CS546: Machine Learning in NLP (Spring 2020)

http://courses.engr.illinois.edu/cs546/

Lecture 8: CNNs; Self-Attention

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Office hours: Monday, 11am—12:30pm

Today's class

Quick overview of CNNs for text classification (also FastText)

Introduction to transformers (self-attention)

Admin

Short Paper Reviews

For 10 lectures where papers are discussed, you will have to submit a review of one of the papers that was discussed in class.

—Due to the size of the class, we can largely grade you for completion (although we will spot-check your answers)
— You will have to submit the reviews through Compass.
— In the past, we've used a LaTeX template for this, but we may switch to tests inside Compass

We encourage you to get into the habit of taking notes about the papers you read. Hopefully this will get you started!

Short paper reviews

In Compass2G: three short papers reviews are out
Do them by March 15 (end of day)

Questions

What is the **topic** of this paper?

What are the main **claims** made in this paper?

What are the main **findings/results** in this paper that justify the claims?

What do you see as the main **contributions of this paper to the field?**

This is your chance to **critique** this paper. Do you agree with the claims (based on what is in this paper, or, perhaps, based on other papers you have read)? Is there anything that could have/should have been done differently/better, or added to the paper?

Is there anything in this paper that you don't understand

What is this paper's impact?

Does this paper use any **special techniques ("tricks of the trade") during training (or perhaps model definition or testing) that are or may be important?**

Paper presentations

02/21 BERT, GPT-2 etc. continued
02/26 Deep Learning for Graphs
03/04 Neural Generation
03/06 Neural Summarization
03/11 Neural Machine Translation
03/13 Neural Sequence Labeling
03/25 Neural Structure Prediction
03/27 Neural Question Answering
04/01 More on Neural Question Answering
04/03 Multimodal NLP
04/08 Neural Dialogue
04/10 More on Neural Dialogue
04/17 Intro to GANs in NLP
04/22 More on GANs in NLP
04/24 Reinforcement Learning in NLP
04/29 Fairness/Ethics in NLP
05/01 Deep Learning in Low Resource Settings

Paper presentations

Everybody needs to prepare a 15-minute oral presentation and a two-page writeup about one research paper to be shared with the class.

NB: This paper shouldn't come from your own research group, nor can it be a paper you presented in your qualifying exam.

- We will send out a sign-up sheet with dates and papers for each class.
- You will have to come to my office hours *the Monday of the week when you're presenting* with your slides to show them to me, otherwise you will only get half credit for your presentation.
- You have one week after your presentation to send in your writeup (so that you can reflect any in-class discussion)

Grading criteria for presentations

— Clarity of exposition and presentation
— Analysis (don't just regurgitate what's in the paper)
— Quality of slides (and effort that went into making them — just re-using other people's slides is not enough)

Research projects

You will have to complete a sizable research project.

Due to the size of the class, you will have to work in groups (we're aiming for 3–4 students/team).

There will be several milestones:

— Initial proposal (Feb 28)
— Intermediate report and presentation
— Final report and presentation

We have applied for accounts and GPU hours on BlueWaters for these projects.

Research projects

The aim is for each team to produce something that could be submitted to a conference:

— You should aim to make an actual contribution to research
— Your presentation should be sufficiently polished

If you build on existing research, talk to me, and loop your advisor in as well if necessary.

If you're doing related projects in other classes, let me and the other professor know.

Convolutional Neural Nets (CNNs)

CNNs for images

2D (and 3D) CNNs are standard neural models for vision tasks.

— Inspired by receptive fields in visual cortex: individual neurons respond to small regions (patches) of the visual field
— Neocognitron (Fukushima, 1980): CNN with convolutional and downsampling (pooling) layers
— In CNNs: parameter sharing among neurons within same convolutional layer, possibly followed by fully connected layers for classification purposes
— CNNs handle images of varying sizes with fixed # parameters
— Can be trained by backprop and gradient descent
— Typical CNNs combine convolution with nonlinear activations (ReLU) and (max)pooling

(2D) CNNs

An image is a 2D (width × height) matrix of pixels (e.g. RGB values)
=> it is a 3D tensor: color channels ("depth") × width × height

Each **convolutional layer** returns a 3d tensor, and is defined by:

- the **depth** (#filters) of its output
- a **filter size** (the square size of the input regions for each filter),
- a **stride** (the step size for how to slide filters across the input)
- **zero padding** (how many 0s are added around edges of input)

=> Filter size, stride, zero padding define the width/height of the output

Each unit in a convolutional layer

- receives input from a square region/patch (across w×h) in the preceding layer (across all depth channels)
- returns the **dot product** of the **input** activations and its **weights**

Within a layer, all units at the same depth use the same weights
Convolutional layers are often followed by ReLU activations
http://cs231n.github.io/convolutional-networks/

Pooling Layers

Pooling layers reduce the size of the representation, and are often used following a pair of conv+ReLU layers

Each **pooling layer** returns a 3D tensor of the same depth as its input (but with smaller height & width) and is defined by

— a **filter** size (what region gets reduced to a single value)

— a **stride** (step size for sliding the window across the input)

— a **pooling function** (**max pooling**, avg pooling, min pooling, …)

Pooling units don't have weights, but simply return the maximum/ minimum/average value of their inputs

Typically, pooling layers only receive input from a single channel. So they don't reduce the depth (#channels).

Other tricks

1x1 conv layers:

- can be used to decrease/increase #channels (depth)
- don't affect width or height of the input

Dropout to prevent overfitting:

- set values of units to 0 with probability p

1D CNNs for text

Text is a (variable-length) sequence of words (word vectors)

We can use a 1D CNN to slide a window of n tokens across:

— filter size n = 3, stride = 1, no padding

The quick brown fox jumps over the lazy dog
The **quick brown fox** jumps over the lazy dog
The quick **brown fox jumps** over the lazy dog
The quick brown **fox jumps over** the lazy dog
The quick brown fox **jumps over the** lazy dog
The quick brown fox jumps **over the lazy** dog

— filter size n = 2, stride = 2, no padding:

The quick brown fox jumps over the lazy dog
The quick **brown fox** jumps over the lazy dog
The quick brown fox **jumps over** the lazy dog
The quick brown fox jumps over **the lazy** dog

CNNs (w/ ReLU and maxpool) are used for text classification

Understanding CNNs for text classification

Jacovi et al.'18 https://www.aclweb.org/anthology/W18-5408/

— Different filters detect (suppress) different types of ngrams
— Max-pooling removes irrelevant n-grams
— Filters can produce erroneous output (abnormally high activations) on artificial input
— In a single-layer CNN with maxpooling, each filter output can be traced back to a single input ngram
— the slots (positions) in a filter are used to check whether specific types of words are present or absent in the input
— each filter can also be associated with a class it predicts

FastText

Fasttext

Library for word embeddings and text classification, based on:

Joulin et al. (2014) Bag of tricks for efficient text classification https://arxiv.org/pdf/1607.01759.pdf

Text classification model, consisting of:

— static word embeddings and ngram features
— that get averaged together in one hidden layer
— hierarchical softmax output over class labels

Bojanowski et al. (2016) Enriching word vectors with subword information https://arxiv.org/pdf/1607.04606.pdf

Skipgram model where each word is a sum of character ngram embeddings and its own embedding

Each word is deterministically mapped to ngrams

From Self-Attention to Transformers

http://peterbloem.nl/blog/transformers

Attention mechanisms

Compute a probability distribution $\alpha = (\alpha_{1t}, \ldots, \alpha_{St})$ over the *encoder's* hidden states $\mathbf{h}^{(s)}$ that depends on the *decoder's* current $\mathbf{h}^{(t)}$

$$\alpha_{ts} = \frac{\exp(s(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}))}{\sum_{s'} \exp(s(\mathbf{h}^{(t)}, \mathbf{h}^{(s')}))}$$

Compute a weighted avg. of the *encoder's* $\mathbf{h}^{(s)}$: $\mathbf{c}^{(t)} = \sum_{s=1..S} \alpha_{ts}\mathbf{h}^{(s)}$

that gets then used with $\mathbf{h}^{(t)}$, e.g. in $\mathbf{o}^{(t)} = \tanh(W_1\mathbf{h}^{(t)} + W_2\mathbf{c}^{(t)})$

- **Hard attention** (degenerate case, non-differentiable): α is a one-hot vector

- **Soft attention** (general case): α is not a one-hot
  - $s(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}) = \mathbf{h}^{(t)} \cdot \mathbf{h}^{(s)}$ is the dot product (no learned parameters)
  - $s(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}) = (\mathbf{h}^{(t)})^T W \mathbf{h}^{(s)}$ (learn a bilinear matrix W)
  - $s(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}) = \mathbf{v}^T \tanh(W_1\mathbf{h}^{(t)} + W_2\mathbf{h}^{(s)})$ concat. hidden states

Self-Attention

Attention so far (in seq2seq architectures):

In the *decoder* (which has access to the complete input sequence), compute attention weights over *encoder* positions that depend on each decoder position

Self-attention:

If the *encoder* has access to the complete input sequence, we can also compute attention weights over *encoder* positions that depend on each *encoder* position

self-attention:

encoder

For each ~~decoder~~ position t,

compute an attention weight for each *encoder* position s

renormalize these weights (that depend on t) w/ softmax

to obtain a new weighted avg. of the input sequence vectors

Self-attention

Given T k-dimensional *input* vectors $\mathbf{x}^{(1)}\ldots\mathbf{x}^{(i)}\ldots\mathbf{x}^{(T)}$,
compute T k-dimensional *output* vectors $\mathbf{y}^{(1)}\ldots\mathbf{y}^{(i)}\ldots\mathbf{y}^{(T)}$
where each $\mathbf{y}^{(i)}$ is a weighted average of the input vectors, and
where the weights w_{ij} depend on $\mathbf{y}^{(i)}$ and $\mathbf{x}^{(j)}$

$$\mathbf{y}^{(i)} = \sum_{j=1..T} w_{ij}\mathbf{x}^{(j)}$$

Computing weights w_{ij} naively:
use dot product: $w'_{ij} = \sum_k x_k^{(i)} x_k^{(j)}$

followed by softmax: $w_{ij} = \frac{\exp(w'_{ij})}{\sum_j \exp(w'_{ij})}$

Queries, keys, values

Let's add learnable parameters ($k \times k$ weight matrices),
and turn each vector $\mathbf{x}^{(i)}$ into three versions:

- **Query** vector $\mathbf{q}^{(i)} = \mathbf{W}_q \mathbf{x}^{(i)}$
- **Key** vector: $\mathbf{k}^{(i)} = \mathbf{W}_k \mathbf{x}^{(i)}$
- **Value** vector: $\mathbf{v}^{(i)} = \mathbf{W}_v \mathbf{x}^{(i)}$

The **attention weight of the *j*-th position** to compute the **new output for the *i*-th position** depends on the **query of i** and the **key of j**:

$$w_j^{(i)} = \frac{\exp(\mathbf{q}^{(i)}\mathbf{k}^{(j)})}{\sum_j \exp(\mathbf{q}^{(i)}\mathbf{k}^{(j)})} = \frac{\exp(\sum_l q_l^{(i)} k_l^{(j)})}{\sum_j \exp(\sum_l q_l^{(i)} k_l^{(j)})}$$

The **new output vector for the i-th position** depends on
the **attention weights** and **value** vectors of all **input positions j**:

$$\mathbf{y}^{(i)} = \sum_{j=1..T} w_j^{(i)} \mathbf{v}^{(j)}$$

Scaling attention weights

Value of dot product grows with vector dimension k

To scale back the dot product, divide by $\sqrt{k}$:

$$w_j^{(i)} = \frac{\exp(\mathbf{q}^{(i)}\mathbf{k}^{(j)})/\sqrt{(k)}}{\sum_j(\exp(\mathbf{q}^{(i)}\mathbf{k}^{(j)})/\sqrt{k})}$$

Multi-head attention

Just like we use multiple filters in CNNs,
we can use multiple attention heads that each have their own sets of key/value/query matrices.

Narrow self-attention:
predefine subsets of dimensions that each attention head applies to

Wide/full self-attention:
each attention head uses all dimensions.