Introduction

Today, we will be learning about the restricted Boltzmann machine, the restricted Boltzmann machine is a deep learning model that generates data on it own by just training on some input vectors, it is a 2 layered network and can be used to perform complex tasks. In this tutorial you will learn about:

1. What Restricted Boltzmann Machine is?.
2. A brief history of the Restricted Boltzmann Machine.
3. The visible and hidden Layers Restricted Boltzmann Machine is built on.
4. Differences between Autoencoders and Restricted Boltzmann Machines
5. The Visible and Hidden biases and how they work.
6. How the Restricted Boltzmann Machine is trained.
7. How the Restricted Boltzmann Machine is Implemented in python.

What are Restricted Boltzmann Machines?

The restricted Boltzmann machine abbreviated as RBM is a generative stochastic artificial neural network that can learn a probability distribution over a given set of inputs, In 1986, Paul Smolensky invented the Restricted Boltzmann machine under the name Harmonium, it later rose to limelight after Geoffery Hinton collaborated with other investors to build fast learning algorithms. The Restricted Boltzmann machine is a two-layered neural network with one of the layers being the visible layer and the other acting as the hidden layer, with generative capabilities, they are a special class of Boltzmann machines, the two layers of the Boltzmann machine are connected by a fully bipartite graph which means that all the nodes in the visible layer are connected to all the nodes in the hidden layer but no two nodes found in the same group can be connected to each other. This attribute allows the Restricted Boltzmann machine to be a more efficient training algorithm than what is allowed for the general class of Boltzmann machines.

The restricted Boltzmann machine is applied to many areas including dimensionality reduction, classification, regression, collaborative filtering, feature learning and topic modelling. From the image above, we can see both the visible layer and the hidden layer, as stated earlier all the nodes are connected to each other in across the layers but not two nodes are linked, this represents the restrictions in the restricted Boltzmann machine, meaning there is not intra communication between layers. Each node contains computations that help in processing inputs and they begin by making stochastic decisions on whether to carry on in transferring the inputs or not. The visible node learns by taking a low-level feature from the given dataset, for example, if we are building a model to classify images, the visible node will each take one pixel each from the total pixels in our image and carry out computations on them to learn their distinct features. The restricted Boltzmann machine os a stochastic neural network meaning each neuron will contain some random behaviour when it is activated, together with the 2 layers, they contain a hidden bias and visible bias. The hidden bias is responsible for producing activation on the forward pass while the visible bias is responsible for reconstructing the input during the backward pass and this reconstructed input is usually always different from the actual input since there are no connections between the visible unit preventing the flow of information among the visible layer.

The image above shows the first step in training a restricted Boltzmann machine with multiple inputs, it multiplies the inputs by the weights and then adds them to the bias, the result from the previous computation is then passed to a sigmoid activation function to produce an output, This output determines if the hidden state will get activated or not.

Difference between Autoencoders and Restricted Boltzmann Machines

1. Autoencoders are a 3-layer neural network where the output nodes are directly connected back to the inputs node. The number of hidden nodes in the hidden layer is less than the number of visible node in the visible layers, so when you pass data through the network it first encodes the input vectors to fit the smallest representation and later tries to decode the vectors back. Its major goal is training to minimize an error or reconstruction, which is to find the most efficient compact encoding of the input data.
2. The restricted Boltzmann machine follows the same idea but differs in its use of a stochastic approach instead of the deterministic distribution. It uses stochastic units with a particular distribution. The intuition behind the restricted Boltzmann machine is that there are some visible random variables and some hidden random variables and the task of training is to find the connection between these two sets of variables.
3. The restricted Boltzmann machine has two biases, the hidden biases and the visible bias, the hidden bias help in producing the activations on the forward pass while the visible bias learns the reconstruction from the backward pass.
4. Autoencoders are intuitively understandable, easy to implement and reason about, The restricted Boltzmann machine is generative which means they can generate their new data with a given joined distribution and are considered more feature-rich and very flexible.

Layers in Restricted Boltzmann Machine

This is what we have been discussing earlier on, the restricted Boltzmann machine only has two layers, the visible and hidden layers. The nodes from both layers are fully connected to each other but not to another node in the same layer and this is the restriction in the Boltzmann restriction.

This prevents intra communication in a layer, each node processes input on its own and makes random decisions on whether to transmit that information or not.

Workings of the Restricted Boltzmann machine

Reconstruction is the process in which the restricted Boltzmann machine can learn to reconstruct data by themselves as we stated earlier that they are a generative model, so they can help in generating new information. The hidden layer activations become the input in a backward pass, they are multiplied by the same weights one per internode edge, just as x was weight-adjusted on the forward pass, the sum of those products is then added to the visible layer bias on each visible node and then we have our reconstruction. The difference between the reconstruction and the original data input is usually very large and this is because the restricted Boltzmann machine weights are randomly initialized, the reconstruction error is the difference between the values of the reconstruction and the input values and this reconstruction error is always backpropagated against the restricted Boltzmann weights and this process is repeated in a learning process until we reach a minimum error value. On the forward pass, the restricted Boltzmann machine uses the inputs to make predictions about a node activations while on the backward pass when the activations are received and reconstructions about the original data are fed out, the restricted Boltzmann machine is at that point attempting to estimate the probability of inputs **x** given an activation **a** which are weighted with the same coefficients as those used in the forward pass.

Unlike regression that estimates a continuous value based on many inputs or classification that makes guesses about a label to apply to a given input, reconstruction is making guesses about the probability distribution of the original input also know as generative learning. For example, imagine we have unique probability distributions for their pixel values, this depends on the kind of images in the set, pixel values are always distributed differently depending on where they come from. If we have a restricted Boltzmann machine that has two output nodes and we feed images of cats and dogs one animal representing an output node, the restricted Boltzmann machine will ask the question, given the pixels, should my weights send a stronger signal to the cat's output node or the dog output node? And on the backward pass, it asks given the cat which distribution of pixels am I expecting? This is a joint probability, the simultaneous probability of *x* given *a* and of *a* given *x*, expressed as the shared weights between the two layers of the restricted Boltzmann machine.

The reconstruction learning process is learning which group of pixels tend to co-occur for a given image set, the activation produced by the nodes of the hidden layers deep in the neural network represent significant co-occurrences.

Training a Restricted Boltzmann Machine

Training the restricted Boltzmann machine is very different from training a regular neural network with stochastic gradient descent.

1. Gibbs Sampling: This is the first part of the training, given an input vector v we are using p(h|v) for predictions of the hidden values h, knowing the hidden values we use p(h|v) for predictions of new input values v. This process is repeated k times, After *k* iterations, we obtain another input vector **v_k** which was recreated from original input values **v_0**.
2. Contrastive Divergence: This is the second step, the weight matrix update happens during the contrastive divergence step, the two vectors v_o and v_k are used to calculate the activation probabilities for hidden values h_o and h_k. The updated matrix is as a result of the difference between the outer products of those probabilities with input vectors v_o and v_k.

We can now use the updated matrix to calculate the new weight using the gradient ascent.

Applying a restricted Boltzmann machine to a Recommendation system
Collaborative Filtering: Assuming we have a rating system where users are asked to rate set of items in the scale of 1-5 and each item can be explained in terms of latent factors such as taste, quality, quantity etc. The restricted Boltzmann machines will be used to analyze and find out these underlying factors. The analysis of the hidden factor is performed by the users rating the system with their rating choice, this can be 1-5 and this acts as the input of the visible layer, given this input, the restricted Boltzmann will then try to discover latent factors in the data that explains some of its choices and each hidden neuron represents one of the latent factors. Considering a scenario where users like the movies Star Wars and Superman but does not like Beauty and the Beast, Cinderella, Cinderella gets a rating of -1 because it has not yet been seen, giving these data as input to the restricted Boltzmann machine can identify three underlying factors such as drama, romance and sci-fi that corresponds to each movie genre. The goal is to predict a binary rating for a movie that has not yet been seen given predicted data for a particular user, the model will be able to find the latent factors based on that users preferences and sample from Bernoulli distribution can be used to get all the visible neurons that are presently active.
In summary, the process from training to the prediction phase goes as follows:

1. Train the network on the data of all users
2. During inference, time take the training data of a specific user
3. Use this data to obtain the activations of hidden neurons
4. Use the hidden neuron values to get the activations of input neurons
5. The new values of input neurons show the rating the user would give yet unseen movies.

Implementing the Restricted Boltzmann Machine

For the implementation, we will be using the restricted Boltzmann machine to classify digits on the MNIST digits dataset hosted on the Keras library. We will be using the Bernoulli Restricted Boltzmann library on scikit-learn, the Bernoulli restricted Boltzmann machine model can perform effective non-linear feature extraction. We are going to build a classification pipeline with a Bernoulli Restricted Boltzmann machine feature extractor and a LogisticRegression classifier.

The Data

For this classification problem, we will be using the MNIST Digit Recognizer dataset, the dataset contains grey-scale images of hands drawn digits from zero through nine. Each of the images is a 28x28 pixels image and a total of 784 pixels in total. Each pixel contains a single pixel value associated with it indicating the lightness or darkness of that pixel, a higher value means that pixel is dark while a lower value means it is lighter and the pixel value is an integer value between 0 and 255. 255 inclusive.

Import packages

```
import numpy as np
import matplotlib.pyplot as plt

from scipy.ndimage import convolve
from sklearn import linear_model, datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.neural_network import BernoulliRBM
from sklearn.pipeline import Pipeline
from sklearn.base import clone
```

Above we import all the packages we need, this is one of the first steps we take when we start implementing any machine learning application. As we can see, we import the NumPy and matplotlib library, then we import the convolve library from scipy, we use the sklearn linear model and get our dataset from the sklearn dataset and metrics for evaluation, we use train test split to create a train, test split and our restricted Boltzmann implementation is gotten from the BernoulliRMB imported from the sklearn neural network part, we use the pipeline to create a training pipeline.

```
def nudge_dataset(X, Y):
   """
   This produces a dataset 5 times bigger than the original one,
   by moving the 8x8 images in X around by 1px to left, right, down, up
   """
   direction_vectors = [
       [[0, 1, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [1, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 1],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 1, 0]]]

   def shift(x, w):
       return convolve(x.reshape((8, 8)), mode='constant', weights=w).ravel()

   X = np.concatenate([X] +
                      [np.apply_along_axis(shift, 1, X, vector)
                       for vector in direction_vectors])
   Y = np.concatenate([Y for _ in range(5)], axis=0)
   return X, Y
```

The nudge dataset is used to create a dataset that is 5 times bigger than the original dataset, it moves the 8x8 images in X around by 1px to the left, right, down and up. This is another way to augement the dataset.

Load Data

```
# Load Data
X, Y = datasets.load_digits(return_X_y=True)
X = np.asarray(X, 'float32')
X, Y = nudge_dataset(X, y)
X = (X - np.min(X, 0)) / (np.max(X, 0) + 0.0001)  # 0-1 scaling

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=0)
```

Next, we load the data from the sklearn dataset library, after loading, we convert the data into an array of floating-point numbers and then nudge the data with the nude function we created above, next we scale the dataset and finally split the data into train and test.

Models

```
# Models we will use
logistic = linear_model.LogisticRegression(solver='newton-cg', tol=1)
rbm = BernoulliRBM(random_state=0, verbose=True)

rbm_features_classifier = Pipeline(
    steps=[('rbm', rbm), ('logistic', logistic)])
```

Here, we load all the models we will use, as you can see we are going to need the logistic regression model and our restricted Boltzmann machine, after instantiating the models we create a pipeline to train the model.

Training

```
rbm.learning_rate = 0.06
rbm.n_iter = 10
# More components tend to give better prediction performance, but larger
# fitting time
rbm.n_components = 100
logistic.C = 6000

# Training RBM-Logistic Pipeline
rbm_features_classifier.fit(X_train, Y_train)

# Training the Logistic regression classifier directly on the pixel
```

```
raw_pixel_classifier = clone(logistic)
raw_pixel_classifier.C = 100.
raw_pixel_classifier.fit(X_train, Y_train)
```

Before training the model, we first need to assign values to some of the hyperparameters we will need, These parameters were set by a cross-validation model that we are not going to perform here as it consumes time. We assigned the RMB learning rate the value of 0.06 and an iter value of 10, we used a component of 100 and assigned the C parameter of the logistic regression model 6000. After assigning the values we train the model on the pipeline created earlier and also train the logistic regression model on the data directly.

Evaluation

```
Y_pred = rbm_features_classifier.predict(X_test)
print("Logistic regression using RBM features:\n%s\n" % (
   metrics.classification_report(Y_test, Y_pred)))

Y_pred = raw_pixel_classifier.predict(X_test)
print("Logistic regression using raw pixel features:\n%s\n" % (
   metrics.classification_report(Y_test, Y_pred)))
```

After training the model, we evaluate the model, we perform the evaluation on the restricted Boltzmann machine and the logistic regression using the raw pixel.

Plot the Result

```
# Plotting

plt.figure(figsize=(10, 10))
for i, comp in enumerate(rbm.components_):
   plt.subplot(10, 10, i + 1)
   plt.imshow(comp.reshape((8, 8)), cmap=plt.cm.gray_r,
              interpolation='nearest')
   plt.xticks(())
   plt.yticks(())
plt.suptitle('100 components extracted by RBM', fontsize=16)
plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)

plt.show()
```

Finally, we plot the result from the restricted Boltzmann machine components, we will get a total of 100 images since we used 100 component value, this value can be lower or higher depending on how many images we want.

The image above is the total components gotten from the restricted Boltzmann components. You can see the notebook here.

Further Reading
I know some of you are eager to learn more about restricted Boltzmann machine, Don't worry I got you covered. I will be sharing some of the resources I found useful when writing this article and learning about the Restricted Boltzmann Machine.

1. Restricted Boltzmann Machine feature for digit classification.
2. Introduction to Restricted Boltzmann Machine.
3. The restricted Boltzmann Machine
4. An intuitive Introduction of Restricted Boltzmann Machine
5. Restricted Boltzmann Machines
6. A beginner's guide to restricted Boltzmann Machine
7. Deep Learning
8. Restricted Boltzmann Machine
9. Restricted Boltzmann Machines

Summary
Now we have come to the end of this tutorial. After going through the tutorial, you should now

have a basic understanding of the restricted Boltzmann machine and although they are not being used much this days because of the discovery of more powerful models like the GAN models and other predictive models, RBM is still a very good model that can help us perform complex deep learning tasks. By now you should know:

1. What Restricted Boltzmann Machine is.
2. Some brief history about the Restricted Boltzmann Machine.
3. Difference between Autoencoders and Restricted Boltzmann Machines.
4. The 2 Layers Restricted Boltzmann Machine is built on.
5. The Visible and Hidden biases and how they work.
6. How the Restricted Boltzmann Machine trains.
7. How the Restricted Boltzmann Machine is Implemented.

If you still have any question, please feel free and drop them in the comment section.