Andy Liu

A stacked autoencoder is a type of autoencoder that uses stacked autoencoders to encode and decode data. It consists of multiple single-layer autoencoders in which the output feature of every layer is fed into the input of the next layer, thus "stacking" the autoencoders on top of each other. SAEs are trained by training each individual AE to minimize the error between output and input data. Similarly to autoencoders, they are usually used for unsupervised feature extraction for other types of predictive models, rather than being used on their own.

A different type of a stacked autoencoder is the stacked denoising autoencoder. The SDAE uses a similar architecture to a stacked autoencoder. However, it uses denoising single-layer autoencoders rather than normal ones. The distinguishing feature of a denoising autoencoder is that it randomly sets some inputs (usually 50%) to 0, in order to avoid a neural network with too many nodes from returning the identity map.

While the authors of the Stacked Autoencoder paper referenced in the ML survey article did not provide their code, they did provide a link to their data, and I was able to locate a GitHub repository that implemented their methods. The paper's authors used a wavelet transform to denoise their financial data before applying an SAE to their data to generate features that they could run a LSTM on. After modifying the repository's Python notebook (now located in my Dasion repository) to work with the original data source, I was able to test the LSTM and stacked autoencoder methods. Testing their LSTM on the S&P 500 data provided, I was able to obtain a Mean Absolute Percentage Error of 0.026%, which is comparable to the paper's result of a 0.017% average MAPE when running their LSTM on S&P 500 index data.

For the SDAE, I was able to adapt code from this paper using VAE and DAE for collaborative filtering (in this case, it was applied to movie rating data to generate recommendations). I modified the Python notebook that was used (test.ipynb) so that it would work with Tensorflow 2 and Python 3, as it was written with many deprecated packages. (Note that you will have to modify the file paths on the test.ipynb notebook and download the data used at this link). The code currently uses a softmax regression, but it can probably be tweaked further in the future to work with a normal distribution and financial data, if necessary. On the ML-20M dataset, the authors reported an NDCG@100 value of 0.419 for the Mult-DAE model, compared to a computed value of 0.425 when I ran it.