

Digital VLSI Design

Model based

Aim:

To train multiple models on the given Datasets, replicate hspice model using machine learning and get good accuracies.

Model -1:

Neural Network using Deep Learning Libraries

Converted the Dataset given from an xlsx file to a csv file.

- Divided the given data into Training Data and Testing Data in the ratio 75 : 25
- Created a three layer neural network and trained the data on different activation functions, loss functions and optimizers.
- When the activation function was ReLu, the loss function increased to infinity. Sigmoid activation function also had a very high loss. Finally, Tanh activation function was found to give the best results.
- Comparing different optimizers such as sgd, Adam and Adadelata, the best results were achieved by using Adadelata.
- Between different loss functions like mean_squared_error, mean_squared_logarithmic_error and mean_absolute_percentage_error, the mean_squared_error method had given the best loss function
- Compared the outputs to that of the HSPICE model and calculated the accuracy.

Results: Achieved a model analogous to HSPICE which is faster with an accuracy of 99% with the given data.

Code file :

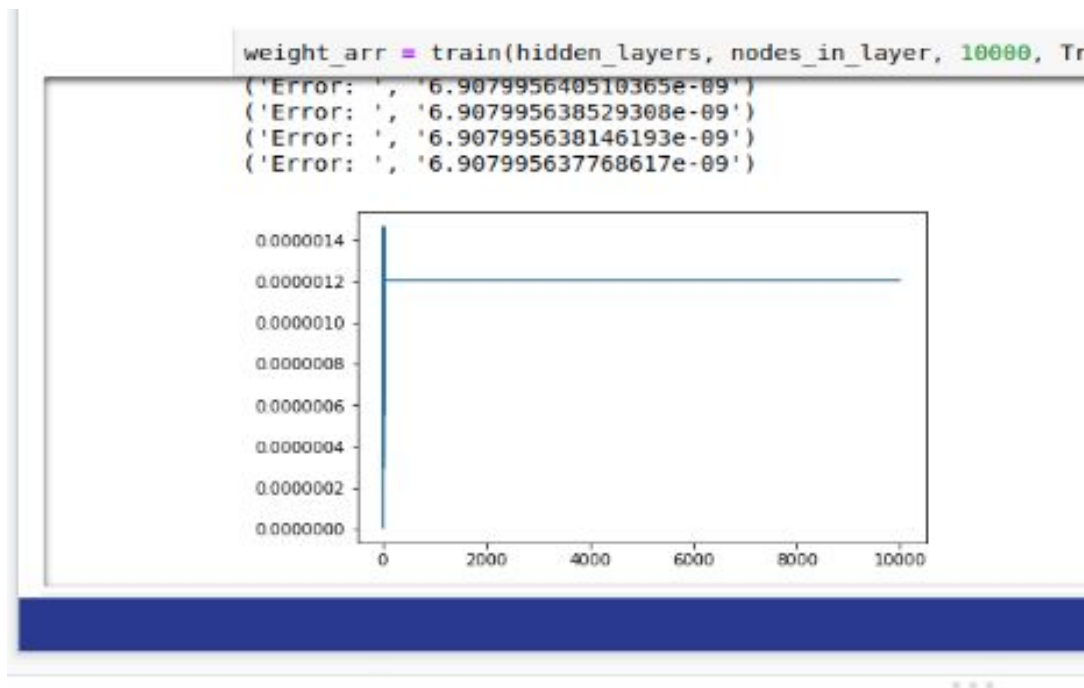
Command:

```
- 0s - loss: 9.4311e-15 - mean_absolute_error: 8.2893e-08 - categorical_accuracy: 1.0000
Epoch 112/128
- 0s - loss: 9.4366e-15 - mean_absolute_error: 8.2994e-08 - categorical_accuracy: 1.0000
Epoch 113/128
- 0s - loss: 9.4371e-15 - mean_absolute_error: 8.2941e-08 - categorical_accuracy: 1.0000
Epoch 114/128
- 0s - loss: 9.4467e-15 - mean_absolute_error: 8.2826e-08 - categorical_accuracy: 1.0000
Epoch 115/128
- 0s - loss: 9.4376e-15 - mean_absolute_error: 8.2841e-08 - categorical_accuracy: 1.0000
Epoch 116/128
- 0s - loss: 9.4333e-15 - mean_absolute_error: 8.2856e-08 - categorical_accuracy: 1.0000
Epoch 117/128
- 0s - loss: 9.4356e-15 - mean_absolute_error: 8.2922e-08 - categorical_accuracy: 1.0000
Epoch 118/128
- 0s - loss: 9.4350e-15 - mean_absolute_error: 8.2829e-08 - categorical_accuracy: 1.0000
Epoch 119/128
- 0s - loss: 9.4381e-15 - mean_absolute_error: 8.2891e-08 - categorical_accuracy: 1.0000
Epoch 120/128
- 0s - loss: 9.4289e-15 - mean_absolute_error: 8.2882e-08 - categorical_accuracy: 1.0000
Epoch 121/128
- 0s - loss: 9.4265e-15 - mean_absolute_error: 8.2833e-08 - categorical_accuracy: 1.0000
Epoch 122/128
- 0s - loss: 9.4469e-15 - mean_absolute_error: 8.3195e-08 - categorical_accuracy: 1.0000
Epoch 123/128
- 0s - loss: 9.4318e-15 - mean_absolute_error: 8.2820e-08 - categorical_accuracy: 1.0000
Epoch 124/128
- 0s - loss: 9.4335e-15 - mean_absolute_error: 8.3198e-08 - categorical_accuracy: 1.0000
Epoch 125/128
- 0s - loss: 9.4532e-15 - mean_absolute_error: 8.3216e-08 - categorical_accuracy: 1.0000
Epoch 126/128
- 0s - loss: 9.4311e-15 - mean_absolute_error: 8.2980e-08 - categorical_accuracy: 1.0000
Epoch 127/128
- 0s - loss: 9.4200e-15 - mean_absolute_error: 8.2829e-08 - categorical_accuracy: 1.0000
Epoch 128/128
- 0s - loss: 9.4180e-15 - mean_absolute_error: 8.2718e-08 - categorical_accuracy: 1.0000
Test loss: [9.491600157208145e-15, 8.333173270784756e-08, 1.0]
~/Doc/Dig/Project
```

Model -2 :

Neural Network without Deep Learning libraries

- We have created a neural network using only numpy library.
- Error plot looks like shown below



ker notes

- Data from the 16nm model file of XOR2 gate is taken and preprocessed initially, normalised by subtracting means, dividing with maximum of the data rows in the given dataset.
- Sigmoid, Relu, Softmax, tanh activation functions are coded, used
- Best results are found with tanh function and a Cross entropy loss is used.
- Code file : Neural_Net.ipynb
- Run it on jupyter notebook

Model - 3 :

Sickit libraries based Regression implementation

Linear Regression:

This had the maximum accuracy of 100% 100% 98% among all the models.

```
~/Doc.../Dig.../Pro.../temp ▶ master ● ? ▶ ./model.py --data XOR2_16nm_stat00.csv --split 0.33
Loaded dataset : (50000, 23)
Training data (33500, 20), Test data (16500, 20)
Accuracy: [100.      100.      98.91515152]
```

Ridge Regression:

Ridge regression addresses some of the problems of **Ordinary Least Squares** by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares,

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2$$

```
~/Doc.../Dig.../Pro.../temp ▶ master ● ? ▶ ./model.py --data XOR2_16nm_stat00.csv --split 0.33
Loaded dataset : (50000, 23)
Training data (33500, 20), Test data (16500, 20)
Accuracy: [100.      100.      8.1030303]
```

Lasso Regression:

Mathematically, it consists of a linear model trained with ℓ_1 prior as regularizer. The objective function to minimize is:

$$\min_w \frac{1}{2n_{\text{samples}}} \|Xw - y\|_2^2 + \alpha \|w\|_1$$

```
~/Doc.../Dig.../Pro.../temp > master ● ? > ./model.py --data XOR2_16nm_stat00.csv --split 0.33
Loaded dataset : (50000, 23)
Training data (33500, 20), Test data (16500, 20)
Accuracy: [100.      100.      98.76969697]
```

Lasso Lars Regression:

```
~/Doc.../Dig.../Pro.../temp > master ● ? > ./model.py --data XOR2_16nm_stat00.csv --split 0.33
Loaded dataset : (50000, 23)
Training data (33500, 20), Test data (16500, 20)
Accuracy: [100.      100.      98.77575758]
```

Model - 4 :

To reduce overfitting on the given dataset of 50,000 samples, introduction of dropout layers is done.

The term “dropout” refers to dropping out units (hidden and visible) in a neural network.

Introducing dropout layers:

Dropout rate is selected to be 0.25 after every neuron layer

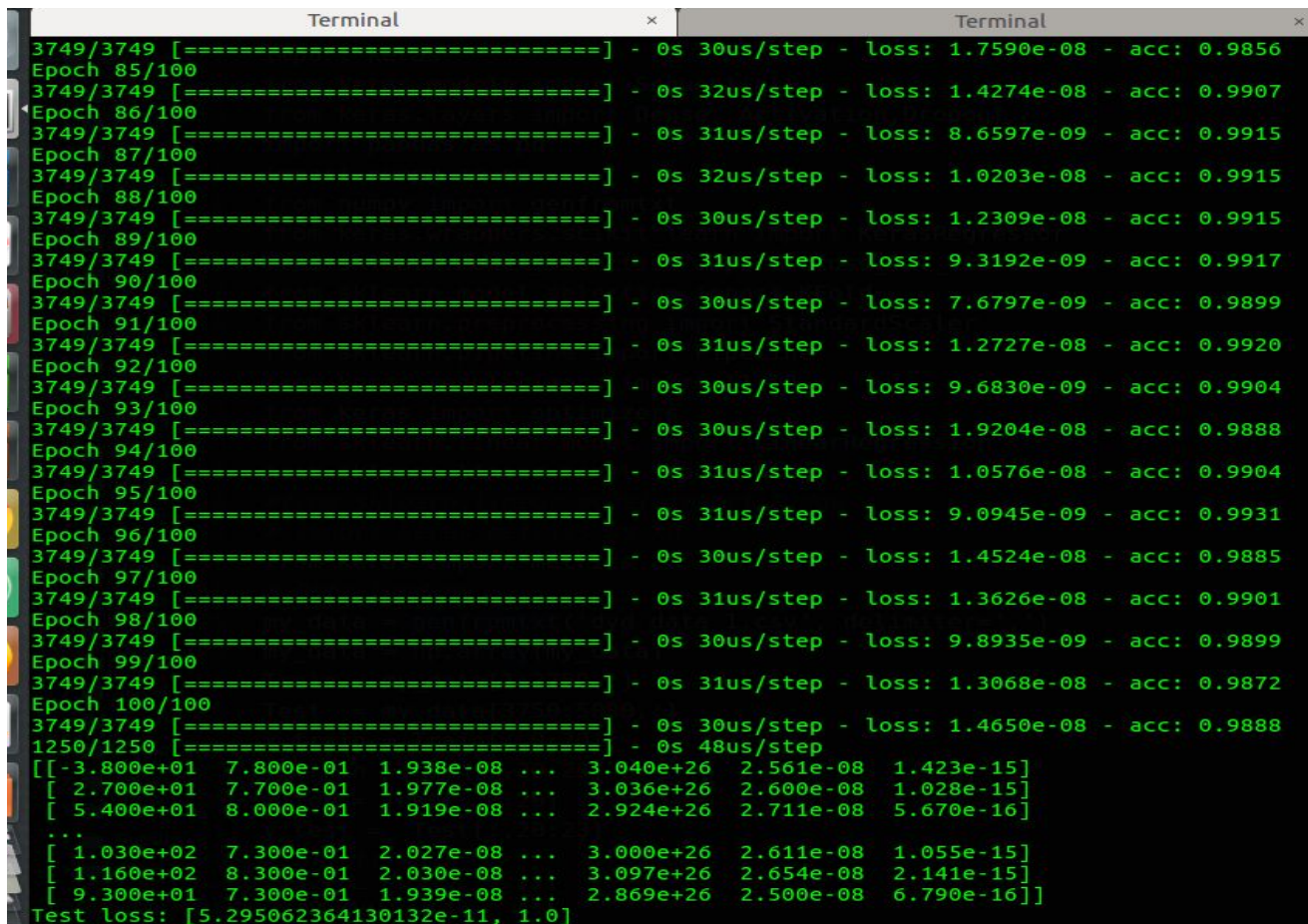
- This helps in increasing the robustness of the model
- Also the efficiency of the model to work on any dataset given is improved

Varying activation functions:

- Relu : This activation function in the inner 2 layers is used
- Softmax : Activation at the output layer

This combined usage of activation functions brought a good accuracy of about 98.9 to 99.2% .

Optimizer used : AdamaxFile : dvd_m22.py can run the file in folder with command “python3 <file_name>”



```
Terminal x Terminal x
3749/3749 [=====] - 0s 30us/step - loss: 1.7590e-08 - acc: 0.9856
Epoch 85/100
3749/3749 [=====] - 0s 32us/step - loss: 1.4274e-08 - acc: 0.9907
Epoch 86/100
3749/3749 [=====] - 0s 31us/step - loss: 8.6597e-09 - acc: 0.9915
Epoch 87/100
3749/3749 [=====] - 0s 32us/step - loss: 1.0203e-08 - acc: 0.9915
Epoch 88/100
3749/3749 [=====] - 0s 30us/step - loss: 1.2309e-08 - acc: 0.9915
Epoch 89/100
3749/3749 [=====] - 0s 31us/step - loss: 9.3192e-09 - acc: 0.9917
Epoch 90/100
3749/3749 [=====] - 0s 30us/step - loss: 7.6797e-09 - acc: 0.9899
Epoch 91/100
3749/3749 [=====] - 0s 31us/step - loss: 1.2727e-08 - acc: 0.9920
Epoch 92/100
3749/3749 [=====] - 0s 30us/step - loss: 9.6830e-09 - acc: 0.9904
Epoch 93/100
3749/3749 [=====] - 0s 30us/step - loss: 1.9204e-08 - acc: 0.9888
Epoch 94/100
3749/3749 [=====] - 0s 31us/step - loss: 1.0576e-08 - acc: 0.9904
Epoch 95/100
3749/3749 [=====] - 0s 31us/step - loss: 9.0945e-09 - acc: 0.9931
Epoch 96/100
3749/3749 [=====] - 0s 30us/step - loss: 1.4524e-08 - acc: 0.9885
Epoch 97/100
3749/3749 [=====] - 0s 31us/step - loss: 1.3626e-08 - acc: 0.9901
Epoch 98/100
3749/3749 [=====] - 0s 30us/step - loss: 9.8935e-09 - acc: 0.9899
Epoch 99/100
3749/3749 [=====] - 0s 31us/step - loss: 1.3068e-08 - acc: 0.9872
Epoch 100/100
3749/3749 [=====] - 0s 30us/step - loss: 1.4650e-08 - acc: 0.9888
1250/1250 [=====] - 0s 48us/step
[[-3.800e+01  7.800e-01  1.938e-08 ...  3.040e+26  2.561e-08  1.423e-15]
 [ 2.700e+01  7.700e-01  1.977e-08 ...  3.036e+26  2.600e-08  1.028e-15]
 [ 5.400e+01  8.000e-01  1.919e-08 ...  2.924e+26  2.711e-08  5.670e-16]
 ...
 [ 1.030e+02  7.300e-01  2.027e-08 ...  3.000e+26  2.611e-08  1.055e-15]
 [ 1.160e+02  8.300e-01  2.030e-08 ...  3.097e+26  2.654e-08  2.141e-15]
 [ 9.300e+01  7.300e-01  1.939e-08 ...  2.869e+26  2.500e-08  6.790e-16]]
Test loss: [5.295062364130132e-11, 1.0]
```

Neural network implementation using SGD optimizer:

A learning rate of 0.01 is used along with

Momentum parameter - 0.9 , this has been introduced to accelerate SGD in the relevant direction and dampens oscillations.

Decay element - 1e-6

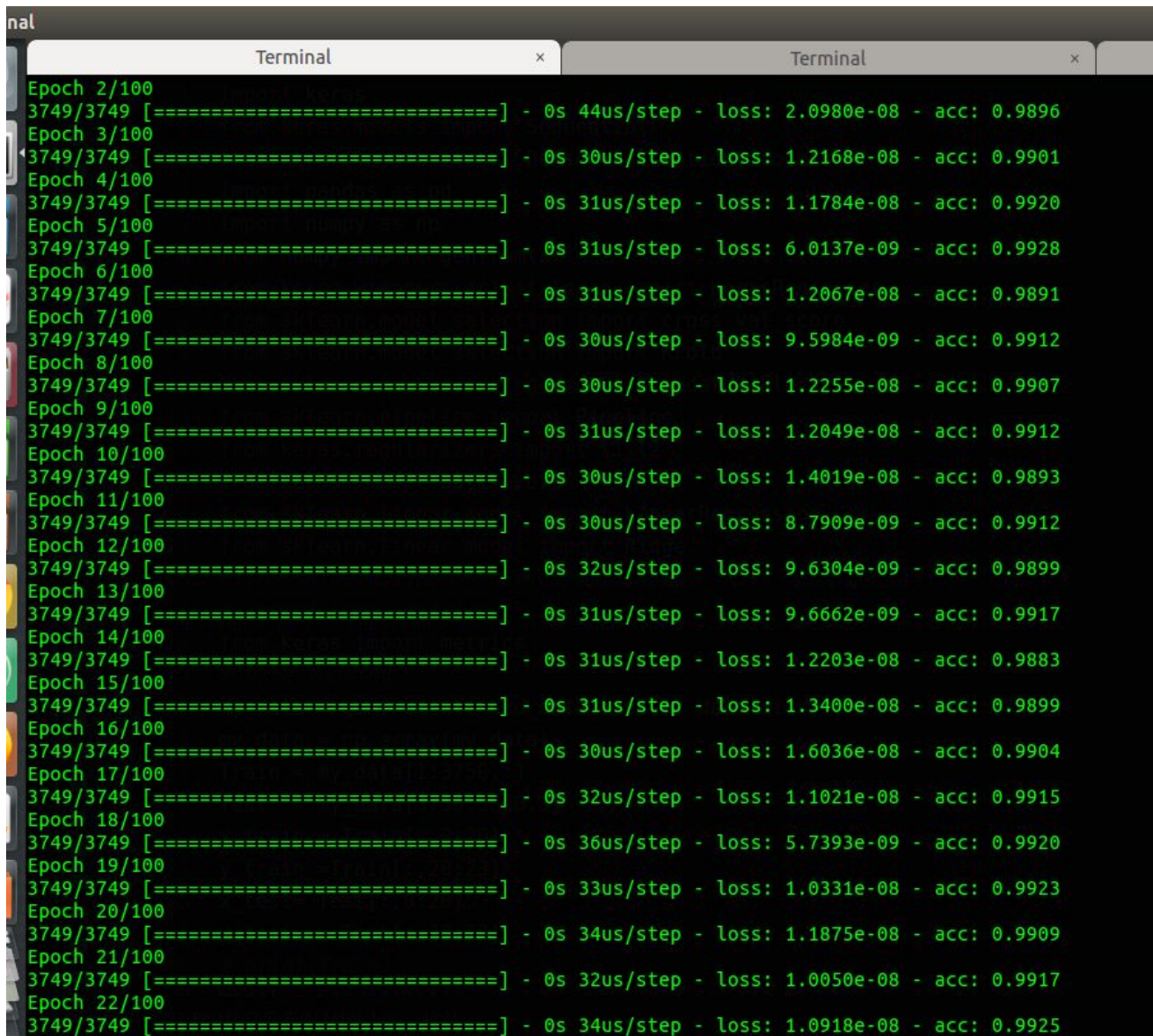
While using all these parameters, in order to clip the gradient from raising above a particular value and gain a consistent accuracy , gradient clipping is done.

A clipnorm of 1.0 is introduced to stop the gradient from growing above threshold.

Activation functions : relu,relu, softmax

File : dvd_m21.py

- can run the file in folder with command `python3 <file_name>`
- Results:



The image shows a terminal window with two tabs, both titled 'Terminal'. The active tab displays the output of a training process over 22 epochs. Each epoch's output consists of a progress bar (3749/3749), a time measurement (0s), a step time (e.g., 44us/step), a loss value, and an accuracy value (acc:). The loss values are in scientific notation, and the accuracy values are decimal numbers. The training appears to be converging as the loss decreases and accuracy increases over time.

Epoch	Progress	Time	Step Time	Loss	Accuracy
2/100	3749/3749	0s	44us/step	2.0980e-08	0.9896
3/100	3749/3749	0s	30us/step	1.2168e-08	0.9901
4/100	3749/3749	0s	31us/step	1.1784e-08	0.9920
5/100	3749/3749	0s	31us/step	6.0137e-09	0.9928
6/100	3749/3749	0s	31us/step	1.2067e-08	0.9891
7/100	3749/3749	0s	30us/step	9.5984e-09	0.9912
8/100	3749/3749	0s	30us/step	1.2255e-08	0.9907
9/100	3749/3749	0s	31us/step	1.2049e-08	0.9912
10/100	3749/3749	0s	30us/step	1.4019e-08	0.9893
11/100	3749/3749	0s	30us/step	8.7909e-09	0.9912
12/100	3749/3749	0s	32us/step	9.6304e-09	0.9899
13/100	3749/3749	0s	31us/step	9.6662e-09	0.9917
14/100	3749/3749	0s	31us/step	1.2203e-08	0.9883
15/100	3749/3749	0s	31us/step	1.3400e-08	0.9899
16/100	3749/3749	0s	30us/step	1.6036e-08	0.9904
17/100	3749/3749	0s	32us/step	1.1021e-08	0.9915
18/100	3749/3749	0s	36us/step	5.7393e-09	0.9920
19/100	3749/3749	0s	33us/step	1.0331e-08	0.9923
20/100	3749/3749	0s	34us/step	1.1875e-08	0.9909
21/100	3749/3749	0s	32us/step	1.0050e-08	0.9917
22/100	3749/3749	0s	34us/step	1.0918e-08	0.9925