

Tools & Models for Data Science

Deep Learning with TensorFlow

Chris Jermaine & Risa Myers

Rice University



This Lecture...

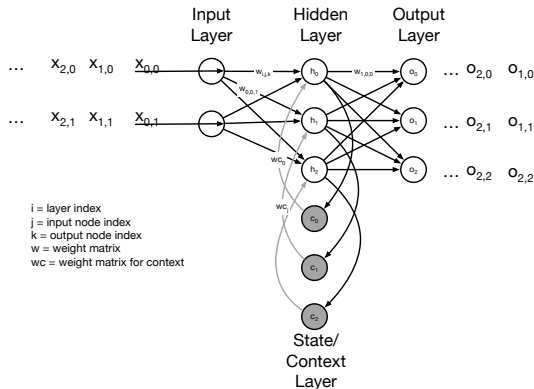
- Is meant to de-mystify RNN.py
 - RNN.py is the code we give you with the TensorFlow lab
 - Use this lecture + the Internet when you puzzle over the code

What is the lab implementing?

- Classifying sequences of raw text
- Using a recurrent neural network (RNN)

What's an RNN?

- Covered in the next lecture!
- Basically, a neural network architecture that maintains state information and feeds that back into the network



What Is TensorFlow?

- Has many components...
- Including a distributed computation engine
- But for our purposes, it is an automatic differentiation engine
 - That is, you specify an “forward” process (the inference process)
 - And it automatically figures out a gradient descent algorithm (the “backward” or learning process)
 - Even if you have a complicated forward process, it will differentiate it for you
 - Even something complicated like an RNN
 - TF will also allow you to efficiently execute the backward propagation algorithm over data

How to Specific A Forward Process

- You push tensors through various computations TF provides
 - A “tensor” is a generalization of a matrix/vector
 - Can have any number of dimensions
 - Tensors are TF’s most fundamental data type
 - Everything is based off of them

- Use floats (32-bit representation)
 - Support for doubles (64-bit) is not great
 - Added in later in TF
 - The extra bits are just not that important in ML
 - Millions of neurons, working together can withstand a lot of inaccuracy from loss of precision
 - Shorter representation can speed computation

Two Kinds of Tensors in Forward Process

■ 1) Placeholders

- These are tensors whose value will be provided at training time
- They are inputs into the forward process
- Declared like

```
inputX = tf.placeholder  
        (tf.float32, [batchSize, 256 * maxSeqLen])
```

- `batchSize` = size of mini-batch used
- `256` = one hot encoding (256 characters)
- `maxSeqLen` = max length of line of text
- `[batchSize, 256 * maxSeqLen]` = 2D tensor dimensions

Two Kinds of Tensors in Forward Process

■ (2) Variables

- These are tensors whose value will be learned
- They are computed during the backward process
- Declared like this for the bias

```
b = tf.Variable(np.zeros((1, hiddenUnits)), dtype=tf.float32)
```

- `(1, hiddenUnits)` = initialization dimensions

Constructing the Forward Process

- The forward process is specified by composing operators over tensors
- Example:

```
next_state = tf.tanh(tf.matmul(inputPlusState, W) + b)
```

- In this line of code, operators are “tanh”, “matmul,” and “+”
 - Note: TF does not actually run these operators!
 - It remembers how you composed them
 - And uses those computations to build the backward process later
- `inputPlusState` = characters concatenated with the next state
- `W` = Weight matrix
- `b` = bias
- Recall: The input to a layer is a matrix multiplication of the weights and the outputs from the prior layer

Another Interesting Operation

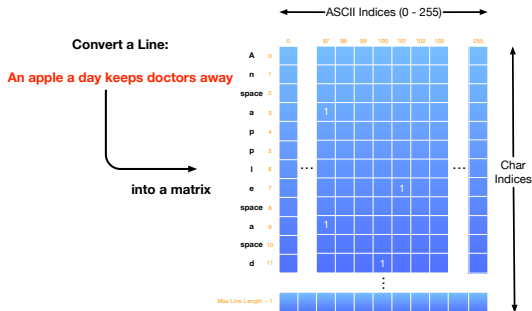
- The “unstack” operation
- Example:

```
sequenceOfLetters = tf.unstack(inputX, axis=2)
```

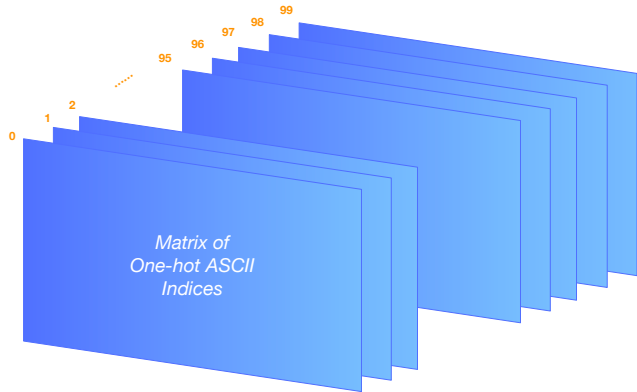
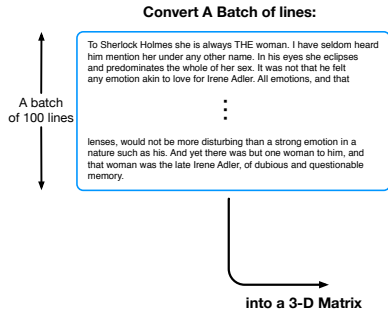
- Takes a d dimension tensor
- And converts it to a list of $d, d - 1$ dimension tensors
- Useful if you want to iterate through a tensor
- In RNN, we want to iterate through the lines of text

Input Data

- Batches of 2D tensors stored as a dictionary
- Key: Line number (sequentially assigned)
- Value: Pair of (class, One-hot-encoding of the character)
- Where
 - Dimension 1: Character in the line
 - Dimension 2: One-hot-encoding of the character
 - Lines are padded to make them the same length



Batches



Cross Entropy Loss for Classification

- Measures how close a prediction (from a vector of probabilities) is to the actual label
- Probabilities come from a softmax function
- We want high probabilities to go to the correct labels
- Cross entropy is:

$$H(p, q) = - \sum_x p(x) \log q(x)$$

- We want to minimize it!
- Used as the loss function in training

Cross-Entropy in TensorFlow

- Trivially implemented in TF
- They give you an operator for it

```
losses = tf.nn.sparse_softmax_cross_entropy_with_logits  
        (logits=outputs, labels=inputY)
```

- Input to function (outputs in this case) is output from tanh functions at top of the NN
- There should be one tanh output per class per data point
- Also accepts a vector with the correct labels
- TF does softmax and compares predictions with the truth
- Then computes cross entropy

- TF has many different gradient-based algorithms to choose from
 - Switching between them means changing one line of code
 - Ex:

```
trainingAlg = tf.train.AdagradOptimizer  
              (0.02).minimize(totalLoss)
```

- This (obviously) runs the Adagrad algorithm (look it up!)
- Alternative to gradient descent
- 0.02 is the learning rate

Invoking One Iter of Gradient Descent

```
_totalLoss, _trainingAlg, _currentState,  
  _predictions, _outputs = sess.run(  
    [totalLoss, trainingAlg, currentState,  
     predictions, outputs],  
    feed_dict={  
        inputX:x,  
        inputY:y,  
        initialState:_currentState  
    })
```

■ Args to “run”:

- “feed_dict” should tell TF what values to use for each placeholder
- You miss a placeholder? TF will complain
- List of vars (totalLoss, trainingAlg, etc)... what are they for?
- Tells TF any Variables whose values you want returned
- Given back to you as NumPy arrays

Invoking One Iter of Gradient Descent

```
_totalLoss, _trainingAlg, _currentState,  
    _predictions, _outputs = sess.run(  
    [totalLoss, trainingAlg, currentState,  
    predictions, outputs],  
    feed_dict={  
        inputX:x,  
        inputY:y,  
        initialState:_currentState  
    })
```

- `sess.run`: 1 iteration
- `_currentState`: numpy matrix of hidden states; all zeros initially, 1st value for hidden states
- Watch the values of the tensors returned

Accessing Variables

- Just ask “run” to return it
 - Like in last slide
 - Or, just use

```
sess.run (W)
```

- Will return last value of W as a NumPy array

Saving Sessions

- When a training session ends, it is gone
- But you can save it

```
saver = tf.train.Saver()  
saver.save(sess, 'checkPoint.tf')
```

- Then later can load it up again

```
sess = tf.Session()  
saver = tf.train.Saver()  
saver.restore(sess, 'checkPoint.tf')
```

- Very useful!!

My Code Doesn't work!

- Make sure your NN is correctly connected
 - Check sizes and shapes of tensors
 - Check the input data
- Monitor the objective function / loss
- Check the initialization (note small variance)
- Check the number of hidden units
- Print out information at each iteration
- Restructure the code to be able to work with it interactively

A6: Keras vs. TensorFlow

- Don't have to declare variables and placeholders
- Running the model uses a slightly different API

1 `compile`

2 Data generator to handle batches

3 Iteration control

4 `fit_generator`

5 `predict`

6 `confusion_matrix`

- Prepares the model for training
 - Creates a TF session
 - Initializes variables in that session
 - Create the specified loss function for the network

- Creates a generator (a data preprocessor)
 - `flow` tells Keras how the data flows through the generator
 - We are using the defaults
 - Tells Keras the training data, labels, and batch size

TF

- Explicit
- We are specifying exactly what outputs we want
- Running one epoch at a time
- Have visibility into the hidden node values

Keras

- Implicit
- Running n_epochs epochs
- Only see final output

- Learn the prepared model
- Batches
- Could add in code to validate the model

- Analogous to `session.run` in TF
- Run the test data through the trained model
- Returns output from final layer (should be softmax) as a numpy array
- `argmax` returns the highest value

- Built-in scikit learn function
- Returns the confusion matrix
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

A6: Image data

- RGB images provided
- Initial size is 320×240
- Be careful of the data types - it is integer, TF and Keras all expect floats
- n_x , n_y are the initial image resolution
- **Use average pooling to downsample by a factor of 4**

A6:Dictionary of models

- Mechanism provided to easily run multiple models
- Add / remove models from the dictionary to have them run / not run

Questions?