Unstructured Data Analysis

Lecture 14: Time series analysis with recurrent neural nets; some other deep learning topics; course wrap-up

George Chen
Last Lecture!

- More on deep learning:
  - Time series analysis with recurrent neural nets
    - The demo is shifted to recitation
  - Extremely important concept we use: “word embeddings”
    - High-level idea shifted to recitation
  - I’ll also talk about some other deep learning topics
    - Roughly how learning a neural net works
    - How to deal with small datasets
    - Generating fake data that look real
    - AI agents that interact with environments
- I’ll end with a course wrap-up
Sequence Data

What we’ve seen so far are “feedforward” NNs
Sequence Data

What we’ve seen so far are “feedforward” NNs

What if we had a video?
Recurrent Neural Nets

Feedforward NN’s:
treat each video frame separately
Recurrent Neural Nets

Feedforward NN's: treat each video frame separately

RNNs: feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options: RNN (vanilla), LSTM, GRU
Recurrent Neural Nets

Feedforward NN’s: treat each video frame separately

RNNs: feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options: RNN (vanilla), LSTM, GRU
Vanilla ReLU RNN

memory that evolves over time; we want to learn how it changes

```python
current_state = np.zeros(num_nodes)
for input in input_sequence:
    linear = np.dot(input, W) + np.dot(current_state, U) + b
    output = np.maximum(0, linear)  # ReLU
    current_state = output
```

Parameters: weight matrices $W$ & $U$, and bias vector $b$

Key idea: it's like a linear layer in a for loop that tracks how memory changes over time
Recurrent Neural Nets (RNNs) are like a linear layer that has memory and readily chains together with other neural net layers. In contrast, Feedforward Neural Networks (FFNNs) treat each video frame separately.

RNNs feed output at the previous time step as input to the RNN layer at the current time step. In PyTorch, different RNN options are available: RNN (vanilla), LSTM, and GRU.

Time series → RNN layer (like a linear layer that has memory) → does not incorporate image structure!!!
Recurrent Neural Nets

readily chains together with other neural net layers

Feedforward NN’s: treat each video frame separately

RNNs: feed output at previous time step as input to RNN layer at current time step

In PyTorch, different RNN options: RNN (vanilla), LSTM, GRU

Time series

RNN layer
like a linear layer that has memory
does not incorporate image structure!!!
Recurrent Neural Nets

- RNN layer readily chains together with other neural net layers.
- RNNs: feed output at previous time step as input to RNN layer at current time step.
- In PyTorch, different RNN options: RNN (vanilla), LSTM, GRU.

Feedforward NN’s: treat each video frame separately.

Use CNN to incorporate image structure!
Intuition: CNNs Encode Semantic Structure for Images
Intuition: CNNs Encode Semantic Structure for Images

final output for different input 6’s is similar
actually, intermediate representations close to the last layer are also similar!

(intuition: recall the crumpled paper analogy!)
Recurrent Neural Nets

- RNN layer readily chains together with other neural net layers.
- CNN: treat each video frame separately.
- RNNs: feed output at previous time step as input to RNN layer at current time step.
- In PyTorch, different RNN options: RNN (vanilla), LSTM, GRU.

Use CNN to incorporate image structure! Does not incorporate image structure!!!
Recurrent Neural Nets

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

Common first step for text: turn words into vector representations that are semantically meaningful
(Flashback) Do Data Actually Live on Manifolds?

Recurrent Neural Nets

Example: Given text (e.g., movie review, Tweet), figure out whether it has positive or negative sentiment (binary classification)

Common first step for text: turn words into vector representations that are semantically meaningful

In PyTorch, use the Embedding layer

Linear layer (2 nodes), Softmax activation
Sentiment Analysis with IMDb Reviews

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>2D Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[-0.57, 0.44]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0.38, 0.15]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[-0.85, 0.70]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[-0.26, 0.66]</td>
</tr>
</tbody>
</table>

Training reviews

Ordering of words matters
Different reviews can have different lengths

Step 2: Encode each review as a sequence of word indices into the vocab

“this movie rocks” → 0 1 2
“this movie sucks” → 0 1 3
“this sucks” → 0 3

Step 3: Use word embeddings to represent each word
Sentiment Analysis with IMDb Reviews

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>2D Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[-0.57, 0.44]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0.38, 0.15]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[-0.85, 0.70]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[-0.26, 0.66]</td>
</tr>
</tbody>
</table>

Step 2: Encode each review as a sequence of word indices into the vocab

“this movie sucks” → 0 1 3

Step 3: Use word embeddings to represent each word

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]

Training reviews
Sentiment Analysis with IMDb Reviews

“this movie sucks”

0 1 3

Embedding

[-0.57, 0.44]
[0.38, 0.15]
[-0.26, 0.66]
Sentiment Analysis with IMDb Reviews

“this movie sucks”

Embedding

0 → [-0.57, 0.44]

1 → [0.38, 0.15]

3 → [-0.26, 0.66]
Sentiment Analysis with IMDb Reviews

0 → Embedding → [-0.57, 0.44] → Logistic Regression

1 → Embedding → [0.38, 0.15] →

3 → Embedding → [-0.26, 0.66] →

"this movie sucks"
Sentiment Analysis with IMDb Reviews

RNN’s work with variable-length inputs

Note: Often in text analysis, the word embeddings are treated as fixed, so we do not update them during training
What if we didn’t use word embeddings?
Sentiment Analysis with IMDb Reviews

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>2D Embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[-0.57, 0.44]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0.38, 0.15]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[-0.85, 0.70]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[-0.26, 0.66]</td>
</tr>
</tbody>
</table>

Step 2: Encode each review as a sequence of word indices into the vocab

“this movie sucks” → 0 1 3

Step 3: Use word embeddings to represent each word

[-0.57, 0.44]  
[0.38, 0.15]  
[-0.26, 0.66]
Bad Strategy: One-Hot Encoding

Training reviews

Step 1: Tokenize & build vocabulary

<table>
<thead>
<tr>
<th>Word index</th>
<th>Word</th>
<th>One-hot encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>this</td>
<td>[1, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>1</td>
<td>movie</td>
<td>[0, 1, 0, 0, 0]</td>
</tr>
<tr>
<td>2</td>
<td>rocks</td>
<td>[0, 0, 1, 0, 0]</td>
</tr>
<tr>
<td>3</td>
<td>sucks</td>
<td>[0, 0, 0, 1, 0]</td>
</tr>
</tbody>
</table>

Step 2: Encode each review as a sequence of word indices into the vocab

“this movie sucks” → 0 1 3

Step 3: Use one-hot encoding to represent each word

This strategy tends to work poorly in practice: distance between every pair of words is the same in one-hot encoding!

[1, 0, 0, 0]  
[0, 1, 0, 0]  
[0, 0, 0, 1]  

"this movie sucks"
Recap/Important Reminder

- Neural nets are *not* doing magic; **incorporating structure is very important to state-of-the-art deep learning systems**

- Word embeddings encode semantic structure—words with similar meaning are mapped to nearby Euclidean points

- CNNs encode semantic structure for images—images that are “similar” are mapped to nearby Euclidean points

- An RNN tracks how what’s stored in memory changes over time — **an RNN’s job is made easier if the memory is a semantically meaningful representation**
Sentiment Analysis with IMDb Reviews

Demo will be in your recitation
A special kind of RNN: an “LSTM”
(Flashback) Vanilla ReLU RNN

```python
current_state = np.zeros(num_nodes)

for input in input_sequence:
    linear = np.dot(input, W) \
    + np.dot(current_state, U) \
    + b

output = np.maximum(0, linear) # ReLU

current_state = output
```

Parameters: weight matrices $W$ & $U$, and bias vector $b$

Key idea: it's like a linear layer in a for loop that tracks how memory changes over time
(Flashback) Vanilla ReLU RNN

current_state = np.zeros(num_nodes)

outputs = []

for input in input_sequence:
    linear = np.dot(input, W) \ + np.dot(current_state, U) \ + b

    output = np.maximum(0, linear)  # ReLU

    outputs.append(output)

    current_state = output
Time series → RNN layer → output prediction
Time $t - 1$

Time $t$

Time $t + 1$

Vanilla RNN tends to forget things quickly

```
outputs[t] = np.maximum(np.dot(input_sequence[t], W) + np.dot(outputs[t-1], U) + b, 0)
```
Time $t - 1$

Time $t$

Time $t + 1$

Long-term memory

Add explicit long-term memory!

But need some way to update long-term memory!

output $t - 1$

output $t$

output $t + 1$
Add explicit long-term memory!

But need some way to update long-term memory!
Time $t - 1$ output $t - 1$

Add explicit long-term memory!

But need some way to update long-term memory!

Time $t$ output $t$
Add explicit long-term memory!

But need some way to update long-term memory!

Called a “long short-term memory” (LSTM) RNN

Remembers things longer than vanilla RNN
Analyzing Times Series with CNNs

• Think about an image with 1 column, and where the rows index time steps: this is a time series!

• Think about a 2D image where rows index time steps, and the columns index features: this is a multivariate time series (feature vector that changes over time!)

• CNNs can be used to analyze time series but inherently the size of the filters used say how far back in time we look

• If your time series does not have long-range dependencies that require long-term memory, CNNs can do well already!

• If you need long-term memory, use RNNs
Some Other Deep Learning Topics
Learning a Deep Net

Suppose the neural network has a single real number parameter $w$

The skier wants to get to the lowest point
The skier should move rightward (positive direction)
The derivative $\frac{\Delta L}{\Delta w}$ at the skier’s position is negative

In general: the skier should move in opposite direction of derivative

In higher dimensions, this is called gradient descent
(derivative in higher dimensions: gradient)
Learning a Deep Net

Suppose the neural network has a single real number parameter $w$
Learning a Deep Net

Suppose the neural network has a single real number parameter $w$
Learning a Deep Net

Suppose the neural network has a single real number parameter $w$
Learning a Deep Net

Suppose the neural network has a single real number parameter $w$

In general: not obvious what error landscape looks like!

→ we wouldn’t know there’s a better solution beyond the hill

Popular optimizers (e.g., RMSprop, Adam, LookAhead, RAdam) are variants of gradient descent

In practice: local minimum often good enough
Learning a Deep Net

2D example

$L(w)$
Remark: In practice, deep nets often have > millions of parameters, so very high-dimensional gradient descent
Automatic differentiation is crucial in learning deep nets!

Careful derivative chain rule calculation: back-propagation
Gradient Descent

We have to compute lots of gradients to help the skier know where to go!

Computing gradients using all the training data seems really expensive!

Loss 1
Loss 2
Loss 3
Loss 4
Loss 5
... Loss n

Average loss

Compute gradient and move skier
Stochastic Gradient Descent (SGD)

SGD: compute gradient using only 1 training example at a time
(can think of this gradient as a noisy approximation of the “full” gradient)
Stochastic Gradient Descent (SGD)

Training example 1
loss 1
compute gradient and move skier

Training example 2
loss 2

Training example 3
loss 3

Training example 4
loss 4

Training example 5
loss 5

... 

Training example n
loss n

SGD: compute gradient using only 1 training example at a time (can think of this gradient as a noisy approximation of the “full” gradient)
Stochastic Gradient Descent (SGD)

SGD: compute gradient using only 1 training example at a time (can think of this gradient as a noisy approximation of the “full” gradient)
Stochastic Gradient Descent (SGD)

Training example 1

loss 1

Training example 2

loss 2

Training example 3

loss 3

Training example 4

loss 4

Training example 5

loss 5

... Training example n

loss n

compute gradient and move skier

SGD: compute gradient using only 1 training example at a time (can think of this gradient as a noisy approximation of the “full” gradient)
Stochastic Gradient Descent (SGD)

SGD: compute gradient using only 1 training example at a time (can think of this gradient as a noisy approximation of the “full” gradient)
Stochastic Gradient Descent (SGD)

SGD: compute gradient using only 1 training example at a time (can think of this gradient as a noisy approximation of the “full” gradient)
Stochastic Gradient Descent (SGD)

SGD: compute gradient using only 1 training example at a time (can think of this gradient as a noisy approximation of the “full” gradient)

An epoch refers to 1 full pass through all the training data
Minibatch Gradient Descent

- Training example 1
- Training example 2
- Training example 3
- Training example 4
- Training example 5
- …
- Training example n

↓
loss 1
↓
loss 2
↓
loss 3
↓
loss 4
↓
loss 5
↓
…
↓
loss n

↓
average loss
↓
compute gradient
↓
and move skier
Minibatch Gradient Descent

Batch size: how many training examples we consider at a time (in this example: 2)
Best optimizer? Best learning rate? Best # of epochs? Best batch size?

Active area of research

Depends on problem, data, hardware, etc

Example: even with a GPU, you can get slow learning (slower than CPU!) if you choose # epochs/batch size poorly!!!
Dealing with Small Datasets
Fine Tuning

If there’s an existing pre-trained neural net, you could modify it for your problem that has a small dataset.

**Example:** classify between Tesla’s and Toyota’s.

You collect photos from the internet of both, but your dataset size is small, on the order of 1000 images.

Strategy: take pre-trained convnet (such as the state-of-the-art ResNet) for ImageNet classification and change final layers to do classification between Tesla’s and Toyota’s instead of classifying 1000 objects.
Fine Tuning

Sentiment analysis RNN demo

Text → Embedding → Classifier → Positive/negative sentiment

Weights here are treated as fixed & come from pre-trained GloVe word embeddings

GloVe vectors pre-trained on massive dataset (Wikipedia + Gigaword)

IMDb review dataset is small in comparison
Data Augmentation

Another way of dealing with small datasets: generate perturbed versions of your training data to get a larger training dataset.

- Training image
  - Training label: cat
- Mirrored
  - Still a cat!
- Rotated & translated
  - Still a cat!

We just turned 1 training example in 3 training examples.

Allowable perturbations depend on data:
(e.g., for handwritten digits, rotating by 180 degrees would be bad: confuse 6’s and 9’s)
Generating Fake Data That Look Real
Generate Fake Data that Look Real

Unsupervised approach: generate data that look like training data

**Example:** Generative Adversarial Network (GAN)

Counterfeiter tries to get better at tricking the cop

Cop tries to get better at telling which examples are real vs fake

Terminology: counterfeiter is the **generator**, cop is the **discriminator**

Other approaches: variational autoencoders, pixelRNNs/pixelCNNs
Generate Fake Data that Look Real

Fake celebrities generated by NVIDIA using GANs (Karras et al Oct 27, 2017)

Google DeepMind’s WaveNet makes fake audio that sounds like whoever you want using pixelRNNs (Oord et al 2016)
Generate Fake Data that Look Real

Image-to-image translation results from UC Berkeley using GANs (Isola et al 2017, Zhu et al 2017)
AI News Anchor

China's Xinhua agency unveils AI news presenter

By Chris Baraniuk
Technology reporter

8 November 2018

Deep Reinforcement Learning

The machinery behind AlphaGo and similar systems

- **AI agent**
  - Al’s current state
  - Deep net

- **Environment**
  - reward
  - take action

- **Deep net**
  - score for different (state, action) pairs

- **Update agent’s state**
The Future of Deep Learning

• Deep learning currently is still very limited in what it can do
  • Learns simple computer programs (functions) comprised of a series of basic operations — need to be able to compute derivatives of these basic operations

• Adversarial examples at test time remain a problem

• Pretty much all the best ideas that lead to amazing prediction results incorporate problem-specific structure
  • For example, think about how CNNs and RNNs incorporate structure of images/time series

• How do we get away with using less expert knowledge?

• How do we do lifelong learning?

• How do we reason about causality?
Unstructured Data Analysis

Question
The dead body
This is provided by a practitioner

Data
The evidence
Some times you have to collect more evidence!

Finding Structure
Puzzle solving, careful analysis
Exploratory data analysis

Insights
Answer original question

There isn’t always a follow-up prediction problem to solve
Some Parting Thoughts

• Remember to **visualize steps of your data analysis pipeline**
  • Helpful in debugging & interpreting intermediate/final outputs
• Very often there are *tons* of models/design choices to try
  • Come up with **quantitative metrics** that make sense for your problem, and use these metrics to **evaluate models** (think about how we chose hyperparameters!)
• But don’t blindly rely on metrics without **interpreting results in the context of your original problem**!
• Often times you won’t have labels! If you really want labels:
  • Manually obtain labels (either you do it or crowdsource)
  • Set up “self-supervised” learning task
• There is a *lot* we did not cover — **keep learning!**
Want to Learn More?

- Some courses at CMU:
  - Natural language processing (analyze text): 11-611
  - Computer vision (analyze images): 16-720
  - Deep learning: 11-785, 10-707
  - Deep reinforcement learning: 10-703
  - Math for machine learning: 10-606, 10-607
  - Intro to machine learning at different levels of math: 10-601, 10-701, 10-715
  - Machine learning with large datasets: 10-605

- One of the best ways to learn material is to teach it!

  Apply to be a TA for me next term!