CS 175, Project in Artificial Intelligence
Winter 2022

Lecture 4: Neural Networks for Text

Padhraic Smyth
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine
Announcements

• **Office hours:**
  – Instructor: Thursdays, 4 to 5:30
    • See online Google signup sheet (link available in Ed)
  – Sakshi (TA): Fridays 10 to 11 and Mondays 10 to 11

• **Discussion sections** with TA Sakshi: Thursday 1 to 2, 2 to 3
  – Bring your questions about Assignment 2, topics from lectures, etc

• **EdD Discussion Board**
  – Post questions about Assignment 2, possible projects, questions from lectures, etc

• **Google Sheet for student teams**
  – See post on Ed, slide in last lecture

• **Assignment 2**
  – Neural networks and Pytorch. Available now. Due Tuesday night next week.
Assignment 2: Neural Networks

• **Problem 1:**
  – Warm-up problem on using PyTorch
    (read tutorials on PyTorch in the Assignment)

• **Problem 2:**
  – Use PyTorch to train a feedforward neural network
  – Inputs = 28 x 28 pixel values for handwritten digits
  – Outputs = digit classes, 0, 1, 2, ... 9
    (concepts covered in last lecture and today’s lecture)

• **Problem 3:**
  – Use PyTorch to train a recurrent neural network (RNN)
    (RNNs discussed in today’s lecture)

Note: start working on this by the weekend at the latest and ask questions early
# Course Grading

<table>
<thead>
<tr>
<th>Activity</th>
<th>Grade Percentage</th>
<th>Date Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment 1</td>
<td>10%</td>
<td>Tonight</td>
</tr>
<tr>
<td>Assignment 2</td>
<td>10%</td>
<td>Tuesday Jan 18th</td>
</tr>
<tr>
<td>Project Proposal</td>
<td>20%</td>
<td>Tuesday Jan 24th</td>
</tr>
<tr>
<td>Progress Report</td>
<td>20%</td>
<td>Sunday Feb 20th</td>
</tr>
<tr>
<td>Weekly Logs</td>
<td>10%</td>
<td>Weeks 4 to 10</td>
</tr>
<tr>
<td>Presentation/Demo</td>
<td>5%</td>
<td>Feb 21/23</td>
</tr>
<tr>
<td>Final Report</td>
<td>25%</td>
<td>Monday Mar 14th</td>
</tr>
</tbody>
</table>
RoadMap for Lecture Topics

• **Text Representations and Pipelines**
  – Tokenization, vocabularies, bag of words, **word embeddings**

• **Machine Learning models**
  – Logistic classifiers
  – Feedforward neural networks
  – Training of neural networks
  – Language models
  – Recurrent networks
  – Sequence-sequence networks
  – Transformer networks (e.g., BERT)

• **Projects**
  – Discussion of possible project topics
  – Will discuss format of project proposals
    • How to write a project proposal
Organizing Project Teams

• **3-person teams need to be fixed by end of next week (Week 3)**
  – Project proposals from each team needed by start of Week 4

• **Use the Google sheet enter team information + to help you find team-mates**
  – Are aware that many of you might not know other students in the course
  – Individuals or teams of 2 can self-identify to find others to complete a team
  – We will work to make sure that everyone is on a team by end of Week 3

• **Other suggestions on finding team members**
  – Create a post on EdD (new category “Finding Teammates”)
  – Can post with short description of interests/skills + contact information
**CS 175 Winter 2022 Schedule**  (note: may be updated after 2\textsuperscript{nd} week)

<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Wednesday</th>
<th>Student Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 3</td>
<td>Lecture: Introduction; class projects</td>
<td>Lecture: Text Classification 1</td>
<td>Work on Assignment 1</td>
</tr>
<tr>
<td>Jan 10</td>
<td>Lecture: Text Classification 2&lt;br&gt;Assignment 1 due, Tuesday 11:59pm</td>
<td><strong>Lecture: Neural Network Models 1</strong></td>
<td>Work on Assignment 2</td>
</tr>
<tr>
<td>Jan 17</td>
<td>No class (university holiday)&lt;br&gt;Assignment 2 due, Tuesday 11:59pm</td>
<td>Lecture: Neural Network Models 2</td>
<td>Form teams; work on project proposal</td>
</tr>
<tr>
<td>Jan 24</td>
<td>Lecture: Project Topics&lt;br&gt;Project proposal due, Tuesday 11:59pm</td>
<td>Lecture: Project Topics</td>
<td>Submit proposal; Begin project</td>
</tr>
<tr>
<td>Jan 31</td>
<td>Office hours (no lecture)</td>
<td>Office hours (no lecture)</td>
<td>Work on project</td>
</tr>
<tr>
<td>Feb 7</td>
<td>Office hours (no lecture)</td>
<td>Office hours (no lecture)</td>
<td>Work on project</td>
</tr>
<tr>
<td>Feb 14</td>
<td>No class (university holiday)</td>
<td>Short lecture: Discussion of progress reports&lt;br&gt;Progress report due, Sunday 11:59pm</td>
<td>Work on project; write progress report</td>
</tr>
<tr>
<td>Feb 21</td>
<td>Project Presentations (in class)&lt;br&gt;Upload material by 11:59pm Sunday</td>
<td>Project Presentations (in class)&lt;br&gt;Upload material by 11:59pm Tuesday</td>
<td>Work on project; make short project presentation</td>
</tr>
<tr>
<td>Feb 28</td>
<td>Office hours (no lecture)</td>
<td>Office hours (no lecture)</td>
<td>Work on project</td>
</tr>
<tr>
<td>Mar 7</td>
<td>Short lecture: Discuss final reports</td>
<td>Office hours (no lecture)</td>
<td>Finish project, write final report</td>
</tr>
<tr>
<td>Mar 14</td>
<td><strong>Final project reports due Monday 9am</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Feedforward Neural Network Classifiers

(quick review of Monday’s lecture)

Reading: Chapter 7 in Jurafsky and Martin, 3rd ed, 2020
https://web.stanford.edu/~jurafsky/slp3/
Example of a 2-Layer Network (one “hidden” layer)

Output layer (produces a single number)

Hidden layer with 3 hidden units (“neurons”) (like 3 different logistic models, blue, gray, red)

5-dimensional input vector \( x \) (4 features + a constant = 1)

From Chapter 7 in Jurafsky and Martin, 3rd ed, 2022
The logistic model is equivalent to a neural network with a single hidden unit.

Mathematically, the NN is more powerful at representing input-output mappings.
Neural Networks with Multiple Outputs (number of classes > 2)

Neural Networks with Multiple Outputs (number of classes > 2)
Softmax: Real-valued Numbers to Probabilities

Let $K$ be the number of classes and let $z_k$ be the weighted sum going into the $k^{th}$ output.

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$  

Transforms each $z_k$ to be $> 0$

Normalization constant to ensure numbers sum to 1

Example:

$$z = [z_1 \ z_2 \ z_3 \ z_4] = [2.1 \ -1.4 \ 1.2 \ 0.3]$$

$$e^z = [8.17 \ 0.25 \ 3.32 \ 1.35]$$

$$\text{softmax}(z) = [0.62 \ 0.02 \ 0.25 \ 0.10]$$
Softmax: Real-valued Numbers to Probabilities

Let K be the number of classes and let $z_k$ be the weighted sum going into the $k^{th}$ output

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

Transforms each $z_k$ to be > 0

Normalization constant to ensure numbers sum to 1

Example:

$$z = [z_1 \ z_2 \ z_3 \ z_4] = [2.1 \ -1.4 \ 1.2 \ 0.3]$$

$$e^z = [8.17 \ 0.25 \ 3.32 \ 1.35]$$

$$\text{softmax}(z) = [0.62 \ 0.02 \ 0.25 \ 0.10]$$

Notes:

1. For classification problems this is the standard output for a neural network
2. This is a generalization of the sigmoid function for 2 classes
Example of a Deep Network

...a network with more than 1 hidden layer

Network architecture:
- Number of layers
- Number of hidden units at each layer
- Nonlinearities at each layer
- ..other extensions

Provides a very large space of complex/flexible function mappings

Even a deep NN is still a mathematical function $f(x)$ mapping inputs $x$ to outputs
Questions?
Training of Neural Network Classifiers

Reading: Chapter 7 in Jurafsky and Martin, 3rd ed, 2022
https://web.stanford.edu/~jurafsky/slp3/
How are Neural Networks Trained?

- **Same principle as for logistic classifiers: gradient descent on log-loss**
  - Find weights that minimize log-loss function on training data
    \[
    \text{Total Log Loss} = L(w) = - \sum_{i=1}^{N} \log P(c_i|x_i, w)
    \]
  - use gradient descent algorithm (implementation is known as “backpropagation”)

How are Neural Networks Trained?

- **Same principle as for logistic classifiers: gradient descent on log-loss**
  - Find weights that minimize log-loss function on training data
    \[
    \text{Total Log Loss} = L(w) = - \sum_{i=1}^{N} \log P(c_i|x_i, w)
    \]
  - use gradient descent algorithm (implementation is known as “backpropagation”)

- **Training algorithm**
  - Begin with random initialization of the weights
  - At each iteration
    - compute the gradient “direction” of the weights (using backpropagation)
      (gradient computation could involve millions of weights)
    - Move the weights a small amount in the negative direction of the gradient
    - Compute the loss function at the new weight values
  - Continue iterations until loss function ceases to decrease
    - (e.g., on a validation set)
Intuition for training a 2-layer Network

Actual answer $y$

System output $\hat{y}$

Loss function $L(\hat{y}, y)$

Forward pass

Backward pass

Training instance $X_1$

From Chapter 7 in Jurafsky and Martin, 3rd ed, 2022
Comments on Gradient Training

• **Computation of gradients**  
  – We can use the chain rule from calculus to “backpropagate” the information needed to compute the gradient at each step in training

• **Selection of size of learning rate $\lambda$ can be tricky**  
  – Various different heuristics are used in practice

• **Gradient training is handled automatically in libraries like PyTorch**  
  – You specify the network architecture  
  – The code automatically derives the gradients that are needed  
  – A general purpose gradient method like “Adam” is used to do gradient descent  
    • Adam takes an initial learning rate and adapts it up/down depending on progress
What is Stochastic Gradient Training?

- Computing the gradient at each iteration takes time $O(N W)$
  - $N$ documents, e.g., $N = 1$ million
  - $W$ = number of weights, can be millions
  - ..and we may have 1000’s of iterations
  - ...so this can be very slow on large data sets
What is Stochastic Gradient Training?

- **Computing the gradient at each iteration takes time $O(NW)$**
  - $N$ documents, e.g., $N = 1$ million
  - $W =$ number of weights, can be millions
  - ..and we may have 1000’s of iterations
  - ...so this can be very slow on large data sets

- **Stochastic gradient is a simple way to make this fast...**
  - Partition the data (randomly) into small “minibatches” of size $M$ (e.g., $M=100$, $M=10$)
  - Iteration
    - Compute a “stochastic” (noisy) gradient on a minibatch
    - Update the model’s weights with this noisy gradient
  - Each iteration is $O(MW)$ instead of $O(NW)$. Savings of $N/M$ per iteration
  - Typically more iterations than full gradient, but overall much faster for large $N$
  - ...and there’s some nice math with theoretical guarantees 😊

[Note: can be used with any classifier, but most useful with deep neural networks]
Stochastic Gradient Descent
(Example in 2-dimensional Parameter Space)

Empirically works very well on large data sets: some theoretical justification
(See Adam algorithm, by Kingma and Ba, ICLR, 2015)
Example: Stochastic Gradient Training with Log-Loss for a Neural Network

Graph shows different variations of gradient descent algorithm

Techniques such as Adam can vary the learning rate automatically, leading to faster convergence

Available in PyTorch (will be used in Assignment 2)

From: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
Assignment 2: Example of Output for Problem 2

```python
>>> train_and_test_simple_net(28 * 28, 200, 10)
Epoch [1/5], Step [100/600], Loss: 1.6241
Epoch [1/5], Step [200/600], Loss: 1.5617
Epoch [1/5], Step [300/600], Loss: 1.5661
Epoch [1/5], Step [400/600], Loss: 1.5843
Epoch [1/5], Step [500/600], Loss: 1.5567
Epoch [1/5], Step [600/600], Loss: 1.5612
Epoch [2/5], Step [100/600], Loss: 1.5115
Epoch [2/5], Step [200/600], Loss: 1.5115
Epoch [2/5], Step [300/600], Loss: 1.5451
Epoch [2/5], Step [400/600], Loss: 1.5409
Epoch [2/5], Step [500/600], Loss: 1.5470
Epoch [2/5], Step [600/600], Loss: 1.5170
Epoch [3/5], Step [100/600], Loss: 1.5506
Epoch [3/5], Step [200/600], Loss: 1.5222
Epoch [3/5], Step [300/600], Loss: 1.5074
Epoch [3/5], Step [400/600], Loss: 1.5366
Epoch [3/5], Step [500/600], Loss: 1.5008
Epoch [3/5], Step [600/600], Loss: 1.5151
Epoch [4/5], Step [100/600], Loss: 1.5104
Epoch [4/5], Step [200/600], Loss: 1.5270
Epoch [4/5], Step [300/600], Loss: 1.4990
Epoch [4/5], Step [400/600], Loss: 1.5113
Epoch [4/5], Step [500/600], Loss: 1.4849
Epoch [4/5], Step [600/600], Loss: 1.4893
Epoch [5/5], Step [100/600], Loss: 1.4989
Epoch [5/5], Step [200/600], Loss: 1.5120
Epoch [5/5], Step [300/600], Loss: 1.4881
Epoch [5/5], Step [400/600], Loss: 1.5039
Epoch [5/5], Step [500/600], Loss: 1.4915
Epoch [5/5], Step [600/600], Loss: 1.4711
Accuracy of the network on the 10000 test images: 96.4 %
```
Practical Issues in Training Neural Networks

• Deep neural networks typically need...
  – Large labeled data sets
  – Significant computation time
  – Careful tuning (trial and error) of model and algorithm parameters

• What are “model and algorithm parameters”
  – Network architecture: number and width of hidden layers
    • Typically involves significant trial and error search
  – Learning rates
    • May need to be different at different layers
  – Regularization methods
    • Different schemes available (L2, L1, dropout, others)
  – Activation functions, etc

• Packages such as Tensorflow, Pytorch, etc, automate much of this
  ....but some experimentation/tuning may be required
Training, Validation, and Testing

Validation data often used for tuning hyperparameters (like regularization)
Training, Validation, and Testing

Data Matrix

Training Data

Validation Data

Hold-Out (Test) Data

Held-out test data used for final comparison of models
Questions?
Language Models for Word Sequences

A model that can generate $P(\text{word } i \mid \text{history up to word } i)$ is a Language Model.

We can use a Language Model to predict how likely a word is, or to randomly generate text, or to answer simple questions.

Example 1: *The dog jumped* $<\text{WORD}>$
The Language Model can compute $P( <\text{WORD}> \mid \text{the, dog, jumped})$
for every possible value of $<\text{WORD}>$

Example 2: *The capital of Japan is* $<\text{WORD}>$
The Language Model can compute $P( <\text{WORD}> \mid \text{the, capital, of, Japan, is, })$
for every possible value of $<\text{WORD}>$ and select the highest probability word as the answer.
An Example of a Real Language Model

```python
>>> pprint(unmasker("The capital of Japan is [MASK]"))

[{'prob': 0.332,
  'sequence': 'the capital of japan is tokyo',
  'token': 5522,
  'token_str': 'tokyo'},

{'prob': 0.137,
  'sequence': 'the capital of japan is osaka',
  'token': 13000,
  'token_str': 'osaka'},

{'prob': 0.119,
  'sequence': 'the capital of japan is kyoto',
  'token': 15008,
  'token_str': 'kyoto'},

........
```

This is the output in Python of a neural language model called “distillBERT” (we will discuss neural language models in more detail later)
An Example of a Real Language Model

```python
>>> pprint(unmasker("The dog jumped [MASK]"))

[{'prob': 0.069,
  'sequence': 'the dog jumped away',
  'token': 2185,
  'token_str': 'away'},

{'prob': 0.067,
  'sequence': 'the dog jumped backwards',
  'token': 11043,
  'token_str': 'backwards'},

{'prob': 0.063,
  'sequence': 'the dog jumped backward',
  'token': 8848,
  'token_str': 'backward'},

........
```

This is the output in Python of a neural language model called “distillBERT” (we will discuss neural language models in more detail later)
Learning Language Models from Data

- **Language models can be learned from any source of text, e.g.,**
  - Web pages, Wikipedia, books, news articles, Reddit, etc
  - No human labels required: trained to predict the next word
    - “self-supervised”

- **Two main language modeling approaches**
  1. Markov models (aka ngrams)
    - Simple, easy to learn
    - ...but have limitations
  2. Neural language models
    - More complex than Markov models
    - Typically better predictions than Markov models
    - Very useful for “downstream” tasks
    - Do have some limitations (e.g., social bias)
1. Markov Chain/N-gram Language Models

Consider the sequence “the dog jumped over the…”

To predict what words might come next the Language Model needs

\[ P(\text{WORD} | \text{the, dog, jumped, over, the}) \]

This can be stored as a table of probabilities: how many entries in this table?
1. Markov Chain/N-gram Language Models

Consider the sequence “the dog jumped over the...”

To predict what words might come next the Language Model needs

\[ P( \text{WORD} | \text{the, dog, jumped, over, the} ) \]

This can be stored as a table of probabilities: how many entries in this table?

\[ P( \text{WORD} | \text{the, dog, jumped, over, the} ) \]

So the table needs \( V^6 \) entries (probability values)

Typically \( V \) is at least 10k, could be 100k, or even millions.
\( V = 100k = 10^5 \), then table has \((10^5)^6 = 10^{30}\) entries!

This creates some big problems

1. Time and memory
2. Enough data to estimate these probabilities
3. What if we have more than 5 words to condition on?
1. Markov (N-Gram) Approximations

Idea: let's approximate \( P(\text{word} \mid \text{all previous words}) \) as \( P(\text{word} \mid \text{last few words}) \)

More formally: an **N-gram** (or N-th order Markov chain) is defined by

\[
P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_1) \approx P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_{N-1})
\]

**Bigrams**

\[
P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_1) \approx P(w_i \mid w_{i-1})
\]

**Trigrams**

\[
P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_1) \approx P(w_i \mid w_{i-1}, w_{i-2})
\]
1. Markov (N-Gram) Approximations in Practice

How do we learn N-grams from data?

- Take a large corpus of text
- Tokenize and define a vocabulary
- Run a sliding window of size N over the text and increment counts
- Normalize counts to generate probabilities

Example with trigrams:

- The dog jumped over the gate and then ran across the road
- The dog jumped over the gate and then ran across the road
- The dog jumped over the gate and then ran across the road
- The dog jumped over the gate and then ran across the road
- The dog jumped over the gate and then ran across the road
-...

How well do N-gram Models work?

Reasonably well for ”local-dependencies”….but not so well for long-range dependencies

Consider the example:
*The dog jumped over the gate and then it* <WORD>

Say we have a 4-gram model:
*P(WORD | and, then, it )*
How well do N-gram Models work?

Reasonably well for ”local-dependencies”….but not so well for long-range dependencies

Consider the example:
*The dog jumped over the gate and then it* <WORD>

Say we have a 4-gram model:
\[ P(\text{WORD} \mid \text{and, then, it}) \]

Problem: words like *dog, jumped, gate* are too far back in the sequence for the N-gram

So, if the next word is a verb, the model has very little to rely on (\textit{and, then, it})
and will likely just spread probability over many verbs (e.g., *flew, spoke, drove*)
many of which don’t make sense in the context.

And going to larger N-grams, e.g., N=6, N=8, N=10, is impractical (discussed earlier)

So N-grams are fundamentally limited
….but can be useful for short range prediction (e.g., auto-complete)
….and as baselines for comparing more complex models to
Key ideas:
• learn “embeddings” of words (so the model can generalize to combinations it did not see)
• use a more flexible representation of memory than N-grams

Example:

```python
>>> print(text_generator("The dog jumped over the gate and", max_length=11, do_sample=False))
[{'generated_text': 'The dog jumped over the gate and ran into the woods'}]
```

This is the output in Python of a neural language model called GPT2 being used in text generation mode (we will discuss neural language models in more detail later)
Simulating/Generating Words from a Language Model

Say we have already seen the partial sequence “The dog jumped ... “

Procedure

Step 1:
Use the language model to compute $P(\text{word } i | \text{ the, dog, jumped})$
Say for example the word “over” has the highest probability and we select it
Simulating/Generating Words from a Language Model

Say we have already seen the partial sequence “The dog jumped ... “

Procedure

Step 1:
Use the language model to compute $P(\text{word } i \mid \text{the, dog, jumped})$
Say for example the word “over” has the highest probability and we select it

Step 2:
Now our sequence is “The dog jumped over...”
Now use the language model to compute $P(\text{word } i \mid \text{the, dog, jumped, over})$
Say for example the word “the” now has the highest probability and we select it

...and so on for $K$ steps until (e.g.,) we sample the token EOS (end of sentence).

Note: at each step we could pick the word with the highest probability. But this has the problem that we may get a lot of repetition of common words. So instead its common to sample from the conditional probability distribution at each step. But even this is not ideal. We will return to this later.
Example of Simulating/Generating Multiple Words

Figure from https://huggingface.co/blog/how-to-generate
Hot off the press!  
(December 2021)

Current state of the art in language models

280 billion weights!

Trained on 300 billion tokens from Web pages, Wikipedia, books, etc

Scaling Language Models: Methods, Analysis & Insights from Training Gopher


Language modelling provides a step towards intelligent communication systems by harnessing large repositories of written human knowledge to better predict and understand the world. In this paper, we present an analysis of Transformer-based language model performance across a wide range of model scales — from models with tens of millions of parameters up to a 280 billion parameter model called Gopher. These models are evaluated on 152 diverse tasks, achieving state-of-the-art performance across the majority. Gains from scale are largest in areas such as reading comprehension, fact-checking, and the identification of toxic language, but logical and mathematical reasoning see less benefit. We provide a holistic analysis of the training dataset and model's behaviour, covering the intersection of model scale with bias and toxicity. Finally we discuss the application of language models to AI safety and the mitigation of downstream harms.

Keywords: Natural Language Processing, Language Models, Deep Learning
Good responses from Gopher

User: “What is the technical name for single-cell organisms?”
Gopher: “They’re known as prokaryotes.”

User: “Are humans prokaryotes?”
Gopher: “No, they’re eukaryotes.”

User: “What’s an example of a prokaryote?”
Gopher: “Well, bacteria and archaea are prokaryotes.”
Good responses from Gopher

User: “What is the technical name for single-cell organisms?”
Gopher: “They’re known as prokaryotes.”

User: “Are humans prokaryotes?”
Gopher: “No, they’re eukaryotes.”

User: “What’s an example of a prokaryote?”
Gopher: “Well, bacteria and archaea are prokaryotes.”

…but it can also make mistakes….and be over-confident about them

User: “Can you confidently tell me who won the Women’s US Open in 2021?”
Gopher: “Sure! It was Naomi Osaka.”

User: “Are there any French-speaking countries in South America?”
Gopher: “No.”

User: “Do you have any uncertainty about your answers so far?”
Gopher: “No.”
Questions?
Recurrent Neural Networks

Bag of Words as Input

Inputs could also be “bag of words”

One input per term in the vocabulary

Very large weight matrix
(number of terms by number of hidden units)
Learning to Predict the Next Word given Current Word

Key idea: treat prediction of next word as a classification problem
Predicting Next Word with Neural Networks

- "water" → Hidden Layer → Softmax → Predictions
  - Probabilities of next word
  - "flows"

- "flows" → Hidden Layer → Softmax → Predictions
  - Probabilities of next word
  - "under"

- "under" → Hidden Layer → Softmax → Predictions
  - Probabilities of next word
  - "the"

- "the" → Hidden Layer → Softmax → Predictions
  - Probabilities of next word
  - "bridge"
Better Approach: Recurrent Neural Network

Predictions
Probabilities of next word

Class Labels (next word)

“flows”

“under”

“the”

“bridge”
Weights in a Recurrent Neural Network

Treat as one large network
W, U, V are weight matrices we learn

"water"

Hidden Layer → Softmax

"flows"

Hidden Layer → Softmax

"under"

Hidden Layer → Softmax

"the"

Hidden Layer → Softmax

Predictions

Class Labels
(next word)

"flows"

"under"

"the"

"bridge"
Generating/Simulating Text (Word Sequences) with RNNs

From Chapter 9 in Jurafsky and Martin, 3rd ed, 2022
Completing a Phrase with a Simple RNN

From Jurafsky and Martin, 3rd edition, 2020
Questions?
Character-level Recurrent Networks

"t" W → Hidden Layer V → Softmax

"h" W → Hidden Layer V → Softmax

"e" W → Hidden Layer V → Softmax

Predictions
Probabilities of next word

Class Labels
(next character)

“h”

“e”

“<SP>”
Simulating Character-Level Sequences from an RNN

Figures from http://cs231n.stanford.edu/slides/2017/
Simulating Character-Level Sequences from an RNN

Figures from http://cs231n.stanford.edu/slides/2017/
Simulating Character-Level Sequences from an RNN

Figures from http://cs231n.stanford.edu/slides/2017/
KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO: Well, your wit is in the care of side and that.

Examples from “The Unreasonable Effectiveness of Recurrent Neural Networks”,
Andrej Karpathy, blog, http://karpathy.github.io/2015/05/21/rnn-effectiveness/
One Stage of a Recurrent Network

\[ h_t = g(U h_{t-1} + W x_t) \]
\[ y_t = f(V h_t) \]

From Chapter 9 in Jurafsky and Martin, 3rd ed, 2022
“Unrolling” a Recurrent Network

From Chapter 9 in Jurafsky and Martin, 3rd ed, 2022
Interpretation of Hidden Layer

The hidden layer in an RNN is keeping track of the “state” of the sequence

In some sense, it’s the model’s summary of the “word history”

So …. the final hidden state is in effect a summary of the whole sequence

Given this, we can use the hidden states as “features” to classify the whole sequence

….. and this broadens the scope of what we can use RNNs for
Sequence Classification with an RNN

"food"  →  Hidden Layer  →  Class Probabilities  "negative"

"was"  →  Hidden Layer  →  Class Probabilities

"horrible"  →  Hidden Layer  →  Class Probabilities
Questions?
Assignment 2

• **Problem 3: Recurrent Neural Networks**
  – Input = family names as sequences of characters (e.g., “Jackson”, “Kim”, etc)
  – Class labels are names of countries (Chinese, Korean, Scottish)
  – You will build an RNN to predict probabilities of what country a name is from
  – The model will learn to generalize to new names (unlike a dictionary)
Assignment 2

• Problem 3: Recurrent Neural Networks
  – Input = family names as sequences of characters (e.g., “Jackson”, “Kim”, etc)
  – Class labels are names of countries (Chinese, Korean, Scottish)
  – You will build an RNN to predict probabilities of what country a name is from
  – The model will learn to generalize to new names (unlike a dictionary)
Assignment 2

• Problem 3: Recurrent Neural Networks
  – Input = family names as sequences of characters (e.g., “Jackson”, “Kim”, etc)
  – Class labels are names of countries (Chinese, Korean, Scottish)
  – You will build an RNN to predict probabilities of what country a name is from
  – The model will learn to generalize to new names (unlike a dictionary)

• More details on RNN in Problem 3
  – RNN input: sequence of characters for a name
  – RNN output: probability distribution over 18 classes (countries)
  – RNN Training: find the weights (for W, U) that minimize the log-loss
  – Standard training algorithm used stochastic gradient descent
    • Gradients are computed via “backpropagation” from output “through” RNN
    • A minibatch size of M=1 will be used (single example at a time)
Log-Loss over Iterations for Recurrent Network (Problem 3)

Note: your numbers may be different, but shape of curve will be similar

From https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html
Examples of Predictions from the trained RNN

> python predict.py Hinton  
(-0.47) Scottish  
(-1.52) English  
(-3.57) Irish

> python predict.py Schmidhuber  
(-0.19) German  
(-2.48) Czech  
(-2.68) Dutch

> python predict.py Satoshi  
(-0.46) Japanese  
(-2.12) Arabic  
(-2.31) Polish

Note: here the output is in the form of ”scores” (what we referred to as z in lecture 3), before they are fed into softmax. After softmax is performed on these scores they are transformed to probabilities taking values between 0 and 1

From https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html
Assignment 2: Examples of Predictions for Problem 3

```python
>>> predict("Kim")
Prediction for Kim:
Korean Probability: (54.51), Score: (-0.66)
Chinese Probability: (23.09), Score: (-1.51)
Arabic Probability: (7.84), Score: (-2.59)
Vietnamese Probability: (6.48), Score: (-2.79)
Russian Probability: (3.72), Score: (-3.34)
Czech Probability: (1.95), Score: (-3.98)
English Probability: (1.36), Score: (-4.35)
German Probability: (1.06), Score: (-4.60)
>>> 
```  
```python
>>> predict("MacDonald")
Prediction for MacDonald:
Scottish Probability: (39.89), Score: (-0.96)
Irish Probability: (30.78), Score: (-1.22)
English Probability: (8.78), Score: (-2.47)
French Probability: (6.22), Score: (-2.82)
Italian Probability: (5.80), Score: (-2.89)
Spanish Probability: (4.38), Score: (-3.17)
Portuguese Probability: (2.25), Score: (-3.84)
German Probability: (1.92), Score: (-4.00)
>>> 
```  
```python
>>> predict("Papodopoulos")
Prediction for Papodopoulos:
Greek Probability: (98.03), Score: (-0.02)
Russian Probability: (0.56), Score: (-5.20)
English Probability: (0.41), Score: (-5.49)
Irish Probability: (0.29), Score: (-5.84)
Dutch Probability: (0.23), Score: (-6.07)
Polish Probability: (0.19), Score: (-6.28)
Spanish Probability: (0.17), Score: (-6.41)
Scottish Probability: (0.13), Score: (-6.68)
```
RNNs + FeedForward Networks for Sequence Classification

Figure 9.8  Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

From Chapter 9 in Jurafsky and Martin, 3rd ed, 2022
How are RNNs trained?

• ...very similar to the way feedforward networks are trained

• **Objective function is Log Loss as before**
  – Negative log probability of next word or sequence label given the words that came before as input
  – Summed over all words or sequence labels in the training data

• **Backpropagation for updating weights?**
  – Network is “unrolled” sequentially, per sequence, to create one “long network”
  – Gradient calculations are now a little more complex
  – But the general principle is the same
    • Also use stochastic gradient/minibatch, learning rate $\lambda$, etc

• **Languages like PyTorch simplify training RNN models**
  – Many details are handled “behind the scenes” (see Prob 3 in Assignment 2)
Backpropagation of Errors during Training an RNN

From Chapter 9 in Jurafsky and Martin, 3rd ed, 2022
Questions?
Part of Speech Tagging with an RNN

From Jurafsky and Martin, 3rd edition, 2020
Stacked RNNs

From Jurafsky and Martin, 3rd edition, 2020
Bidirectional RNNs

From Jurafsky and Martin, 3rd edition, 2020
Bidirectional RNN for Sequence Classification

From Jurafsky and Martin, 3rd edition, 2020
Generative Responses, e.g., with Encoder-Decoders

(also known as sequence-sequence or Seq2Seq models)

From Jurafsky and Martin, 3rd edition, 2022
Topics we did not cover (yet) ..... 

- Different types of RNN units (LSTMs, GRUs)
- Word embeddings
- Attention mechanisms
- Transformer models
Questions?
Suggestions for Project Topics
General Tips for Finding Project Ideas

- **Relevant links on course Web page**
  
  https://www.ics.uci.edu/~smyth/courses/cs175/project_reading.html

- **Read chapters in Jurafsky and Martin text**
  
  https://web.stanford.edu/~jurafsky/slp3/

- **Look at research datasets and associated benchmark problems**
  
  http://nlpprogress.com/
  
  https://paperswithcode.com/area/natural-language-processing
  
  https://www.ics.uci.edu/~smyth/courses/cs175/text_data_sets.html

- **Google Scholar search on topics of interest, e.g., [twitter sentiment]**

- **Review of past project reports**
  
  (examples will be posted on Ed.....soon)
Potential Ideas for Projects

• **Predicting Sentiment from Text**
  – Product/restaurant reviews
  – Would need to go significantly beyond Assignment 1
  – Detecting fake reviews

• **Simulating Text**
  – Generating a news article given a summary of the article
  – Generating new text for a particular author
  – Generating a response sentence in a dialog

• **Analyzing Bias in Large Language Models (BERT)**
  – Gender, ethnicity, etc

• **Toxic comment detection in blogs, tweets**

• **Analyzing human-human conversations (e.g., detecting emotion)**
Language Modeling Project: Comparing N-grams and RNNs

- **Task:** predict the most likely next word in text
  - Reading: Chapters 3 and 7 in Jurafsky and Martin

- **Technical approaches**
  - Baseline: Ngrams, Advanced: RNNs/Transformers

- **Datasets:**
  - Any large corpus, e.g., all or part of Wikipedia

- **Evaluation**
  - Rank words by probability of being next: higher rank for actual word is better
  - Log probability of next word given preceding words (or perplexity)

- **Enhancements**
  - Doing transfer learning on a domain-specific corpus
Question-Answering System

• Task
  – Generate an answer from within a document to specific types of questions
  – Reading: Chapter 23 in Jurafsky and Martin

• Technical approaches
  – RNNs, Encoder-decoder architectures

• Datasets
  – Multiple well-known Q/A data sets
    • e.g, see https://paperswithcode.com/task/question-answering

• Evaluation
  – Accuracy relative to known answers

• Extensions
  – Can models “know what they don’t know”? 
  – e.g., see this research paper: https://aclanthology.org/2021.findings-emnlp.385/
Automated Summarization of a Set of Documents

• Task
  – Automatically summarize documents in a corpus, e.g., reviews of restaurants
  – Reading, e.g.,
    • [link to MITRE publication]
    • See others under Text Summarization on Course Website

• Technical approaches
  – Various summarization algorithms

• Datasets
  – Reviews, news articles, scientific papers: any large set of documents with common topics

• Evaluation
  – ROUGE method; also human user studies

• Extensions
  – Generate both graphical and text representations of the summary
Analysis of Aspects of Positive/Negative Reviews

• **Task**
  – Using labeled review data:
    • learn a classifier to predict positive/negative sentiment from the review
    • Automatically extract from reviews what aspects of a product/movie/restaurant are the basis for the positive or negative reviews
  – **Reading: parts of chapters 5 to 9, 11, 17, 20 in Jurafsky and Martin**

• **Technical approaches**
  – Part-of-speech parsing
  – Sentiment classification (logistic, feed-forward, RNNs, etc)

• **Datasets**
  – Yelp reviews, Amazon reviews, Movie reviews

• **Evaluation**
  – Accuracy for classification part; human user study for aspects part
Chatbot Project

• **Task**
  – Automatically generate the next (response) utterance in a conversation
  – Reading: Chapter 7, 9, 11, 24 in Jurafsky and Martin

• **Technical approaches**
  – Information retrieval methods
  – Neural architectures (encoder-decoder, transformers)

• **Datasets**
  – Switchboard, movie scripts, + several others
  – See

• **Evaluation, e.g., system A versus B**
  – Human user study (a type of Turing test: which responses are more real?)

• **Enhancements**
  – Build a domain-specific chatbot
Chatbots

- **Chatbot Setup**
  - Human and agent alternate utterances (written or spoken)
  - Goal of agent is to generate an appropriate response utterance at each turn
    - conditioned on the history of the dialog up to that point

- **Variations**
  - Single-turn
    - Respond only to the last utterance
    - Will not be able to maintain coherence well (i.e., will “forget” earlier information)
  - Multiturn
    - Takes as input the previous K utterances (K is usually small)
Approaches to Chatbot Systems

- **Rule-based**
  - Early work in 1960’s, still widely-used today
  - Based on manually-defined pattern-matching + rules and slot-filling

- **Corpus-based**
  - Builds models for large corpora. Two general types
    - **Retrieval-based**: finds best-matching response utterance from a corpus
    - **Generative models**: generate a new response utterance
      - generator learned via machine learning, e.g., via an encoder-decoder RNN

- **Real-world chatbots often use a combination of techniques, e.g., Alexa**
Information Retrieval Response Generation

- Given a large corpus with utterances \( t \), and given a user’s sentence \( q \)
  - Define similarity \( \text{sim}(t, q) \)
  - e.g., cosine similarity over bag-of-words/tf-idf or over sentence embeddings

- Two typical methods
  1. Maximize \( \text{sim}(t, q) \) and return utterance/response that came after \( t \)
  2. Maximize \( \text{sim}(t, q) \) and return \( t \)

- Many possible extensions....
  - Let \( q \) be the whole dialog so far
  - Query reformulation for \( q \) (for questions)

- Corpus C can be dialog turns or even non-dialog sentences
  - Transcripts of actual conversations, movie scripts
  - Wikipeida articles, news articles
Generative Responses, e.g., with Encoder-Decoders

From Jurafsky and Martin, 3rd edition, 2020
Generative Responses, e.g., with Encoder-Decoders

- Issues with basic encoder-decoder architectures
  - Tendency to produce predictable/short responses, e.g., “ok” or “I don’t know”
  - Inability to model longer context of conversation
  - Coherence of responses across multiple turns
  - See Jurafsky and Martin for suggestions on how to handle these issues
  - Better performance with more recent neural models, Transformers (future lecture)

From Jurafsky and Martin, 3rd edition, 2020
Enhancements (applicable to multiple projects)

- **Compare text representations**
  - Different types of word embeddings (BERT, word2vec) to BOW

- **Compare character embeddings to word embeddings**

- **Systematically investigate effect of document length on classification accuracy**

- **Add a speech recognition front-end**

- **Evaluate domain transfer**
  - Train a model on one domain, evaluate on a different domain, perhaps with some transfer learning
    - How well does a sentiment classifier for reviews “transfer” to tweets?
Questions?
Wrapup
RoadMap for Lecture Topics

• **Text Representations and Pipelines**
  – Tokenization, vocabularies, bag of words, *word embeddings*

• **Machine Learning models**
  – Logistic classifiers
  – Feedforward neural networks
  – Training of neural networks
  – Language models
  – Recurrent networks
  – Sequence-sequence networks
  – Transformer networks (e.g., BERT)

• **Projects**
  – Discussion of possible project topics
  – Will discuss format of project proposals
    • How to write a project proposal
Announcements

• **Office hours:**
  – Instructor: Thursdays, 4 to 5:30
  – See online Google signup sheet (link available in Ed)
  – Sakshi (TA): Fridays 10 to 11 and Mondays 10 to 11

• **Discussion sections** with TA Sakshi: Thursday 1 to 2, 2 to 3
  – Bring your questions about Assignment 2, topics from lectures, etc

• **EdD Discussion Board**
  – Post questions about Assignment 2, possible projects, questions from lectures, etc

• **Google Sheet for student teams**
  – See post on Ed, slide in last lecture

• **Assignment 2**
  – Neural networks and Pytorch. Available now. Due Tuesday night next week.
# CS 175 Winter 2022 Schedule

(Notes: may be updated after 2nd week)

<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Wednesday</th>
<th>Student Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan 3</td>
<td>Lecture: Introduction; class projects</td>
<td>Lecture: Text Classification 1</td>
<td>Work on Assignment 1</td>
</tr>
<tr>
<td>Jan 10</td>
<td>Lecture: Text Classification 2</td>
<td>Lecture: Neural Network Models 1</td>
<td>Work on Assignment 2</td>
</tr>
<tr>
<td></td>
<td>Assignment 1 due, Tuesday 11:59pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan 17</td>
<td>No class (university holiday)</td>
<td>Lecture: Neural Network Models 2</td>
<td>Form teams; work on project proposal</td>
</tr>
<tr>
<td></td>
<td>Assignment 2 due, Tuesday 11:59pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan 24</td>
<td>Lecture: Project Topics</td>
<td>Lecture: Project Topics</td>
<td>Submit proposal; Begin project</td>
</tr>
<tr>
<td></td>
<td>Project proposal due, Tuesday 11:59pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan 31</td>
<td>Office hours (no lecture)</td>
<td>Office hours (no lecture)</td>
<td>Work on project</td>
</tr>
<tr>
<td>Feb 7</td>
<td>Office hours (no lecture)</td>
<td>Office hours (no lecture)</td>
<td>Work on project</td>
</tr>
<tr>
<td>Feb 14</td>
<td>No class (university holiday)</td>
<td>Short lecture: Discussion of progress reports</td>
<td>Work on project; write progress report</td>
</tr>
<tr>
<td></td>
<td>Progress report due, Sunday 11:59pm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb 21</td>
<td>Project Presentations (in class)</td>
<td>Project Presentations (in class)</td>
<td>Work on project; make short project presentation</td>
</tr>
<tr>
<td></td>
<td>Upload material by 11:59pm Sunday</td>
<td>Upload material by 11:59pm Tuesday</td>
<td></td>
</tr>
<tr>
<td>Feb 28</td>
<td>Office hours (no lecture)</td>
<td>Office hours (no lecture)</td>
<td>Work on project</td>
</tr>
<tr>
<td>Mar 7</td>
<td>Short lecture: Discuss final reports</td>
<td>Office hours (no lecture)</td>
<td>Finish project, write final report</td>
</tr>
<tr>
<td>Mar 14</td>
<td>Final project reports due Monday 9am</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questions? (with video recording off)