**MEASURE ENERGY CONSUMPTION PREPROCESSING IN DATASET**

[TensorFlow](https://tensorflow.org/) and [Neural Designer](https://www.neuraldesigner.com/) are popular machine learning platforms developed by [Google](https://research.google/teams/brain/) and [Artelnics](https://www.artelnics.com/), respectively.

Although all those frameworks are based on neural networks, they present essential differences in functionality, usability, performance, consumption, etc.

This post compares the energy consumption of TensorFlow and Neural Designer using the GPU for an approximation benchmark.

As we will see, Neural Designer consumes **42**% less than its competitor machine learning platform.

In this article, we provide all the steps you need to reproduce the result using the [free trial](https://www.neuraldesigner.com/free-trial) of Neural Designer.

**Contents:**

* [Introduction](https://www.neuraldesigner.com/blog/energy-consumption-comparison/#Introduction).
* [Benchmark application](https://www.neuraldesigner.com/blog/energy-consumption-comparison/#BenchmarkApplication).
* [Reference computer](https://www.neuraldesigner.com/blog/energy-consumption-comparison/#ReferenceComputer).
* [Reference electricity consumption meter](https://www.neuraldesigner.com/blog/energy-consumption-comparison/#Referenceelectricityconsumptionmeter).
* [Results](https://www.neuraldesigner.com/blog/energy-consumption-comparison/#Results).
* [Conclusions](https://www.neuraldesigner.com/blog/energy-consumption-comparison/#Conclusions).

## **Introduction**

Two of the most essential features of machine learning platforms are their training speed and the total amount of energy consumed during this process.

In most cases, modeling huge data sets is very expensive in computational terms, which leads to a high economic cost of neural network training and a high environmental impact.

Thus, this article aims to measure the GPU energy consumption of TensorFlow and Neural Designer for a benchmark application. Also, a couple of instructions are given to enable anyone to repeat this one or a similar benchmark and check on their own the fantastic results obtained when Neural Designer is used.

The following table summarizes the technical features of these tools that might impact their GPU performance.

|  | **TensorFlow** | **Neural Designer** |
| --- | --- | --- |
| **Written in** | C++, CUDA, Python | C++, CUDA |
| **Interface** | Python | Graphical User Interface |
| **Differentiation** | Automatic | Analytical |

The above table shows that TensorFlow is programmed in C++ and Python, whereas Neural Designer is entirely programmed in C++.

Interpreted languages like Python have advantages over compiled languages like C ++, such as their ease of use.

However, the performance of Python is generally lower than that of C++. Indeed, Python takes significant time to interpret sentences during the program’s execution.

On the other hand, TensorFlow uses automatic differentiation, while Neural Designer uses analytical differentiation.

As before, automatic differentiation has some advantages over analytical differentiation. Indeed, it simplifies obtaining the gradient for new architectures or loss indices.

However, the performance of automatic differentiation is, in general, lower than that of analytical differentiation:  
The first derives the gradient during the program’s execution, while the second has that formula pre-calculated.

Next, we use TensorFlow and Neural Designer to measure the energy consumption for a benchmark problem on a reference computer. The results produced by these platforms are then compared.

## **Benchmark application**

The first step is to choose a benchmark application that is general enough to conclude the performance of the machine learning platforms. As previously stated, we will train a neural network that approximates a set of input-target samples.

In this regard, an approximation application is defined by a data set, a neural network, and an associated training strategy.  
The following table uniquely defines these three components.

|  |  |
| --- | --- |
| **Data set** | * Benchmark: Rosenbrock * Inputs number: 1000 * Targets number: 1 * Samples number: 1000000 * File size: 22 GB ([download](https://www.neuraldesigner.com/files/datasets/R_new.rar)) |
| **Neural network** | * Layers number: 2 * Layer 1:   + -Type: Perceptron (Dense)   + -Inputs number: 1000   + -Neurons number: 1000   + -Activation function: Hyperbolic tangent (tanh) * Layer 2:   + -Type: Perceptron (Dense)   + -Inputs number: 1000   + -Neurons number: 1   + -Activation function: Linear * Initialization: Random uniform [-1,1] |
| **Training strategy** | * Loss index:   + -Error: Mean Squared Error (MSE)   + -Regularization: None * Optimization algorithm:   + -Algorithm: Adaptive Moment Estimation (Adam)   + -Batch size: 1000   + -Maximum epochs: 20000 |

Once the TensorFlow and Neural Designer applications have been created, we must run them.

## **Reference computer**

The next step is to choose the computer in which the neural network will be trained with TensorFlow and Neural Designer.  
The table below shows the features of the computer used for this instance.

|  |  |
| --- | --- |
| **Operating system:** | Windows 11 Home 64-bit |
| **Processor:** | Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 3192 Mhz, 6 Core(s), 12 Logical Processor(s) |
| **Physical RAM:** | 31.9 GB |
| **Device (GPU):** | NVIDIA GeForce GTX 1050 Ti |

Once the computer has been selected, we install TensorFlow (2.1.0) and Neural Designer (5.9.9) on it.

Below, the TensorFlow code used is shown.

**import** **tensorflow** **as** **tf**

**import** **pandas** **as** **pd**

**import** **time**

**import** **numpy** **as** **np**

**from** **tensorflow.keras.utils** **import** Sequence

#read data float32

filename = "C:/Users/Usuario/Downloads/rosenbrock.csv"

df\_test = pd.read\_csv(filename, nrows=**100**)

float\_cols = [c **for** c **in** df\_test **if** df\_test[c].dtype == "float64"]

float32\_cols = {c: np.float32 **for** c **in** float\_cols}

data = pd.read\_csv(filename, engine='c', dtype=float32\_cols)

x = data.iloc[:,:-**1**].values

y = data.iloc[:,[-**1**]].values

initializer = tf.keras.initializers.RandomUniform(minval=-**1.**, maxval=**1.**)

#build model

model = tf.keras.models.Sequential([tf.keras.layers.Dense(**1000**,activation = 'tanh', kernel\_initializer = initializer, bias\_initializer=initializer),

tf.keras.layers.Dense(**1**, activation = 'linear', kernel\_initializer = initializer, bias\_initializer=initializer)])

#compile model

model.compile(optimizer='adam', loss = 'mean\_squared\_error')

#train model

**class** **DataGenerator**(Sequence):

**def** **\_\_init\_\_**(self, x\_set, y\_set, batch\_size):

self.x, self.y = x\_set, y\_set

self.batch\_size = batch\_size

**def** **\_\_len\_\_**(self):

**return** int(np.ceil(len(self.x) / float(self.batch\_size)))

**def** **\_\_getitem\_\_**(self, idx):

batch\_x = self.x[idx \* self.batch\_size:(idx + **1**) \* self.batch\_size]

batch\_y = self.y[idx \* self.batch\_size:(idx + **1**) \* self.batch\_size]

**return** batch\_x, batch\_y

train\_gen = DataGenerator(x, y, **1000**)

start\_time = time.time()

**with** tf.device('/gpu:0'):

history = model.fit(train\_gen, epochs=**20000**)

**print**("Training time: ", round(time.time() - start\_time), " seconds")

## **Reference electricity consumption meter**

This section describes the device used for the energy consumption measurements so that the reader can reproduce the results obtained in the following section with maximum accuracy.

|  |  |
| --- | --- |
| **Device model** | Perel E305EM5-G |

## **Results**

The last step is to run the benchmark application on the selected machine with TensorFlow and Neural Designer and compare the energy consumed by those platforms during training.

The following figure shows the training time with TensorFlow.

As we can see, TensorFlow takes 30:14:30 to train the neural network for 20000 epochs (5.44 seconds/epoch).  
The final mean squared error is 0.0003. The overall energy consumption of the training process is 4.5 kWh, as shown below.

Finally, the following figure shows the training time with Neural Designer.

Neural Designer takes 21:03:43 to train the neural network for 20000 epochs (3.79 seconds/epoch). During that time, it reaches a mean squared error of 0.023. The overall energy consumption of the training process is 2.6 kWh, as shown below.

The following table summarizes the the most important metrics the two machine learning platforms yielded.

|  | **TensorFlow** | **Neural Designer** |
| --- | --- | --- |
| **Training time** | 30:14:30 | 21:03:43 |
| **Epoch time** | 5.44 seconds/epoch | 3.79 seconds/epoch |
| **Training speed** | 183,824 samples/second | 263.852 samples/second |
| **Total energy consumed** | 4.5kWh | 2.6 kWh |

The following chart depicts the energy consumed using TensorFlow and Neural Designer graphically in this case.

**Electric energy consumption**

TensorFlow

4.50 kWh

4.50 kWh

Neural Designer

2.60 kWh

2.60 kWh

As we can see, the energy consumption of Neural Designer for this application is **42** % lower than that of TensorFlow.

## **Conclusions**

Neural Designer is entirely written in C ++, uses analytical differentiation, and has been optimized to minimize the number of operations during training.

As a result, its energy consumption during the training process using Neural Designer is **42** % lower than that using TensorFlow.

To reproduce these results, download the [free trial](https://www.neuraldesigner.com/free-trial) of Neural Designer and follow the steps described in this article.