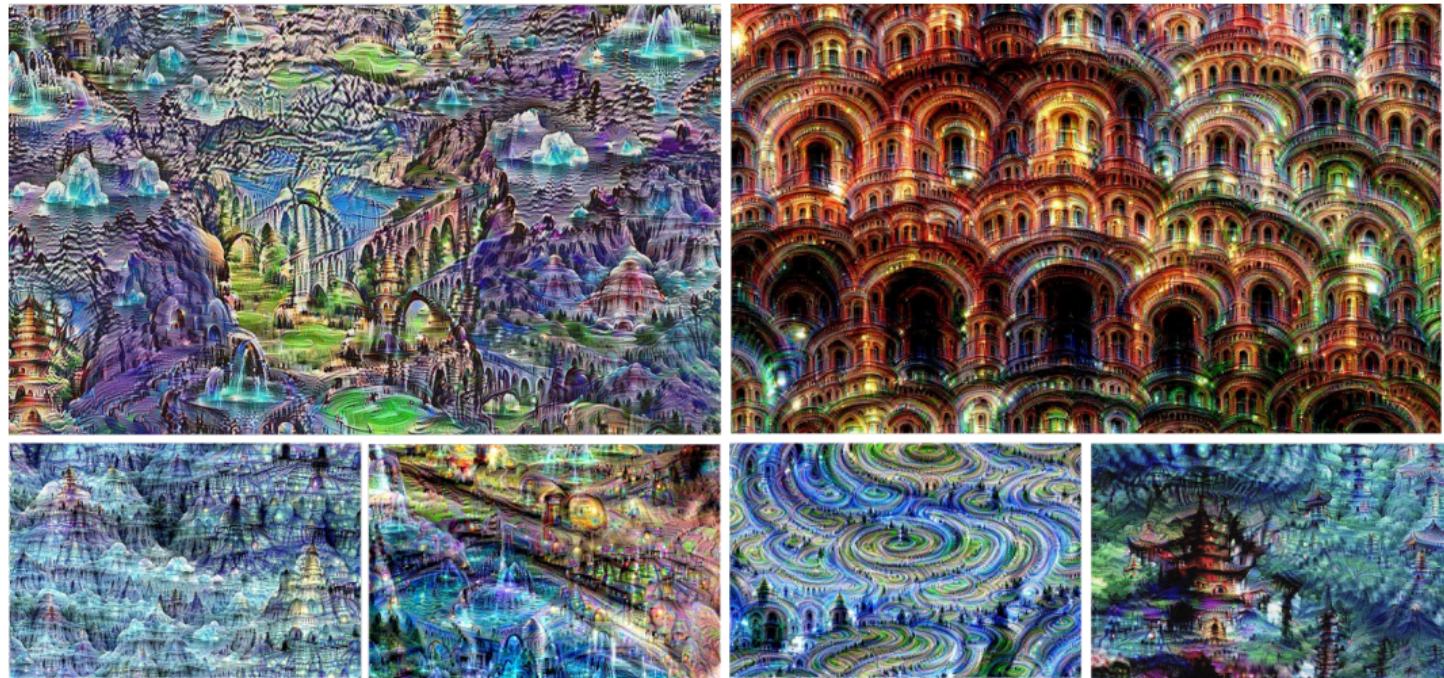
Figure 15-11. WaveNet architecture

- ▶ WaveNet is a 1-D CNN:
 - ▶ stacked convolution layers, doubling the dilation rate at every layer.
 - ▶ gets state-of-the-art performance on audio processing tasks, e.g. text to speech generation.
- ▶ For a similar NLP application, see Kalchenbrenner et al (2016).

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
(keras.preprocessing.text.hashing_trick) as an efficient alternative to CNN's:

- ▶ Allocate $n_w \approx 1$ million rows to an embedding matrix 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

(`keras.preprocessing.text.hashing_trick`) as an efficient alternative to CNN's:

- ▶ Allocate $n_w \approx 1$ million rows to an embedding matrix 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)

Sanford, Roberts, and Li (2019)

Attention / Tranformers

From vectors to sequences

- ▶ The models we have looked at so far took inputs of fixed dimensions across rows.

From vectors to sequences

- ▶ The models we have looked at so far took inputs of fixed dimensions across rows.
- ▶ RNNs work with sequences of arbitrary length:
 - ▶ can encode sequences into vectors.
 - ▶ can decode vectors in sequences.
 - ▶ therefore especially useful for language tasks such as translation.

From vectors to sequences

- ▶ The models we have looked at so far took inputs of fixed dimensions across rows.
- ▶ RNNs work with sequences of arbitrary length:
 - ▶ can encode sequences into vectors.
 - ▶ can decode vectors in sequences.
 - ▶ therefore especially useful for language tasks such as translation.
- ▶ state-of-the-art performance on some machine learning tasks, especially with short documents.

$$\mathbf{y}(\mathbf{x}_{1:n}) = \text{RNN}(\mathbf{x}_{1:n})$$

- ▶ $\mathbf{y}(\mathbf{x}_{1:n}) \in \mathbb{R}^y$, a label to predict, e.g. the topic of a news article.
- ▶ $\mathbf{x}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ (embeddings of words $w_{1:n}$), $\mathbf{x}_l \in \mathbb{R}^{n_E}$
- ▶ a state vector $s_t \in \mathbb{R}^{n_s}$ at initial state s_0 (usually zeros by convention)

$$\mathbf{y}(\mathbf{x}_{1:n}) = \text{RNN}(\mathbf{x}_{1:n})$$

- ▶ $\mathbf{y}(\mathbf{x}_{1:n}) \in \mathbb{R}^y$, a label to predict, e.g. the topic of a news article.
- ▶ $\mathbf{x}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ (embeddings of words $w_{1:n}$), $\mathbf{x}_l \in \mathbb{R}^{n_E}$
- ▶ a state vector $s_t \in \mathbb{R}^{n_s}$ at initial state s_0 (usually zeros by convention)
- ▶ At each step t :
 - ▶ a recursion function $R(s_{t-1}, x_t)$ computes the state vector s_t

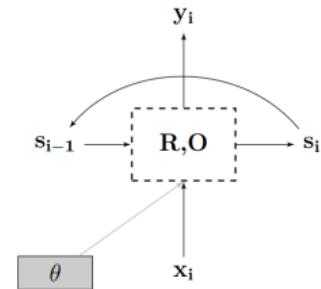
$$\mathbf{y}(\mathbf{x}_{1:n}) = \text{RNN}(\mathbf{x}_{1:n})$$

- ▶ $\mathbf{y}(\mathbf{x}_{1:n}) \in \mathbb{R}^y$, a label to predict, e.g. the topic of a news article.
- ▶ $\mathbf{x}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ (embeddings of words $w_{1:n}$), $\mathbf{x}_l \in \mathbb{R}^{n_E}$
- ▶ a state vector $s_t \in \mathbb{R}^{n_s}$ at initial state s_0 (usually zeros by convention)
- ▶ At each step t :
 - ▶ a recursion function $R(s_{t-1}, x_t)$ computes the state vector s_t
 - ▶ An output function $O(s_t)$ computes an output vector y_t (to be compared to the output data).

RNN Schema

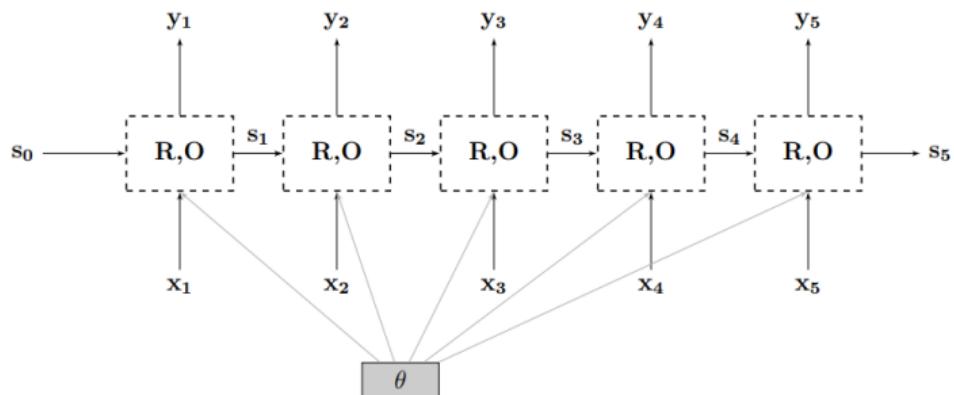
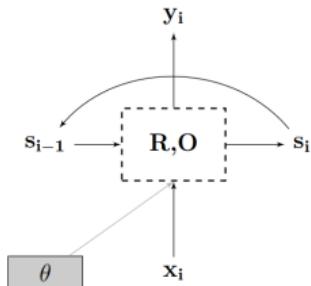
$$\hat{y}_t = O(s_t)$$

$$s_t = R(s_{t-1}, x_t)$$



RNN Schema

$$\hat{y}_t = O(s_t)$$
$$s_t = R(s_{t-1}, x_t)$$



s_t and y_t encode the entire input sequence

$$s_4 = R(s_3, x_4)$$

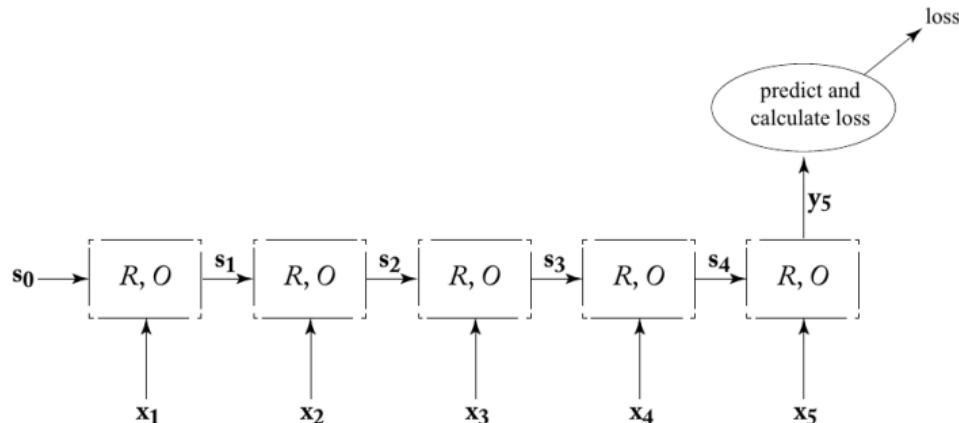
$$= R(\overbrace{R(s_2, x_3)}^{s_3}, x_4)$$

$$= R(R(\overbrace{R(s_1, x_2)}^{s_2}, x_3), x_4)$$

$$= R(R(R(\overbrace{R(s_0, x_1)}^{s_1}, x_2), x_3), x_4)$$

RNN Usage

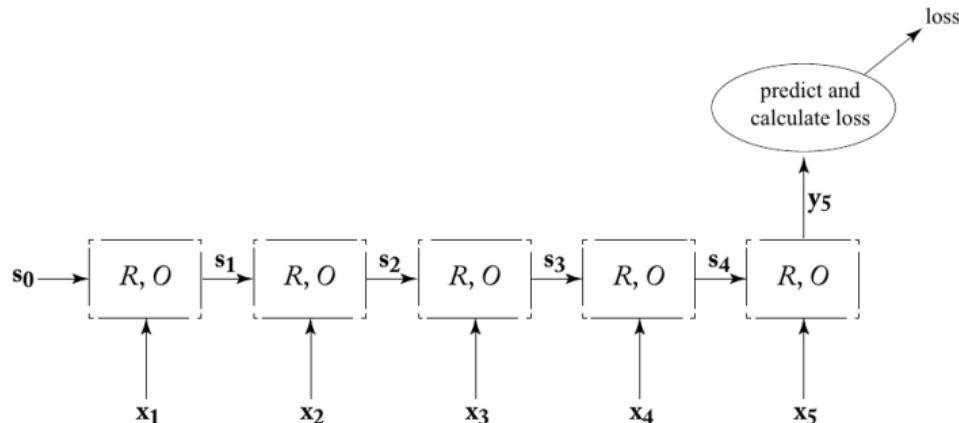
- ▶ “**Acceptor**”: predict and calculate loss only for the last item in the sequence:



- ▶ “**Transducer**”: compute the average loss for the output at every stage.

RNN Usage

- ▶ “**Acceptor**”: predict and calculate loss only for the last item in the sequence:



- ▶ “**Transducer**”: compute the average loss for the output at every stage.
- ▶ “**Encoder**”: produces a vector for a sequence for input into a larger network/task.
 - ▶ e.g., translation into a sequence in another language.

Simple RNN

$$\hat{\mathbf{y}}_t = O(\mathbf{s}_t) = \mathbf{s}_t$$

$$\mathbf{s}_t = R(\mathbf{s}_{t-1}, \mathbf{x}_t) = \mathbf{g}(\mathbf{s}_{t-1} \cdot \omega_s + \mathbf{x}_t \cdot \omega_x)$$

$$\mathbf{s}_t, \mathbf{y}_t \in \mathbb{R}^{n_y}, \mathbf{x}_t \in \mathbb{R}^{n_x}, \omega_s \in \mathbb{R}^{n_y \times n_y}, \omega_x \in \mathbb{R}^{n_x \times n_y}$$

- ▶ linear combination of state \mathbf{s}_{t-1} and input \mathbf{x}_t are passed through $g(\cdot)$ (typically ReLU or tanh) to get next state.
- ▶ output is just the state.

Simple RNN

$$\hat{\mathbf{y}}_t = O(\mathbf{s}_t) = \mathbf{s}_t$$

$$\mathbf{s}_t = R(\mathbf{s}_{t-1}, \mathbf{x}_t) = \mathbf{g}(\mathbf{s}_{t-1} \cdot \omega_s + \mathbf{x}_t \cdot \omega_x)$$

$$\mathbf{s}_t, \mathbf{y}_t \in \mathbb{R}^{n_y}, \mathbf{x}_t \in \mathbb{R}^{n_x}, \omega_s \in \mathbb{R}^{n_y \times n_y}, \omega_x \in \mathbb{R}^{n_x \times n_y}$$

- ▶ linear combination of state \mathbf{s}_{t-1} and input \mathbf{x}_t are passed through $g(\cdot)$ (typically ReLU or tanh) to get next state.
- ▶ output is just the state.
- ▶ Simple RNN works well for simple NLP tasks
 - ▶ but anything that uses long-range dependencies will not work well because the state forgets information quickly.

Bidirectional RNNs

- ▶ train additional parallel RNN starting at end of document and moving backward.
- ▶ predicted output is function of both forward and backward states: $y_t = O(s_t^F, s_t^B)$

Bidirectional RNNs

- ▶ train additional parallel RNN starting at end of document and moving backward.
- ▶ predicted output is function of both forward and backward states: $y_t = O(s_t^F, s_t^B)$

Stacked RNNs

- ▶ RNN's can be stacked, with multiple layers at each period.

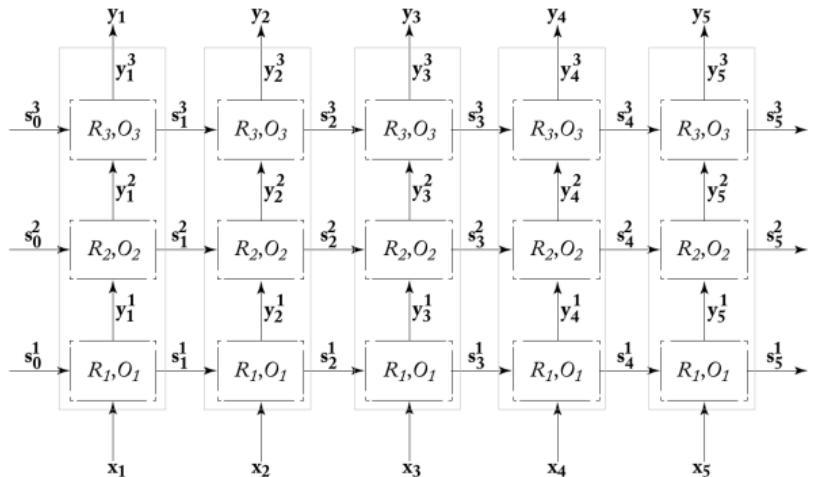


Figure 14.7: A three-layer (“deep”) RNN architecture.

Gated Architectures – LSTM (Long Short-Term Memory)

- ▶ Inputs $x_t \in \mathbb{R}^{n_x}$
- ▶ State vector has two memory components:
 - ▶ long term $s_t^L \in \mathbb{R}^{n_y}$ and short term $s_t^S \in \mathbb{R}^{n_y}$

Gated Architectures – LSTM (Long Short-Term Memory)

- ▶ Inputs $\mathbf{x}_t \in \mathbb{R}^{n_x}$
- ▶ State vector has two memory components:
 - ▶ long term $\mathbf{s}_t^L \in \mathbb{R}^{n_y}$ and short term $\mathbf{s}_t^S \in \mathbb{R}^{n_y}$
- ▶ Output:

$$\mathbf{y}_t = \mathbf{s}_t^S = g_O(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes \tanh(\mathbf{s}_t^L)$$

- ▶ output gate: $g_O = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^O + \mathbf{s}_{t-1}^S \cdot \omega_s^O)$
 - ▶ \otimes = element-wise multiplication

Gated Architectures – LSTM (Long Short-Term Memory)

- ▶ Inputs $\mathbf{x}_t \in \mathbb{R}^{n_x}$
- ▶ State vector has two memory components:
 - ▶ long term $\mathbf{s}_t^L \in \mathbb{R}^{n_y}$ and short term $\mathbf{s}_t^S \in \mathbb{R}^{n_y}$
- ▶ Output:

$$\mathbf{y}_t = \mathbf{s}_t^S = g_O(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes \tanh(\mathbf{s}_t^L)$$

- ▶ output gate: $g_O = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^O + \mathbf{s}_{t-1}^S \cdot \omega_s^O)$
 - ▶ \otimes = element-wise multiplication
- ▶ Long-Term Memory Update:

$$\mathbf{s}_t^L = g_F(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes \mathbf{s}_{t-1}^L + g_I(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes g_U(\mathbf{x}_t, \mathbf{s}_{t-1}^S)$$

- ▶ forget gate: $g_F = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^F + \mathbf{s}_{t-1}^S \cdot \omega_s^F)$
- ▶ input gate: $g_I = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^I + \mathbf{s}_{t-1}^S \cdot \omega_s^I)$
- ▶ state updater: $g_U = \tanh(\mathbf{x}_t \cdot \omega_x^U + \mathbf{s}_{t-1}^S \cdot \omega_s^U)$

Gated Architectures – LSTM (Long Short-Term Memory)

- ▶ Inputs $\mathbf{x}_t \in \mathbb{R}^{n_x}$
- ▶ State vector has two memory components:
 - ▶ long term $\mathbf{s}_t^L \in \mathbb{R}^{n_y}$ and short term $\mathbf{s}_t^S \in \mathbb{R}^{n_y}$
- ▶ Output:

$$\mathbf{y}_t = \mathbf{s}_t^S = g_O(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes \tanh(\mathbf{s}_t^L)$$

- ▶ output gate: $g_O = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^O + \mathbf{s}_{t-1}^S \cdot \omega_s^O)$
 - ▶ \otimes = element-wise multiplication
- ▶ Long-Term Memory Update:

$$\mathbf{s}_t^L = g_F(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes \mathbf{s}_{t-1}^L + g_I(\mathbf{x}_t, \mathbf{s}_{t-1}^S) \otimes g_U(\mathbf{x}_t, \mathbf{s}_{t-1}^S)$$

- ▶ forget gate: $g_F = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^F + \mathbf{s}_{t-1}^S \cdot \omega_s^F)$
- ▶ input gate: $g_I = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^I + \mathbf{s}_{t-1}^S \cdot \omega_s^I)$
- ▶ state updater: $g_U = \tanh(\mathbf{x}_t \cdot \omega_x^U + \mathbf{s}_{t-1}^S \cdot \omega_s^U)$
- ▶ gating mechanisms prevent vanishing/exploding gradients.

GRU (Gated Recurrent Unit)

- ▶ GRU is a simplified LSTM with fewer parameters but similar performance (Cho et al 2014; Chung et al 2014).
- ▶ Does not separate long/short term memory, so just has a state vector $s_t \in \mathbb{R}^{n_y}$
- ▶ Output:

$$y_t = s_t$$

GRU (Gated Recurrent Unit)

- ▶ GRU is a simplified LSTM with fewer parameters but similar performance (Cho et al 2014; Chung et al 2014).
- ▶ Does not separate long/short term memory, so just has a state vector $\mathbf{s}_t \in \mathbb{R}^{n_y}$
- ▶ Output:

$$\mathbf{y}_t = \mathbf{s}_t$$

- ▶ State Update:

$$\mathbf{s}_t = (1 - g_F(\mathbf{x}_t, \mathbf{s}_{t-1})) \otimes \mathbf{s}_{t-1} + g_F(\mathbf{x}_t, \mathbf{s}_{t-1}) \otimes g_U(\mathbf{x}_t, \mathbf{s}_{t-1})$$

- ▶ forget gate: $g_F = \text{sigmoid}(\mathbf{x}_t \cdot \omega_x^F + \mathbf{s}_{t-1} \cdot \omega_s^F)$
- ▶ update proposal: $g_U = \tanh(\mathbf{x}_t \cdot \omega_x^U + (\text{sigmoid}(\mathbf{x}_t \cdot \omega_x^U + \mathbf{s}_{t-1} \cdot \omega_s^U) \otimes \mathbf{s}_{t-1}) \cdot \omega_s^S)$

Dropout / Normalization in RNNs

- ▶ Gal (2015):
 - ▶ when applying dropout to RNNs, use same mask at all t .
 - ▶ dropout mask sampled once per sequence, not once per time period.

Dropout / Normalization in RNNs

- ▶ Gal (2015):
 - ▶ when applying dropout to RNNs, use same mask at all t .
 - ▶ dropout mask sampled once per sequence, not once per time period.
- ▶ Batch normalization doesn't work well for RNN's.
- ▶ Instead, can use **layer normalization**:
 - ▶ normalize each instance across feature dimensions
 - ▶ (rather than normalizing each feature across instances in the batch, as in BN)

Combining CNN's and RNN's

- ▶ Geron (Ch. 15, pp. 520-521) suggests the following hybrid network for sequence-to-vector tasks:
 1. four-word convolutional layer with twenty filters and a stride of two.
 2. GRU on top of the twenty filters
- ▶ Actually gets best performance on a sample task forecasting stock market prices.

RNNs: Practical Use for Sequence-to-Vector Task

For example, sentiment analysis:

- ▶ tokenize the documents into $w_{1:n}$
- ▶ embedding $x_t = w_t \cdot E$
 - ▶ embedding matrix E can be initialized with pre-trained GloVe embeddings.
- ▶ bidirectional RNN (e.g. LSTM) on $x_{1:n}$ (and $x_{n:1}$) to generate document vector s
- ▶ MLP takes s to predict y

RNNs: Practical Use for Sequence-to-Vector Task

For example, sentiment analysis:

- ▶ tokenize the documents into $\mathbf{w}_{1:n}$
- ▶ embedding $\mathbf{x}_t = \mathbf{w}_t \cdot \mathbf{E}$
 - ▶ embedding matrix \mathbf{E} can be initialized with pre-trained GloVe embeddings.
- ▶ bidirectional RNN (e.g. LSTM) on $\mathbf{x}_{1:n}$ (and $\mathbf{x}_{n:1}$) to generate document vector \mathbf{s}
- ▶ MLP takes \mathbf{s} to predict \mathbf{y}

keras functional API makes this straightforward.

RNNs: Practical Tips

- ▶ Initialization:
 - ▶ recurrent matrices to be orthogonal
 - ▶ other matrices to have small scale
 - ▶ forget gate to 1 (default to remember)
- ▶ Learning rate:
 - ▶ use adaptive learning rate, e.g. Adam
- ▶ Gradient clipping: clip the norm, 1-5 seems reasonable threshold.

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.

Predicting partisanship: Disagreement Analysis

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

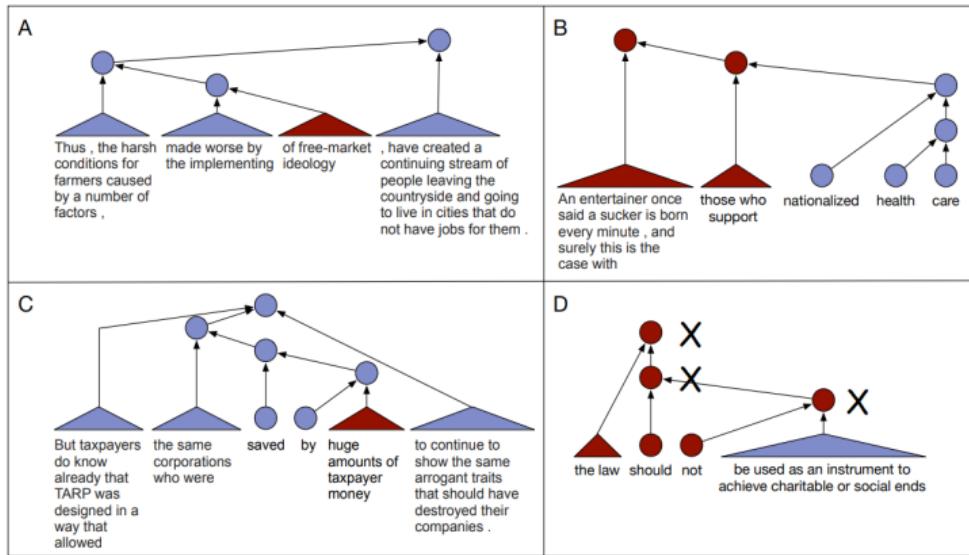


Figure 5: Predictions by **RNN2-(W2V)** on four sentences from the IBC. Node color is the true label (red for conservative, blue for liberal), and an “X” next to a node means the model’s prediction was wrong. In A and C, the model accurately detects conservative-to-liberal polarity switches, while in B it correctly predicts the liberal-to-conservative switch. In D, negation confuses our model.

- ▶ A, B, and C were all mis-classified by standard bag-of-ngrams models. D tricks all the models.

Vectors and Sequences

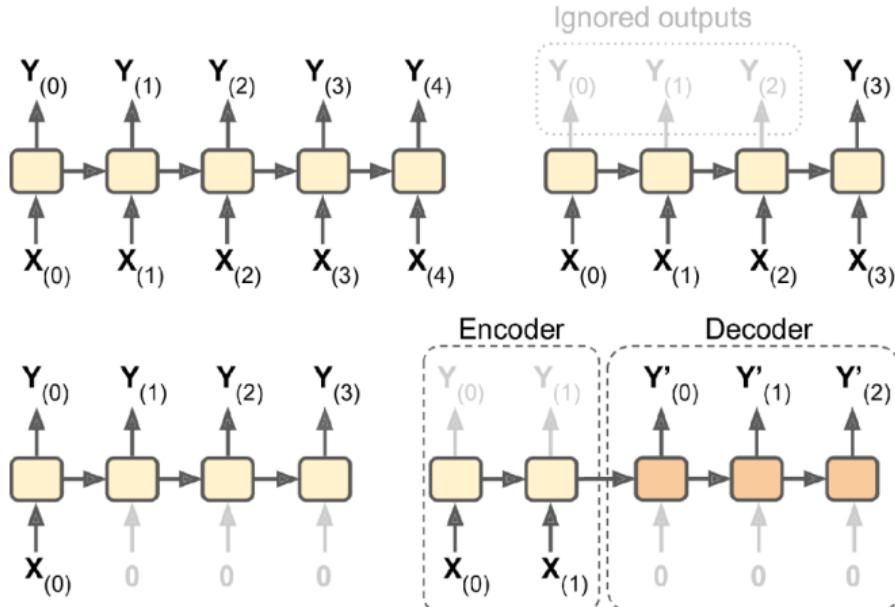


Figure 15-4. Seq-to-seq (top left), seq-to-vector (top right), vector-to-seq (bottom left), and Encoder-Decoder (bottom right) networks

- ▶ top left: sequence to sequence; top right: sequence to vector; bottom left: vector to sequence.
- ▶ bottom right: encoder-decoder.

Recurrent Autoencoders

To compress text (or other sequential) data, 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

To compress text (or other sequential) data, 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.

Training an RNN Translator

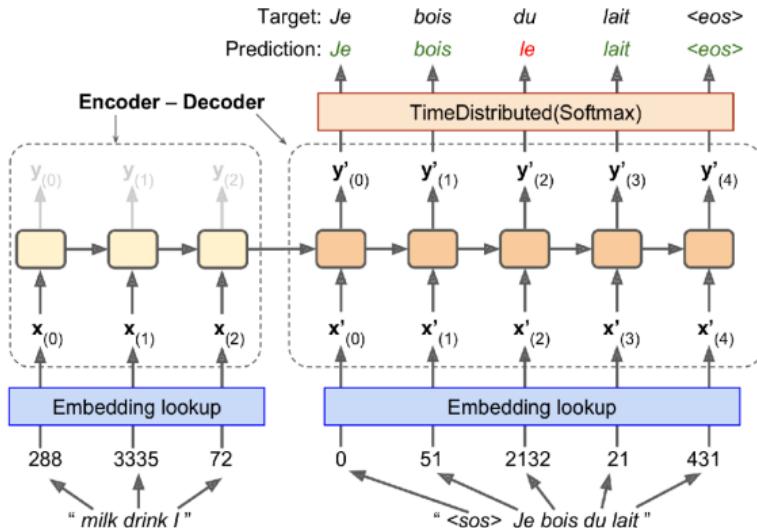


Figure 16-3. A simple machine translation model

1. English sentences are input to the encoder in reverse.
2. Decoder takes the encoded vector and the true French sentence, lagged by one word.
3. Output layer is softmax over French vocabulary.

Using the Translator

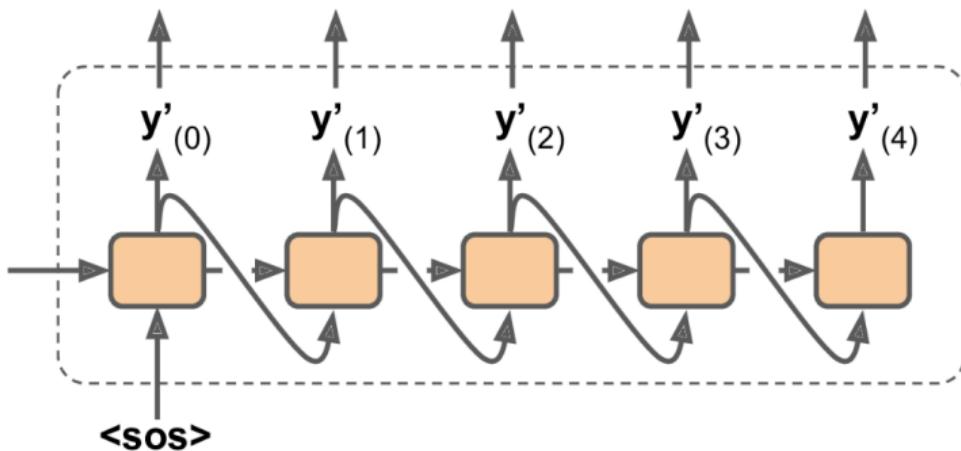


Figure 16-4. Feeding the previous output word as input at inference time

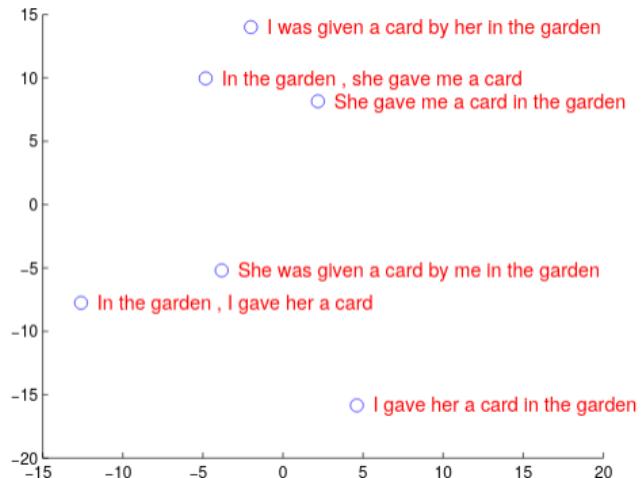
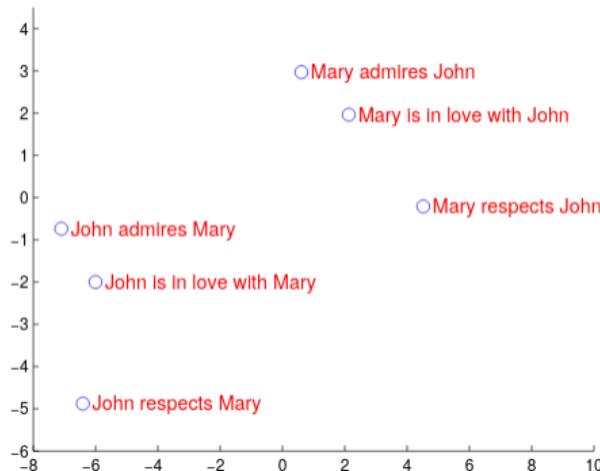
- ▶ When using the machine for translation, give it the start-of-sentence tag to get started, then feed in the generated words.

Geometry of Embedded Sequences

- ▶ Like CNN's, RNN's produce document vectors s with interpretable geometry.

Geometry of Embedded Sequences

- ▶ Like CNN's, RNN's produce document vectors s with interpretable geometry.



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

Outline

Convolutional Neural Nets

Recurrent Neural Nets

Rationales

Lei et al (2016)

Sanford, Roberts, and Li (2019)

Attention / Tranformers

Outline

Convolutional Neural Nets

Recurrent Neural Nets

Rationales

Lei et al (2016)

Sanford, Roberts, and Li (2019)

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.

- ▶ Key idea: find minimal span(s) of text that can (by themselves) explain the prediction

Architecture

Lei, Barzilay, and Jaakola (2016)

- ▶ x = input text sequence, y = output class

Architecture

Lei, Barzilay, and Jaakola (2016)

- ▶ x = input text sequence, y = output class
- ▶ Encoder network:
 - ▶ takes the output of the generator, inputs to an RNN (e.g. bidirectional GRU) to predict the outcome from the text.

Architecture

Lei, Barzilay, and Jaakola (2016)

- ▶ x = input text sequence, y = output class
- ▶ Encoder network:
 - ▶ takes the output of the generator, inputs to an RNN (e.g. bidirectional GRU) to predict the outcome from the text.
- ▶ Generator network:
 - ▶ outputs a probability distribution for sub-sequences being rationales – snippets of text that are sent to the encoder network.

Architecture

Lei, Barzilay, and Jaakola (2016)

- ▶ x = input text sequence, y = output class
- ▶ Encoder network:
 - ▶ takes the output of the generator, inputs to an RNN (e.g. bidirectional GRU) to predict the outcome from the text.
- ▶ Generator network:
 - ▶ outputs a probability distribution for sub-sequences being rationales – snippets of text that are sent to the encoder network.
 - ▶ in conjunction with the encoder, works to extract a subset of text that is as predictive as the full text.

Architecture

Lei, Barzilay, and Jaakola (2016)

- ▶ x = input text sequence, y = output class
- ▶ Encoder network:
 - ▶ takes the output of the generator, inputs to an RNN (e.g. bidirectional GRU) to predict the outcome from the text.
- ▶ Generator network:
 - ▶ outputs a probability distribution for sub-sequences being rationales – snippets of text that are sent to the encoder network.
 - ▶ in conjunction with the encoder, works to extract a subset of text that is as predictive as the full text.
- ▶ Joint objective:
 - ▶ predicting y correctly.
 - ▶ regularizer that penalizes long rationale sequences.

Sentiments about Beer

a beer that is not sold in my neck of the woods , but managed to get while on a roadtrip . poured into an imperial pint glass with a generous head that sustained life throughout . nothing out of the ordinary here , but a good brew still . body was kind of heavy , but not thick . the hop smell was excellent and enticing . very drinkable

very dark beer . pours a nice finger and a half of creamy foam and stays throughout the beer . smells of coffee and roasted malt . has a major coffee-like taste with hints of chocolate . if you like black coffee , you will love this porter . creamy smooth mouthfeel and definitely gets smoother on the palate once it warms . it 's an ok porter but i feel there are much better one 's out there .

i really did not like this . it just seemed extremely watery . i dont ' think this had any carbonation whatsoever . maybe it was flat , who knows ? but even if i got a bad brew i do n't see how this would possibly be something i 'd get time and time again . i could taste the hops towards the middle , but the beer got pretty nasty towards the bottom . i would never drink this again , unless it was free . i 'm kind of upset i bought this .

a : poured a nice dark brown with a tan colored head about half an inch thick , nice red/garnet accents when held to the light . little clumps of lacing all around the glass , not too shabby . not terribly impressive though s : smells like a more guinness-y guinness really , there are some roasted malts there , signature guinness smells , less burnt though , a little bit of chocolate ... m : relatively thick , it is n't an export stout or imperial stout , but still is pretty hefty in the mouth , very smooth , not much carbonation . not too shabby d : not quite as drinkable as the draught , but still not too bad . i could easily see drinking a few of these .

Figure 3: Examples of extracted rationales indicating the sentiments of various aspects. The extracted texts for appearance, smell and palate are shown in red, blue and green color respectively. The last example is shortened for space.

Relevant Features for Identifying Duplicate Questions

what is the easiest way to [install all the media codec available](#) for ubuntu ? i am having issues with multiple applications prompting me to install codecs before they can play my files . how do i [install media codecs](#) ?

what should i do when i see <unk> [report](#) this <unk> ? an [unresolvable problem occurred](#) while initializing the package information . please report this bug against the 'update-manager' package and include the following error message : e : encountered a [section with no package : header e : problem with mergelist <unk>](#) e : the package lists or status file could not be parsed or opened .

please any one give the solution for this whenever i try [to convert the rpm file to deb](#) file i always get this problem error : <unk> : not an [rpm package](#) (or package [manifest](#)) error executing `` lang=c rpm -qp -- queryformat % { name } <unk> '' : at <unk> line 489 thanks converting [rpm file](#) to debian fle

how do i [mount a hibernated partition with windows 8 in](#) ubuntu ? i ca n't mount my other partition with windows 8 , i have ubuntu 12.10 amd64 : [error mounting /dev/sda1](#) at <unk> : command-line 'mount -t " ntfs " -o " uhelper=udisks2 , nodev , [nosuid](#) , uid=1000 , gid=1000 , dmask=0077 , fmask=0177 " "/dev/sda1 " <unk> " ' exited with non-zero exit status 14 : windows is hibernated , refused to mount . failed to mount '/dev/sda1 ' : operation not permitted the ntfs partition is hibernated . please resume and [shutdown windows](#) properly , or mount the volume read-only with the 'ro ' mount option

Figure 7: Examples of extracted rationales of questions in the AskUbuntu domain.

Outline

Convolutional Neural Nets

Recurrent Neural Nets

Rationales

Lei et al (2016)

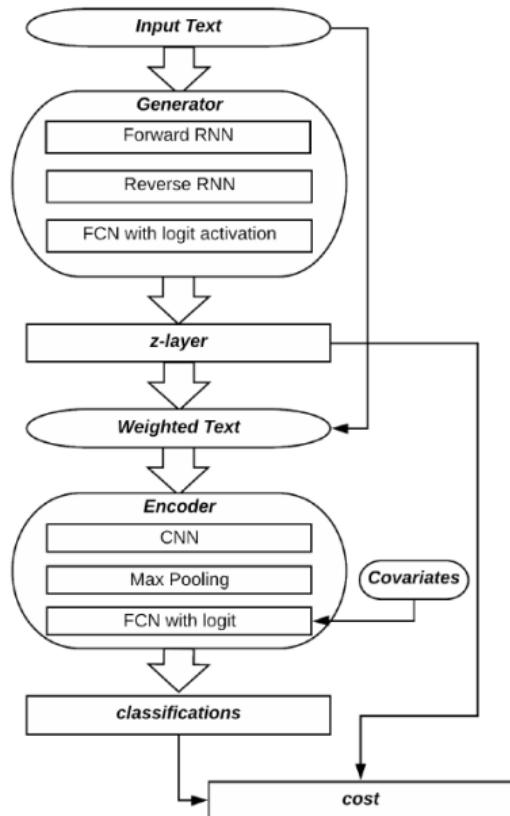
Sanford, Roberts, and Li (2019)

Attention / Tranformers

Discovering Influential Text

- ▶ Sanford, Roberts, and Li (2019) apply the Lei et al (2016) model in a social science setting.
- ▶ The stated goals:
 1. Recognize phrases of varying length that are predictive of an outcome.
 2. Allow for highlighted phrases to flexibly express the same idea.
 3. Map identified phrases to a common vector for interpretability.
 4. Allow for the inclusion of non-text covariates to aid in prediction.

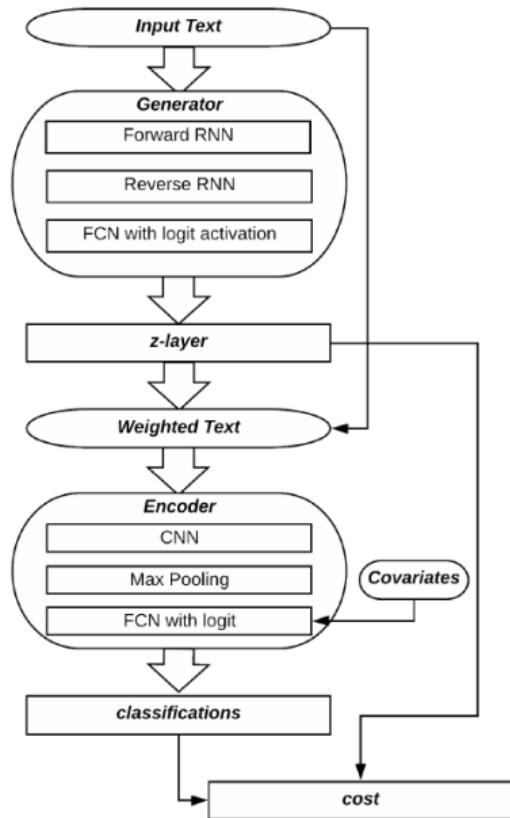
Model Overview



► Pre-Processing:

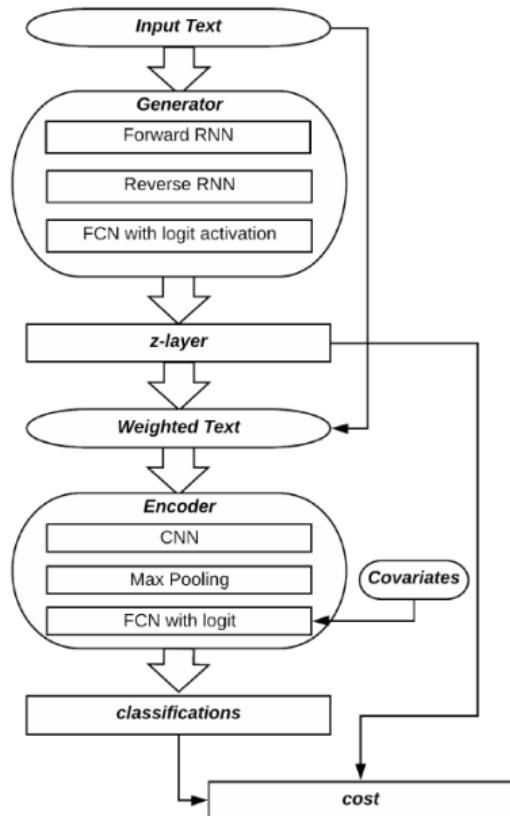
- tokenize
- convert to lower-case
- pad to constant length (e.g. 100 words)

Model Overview



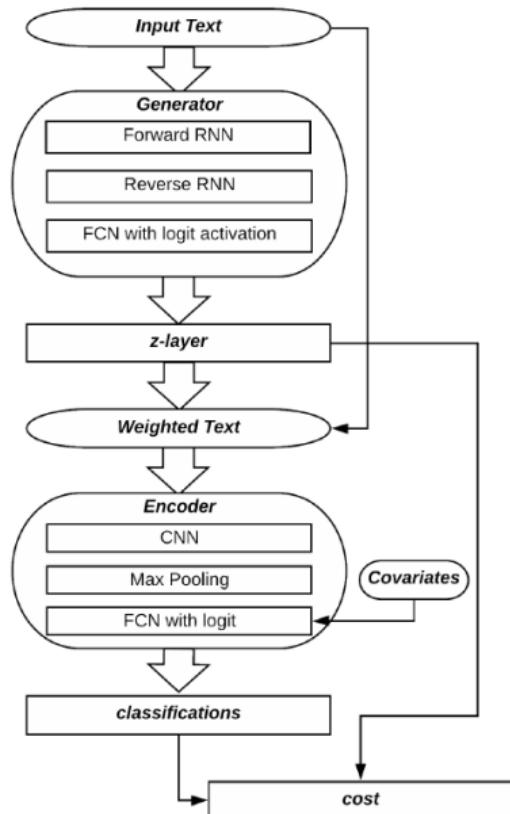
- ▶ Pre-Processing:
 - ▶ tokenize
 - ▶ convert to lower-case
 - ▶ pad to constant length (e.g. 100 words)
- ▶ Embedding:
 - ▶ 300-dimensional word embeddings, pre-initialized from fastText embeddings.

Model Overview



- ▶ Pre-Processing:
 - ▶ tokenize
 - ▶ convert to lower-case
 - ▶ pad to constant length (e.g. 100 words)
- ▶ Embedding:
 - ▶ 300-dimensional word embeddings, pre-initialized from fastText embeddings.
- ▶ Generator:
 1. $x \rightarrow$ forward RNN $\rightarrow h_1$
 2. $h_1 \rightarrow$ backward RNN $\rightarrow h_2$
 3. $h_2 \rightarrow$ MLP $\rightarrow z$, the rationale filters

Model Overview



- ▶ Pre-Processing:
 - ▶ tokenize
 - ▶ convert to lower-case
 - ▶ pad to constant length (e.g. 100 words)
- ▶ Embedding:
 - ▶ 300-dimensional word embeddings, pre-initialized from fastText embeddings.
- ▶ Generator:
 1. $x \rightarrow$ forward RNN $\rightarrow h_1$
 2. $h_1 \rightarrow$ backward RNN $\rightarrow h_2$
 3. $h_2 \rightarrow$ MLP $\rightarrow z$, the rationale filters
- ▶ Encoder
 - ▶ $x \cdot z \rightarrow$ weighted text w
 - ▶ $w \rightarrow$ convolution, max pooling $\rightarrow c$ (feature map)
 - ▶ $c \rightarrow$ softmax (add covariates) $\rightarrow y$

Joint Objective

$$\underbrace{\mathcal{L}(z, x, y)}_{\text{cross-entropy}} + \lambda_1 \underbrace{\left(\sum_{i=1}^{n_z} z_i - \bar{z} \right)^2}_{\text{sparsity}} + \lambda_2 \underbrace{\sum_{i=1}^{n_z} |z_i - z_{i-1}|}_{\text{coherence}}$$

- ▶ cross-entropy rewards accurate predictions.
- ▶ sparsity: penalty on more rationales.
 - ▶ n_z = maximum rationale length.
 - ▶ \bar{z} is a sparsity “target”; encourages rationales to be this length.
- ▶ coherence:
 - ▶ encourages high weights to be consecutive – encourages n-grams rather than single words.

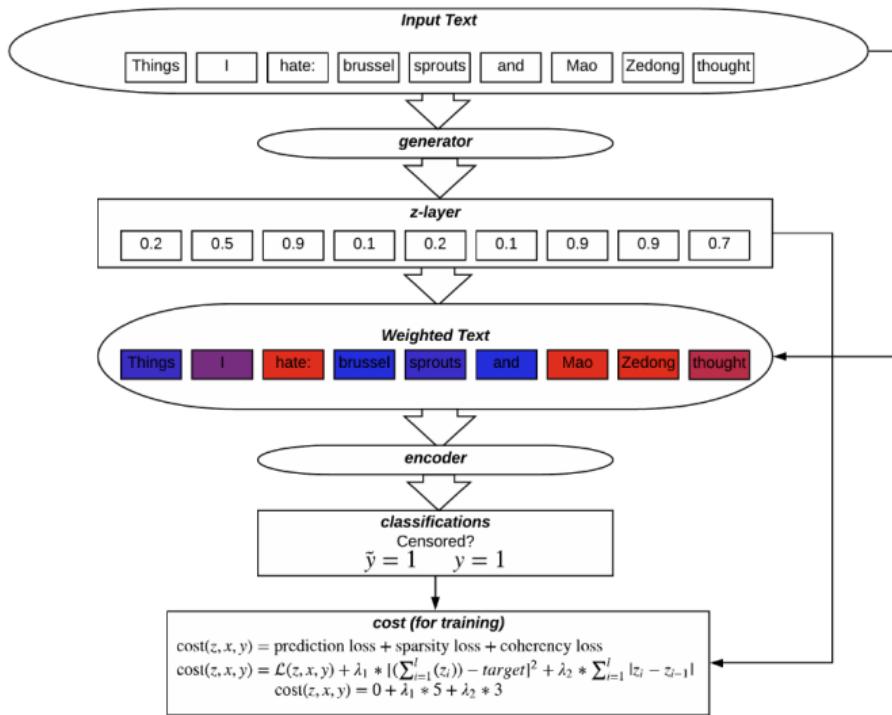


Figure 4: Detail view of how a text passes through the network. The z-layer tells the network which words to select, the rationale is a new text that the encoder uses to generate its classification. The loss function is composed of prediction error, a sparsity loss, and a coherency cost.

Interpreting the Rationales

- ▶ Model provides convolutional filters:
 - ▶ by themselves, not interpretable
 - ▶ but can see which (frequent) phrases in the corpus activate them.

Interpreting the Rationales

- ▶ Model provides convolutional filters:
 - ▶ by themselves, not interpretable
 - ▶ but can see which (frequent) phrases in the corpus activate them.
 - ▶ Sanford et al find that doing this in the test set produces more intuitive rationales.

Example Rationales: Chinese Censorship

Phrase	Translation
饥荒 饿死 3600万	Famine that starved 36 million
被 暴徒 殴打	Beaten by a mob
甘肃 省委 常委	Gansu Provincial Party Committee Standing Committee
爱国 主义 信仰	Patriotism
抢 劫 国有 资产	Robbery of state-owned assets
市委 书记 ——	Party secretary
变相 颠覆 国家	Disguised subversion
劳动 人民 在	Working People
人民 谋 和平	People want peace
人民 书记 人民	People's secretary, people
习近平 书记 领导	Xi Jinping Secretary leader
中央 经济 会议	Central economic conference
中国 民主 前途	China's democratic future

Table 1: Strongest Filter Phrases Associated With Censorship

Phrase	Translation
11 点 开 抢	Opens at 11 !!
!! 吧 !	!! right? !
! love your [爱	love your love
! your is so good	! your is so good
11 点 开	Opens at 11 !!
>> http :	>> http :
!!	!!
美[爱	beautiful love
!!	!!
bb < 3	bb < 3
with bb < 3	with bb < 3
包邮】11 点	Shipping 11 o'clock
!! 吧	right?

Table 2: Phrases Associated with Lowest Censorship

Corpus of 20,000 social media posts; 10,000 were censored; pre-trained fastText embeddings; 20 filters of width 3; 80% test-set accuracy.

Outline

Convolutional Neural Nets

Recurrent Neural Nets

Rationales

Lei et al (2016)

Sanford, Roberts, and Li (2019)

Attention / Tranformers

Attention Mechanisms

- ▶ In 2014 the addition of “attention mechanisms” revolutionized neural machine translation.
 - ▶ and by 2019, NLP generally.
- ▶ The key difference is that a sequence is not represented as a single vector, but rather a sequence of vectors, weighted by their importance to a task.

Attention for Translation

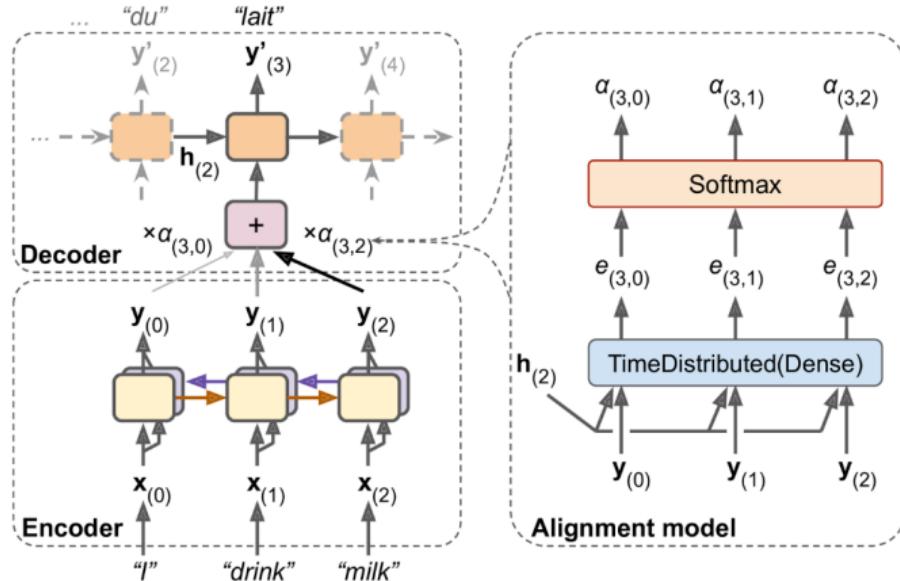


Figure 16-6. Neural machine translation using an Encoder-Decoder network with an attention model

- ▶ where do these α 's come from?

Attention is All you Need: Transformers

▶ ...