Research of Neural Networks: Theory and Implementation

Chuan Lin CLIN146@JH.EDU

Editor:

Abstract

We first introduce the theoretical basis of neural networks and deep learning, including neurons and neural networks, forward propagation and back propagation. Secondly, we also introduce two neural network models, feedforward neural network model and autoencoder model, including encoder, decoder and predictor. Finally, we also implement the neural network model in code, and verify the superiority of the neural network model over the null model and the linear model on six classical data sets.

Keywords: Neural Network, Feed Forward Network, Autoencoder

1 Introduction

Deep learning has received more and more attention in recent years. It is based on a mathematical concept of neurons, which are connected in a specific way to form a variety of network structures, called neural networks; At the same time, by using forward propagation technique and back propagation technique, we can realize value transfer and gradient transfer between neurons respectively, and finally use neural networks to predict or train neural networks.

There are several reasons for the rise and success of neural networks:

Firstly, the representation bias of neural networks is small. The existing traditional machine learning models make a variety of assumptions about the data, also known as representation biases. For example, the support vector machine model assumes that two classes of data are linearly separable in the feature space, and the decision tree model assumes that the boundaries between different classes of data are parallel. However, given a particular data set, it isn't easy to verify which assumptions the data fits, or to find the corresponding model whose assumptions fit the data. All these problems make it difficult to apply traditional machine learning models in reality. Neural network is a model to solve these problems. The available research shows that a three-layer feedforward neural network with enough neurons can fit any function with arbitrary precision (Universal Representation Theorem). Besides, the relationship between the features and the label of each sample in supervised learning is functional, as long as all data are encoded as numbers. In this case, the neural network model does not have obvious representation bias, which makes its representation ability stronger than other models. In addition, we can further reduce its representation bias by increasing the number of network nodes or the depth of the network. This scalability is also not available in other models.

Secondly, the diversity of neural networks is high. For traditional machine learning models, the representation bias of these models makes them biased towards certain data, making it difficult to develop variants of these models to adapt to different data and tasks.

For example, linear regression models can only deal with data with a linear relationship between label values and feature values; If we introduce higher order terms to achieve nonlinear regression, the problem of dimensional explosion will occur. However, we have a variety of variations on neural network models. For example, we can modify the way neurons are connected or the way data is processed within each neuron to achieve different network characteristics. It is these changes that allow neural networks to cope with a wide variety of data and machine learning tasks. For example, convolutional neural networks can process image classification, recurrent neural networks can be used for text analysis, generative adversarial neural networks can generate data, and so on. This kind of variability is not available in traditional machine learning methods.

Thirdly, the training method of neural network is relatively unified and simple. The training of neural networks is essentially just the optimization of the parameters inside each neuron, and these parameters are not constrained in any way. Therefore, the training of neural networks can be regarded as an unconstrained optimization problem with the loss function as the target and the network parameters as the optimization variables. There are various simple gradient methods such as gradient descent and momentum method. In addition, the emergence of backpropagation algorithms makes it easy to calculate the gradient of the loss function to any parameter, regardless of the structure of the network. Therefore, the training methods of neural networks are relatively uniform and simple. In contrast, the training of traditional machine learning models is more complex; Different models have different training methods. For example, due to the non-negative properties of Lagrange multipliers, the training process of support vector machines is a constrained optimization process, and we need specific optimization methods such as Sequence Minimum Optimization (SMO).

Among many neural networks, two models are particularly classical, namely feedforward neural network and autoencoder model. The former is pioneering to learn the complex relationship between data through deeper networks, which makes the neural network have nonlinear fitting ability for the first time. The latter pioneered the use of neural networks to reduce the dimensionality of data before classifying it.

We assume that the feedforward neural network with two hidden layers has sufficient representation ability and low training difficulty, and carry out code implementation and verify on six classical data sets. At the same time, we assume that the autoencoder is an effective technique, and designs a one hidden layer autoencoder network and two hidden layer prediction networks, so as to carry out code implementation and verify on six classical data sets.

The rest of this paper is organized as follows: In the second section, we introduce the theoretical basis of neural network in detail, including the definition of neuron and network, forward propagation technology and backpropagation technology. In addition, we also introduce two specific neural network models, the feedforward neural network model and the autoencoder model. At the same time, we also introduce the experimental approach in detail. In the third section, we present the results of the experiment. In the fourth section, we analyze and demonstrate the experimental results. In the fifth section, we summarize the neural network model, point out its shortcomings, and make a prospect for the future.

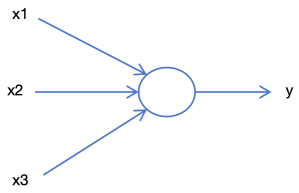

Figure 1: Diagram Representation of a Neuron

2 Algorithm and Experimental Method

2.1 Algorithm

2.1.1 Neuron

As a mathematical concept, a neuron can be regarded as a real-valued multivariate function $f : R^n \to R$. This function outputs a real value as the output of this neuron; This output value may be passed on to various other neurons. At the same time, this function accepts multiple real values as inputs to this neuron; These input values come from other neurons. The diagram representation of neurons can be seen in Figure 1; Where a circle represents a neuron, each input arrow represents a real-valued input, and the output arrow represents a real-valued output. Specifically, each neuron first performs a linear weighted sum of all its input values, and then applies a special class of real-valued unary functions, known as activation functions, to the result of the sum. Refer to the following formula.

$f(x_1, x_2....x_n) = \theta_1 * x_1 + \theta_2 * x_2....\theta_n * x_n + b$

The activation function provides nonlinear fitting ability for neurons and even the whole neural network. If no activation function is used, any neural network model will degenerate into a linear model. For the activation function, we require it to be continuously derivable in order to facilitate the subsequent use of the backpropagation algorithm. Common activation functions are sigmoid function, tanh function, relu function and so on.

In addition, the weights and biases used in the linear weighting process are adjustable and trainable, known as the parameters of this neuron. For the parameters of neurons, we have no constraints, so the parameter optimization problem becomes an unconstrained optimization problem. However, parameters of different neurons can be shared, thus reducing the number of parameters in a network, such as convolutional neural network.

In addition, it must be noted that not all data processing processes within neurons conform to the form of linear weighted sum and activation described above. Take the Max pooling layer the in convolutional neural network as an example. Each neuron directly takes the maximum value of the input values as its output. We're not going to talk about these kinds of neuronal nodes.

In addition, it must be noted that not all neurons receive inputs from other neurons or transmit their outputs to other neurons. Some neurons input directly from the outside world or output directly to the outside world. The input layer neuron takes an outside real value as the input to this neuron, and the input arrow is omitted in the diagram. The output layer neuron outputs a real value to the outside world and omits the output arrow in the diagram.

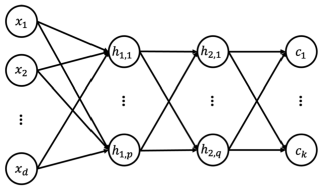

Figure 2: Diagram Representation of 2-hidden-layer FNN

2.1.2 Neural Network

As a mathematical concept, a neural network can be regarded as a multivalued multivariate function $f : R^n \to R^m$. This function accepts the real values from outside as the network input, which are usually the features of a data point. At the same time, this function outputs multiple real values to the outside as the network output, which is usually the prediction for this data point. Specifically, the function represented by a neural network is a composition of the functions represented by the neurons that compose it. This composition is often complex and is determined by the network's structure. To conveniently represent the complex relationships between neurons, we use a directed acyclic graph (DAG) for representation. That is, the diagram representation of a neural network. Each circle represents a neuron, and each directed edge represents the value pass between neurons. For any one neuron and the arrows around it, the local meaning is consistent with the diagram representation of the neuron, as mentioned above. For example, a feedforward neural network with two hidden layers is shown in Figure 2.

Generally speaking, the structure of the network has a great influence on the learning ability of the neural network. The performance of neural networks with different network structures is often very different. For example, the greater the depth of the feedforward neural network, the stronger the learning ability of the network; For another example, convolutional neural networks are better suited to image-input machine learning tasks than feedforward neural networks. All in all, the design of network structure is a complex problem, and more and more research and experiments are being carried out.

2.1.3 Forward Propogation Technique

As a function, given its outside input, we can calculate the output of the network. This process is called the forward propagation algorithm. Since the function represented by the neural network is a composite of the functions represented by each neuron, we can compute the function of the entire neural network by calculating the functions of each neuron. However, for any neuron, before we calculate its function, we need to ensure that its input values have been calculated. Considering that the diagram representation of a neural network is a DAG, we can compute the function represented by each node in turn according to the topological ordering of each node. The pseudocode is shown in $ForwardPropogation$ and $Predict$.

Algorithm 1 ForwardPropogation

Ensure: output $y_1, y_2, ...y_m$
Require: network $network$, input $x_1, x_2, ..x_n$
  $inputNode1, inputNode2...inputNodeN \leftarrow findInputNodes(network.nodes)$
  $inputNode1.output, inputNode2.output.... \leftarrow x1, x2, ...$
  for $node \leftarrow topologicalSort(network.nodes)$ **do**
    $node.input \leftarrow emptyset$
    for $parent \leftarrow node.parents$ **do**
      append $parent.output$ to $node.input$
    end for
    $node.output \leftarrow node.func(node.input)$
  end for
  $outputNode1, outputNode2...outputNodeM \leftarrow findOutputNodes(network.nodes)$
  $y1, y2, ... \leftarrow outputNode1.output, outputNode2.output, ...$
  return $y1, y2...$

2.1.4 Backward Propagation Technique

To measure the performance of current neural networks, we use a real-valued multivariate loss function $L : R^m \rightarrow R$. Specifically, given a sample, the input to the loss function is the output values of the neural network on this sample and the label value of this sample, and the output is the loss value on this sample. The loss value on the whole data set is the sum of the loss value on the individual samples. In general, the smaller the loss, the better. For the regression problem, the output of the neural network is the predicted label value, and the sum of squared error (SSE) loss is generally adopted. For classification problems, the output of the neural network is the probability distribution of the samples belonging to various classes, and we use the cross entropy loss.

To optimize the parameters in the network, we use various gradient methods such as gradient descent and momentum method, so it is necessary to calculate the partial derivative of the loss value relative to each parameter, that is, to calculate the gradient of the parameters. This process is called the backward propagation algorithm.

We first discuss how to calculate the partial derivative of the loss value to each parameter. Given a real-valued parameter θ, we assume that the output/activation value of the neuron in which it is located is value δ and the loss value of the neural network is value l. Considering that the only value directly affected by the θ is the δ, according to the chain rule in calculus, we have the following formula:

$\frac{dl}{d\theta} = \frac{dl}{d\delta} * \frac{d\delta}{d\theta}$

Therefore, the partial derivative of the l with respect to θ can be decomposed into the product of the partial derivative of l to the δ and the partial derivative of δ to the θ. The partial derivative of δ to the θ is very easy to find. Therefore, as long as we can calculate the partial derivative of the loss value with respect to the activation value of each neuron, we can find the partial derivative of the loss value with respect to each parameter.

Next, we discuss how to calculate the partial derivative of the loss value to the activation value of each neuron. Given a neuron, we assume that its activation value is the real value δ_0,

and the output value of its k children is $\delta_1, \delta_2 ... \delta_k$ and the loss value of the neural network is value l. According to the chain rule in multivariable calculus, we have the following formula:

$\frac{dl}{d\delta_0} = \frac{d\delta_1}{d\delta_0} * \frac{dl}{d\delta_1} + \frac{d\delta_2}{d\delta_0} * \frac{dl}{d\delta_2} .... \frac{d\delta_k}{d\delta_0} * \frac{dl}{d\delta_k}$

Therefore, we can find that the partial derivative $\frac{dl}{d\delta_0}$ can be decomposed into the sum of multiple products; Each product term corresponds to a child node i, which is the product of partial derivative $\frac{d\delta_i}{d\delta_0}$ and partial derivative $\frac{dl}{d\delta_i}$. The partial derivative $\frac{d\delta_i}{d\delta_0}$ is very easy to find. Therefore, as long as we can calculate all $\frac{dl}{d\delta_i}$, we can find $\frac{dl}{d\delta_0}$. Using this feature, we can calculate the partial derivative to the activation value of each node in turn according to the reversed topological order, so as to calculate the partial derivative to each parameter and realize the backward propagation algorithm. The pseudo-code is shown in *BackwardPropogation*.

Algorithm 2 BackwardPropogation

Ensure: None
Require: network *network*
Require: the partial derivative of l to $y1, y2, ..y_m$: $d1, d2, ...d_m$
  $outputNode1, outputNode2 ... outputNodeM \leftarrow findOutputNodes(network.nodes)$
  $outputNode1.grad, outputNode2.grad ... \leftarrow d1, d2, ...$
  for $node \leftarrow inverse(topologicalSort(network.nodes))$ **do**
    $node.grad \leftarrow 0$
    for $child \leftarrow node.children$ **do**
      $d \leftarrow$ the derivative of $child.output$ to $node.output$
      $node.grad += d * child.grad$
    end for
    calculate the derivative to $node.parameters$ based on $node.grad$
  end for
  return None

2.1.5 Feedforward neural network

Feedforward Neural Network (FNN) is a neural network with neurons connected by layers. The core idea is to learn more complex data relationships through deeper network layers. Specifically, we divide all neurons into different sets, and each set is called a layer. In addition, there is a certain order between the different layers. The neurons of each layer only accept the value from the neurons of the previous layer and only pass the value to the neurons of the next layer. The neurons are activated layer by layer. Although the structure of this model is simple, it is the first model with non-linear fitting ability. This model inspires the development of all subsequent models. The diagram representation of the network shown in Figure 2:

2.1.6 Autoencoder

Autoencoder can be viewed as two feedforward neural networks that share some layers. The core idea is to reduce the dimension of the sample first and then classify it. The first FNN to be trained is called an autoencoder network, whose goal is to compress the input data

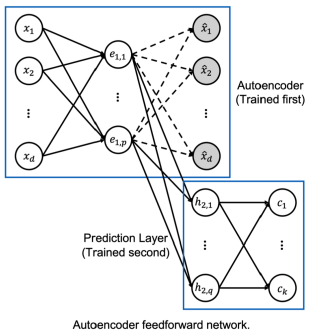

Figure 3: Diagram Representation of 2-hidden-layer Autoencoder

into some low-dimensional representation called a latent vector, and then to recover the original input data from the latent vector. The first k layers of this network are called the encoding layers, and all subsequent layers are called the decoding layers (k is the number of layers required to achieve data compression). The second feedforward neural network to be trained is called the predictor network. The goal is to first compress the input data into a latent vector, and then use it for classification. The first k layers of this network are also the encoding layers of the autoencoder network, and all subsequent layers are called prediction layers. Compared with the traditional neural network model, the Autoencoder first reduces the dimension of the data and then classifies it, which greatly reduces the amount of data accepted by the classification layer network and thus reduces the training difficulty. The diagram representation of the network is shown in Figure 3:

2.2 Experimental Approach and Underlying Assumption

2.2.1 Underlying Assumptions

In this experiment, the model hypothesis of linear model is firstly made: for regression task, the label value data and feature values show a linear relationship; For classification tasks, any two categories of data are linearly separable. Secondly, the model hypothesis of the double-hidden-layer FNN is made: two hidden layers are sufficient to capture the complex relationship between the data. Thirdly, the model assumption of single-hidden-layer Autoencoder network is made: one hidden layer can effectively reduce the dimensionality of the original data.

At the same time, this experiment also made the relevant experimental hypothesis. First of all, this experiment assumes that 5×2 cross-validation can simulate the scenario when the model predicts data points that have never occurred, and the results can well reflect the generalization error of the model. Secondly, this experiment assumes that the following grid search method can well find a good number of hidden layer nodes, that is, the corresponding model is close to the optimal model with the same number of network layers.

2.2.2 Experimental Approach

Firstly, we introduce the data preprocessing method. Since both linear model and neural network model require the input to be numerical. Therefore, we perform the following

operations on each data set: firstly, encode each categorical feature into a numerical feature (ordinal feature is encoded as an integer; the nominal feature is encoded as one-hot vector); secondly, standardize each numerical feature.

Secondly, we introduce a grid search method for the number of hidden layer nodes. Under any data set, for any hidden layer of any feedforward neural network, we traverse the three values, 5, 10,20,30. Taking the double-hidden-layer FNN model in this experiment as an example, we need to traverse nine combinations, (5,5), (5,10)....(30,30). It should be noted that for the Autoencoder model, the number of hidden layer nodes in the autoencoder network must be strictly less than the number of input nodes. In this case, we might exclude some values in 10,20,30 accordingly. After we find the optimal number of hidden layer nodes for each model in each dataset, the following experiments are carried out.

Thirdly, we briefly introduce the training methods of the model. For linear model, FNN model, and autoencoder model, we use the mini-batch gradient descent method for model training, which stops when the epoch number reaches the preset value. The batch size is the preset value of 50, the epoch number is 1000, and the learning rate is 0.001. According to the results of some simple prior experiments, such hyperparameter values of the optimizer can make the three models achieve a good balance between underfitting and overfitting. In addition, in the regression task, the loss function is sum of squared error (SSE) loss function and the evaluation metric is mean squared error (MSE) loss function. In the classification task, the loss function is softmax cross entropy loss function, and the evaluation metric is zero-one loss function. We chose not to add the softmax layer to the network and use the cross entropy loss function because the last layer of the neural network model generally defaults to the linear layer rather than the activation layer. In addition, using the softmax cross entropy loss function can reduce the learning difficulty of the network and improve the training efficiency.

Lastly, the following is the experimental process. Our experiments compare the null model, linear model, FNN model, and Autoencoder model on six classical datasets. We want to verify the superiority of linear models over Null models and the superiority of neural network models over linear models. The experimental methods of this experiment are as follows: Given any data set, we first perform the corresponding preprocessing operations. We then use the above four models for 5*2 cross-validation. This is in fact five independent 2-fold cross-validations. In each experiment of 2-fold cross-validation (2-fold cross-validation has two experiments), half of the data is used to train the four models, and the other half is used to test the four models. Finally, 10 test metrics are obtained for each model, and the corresponding average metric for each model is calculated. Finally, we can compare the performance of the four models on this dataset. The above process is repeated for each of the six classical data sets.

3 Results

The result is shown in the following Table 1.

Besides, here are the number of hidden-layer nodes for 2-hidden-layer FNN model and 2-hidden-layer Autoencoder model on each dataset: in Abalone dataset, the number of hidden-layers nodes for FNN is (30,20) while that for Autoencoder model is (5,10); in Cancer dataset, the number of hidden-layers nodes for FNN is (20,10) while that for Autoencoder

Table 1: Comparing Null, Linear Model, FNN and Autoencoder on Six Datasets

		experiment 0		experiment 1		experiment 2		experiment 3		experiment 4		Mean
		fold 0	fold 1	fold 0	fold 1	fold 0	fold 1	fold 0	fold 1	fold 0	fold 1	
Abalone Dataset	Null	9.507	9.636	9.507	9.636	9.507	9.636	9.507	9.636	9.507	9.636	9.571
	Lin	4.695	4.640	4.751	4.772	4.855	4.537	4.595	4.775	5.113	4.628	4.736
	FNN	4.339	4.488	4.499	4.571	4.629	4.484	4.571	4.738	4.622	4.522	4.546
	Auto	4.209	3.966	4.031	4.113	4.153	4.077	4.091	4.102	4.187	4.053	4.098
Cancer Dataset	Null	83	83	83	83	83	83	83	83	83	83	83
	Lin	11	10	11	13	11	12	16	21	10	10	12.5
	FNN	13	19	15	14	22	13	19	15	19	15	16.4
	Auto	13	20	17	14	26	14	22	11	18	16	17.1
Car Dataset	Null	255	257	255	257	255	257	255	257	255	257	256
	Lin	154	128	154	145	149	141	130	161	149	151	146.2
	FNN	8	27	15	21	15	14	30	14	21	35	20.0
	Auto	84	39	48	45	39	51	75	58	59	49	54.7
Forestfire Dataset	Null	0.000	0.030	0.000	0.030	0.000	0.030	0.000	0.030	0.000	0.030	0.015
	Lin	0.010	0.039	0.023	0.052	0.030	0.042	0.023	0.042	0.046	0.044	0.035
	FNN	0.067	0.060	0.038	0.039	0.082	0.078	0.041	0.063	0.010	0.047	0.053
	Auto	0.000	0.030	0.000	0.030	0.003	0.029	0.000	0.030	0.001	0.029	0.015
House Dataset	Null	83	83	83	83	83	83	83	83	83	83	83
	Lin	14	8	6	14	18	10	9	16	11	6	11.2
	FNN	12	10	12	13	9	11	10	9	11	17	11.4
	Auto	14	8	7	11	14	9	9	13	9	15	10.9
Hardware Dataset	Null	208	209	208	209	208	209	208	209	208	209	208
	Lin	1510	1430	1328	1327	1435	1337	1367	1182	1388	1212	1352
	FNN	1220	1430	1416	1210	1324	1333	1343	1231	1561	1289	1336
	Auto	1346	1287	1339	1290	1337	1291	1350	1293	1332	1299	1316

model is (5,10); in Car dataset, the number of hidden-layers nodes for FNN is (20,10) while that for Autoencoder model is (5,10); in Forestfire dataset, the number of hidden-layers nodes for FNN is (30,20) while that for Autoencoder model is (5,10); in House dataset, the number of hidden-layers nodes for FNN is (20,10) while that for Autoencoder model is (10,20); in Hardware dataset, the number of hidden-layers nodes for FNN is (30,20) while that for Autoencoder model is (20,30).

4 Discussion

It can be seen from the experimental results that the linear model has obvious advantages over the null model. In both classification and regression tasks, the linear model has a significant improvement over the null model. The reasons are as follows: Even if the relationship between label values and feature values is not completely linear, this complex relationship can also reflect a linear relationship to some extent; The linear model can capture this linear relationship well, which is superior to the null model which does not capture any relationship.

It can also be seen from the experimental results that the autoencoder model is often better than the feedforward neural network model. In the six data sets tested, except for the Cancer data set, Autoencoder's experimental results were the best of the four models. We speculate that the reasons are as follows: The Autoencoder model learns the latent vector of the input sample in the low-dimensional space and makes predictions on it. It is precisely this learning of the latent vector that enables the Autoencoder model to better grasp the key features of the data while avoiding overfitting caused by the training on a small dataset.

It can also be seen from the experimental results that the neural network model does not necessarily have obvious advantages over the linear model. In both classification and regression tasks, the FNN model has very little improvement compared with the linear model (although Autoencoder is indeed better than the linear model). We speculate that the reasons are as follows: Because the number of our training samples is small, the relationship between label values and eigenvalues is basically linear; In this case, even though the FNN model has a certain nonlinear fitting ability compared with the linear model, due to the absence of complex data relationships in the training samples, the FNN model can only capture linear relationships and thus its performance is roughly the same as that of the linear model.

5 Conclusion

From the relevant theory and practice of neural network, we can see that the neural network model is indeed an excellent model, which has the advantages of strong representation ability, good diversity, simple training methods, and so on. Finally, on the six data sets, two neural network models (the FNN model and the Autoencoder model) outperform the linear model and the null model. However, we must also see that there are still many problems and future prospects for neural networks.

First, the interpretability of neural networks needs to be improved. We need a more complete theory to guide us to design the structure of network. These problems include: how to connect the individual neurons, how to choose the operation in the neuron and so on. At present, our theory is not sufficient either qualitatively or quantitatively. For example, since the universal representation theorem states that three-layer feedforward neural networks can fit any function with arbitrary precision, why are feedforward neural network models inferior to other special models for many problems and require us to develop new network structures? At present, the research on the explainability and theoretical basis of neural networks is constantly being carried out.

In addition, the training of neural networks is difficult, and the risk of underfitting is high. For network models with deeper depth and more nodes, although their representation ability is stronger in theory, their actual result may not be as good as some simpler models in practical application. The reason is that it is difficult to train these models with strong expression ability. When the amount of training data provided is insufficient, there may be a case as follows: Our model parameters have multiple local minima, which leads to the failure of the model to reach the global minima with the best generalization error, resulting in the model underfitting. The relevant effort on reducing training difficulty is still carried on.