RNN

Mengye Ren
mren@cs.toronto.edu
Agenda

- Basics
- Bag of applications
- LSTM & GRU
- Bag of tricks
- Other architecture considerations (attention, memory etc.)

Picture credits: http://colah.github.io/
[Hillary Clinton] was all talk. I was screaming -- jobs and extremists, not policy. But I won.

@ChadHGriffin @HillaryClinton #debates2016
Recurrent Neural Networks (RNN)

- A neural network with a closed loop -> Wire the output back to the input.
- Lots of sequential data around us: text, music, speech, video, etc.
- We can feed sequential data into RNN frame by frame: speech recognition, video classification, etc.
- We can also expect RNN to emit sequential output: Language modeling, machine translation, speech generation, video generation.
Recurrent Neural Networks (RNN)

\[
a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}
\]

\[
h^{(t)} = \tanh(a^{(t)})
\]

\[
o^{(t)} = c + Vh^{(t)}
\]

\[
y^{(t)} = \text{softmax}(o^{(t)})
\]
Machine Translation

- Encoder-Decoder RNN model. Encodes the sentence using an RNN, and decode the hidden state into another language.
- Since 2015, RNNs started to beat traditional phrase-based systems.

(Sutskever et al., 2015)
DRAW

(Gregor et al., 2015)

https://youtu.be/Zt-7MI9eKEo
Instance Segmentation

(Ren et al., 2016)

https://youtu.be/wJWydIqtFMM
Detecting Events and Key Actors

- Use bidirectional RNN to model features of a moving player.

(Vignesh Ramanathan et al., 2016)
Music Generation

https://vimeo.com/192711856

(Chu et al., 2016)
Training RNN with BPTT

- BPTT = Backpropagation through time
- Unroll the RNN as if they are sharing weights every layer.
- Each time step generates a gradient vector towards the weights.
- Average all the gradients collected at each time step.
RNN vanishing gradients

- Vanilla RNNs suffers at vanishing gradients problems.
- The derivative the activation nonlinearities, sigmoid or tanh, is smaller than 1.
- Therefore, the hidden transfer function should be a linear function.
LSTM

- Stands for Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)
- A linear pathway for the gradient to flow effortlessly.
- Gated units: multiplicative gates controls input, forget, and output.
LSTM

\[ f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f) \]
\[ i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i) \]
\[ \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \]
\[ C_t = f_t \ast C_{t-1} + i_t \ast \tilde{C}_t \]
\[ o_t = \sigma (W_o \cdot [h_{t-1}, x_t] + b_o) \]
\[ h_t = o_t \ast \tanh(C_t) \]
GRU

- Stands for Gated Recurrent Unit (Chung et al., 2015)

\[
\begin{align*}
    z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
    r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
    \tilde{h}_t &= \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
Examples (Python for-loop)

dim = 256  # Hidden dimension of LSTM
batch_size = 32  # Batch size
cell = tf.nn.rnn_cell.BasicLSTMCell(dim)
state = tf.zeros([batch_size, dim * 2])
inputs = ...  # List of input tensors
outputs = []
for t in range(20):
    output, state = cell(inputs[t], state)
    outputs.append(output)
    tf.get_variable_scope().reuse_variables()
loss = f(outputs, targets)
RNN Cell Object

- In TensorFlow, RNNs are usually implemented in a cell object.
- Cell is callable, which will unroll the RNN for one more iteration.
- Input: $x_t$, state$_{(t-1)}$
- Output: $h_t$, state$_{(t)}$
For faster computation and better gradient estimates, typically make the input into mini-batches.

However, not all sequences have the same length.

We need to pad the shorter sequences with zeros.

We can also select the hidden states at specific length by the end of computation.
Example (Dynamic RNN)

cell = tf.nn.rnn_cell.BasicLSTMCell(dim)

# Sequence length of each example.
seq_len = tf.constant(np.array([1, 2, 3, 4]))
outputs, last_states = tf.nn.dynamic_rnn(
    cell=cell, dtype=tf.float32, sequence_length=seq_len, inputs=x)
Use of DynamicRNN Function

- Handles different sequence length per batch.
- Faster graph building time, by using \texttt{tf.while_loop} internally.
- Syntax is stricter, have to wrap everything in the for loop inside the cell object.
Truncated BPTT

- Many RNN training is bounded by the GPU memory.
- All the activations need to be stored at every timestep.
- For very long sequence length (>1000 steps), this is not realistic.
- We can fix the unrolling at K timesteps, upper bound memory usage by K.
- Sacrificing the quality of the gradient, or in other word, the long-range dependency error signal.
Checkpointing

- We can sacrifice the computation time in exchange for memory space.
- Only store forward pass activation every K steps.
- Need to recompute activations during backward pass.
- Takes 2 forward passes + 1 backward pass, but \( \max(K, T/K) \) timesteps memory usage.
Gradient Clipping

- Even with LSTMs, sometimes the gradient can still explode.
- When that happens, you will get a NaN after hours of training.
- Always good to add gradient clipping in your code of training RNN.

```python
tvars = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(
    tf.gradients(_opt_cost, tvars), 5.0)
optimizer.apply_gradients(zip(grads, tvars))
```
Training Generative RNN Model

- Common way of training generative RNN is to maximize the log probability of the training sequence.
- Apply the chain rule: \( \sum_t ( \log p(x_{t+1} \mid x_{1:t}) ) \)
- Common way is to always use the groundtruth sequence \( x_{1:t} \) to warm up.
Training Generative RNN Model

- At test time, to generate new sequences, we take the output of the RNN, and feed it back into the input.
- May not be optimal, never trained to plan for more than 1 step.
Beam Search

- While optimal decoding is combinatorial, we can allow keeping a candidate list of size $B$ (beam size).
- Keep top $N$ most probable sequences, and expand each of them with the next output choice, and take $N$ top candidates.
- Similar to sequential Monte Carlo.
- End up running $B$ forward passes.
Scheduled Sampling

- At training stage, the network has never seen any fake sentences as input.
- At test stage, once the network generates something unlike a real sentence, it will have less experience keep generating.
- Remedy #1: At training, at each timestep, we do a coinflip:
  - Either feed in the groundtruth input
  - Or use the network's current output as input

But we still train the output against the groundtruth, even if the input is already sampled from the network.

- Annealing, and eventually always using network’s output as input.
- Used in Google’s Image Captioning algorithm (Bengio et al., 2015).
Scheduled Sampling

- An inconsistent learning objective (Huszar, 2015)
Professor-Forcing

- Use a Generative Adversarial Network idea (Goodfellow et al. 2015).
- Train RNN as usual [1], but also sample free-runs [2]
- Send the hidden states of both [1] and [2] to a discriminator (also an RNN).

(Lamb et al., 2016)
Reward-Augmented ML

- Maximum likelihood may not be the best objective, certainly not when we evaluate those generative models.
- In MT, BLEU scores and variants are widely used (non-differentiable).
- Tempting to use BLEU as the reward function and just apply RL, except it’s very difficult to kick off training.
- Idea is that we can treat reward as unnormalized energy of true distribution, and minimize the KL between model distribution, and data distribution soften by the reward(y, y*).
- Returns to log likelihood, if reward(y, y*) is the delta function.

(Norouzi et al., 2016)
Recurrent Attention

- In MT, RNN can be forgetful in long sequences.
- Want to have a look back mechanism while decoding.
- Attention in the form of a weighted sum of a bag of items (Bahdanau et al., 2014).
Recurrent Attention

- Can also attend on spatial level in vision tasks.
- Initialize at uniform.
- Use the previous hidden state to control the attention of the next.

(Xu et al., 2015)

(b) A woman is throwing a frisbee in a park.
Neural Turing Machine

- Proposed an external memory which supports soft-read and soft-write using attention mechanism (Graves et al., 2014).
Phased LSTM

(Neil et al., 2016)