Deep learning for natural language processing
Convolutional and recurrent neural networks

Benoit Favre <benoit.favre@univ-mrs.fr>

Aix-Marseille Université, LIF/CNRS

22 Feb 2017

Deep learning for Natural Language Processing

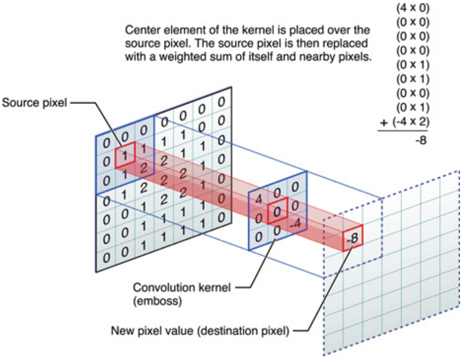
- Day 1
  - Class: intro to natural language processing
  - Class: quick primer on deep learning
  - Tutorial: neural networks with Keras
- Day 2
  - Class: word representations
  - Tutorial: word embeddings
- Day 3
  - Class: convolutional neural networks, recurrent neural networks
  - Tutorial: sentiment analysis
- Day 4
  - Class: advanced neural network architectures
  - Tutorial: language modeling
- Day 5
  - Tutorial: Image and text representations
  - Test

Extracting basic features from text

- Historical approaches
  - Text classification
  - Information retrieval
- The bag-of-word model
  - A document is represented as a vector over the lexicon
  - Its components are weighted by the frequency of the words it contains
  - Compare two texts as the cosine similarity between
- Useful features
  - Word n-grams
  - tf$\times$idf weighting
  - Syntax, morphology, etc
- Limitations
  - Each word is represented by one dimension (no synonyms)
  - Word order is only lightly captured
  - No long-term dependencies

Convolutional Neural Networks (CNN)

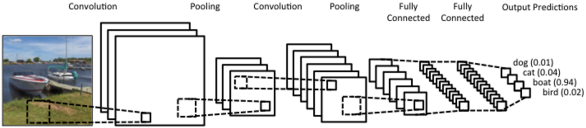
- Main idea
  - Created for computer vision
  - How can location independence be enforced in image processing?
  - Solution: split the image in overlapping patches and apply the classifier on each patch
  - Many models can be used in parallel to create filters for basic shapes



Source: https://i.stack.imgur.com/GvsBA.jpg

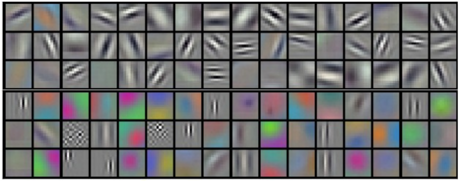
CNN for images

- Typical network for image classification (Alexnet)



Source: http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-07-at-7.26.20-AM.png

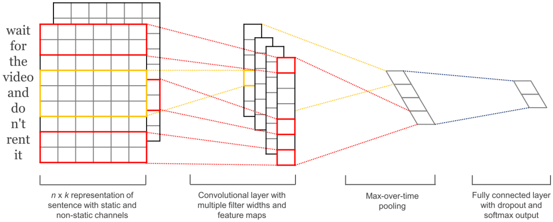
- Example of filters learned for images



Source: http://cs231n.github.io/convolutional-networks

CNN for text

- In the text domain, we can learn from sequences of words
  - Moving window over the word embeddings
  - Detects relevant word n-grams
  - Stack the detections at several scales



Source: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow

CNN Math

- Parallel between text and images
  - Images are of size (width, height, channels)
  - Text is a sequence of length n of word embeddings of size d
  - $\rightarrow$ Text is treated as an image of with n and height d
- x is a matrix of n word embeddings of size d
  - $x_{i-\frac{l}{2}:i+\frac{l}{2}}$ is a window of word embeddings centered in i, of length l
  - First, we reshape $x_{i-\frac{l}{2}:i+\frac{l}{2}}$ to a size of $(1, l \times d)$ (vertical concatenation)
  - Use this vector for $i \in [\frac{l}{2} \dots n - \frac{l}{2}]$ as CNN input
- A CNN is a set of k convolution filters
  - $\text{CNN}_{out} = activation(W\, \text{CNN}_{in} + b)$
  - CNN_{in} is of shape $(l \times d, n - l)$
  - W is of shape $(k, l \times d)$, b is of shape $(k, 1)$ repeated $n - l$ times
  - CNN_{out} is of shape $(k, n - l)$
- Interpretation
  - If $W(i)$ is an embedding n-gram, then $\text{CNN}_{out}(i, j)$ is high when this embedding n-gram is in the input

Pooling

- A CNN detects word n-grams at each time step
  - We need position independence (bag of words, bag of n-grams)
  - Combination of n-grams
- Position independence (pooling over time)
  - Max pooling $\rightarrow max_t(\mathrm{CNN}_{out}(:, t))$
  - Only the highest activated n-gram is output for a given filter
- Decision layers
  - CNNs of different lengths can be stacked to capture n-grams of variable length
  - CNN+Pooling can be composed to detect large scale patterns
  - Finish by fully connected layers which input the flatten representations created by CNNs

Online demo

- CNN for image processing
  - Digit recognition
    - `http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html`
  - 10-class visual concept
    - `http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html`

Recurrent Neural Networks

- CNNs are good at modeling topical and position-independent phenomena
  - Topic classification, sentiment classification, etc
  - But they are not very good at modeling order and gaps in the input
    - Not possible to do machine translation with it
- Recurrent NNs have been created for language modeling
  - Can we predict the next word given a history?
  - Can we discriminate between a sentence likely to be correct language and garbage?
- Applications of language modeling
  - Machine translation
  - Automatic speech recognition
  - Text generation...

Language modeling

Measure the quality of a sentence

- Word choice and word order
  - (+++) *the cat is drinking milk*
  - (++) *the dog is drinking lait*
  - (+) *the chair is drinking milk*
  - (-) *cat the drinking milk is*
  - (–) *cat drink milk*
  - (—) *bai toht aict*

If $w_1 \dots w_n$ is a sequence of words, how to compute $P(w_1 \dots w_n)$?

- Could be estimated with probabilities over a large corpus

$$P(w_1 \dots w_n) = \frac{count(w_1 \dots w_n)}{count(\text{possible sentences})}$$

Exercise – reorder:

- *cat the drinking milk is*
- *taller is John Josh than*

How to estimate a language model

Rewrite probability to marginalize parts of sentence

$$\begin{aligned}
P(w_1 \dots w_n) &= P(w_n|w_{n-1} \dots w_1) P(w_{n-1} \dots w_1) \\
&= P(w_n|w_{n-1} \dots w_1) P(w_{n-1}|w_{n-2} \dots w_1) \\
&= P(w_1) \prod_i P(w_i|w_{i-1} \dots w_1)
\end{aligned}$$

Note: add $\langle S \rangle$ and $\langle E \rangle$ symbols at beginning and end of sentence

$$\begin{aligned}
P(\langle S \rangle \text{cats like milk} \langle E \rangle) = & P(\langle S \rangle) \\
& \times P(\text{cats}|\langle S \rangle) \\
& \times P(\text{like}|\langle S \rangle \text{cats}) \\
& \times P(\text{milk}|\langle S \rangle \text{cats like}) \\
& \times P(\langle E \rangle|\langle S \rangle \text{cats like milk})
\end{aligned}$$

n-gram language models (Markov chains)

- Markov hypothesis: ignore history after k symbols

$$\begin{aligned} P(word_i|history_{1..i-1}) &\simeq P(word_i|history_{i-k,i-1}) \\ P(w_i|w_1 \dots w_{i-1}) &\simeq P(w_i|w_{i-k} \dots w_{i-1}) \end{aligned}$$

- For $k = 2$:

$$\begin{aligned} P(\langle S\rangle \text{cats like milk} \langle E\rangle) \simeq & P(\langle S\rangle) \times P(\text{cats}|\langle S\rangle) \times P(\text{like}|\langle S\rangle\text{cats}) \\ & \times P(\text{milk}|\text{cats like}) \times P(\langle E\rangle|\text{like milk}) \end{aligned}$$

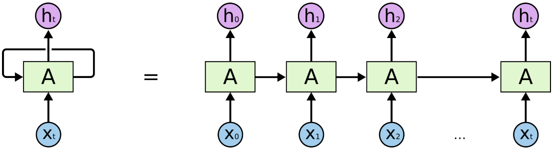
- Maximum likelihood estimation

$$P(\text{milk}|\text{cats like}) = \frac{count(\text{cats like milk})}{count(\text{cats like})}$$

- n-gram model ($n = k + 1$), use n words for estimation
  - $n = 1$: unigram, $n = 2$: bigram, $n = 3$: trigram...

Recurrent Neural Networks

- N-gram language models have proven useful, but
  - They require lots of memory
  - Make poor estimations in unseen context
  - **ignore** long-term dependencies
- We would like to account for the history all the way from w_1
  - Estimate $P(w_i|h(w_1 \dots w_{i-1})$
  - What can be used for h?
- Recurrent definition
  - $h_0 = 0$
  - $h(w_1 \dots w_{i-1}) = h_i = f(h_{i-1})$
  - That's a classifier that uses its previous output to predict the next word



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png

Simple RNNs

- Back to the $y = \text{neural\_network}(x)$ notation
  - $x = x_1 \dots x_n$ is a sequence of observations
  - $y = y_1 \dots y_n$ is a sequence of labels we want to predict
  - $h = h_1 \dots h_n$ is a hidden state (or history for language models)
  - t is discrete time (so we can write x_t for the t-th timestep

- We can define a RNN as

$$h_1 = 0 \tag{1}$$

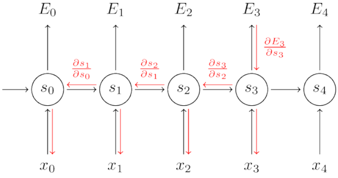
$$h_t = tanh(Wx_t + Uh_{t-1} + b) \tag{2}$$

$$y_t = softmax(W_o h_t + b_o) \tag{3}$$

- Tensor shapes
  - x_t is of shape $(1, d)$ for embeddings of size d
  - h_t is of shape $(1, H)$ for hidden state of size H
  - y_t is of shape $(1, c)$ for c labels
  - W is of shape (d, H)
  - U is of shape (H, H)
  - W_o is of shape (c, H)

Training RNNs

- Back-propagation through time (BPTT)
  - Unroll the network
  - Forward
    - Compute h_t one by one until end of sequence
    - Compute y_t from h_t
  - Backward
    - Propagate error gradient from y_t to h_t
    - Consecutively back-propagate from h_n to h_1



Source: https://pbs.twimg.com/media/CQ0CJtwUkAAL__H.png

- What if the sequence is too long?
  - Cut after n words: truncated-BPTT
  - Sample windows in the input
  - How to initialize the hidden state?
    - Use the one from the previous window (statefull RNN)

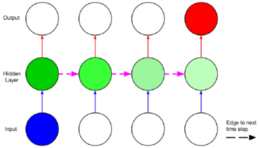
Potential problems with recurrent state

- "On the difficulty of training recurrent neural networks", Pascanu et al ICML 2013
  - Recurrent equations can be rewritten without loss of generality

$$h_t = Uf(h_{t-1}) + \text{input}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=t}^{k} U^T diag(f'(h_{i-1}))$$

- Vanishing gradient ($det \frac{\partial h_t}{\partial h_{t-1}} < 1$)
  - Gradient quickly goes to zero, preventing to learn long dependencies
- Exploding gradient ($det \frac{\partial h_t}{\partial h_{t-1}} > 1$)
  - Gradient quickly increases, making the system unstable



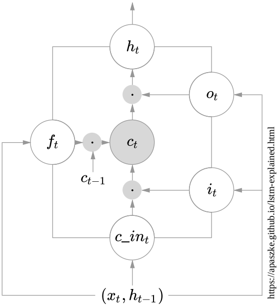
Source: https://www.researchgate.net/profile/Zachary_Lipton/publication/277603865/figure/fig8/AS:294356339707931@1447191428668/Figure-8-A-visualization-of-the-vanish

Long-short term memory

- Idea: use gating mechanism to keep information in the hidden state
  - RNN would have to refresh its memory with every input
  - LSTM output depends on gates which are trained to open at the right time
- Gating mechanism

$$g = f(x_t, h_t) \in [0, 1]$$
$$x_{\text{gated}} = g \odot x_t$$

- LSTMs have two hidden states: h and c



LSTM Math

- LSTM

$$
\begin{aligned}
i_t =& \sigma(W_i x_t + U_i h_t + b_i) && \text{input} \\
f_t =& \sigma(W_f x_t + U_f h_t + b_f) && \text{forget} \\
o_t =& \sigma(W_o x_t + U_o h_t + b_o) && \text{output} \\
c'_t =& \tanh(W_c x_t + U_c h_t + b_c) && \text{cell state} \\
c_{t+1} =& f_t \odot c_t + i_t \odot c'_t \\
h_{t+1} =& o_t \odot \tanh(c_{t+1}) \\
\mathrm{LSTM}(x_t, h_t, c_t) =& h_{t+1}
\end{aligned}
$$

- Parameters
  - $W_i, U_i, b_i, W_f, U_f, b_f, W_o, U_o, b_o, W_c, U_c, b_c$
- LSTMs output their hidden state like simple RNNs
  - Need to add a dense layer to predict labels

LSTM: how can it memorize things?

- Let's have a closer look at the gated output

$$\text{cell}_{t+1} = \text{forget}_t \odot \text{cell}_t + \text{input}_t \odot \text{cell}'_t$$
$$\text{hidden}_{t+1} = \text{output}_t \odot \tanh(\text{cell}_{t+1})$$

- Interpretation
  - if $\text{forget}_t = 1$ and $\text{input}_t = 0$: previous cell state is used
  - if $\text{forget}_t = 0$ and $\text{input}_t = 1$: previous cell state is ignored
  - if $\text{output}_t = 1$: output is set to cell state
  - if $\text{output}_t = 0$: output is set to 0

Gated recurrent units (GRU)

- Same principle but less operations / parameters (Cho et al, 2014)
  - s_t is the hidden state
  - Has to balance between update and forget
- GRU

$$\begin{aligned}
z_t =& \sigma(W_z x_t + U_z s_t + b_z) && \text{update} \\
r_t =& \sigma(W_r x_t + U_r s_t + b_r) && \text{forget} \\
h_t =& \tanh(W_h x_t + U_h (r_t \odot s_t) + b_h) && \text{input} \\
s_{t+1} =& (1 - z_t) \odot h_t + z_t \odot s_t && \text{new state} \\
\text{GRU}(s_t, x_t) =& s_{t+1}
\end{aligned}$$

- Parameters
  - $W_z, U_z, b_z, W_r, U_r, b_r, W_h, U_h, b_h$
- Interpretation
  - If $r_t = 0$, h_t does not depend on s_t
  - If $z_t = 0$, use h_t as new state
  - If $z_t = 1$, use s_t as new state

How to use RNNs

- Classification
  - Drop the prediction of y_t
  - Build hidden state
  - Use the final hidden state as representation for classification
- Language models
  - x_t is the current word
  - y_t is the next word
  - So we estimate $P(w_i|w_{i-1}, h_{i-1})$

Batches

- We saw that for training we need to unroll the RNN
  - Cannot process sequences in parallel because they have different length
- Need to introduce a padding symbol
  - Example for 3 sequences of size 3, 6 and 2:

x1	x2	x3	pad	pad	pad
y1	y2	y3	y4	y5	y6
z1	z2	pad	pad	pad	pad

- RNN cells like LSTMs have no problem learning the padding symbol

Online demo

- Deep Recurrent Nets character generation demo
  - `http://cs.stanford.edu/people/karpathy/recurrentjs/`

Conclusion

- Convolutional Neural Networks (CNN)
  - Learn to apply a filter on a moving window of the input
  - Position independent
  - Interpretable as word n-grams
  - Useful for topic classification, sentiment analysis
- Recurrent Neural Networks (RNN)
  - State depends on previous state
  - Can model varying length history
  - Potentially model the whole history
  - Useful for language models, sequence prediction