Autoencoders Internals

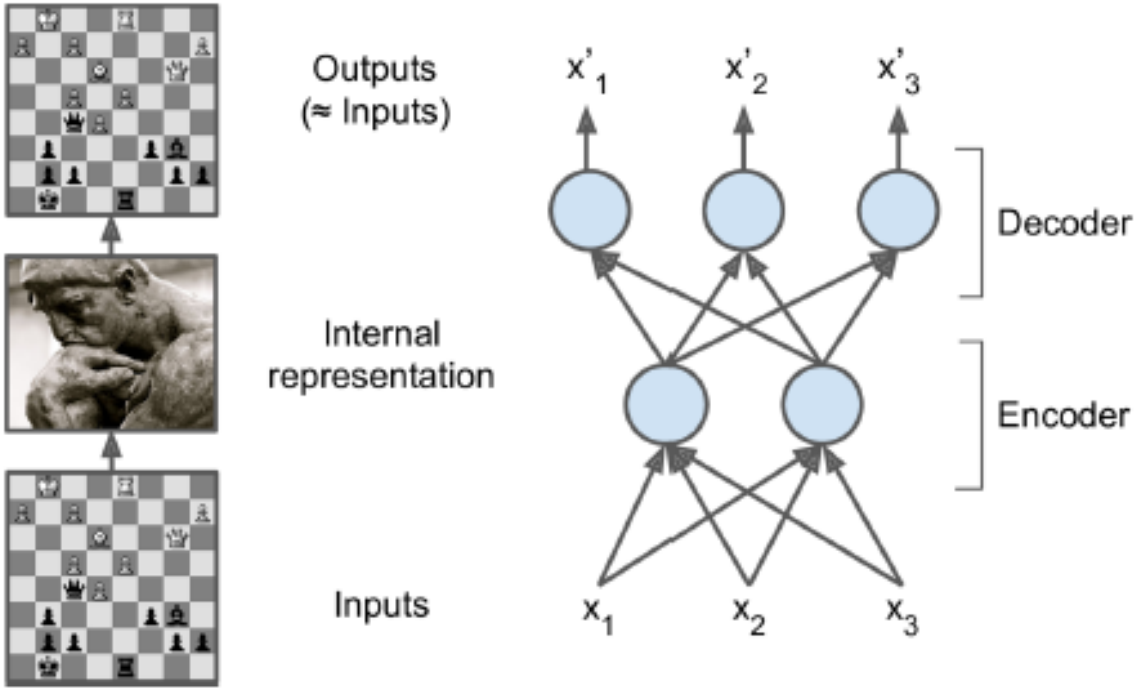
By Mohit Kumar

AutoEncoders:Why?

- Autoencoders are artificial neural networks capable of learning efficient representations of the input data, called codings, without any **supervision (i.e., the training set is unlabeled)**.
- These codings typically have a much **lower dimensionality** than the input data, making autoencoders useful for dimensionality reduction(much better that 2D PCA).
- More importantly, autoencoders act as **powerful feature detectors**, and they can be used for **unsupervised pretraining** of deep neural networks

AutoEncoders:Why?

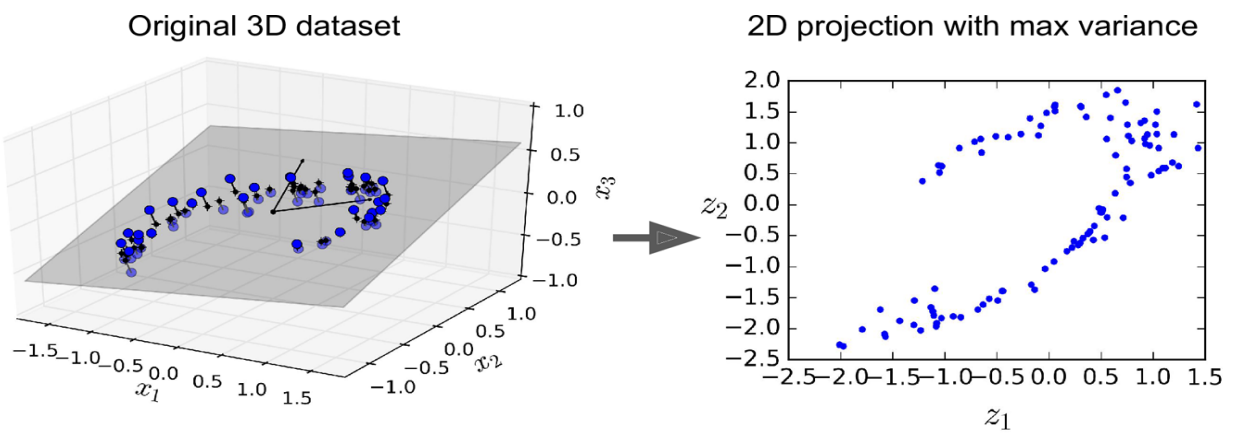
- An autoencoder is always composed of two parts: an encoder (or recognition network) that converts the inputs to an internal representation, followed by a decoder (or generative network) that converts the internal representation to the outputs.



AutoEncoders:Why?

- Because the internal representation has a **lower dimensionality than the input data** (it is 2D instead of 3D), the autoencoder is said to be undercomplete.
- An undercomplete autoencoder cannot trivially copy its inputs to the codings, yet it must **find a way to output a copy of its inputs**.
- It is forced to learn the **most important features** in the input data (and drop the unimportant ones)

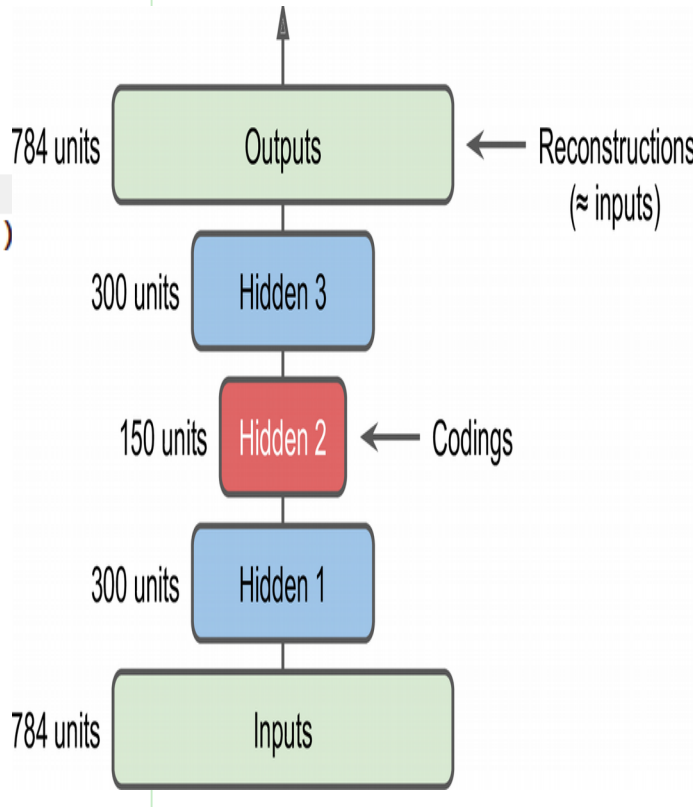
AutoEncoders:Tensorflo




```
tf.reset_default_graph()

n_inputs = 3
n_hidden = 2  # codings
n_outputs = n_inputs

learning_rate = 0.01

X = tf.placeholder(tf.float32, shape=[None, n_inputs])
hidden = tf.layers.dense(X, n_hidden)
outputs = tf.layers.dense(hidden, n_outputs)

mse = tf.reduce_mean(tf.square(outputs - X))

optimizer = tf.train.AdamOptimizer(learning_rate)
training_op = optimizer.minimize(mse)

init = tf.global_variables_initializer()

n_iterations = 10000
codings = hidden

with tf.Session() as sess:
    init.run()
    for iteration in range(n_iterations):
        training_op.run(feed_dict={X: X_train})
    codings_val = codings.eval(feed_dict={X: X_train})

fig = plt.figure(figsize=(4,3))
plt.plot(codings_val[:,0], codings_val[:, 1], "b.")
plt.xlabel("$z_1$", fontsize=18)
plt.ylabel("$z_2$", fontsize=18, rotation=0)
fl.save_fig("linear_autoencoder_pca_plot")
plt.show()
```

AutoEncoders:Tensorflow:Stacked/deep:encoding

```
n_inputs = 28*28
n_hidden1 = 300
n_hidden2 = 150  # codings
n_hidden3 = n_hidden1
n_outputs = n_inputs

learning_rate = 0.01
l2_reg = 0.0001

initializer = tf.contrib.layers.variance_scaling_initializer() # He initialization
X = tf.placeholder(tf.float32, shape=[None, n_inputs])
my_dense_layer = partial(
    tf.layers.dense,
    activation=tf.nn.elu,
    kernel_initializer=initializer,
    kernel_regularizer=tf.contrib.layers.l2_regularizer(l2_reg))

hidden1 = my_dense_layer(X, n_hidden1)
hidden2 = my_dense_layer(hidden1, n_hidden2)
hidden3 = my_dense_layer(hidden2, n_hidden3)
outputs = my_dense_layer(hidden3, n_outputs, activation=None)

mse = tf.reduce_mean(tf.square(outputs - X))
reg_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
loss = tf.add_n([mse] + reg_losses)
optimizer = tf.train.AdamOptimizer(learning_rate)
training_op = optimizer.minimize(loss)
init = tf.global_variables_initializer()
saver = tf.train.Saver()

n_epochs = 4
batch_size = 150

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        n_batches = mnist.train.num_examples // batch_size
        for iteration in range(n_batches):
```

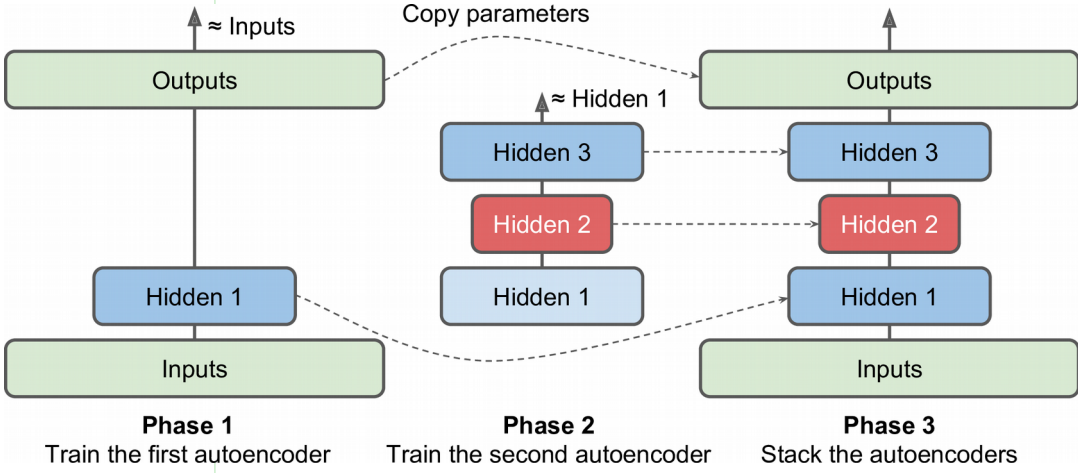


AutoEncoders:Tensorflow:Stacked/deep:reconstruction

```
hidden2 = my_dense_layer(hidden1, n_hidden2)
hidden3 = my_dense_layer(hidden2, n_hidden3)
outputs = my_dense_layer(hidden3, n_outputs, activation=None)

mse = tf.reduce_mean(tf.square(outputs - X))

reg_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
loss = tf.add_n([mse] + reg_losses)

optimizer = tf.train.AdamOptimizer(learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
saver = tf.train.Saver()

def plot_image(image, shape=[28, 28]):
    plt.imshow(image.reshape(shape), cmap="Greys", interpolation="nearest")
    plt.axis("off")


def show_reconstructed_digits(X, outputs, model_path = None, n_test_digits = 2):
    with tf.Session() as sess:
        if model_path:
            saver.restore(sess, model_path)
        X_test = mnist.test.images[:n_test_digits]
        outputs_val = outputs.eval(feed_dict={X: X_test})

    fig = plt.figure(figsize=(8, 3 * n_test_digits))
    for digit_index in range(n_test_digits):
        plt.subplot(n_test_digits, 2, digit_index * 2 + 1)
        plot_image(X_test[digit_index])
        plt.subplot(n_test_digits, 2, digit_index * 2 + 2)
        plot_image(outputs_val[digit_index])

show_reconstructed_digits(X, outputs, root_logdir+"/my_model_all_layers.ckpt")
fl.save_fig("reconstruction_plot")
plt.show()
```

AutoEncoders:Tensorflow:Train:One at a time:Singlegraph

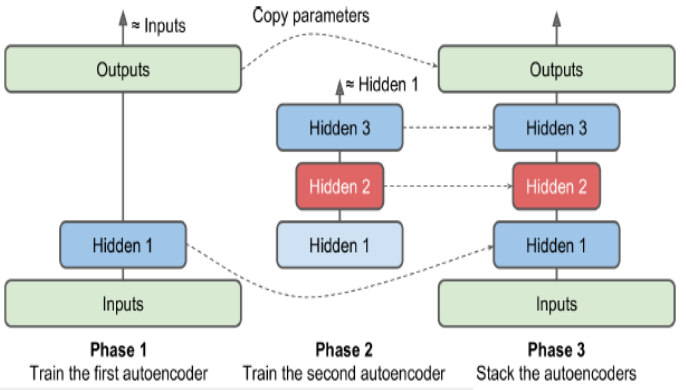
```
with tf.name_scope("phase1"):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    phase1_outputs = tf.matmul(hidden1, weights4) + biases4  # bypass hidden2 and hidden3
    phase1_mse = tf.reduce_mean(tf.square(phase1_outputs - X))
    phase1_reg_loss = regularizer(weights1) + regularizer(weights4)
    phase1_loss = phase1_mse + phase1_reg_loss
    phase1_training_op = optimizer.minimize(phase1_loss)

with tf.name_scope("phase2"):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    phase2_mse = tf.reduce_mean(tf.square(hidden3 - hidden1))
    phase2_reg_loss = regularizer(weights2) + regularizer(weights3)
    phase2_loss = phase2_mse + phase2_reg_loss
    phase2_training_op = optimizer.minimize(phase2_loss, var_list=[weights2, biases2, weights3, biases3]) # freeze hidden1

init = tf.global_variables_initializer()
saver = tf.train.Saver()

training_ops = [phase1_training_op, phase2_training_op]
mses = [phase1_mse, phase2_mse]
n_epochs = [4, 4]
batch_sizes = [150, 150]

with tf.Session() as sess:
    init.run()
    for phase in range(2):
        print("Training phase #{}".format(phase + 1))
        for epoch in range(n_epochs[phase]):
            n_batches = mnist.train.num_examples // batch_sizes[phase]
            for iteration in range(n_batches):
                print("\r{}%".format(100 * iteration // n_batches), end="")
                sys.stdout.flush()
                X_batch, y_batch = mnist.train.next_batch(batch_sizes[phase])
                sess.run(training_ops[phase], feed_dict={X: X_batch})
            mse_train = mses[phase].eval(feed_dict={X: X_batch})
            print("\r{}".format(epoch), "Train MSE:", mse_train)
            saver.save(sess, root_logdir+"/my_model_one_at_a_time.ckpt")
    mse_test = mses[phase].eval(feed_dict={X: mnist.test.images})
```



AutoEncoders:Tensorflow:Train:Freeze layers

```
with tf.name_scope("phase1"):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    phase1_outputs = tf.matmul(hidden1, weights4) + biases4  # bypass hidden2 and hidden3
    phase1_mse = tf.reduce_mean(tf.square(phase1_outputs - X))
    phase1_reg_loss = regularizer(weights1) + regularizer(weights4)
    phase1_loss = phase1_mse + phase1_reg_loss
    phase1_training_op = optimizer.minimize(phase1_loss)

with tf.name_scope("phase2"):
    optimizer = tf.train.AdamOptimizer(learning_rate)
    phase2_mse = tf.reduce_mean(tf.square(hidden3 - hidden1))
    phase2_reg_loss = regularizer(weights2) + regularizer(weights3)
    phase2_loss = phase2_mse + phase2_reg_loss
    phase2_training_op = optimizer.minimize(phase2_loss, var_list=[weights2, biases2, weights3, biases3]) # freeze hidden1

init = tf.global_variables_initializer()
saver = tf.train.Saver()
training_ops = [phase1_training_op, phase2_training_op]
mses = [phase1_mse, phase2_mse]
n_epochs = [4, 4]
batch_sizes = [150, 150]

with tf.Session() as sess:
    init.run()
    for phase in range(2):
        print("Training phase #{}".format(phase + 1))
        if phase == 1:
            mnist_hidden1 = hidden1.eval(feed_dict={X: mnist.train.images})
        for epoch in range(n_epochs[phase]):
            n_batches = mnist.train.num_examples // batch_sizes[phase]
            for iteration in range(n_batches):
                print("\r{}%".format(100 * iteration // n_batches), end="")
                sys.stdout.flush()
                if phase == 1:
                    indices = rnd.permutation(len(mnist_hidden1))
                    hidden1_batch = mnist_hidden1[indices[:batch_sizes[phase]]]
                    feed_dict = {hidden1: hidden1_batch}
                    sess.run(training_ops[phase], feed_dict=feed_dict)
```

- Be sure to leave out weights1 and biases1.



AutoEncoders:Tensorflow:Unsupervised Pretraining