Artificial Neural Networks and Applications

Ioannis Liodis[1], *Supervisor* Prof. Nikolaos Stergioulas [1]

Aristotle University of Thessaloniki[1]

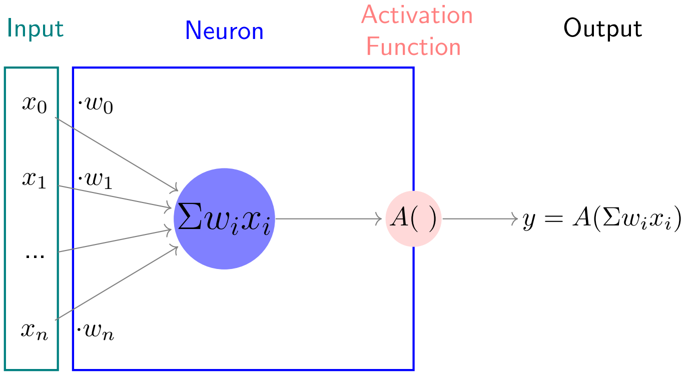
Bachelor Thesis
2022 - 2023

Outline

Introduction to ANNs

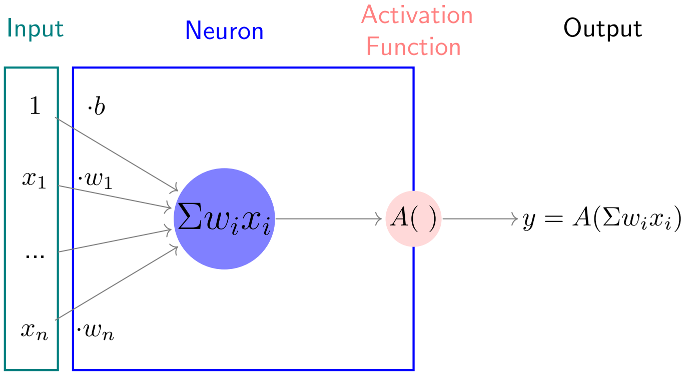
Structure

The neuron is the building block of neural networks.

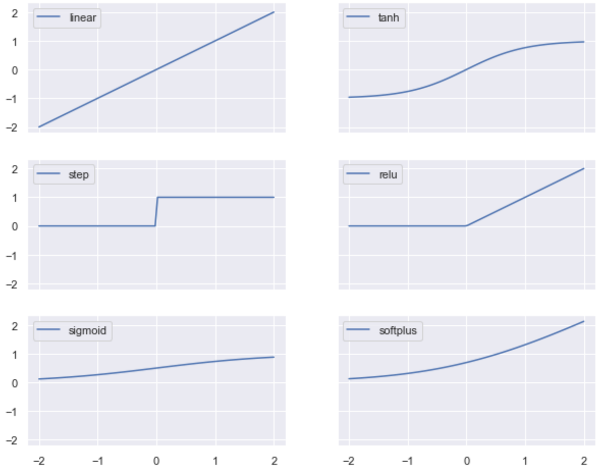


Structure

The neuron is the building block of neural networks.



Activation Functions



Tensorflow activation functions .

Training for a single Neuron

A single neuron can be easily trained to simulate a linear function.

Example

$$f(x_1, x_2, x_3) = ax_1 + bx_2 + cx_3$$

$$y = A\left(\sum_{i=0}^{3} w_i x_i\right)$$

The initial vector $\vec{w}$ is random. Goal: $\vec{w} \to (0, a, b, c)$ via consecutive updates and check how close y gets to the true $\tilde{y} = f(\vec{x}_{input}) = y_{true}$.

To measure this we need a loss function.

Tensorflow loss functions.

Training for a single Neuron - Loss Function

A simple loss function:

$$L(y) = (y - \tilde{y})^2$$

Choosing the identity activation function:

$$y = A\left(\sum_{i=0}^{3} w_i x_i\right) = \sum_{i=0}^{3} w_i x_i$$

And:

$$\tilde{y} = f(\vec{x}_{input})$$

We want to know the weight dependence on the loss function, in order to minimize the loss.

Training for a single Neuron - Loss Minimisation

So we calculate the partial derivative:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_i} = 2(y - \tilde{y})x_i$$

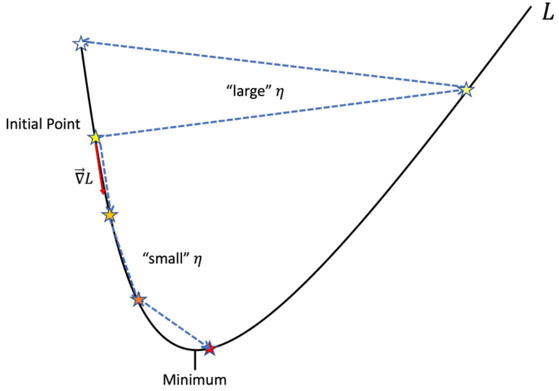
Back-propagation is the process of finding those derivatives. Then we proceed to updating the weights proportionally to their influence:

$$w_i \to w_i - \eta\frac{\partial L}{\partial w_i}$$

where η is called learning rate.

Training for a single Neuron - Loss Minimisation

The loss minimisation method is called gradient descent.



Note: We have to be careful when choosing η!

Training for a single Neuron - Gradient Descent

Types of gradient descent (fixed learning rate):

- Stochastic gradient descent - STG
- Batch gradient descent
- Mini-batch gradient descent

Optimizer algorithms (flexible learning rate):

- Momentum
- Nesterov momentum
- ADAptive momentum estimation - ADAM

Training for a single Neuron - Epochs

Steps so far for a single neuron:

- Random choice of $\vec{w}$
- Choice of an activation function
- Choice of a loss function
- Calculation of the output $A(\vec{x} \cdot \vec{w})$
- Back propagation and loss minimisation

Training for a single Neuron - Epochs

Steps so far for a single neuron:

- Random choice of $\vec{w}$
- Choice of an activation function
- Choice of a loss function
- Calculation of the output $A(\vec{x} \cdot \vec{w})$
- Back propagation and loss minimisation

} not arbitrary combination

} one epoch

This process has to be repeated to minimise the loss!

Note: Activation and loss function are chosen empirically.

Relation between Activation and Loss function

Activation and Loss functions cannot be chosen independently!
Since

$$L(y) = L(A(\Sigma))$$

the co-domain of A cannot be a larger set than the domain of L.
In other words:

$$L : X_L \to Y_L$$

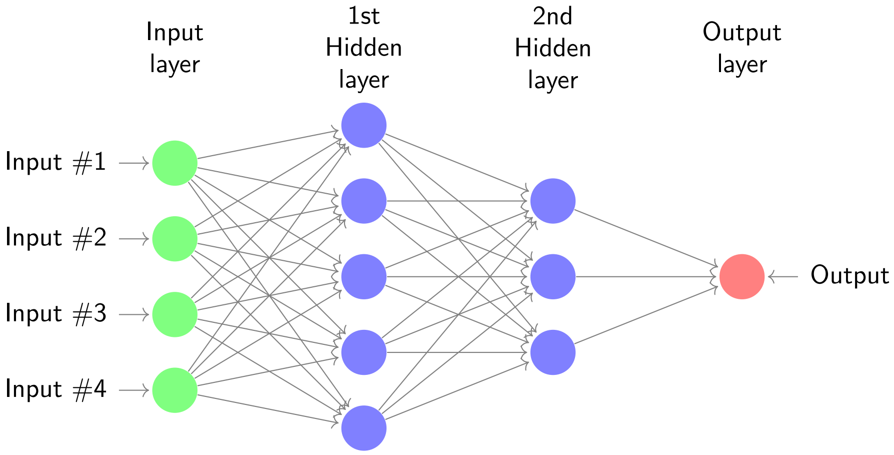
$$A : X_A \to Y_A$$

thus it must hold:

$$Y_A \subseteq X_L$$

Structure

Until now we focused on one neuron. A neural network consists of multiple neurons, grouped in layers.



Hyper parameters

There are several parameters that define the neural network, regarding:

The structure of the Network

- Number of hidden layers and neurons
- Weight initialization
- Activation function

The training algorithm

- Learning rate
- Epochs
- Batch size

Training

The training requires a data set with the real inputs and outputs. This data set is divided into:

- **Training set** (usually 70% of the whole data set): These data are used to train the network, which means that in each epoch the goal is to minimise the loss between the real output values of the training set and the predicted values of the network.
- **Testing set** (usually 30% of the whole data set): These data are used to test the network after it is trained, which means that after all epochs the predicted values are compared to the real output values of the testing set.
- **Validation set** (usually 20% of the training set): These data are used to validate the network while it is being trained, which means that after each epoch the loss between the real output values of the validation and the predicted values of the network is calculated.

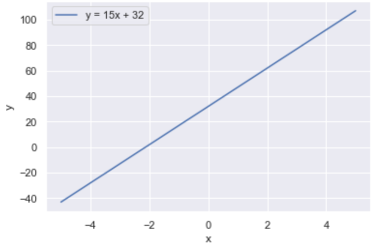
Note: The network is trained only by the training set.

Linear regression

Topic: Make a neural network to predict the behavior of a line

$$y = f(x) = ax + b$$

The simplest Neural Network: one input and one output.
First we need a data set.

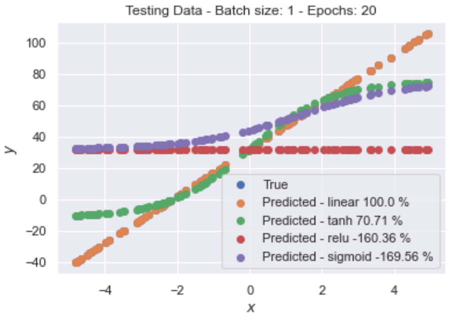


$$\text{Input} \longrightarrow x \xrightarrow[+b]{\cdot a} y \longleftarrow \text{Output}$$

Linear regression

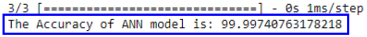
After creating the data set (here 300 points) we devide it into training (70 %), testing (30 %) and validation (0 %).
We need to choose the hyper parameters:

- One input, one output
- Random weight initialization
- **Activation function?**
- Learning rate (adam optimizer)
- Epochs = 20
- Batch size = 1

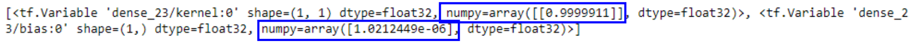


Linear regression

The right choice of the activation function is obvious for this case.
But when it comes to checking the weights we notice something weird...



```
3/3 [==============================] - 0s 1ms/step
The Accuracy of ANN model is: 99.99740763178218
```



```
[<tf.Variable 'dense_23/kernel:0' shape=(1, 1) dtype=float32, numpy=array([[0.9999911]], dtype=float32)>, <tf.Variable 'dense_2
3/bias:0' shape=(1,) dtype=float32, numpy=array([1.0212449e-06], dtype=float32)>]
```

$$a \approx 1 \neq 15$$

$$b \approx 0 \neq 32$$

$$Accuracy \approx 100\%$$

Why?

A "standard" error

When splitting the data set we also **standardized** them. This step is important because it improves the training speed and helps the gradient descent algorithm!

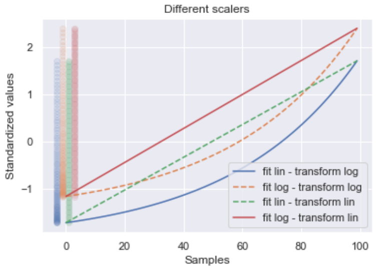
How does sklearn.preprocessing.StandardScaler work? The standardized data x'_i (i.e. $x_1 = x$ and $x_2 = y$ for this example) are given by the transformation

$$x'_i = \frac{x_i - u_i}{s_i}$$

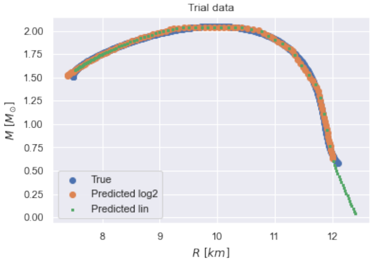
where u_i is the mean and s_i is the standard deviation of the training samples x_i (i.e. 210 samples for this example).

We first fit a StandardScaler on the training set and then implement the **same** StandardScaler on the testing/prediction set. Otherwise...

A "standard" error



(a) StandardScaler for linspace and logspace with (1,10,100)

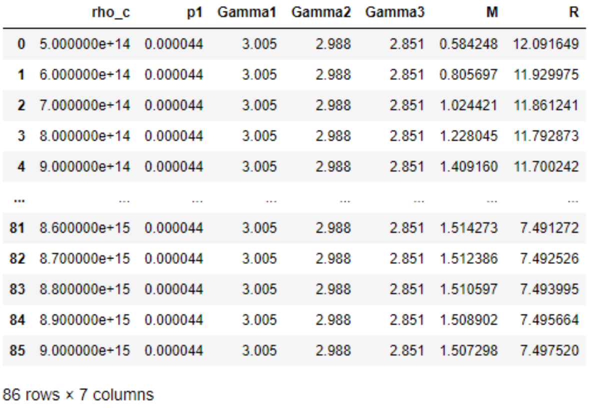


(b) Piecewise Polytropic EOS (SLy) - TOV

TOV Application

Data

pyTOVpp - SLy EOS - linear space:



	rho_c	p1	Gamma1	Gamma2	Gamma3	M	R
0	5.000000e+14	0.000044	3.005	2.988	2.851	0.584248	12.091649
1	6.000000e+14	0.000044	3.005	2.988	2.851	0.805697	11.929975
2	7.000000e+14	0.000044	3.005	2.988	2.851	1.024421	11.861241
3	8.000000e+14	0.000044	3.005	2.988	2.851	1.228045	11.792873
4	9.000000e+14	0.000044	3.005	2.988	2.851	1.409160	11.700242
...	...	...	...	...	...	...	...
81	8.600000e+15	0.000044	3.005	2.988	2.851	1.514273	7.491272
82	8.700000e+15	0.000044	3.005	2.988	2.851	1.512386	7.492526
83	8.800000e+15	0.000044	3.005	2.988	2.851	1.510597	7.493995
84	8.900000e+15	0.000044	3.005	2.988	2.851	1.508902	7.495664
85	9.000000e+15	0.000044	3.005	2.988	2.851	1.507298	7.497520

86 rows × 7 columns

Trial 1

- **3 hidden layers**: 5-10-5 neurons.
- **Activation functions**:relu-relu-relu vs relu-tanh-tanh.
- **Optimizer**: adam.
- **batch size**: 3.
- **epochs**: 100.

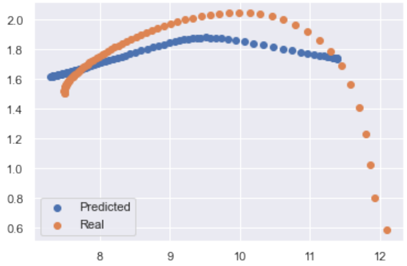


Figure: relu-relu-relu

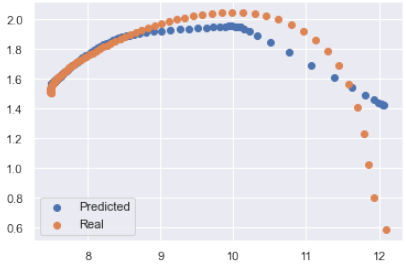
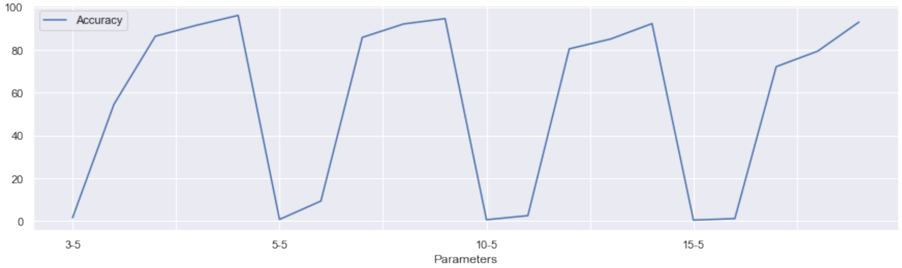


Figure: relu-tanh-tanh

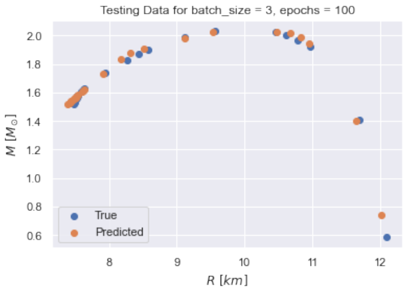
Trial 2

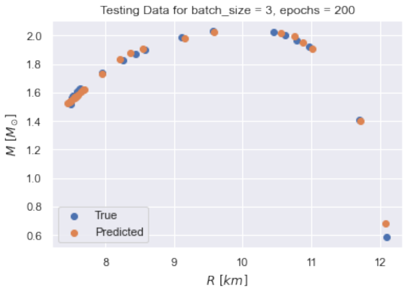
- **3 hidden layers**: 5-10-10 neurons.
- **Activation functions**: relu-tanh-relu.
- **Optimizer**: adam.
- **Faster loss decrease!**



Trial 2

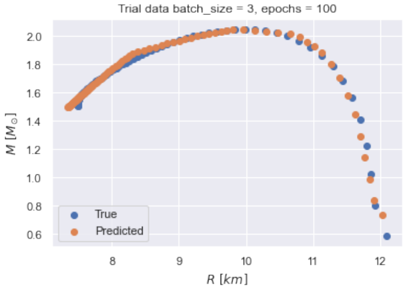
- **batch size**: 3.
- **epochs**: 100 vs 200.
- **Accuracy**: over 99%.

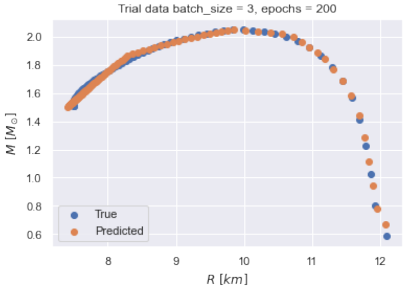




Trial 2

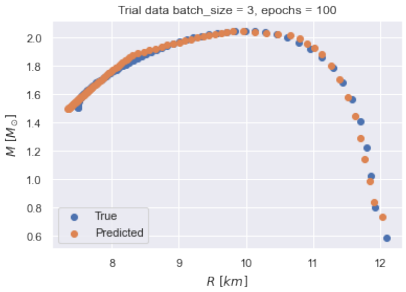
- **batch size**: 3.
- **epochs**: 100 vs 200.
- **Accuracy**: over 99%.

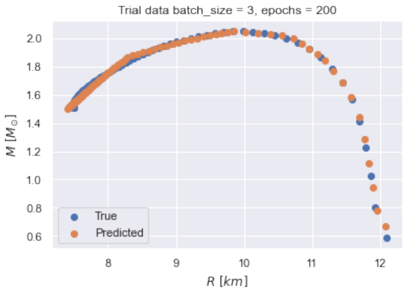




Remarks

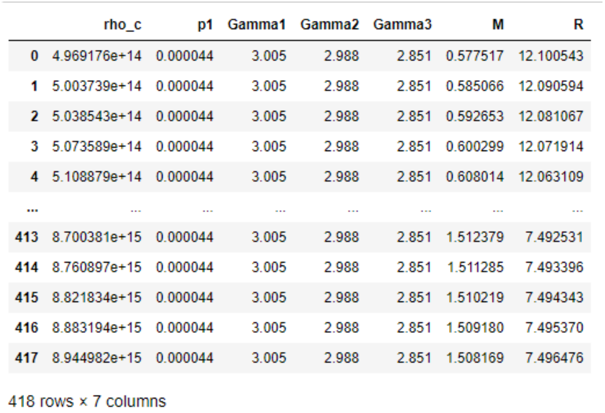
- **Linear** space for ρ_c, maybe **logspace** needed.
- **More epochs?**
- **Other activation functions?**





New Data

pyTOVpp - SLy EOS - log2 space and more points:

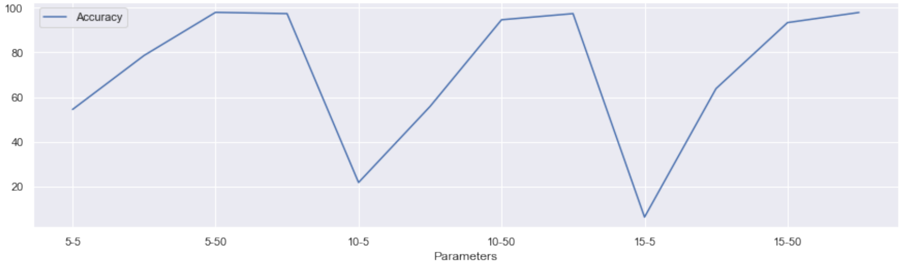


	rho_c	p1	Gamma1	Gamma2	Gamma3	M	R
0	4.969176e+14	0.000044	3.005	2.988	2.851	0.577517	12.100543
1	5.003739e+14	0.000044	3.005	2.988	2.851	0.585066	12.090594
2	5.038543e+14	0.000044	3.005	2.988	2.851	0.592653	12.081067
3	5.073589e+14	0.000044	3.005	2.988	2.851	0.600299	12.071914
4	5.108879e+14	0.000044	3.005	2.988	2.851	0.608014	12.063109
...	...	...	...	...	...	...	...
413	8.700381e+15	0.000044	3.005	2.988	2.851	1.512379	7.492531
414	8.760897e+15	0.000044	3.005	2.988	2.851	1.511285	7.493396
415	8.821834e+15	0.000044	3.005	2.988	2.851	1.510219	7.494343
416	8.883194e+15	0.000044	3.005	2.988	2.851	1.509180	7.495370
417	8.944982e+15	0.000044	3.005	2.988	2.851	1.508169	7.496476

418 rows × 7 columns

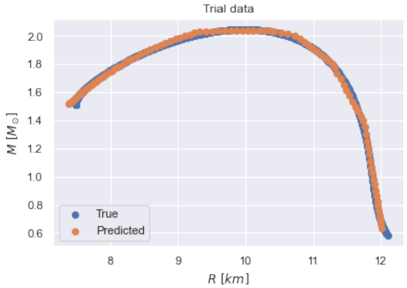
Trial 3

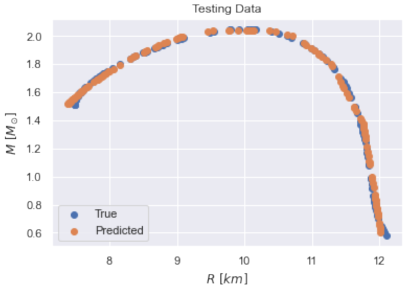
- **3 hidden layers**: 5-10-10 neurons.
- **Activation functions**: relu-tanh-relu.
- **Optimizer**: adam.
- **Extremely faster loss decrease!**



Trial 3 - The best choice

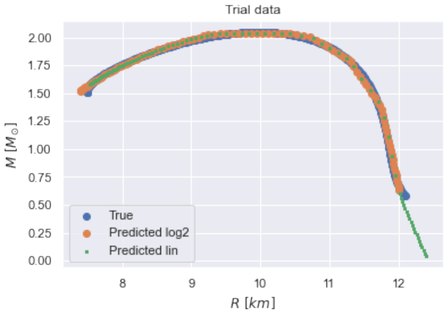
- **batch size**: 5.
- **epochs**: 50.
- **Accuracy**: over 99.7%.





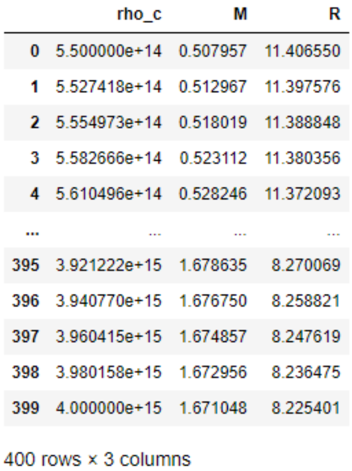
Trial 3 - The error

Linear vs **Log2** space.



Data - A more systematic tuning investigation

pyTOVtab - FPS PP HnG EOS - log space:



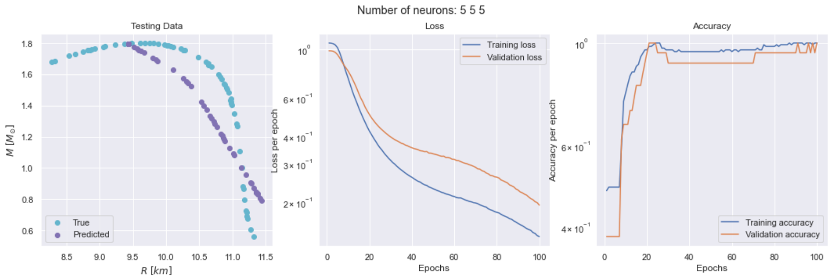
	rho_c	M	R
0	5.500000e+14	0.507957	11.406550
1	5.527418e+14	0.512967	11.397576
2	5.554973e+14	0.518019	11.388848
3	5.582666e+14	0.523112	11.380356
4	5.610496e+14	0.528246	11.372093
...	...	...	...
395	3.921222e+15	1.678635	8.270069
396	3.940770e+15	1.676750	8.258821
397	3.960415e+15	1.674857	8.247619
398	3.980158e+15	1.672956	8.236475
399	4.000000e+15	1.671048	8.225401

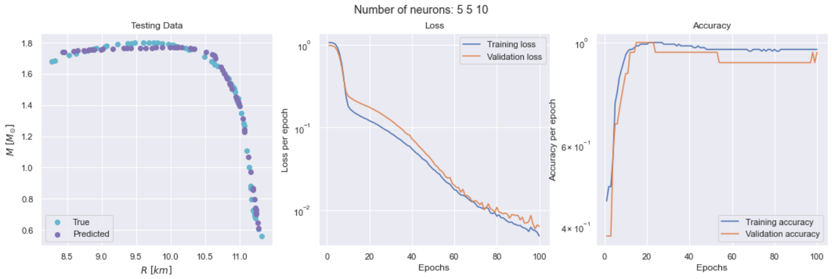
400 rows × 3 columns

Investigations

- **Straight lines correlation to number of neurons**
  hidden layers = 3, neuron choices = [5,10,15], activation functions = [tanh, relu, relu], batch size = 5, epochs = 100, training = 70 %, testing = 30 %, validation = 15 , Data length = 200.
- **Searching for the best number of neurons**
  hidden layers = 4, neuron choices = [15,20,25], activation functions = [tanh, tanh, relu, tanh], batch size = 5, epochs = 100, training = 70 %, testing = 30 %, validation = 15 , Data length = 200.
- **Searching for the best activation functions**
  hidden layers = 4, neurons = [25,25,20,25], activation functions choices = [relu, tanh], batch size = 5, epochs = 100, training = 70 %, testing = 30 %, validation = 15 , Data length = 200.
- **Searching for the best tuning**
  hidden layers = 3, neuron choices = [15,20], activation functions choices = [relu, tanh], batch size = 5, epochs = 150, training = 70 %, testing = 30 %, validation = 20 , Data length = 400.

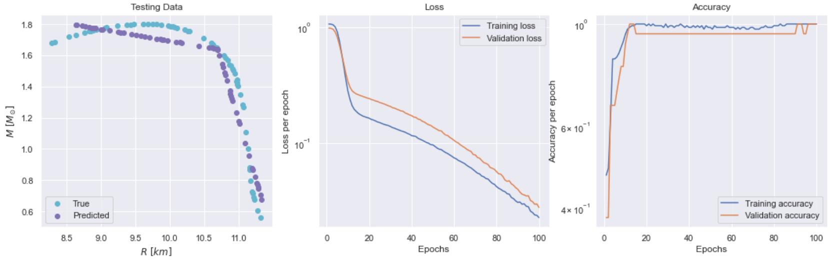
Straight lines correlation to number of neurons



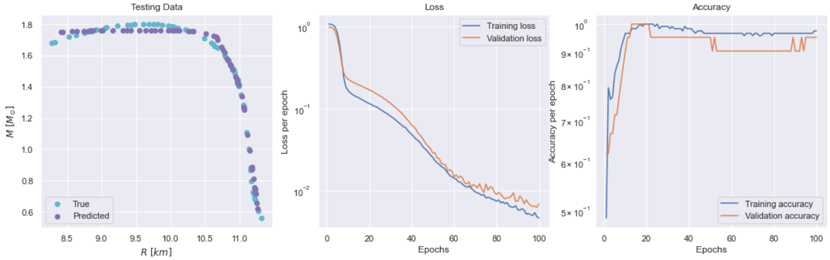


Straight lines correlation to number of neurons

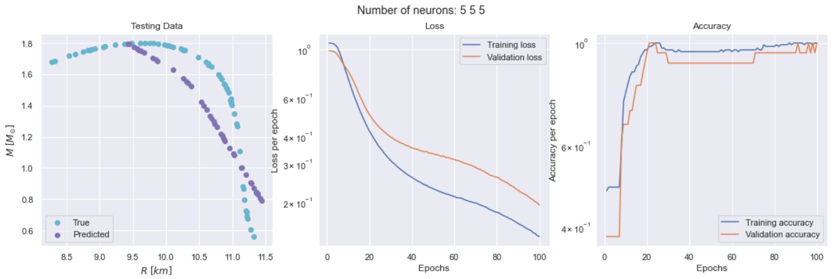
Number of neurons: 15 5 5

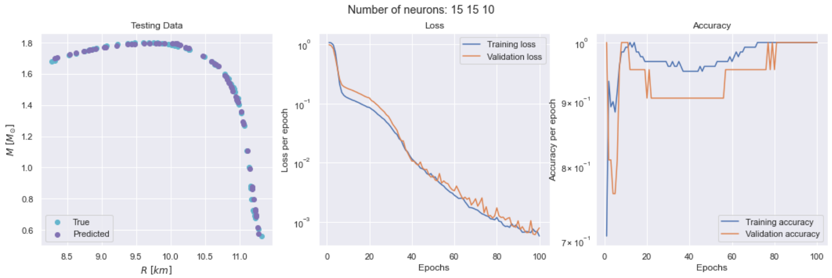


Number of neurons: 5 5 15



Straight lines correlation to number of neurons

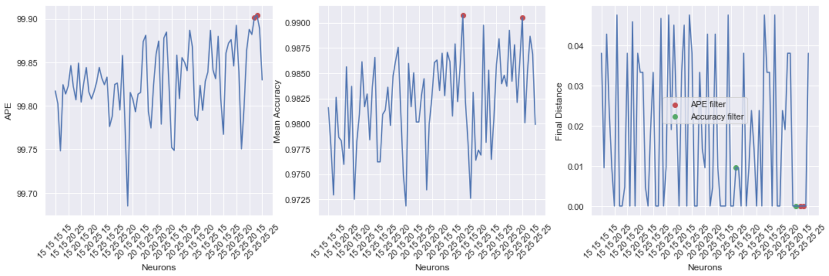




Searching for the best number of neurons

Filters

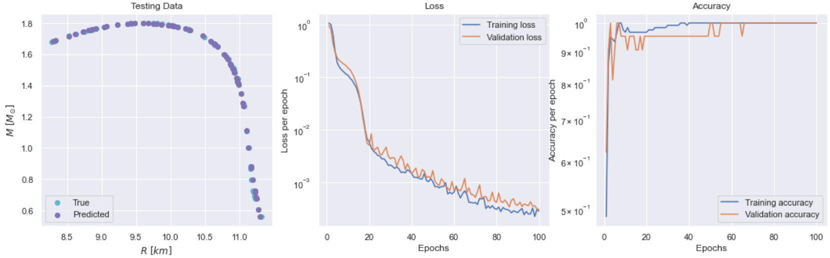
- APE > 99.9
- Accuracy > 0.99



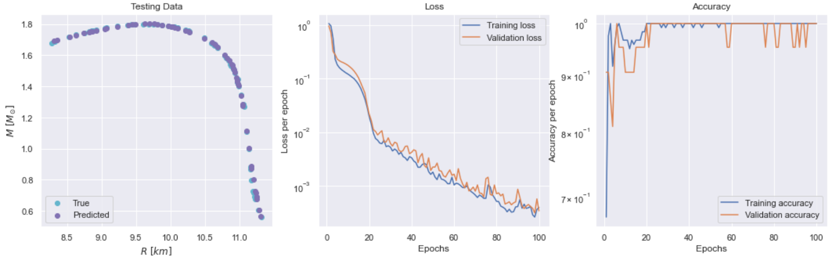
Conclusion: The best choice is not max neurons!

Searching for the best number of neurons

Number of neurons: 25 25 20 25



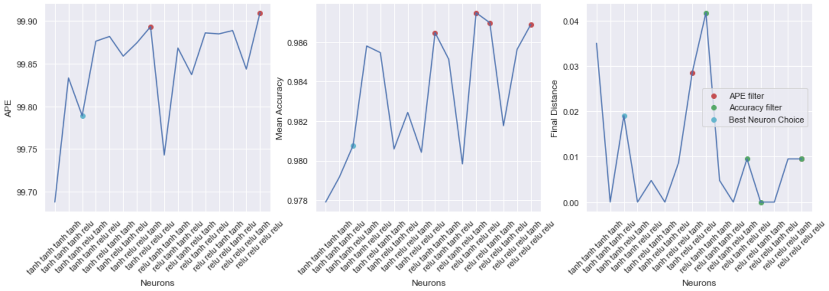
Number of neurons: 20 25 25 20



Searching for the best activation functions

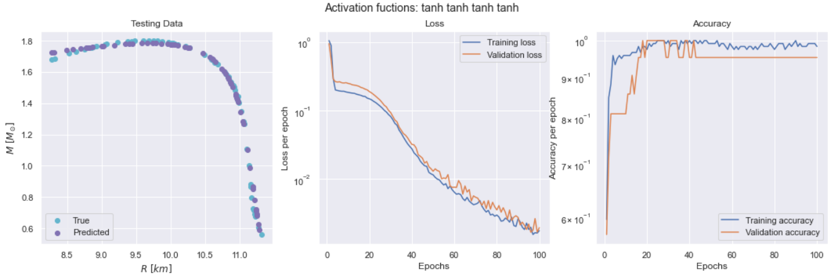
Filters

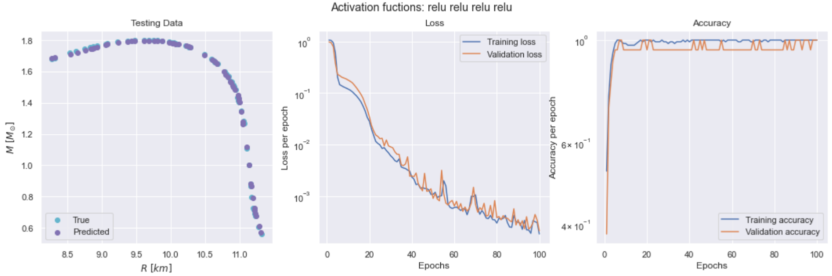
- APE > 99.89
- Accuracy > 0.986



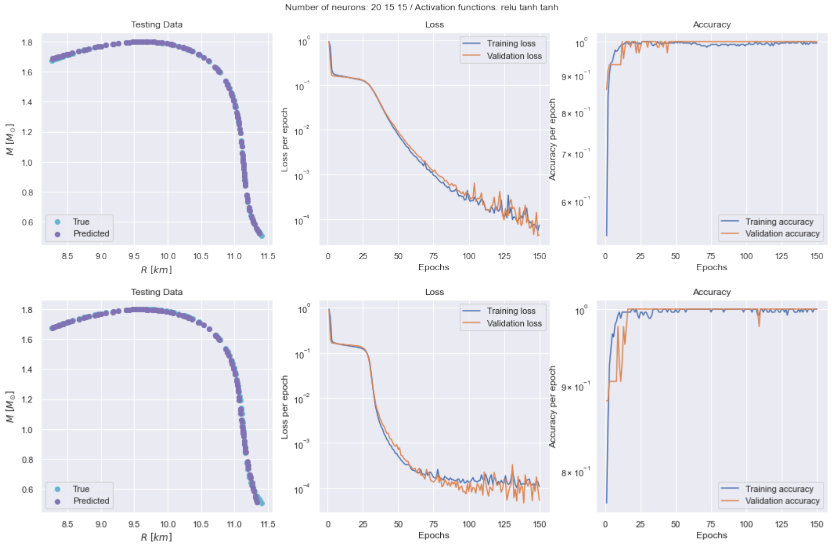
Conclusion: The best choice is not all tanh or all relu!

Searching for the best activation functions



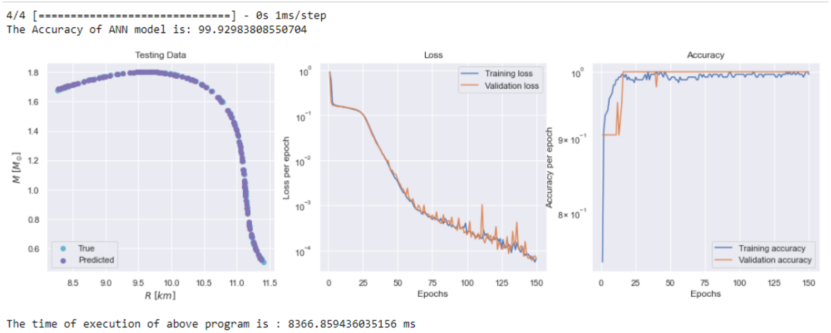


Searching for the best tuning



Searching for the best tuning - Duration

The training execution time is less than 9 sec!



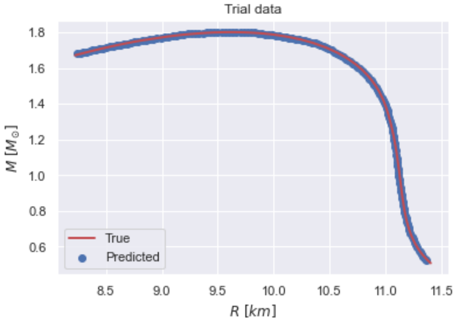
```
4/4 [==============================] - 0s 1ms/step
The Accuracy of ANN model is: 99.92983808550704
```

```
The time of execution of above program is : 8366.859436035156 ms
```

Searching for the best tuning - Duration

The prediction execution time is less than 0.1 sec!

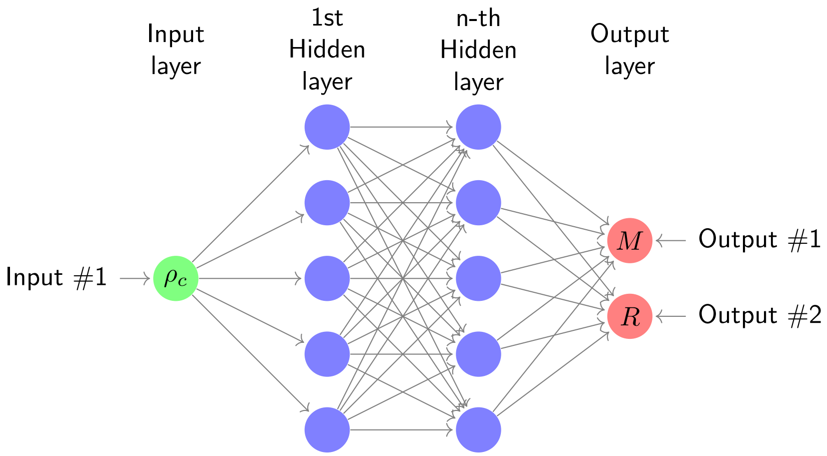
```
5/5 [==============================] - 0s 1ms/step
The time of execution of above program is : 52.233219146728516 ms

Text(0.5, 1.0, 'Trial data')
```



The next step

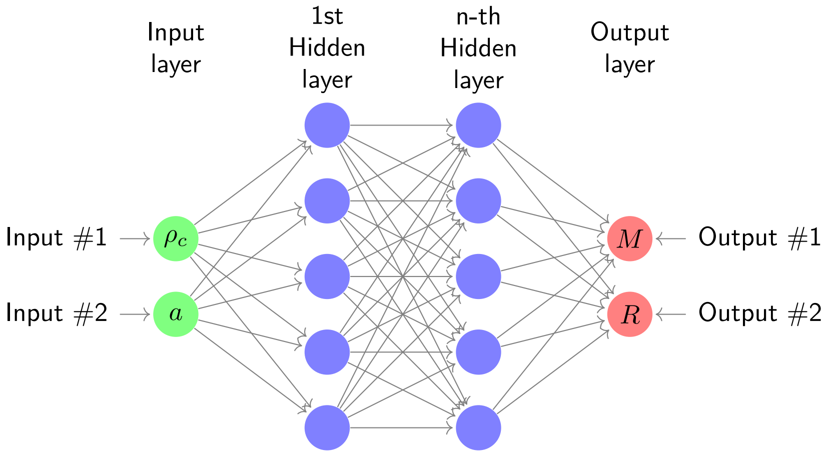
General Relativity

Up to this point we had one input and two outputs, and one network for every EOS. This corresponds to a **curve**.



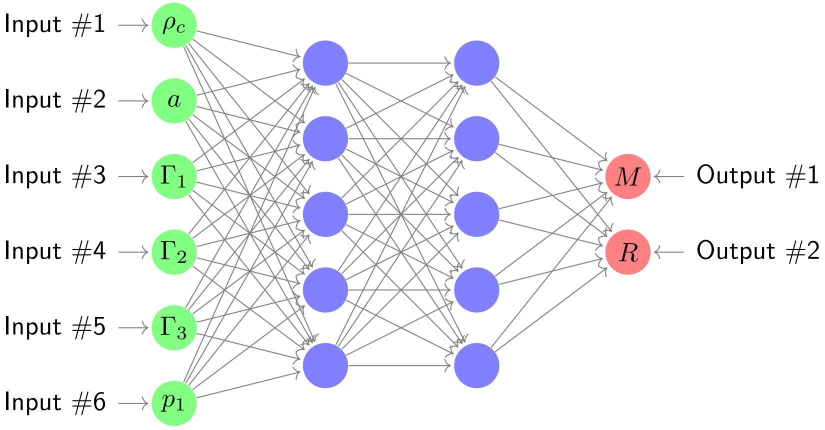
Einstein-Gauss-Bonnet Gravity

The first target is to add one more input parameter, which parameterizes how "far" the modified gravity is from GR (one network for every EOS). This corresponds to a **surface**.



Einstein-Gauss-Bonnet Gravity

The second target is to add 4 more input parameters, except a, which parameterize the EOS. This corresponds to a **surface**?



Thank you!

Sources

- Prof. Nikolaos Stergioulas' presentation at 11th Aegean Summer School, Syros - Machine Learning Applications in Gravitational Wave Astronomy
- Vasileios Skliris PhD - Machine Learning To Extract Gravitational Wave Transients - Cardiff
- https://towardsdatascience.com/
- https://thinkingneuron.com/
- https://texample.net/
- https://www.tensorflow.org/
- https://github.com/niksterg/pyTOVtab