Bootcamp 3: Recurrent Neural Nets in Tensorflow

ML@B Bootcamp

Hosted by Machine Learning at Berkeley Education

Agenda

Background and Motivation

Introducing RNN's

RNN's in Tensorflow

Time to Code

Mind Expanding Stuff

Applications

Questions

Background and Motivation

Framing

Say we want to train a model to analyze text like this.

```
}

const mapStateToProps = (state) => ({
  session: state.session
})


class CoreLayout extends Component {
  static propTypes = {
    children: PropTypes.object.isRequired
  }

  constructor(props) {
    super(props)
    this.handleLogin = this.handleLogin.bind(this)
  }

  handleLogin(loginObj, e) {
    e.preventDefault()
    this.props.loginAsync(loginObj)
  }

  render () {
    return (
      <div className='container text-center'>
        <Header
          handleLogin={this.handleLogin}
          session={this.props.session} />
        <div className={classes.mainContainer}>
          {this.props.children}
        </div>
      </div>)

    }
}


export default connect(mapStateToProps, mapActionCreators)(CoreLayout)
```

What are the most important types of relationships in this data?

Framing

Code (and any text for that matter) is read sequentially:

- Open brackets are followed by close brackets later down
- If, For, While controls mean the next line is indented
- Other relationships - dot notation means dots are followed by parantheses, etc.

What do all of these have in common? Each have a TEMPORAL relationship over data. Things in the past affect things in the future, and we need a way to capture those relationships.

- CNNs and VNNs look at the whole document all at once, not as something that progresses over time

The Solution - RNNs!

The solution is to have a concept of "internal state", and to allow sequential data to alter that state in time

$$h^t = \sigma(U * x^t + W * h^{t-1})$$

- This is really convenient for sequential data: the more data you have, just iterate more in time - it can comprehend as much data as you pass into it
- This is also really good for continuous generation - it's internal state can predict the next character for instance, and the next time step will output another one based on the update

The Solution - RNNs!

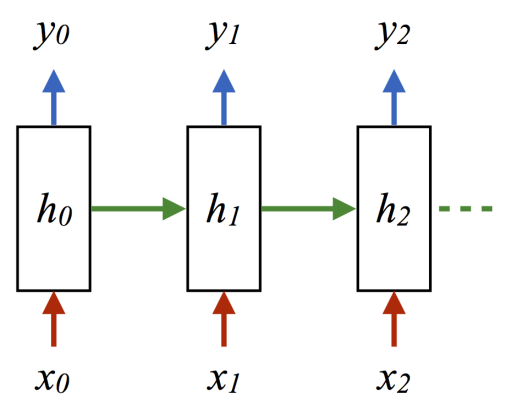
- We now have a model that understands "trends" over time.
- Similar to how CNNs mimic biological perception, RNNs mimic how our brains process sequential data (for example, when reading a book, every word/sentence slightly alters your mind's constructed understanding/representation of the book)
- These models have given us state of the art results in speech/writing recognition, summarizing, other NLP, sound processing

Introducing RNN's

What does Recurrent mean?

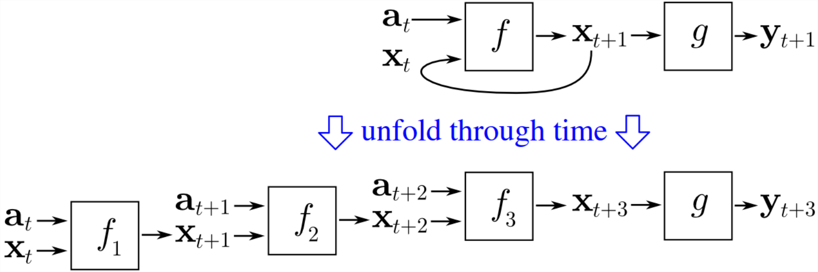
A neural network is recurrent if its outputs directly alter its parameters **without backpropagation**.

- The altered parameters are the hidden state: $\mathbf{h_t}$. These can be individual units or entire layers.



The computation that produces output y_T necessarily involves the parameters h_t with $t < T$.

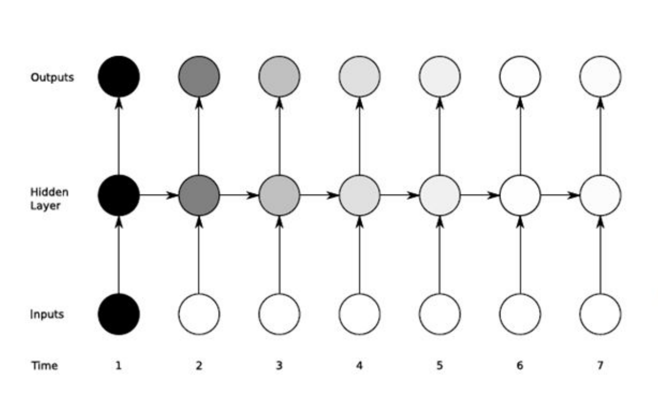
- If we unroll n steps, we get an $n+1$ layer network with each layer being different time steps.
- Backpropagation will alter h_T by computing $\frac{\partial J(\theta)}{\partial h_t}$



Complications in Training

Incorporating inputs from many timesteps is hard due to **vanishing gradients.**

- Early inputs lose influence over network by exponential decay.
- Calculating gradient over a product can be wrecked by noise.



LSTM Networks

The LSTM cell gives additional learnable parameters that can process hidden states further.

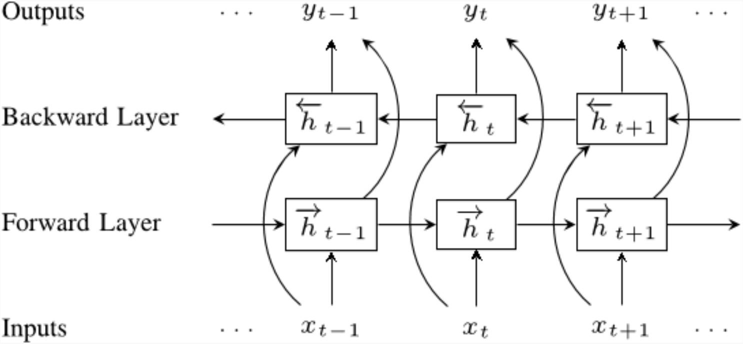
- The gates are NN layers with nonlinearities that can preserve or erase/overwrite hidden states with substantial control.
- Can keep "context" alive and learn "Long Short-Term Memory."

$$\begin{pmatrix} i \\ f \\ o \\ g \\ a_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \\ \mathrm{softmax} \end{pmatrix} \circ W \begin{pmatrix} I \circ a_{t-1} \\ h_{t-1} \\ x \end{pmatrix}$$

Some other cool things

Like most other architectures, the definition of RNNs allows for wild customizations.

- Peephole connections: h_t considers h_{t-1} and at least one other h_{t-i}: $1 < i < t$.
- Bidirectional RNN: h_t considers h_{t-1} and (usually) h_{t+1}.



RNN's in Tensorflow

tf.nn.rnn_cell.RNNCell

Tensorflow has a nice collection of RNN Cells and an abstract class from which you may make your own.

- A cell as used in TF documentation refers to a layer.
- All tf.nn.rnn cell.RNNCell's must implement the call function:
  `call(inputs, state) = (output, next_state)`

`class BasicLSTMCell` : Basic LSTM recurrent network cell.

`class BasicRNNCell` : The most basic RNN cell.

`class DeviceWrapper` : Operator that ensures an RNNCell runs on a particular device.

`class DropoutWrapper` : Operator adding dropout to inputs and outputs of the given cell.

`class GRUCell` : Gated Recurrent Unit cell (cf. http://arxiv.org/abs/1406.1078).

`class LSTMCell` : Long short-term memory unit (LSTM) recurrent network cell.

`class LSTMStateTuple` : Tuple used by LSTM Cells for `state_size`, `zero_state`, and output state.

`class MultiRNNCell` : RNN cell composed sequentially of multiple simple cells.

`class RNNCell` : Abstract object representing an RNN cell.

`class ResidualWrapper` : RNNCell wrapper that ensures cell inputs are added to the outputs.

tf.nn.rnn_cell.LSTMCell

This is the standard LSTM implementation in Tensorflow.

- num_units is the size of the layer.
- Can easily set up hyperparameters, meta-heuristics (e.g. peepholes, gradient clipping)

```
__init__(
    num_units,
    use_peepholes=False,
    cell_clip=None,
    initializer=None,
    num_proj=None,
    proj_clip=None,
    num_unit_shards=None,
    num_proj_shards=None,
    forget_bias=1.0,
    state_is_tuple=True,
    activation=None,
    reuse=None
)
```

Just define multiple cells and join them however you wish.

- The composition of RNNCells is still an RNNCell.

```
__init__(
    cells,
    state_is_tuple=True
)
```

tf.nn.rnn_cell.DropoutWrapper

Wrapping a cell produces another cell, so these are very easy modifications.

- Checkout EmbeddingWrapper and ResidualWrapper. Basically layer accessories.

```
__init__(
    cell,
    input_keep_prob=1.0,
    output_keep_prob=1.0,
    state_keep_prob=1.0,
    variational_recurrent=False,
    input_size=None,
    dtype=None,
    seed=None
)
```

tf.nn.dynamic_rnn

Amazing class that takes a cell (and inputs) and creates a working model. Dynamic refers to the model being able to process variable length inputs.

- Allows different max_time values for different mini-batches. Alternatively use sequence_length.

```
dynamic_rnn(
    cell,
    inputs,
    sequence_length=None,
    initial_state=None,
    dtype=None,
    parallel_iterations=None,
    swap_memory=False,
    time_major=False,
    scope=None
)
```

Time to Code

Mind Expanding Stuff

Augmented RNN's

We can use RNNs as the core processing unit of a more complex model.

- RNN can give commands rather than output answer
- We can convert discrete space into continuous with **attention**
- Jump into the rabbit hole...

Takeaway

The Big Picture

A human with a piece of paper is, in some sense, much smarter than a human without. A human with mathematical notation can solve problems they otherwise couldn't. Access to computers makes us capable of incredible feats that would otherwise be far beyond us.

In general, it seems like a lot of interesting forms of intelligence are an interaction between the creative heuristic intuition of humans and some more crisp and careful media, like language or equations. Sometimes, the medium is something that physically exists, and stores information for us, prevents us from making mistakes, or does computational heavy lifting. In other cases, the medium is a model in our head that we manipulate. Either way, it seems deeply fundamental to intelligence.

- Olah & Carter, "Attention and Augmented Recurrent Neural Networks", Distill, 2016.

Impermanence

It may seem that augmented LSTM/GRU networks are amazing and here to stay. But don't be fooled!

- Tunable Efficient Unitary Neural Networks (EUNN) [Jing, et. al. 2017] use unitary matrices to completely sidestep the vanishing gradient problem and may render LSTM/GRU's obsolete.

Applications

Image Captioning

The main idea is that you first pass the image through a few convolutional layers, and then pass that learned representation into an LSTM RNN which generates characters based on that.

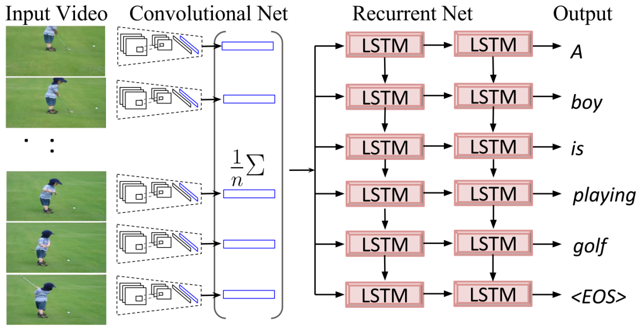


Image Captioning

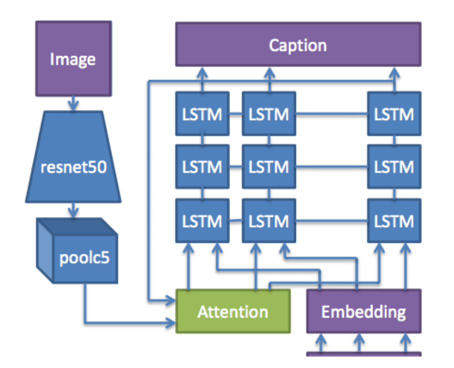
The biggest problem with this architecture is that there's no way to weight parts of the image according to their "importance". We can fix this by adding an "attention mechanism" that trains the network to weight certain parts of an image more. The attention LSTM update equations are:

$$\begin{pmatrix} i \\ f \\ o \\ g \\ a_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \\ \mathrm{softmax} \end{pmatrix} \circ W \begin{pmatrix} I \circ a_{t-1} \\ h_{t-1} \\ x \end{pmatrix}$$

a_t represents attention parameters at time t

Image Captioning

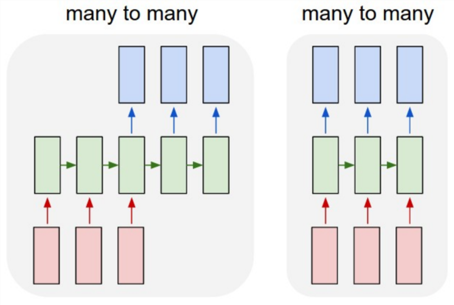
The entire model looks something like this:



resnet50 is the CNN, the attention parameters (in green) are both influencing and influenced by the LSTM, and the embedding is a transformation of the provided word vocabulary (one-hot encoded)

Sequence to Sequence

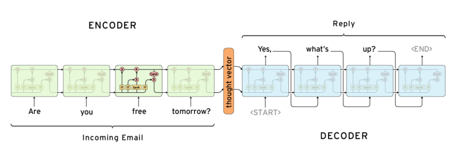
A sequence to sequence architecture looks something like this:



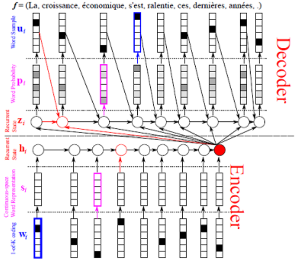
This is widely used in language translation, for example. You pass in the sentence you want to translate as word tokens, the RNN learns that representation, and spits out that sentence in another language.

Sequence to Sequence

More concretely, you have an encoder and a decoder:

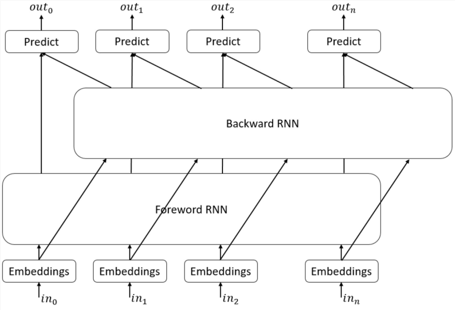


State of the art implementations use the thought vector for attention as well:



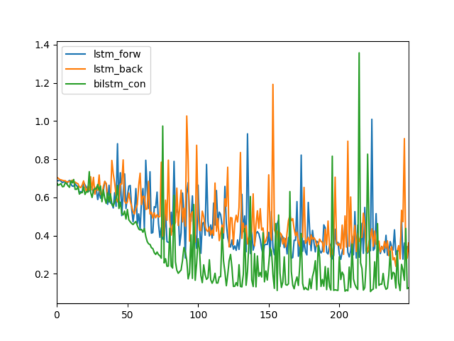
The inclusion of attention is an important step in language translation, but even with it, you only have access to input from previous timesteps. What if there was a way to be able to look into the future....

Sequence to Sequence



We can use a bidirectional RNN - the forward layer is trained on words from left to right through the corpus, and the backward layer is trained on words from right to left through the corpus. The output for each time step is some combination of both layers. This mechanism gives us much more understanding and accuracy of translation

Sequence to Sequence



Bidirectional RNN outperforms RNN on sequence classification task

Questions

Questions?!