Satyam Kumar · Follow
Dec 23, 2021 · 4 min read ★

Improve your Model Performance with Auto-Encoders

Use Autoencoders as a Feature Extractor

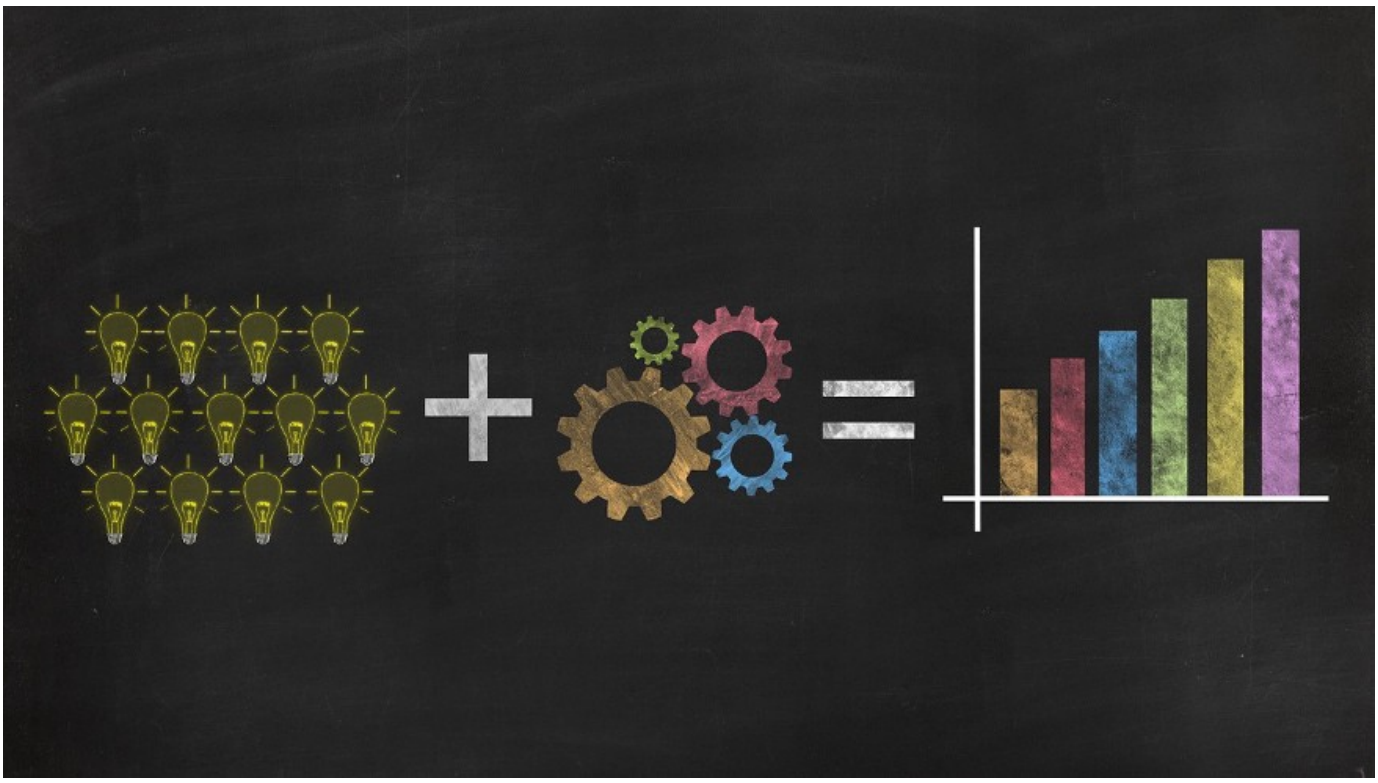


Image by ar130405 from Pixabay

You never know how your model performs unless you evaluate the performance of the model. The goal of a data scientist is to develop a robust data science model. The robustness of the model is decided by computing how it performs on the validation and test metrics. If the model performs well on the validation and test data, it tends to perform well during the production inference.

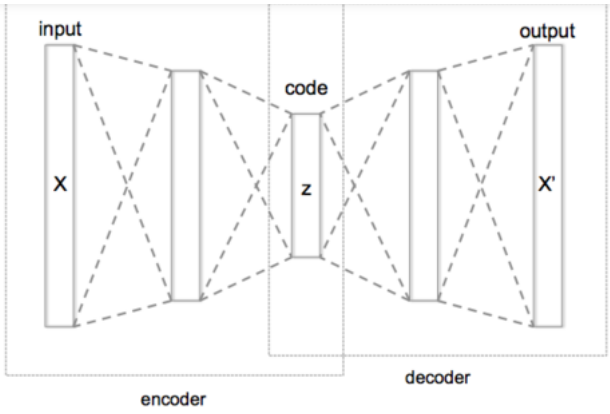
There are various techniques or hacks to improve the performance of the model. A correct feature engineering strategy tends to improve the performance of the model. In this article, we will discuss and implement feature extraction using autoencoders.

What are AutoEncoders?

AutoEncoders are an unsupervised neural network that learns efficient coding from the input unlabelled data. The autoencoders try to reconstruct the input data by minimizing the reconstruction loss.

A standard autoencoder architecture has an encoder and decoder layer:

- **Encoder:** Mapping from Input space to lower dimension space
- **Decoder:** Reconstructing from lower dimension space to Output space



(Source), Autoencoder architecture

Follow my previous article to know 7 applications or usage of Auto-Encoders

7 Applications of Auto-Encoders every Data Scientist should know
Essential guide to Auto-Encoders and its usage
towardsdatascience.com

Idea:

An autoencoder encodes the input data (X) into another dimension (Z), and then reconstructs the output data (X') using a decoder network. The encoded embedding is preferably lower in dimension compared to the input layer and contains all the efficient coding of the input layer.

The idea is to learn the encoder layer weights by training an autoencoder on the training sample. By training the autoencoder on the training sample, it will try to minimize the reconstruction error and generate the encoder and decoder network weights.

Later the decoder network can be cropped out and feature extraction embeddings can be generated using the encoder network. This embedding can be used for supervised tasks.

Now, let's dive deep into the step-by-step implementation of the above-discussed idea.

Step 1 — Data:

I am using a sample dataset generated using the make_classification function having 10k instances and 500 features. Split the dataset into train, validation, and test samples to avoid data leakage. Further, Normalize the data samples.

```
X, y = make_classification(n_samples=10000, n_features=500, n_informative=100, n_redundant=400, random_state=42)

# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=1)

# scale data
t = MinMaxScaler().fit(X_train)
X_train = t.transform(X_train)
X_val = t.transform(X_val)
X_test = t.transform(X_test)
```

ae_fe_data.py hosted with ❤ by GitHub view raw

Step 2 — Define AutoEncoder:

I have defined a two-layer of an encoder and a two-layer of decoder network in the autoencoder architecture. Each of the encoder and decoder layers has a batch normalization layer.

The dimension of the encoded layer needs to be decided by performing experiments.

```
# define encoder
visible = Input(shape=(n_inputs,))
# encoder level 1
e = Dense(n_inputs*2)(visible)
e = BatchNormalization()(e)
e = LeakyReLU()(e)
# encoder level 2
e = Dense(n_inputs)(e)
e = BatchNormalization()(e)
e = LeakyReLU()(e)
# bottleneck
n_bottleneck = int(bottleneck_features)
bottleneck = Dense(n_bottleneck)(e)
# define decoder, level 1
d = Dense(n_inputs)(bottleneck)
d = BatchNormalization()(d)
d = LeakyReLU()(d)
# decoder level 2
d = Dense(n_inputs*2)(d)
d = BatchNormalization()(d)
d = LeakyReLU()(d)
# output layer
output = Dense(n_inputs, activation='linear')(d)
```

ae_fe_autoencoder_architecture.py hosted with ❤ by GitHub view raw

Step 3 — AutoEncoder Training:

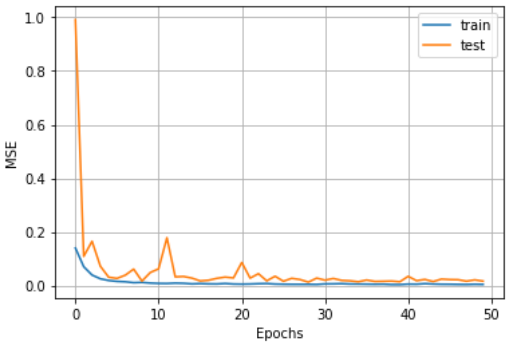
Compile and run the autoencoder architecture using Keras with adam optimizer, and mean squared error loss, to reconstruct the input data.

I will be running the pipeline for 50 epochs with a batch size of 64.

```
# define autoencoder model
model = Model(inputs=visible, outputs=output)
# compile autoencoder model
model.compile(optimizer='adam', loss='mse')

# fit the autoencoder model to reconstruct input
history = model.fit(X_train, X_train, epochs=50, batch_size=64, verbose=2, validation_data=(X_val,y_val))
```

ae_fe_autoencoder_training.py hosted with ❤ by GitHub view raw



(Image by Author), Train and Validation MSE with Epochs

```
# plot loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.xlabel("Epochs")
pyplot.ylabel("MSE")
pyplot.show()
```

ae_fe_plot.py hosted with ❤ by GitHub view raw

Step 5 — Define Encoder and Save the Weights:

Once the autoencoders, weights are optimized, we can crop the decoder network and use only the encoder network to compute the embeddings of the input data. These embeddings can be further used as features for supervised machine learning tasks.

```
# define an encoder model (without the decoder)
encoder = Model(inputs=visible, outputs=bottleneck)
# save the encoder to file
encoder.save('encoder.h5')
```

ae_fe_save_decoder.py hosted with ❤ by GitHub view raw

Step 6 — Train a Supervised task:

The embeddings from the decoder network can be used as features for the classification or regression tasks.

```
# load the model from file
encoder = load_model('encoder.h5')

# encode the train data
X_train_encode = encoder.predict(X_train)
# encode the test data
X_test_encode = encoder.predict(X_test)
```

ae_fe_encoder_generate.py hosted with ❤ by GitHub view raw

Benchmarking:

		Encoded Feature		
	Raw Input Data (500-dim)	Encoding 500-dim	Encoding 250-dim	Encoding 100-dim
AUC-ROC	0.87	0.99	0.99	0.98
Accuracy	0.78	0.98	0.97	0.97
F1-score	0.78	0.98	0.97	0.94
Precision	0.76	0.98	0.97	0.93
Recall	0.8	0.98	0.97	0.94

(Image by Author), Benchmarking performances metrics

The 1st column mentions the metrics performance for raw input data having 500 features. For the encoded features using the encoder network of the autoencoder, we observe an improvement in performance metrics.

Conclusion:

Feature extraction using autoencoders captures the efficient coding of the input data and projects it into the same or lower dimension. For input data having a lot of input features, it's a great idea to project the data into lower dimensions and use the features for supervised learning tasks.

Also, read some of my previous articles to improve the model performance:

Essential guide to Improve Imbalanced Data Classification Model Performance
Combine the oversampling and undersampling techniques
medium.com

References:

[1] https://machinelearningmastery.com/autoencoder-for-classification/

Loved the article? Become a Medium member to continue learning without limits. I'll receive a small portion of your membership fee if you use the following link, with no extra cost to you.

Join Medium with my referral link - Satyam Kumar
As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...
satyam-kumar.medium.com

Thank You for Reading