Recurrent Neural Networks and Neural Language Modelling

COM4513/6513 Natural Language Processing

Nikos Aletras
n.aletras@sheffield.ac.uk

@nikaletras

Computer Science Department

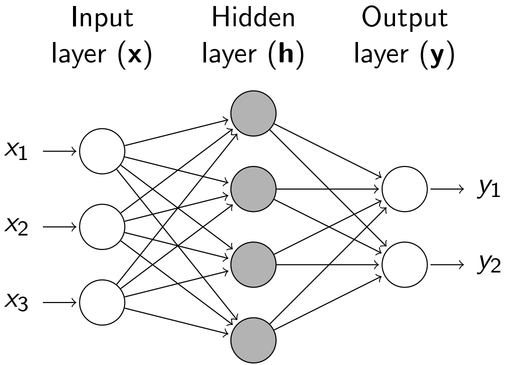
Week 8
Spring 2020

The University Of Sheffield.

In lecture 6...

- **Feedforward Neural Networks** and how to train them with **Backprop**
- Feedforward nets are useful to learn word representations but they ignore word order and dependencies between words in a given document.

Feedforward Neural Network



$\mathbf{h} = g(\mathbf{W}_h \mathbf{x})$
$\mathbf{y} = softmax(\mathbf{W}_o \mathbf{h})$
$\mathbf{W}_o \in \mathcal{R}^{h \times y}$

$g(\cdot)$ is an activation function

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**
- Language modelling: return sentence probabilities as well as representations

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**
- Language modelling: return sentence probabilities as well as representations
- Text classification: learn contextualised word representations and use them to predict a given class

In this lecture...

- **Recurrent Neural Networks (RNNs)** to capture long-range dependencies in a document
- Train with **SGD** and **Backpropagation through Time**
- RNN extensions: **Long-Short Term Memory (LSTM)** and **Gated-Recurrent Unit (GRU)**
- Language modelling: return sentence probabilities as well as representations
- Text classification: learn contextualised word representations and use them to predict a given class
- Improve RNNs with **Attention**

Neural Language Modelling: Problem Setup

Training data is a (large) set of word sequences:

$$D_{train} = \{\mathbf{x}^1, ..., \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, ...x_N]$$

Neural Language Modelling: Problem Setup

Training data is a (large) set of word sequences:

$$D_{train} = \{\mathbf{x}^1, ..., \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, ...x_N]$$

for example:

$$\mathbf{x} = [its, water, is, so, transparent, STOP]$$

Neural Language Modelling: Problem Setup

Training data is a (large) set of word sequences:

$$D_{train} = \{\mathbf{x}^1, ..., \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, ...x_N]$$

for example:

$$\mathbf{x} = [its, water, is, so, transparent, STOP]$$

We want to learn a model that returns:

$$P(\mathbf{x}), \textbf{for } \forall \mathbf{x} \in V^{maxN}$$

V is the vocabulary and V^{maxN} all possible sentences

Language modelling as classification

$$
\begin{aligned}
P(\mathbf{x}) &= P(x_1, ..., x_N) \\
&= P(x_1)P(x_2...x_N|x_1) \\
&= P(x_1)P(x_2|x_1)...P(x_N|x_1, ..., x_{N-1})
\end{aligned}
$$

Language modelling as classification

$$
\begin{aligned}
P(\mathbf{x}) &= P(x_1, ..., x_N) \\
&= P(x_1)P(x_2...x_N|x_1) \\
&= P(x_1)P(x_2|x_1)...P(x_N|x_1, ..., x_{N-1})
\end{aligned}
$$

Let's write the probabilities as LR (remember the CRF?):

$$
p(x_n = k|x_{n-1}...x_1) = \frac{\exp(\mathbf{w}_k \cdot \phi(x_{n-1}...x_1))}{\sum_{k'=1}^{|\mathcal{V}|} \exp(\mathbf{w}_{k'} \cdot \phi(x_{n-1}...x_1))}
$$

- $\mathbf{w}_k$ are the weights for word k
- $\phi(x_{n-1}...x_1)$ are the features extracted from the previous words (one-hot encoding of $x_{n-1}...x_1$)

Representing word sequences

Looks like a neural network:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{W}\phi(x_{n-1}...x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{C}|}$ has weights for each word and context

Representing word sequences

Looks like a neural network:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{W}\phi(x_{n-1}...x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{C}|}$ has weights for each word and context

Let's represent the context with a vector $s_{n-1} \in \mathcal{R}^d$:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{V}s_{n-1})$$

Representing word sequences

Looks like a neural network:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{W}\phi(x_{n-1}...x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{C}|}$ has weights for each word and context

Let's represent the context with a vector $s_{n-1} \in \mathcal{R}^d$:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{V}s_{n-1})$$

$\mathbf{V} \in \mathcal{R}^{|\mathcal{V}|\times d}$ maps the context to a probability distribution over the words.

Representing word sequences

Looks like a neural network:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{W}\phi(x_{n-1}...x_1))$$

$\mathbf{W} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{C}|}$ has weights for each word and context

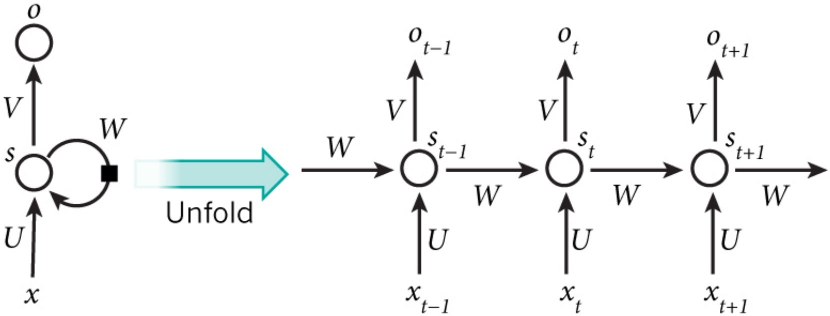
Let's represent the context with a vector $s_{n-1} \in \mathcal{R}^d$:

$$p(x_n|x_{n-1}...x_1) = softmax(\mathbf{V}s_{n-1})$$

$\mathbf{V} \in \mathcal{R}^{|\mathcal{V}|\times d}$ maps the context to a probability distribution over the words.

How do we get s_{n-1}?

Recurrent neural networks



When generating, x_t is the highest-scoring word in o_{t-1}

Recurrent Neural Networks

$$s_n = \sigma(\mathbf{W}s_{n-1} + \mathbf{U}x_n)$$

- $s_{n-1} \in \mathcal{R}^d$: "memory" of the context until word x_{n-1}
- $\mathbf{W} \in \mathcal{R}^{d \times d}$: controls how this memory is passed on
- $\mathbf{U} \in \mathcal{R}^{|\mathcal{V}| \times d}$: matrix containing the word vectors for all the words, x_n picks one

Recurrent Neural Networks

$$s_n = \sigma(\mathbf{W}s_{n-1} + \mathbf{U}x_n)$$

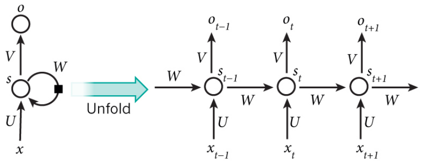
- $s_{n-1} \in \mathcal{R}^d$: "memory" of the context until word x_{n-1}
- $\mathbf{W} \in \mathcal{R}^{d \times d}$: controls how this memory is passed on
- $\mathbf{U} \in \mathcal{R}^{|\mathcal{V}| \times d}$: matrix containing the word vectors for all the words, x_n picks one

To get the probability distribution for word x_n:

$$\mathbf{o}_{n-1} = p(x_n|x_{n-1}...x_1) = softmax(\mathbf{V}s_{n-1})$$

- $\mathbf{V} \in \mathcal{R}^{d \times |\mathcal{V}|}$: output weight matrix

Training RNNs



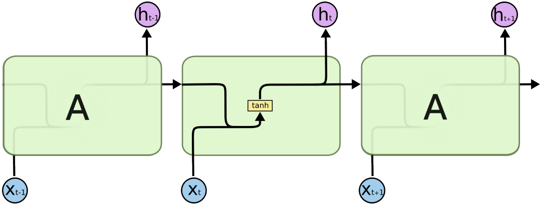
- We need to learn the word vectors $\mathbf{U}$, hidden and output layer parameters $\mathbf{W}, \mathbf{V}$
- Standard backpropagation can't work because of the recurrence: we reuse the hidden layer parameters $\mathbf{W}$
- **Backpropagation Through Time**: unroll the graph for n steps and sum the gradients in updating
- Not as restrictive as the nth-order Markov: we still use all previous words through the recurrence.

Limitations of RNNs

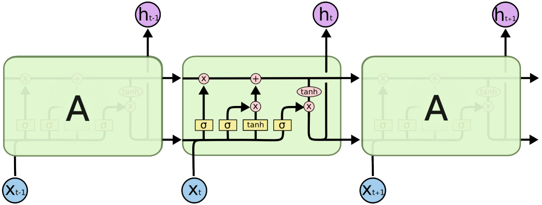
RNNs can't capture long-range dependencies:

- effectively have one layer per word in the sentence
- all context information has to be passed by the hidden layer
- vanishing gradients: the gradient from the last word often never reaches the first

Long-Short Term Memory (LSTM) network[1]



Simple RNN



LSTM

A memory cell is used in addition to the hidden layer to control what information from previous timesteps is useful in predicting.

[1] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input gate** (what new info will be stored in the memory cell):

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input gate** (what new info will be stored in the memory cell):

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

- New **memory cell candidate values**:

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t])$$

Long-Short Term Memory (LSTM) network

- **Forget gate** (what info to throw away from previous steps):

$$f_t = \sigma(W_f[h_{t-1}, x_t])$$

- **Input gate** (what new info will be stored in the memory cell):

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$

- New **memory cell candidate values**:

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t])$$

- Update memory cell (using input and output gates):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long-Short Term Memory (LSTM) network

- **Output** (decide what's the output filtered by the memory cell):

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$

$$h_t = o_t * tanh(C_t)$$

Gated Recurrent Unit (GRU[2])

- LSTM variant

[2] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Gated Recurrent Unit (GRU[2])

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

[2] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Gated Recurrent Unit (GRU[2])

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

- **Recurrent state** (merges cell state with hidden state):

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

[2] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Gated Recurrent Unit (GRU[2])

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

- **Recurrent state** (merges cell state with hidden state):

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

- New **output candidate values**:

$$\tilde{h}_t = tanh(W[r_t * h_{t-1}, x_t])$$

[2] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Gated Recurrent Unit (GRU[2])

- LSTM variant
- **Update gate** (combines input and forget gates):

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$

- **Recurrent state** (merges cell state with hidden state):

$$r_t = \sigma(W_r[h_{t-1}, x_t])$$

- New **output candidate values**:

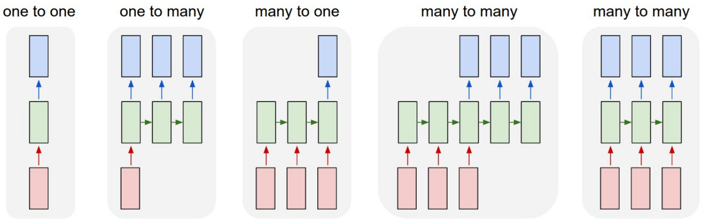
$$\tilde{h}_t = tanh(W[r_t * h_{t-1}, x_t])$$

- Output:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

[2] Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

RNN Architecture Variants in NLP



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- many to one: e.g. text classification
- many to many (equal): e.g. PoS tagging
- many to many (unequal): e.g. machine translation (coming next week), language generation, summarisation

Representation learning with RNNs

- RNNs learn word and **sentence/document representations**
- Words are not as interesting since RNNs are slower to train than Skip-Gram: thus use less data
- hint: use pre-trained word vectors (e.g. skipgram) to initialise the RNN word vectors
- RNN sentence/document representations though are used often!
- Bi-directional RNNs can also be used to learn document representations: one RNN parsing the input from start to end and another one from end to start.

Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.

Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.

Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.
- But that might not be the case!

Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.
- But that might not be the case!
- **Attention**: compute a weighted linear combination of all the contextualised representations obtained from the RNN:

$$\mathbf{c} = \sum_i \mathbf{h}_i \alpha_i$$

Improving RNNs with Attention

- In many-to-one tasks (e.g. text classification), usually the outputs from each timestep are combined (concatenated/averaged/summed) and then passed to the output layer.
- This **naive combination** assumes that all representations (from each timestep) have **equal contribution**.
- But that might not be the case!
- **Attention**: compute a weighted linear combination of all the contextualised representations obtained from the RNN:

$$\mathbf{c} = \sum_i \mathbf{h}_i \alpha_i$$

- Pass **c** to the output layer for classification

Attention Mechanism

- Attention usually consists of a similarity function ϕ followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h_i}, \mathbf{q}))}{\sum_{k=1}^{t} \exp(\phi(\mathbf{q}, \mathbf{h_k}))}$$

[3] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[4] Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

Attention Mechanism

- Attention usually consists of a similarity function ϕ followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h_i}, \mathbf{q}))}{\sum_{k=1}^{t} \exp(\phi(\mathbf{q}, \mathbf{h_k}))}$$

- $\mathbf{q} \in R^N$ is a trainable vector (learns task specific information)

[3] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[4] Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

Attention Mechanism

- Attention usually consists of a similarity function ϕ followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h_i}, \mathbf{q}))}{\sum_{k=1}^{t} \exp(\phi(\mathbf{q}, \mathbf{h_k}))}$$

- $\mathbf{q} \in R^N$ is a trainable vector (learns task specific information)
- Additive (or tanh):[3]

$$\phi(h_i, \mathbf{q}) = \mathbf{q}^T \tanh(W\mathbf{h}_i)$$

[3] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[4] Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

Attention Mechanism

- Attention usually consists of a similarity function ϕ followed by softmax:

$$a_i = \frac{\exp(\phi(\mathbf{h_i}, \mathbf{q}))}{\sum_{k=1}^{t} \exp(\phi(\mathbf{q}, \mathbf{h_k}))}$$

- $\mathbf{q} \in R^N$ is a trainable vector (learns task specific information)
- Additive (or tanh):[3]

$$\phi(h_i, \mathbf{q}) = \mathbf{q}^T \tanh(W\mathbf{h}_i)$$

- Scaled Dot-Product:[4]

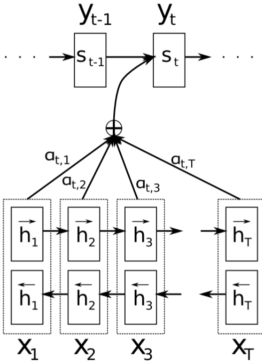
$$\phi(h_i, \mathbf{q}) = \frac{\mathbf{h}_i^T \mathbf{q}}{\sqrt{N}}$$

---

[3] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[4] Vaswani, Ashish et al. (2017). Attention is all you need. In Advances in neural information processing systems, pp. 5998-6008.

Attention Mechanism



http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

Bibliography

- Chapters 6-8 from Goodfellow et al.
- Section 6.3 from Eisenstein
- Section 10 from Goldberg
- Blog post on RNNs
- Blog post on LSTMs from where some of the figures were taken
- Blog post on attention

Coming up next..

- Sequence-to-Sequence models and Machine Translation by Dr. Fred Blain