1. Truncated normal initialization of weights in the statement tflearn.layers.core.fully\_connected.

tflearn.initializations.truncated\_normal (shape=None, mean=0.0, stddev=0.02, dtype=tf.float32, seed=None)

Initialization with random values from a normal truncated distribution.

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.

Hence our weights are in the range (-0.04, 0.04) and normally distributed. Can we improve this by using Xavier initialization?

1. Does relu force weights to be positive always? Will changing the activation function to leaky relu help?
2. Does anything enforce the net position in a coin to be an insignificant portion of market value? If no, can we implement it?
3. How does the portfolio perform wrt. benchmark? What should be the benchmark?
4. In SGD optimization, one typically initiates model weights at random and tries to go towards minimum cost by following the opposite of gradient of objective function. For deep nets, this has not shown much of success and it is believed to be result of extremely non-convex (and high-dimensional) nature of their objective function.
5. Unsupervised Pre-training

What Y. Bengio and others found out was that, instead of starting weights at random and hoping that SGD will take you to minimum point of such a rugged landscape, you can pre-train each layer like an autoencoder. Here is how it works: you build an autoencoder with first layer as encoding layer and the transpose of that as decoder. And you train it unsupervised, that is you train it to reconstruct the input (refer to AutoEncoders, they are great for unsupervised feature extraction tasks). Once trained, you fix weights of that layer to those you just found. Then, you move to next layers and repeat the same until you pre-train all layers of deep net (greedy approach). At this point, you go back to the original problem that you wanted to solve with deep net (classification/regression) and you optimize it with SGD but starting from weights you just learned during pre-training.

They found that this gives much better results. I think no one knows why exactly this works, but the idea is that by pre-training you start from more favorable regions of feature space.