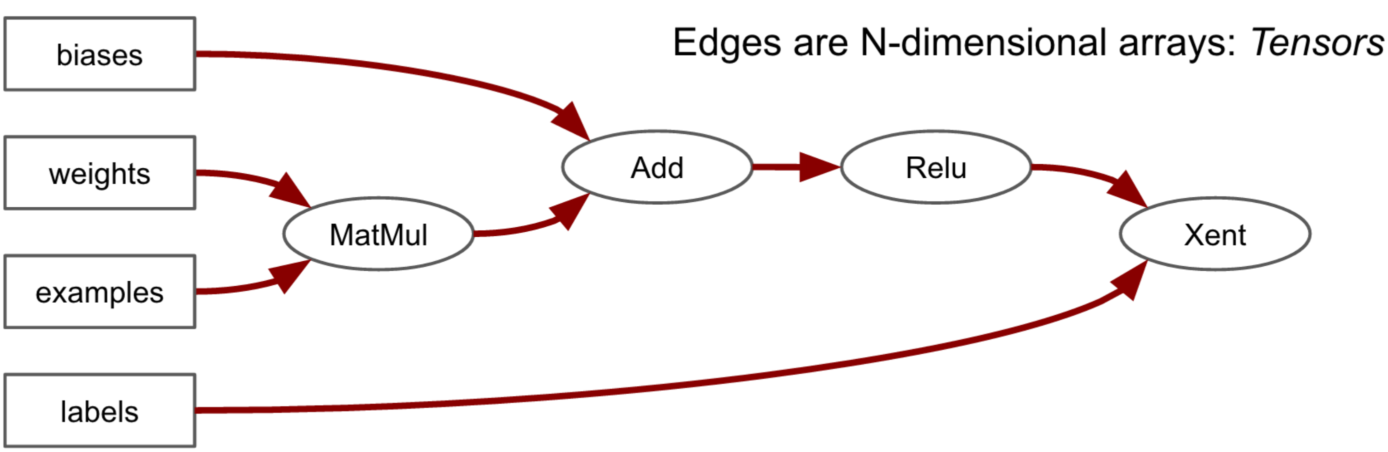
TensorFlow Introduction

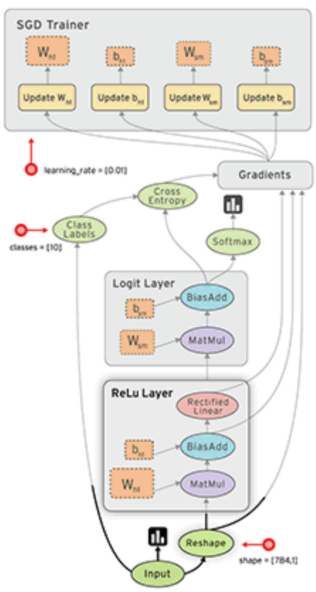
Presentation by Rafal Jozefowicz, Google Brain

TensorFlow is an open source software library for numerical computation using data flow graphs.



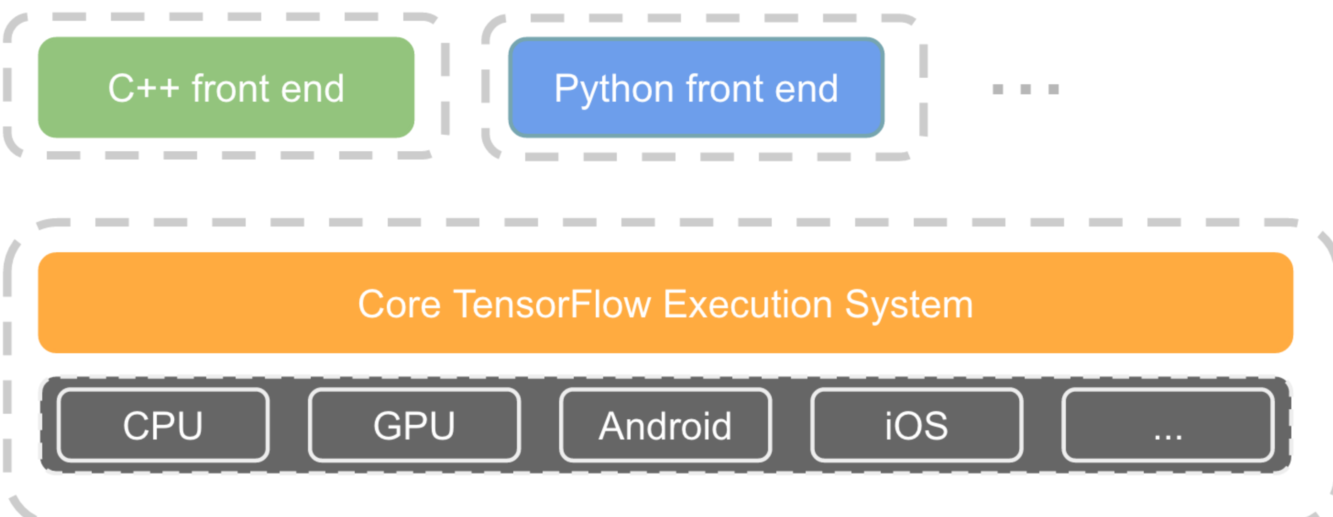
Key Features

- Deep Flexibility
- True Portability
- Connect Research and Production
- Auto-Differentiation
- Language Options
- Maximize Performance



What is a Data Flow Graph?

Data flow graphs describe mathematical computation with a directed graph of nodes & edges. Nodes typically implement mathematical operations, but can also represent endpoints to feed in data, push out results, or read/write persistent variables. Edges describe the input/output relationships between nodes. These data edges carry dynamically-sized multidimensional data arrays, or tensors. The flow of tensors through the graph is where TensorFlow gets its name. Nodes are assigned to computational devices and execute asynchronously and in parallel once all the tensors on their incoming edges becomes available.



Parallel Execution

- Launch graph in a Session
- Request output of some Ops with Run API
- TensorFlow computes set of Ops that must run to compute the requested outputs
- Ops execute, in parallel, as soon as their inputs are available

Follow the installation steps at:

https://www.tensorflow.org/versions/r0.7/get_started/os_setup.html
(https://www.tensorflow.org/versions/r0.7/get_started/os_setup.html)

Main differences between TF and Numpy for numerical processing

Input:

```
a - matrix [N, N]
```

NumPy

```
import numpy as np

# b is now a result of matrix multiplication  a * a
b = np.dot(a, a)
```

TensorFlow - lazy evaluation

```
import tensorflow as tf
# Session is needed to determine what devices are available for computa
tions (e.g. GPUs)
sess = tf.InteractiveSession()

# b is a symbolic representation of matrix multiplication a * a
b_node = tf.matmul(a, a)

# Calling .eval() evaluates the graph using created session. At this po
int the result is a numpy array
b = b_node.eval()

# More general way of evaluation:
b = sess.run(b_node)
```

This additional layer of abstraction allows for scheduling work on different devices ...

TensorFlow - this code now runs on a GPU - the inputs and the outputs are automatically transported between devices

```
with tf.device("/gpu:0"):  # The only difference
  b_node = tf.matmul(a, a)

b = b_node.eval()
```

Or on different machines (not yet open-sourced)

```
with tf.device(<address of a different machine>):  # The only differenc
e
  b_node = tf.matmul(a, a)

b = b_node.eval()
```

It also allows for much greater flexibility of parallelizing work (when you have additional resources)

```
a - matrix [N, N]
b - matrix [N, 1000000]  # a big matrix
```

Naive approach

```
c = tf.matmul(a, b).eval()
```

Multiplication parallelized onto 4 GPUs

```
parts = []
for i in range(4):
  with tf.device("/gpu:%d" % i):
    # Each part is now [N, 250000]
    parts += [tf.matmul(
        a, b[:, 250000*i:250000*(i+1)])]

# Concatenate all the partial results.
c = tf.concat(1, parts).eval()
```

In [1]:
```
import tensorflow as tf
import numpy as np
print tf.__version__

sess = tf.InteractiveSession()
```

```
0.7.0
```

In [2]:
```
a = np.random.random([1000, 2000])
b = np.random.random([2000, 500])
ab_tf = tf.matmul(a, b)
ab_tf
```

Out[2]: `<tf.Tensor 'MatMul:0' shape=(1000, 500) dtype=float64>`

In [87]:
```
# tf.reduce_sum(ab_tf, 0).eval().shape
# tensor = ab_tf / tf.sqrt(tf.reduce_sum(ab_tf * ab_tf, 1, keep_dim
s=True))
# tensor[15, :].eval().shape
ab = sess.run(ab_tf)
ab.shape
```

Out[87]: (1000, 500)

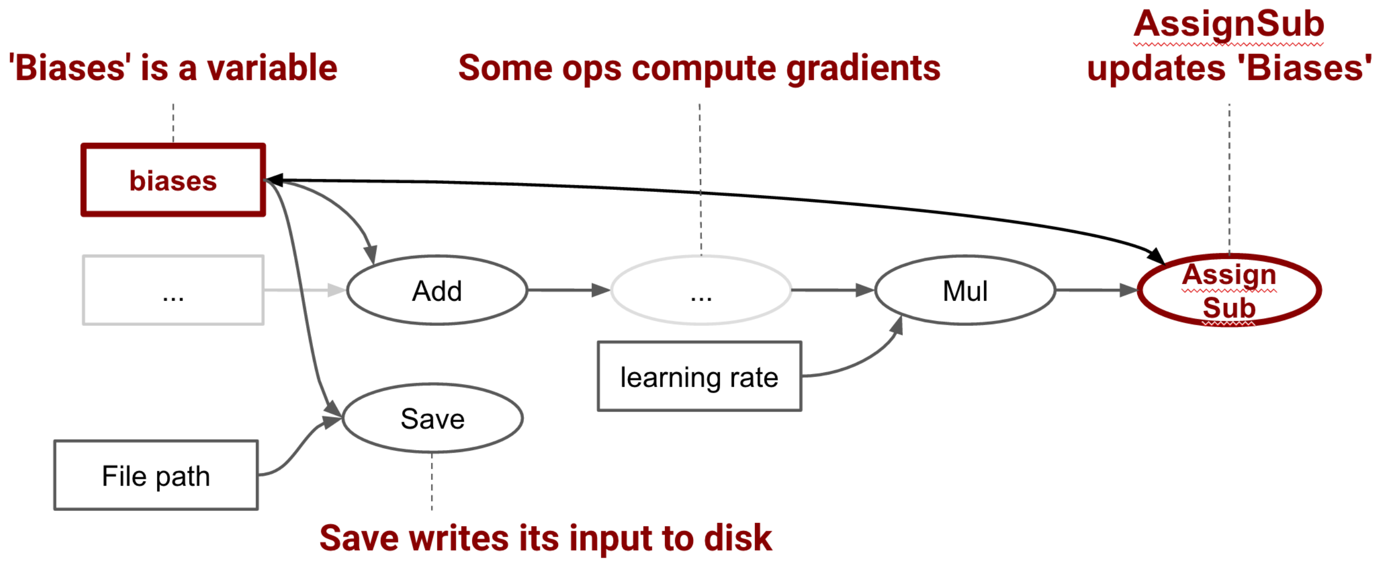
There are hundreds of predefined Ops (and easy to add more)

- Basics: constant, random, placeholder, cast, shape
- Variables: assign, assign_sub, assign_add
- Queues: enqueue, enqueue_batch, dequeue, blocking or not.
- Logical: equal, greater, less, where, min, max, argmin, argmax.
- Tensor computations: all math ops, matmul, determinant, inverse, cholesky.
- Images: encode, decode, crop, pad, resize, color spaces, random perturbations.
- Sparse tensors: represented as 3 tensors.
- ...

And many neural net specific Ops

- Activations: sigmoid, tanh, relu, dropout, ...
- Pooling: avg, max.
- Convolutions: with many options.
- Normalization: local, batch, moving averages.
- Classification: softmax, softmax loss, cross entropy loss, topk.
- Embeddings: lookups/gather, scatter/updates.
- Sampling: candidate sampler (various options), sampling softmax.
- Updates: "fused ops" to speed-up optimizer updates (Adagrad, Momentum.)
- Summaries: Capture information for visualization.

Creating Variables - used for trained parameters



When you train a model, you use variables to hold and update parameters. Variables are in-memory buffers containing tensors. They must be explicitly initialized and can be saved to disk during and after training. You can later restore saved values to exercise or analyse the model.

Create two variables.

```
weights = tf.Variable(tf.random_normal([10, 200], stddev=0.1))
biases = tf.Variable(tf.zeros([200]))
```

Initialization

```
init_op = tf.initialize_all_variables()

# Runs the initialization.
sess.run(init_op)

# You can now evaluate the variables.
print weights.eval()

# Or, e.g., its mean value along second dimension
print tf.reduce_mean(weights, 1).eval()
```

Saving variables

```
# Add ops to save and restore all the variables.
tf.train.Saver()

# Save the variables to disk.
save_path = saver.save(sess, "/tmp/model.ckpt")
```

And then restoring them

```
# Restore variables from disk.
saver.restore(sess, "/tmp/model.ckpt")
```

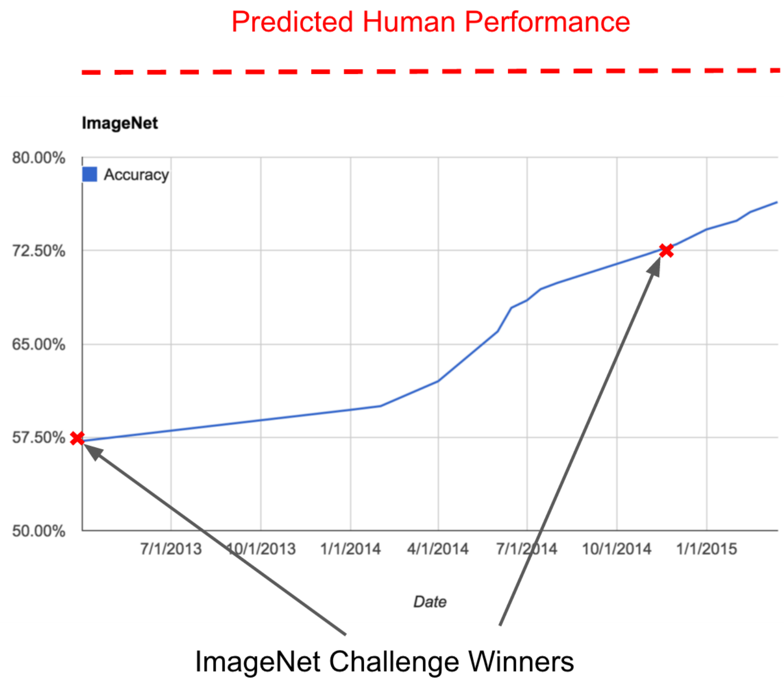
Why put so much effort on a computation engine?

Neural Networks work really well in many different domains

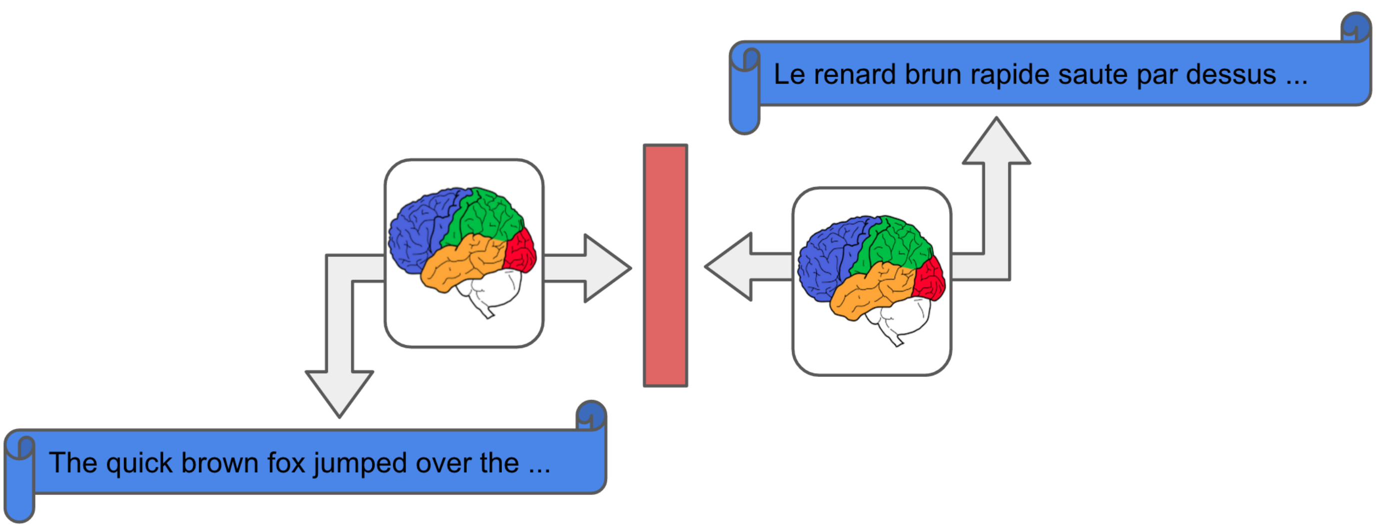
(and can be constructed using our building blocks)

Object Recognition Improvement Over Time

⇨ “cat”



Machine Translation

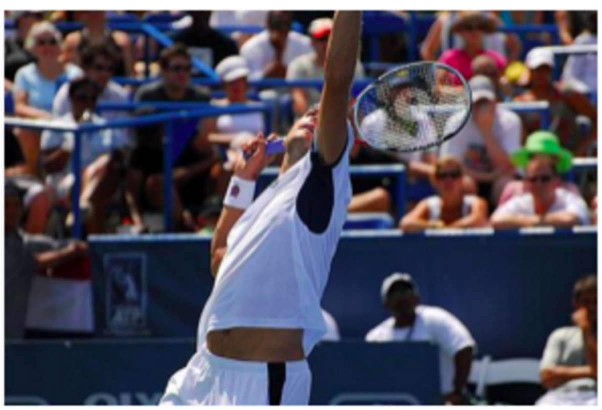


WMT'14	BLEU
State-of-the-art	37.0
Neural Translation Model	**37.3**

Sequence to Sequence Learning with Neural Networks
Ilya Sutskever, Oriol Vinyals, Quoc V. Le (NIPS 2014)

Addressing Rare Word Problems in Neural Translation Models (arxiv.org/abs/1410.8206)
Thang Luong, Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Wojciech Zaremba

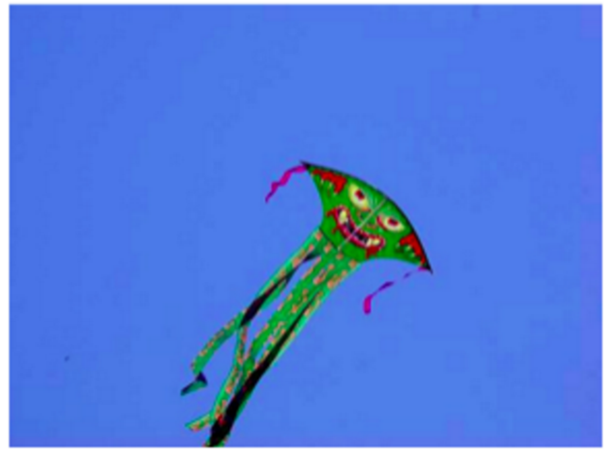
... Or Image Captioning



A man holding a tennis racquet on a tennis court.

Two pizzas sitting on top of a stove top oven

A group of young people playing a game of Frisbee



A man flying through the air while riding a snowboard

Deep Learning for HEP

HIGGS data set

Description

- 11M examples
- 21 basic numerical features + 7 derived based on them
- binary targets (signal vs background)

Can be downloaded from: [https://archive.ics.uci.edu/ml/datasets/HIGGS (https://archive.ics.uci.edu/ml/datasets/HIGGS)](https://archive.ics.uci.edu/ml/datasets/HIGGS)

	AUC		
Technique	Low-level	High-level	Complete
BDT	0.73 (0.01)	0.78 (0.01)	0.81 (0.01)
NN	0.733 (0.007)	0.777 (0.001)	0.816 (0.004)
DN	0.880 (0.001)	0.800 ($<$ 0.001)	0.885 (0.002)

	Discovery significance		
Technique	Low-level	High-level	Complete
NN	2.5σ	3.1σ	3.7σ
DN	4.9σ	3.6σ	5.0σ

Baldi et al., 2014 [Searching for Exotic Particles in High-Energy Physics with Deep Learning (http://arxiv.org/abs/1402.4735)](http://arxiv.org/abs/1402.4735)

Simple Preprocessing

In [4]:
```
data_dir = "/Users/rafalj/higgs/"
```

In [66]:
```
%%time
import numpy as np
import pandas as pd

df = pd.read_csv(data_dir + "HIGGS.csv.gz", header=None)
df = df.astype(np.float32)

train_data = df.values[:-500000]
perm = np.random.permutation(len(train_data))
ptrain_data = train_data[perm]

np.save(data_dir + "higgs_train.npy", ptrain_data[:-500000])
np.save(data_dir + "higgs_valid.npy", ptrain_data[-500000:])
np.save(data_dir + "higgs_test.npy", df.values[-500000:])
```

```
CPU times: user 4min 6s, sys: 33.6 s, total: 4min 39s
Wall time: 4min 48s
```

In [5]:
```
class Dataset(object):
  def __init__(self, x, y):
    self.x = x
    self.y = y

    self.n = x.shape[0]
    self.shuffle()

  def shuffle(self):
    perm = np.arange(self.n)
    np.random.shuffle(perm)
    self.x = self.x[perm]
    self.y = self.y[perm]
    self._next_id = 0

  def next_batch(self, batch_size):
    if self._next_id + batch_size >= self.n:
      self.shuffle()

    cur_id = self._next_id
    self._next_id += batch_size
    return self.x[cur_id:cur_id+batch_size], self.y[cur_id:cur_id+b
atch_size]
```

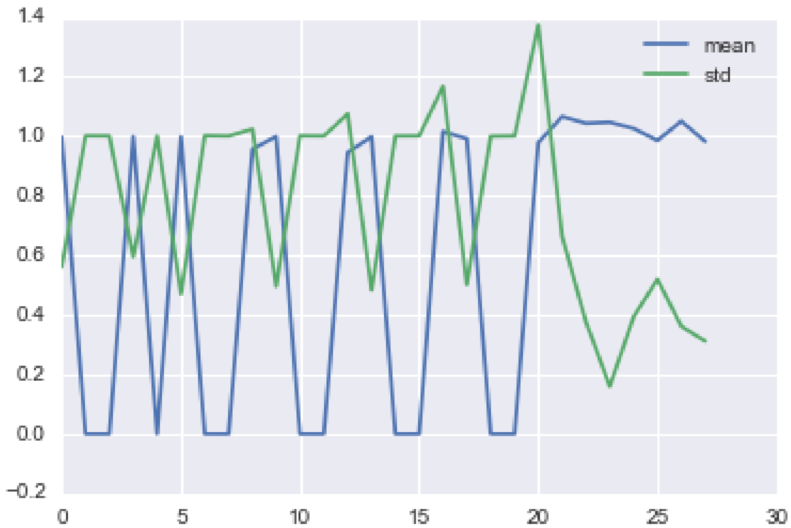
In [7]:
```
%%time
class HiggsDataset(object):
  def __init__(self, data_dir):
    data = np.load(data_dir + "/higgs_train.npy")
    self.train = Dataset(data[:, 1:], data[:, 0])
    data = np.load(data_dir + "/higgs_valid.npy")
    self.valid = Dataset(data[:, 1:], data[:, 0])
    data = np.load(data_dir + "/higgs_test.npy")
    self.test = Dataset(data[:, 1:], data[:, 0])

higgs = HiggsDataset(data_dir)
```

```
CPU times: user 4.16 s, sys: 1.82 s, total: 5.98 s
Wall time: 7.27 s
```

In [9]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

The data appears to be normalized

In [10]:
```
plt.plot(higgs.train.x.mean(0), label="mean")
plt.plot(higgs.train.x.std(0), label="std")
plt.legend();
```



Let's start with the simplest model to get some reasonable baselines.

For the purposes of the presentation let's assume that ROC AUC score on the validation set is all we care about.

But, optimizing AUC is non-trivial because it is non-differentiable scoring function.

Common optimization targets for neural networks are based on modeling the log likelihoods of the targets assuming some simple distributions

- Normal - linear regression
- Multinomial - softmax regression
- **Bernoulli - logistic regression**

In [20]:
```
def linear(x, name, size, bias=True):
    w = tf.get_variable(name + "/W", [x.get_shape()[1], size])
    b = tf.get_variable(name + "/b", [1, size],
                        initializer=tf.zeros_initializer)
    return tf.matmul(x, w) + b

class HiggsLogisticRegression(object):
  def __init__(self, lr=0.1):
    self.x = x = tf.placeholder(tf.float32, [None, 28])
    self.y = tf.placeholder(tf.float32, [None])
    x = linear(x, "regression", 1)
    self.p = tf.nn.sigmoid(x)
    self.loss = loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            x, tf.reshape(self.y, [-1, 1])))
    self.train_op = tf.train.GradientDescentOptimizer(lr).minimize
(loss)
```

In [35]:
```
def train(model, dataset, batch_size=128):
    epoch_size = dataset.n / batch_size
    losses = []

    for i in range(epoch_size):
        train_x, train_y = dataset.next_batch(batch_size)
        loss, _ = sess.run([model.loss, model.train_op],
                           {model.x: train_x, model.y: train_y})
        losses.append(loss)
        if i % (epoch_size / 5) == 5:
            tf.logging.info("%.2f: %.3f", i * 1.0 / epoch_size, np.
mean(losses))
    return np.mean(losses)
```

In [36]:
```
from sklearn.metrics import roc_auc_score

def evaluate(model, dataset, batch_size=1000):
    dataset.shuffle()
    ps = []
    ys = []

    for i in range(dataset.n / batch_size):
        tx, ty = dataset.next_batch(batch_size)
        p = sess.run(model.p, {model.x: tx, model.y: ty})
        ps.append(p)
        ys.append(ty)
    ps = np.concatenate(ps).ravel()
    ys = np.concatenate(ys).ravel()
    return roc_auc_score(ys, ps)
```

In [43]:
```
with tf.variable_scope("model1", reuse=True):
    model = HiggsLogisticRegression()
tf.initialize_all_variables().run()
```

In [44]:

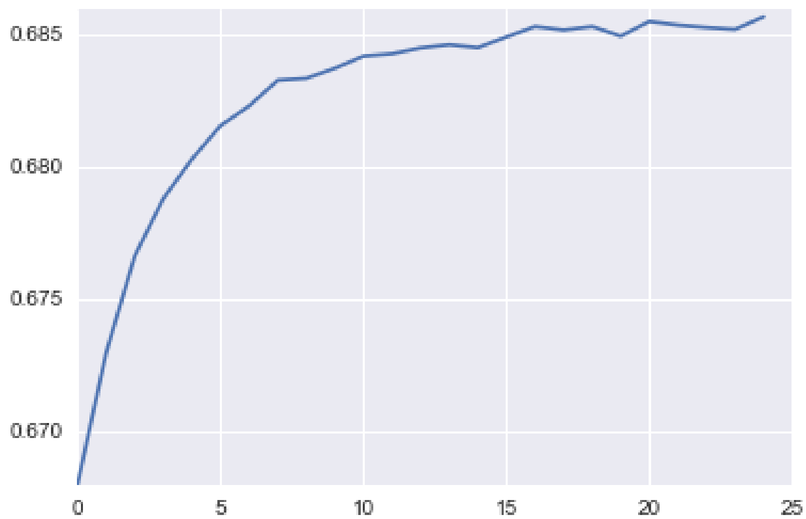
```
%%time
logistic_aucs = []
for i in range(25):
    sys.stdout.write("EPOCH: %d " % (i + 1))
    train(model, higgs.train, 8 * 1024)
    valid_auc = evaluate(model, higgs.valid, 20000)
    print "VALID AUC: %.3f" % valid_auc
    logistic_aucs += [valid_auc]
```

```
EPOCH: 1 VALID AUC: 0.668
EPOCH: 2 VALID AUC: 0.673
EPOCH: 3 VALID AUC: 0.677
EPOCH: 4 VALID AUC: 0.679
EPOCH: 5 VALID AUC: 0.680
EPOCH: 6 VALID AUC: 0.682
EPOCH: 7 VALID AUC: 0.682
EPOCH: 8 VALID AUC: 0.683
EPOCH: 9 VALID AUC: 0.683
EPOCH: 10 VALID AUC: 0.684
EPOCH: 11 VALID AUC: 0.684
EPOCH: 12 VALID AUC: 0.684
EPOCH: 13 VALID AUC: 0.684
EPOCH: 14 VALID AUC: 0.685
EPOCH: 15 VALID AUC: 0.685
EPOCH: 16 VALID AUC: 0.685
EPOCH: 17 VALID AUC: 0.685
EPOCH: 18 VALID AUC: 0.685
EPOCH: 19 VALID AUC: 0.685
EPOCH: 20 VALID AUC: 0.685
EPOCH: 21 VALID AUC: 0.685
EPOCH: 22 VALID AUC: 0.685
EPOCH: 23 VALID AUC: 0.685
EPOCH: 24 VALID AUC: 0.685
EPOCH: 25 VALID AUC: 0.686
CPU times: user 3min 44s, sys: 3min 4s, total: 6min 48s
Wall time: 2h 35min 26s
```

The AUC flattens out at 0.686

In [46]:
```
plt.plot(logistic_aucs)
```

Out[46]: [<matplotlib.lines.Line2D at 0x116dd3e50>]

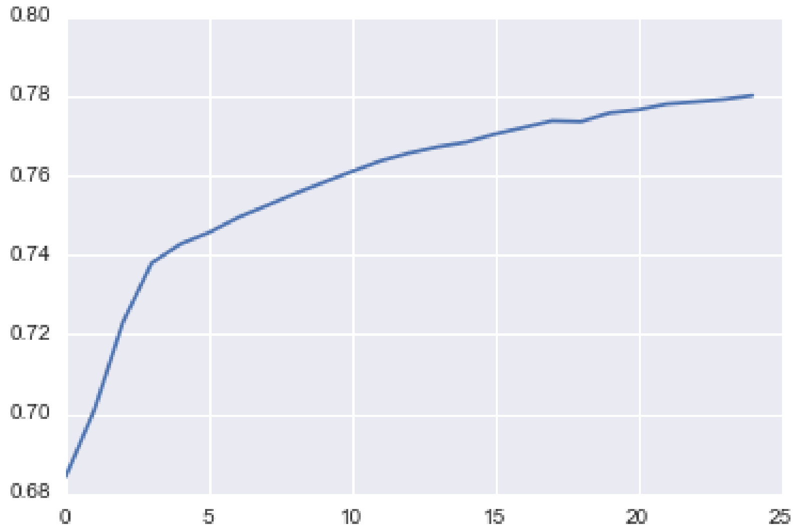


In [51]:
```
class HiggsNeuralNetwork(object):
  def __init__(self, num_layers=1, size=100, lr=0.1):
    self.x = x = tf.placeholder(tf.float32, [None, 28])
    self.y = tf.placeholder(tf.float32, [None])
    for i in range(num_layers):
        x = tf.nn.relu(linear(x, "linear_%d" % i, size))
    x = linear(x, "regression", 1)
    self.p = tf.nn.sigmoid(x)
    self.loss = loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            x, tf.reshape(self.y, [-1, 1])))
    self.train_op = tf.train.GradientDescentOptimizer(lr).minimize
(loss)

with tf.variable_scope("model2", reuse=True):
    model = HiggsNeuralNetwork()

tf.initialize_all_variables().run()
```

In [52]:
```
%%time
nn_aucs = []
for i in range(25):
    sys.stdout.write("EPOCH: %d " % (i + 1))
    train(model, higgs.train, 8 * 1024)
    valid_auc = evaluate(model, higgs.valid, 20000)
    print "VALID AUC: %.3f" % valid_auc
    nn_aucs += [valid_auc]
```

```
EPOCH: 1 VALID AUC: 0.684
EPOCH: 2 VALID AUC: 0.701
EPOCH: 3 VALID AUC: 0.723
EPOCH: 4 VALID AUC: 0.738
EPOCH: 5 VALID AUC: 0.743
EPOCH: 6 VALID AUC: 0.746
EPOCH: 7 VALID AUC: 0.749
EPOCH: 8 VALID AUC: 0.752
EPOCH: 9 VALID AUC: 0.755
EPOCH: 10 VALID AUC: 0.758
EPOCH: 11 VALID AUC: 0.761
EPOCH: 12 VALID AUC: 0.764
EPOCH: 13 VALID AUC: 0.766
EPOCH: 14 VALID AUC: 0.767
EPOCH: 15 VALID AUC: 0.768
EPOCH: 16 VALID AUC: 0.770
EPOCH: 17 VALID AUC: 0.772
EPOCH: 18 VALID AUC: 0.774
EPOCH: 19 VALID AUC: 0.774
EPOCH: 20 VALID AUC: 0.776
EPOCH: 21 VALID AUC: 0.776
EPOCH: 22 VALID AUC: 0.778
EPOCH: 23 VALID AUC: 0.778
EPOCH: 24 VALID AUC: 0.779
EPOCH: 25 VALID AUC: 0.780
CPU times: user 17min 8s, sys: 8min 58s, total: 26min 6s
Wall time: 11min 29s
```

In [54]:
```
plt.plot(nn_aucs);
```



Batch Normalization

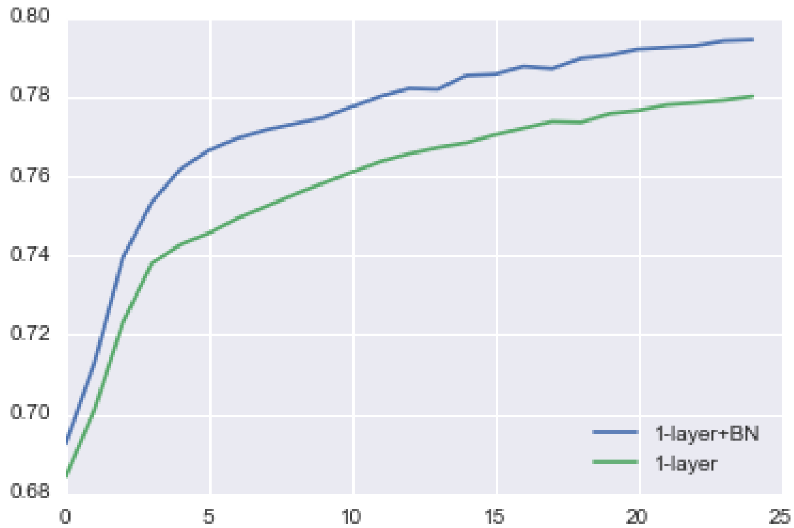
Ioffe & Szegedi, 2015 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (http://arxiv.org/abs/1502.03167)](http://arxiv.org/abs/1502.03167)

In [55]:
```
def batch_norm(x, name):
    mean, var = tf.nn.moments(x, [0])
    normalized_x = (x - mean) * tf.rsqrt(var + 1e-8)
    gamma = tf.get_variable(name + "/gamma", [x.get_shape()[-1]],
                            initializer=tf.constant_initializer(1.
0))
    beta = tf.get_variable(name + "/beta", [x.get_shape()[-1]])
    return gamma * normalized_x + beta

class HiggsBNNeuralNetwork(object):
  def __init__(self, num_layers=1, size=100, lr=0.1):
    self.x = x = tf.placeholder(tf.float32, [None, 28])
    self.y = tf.placeholder(tf.float32, [None])
    for i in range(num_layers):
        x = tf.nn.relu(batch_norm(linear(x, "linear_%d" % i, size),
"bn_%d" % i))
    x = linear(x, "regression", 1)
    self.p = tf.nn.sigmoid(x)
    self.loss = loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            x, tf.reshape(self.y, [-1, 1])))
    self.train_op = tf.train.GradientDescentOptimizer(lr).minimize
(loss)

with tf.variable_scope("model4"):
    model = HiggsBNNeuralNetwork()

tf.initialize_all_variables().run()
```

In [56]:
```
%%time
bn_aucs = []
for i in range(25):
    sys.stdout.write("EPOCH: %d " % (i + 1))
    train(model, higgs.train, 8 * 1024)
    valid_auc = evaluate(model, higgs.valid, 20000)
    print "VALID AUC: %.3f" % valid_auc
    bn_aucs += [valid_auc]
```

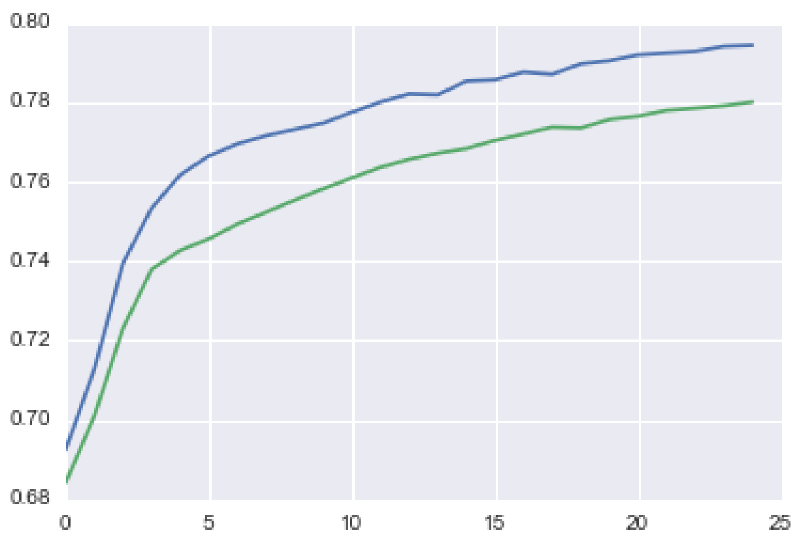
```
EPOCH: 1 VALID AUC: 0.693
EPOCH: 2 VALID AUC: 0.713
EPOCH: 3 VALID AUC: 0.739
EPOCH: 4 VALID AUC: 0.753
EPOCH: 5 VALID AUC: 0.762
EPOCH: 6 VALID AUC: 0.766
EPOCH: 7 VALID AUC: 0.770
EPOCH: 8 VALID AUC: 0.772
EPOCH: 9 VALID AUC: 0.773
EPOCH: 10 VALID AUC: 0.775
EPOCH: 11 VALID AUC: 0.777
EPOCH: 12 VALID AUC: 0.780
EPOCH: 13 VALID AUC: 0.782
EPOCH: 14 VALID AUC: 0.782
EPOCH: 15 VALID AUC: 0.785
EPOCH: 16 VALID AUC: 0.786
EPOCH: 17 VALID AUC: 0.788
EPOCH: 18 VALID AUC: 0.787
EPOCH: 19 VALID AUC: 0.790
EPOCH: 20 VALID AUC: 0.790
EPOCH: 21 VALID AUC: 0.792
EPOCH: 22 VALID AUC: 0.792
EPOCH: 23 VALID AUC: 0.793
EPOCH: 24 VALID AUC: 0.794
EPOCH: 25 VALID AUC: 0.794
CPU times: user 1h 16min 2s, sys: 13min 49s, total: 1h 29min 51s
Wall time: 3h 5min 10s
```

In [62]:
```
plt.plot(bn_aucs, label="1-layer+BN")
plt.plot(nn_aucs, label="1-layer")
plt.legend(loc="lower right");
```



In [58]:
```
plt.plot(bn_aucs, label="1-layer+BN")
plt.plot(nn_aucs, label="1-layer")
```

Out[58]:
```
[<matplotlib.lines.Line2D at 0x118579a10>]
```



In [59]:
```
with tf.variable_scope("model5"):
    model = HiggsBNNeuralNetwork(num_layers=3)

tf.initialize_all_variables().run()

bn3_aucs = []
for i in range(25):
    sys.stdout.write("EPOCH: %d " % (i + 1))
    train(model, higgs.train, 8 * 1024)
    valid_auc = evaluate(model, higgs.valid, 20000)
    print "VALID AUC: %.3f" % valid_auc
    bn3_aucs += [valid_auc]
```

```
EPOCH: 1 VALID AUC: 0.727
EPOCH: 2 VALID AUC: 0.758
EPOCH: 3 VALID AUC: 0.774
EPOCH: 4 VALID AUC: 0.785
EPOCH: 5 VALID AUC: 0.792
EPOCH: 6 VALID AUC: 0.798
EPOCH: 7 VALID AUC: 0.803
EPOCH: 8 VALID AUC: 0.808
EPOCH: 9 VALID AUC: 0.811
EPOCH: 10 VALID AUC: 0.815
EPOCH: 11 VALID AUC: 0.818
EPOCH: 12 VALID AUC: 0.820
EPOCH: 13 VALID AUC: 0.822
EPOCH: 14 VALID AUC: 0.824
EPOCH: 15 VALID AUC: 0.826
EPOCH: 16 VALID AUC: 0.827
EPOCH: 17 VALID AUC: 0.828
EPOCH: 18 VALID AUC: 0.829
EPOCH: 19 VALID AUC: 0.830
EPOCH: 20 VALID AUC: 0.830
EPOCH: 21 VALID AUC: 0.831
EPOCH: 22 VALID AUC: 0.832
EPOCH: 23 VALID AUC: 0.833
EPOCH: 24 VALID AUC: 0.833
EPOCH: 25 VALID AUC: 0.834
```

In [ ]:
```
evaluate(model, higgs.train, 20000)
# validation loss: 0.834
# training loss: 0.857
```

Pretty much no overfitting!

Now lets switch to Adam optimizer without making any other changes

Kingma & Ba, 2014 Adam: A Method for Stochastic Optimization (http://arxiv.org/abs/1412.6980)

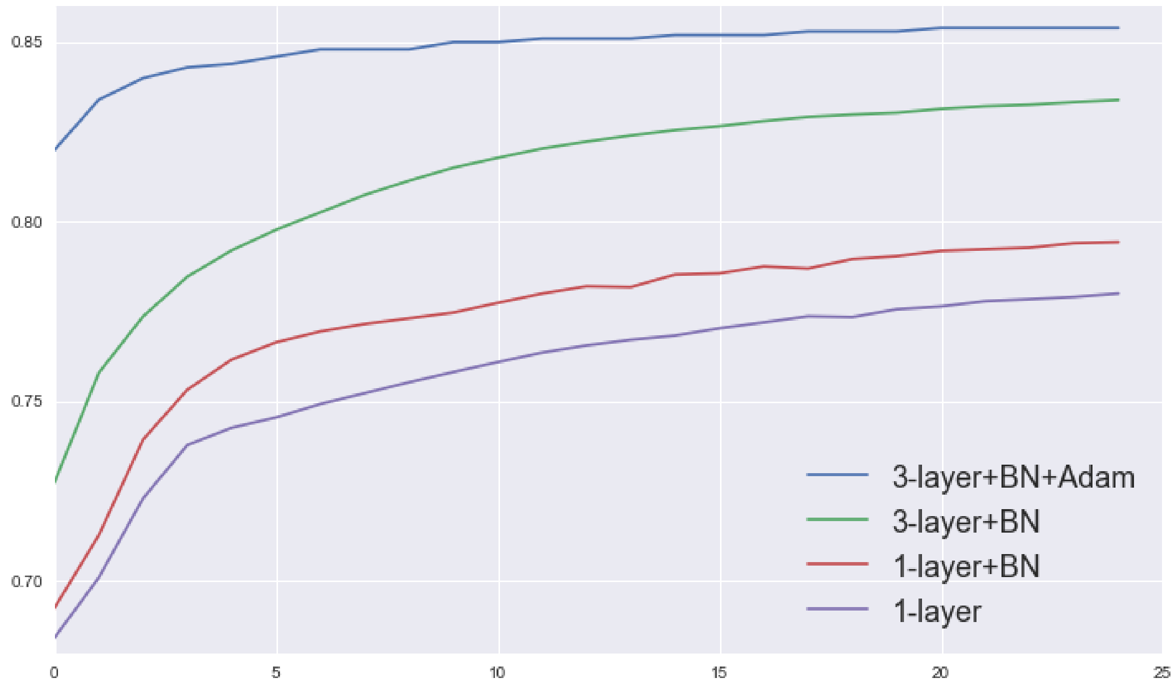
In [ ]:

```
class HiggsAdamBNNeuralNetwork(object):
  def __init__(self, num_layers=1, size=100, lr=1e-3):
    self.x = x = tf.placeholder(tf.float32, [None, 28])
    self.y = tf.placeholder(tf.float32, [None])
    for i in range(num_layers):
        x = tf.nn.relu(batch_norm(linear(x, "linear_%d" % i, size),
"bn_%d" % i))
    x = linear(x, "regression", 1)
    self.p = tf.nn.sigmoid(x)
    self.loss = loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            x, tf.reshape(self.y, [-1, 1])))
    self.train_op = tf.train.AdamOptimizer(lr).minimize(loss)  # ch
ange!

with tf.variable_scope("model6"):
    model = HiggsBNNeuralNetwork()

tf.initialize_all_variables().run()
```

In [64]:

```
# Evaluated this model on my GPU to save some time.
# It takes about 7.5min on Tesla K40 without making
# a single change in code.

abn3_aucs = [
    0.820, 0.834, 0.840, 0.843, 0.844,
    0.846, 0.848, 0.848, 0.848, 0.850,
    0.850, 0.851, 0.851, 0.851, 0.852,
    0.852, 0.852, 0.853, 0.853, 0.853,
    0.854, 0.854, 0.854, 0.854, 0.854
]

# Train AUC: 0.857
```

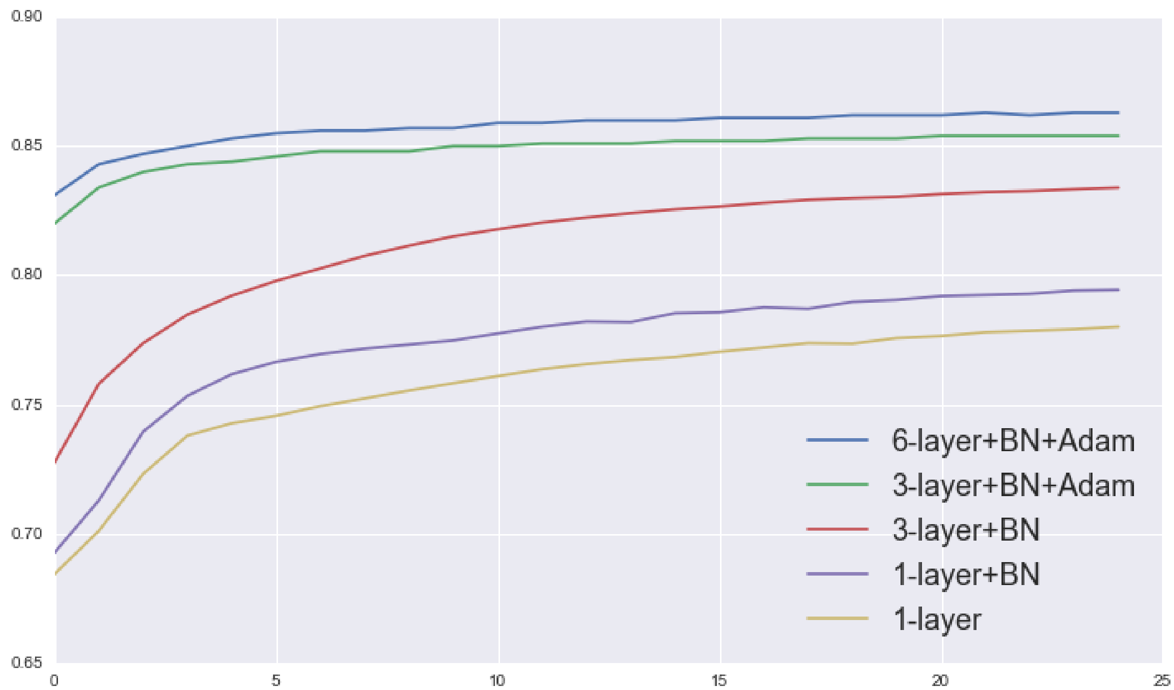
In [68]:
```
plt.figure(figsize=(12, 7))
plt.plot(abn3_aucs, label="3-layer+BN+Adam")
plt.plot(bn3_aucs, label="3-layer+BN")
plt.plot(bn_aucs, label="1-layer+BN")
plt.plot(nn_aucs, label="1-layer")
plt.legend(loc="lower right", fontsize=18);
```



And let's try a 6-layer network....

In [69]:
```
abn6_aucs = [
    0.831, 0.843, 0.847, 0.850, 0.853,
    0.855, 0.856, 0.856, 0.857, 0.857,
    0.859, 0.859, 0.860, 0.860, 0.860,
    0.861, 0.861, 0.861, 0.862, 0.862,
    0.862, 0.863, 0.862, 0.863, 0.863
]

# training loss: 0.867
```

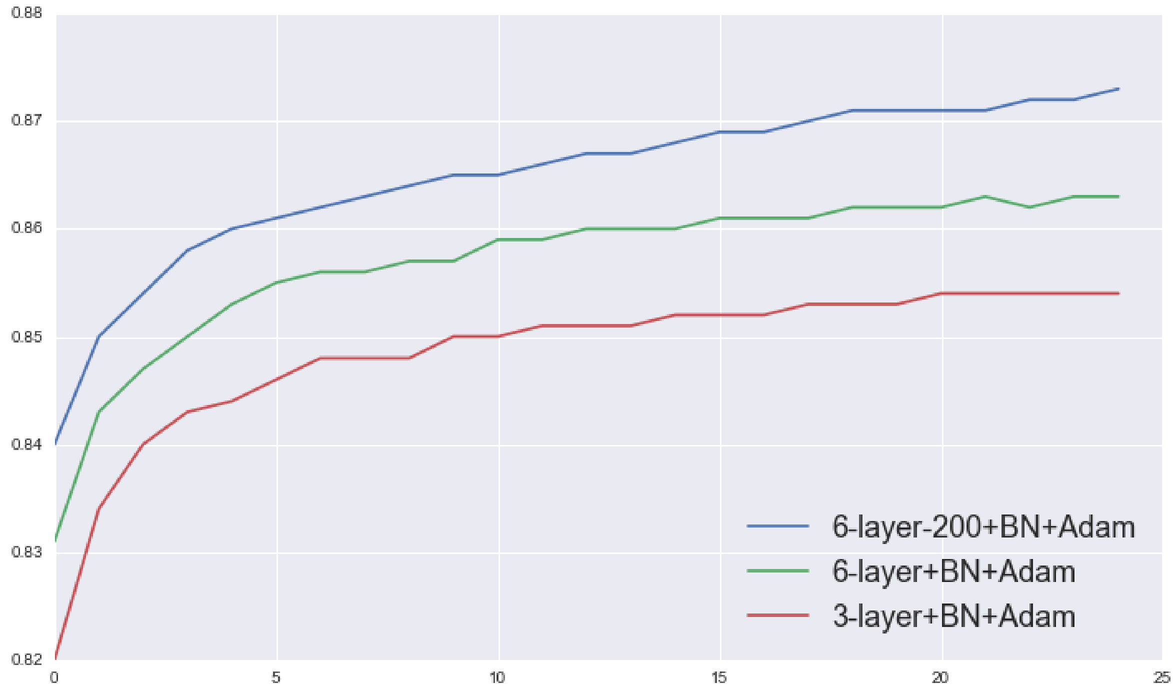
In [70]:
```
plt.figure(figsize=(12, 7))
plt.plot(abn6_aucs, label="6-layer+BN+Adam")
plt.plot(abn3_aucs, label="3-layer+BN+Adam")
plt.plot(bn3_aucs, label="3-layer+BN")
plt.plot(bn_aucs, label="1-layer+BN")
plt.plot(nn_aucs, label="1-layer")
plt.legend(loc="lower right", fontsize=18);
```



The improvements are decreasing, let's add more units!

In [72]:
```
abn6_200_aucs = [
    0.840, 0.850, 0.854, 0.858, 0.860,
    0.861, 0.862, 0.863, 0.864, 0.865,
    0.865, 0.866, 0.867, 0.867, 0.868,
    0.869, 0.869, 0.870, 0.871, 0.871,
    0.871, 0.871, 0.872, 0.872, 0.873
]

# training loss: 0.881
```

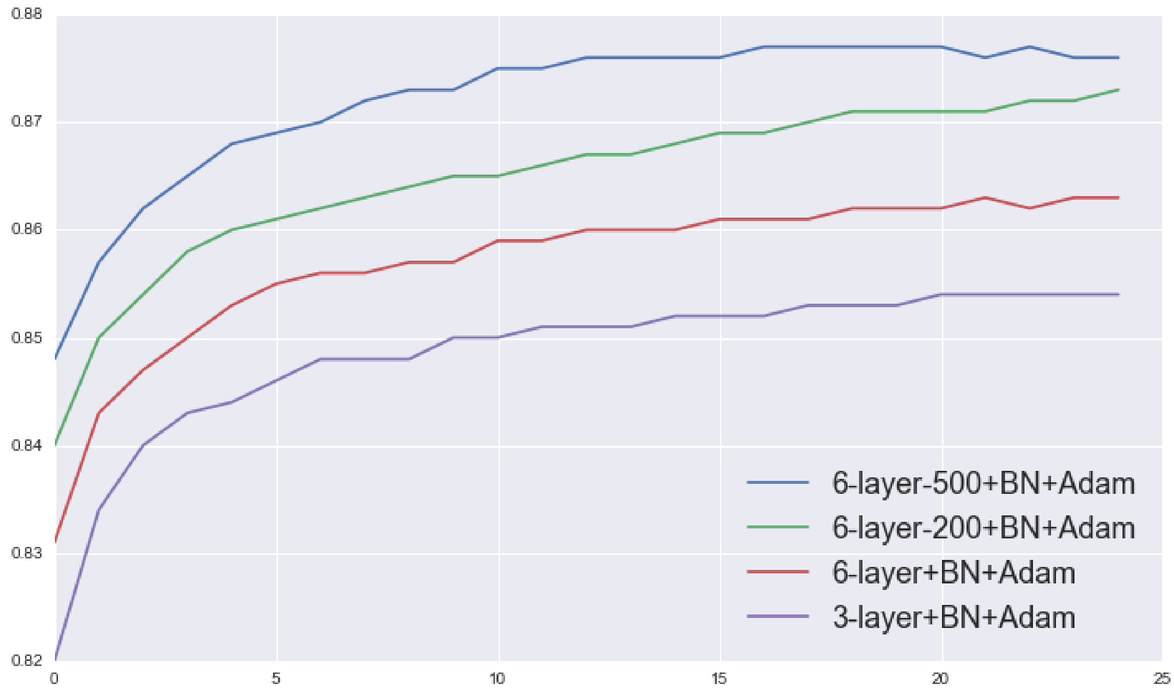
In [73]:
```
plt.figure(figsize=(12, 7))
plt.plot(abn6_200_aucs, label="6-layer-200+BN+Adam")
plt.plot(abn6_aucs, label="6-layer+BN+Adam")
plt.plot(abn3_aucs, label="3-layer+BN+Adam")
plt.legend(loc="lower right", fontsize=18);
```



Increasing the size even further causes the model to heavily overfit

In [76]:
```
abn6_500_aucs = [
    0.848, 0.857, 0.862, 0.865, 0.868,
    0.869, 0.870, 0.872, 0.873, 0.873,
    0.875, 0.875, 0.876, 0.876, 0.876,
    0.876, 0.877, 0.877, 0.877, 0.877,
    0.877, 0.876, 0.877, 0.876, 0.876
]

# Training loss: 0.903
```

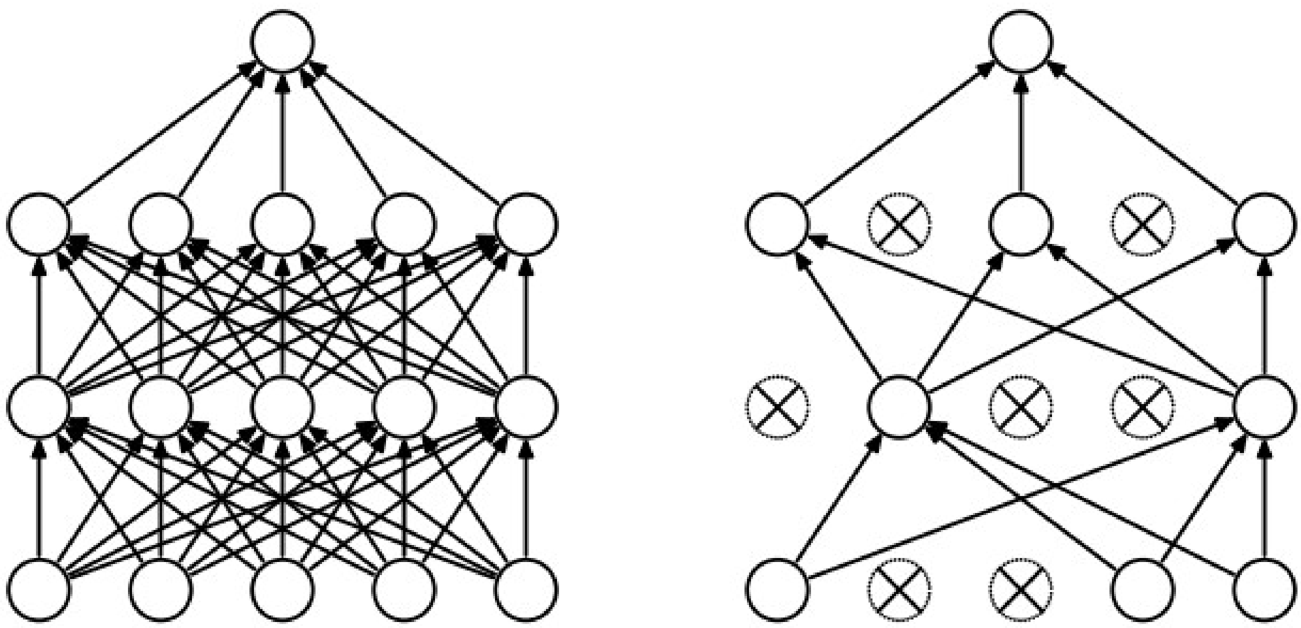
In [77]:
```
plt.figure(figsize=(12, 7))
plt.plot(abn6_500_aucs, label="6-layer-500+BN+Adam")
plt.plot(abn6_200_aucs, label="6-layer-200+BN+Adam")
plt.plot(abn6_aucs, label="6-layer+BN+Adam")
plt.plot(abn3_aucs, label="3-layer+BN+Adam")
plt.legend(loc="lower right", fontsize=18);
```



This is great!

Dropout to the rescue

Hinton et al., 2012 [Improving neural networks by preventing co-adaptation of feature detectors (http://arxiv.org/abs/1207.0580)](http://arxiv.org/abs/1207.0580)



(a) Standard Neural Net (b) After applying dropout.

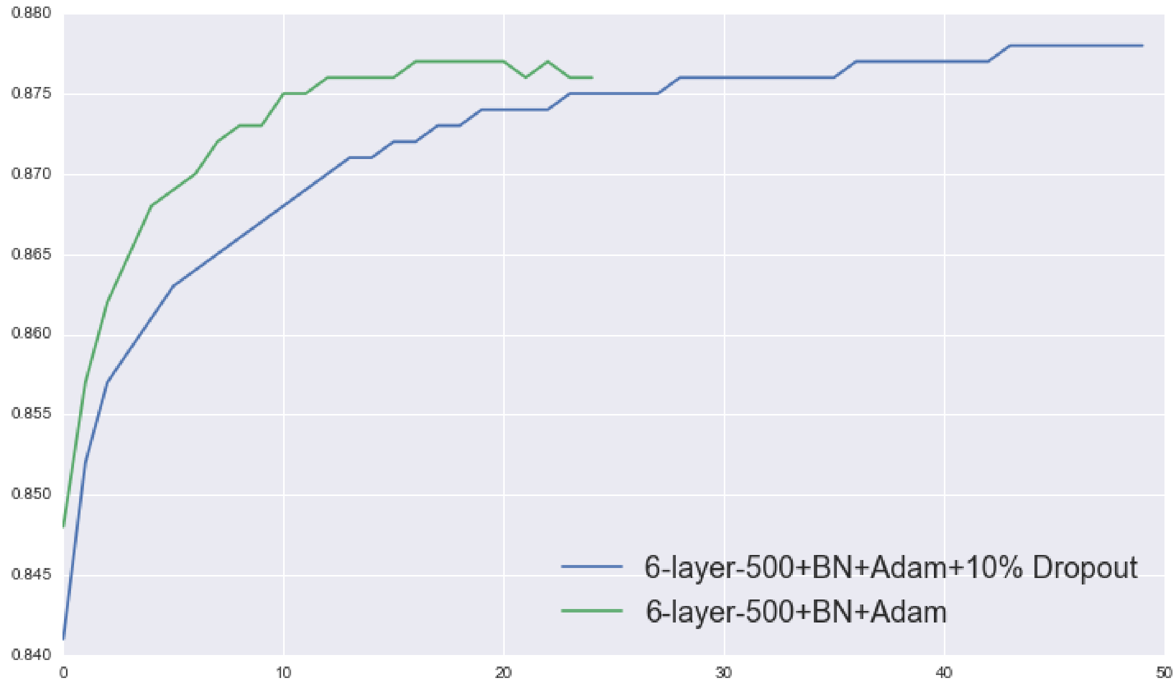
In [78]:
```
class HiggsAdamBNDropout(object):
  def __init__(self, num_layers=1, size=100, lr=1e-3, keep_prob=1.
0):
    self.x = x = tf.placeholder(tf.float32, [None, 28])
    self.y = tf.placeholder(tf.float32, [None])
    for i in range(num_layers):
        x = tf.nn.relu(batch_norm(linear(x, "linear_%d" % i, size),
"bn_%d" % i))
        if keep_prob < 1.0:  # The only addition!
          x = tf.nn.dropout(x, keep_prob)

    x = linear(x, "regression", 1)
    self.p = tf.nn.sigmoid(x)
    self.loss = loss = tf.reduce_mean(
        tf.nn.sigmoid_cross_entropy_with_logits(
            x, tf.reshape(self.y, [-1, 1])))
    self.train_op = tf.train.AdamOptimizer(lr).minimize(loss)
```

In [ ]:
```
with tf.variable_scope("model11") as vs:
    model = HiggsAdamBNDropout(num_layers=6, size=500, keep_prob=0.
9)
    # Sharing variables is useful here!
    vs.reuse_variables()
    emodel = HiggsAdamBNDropout(num_layers=6, size=500)

tf.initialize_all_variables().run()
```

In [80]:
```
dabn6_500_aucs = [
    0.841, 0.852, 0.857, 0.859, 0.861,
    0.863, 0.864, 0.865, 0.866, 0.867,
    0.868, 0.869, 0.870, 0.871, 0.871,
    0.872, 0.872, 0.873, 0.873, 0.874,
    0.874, 0.874, 0.874, 0.875, 0.875,
    0.875, 0.875, 0.875, 0.876, 0.876,
    0.876, 0.876, 0.876, 0.876, 0.876,
    0.876, 0.877, 0.877, 0.877, 0.877,
    0.877, 0.877, 0.877, 0.878, 0.878,
    0.878, 0.878, 0.878, 0.878, 0.878
]
# training loss: 0.878
```

Zero overfitting with only 10% dropout!

In [84]:
```
plt.figure(figsize=(12, 7))
plt.plot(dabn6_500_aucs, label="6-layer-500+BN+Adam+10% Dropout")
plt.plot(abn6_500_aucs, label="6-layer-500+BN+Adam")
plt.legend(loc="lower right", fontsize=18);
```



More easy things to try:

Increased learning rate (with batch normalization, on ImageNet authors were able to use 30x the original learning rate)

Decreased learning rate towards the end of the training - finetuning

Averaging parameters of the models (between current and e.g. previous epoch or a few previous iterations) - this helps on image models

Averaging predictions - this almost always helps. If you optimize likelihood of some distributions, replacing predictions with an average of a few models is still an estimator but has typically lower variance

Reducing batch size and training for more time once we find good parameters (potentially also increasing dropout rate)

Or ... any tuning of the parameters - we didn't do almost any!

Deeper networks!

Thank you for attention!