Optimization of Backpropagation algorithm in Neural Networks

A DATS 6202_12 Machine Learning project report by Team – 9 Aditya Kumar,

Aditya Nayak & Medhasweta Sen

I. Abstract:

A neural network is a type of machine learning algorithm that is modeled after the structure of the human brain. Neurons are arranged in layers, with each layer connected to the previous and next layer. The input layer receives the input data, and the output layer produces the output. In between, there can be one or more hidden layers, where the computations are performed. Training a neural network involves adjusting the weights & biases of the neurons to minimize the difference between the predicted output and the actual output. This is typically done using backpropagation.

Backpropagation is an algorithm used to train such neural networks. It's a supervised learning method that adjusts the weights of the network to minimize the difference between the predicted output and the actual output. It works by first making a forward pass through the network to generate a prediction. The difference between the predicted output and the actual output is then calculated using a loss function like mean squared error. The research on faster algorithms falls roughly into two categories. The first category involves the development of heuristic techniques, which arise out of a study of the distinctive performance of the standard backpropagation algorithm. These heuristic techniques include such ideas as varying the learning rate, using momentum, and rescaling variables & standard numerical optimization techniques.

II. Dataset Descriptions:

To implement our project, we have considered 3 datasets (Small/Medium/Large) from Kaggle having features less than 5 for a Small dataset, between 5 to 15 for Medium and more than 15 for a Large dataset.

1. Small: Housing dataset

This dataset is based on data from the 1990 California Census, and includes metrics such as the population, median income, and median housing price for each block group in California. Block groups are the smallest geographical unit for which the

US Census Bureau publishes sample data., and each block typically has between 600 to 3000 people.

2. Medium: Spotify YouTube dataset

Dataset of songs of various artist in the world and for each song is present:

- Several statistics of the music version on Spotify, including the number of streams.
- Number of views of the official music video of the song on YouTube.

Content:

It includes 26 variables for each of the songs collected from Spotify. For our project, we considered 17 variables and are briefly described next:

- **Danceability**: describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
- **Energy**: is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
- **Key**: the key the track is in. Integers map to pitches using standard Pitch Class notation. E.g., 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
- **Loudness**: the overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlation of physical strength (amplitude). Values typically range between -60 and 0 db.
- **Speechiness**: detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g., talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
- **Acousticness**: a confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.

- **Instrumentalness**: predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
- **Liveness**: detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides a strong likelihood that the track is live.
- **Valence**: a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g., happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g., sad, depressed, angry).
- **Tempo**: the overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
- **Duration_ms**: the duration of the track in milliseconds.
- **Stream**: number of streams of the song on Spotify.
- **Views**: number of views.
- **Likes**: number of likes.
- **Comments**: number of comments.
- **Description**: description of the video on YouTube.
- **Licensed**: Indicates whether the video represents licensed content, which means that the content was uploaded to a channel linked to a YouTube content partner and then claimed by that partner.
- **official_video**: Boolean value that indicates if the video found is the official video of the song.

3. Large: cancer_reg

The following dataset consists of 33 predictor variables along with a target variable TARGET_deathRate. No further description of the data variables was provided on Kaggle.

III. Description of Network & training algorithms:

1. Backpropagation

The backpropagation algorithm works by first making a forward pass through the network to generate a prediction. The difference between the predicted output and the actual output is then calculated using a loss function,

such as mean squared error. The algorithm then works backwards through the network to compute the gradient of the loss function with respect to the weights of each neuron. This is done using the chain rule of calculus, which allows the gradient to be propagated backwards through the layers of the network.

2. ***Steepest Gradient Descent(SGD) with Momentum***

It's an optimization algorithm commonly used to minimize a loss function. The basic idea is to iteratively adjust the parameters of a model, such as the weights of a neural network, in the direction that minimizes the loss function. However, SGD can be slow and prone to getting stuck in local minima.

Momentum is a technique that helps accelerate the optimization process and avoid getting stuck in local minima. Momentum works by introducing a "velocity" term that accumulates gradients over time. Instead of just adjusting the parameters based on the current gradient, the velocity term considers the past gradients as well. This helps smooth out the optimization process and avoid oscillations in the parameter updates.

3. ***Variable Learning rate***

It is a technique used to adjust the learning rate of an optimization algorithm during training. The idea is to use a higher learning rate at the beginning of the training process, when the parameters are far from the optimal values, and then gradually decrease the learning rate as the optimization process progresses.

The main advantage of the algorithm is that it can help improve the convergence & stability of the optimization process. A high learning rate at the beginning of training can help the algorithm quickly move towards a good solution, while a lower learning rate towards the end can help fine-tune the parameters and prevent overshooting.

4. ***Conjugant Gradient Algorithm***

The conjugate gradient algorithm is an iterative optimization algorithm used for solving linear systems of equations. It can also be used to optimize convex quadratic functions, which are used as loss functions in machine learning.

The basic idea behind the conjugate gradient algorithm is to iteratively find a search direction that is conjugate to the previous search direction, meaning that it is orthogonal to all previous search directions. This helps the algorithm efficiently search the space of possible solutions and converge to the optimal solution in fewer iterations than traditional gradient descent algorithms.

5. ***Levenberg-Marquardt Algorithm***

The Levenberg-Marquardt algorithm is an optimization algorithm commonly used for solving non-linear least squares problems. It is named after Kenneth Levenberg and Donald Marquardt, who independently proposed the algorithm in the 1960s. The goal of the algorithm is to find the set of parameters that minimize the sum of the squared errors between the model predictions and the actual target values. This is a common problem in machine learning, where the goal is to fit a model to a set of data.

The Levenberg-Marquardt algorithm combines the strengths of two other optimization algorithms: the steepest descent algorithm and the Gauss-Newton algorithm. Overall, the Levenberg-Marquardt algorithm is a powerful optimization algorithm and can be used for tasks such as curve fitting, image processing, and computer vision.

6. ***Bayesian Regularization***

Bayesian regularization, also known as Bayesian ridge regression, is a statistical approach to regularization in machine learning. It is based on the principles of Bayesian inference and uses a prior distribution over the model parameters to regularize the model.

The goal of Bayesian regularization is to find the optimal set of parameters that fit the data well while also being consistent with the prior knowledge about the problem. This is done by adding a penalty term to the loss function that encourages the parameters to have values that are close to the prior distribution.

In summation, Bayesian regularization helps in preventing overfitting and improves the generalization performance of machine learning models. It is used in tasks such as regression, classification, & feature selection.

7. ***LM algorithm with momentum***

The Levenberg-Marquardt algorithm with momentum is a variant of the Levenberg-Marquardt algorithm that includes a momentum term to improve convergence speed and stability.

It is used to solve non-linear least squares problems. It works by iteratively updating the parameters of a model to minimize the sum of the squared errors between the model predictions and the actual target values. The algorithm uses a damping parameter to balance between the steepest descent and Gauss-Newton methods of optimization.

The momentum term in the Levenberg-Marquardt algorithm with momentum adds an additional component to the parameter update rule, which considers the previous parameter update. This helps the algorithm to converge faster and to avoid getting stuck in local optima.

IV. Experimental Setup

To ensure sufficient observations for our neural network, we collected datasets with more than 1000 observations. We followed Prof. Amir Jafari's instructions and chose three Kaggle datasets, characterized by a small, medium, and large number of features.

Data collection, cleaning, and pre-processing techniques were successfully implemented; however, difficulties were encountered in implementing the variable learning rate optimization algorithm. Despite several attempts, unforeseen technical challenges prevented the delivery of a functioning solution. It highlights the importance of further research and development in this area. Future work will continue to explore optimization techniques and build upon the knowledge gained to improve model performance.

To implement various optimization algorithms such as conjugate gradient, LM algorithm, steepest descent, and variable learning rate with momentum from scratch in backpropagation, we need to follow some general steps. First, we should define the architecture of the neural network, including the number of layers, the number of neurons in each layer, the activation function, and the loss function. Next, we initialize the weights and biases of each layer, choosing from different initialization methods such as random or Xavier initialization.

After that, we choose the optimization algorithm we want to implement. Each optimization algorithm has its own set of update rules for the weights and biases. The four optimization algorithms we're considering here are conjugate gradient, LM algorithm, steepest descent, and variable learning rate with momentum.

To train the neural network, we use a training dataset and a validation dataset. We iterate over the training dataset and perform forward and backward propagation to calculate the gradients. Then, we use the gradients to update the weights and biases according to the chosen optimization algorithm. We repeat this process until we achieve convergence, which means the neural network has reached a satisfactory level of accuracy.

After training the neural network, we evaluate its performance on the validation dataset. We can calculate metrics such as accuracy, precision, recall, and F1 score to assess the performance of our model. Depending on the performance of the model, we may need to adjust the learning rate and momentum.

The specific steps for each optimization algorithm are as follows:

- **Conjugate Gradient:** We initialize the search direction and the conjugate gradient vector. Then, we calculate the gradient of the cost function with respect to the weights and biases. We update the weights and biases

using the conjugate gradient update rule. We repeat these steps until convergence.

- **LM Algorithm:** We calculate the Hessian matrix, which is the second derivative of the cost function with respect to the weights and biases. Then, we calculate the gradient of the cost function with respect to the weights and biases. We use the Hessian matrix and the gradient to update the weights and biases using the LM algorithm update rule. We repeat these steps until convergence.
- **Steepest Descent:** We calculate the gradient of the cost function with respect to the weights and biases. We update the weights and biases using the steepest descent update rule. We repeat these steps until convergence.
- **Variable Learning Rate with Momentum:** We initialize the learning rate and momentum. Then, we calculate the gradient of the cost function with respect to the weights and biases. We update the weights and biases using the variable learning rate with momentum update rule. We adjust the learning rate and momentum based on the performance of the model. We repeat these steps until convergence.

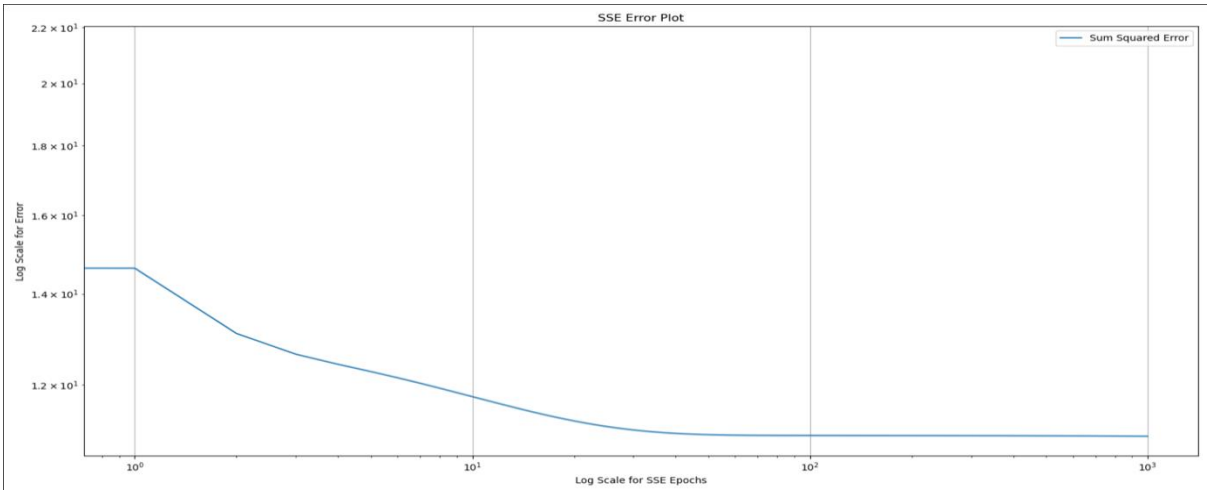
V. Results

The data collection, cleaning, and pre-processing were critical for our machine learning project's success. Careful data selection and cleaning ensured accuracy and completeness. Pre-processing steps such as feature scaling and selection optimized neural network performance.

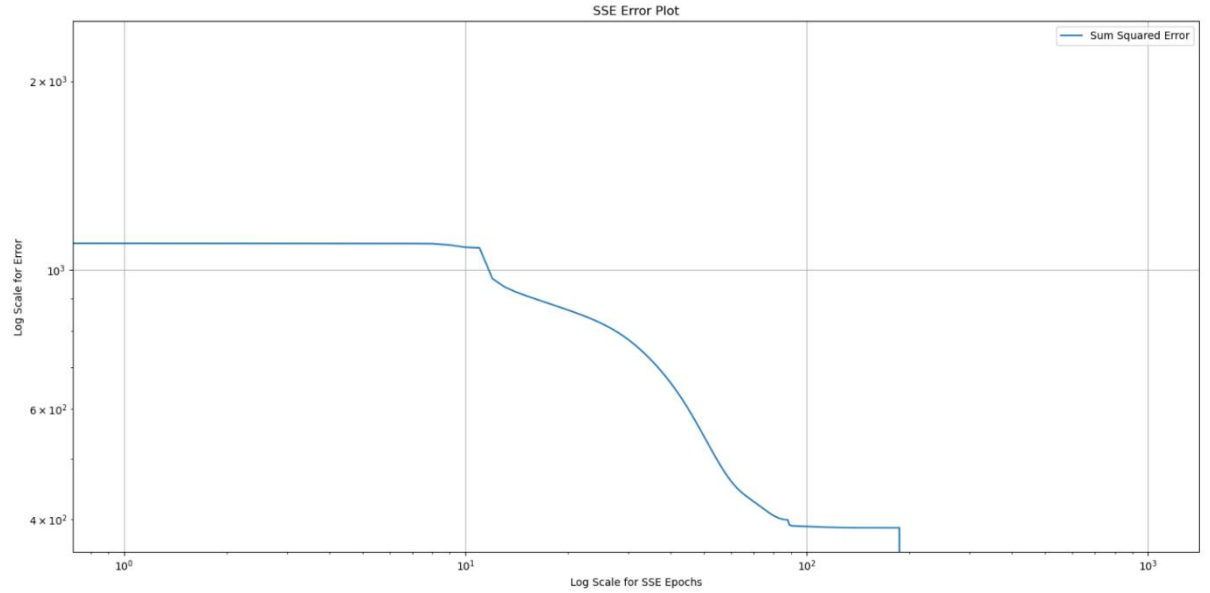
We also ran the datasets for conjugate gradient for the first PC and calculated the SSE:

```
...: Xtrain1, Xtest1, ytrain1, ytest1 = train_test_split( Xdf1,ydf1,test_size=0.2, random_state=42 )
...: network13 = Generalized_NeuralNetwork_Backpropagation([1,10,1],['sigmoid','linear'])
...: network13.train(Xtrain1,ytrain1,learning_rate=0.01,epochs=300,optimizer='conjugate gradient',batch_size=20)
...: p1 = network13.prediction(Xtest1)
...: error1 = ytest1-p1
...: # x = error1[~np.isnan(error1)]
...: print(np.sum(error1**2))
...:
...: Xtrain2, Xtest2, ytrain2, ytest2 = train_test_split( Xdf2,ydf2,test_size=0.2, random_state=42 )
...: network23 = Generalized_NeuralNetwork_Backpropagation([1,10,1],['sigmoid','linear'])
...: network23.train(Xtrain2,ytrain2,learning_rate=0.01,epochs=300,optimizer='conjugate gradient',batch_size=20)
...: p2 = network13.prediction(Xtest2)
...: error2 = ytest2-p2
...: # x = error2[~np.isnan(error2)]
...: print(np.sum(error2**2))
...:
...: Xtrain3, Xtest3, ytrain3, ytest3 = train_test_split( Xdf3,ydf3,test_size=0.2, random_state=42 )
...: network34 = Generalized_NeuralNetwork_Backpropagation([1,100,1],['sigmoid','linear'])
...: network34.train(Xtrain3,ytrain3,learning_rate=0.01,epochs=300,optimizer='conjugate gradient',batch_size=20)
...: p3 = network13.prediction(Xtest3)
...: error3 = ytest3-p3
...: print(np.mean(error3**2))
...:
9.4930813307376446+17
1.0895335760787229e+24
31879.04742498154
```

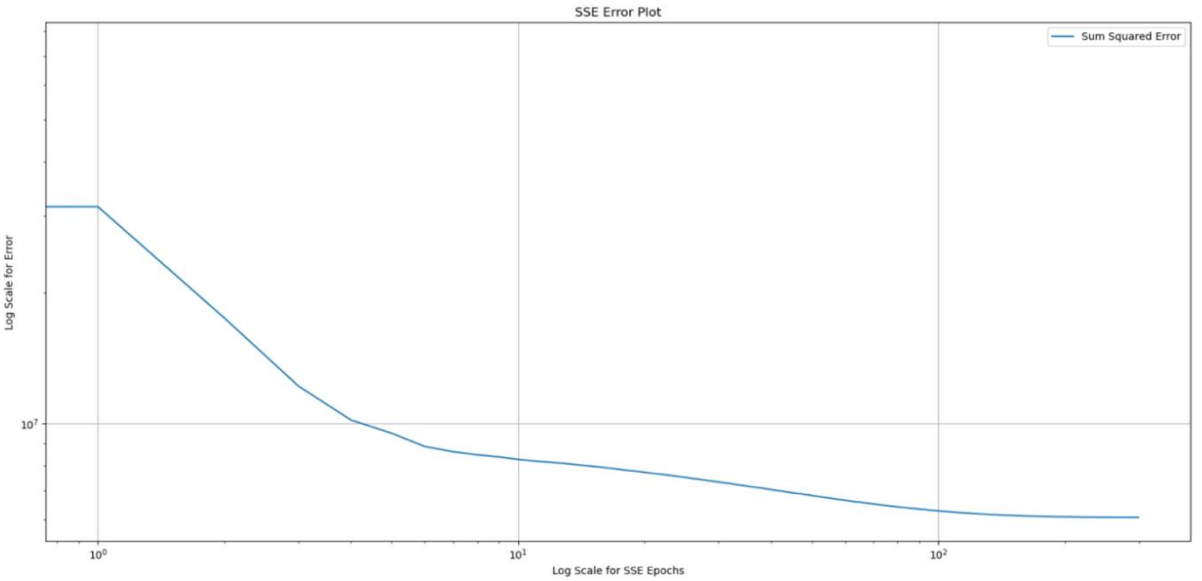
After compiling the code with backpropagation with gradient descent , conjugate gradient, and LM algorithm into one train function with some additional feature. I ran the function for 2 different use cases. The first one is for synthetic sinusoidal data. While running that we generated 3 Mean Square Errors for each of the methods. The results are as follows:



SSE for steepest descent



SSE for LM



SSE for Conjugate Gradient

VI. Conclusion

Thus, we have implemented various optimization techniques such as Backpropagation, Variable Learning Rate, Steepest Gradient Descent etc. on our Neural Network model and analyzed which algorithm is more accurate in prediction.

VII. References

a. "Neural Network Design" (2nd Ed), by Martin T Hagan, ISBN 0971732116
b. Henry P. Gavin (2022), "The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems", Duke University, Department of Civil & Environmental Engineering.
c. Rauf Bhat (2020), "Gradient Descent with Momentum", Towards Data Science.
d. I. Khan et al (2020), "Design of Neural Network with Levenberg-Marquardt and Bayesian Regularization Backpropagation for Solving Pantograph Delay Differential Equations" in IEEE Access, vol. 8, pp. 137918-137933, doi: 10.1109/ACCESS.2020.3011820.