TensorFlow Basics

by: Chris Dongjoo Kim

Basic intro slides derived from web

Why Tensor + Flow ?

Tensors: n-dimensional arrays

Vector: 1-D tensor

Matrix: 2-D tensor

Deep learning process are **flows of tensors**

- A sequence of tensor operations
- Can represent also many machine learning algorithms

The Big Picture

There are basically 2 phases like many other deep learning libraries.

1. Defining Phase: result cannot be obtained
- Defines data, variables
- model architecture
- Cost function, optimizer
2. Execution Phase: result can be obtained
- Executes the predefined ops and variables.
- Learning phase

ReLU net using simple operations

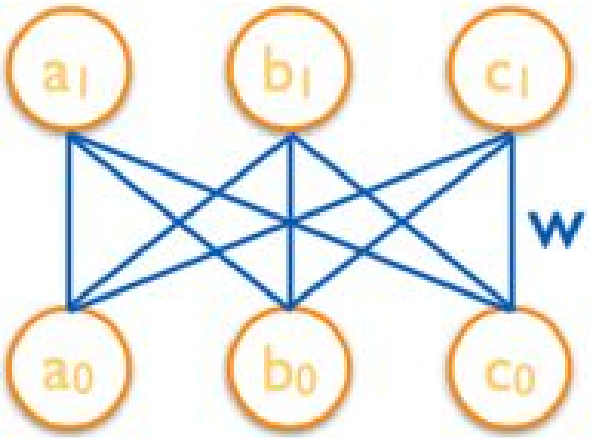
If we were to perform using a per unit operation where,

And apply relu() on each, it would be a lot less efficient.

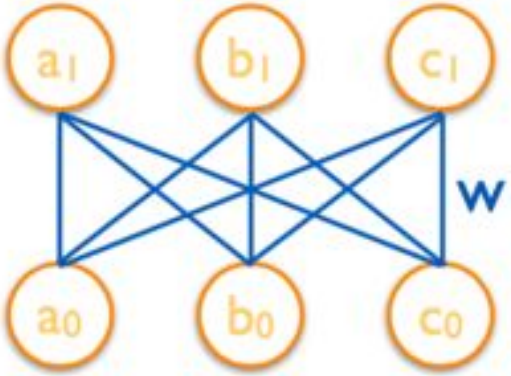
$$a_1 = a_0 w_{a,a} + b_0 w_{b,a} + c_0 w_{c,a}$$

$$b_1 = a_0 w_{a,b} + b_0 w_{b,b} + c_0 w_{c,b}$$

$$c_1 = a_0 w_{a,c} + b_0 w_{b,c} + c_0 w_{c,c}$$



ReLu net using Matrix Operations

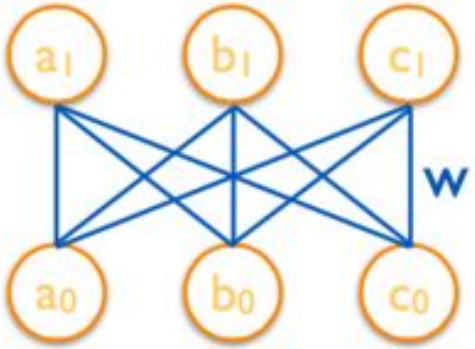


$$\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} \cdot \begin{bmatrix} w_{a,a} & w_{a,b} & w_{a,c} \\ w_{b,a} & w_{b,b} & w_{b,c} \\ w_{c,a} & w_{c,b} & w_{c,c} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix}$$

$$a_1 = \text{relu}(a_1)$$
$$b_1 = \text{relu}(b_1)$$
$$c_1 = \text{relu}(c_1)$$

Instead of us having to type in the calculations one-by-one, we can now use matrix operations i.e. dot product etc. To increase the efficiency.

Matrix ops ReLU with Tensorflow.




```
import tensorflow as tf
```

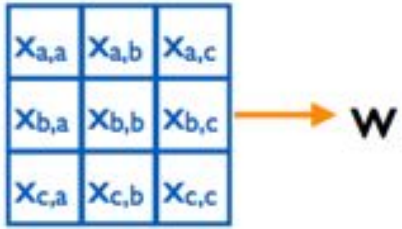
$$\underset{x}{\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix}} \cdot \underset{w}{\begin{bmatrix} w_{a,a} & w_{a,b} & w_{a,c} \\ w_{b,a} & w_{b,b} & w_{b,c} \\ w_{c,a} & w_{c,b} & w_{c,c} \end{bmatrix}} = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix}$$

```
y = tf.matmul(x, w)
```

$$\begin{bmatrix} a_1 \end{bmatrix} = \text{relu}(\begin{bmatrix} a_1 \end{bmatrix})$$
$$\begin{bmatrix} b_1 \end{bmatrix} = \text{relu}(\begin{bmatrix} b_1 \end{bmatrix})$$
$$\begin{bmatrix} c_1 \end{bmatrix} = \text{relu}(\begin{bmatrix} c_1 \end{bmatrix})$$

```
out = tf.nn.relu(y)
```

How to Define Tensors/Variables



Variable(<initial-value>,
name=<optional-name>)

```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
```

Variable stores the state of current execution

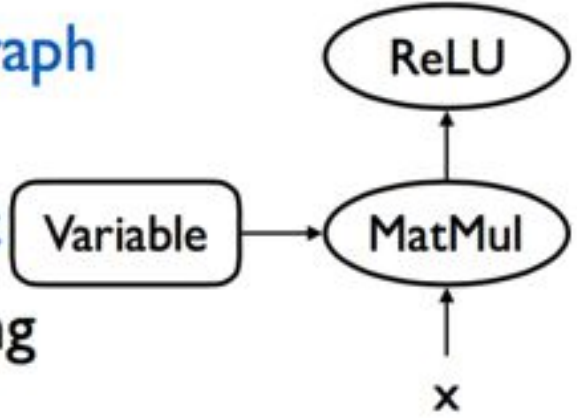
Others are operations

Nodes in the tf Graph

Code so far defines a data flow graph

Each variable corresponds to a node in the graph, not the result

Can be confusing at the beginning

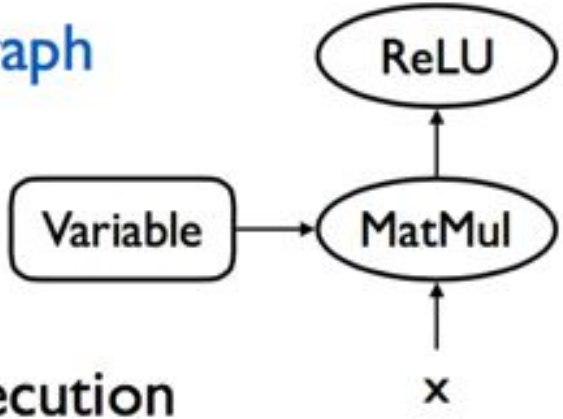



```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
```

Execution

Code so far defines a data flow graph

Needs to specify how we want to execute the graph



Session

Manage resource for graph execution

```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
result = sess.run(relu_out)
```

Session management

Needs to release resource after use

```
sess.close()
```

Common usage

```
with tf.Session() as sess:
    ...
```

Interactive

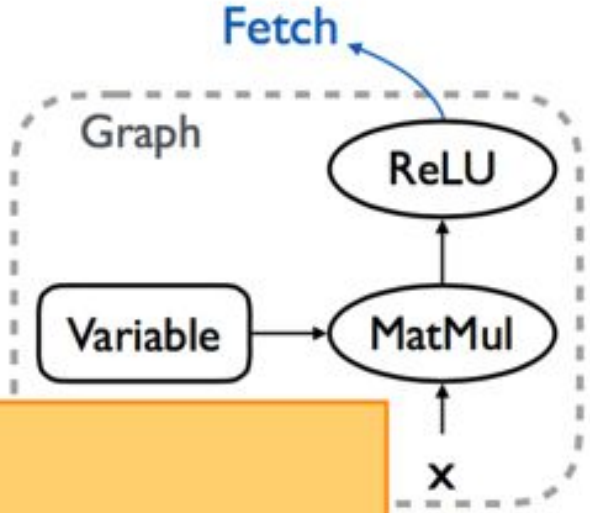
```
sess = InteractiveSession()
```

Fetch

Retrieve content from a node

We have assembled the pipes

Fetch the liquid

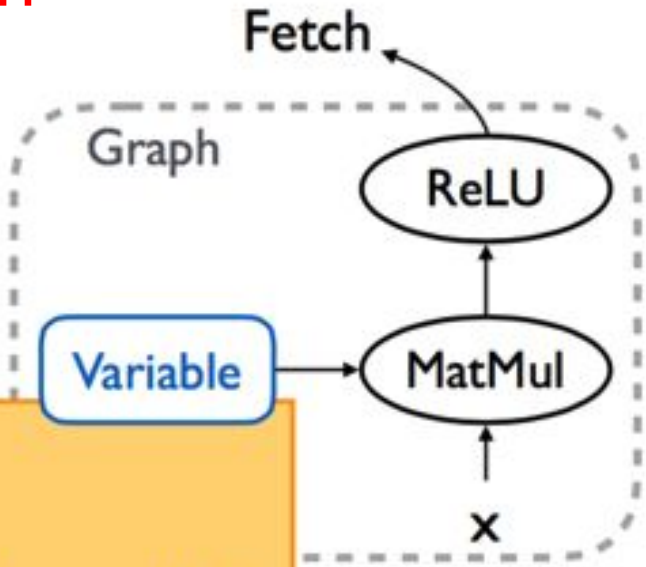



```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
print sess.run(relu_out)
```

Variable Initialization. Execution

Variable is an empty node

Fill in the content of a Variable node




```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out)
```

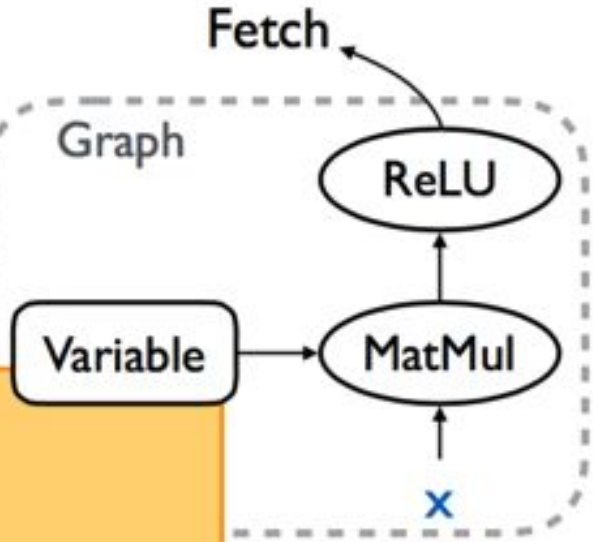
```
tf.Session.run(fetches, feed_dict=None, options=None, run_metadata=None)
```

PlaceHolders Define

How about x?

placeholder(<data type>,
shape=<optional-shape>,
name=<optional-name>)

Its content will be fed

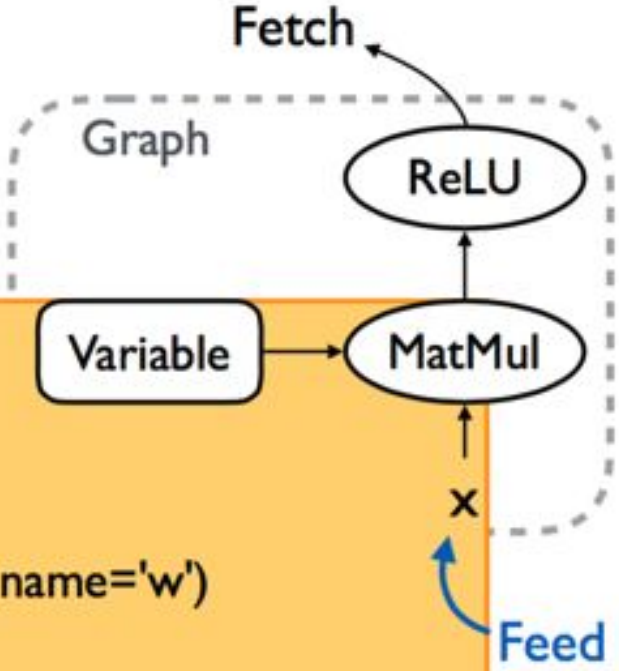



```
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float", [1, 3])
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out)
```

Feed

Use Placeholders: Execution

Pump liquid into the pipe




```
import numpy as np
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float", [1, 3])
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out, feed_dict={x:np.array([[1.0, 2.0, 3.0]])})
```

Define

Prediction

Softmax

Make predictions for n targets that sum to 1

```
import numpy as np
import tensorflow as tf

with tf.Session() as sess:
  x = tf.placeholder("float", [1, 3])
  w = tf.Variable(tf.random_normal([3, 3]), name='w')
  relu_out = tf.nn.relu(tf.matmul(x, w))
  softmax = tf.nn.softmax(relu_out)
  sess.run(tf.initialize_all_variables())
  print sess.run(softmax, feed_dict={x:np.array([[1.0, 2.0, 3.0]])})
```

Define

Learn parameters: Loss

Define loss function

Loss function for softmax

```
softmax_cross_entropy_with_logits(
    logits, labels, name=<optional-name>)
```

```
labels = tf.placeholder("float", [1, 3])
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
    relu_out, labels, name='xentropy')
```

Define

Learn parameters: Optimization

Gradient descent

class GradientDescentOptimizer

GradientDescentOptimizer(learning rate)

learning rate = 0.1

```
labels = tf.placeholder("float", [1, 3])
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
    relu_out, labels, name='xentropy')
optimizer = tf.train.GradientDescentOptimizer(0.1)
train_op = optimizer.minimize(cross_entropy)
sess.run(train_op,
         feed_dict= {x:np.array([[1.0, 2.0, 3.0]]), labels:answer})
```

Iterative update

Gradient descent usually needs more than one step

Run multiple times

```
labels = tf.placeholder("float", [1, 3])
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
    relu_out, labels, name='xentropy')
optimizer = tf.train.GradientDescentOptimizer(0.1)
train_op = optimizer.minimize(cross_entropy)
for step in range(10):
  sess.run(train_op,
           feed_dict= {x:np.array([[1.0, 2.0, 3.0]]), labels:answer})
```

Add biases

Define

Biases initialized to zero

```
…
w = tf.Variable(tf.random_normal([3, 3]))
b = tf.Variable(tf.zeros([1, 3]))
relu_out = tf.nn.relu(tf.matmul(x, w) + b)
softmax_w = tf.Variable(tf.random_normal([3, 3]))
softmax_b = tf.Variable(tf.zeros([1, 3]))
logit = tf.matmul(relu_out, softmax_w) + softmax_b
softmax = tf.nn.softmax(logit)
…
```

Make it deep

Add layers

```
...
x = tf.placeholder("float", [1, 3])
relu_out = x
num_layers = 2
for layer in range(num_layers):
  w = tf.Variable(tf.random_normal([3, 3]))
  b = tf.Variable(tf.zeros([1, 3]))
  relu_out = tf.nn.relu(tf.matmul(relu_out, w) + b)
...
```

Save and load models

tf.train.Saver(…)

Default will associate with all variables

all_variables()

save(sess, save_path, …)

restore(sess, save_path, …)

Replace initialization

That's why we need to run initialization separately

Let’s see how it comes together in code.

TensorFlow Basics # 2

by: Chris Dongjoo Kim

Today

- Quick recap of Tensorflow
- Very brief intro to RNNs and its derivatives
  - how RNN is implemented in Tensorflow
- Very brief intro to CNNs
  - how CNN is implemented in Tensorflow

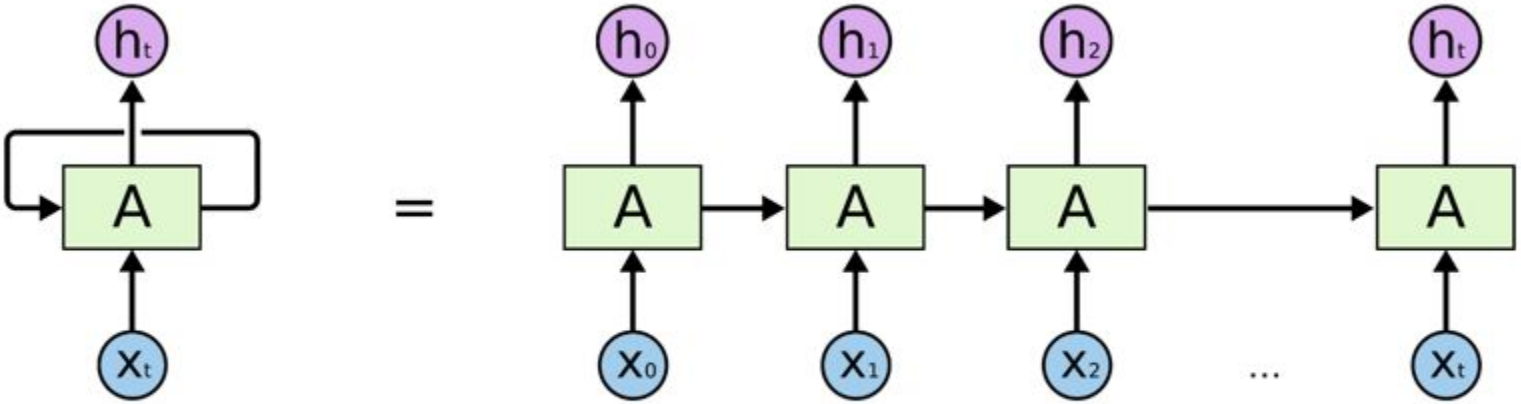
Quick Recap of Tensorflow.

2 Phases.

1. Define Phase:
   - variables: weights, biases,
   - placeholder: input
   - hidden layer: # of layers, activations, type of node
   - cost function and optimization method
2. Execution Phase:
   - create a session to execute the graph.
   - feed in the training data
   - train !

Today, I will do a simple tutorial on how to define hidden layers of RNNs and CNNs.

Vanilla Recurrent Neural Nets.

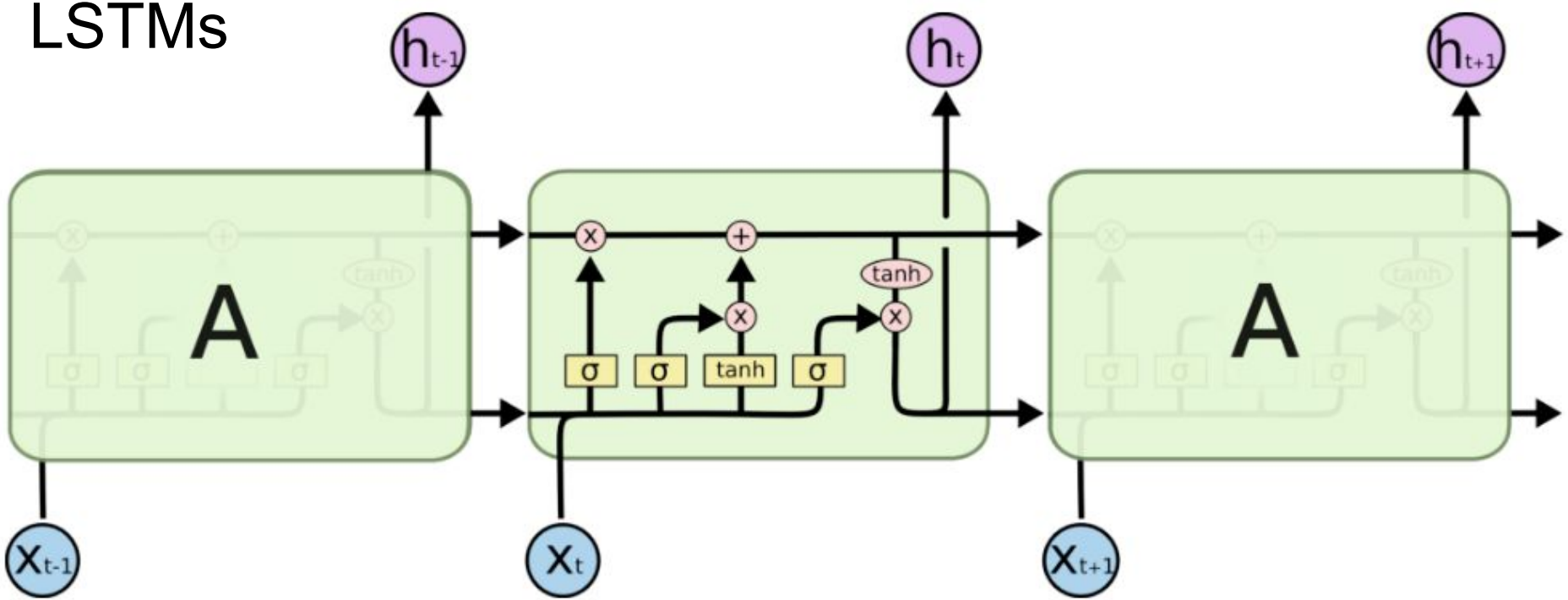


An unrolled recurrent neural network.

- Traditional FFnets and ConvNets, takes a predefined fixed size input and produces a fixed size output. RNNs however are able to work with input of dynamic sizes, making it optimal for many problems in Machine Learning, and especially NLP.

Img from Colah's blog

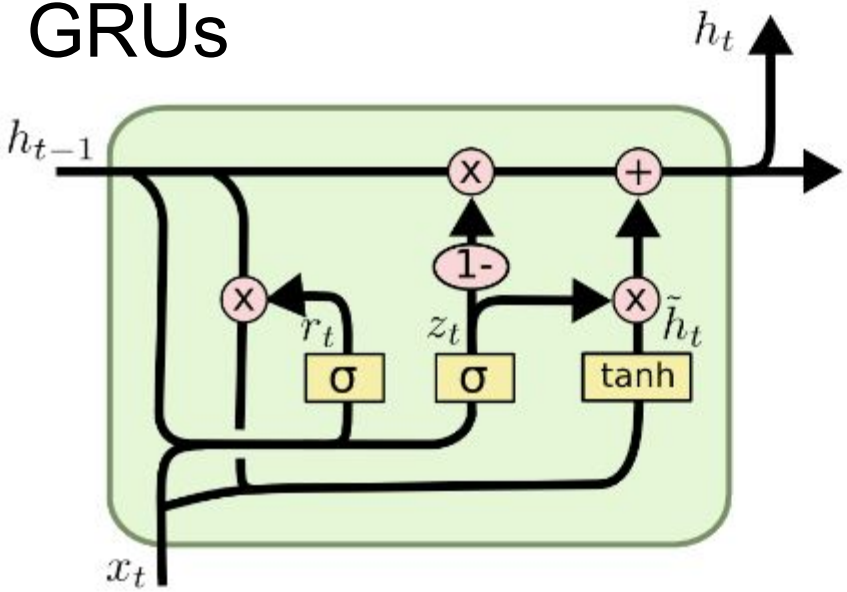
LSTMs



- Uses the idea of forget gate / input gate / filter gate to resolve RNN's vanishing gradient problem which was problematically preventing long-term dependencies.

Img from Colah's blog

GRUs



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- A derivative of LSTMs, where the gates(input/forget) are merged, as well as the cell and hidden states. The resulting model has less parameters, and hence trains a lot faster and easier. It performs as well as LSTMs, hence quite popular at the moment.

Img from Colah's blog

In Tensorflow?

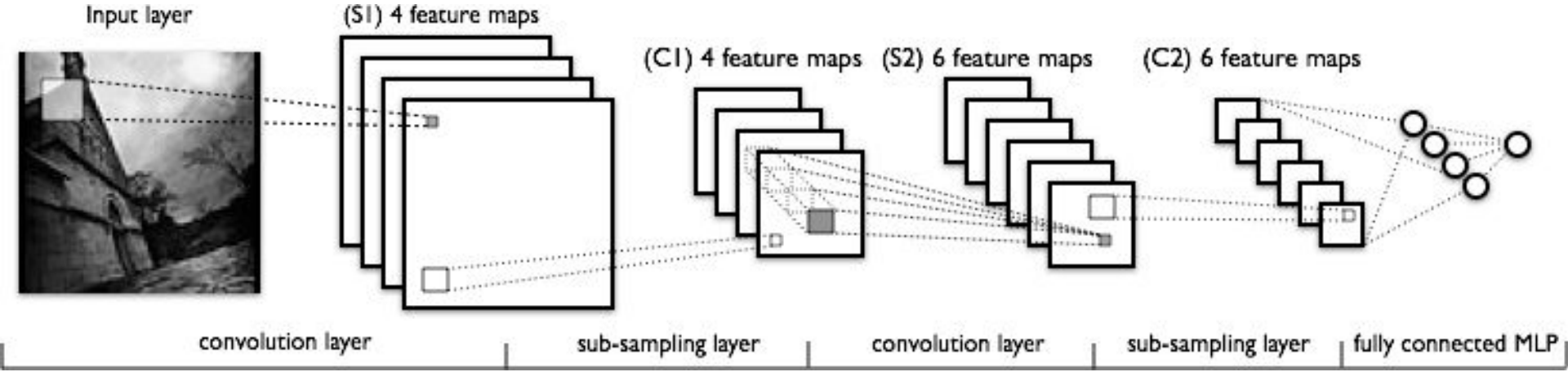
Abstractly, 3 pieces of code are needed to create a RNN layer.

```
1.  cell = rnn_cell.BasicLSTMCell()
    cell = rnn_cell.BasicRNNCell(rnn_size)
    cell = rnn_cell.GRUCell()
2.  rdy_input = tf.split(dim_2_split, input_dim, input)
3.  out,state =rnn.rnn(cell, rdy_input, opt:initial_state,
    opt:dtype, opt:seq_len, opt:scope)
```

Optionally, to stack the rnn layers, use:

```
cell_stk = rnn_cell.MultiRNNCell(cell list from 1.)
```

Brief ConvNets



- Consists of a convolution layer, pooling/sub-sampling layer and a fully connected layer.

Img from Bengio et al

CNN in Tensorflow?

tf.nn.conv2d()
- can also do a separable convolution/ convolution transpose

tf.nn.max_pool()
- you can also do average pooling.