Lecture 3

Introduction to Natural Language Processing

1. Deep Learning for NLP
2. Word Representation
3. Sequence to Sequence
4. Attention

Stanford CS224v Course
Conversational Virtual Assistants with Deep Learning

Sina Semnani, Giovanni Campagna, Monica Lam
From Lecture 1

Exciting Times for NLP Research

• 2012-2017: Basic ideas of deep learning for NLP
  Deep learning basics to transformers

• 2017-2022: Transformers to
  1.7T parameter large language models

• 2022-2027: Apple Knowledge Navigator
  Elderly’s companion
  Lots of interesting research opportunities

This Lecture
Purpose of this Lecture

“Neural network is a black box. We don’t know what it does!”

• Neural networks are not a black box
  • Defined by a few equations with a large number of parameters
  • A statistical engine, based on a massive amount of training data

This lecture

• High-level ideas: from deep-learning to transformers (used in LLMs)
  • A review for most of you
Outline

1. Deep Learning for Natural Language Processing
2. Word Representation
3. Sequence to Sequence
4. Attention
Machine Learning

• Overall goal: Find $F(x)$
  
  • Given inputs $x$ in $X$, with a probability distribution $p(x)$, for some unknown function $F(x)$
  
  • From a set of input-output pairs $\{[x_1, F(x_1)], [x_2, F(x_2)], \ldots\}$ (Training Data)
  
  • Approximate $F$ with a parametrized function $F'$ with unknown parameters $\theta$
  
  • We use training data to find $\theta$

Quiz: What are the desirable properties of training data?
Desirable Training Data Characteristics

• The answers are correct
• A representative distribution
  • How important are outliers in natural language?
  • How to generalize to unseen outliers? Composition
Learning a Parametrized Function

• The parametrized function can have various shapes:
  • Logistic Regression
  • Support Vector Machines
  • Decision Trees
  • **Neural Networks**

• Inputs and outputs can be many different things:
  • **Text**
  • Image
  • Integer
  • \( y \in \mathbb{R}^m \)
  • ...
  • **Text**
  • Image
  • **Integer**
  • \( y \in \mathbb{R}^n \)
  • ...

Stanford University
Deep Learning: FeedForward Neural Network

- Feedforward Neural Network, \( y = f(x; \theta) \) where
  \( f \) is a composition of functions of the form
  \[
  y = g(Wx + b) \quad \theta = \{W_i, b_i\}
  \]
  \( h_i \) are the intermediate result vectors in the hidden layer
  \( g \) is a non-linear activation function

Lots of matrix multiplications

Neural Science terminology
Logistic (Activation) Function Cheatsheet

Non-linear functions

- **Sigmoid**: Probability $[0,1]$ that $x$ should be classified as 1 in binary classification
  \[
  f(x) = \frac{1}{1 + e^{-x}}
  \]
  - From 0 at $-\infty$ to 1 at $+\infty$, smooth

- **Tanh**: (Hyperbolic Tangent)
  \[
  f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
  \]
  - From -1 at $-\infty$ to 1 at $+\infty$, smooth
  - Similar to sigmoid but shifted down and dilated vertically used by many RNN cells

Picture from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
Activity Function Cheatsheet (2)

- **ReLU**: Rectified Linear Unit
  - Like a binary switch: 0 for negative, 1 for positive
  \[
  \begin{cases} 
  0 & x < 0 \\
  x & x \geq 0 
  \end{cases}
  \]
  - Constant 0 in the negative semi-plane, linear in the positive
  - Differentiable *almost* everywhere (except at 0)
  - Used by Transformers
    best choice for internal layers without knowing anything else
Activity Function Cheatsheet (3)

E.g. \( \text{softmax}([1, 2, 3, 4, 1, 2, 3]) = [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175] \)

- **Softmax**: Multi-class classification \((n: \text{number of classes})\)
  - Input vector \((x_0, x_1, ..., x_{n-1})\): \(x_i\) represents the strength for choosing class \(i\)
  - A non-linear function that maps a vector of size \(n\) to another vector of size \(n\)
  - \(\text{softmax}_i(x_0, x_1, ..., x_{n-1})\): Probability of choosing the \(i\)th class
    \[
    \text{softmax}_i(x_0, x, ..., x_{n-1}) = \left[ \frac{e^{x_i}}{\sum_{j=0}^{n-1} e^{x_j}} \right] \quad \text{for } 0 \leq i < n
    \]
  - Input to softmax is log-probability (up to an offset)
  - It is differentiable
  - Why is it a probability distribution?
    - \(\text{softmax}_i\) is positive because \(e^x\) is positive
    - \(\text{softmax}_i\) is \(\leq 1\) because denominator is greater than numerator
    - The sum of all the softmax is 1 (\(\sum_{i=0}^{n-1} \text{softmax}_i = 1\))
Feedforward Neural Network

- Example: Feedforward Neural Network, $y = f(x; \theta)$ where $f$ is a composition of functions of the form $y = g(Wx + b)$, $\theta = \{W_i, b_i\}$
  - $h_i$ are the intermediate result vectors in the hidden layer
  - $g$ is a non-linear activation function

Why do we need non-linear functions?
Quiz:
Why we need nonlinear functions in lower layers?

Shape the intermediate results

**IMPORTANT**: linear functions will collapse to one linear function otherwise!
Training Neural Models

• Loss $L(\theta)$ quantifies how far is the prediction from the gold (ground truth)
• At training time, we search for $\theta$ to minimize the loss:
  $$\arg\min_{\theta} L(\theta)$$
• The parameters with min loss give the best function approximation of the data
• Typically, $L(\theta)$ is (almost everywhere) differentiable and search uses gradient descent
Loss Function and Gradient Descent

• Calculate gradient of loss with respect to parameters
• Iteratively update parameters to minimize loss
• Let $\alpha$ be the learning rate,
  \[ \nabla_\theta \] (gradient: vector of derivatives)

\[
\theta^{new} = \theta^{old} - \alpha \nabla_\theta L(\theta)
\]
Approximating Too Well?

- The more parameters you have, the better you approx. the training data
- **Overfitting**: perfect approx. of *limited* training data, no generalization

- **Solution 1: Cross-Validation**
  - During training, evaluate model on held-out *validation* set (aka dev set)
  - Pick the model that has highest validation accuracy, not training loss

- **Solution 2: Regularization**
  - Limit capacity of the model during training (but not test time)
  - *(common) dropout*: randomly set some entries of each layer output to 0
Summary

- **Deep learning**
  - Find parameters $\theta$ in an approximation function for data with a probability distribution function

- **Feed forward networks**
  - Each layer: a linear function followed by a non-linear activation function (sigmoid, tanh, relu, softmax)

- **Training**: Use gradient descent to update $\theta$ to minimize loss distribution

\[
L(\theta) = L(a),
\]

\[
g(Wx + b_0) \rightarrow g(W_2 h_1 + b_2) \rightarrow g(W_3 h_2 + b_3)
\]

- input
- $h_1$
- $h_2$
- $\hat{y}$
- $y$

loss $L(\theta)$
model prediction
gold label
Outline

1. Deep Learning for Natural Language Processing
2. Word Representation
3. Sequence to Sequence
4. Attention
Word Representation: One-Hot Vectors

• Assume: a calculus of functions $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$
• Represent words as a real vector
  • One-hot vector: $v_i$ is 1 for word $i$, 0 otherwise
  • E.g. A classification problem: detect if the sentence is about restaurants
  Answer: 1 means is a restaurant skill, 0 means everything else

$$L(\theta) = -\log p_{y=\hat{y}}$$

Show me restaurants around here
Sequence Representation: Recurrent Neural Networks RNN

- Recurrent: repeat the same box, with the same $\theta$ for each word in the sequence
- Proposed RNN cells:
  - Gated Recurrent Unit (GRU)
  - Long Short-Term Memory (LSTM)

Show me restaurants around here

Encoder: input-sentence $\rightarrow$ fixed-size vector

Classifier head (feedforward)

See [Sutskever 2014, Graves 2013] for details of LSTM details are unimportant, since this is obsolete
Encode Sequences

- It can be bi-directional

Encoder: input-sentence $\rightarrow$ fixed-size vector

Show me restaurants around here
General Definition: Encoder (Need not be RNN)

- a sequence of inputs → one or more fixed size vectors
  - one for each token
  - one for the whole sentence
Quiz

• What is the limitation of the one-hot representation?
Quiz

- What is the limitation of the one-hot representation?

Large size of input would result in inefficient computations. Words with similar meanings would have nothing in common.
Better Word Representations than 1-Hot Vectors

- Neural networks learn to map regions of the input space to outputs
- Word embeddings: similar words → similar regions makes it easier for the neural network

1-Hot
- restaurant = [1 0 0 ... 0]
- diner = [0 .. 0 1 0 .. 0]
...
Word Representation: Embedding (Dense Vectors)

• Embedding: Represent a word with
  ~100-1000 dimensional vectors (much smaller than |V|)
• Learned from large text corpora (unsupervised)

I went to this amazing restaurant last night.
We were at the diner when we saw him.
Ali went to the movies.
She was at the movies.
...

Learn embeddings \( \theta \) that maximize our ability to predict the surrounding words of a word

\[
L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=-m}^{+m} \log P(w_{t+j} | w_t ; \theta)
\]

Given word position \( t = 1, \ldots, T \),
loss: maximize joint probability (product = sum of logs) of words surrounding \( w_t \) in a window of size 2m around the current word \( w_t \)
Word Representation: Dense Vectors

*Images from GloVe: Global Vectors for Word Representation (2014)*

Stanford University
There exists a 300-dimensional vector $z$ such that adding $z$ to the vector of a city name, yields the vector of their zip codes!
Limitation of Dense Vectors

• Each word is represented by a unique vector $v$.
• $v$ has to encode all aspects and meanings of $w$.

• These two sentences will be almost identical in terms of word embeddings.
  • How much does a share of Apple cost?
  • How much does a pound of apple cost?

• We can do better by assigning a representation based on the context in the sentence (Language Model).
Language Modeling

• Learn word embeddings and a neural network for the task of estimating the probability of a sequence of words

\[ P(w_1 w_2 w_3 \ldots w_m) \]

• Auto-regressive: Predict one word at a time (given previous words)

\[ P(w_1 w_2 w_3 \ldots w_m) = \prod_{i=1}^{m} P(w_i|w_1 \ldots w_{i-1}) \approx \prod_{i=1}^{m} P(w_i|w_{i-n} \ldots w_{i-1}) \]

limiting the window to n words
Autoregressive Language Models

- Autoregressive: predict the next word based on past words

\[ P( . | \text{show}) \quad P( . | \text{show me}) \quad \ldots \]

Encoder

Show \quad me \quad restaurants \quad around \quad here
Masked Language Models (BERT [Devlin et al 2018])

- Masked: fill in the blank

$$P(\cdot \mid \text{show me } \_ \text{ around here})$$

(Bidirectional) Encoder

Show  me  _  around  here
Word Representation: Contextual

- Training data for a task is limited
- Pre-train a language model on a very large text corpus

Embeddings from Language Models:
- ELMo (Oct. 2017), corpus size 800 million words
- Generative Pre-training: GPT (June 2018), 1x
- Bidirectional Encoder Representations from Transformers: BERT (Oct. 2018), 4x
- GPT-2 (Feb. 2019), 48x
- T5 (Oct. 2019), 47x
- GPT-3 (May 2020), 35x
- ...

Transformers: to be discussed later
A language model is trained to be good at predicting missing words. How can we test if the contextual representations learned by the language model are good at capturing the meaning of sentences as well?
A language model is trained to be good at predicting missing words. How can we test if the contextual representations learned by the language model are good at capturing the meaning of sentences as well?

- By looking at the representations themselves.
- By evaluating them on downstream tasks.
  BERT for instance improved state of the art results for several NLP tasks by 4-8%.
Outline

1. Deep Learning for Natural Language Processing
2. Word Representation
3. **Sequence to Sequence**
4. Attention
When the Desired Outputs are Sentences

- Sequence-to-sequence (Seq2Seq)
- Seq2Seq has many use cases
  - Machine Translation
  - Question Generation
  - Semantic Parsing

- We will use examples from semantic parsing

  Show me restaurants around here
  
  now => @QA.Restaurant(), ,
  geo == current_location => notify
Sequence to Sequence (Seq2Seq)

• Dataset: pairs of source sentence $x_1 x_2 \ldots x_s$ and target sentence $y_1 y_2 \ldots y_t$
• For instance, pairs of natural sentences and their ThingTalk programs
• The objective is to learn $\theta$ that maximizes probability of output sequence conditioned on input sequence:

$$L(\theta) = -P(y_1 y_2 \ldots y_t \mid x_1 x_2 \ldots x_s ; \theta)$$

$$= -P(y_1 \mid x_1 x_2 \ldots x_s ; \theta) \times P(y_2 \mid y_1 x_1 x_2 \ldots x_s ; \theta) \times \ldots$$

simplifying assumption: factor $P(y_1 y_2 \ldots y_t)$ as if each word depends only on the previous ones
Encoder

- a sequence of inputs $\rightarrow$ one or more fixed size vectors
  - one for each token
  - one for the whole sentence
Decoder (the Inverse of Encoder)

- A fixed size vector → probability distributions over words
  i.e. vectors of size $|V|$ whose elements sum to 1
  $V$ is the vocabulary (set of all words)

  "@restaurant" = 0.8
  "@hotel" = 0.15
  "," = 0.1
  ...

Stanford University
Encoder-Decoder

- We can use encoder-decoder models for Seq2Seq tasks

Show me restaurants around here

now => @QA.Restaurant(), geo == ...

Encoder

Decoder
Encoder-Decoder

- In practice, we also input the previous output token to the decoder

Show me restaurants around here

<start> now => @QA.Restaurant() , ...

Encoder

Decoder
Encoder-Decoder

- At training time, decoder always gets the **gold target** as input.
Encoder-Decoder

- At inference time, we feed in the word generated by the decoder at previous time step.
- Pro: very fast to converge in practice
- Con: model is never exposed to its own errors during training
From Word Probabilities to Output Sequence

• Greedy decoding: at each step, pick the most probable word

• Greedy decoding can make errors:
  if we choose a wrong word at a step, we might never recover

• Beam Search: at each step, keep the K most probable observed outputs

• Sampling: pick a word at random according to the distribution

• ...
Downside of Word-Level Loss

**Source:** Show me restaurants around here.

**Gold target:** now => @QA.Restaurant(), geo == current_location => notify

**Model output:** now => @QA.Hospital(), geo == current_location => notify

- Most of the sentence is the same as the gold, so low loss, but you will –literally- end up in a hospital!

- A small difference in words is not the same as a small difference in meaning.
Downside of Word-Level Loss

**Source:** Show me nearby restaurants.

**Gold target:** mostrami ristoranti nelle vicinanze

**Model output:** sto cercando un ristorante qui attorno
(I’m looking for a restaurant around here)

- Most of the sentence is different from the gold, so high loss, but the answer is correct.

- Difference in words is not the same as difference in meaning.
Outline

1. Deep Learning for Natural Language Processing
2. Word Representation
3. Sequence to Sequence
4. Attention
Capturing Long Term Dependencies is Important in NL

• When generating a word, the model has to attend to words potentially far from each other.

Alice is young, lively and beautiful  →  Alice è giovane, vivace e

• Some words are more important than others

How far away is the closest Italian restaurant to me?  →  now => [ distance ] of ( compute distance ...
Encoder-Decoder with Attention

- When generating a word for the output, directly look at all the words in the input.
Dot-Product Attention

Parameters to be trained: $K, Q, V$

- At each decoding step, compute *alignment* scores by combining key $K$ (encoder, $E \times H$ sized) and query $Q$ (decoder, $1 \times H$)
  \[
  A = KQ^T \in \mathbb{R}^{E \times 1}
  \]

- Normalize scores with *softmax* to obtain a probability distribution
  \[
  S = \text{softmax}(a_0, a_1, \ldots, a_{n-1}) = \frac{e^{a_i}}{\sum_{j=0}^{n-1} e^{a_j}} \in \mathbb{R}^{E \times 1}
  \]

- Mix *attention* value $V$ (typically encoder, can vary, $E \times H$ sized) into a context vector
  \[
  C = S^T V \in \mathbb{R}^{1 \times H}
  \]

Mix the answer (context vector) with the decoder state (Add or Concatenate)

$H =$ hidden dimension
$E =$ input seq length
Transformer: Attention is All You Need!

- A class of parametrized functions
- Instead of RNNs, it is entirely made up of attentions
- Attentions are easy to compute in parallel, which is especially beneficial when using GPUs
Transformer Architecture (encoder or decoder)

input

(multihead) self-attention

addition

feed forward-network:
relu(Wx+b)

addition

batch normalization

repeat for L layers
(L = 12 … 60 depending on architecture)
GPT-2, 3, 4, …, N

- Very large transformer models with 175 billion parameters
- Trained on large datasets of Books, Wikipedia and the rest of the Web
- Trained for $3.14 \times 10^{23}$ FLOPS
- Autoregressive: To predict the next word given the previous words
- They pick up a lot of knowledge about English grammar, the world, and some logic.
- Empirically, Transformer outperforms RNN in a wide range of tasks and datasets.
## Large Language Models

<table>
<thead>
<tr>
<th>Year</th>
<th>Model name</th>
<th># parameters (Billion)</th>
<th>Language(s)</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020</td>
<td>GPT-3</td>
<td>175</td>
<td>English</td>
<td>OpenAI</td>
</tr>
<tr>
<td></td>
<td>Jurassic-1</td>
<td>178</td>
<td>English</td>
<td>AI21 Labs</td>
</tr>
<tr>
<td></td>
<td>Gopher</td>
<td>280</td>
<td>English</td>
<td>DeepMind</td>
</tr>
<tr>
<td></td>
<td>LaMDA</td>
<td>137</td>
<td>English</td>
<td>Google</td>
</tr>
<tr>
<td></td>
<td>Yuan</td>
<td>245</td>
<td>Chinese</td>
<td>Inspur</td>
</tr>
<tr>
<td></td>
<td>PanGu-α</td>
<td>200</td>
<td>Chinese</td>
<td>Huawei</td>
</tr>
<tr>
<td></td>
<td>HyperCLOVA</td>
<td>82</td>
<td>Korean</td>
<td>Naver</td>
</tr>
<tr>
<td>2021</td>
<td>OPT</td>
<td>175</td>
<td>English</td>
<td>Meta</td>
</tr>
<tr>
<td></td>
<td>PaLM</td>
<td>540</td>
<td>English</td>
<td>Google</td>
</tr>
<tr>
<td></td>
<td>BLOOM</td>
<td>176</td>
<td>46 languages</td>
<td>BigScience</td>
</tr>
<tr>
<td></td>
<td>Megatron-Turing NLG</td>
<td>530</td>
<td>English</td>
<td>Microsoft &amp; NVIDIA</td>
</tr>
<tr>
<td></td>
<td>YaLM</td>
<td>100</td>
<td>English, Russian</td>
<td>Yandex</td>
</tr>
<tr>
<td></td>
<td>Wu Dao</td>
<td>1750</td>
<td>Chinese, English,</td>
<td>Tsinghua</td>
</tr>
</tbody>
</table>
Summary

1. Deep Learning for Natural Language Processing
2. Word Representation
3. Sequence to Sequence
4. Attention

Current research directions
1. LLM training: efficiency, size, etc
2. Changing LLM characteristics
   • Changing the model (e.g. adding long term memory)
   • ”Artificial prefrontal cortex”
     • Conditioning LLMs (fine-tuning, prompting)
     • Filtering, modifying the outputs