Project I.1 - Gait Recognition

Liam Adams (lbadams2)

I. Generative Model

I trained a deep belief network of stacked Restricted Boltzmann Machines. This network accepts the raw gait data and passes it through 4 RBMs ultimately reaching an RBM that expresses it in a 3D latent space.

An RBM is a parameterized generative model representing a probability distribution. When fed training data, it learns parameters that represent the training data as well as possible. It specifically tries to learn the unknown distribution q describing the training data. An RBM consists of 2 layers, a visible layer and a hidden layer. The nodes in each layer are independent of one another but there is a connection from each visible node to each hidden node, making them dependent. The hidden nodes can learn relationships between the visible nodes. [1]

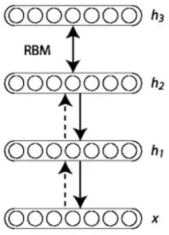

Fig. 1. 2 RBMs stacked

During data generation, the bottom RBM i's visible layer will accept the raw training data and pass it to its lower dimensional hidden layer which will have the same dimensions as the $i+1$ visible layer. Then the $i+1$ visible layer will pass on to its hidden layer, which will be lower dimensional and so on. The top most RBM will have a hidden layer of 3 dimensions, which was can use as the latent space of the training data in order to visualize it.

This is similar to a standard feed forward neural network, the difference being the connections between the visible and hidden layers are bidirectional, and there is a stochastic element in Gibbs sampling used during training. Each node in an RBM has a probability of being activated defined by the below for the hidden and visible nodes respectively [1]

$$p(H_i = 1|v) = \sigma(\sum_{j=1}^{m} w_{ij}v_j + c_i) \tag{1}$$

$$p(V_i = 1|h) = \sigma(\sum_{j=1}^{m} w_{ij}h_i + b_j) \tag{2}$$

Gibbs sampling is used in order to approximate $p(v, h)$, it involves traversing the Markov chain that is the RBM graph k times and obtaining samples of the visible and hidden vectors. If $k = \infty$ we would reach the stationary distribution in which the probability of sampling the RBM (Markov chain) would not change from one time step to the next, but we approximate using only k.

A. Model Training

The data was first transformed into one sample per time window. This meant that each sample was a $40x4x6 = 960$ dimensional vector, plus the training label. There were 40 samples per second, the window is 4 seconds, and there are 6 columns per sample. The data were also normalized to the range [0,1] so samples could be interpreted as probabilities. The RBMs had (visible_dim, hidden_dim) as following (960, 480), (480, 200), (200, 75), (75, 3). I use a learning rate of .01, 1 epoch, batch size of 25, and k=5 for Gibbs sampling.

The training process happens on one RBM at a time. During training Gibbs sampling uses these probabilities for generating the hidden vector from the input provided to the visible layer. This hidden vector will be all 0s or 1s depending on if the probability of the node is less than or greater than .5. Then it utilizes the birectionality, feeding the hidden vector to equation 2 and producing a new visible vector. Once you get back to the visible layer you've completed one time step. Each batch of training data in an RBM is trained for k time steps. [2]

Now that we have generated a new batch of visible vectors we can calculate MSRE between the new and original vectors and also make a step in gradient ascent. In an RBM we are maximizing the log likelihood of the model parameters given the training data

$$ln \sum_{h} e^{-E(v,h)} - ln \sum_{v,h} e^{-E(v,h)} \tag{3}$$

where $E(v, h)$ is the energy function of the Gibbs distribution that factorizes over the graph of the RBM. The gradient of the log likelihood is

$$-\sum_{h} p(h|v)\frac{\partial E(v,h)}{\partial \Theta} + \sum_{v,h} p(v,h)\frac{\partial E(v,h)}{\partial \Theta} \tag{4}$$

Now we have to find the partial derivatives of the model parameters which are the weights and biases in equations 1 and 2 above. These gradients are given by

$$\sum_{v \in S} \frac{\partial lnL(\theta|v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{5}$$

$$\frac{\partial lnL(\theta|v)}{\partial b_j} = v_j - \sum_v p(v)v_j \tag{6}$$

$$\frac{\partial lnL(\theta|v)}{\partial c_i} = p(H_i = 1|v) - \sum_v p(v)p(H_i = 1|v) \tag{7}$$

However the complexity of calculating these gradients exactly is too great so we need to approximate. In my model I used Persistent Contrastive Divergence to approximate the gradients which essentially is the Gibbs sampling process described above. We use the vectors returned from the Gibbs sampling process as the expectations used in the gradient equations above. For this reason Gibbs sampling is a Monte Carlo approximation technique. Before starting the traversal of the Markov chain for a batch of training data we initialize the visible layer with a Gibbs sample, this is what makes it Persistent Contrastive Divergence. If we used the raw training sample on the visible layer it would be Contrastive Divergence [3]. PCD persists the effects of the model parameters for all traversals of the chain. Now that we can approximate the gradient equations above from approximate expectations, we pass them to the Adam optimizer to optimize the gradient step, then update the actual model parameters using the step.

Once we get through the entire training set on one RBM, we initialize the model parameters of the next RBM with those from the just learned one, and repeat the above process.

B. Model Metrics

During training I gathered several metrics, for example the mean of the norm of the gradient vectors. The gradient vector consists of the 3 model parameters (weights, hidden bias, visible bias). Its calculated for each training sample so each batch has a matrix of these gradients. Below is the mean gradient norm of the top RBM (last one to be trained).

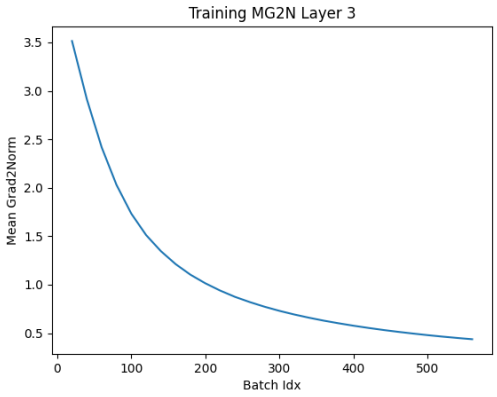

Fig. 2. Mean gradient norm top RBM

I also calculated the Mean Squared Recontstruction Error between the new visible vector generated from Gibbs Sampling and the original visible vector passed to the RBM. Below is the MSRE for the bottom (first) RBM

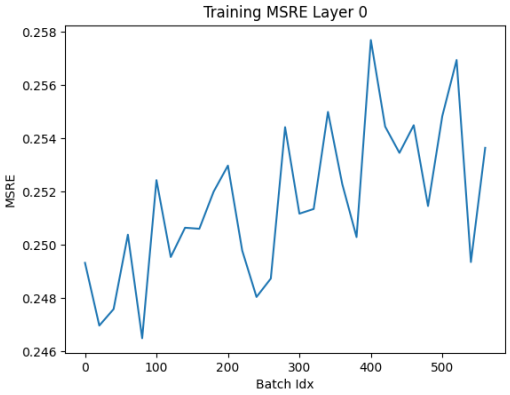

Fig. 3. MSRE bottom RBM

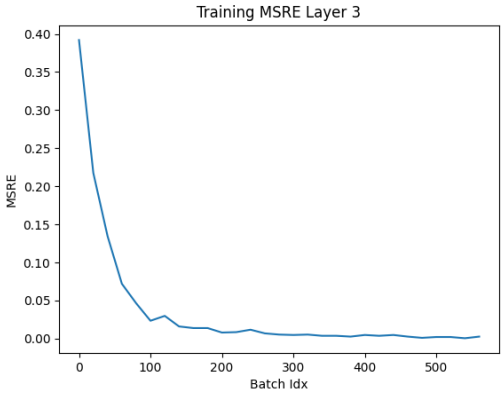

Fig. 4. MSRE top RBM

Below is the MSRE for the last RBM, it is clear that it has improved a lot as the trained parameters get transferred to subsequent RBMs in the network.

I also calculated MSRE on a validation set, which was 15% of the overall training set provided. This was run twice during the training of each RBM. The validation set was run on the full DBN rather than on individual RBMs.

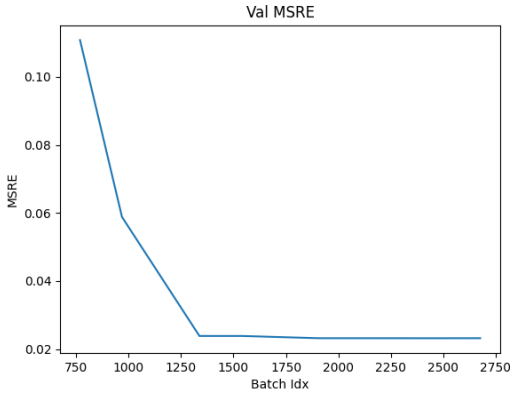

Fig. 5. Validation MSRE DBN

II. Generative Model Evaluation

The trained model can now be used to generate new samples or to reduce the dimensions of test data. Running the trained model uses the same Gibbs Sampling process we used previously. At each layer we calculate the probabilities of the neurons being activated based on equations 1 and 2 and generate binary vectors from these probabilities. This means the first layer is the only layer that sees the raw data from the sensors. This is repeated for k steps before producing the final synthetic sample. On the last time step we use the raw probabilities rather than generating a binary vector.

A. Visualization

In order to visualize the data I saved the top RBM's hidden layer of dimension 3 during the final time step in the Gibbs sampling. I used k=5 for the time steps.

I found visualizing the data fairly challenging. The model parameters in each RBM were different, however the same probabilities seemed to be generated for every sample. These probabilities were either very close to 1 or very close to 0 with no in between. Below is a visualization of 100 samples color coded by y label

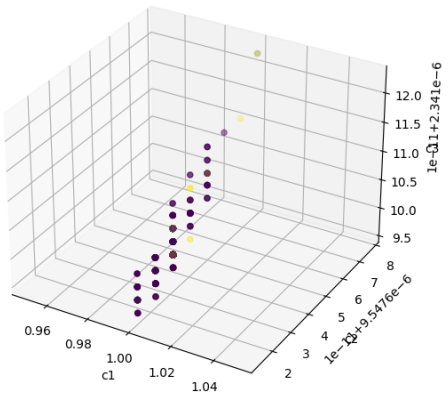

Fig. 6. Latent space visualization 100 samples

Ideally there would be 4 clusters since we have 4 labels. I will research strategies to deal with underflow and perhaps use the raw probabilities more rather than using the binary activations of the neurons. Other possibilities could be the learning rate being too large and causing the data to be pushed to the extremes of 0 and 1. Perhaps I could experiment with smaller learning rates. Also the number of Gibbs Sampling steps k could be a factor, running it too many times could cause the activations to be more uniform.

B. Data Generation

For data generation, I ran the validation set through the trained model. Again this used the Gibbs sampling process described above for k=5. In the final time step I used the raw probabilities rather than binary activations in order to match the domain of the raw data.

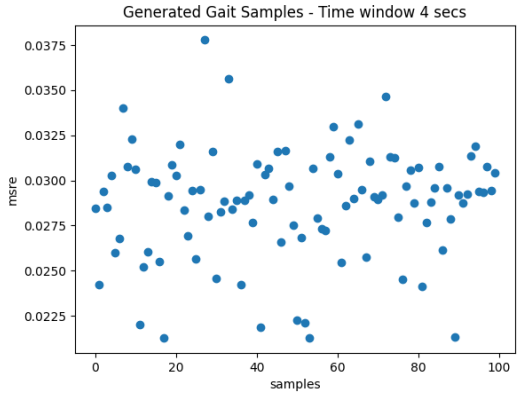

Fig. 7. MSRE between Synthetic Data and Original

As the synthetic data is 960 dimensional this is the best way I could think of to portray it graphically. You can see some samples with an MSRE of about .01. The MSRE was calculated between the synthetic sample and the sample from the validation set that generated it.

References

[1] A. Fischer and C. Igel. (2019) An introduction to restricted boltzmann machines. [Online]. Available: Fischer-Igel2012$_{C}hapter_{A}nIntroductionToRestrictedBolt.pdf$

[2] G. Tancev. (2016) Bnn with tensorflow probability. [Online]. Available: https://towardsdatascience.com/bayesian-neural-networks-with-tensorflow-probability-fbce27d6ef6

[3] R. Code. (2016) Introduction to restricted boltzmann machines. [Online]. Available: https://rubikscode.net/2018/10/01/introduction-to-restricted-boltzmann-machines/