Project #2

Group 17:

Maddireddy Avinashkarthikeya SE20UCSE081

Rohith Dandi SE20UCSE148

Stage 1: Basic ANN and familiarization with Deep Learning Libraries

Comparisons between ANN without DLL and ANN with DLL:

	ANN WITHOUT DLL	ANN WITH DLL
ANN ARCHITECTURE	No of layers: 5 ([4, 28, 16, 10, 1]) Input layer: 4, hidden layer: 28, 16, 10 neurons, output layer: 1	No of layers: 5 ([4, 28, 16, 10, 1]) Input layer: 4, hidden layer: 28, 16, 10 neurons, output layer: 1
TRAINING DATA GRANULATION	SGD with mini batches Evaluation metric: MAPE Batch size (1): Training Cost: 44.592 validation Cost: 99.575 test Cost: 90.236 Batch size (64): Training Cost: 46.399 validation Cost: 88.527 test Cost: 82.482 Batch size (256): Training Cost: 44.592 validation Cost: 99.574 test Cost: 90.534 Batch size (full batch): Training Cost: 50.594 validation Cost: 102.571 test Cost: 94.235	SGD with mini batches Evaluation metric: MAPE Batch size (1): Training Cost: 14.753 validation Cost: 14.888 test Cost: 14.451 Batch size (64): Training Cost: 14.7184 validation Cost: 14.8872 test Cost: 14.452 Batch size (256): Training Cost: 14.718 validation Cost: 14.887 test Cost: 14.561 Batch size (full batch): Training Cost: 105.360 validation Cost: 106.302 test Cost: 108.83

	All the above after 1000 iterations except full batch with η = 0.001.	All the above after 1000 iterations except full batch with η = 0.001.

ACTIVATION FUNCTIONS AND I/O NORMALIZATION	Experimenting with tanh, logistic and ReLU activation functions
	Tanh is used for final implementation
	
	I/O NORMALIZATION:
	def normalize_dataframe(df):
	normalized_df = df.copy()
	for column in df.columns:
	x_min = df[column].min()
	x_max = df[column].max()
	virtual_x_min = x_min - 0.05 * (x_max - x_min)
	virtual_x_max = x_max + 0.05 * (x_max - x_min)
	normalized_df[column] = (2 * df[column] - (virtual_x_max+ virtual_x_min)) / (virtual_x_max - virtual_x_min)
	return normalized_df
		Experimenting with tanh, logistic and ReLU activation functions
		Tanh is used for final implementation
		
		I/O NORMALIZATION:
		from sklearn.preprocessing import MinMaxScaler
		scaler = MinMaxScaler((-0.9, 0.9))

WEIGHT INITIALIZATION	Initializing weights from a standard normal distribution, each weight is drawn independently from a Gaussian distribution with mean 0 and standard deviation 1.	Initializing weights from a standard normal distribution, each weight is drawn independently from a Gaussian distribution with mean 0 and standard deviation 1.

LEARNING RATE PARAMETER	For Batch size (64)
	1. η = 0.001, it takes 1000 iterations to converge
	2. η = 0.01 for λ = 0, it takes 300 iterations to converge
	3. η = 0.0001 for λ = 0, it takes 4000 iterations to converge
		For Batch size (64)
		η = 0.001, it takes 400 iterations to converge
		η = 0.01 for λ = 0, it takes 50 iterations to converge
		η = 0.0001 for λ = 0, it takes 3800 iterations to converge

L2 REGULARIZATION PARAMETER	With learning rate is 0.001 and experiment lambda the resultant lambda value is 0.92, it takes 1000 iterations to converge.	With learning rate is 0.001 and experiment lambda the resultant lambda value is 0.1 η = 0.01 for λ = 0.1, it takes 500 iterations to converge Batch size (64): Training Cost: 6.3582 validation Cost: 6.3531 test Cost: 6.3421

Conclusions:

Conclusion Without Deep Learning Libraries:

In the absence of DLL, implementing a neural network from first principles typically involves coding the forward and backward passes, defining activation functions, and updating weights manually. This approach can be computationally intensive and may require more effort to achieve optimization. Additionally, without the convenience of high-level libraries like TensorFlow or PyTorch, the implementation might be less efficient.

Conclusion With Deep Learning Libraries – TensorFlow:

Utilizing TensorFlow for building and training the neural network streamlines the process significantly. It provides a higher-level abstraction, making it easier to define, train, and evaluate complex neural network architectures. The comparison between the TensorFlow-based ANN and the first-principles program would likely show that the former is more convenient, efficient, and less error-prone. By leveraging TensorFlow, you benefit from optimized operations and automatic differentiation, simplifying the implementation of backpropagation and gradient descent. The ease of experimentation with different parameters and architectures also contributes to a faster development cycle. The convergence histories and test data errors from the TensorFlow-based ANN should demonstrate the advantages of using deep learning libraries for neural network development.