Introductory Overview Lecture
The Deep Learning Revolution

Christopher Manning & Russ Salakhutdinov
Stanford University ♔ Carnegie Mellon University
JSM, 2018-07-29
Plan for Part 4

Big ideas for bigger, structured neural models

a. Convolutional models
b. Recurrent models
c. Gated and residual connections
d. Attention
e. Final thoughts
4a. Vision: Convolutional models

- For computer vision, a key property that we usually wish to capture is translation invariance.
- We would like to have visual “feature detectors” that find something in an image regardless of precisely where it is located.
- We do this with a convolutional layer.

[Slides from Karpathy/Johnson/Yeung/Fei-Fei, cs231n.stanford.edu]
Motivating a Convolution Layer

32x32x3 image → 10 classes

Could stretch to 3072 x 1 and use an FC layer

A number
A probability of the object being a certain class

input

3072

weights

10 x 3072

activation

1

10
**Convolution Layer**

A convolution layer takes a 32x32x3 image and convolves it with a 5x5x3 filter. Filters always extend the full depth of the input volume.

Convolving a filter with the image, i.e., “slide over the image spatially, computing dot products”.

[Slides from Karpathy/Johnson/Yeung/Fei-Fei, cs231n.stanford.edu]
Convolution Layer

32x32x3 image
5x5x3 filter $w$

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

Do it for multiple filters

We now have a new 28x28x4 “image”
Pooling layer

- Allows convolutional features with a broader view without a lot of parameters
- Can make representations more manageable
Pooling layer

Single depth slice

max pool with 2x2 filters and stride 2
A (Very Simple) Modern ConvNet
ConvNet Representation Learning

- At the beginning of the ConvNet, features are edge and color/texture detectors
- High in the ConvNet, features detect parts and object types
4b. Language: Recurrent models

- Until now, we’ve dealt with classifying/generating fixed-size objects.
  - We just resized images to our procrustean bed!
- How can we deal with variable-size inputs, such as the word sequences in human language text or bioinformatic gene sequences?
- We do this with a recurrent layer
Recurrent Neural Networks (RNN)

Core idea: Apply the same weights \( W \) repeatedly

Read, update, predict
Example: a Recurrent “Language Model” that generates sentences

Transition Function \( h_t = f(h_{t-1}, x_t) \)

Inputs
i. Current word \( x_t \in \{1, 2, \ldots, |V|\} \)
ii. Previous state \( h_{t-1} \in \mathbb{R}^d \)

Parameters
i. Input weight matrix \( W \in \mathbb{R}^{|V| \times d} \)
ii. Transition weight matrix \( U \in \mathbb{R}^{d \times d} \)
iii. Bias vector \( b \in \mathbb{R}^d \)
Building a Recurrent Language Model

Transition Function $h_t = f(h_{t-1}, x_t)$

Naïve Transition Function

$$f(h_{t-1}, x_t) = \tanh(W [x_t] + U h_{t-1} + b)$$

Element-wise nonlinear transformation

Trainable word vector

Linear transformation of previous state

Diagram:
- $h_0$ to $h_1$
- $h_1$ to $h_2$
- $h_2$ to $h_3$
- $h_3$ to $h_4$

Words:
- $p(\text{the})$
- $p(\text{cat}|\ldots)$
- $p(\text{is}|\ldots)$
- $p(\text{eating}|\ldots)$

Tokens:
- \langle \text{bos} \rangle
- the
- cat
- is
Building a Recurrent Language Model

Prediction Function \( p(x_{t+1} = w | x_{\leq t}) = g_w(h_t) \)

Inputs
   i. Current state \( h_t \in \mathbb{R}^d \)

Parameters
   i. Softmax matrix \( R \in \mathbb{R}^{|V| \times d} \)
   ii. Bias vector \( c \in \mathbb{R}^{|V|} \)
Building a Recurrent Language Model

**Prediction Function**\( p(x_{t+1} = w | x_{\leq t}) = g_w(h_t) \)

\[
p(x_{t+1} = w | x_{\leq t}) = g_w(h_t) = \frac{\exp(R[w]^\top h_t + c_w)}{\sum_{i=1}^{|V|} \exp(R[i]^\top h_t + c_i)}
\]

This gives a probability distribution over next words. To generate text we take max. prob. word (or sample a word), and then use what we generated as the input at the next time step.
Learned word representations show words’ semantic similarity

\[
\text{expect} = \begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
0.109 \\
-0.542 \\
0.349 \\
0.271 \\
0.487
\end{pmatrix}
\]
The protests escalated over the weekend.
4c. Gated Units
Fixing Backpropagation through Time

Intuitively, what happens with RNNs?

1. Measure the influence of the past on the future

$$\frac{\partial \log p(x_{t+n} \mid x_{<t+n})}{\partial h_t} = \frac{\partial \log p(x_{t+n} \mid x_{<t+n})}{\partial g} \frac{\partial g}{\partial h_{t+n}} \frac{\partial h_{t+n}}{\partial h_{t+n-1}} \ldots \frac{\partial h_{t+1}}{\partial h_t}$$

2. How does the perturbation at $t$ affect $p(x_{t+n} \mid x_{<t+n})$?
Backpropagation through Time

Problem: Often get a vanishing gradient which is super-problematic

- With the naïve transition function
  \[ f(h_{t-1}, x_t) = \tanh(W \left[ x_t \right] + Uh_{t-1} + b) \]
- The temporal derivative is
  \[ \frac{\partial h_{t+1}}{\partial h_t} = U^\top \frac{\partial \tanh(a)}{\partial a} \]
- When we only observe
  \[ \left\| \frac{\partial h_{t+N}}{\partial h_t} \right\| = \left\| \prod_{n=1}^{N} U^\top \text{diag} \left( \frac{\partial \tanh(a_{t+n})}{\partial a_{t+n}} \right) \right\| \to 0 \]
- We cannot tell whether there is
  1. No dependency between \( t \) and \( t+N \) in data, or
  2. Small weights (leading eigenvalue < 1) \( \to \) gradient vanishes
Gated Recurrent Unit

- With a simple recurrent model, the error must backpropagate through all the intermediate nodes:

- Perhaps we can create shortcut connections
Gated Recurrent Unit

- Perhaps we can create *adaptive* shortcut connections

- Candidate Update
  \[ \tilde{h}_t = \tanh(W [x_t] + Uh_{t-1} + b) \]

- Update gate
  \[ u_t = \sigma(W_u [x_t] + U_u h_{t-1} + b_u) \]

\[ f(h_{t-1}, x_t) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1} \]

\( \odot \): element-wise multiplication
Gated Recurrent Unit

- Let the net prune unnecessary connections \textit{adaptively}

\[
\tilde{h}_t = \tanh(W [x_t] + U (r_t \odot h_{t-1}) + b)
\]

- Candidate Update

\[
r_t = \sigma(W_r [x_t] + U_r h_{t-1} + b_r)
\]

- Reset gate

\[
u_t = \sigma(W_u [x_t] + U_u h_{t-1} + b_u)
\]

- Update gate

\[
f(h_{t-1}, x_t) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}
\]
Gated Recurrent Unit

\textit{tanh-RNN} ....

\begin{align*}
\text{Execution} \\
1. \text{Read the whole register } h \\
2. \text{Update the whole register} \\
   & h \leftarrow \tanh(W \left[ x \right] + Uh + b)
\end{align*}
Gated Recurrent Unit

GRU ...

Registers $h$

Execution
1. Select a readable subset $r$
2. Read the subset $r \odot h$
3. Select a writable subset $u$
4. Update the subset
   $$h \leftarrow u \odot \tilde{h} + (1 - u_t) \odot h$$

Gated recurrent units are much more realistic!
Gated Recurrent Units

Two most widely used gated recurrent units

Gated Recurrent Unit
[Cho et al., EMNLP2014; Chung, Gulcehre, Cho, Bengio, DLUFL2014]

\[
\begin{align*}
    h_t &= u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1} \\
    \tilde{h}_t &= \tanh(W[x_t] + U(r_t \odot h_{t-1}) + b) \\
    u_t &= \sigma(W_u [x_t] + U_u h_{t-1} + b_u) \\
    r_t &= \sigma(W_r [x_t] + U_r h_{t-1} + b_r)
\end{align*}
\]

Long Short-Term Memory
[Hochreiter & Schmidhuber, NC1999; Gers, Thesis2001]

\[
\begin{align*}
    h_t &= o_t \odot \tanh(c_t) \\
    c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
    \tilde{c}_t &= \tanh(W_c [x_t] + U_c h_{t-1} + b_c) \\
    o_t &= \sigma(W_o [x_t] + U_o h_{t-1} + b_o) \\
    i_t &= \sigma(W_i [x_t] + U_i h_{t-1} + b_i) \\
    f_t &= \sigma(W_f [x_t] + U_f h_{t-1} + b_f)
\end{align*}
\]

In many ways similar, but the LSTM is actually more powerful: Its cell can count
Gated units

Gating is a general idea, which is now used in a whole bunch of places

You can also gate vertically

- Indeed the key idea – summing candidate update with shortcut connection – is needed for very deep networks to work
4d. (Selective) Attention

• Problem: The NMT system had to remember the entire input in one hidden vector ... regardless of how long the input sentence was.

• And in general we might like to have a large external memory and to selectively look at certain bits of information in it at various times.

• Solution: Neural Attention
Sequence-to-sequence: bottleneck problem

Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

Encoder RNN

Source sentence (input)
les pauvres sont démunis

Decoder RNN

Target sentence (output)
<START> the poor don’t have any money <END>
Sequence-to-sequence with attention

On this decoder timestep, we’re mostly focusing on the first encoder hidden state (“les”)

Take softmax to turn the scores into a probability distribution

Calculate a compatibility score between current and input hidden states

Source sentence (input)
Sequence-to-sequence with attention

Use the attention distribution to take a **weighted sum** of the encoder hidden states.

Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

$$
e_t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

$$
\alpha_t = \text{softmax}(e_t) \in \mathbb{R}^N
$$

$$
\mathbf{a}_t = \sum_{i=1}^{N} \alpha_i^t h_i \in \mathbb{R}^h
$$

Output: $g_w(\mathbf{a}_t, s_t)$
Attention is great

- **Attention significantly improves NMT performance**
  - It’s very useful to allow decoder to focus on certain parts of the source
- **Attention solves the bottleneck problem**
  - Attention allows decoder to look directly at source; bypass bottleneck
- **Attention helps with vanishing gradient problem**
  - Rather than traditional optimization problems, attention provides an alternative shortcut to faraway states
- **Attention provides some interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get alignment in MT for free!
  - This is cool because the network learned alignment by itself
Attention is a *general* Deep Learning technique

- **Variant attention functions:**
  - Basic dot-product attention:
  - Multiplicative attention:
  - Additive attention:

  \[ e_i = s^T h_i \in \mathbb{R} \]
  \[ e_i = s^T W h_i \in \mathbb{R} \]
  \[ e_i = v^T \tanh(W_1 h_i + W_2 s) \in \mathbb{R} \]

- **More general definition of attention:**
  - Given a set of vector *keys* with associated vector *values*, and a vector *query*, *attention* is a technique to compute a weighted sum of the values, dependent on the query.
  - We measure a score between the query and each key and then return an aggregate value which is a score-weighted average of the values.
Attention in Question Answering
(Stanford Attentive Reader: Chen et al. 2016, 2017)

Who did Genghis Khan unite before he began conquering the rest of Eurasia?

Process with LSTMs

\[ \alpha_i = \text{softmax}(q^T W_i) \]

\[ \alpha_i' = \text{softmax}(q^T W_i' \tilde{p}_i) \]

He came to power by uniting many of the nomadic tribes of Northeast Asia. After founding the Mongol Empire and being proclaimed "Genghis Khan", he started the Mongol invasions that resulted in the conquest of most of Eurasia. These included raids or invasions of the Qara Khitai, Caucasus, Khwarezmid Empire, Western Xia and Jin dynasties. These campaigns were often accompanied by wholesale massacres of the civilian populations – especially in the Khwarezmian and Xia controlled lands. By the end of his life, the Mongol Empire occupied a substantial portion of Central Asia and China.

\[ \rightarrow \text{predict start token} \]

\[ \rightarrow \text{predict end token} \]
Final thoughts

Here are a few “empirical understandings” of a decade of deep learning, some of which deserve better formal understanding.
Non-convex models really are more powerful than convex models

- Extremely deep models using the residual connections (ResNets) idea, have powered the 2015–2017 gains on ImageNet
- But what are they doing? With residual connections it doesn’t seem to be quite the traditional idea of layers of representation
Use of distributed representations of categorical values like words allows very effective modeling and sharing of dimensions of similarity.
Models with orders of magnitude more parameters than there are training data available, if trained with ample amounts of regularization (including new forms like dropout), will *outperform* simpler models – *generalizing better* to new data

- Practitioner’s recipe: build a sufficiently high-capacity model that it can be trained to 0% training error, and then increase its regularization until it doesn’t overfit on dev
When we were young, our parents used to warn us with tales of bad local minima, but it turns out that bad local minima typically don’t exist and any different minima seem to be roughly equivalently good. No need to worry!
Rather than a clean separation between an objective (loss) function and an optimizer that is trying to minimize it, we find that in current neural network successes, the optimizer is mixed up into the loss function – small batch SGD is acting as a form of regularizer.
Thanks!

Questions?