# Research on the Relationship Between Face and Emotion: Bidirectional Neural Networks

Yu Qiao

School of Computer Science and Engineering
Australia National University
Email: u6706067@anu.edu.au

**Abstract.** Among the current existing research on faces and emotions, the data recorded in the real environment is vital for the results. I did experiments with different types of neural networks on a new static facial expression dataset Static Facial Expressions in the Wild. I compared the results of a simple neural networks and bidirectional neural networks. The performance of bidirectional neural networks is better than that of common neural networks.

**Keywords:** SFEW database, Neural Networks, Bidirectional Neural Networks

## 1 Introduction

Neural network (NN), also known as artificial neural network (ANN). The artificial neural network basically consists of the following components: Input layer: receive and pass data; Hidden layer; Output layer; Weights between layers. Each hidden layer has an activation function. In this simple neural network, I will use the Sigmoid activation function. As for bidirectional neural network, I am going to use Bi-LSTM.

## 2 Method

### 2.1 Description of dataset

In the paper, I solve a classification problem on a given dataset, which has 10 attributes(inputs) and 1 label(output). The numbers (from 1 to 7) in the column of label means angry, disgust, fear, happy, neutral, sad and surprise. 5 of the attributes are Principal Components of Local Phase Quantization (LPQ) features, and the other

are Principal Components of Pyramid of Histogram of Gradients (PHOG) features. In the first column, they are the source of the data. We have 675 records.

The Figure 2 shows a part of the raw data.

| | A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | label | First 5 Principal Components of Local Phase | | | | | First 5 Principal Components of Pyramid of Histogram o | | | | |
| 2 | Airheads_0( | 1 | -0.0008172 | 0.0034698 | -0.0075171 | -0.010912 | -0.0054296 | 0.0095511 | 0.0067755 | 0.0035193 | -0.0009996 | 0.0043082 |
| 3 | AlexEmma_0( | 1 | -0.0019823 | -0.000265 | 0.00016096 | -0.010747 | 0.00082863 | -0.0004955 | -0.0047233 | -0.0053007 | 2.90E-05 | 0.0024337 |
| 4 | AlexEmma_0( | 1 | -0.0030116 | 0.013759 | -0.0087055 | 0.0019429 | 0.0061133 | 0.01235 | -0.0044805 | -6.89E-05 | 0.0022386 | 0.0063954 |
| 5 | AlexEmma_0( | 1 | 0.00022408 | 0.0048942 | -0.0054845 | -0.0044262 | -0.0098829 | -0.0012781 | -0.0030751 | -0.0022199 | 0.00015534 | 0.00037549 |
| 6 | AlexEmma_0( | 1 | -0.0004422 | -0.0023756 | -0.0024227 | -0.010372 | 0.0021083 | 0.00092784 | 0.00082337 | -0.0031689 | -0.0039125 | 0.0051525 |
| 7 | AlexEmma_0( | 1 | -0.0011055 | 0.0021197 | -0.0032849 | -0.0064262 | -0.011666 | 0.0025168 | -0.0088354 | -0.0023136 | -0.0041014 | 0.001039 |
| 8 | AlexEmma_0( | 1 | -0.0012627 | -0.022519 | -0.0037013 | -0.0056274 | -0.009489 | -0.013399 | 0.0034253 | -0.0070648 | 0.0036605 | 0.0025155 |
| 9 | AlexEmma_0( | 1 | -0.0011693 | -0.022367 | -0.014832 | -0.0011779 | -0.011345 | -0.016939 | 0.0048949 | -0.012611 | 0.0053144 | 0.0066979 |
| 10 | AlexEmma_0( | 1 | 0.0020192 | -0.0021889 | 0.0016372 | -0.0078953 | -0.0076976 | 0.014068 | 0.0079642 | 0.0068933 | -0.001063 | 0.0011439 |
| 11 | AlexEmma_0( | 1 | -0.0045181 | -0.013468 | -0.0054261 | -0.0030916 | -0.0023567 | 0.010618 | 0.0068644 | -0.0049586 | 0.0011314 | 0.0027233 |
| 12 | AlexEmma_0 | 1 | -0.0026443 | -0.0080293 | -0.014288 | -0.0058553 | -0.0030814 | -0.01372 | 0.0063709 | -0.0033444 | 0.00086169 | 0.0098901 |

**Fig. 1.** a part of the raw data

## 2.2 Method

The steps of building a neural network to solve a classification problem are shown as follows:

### 2.2.1 Load and process the dataset

The first step is loading the data. Read data from the 'SFEW.xlsx' and delete the first row and column. The Figure 2 shows a part of the data after this step. I adjust the sequence of columns in the original dataset.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | -0.00081724 | 0.0034698 | -0.0075171 | -0.010912 | -0.0054296 |
| 1 | -0.0019823 | -0.00026502 | 0.00016096 | -0.010747 | 0.00082863 |
| 2 | -0.0030116 | 0.013759 | -0.0087055 | 0.0019429 | 0.0061133 |
| 3 | 0.00022408 | 0.0048942 | -0.0054845 | -0.0044262 | -0.0098829 |
| 4 | -0.00044217 | -0.0023756 | -0.0024227 | -0.010372 | 0.0021083 |

**Fig. 2.** The result of the first step

Then, I define a customize torch dataset on it and normalize the input data. The next step is splitting data into training data and testing data. Here, I set training set' size as 80% of the whole dataset. Then, I define a data loader for this neural network. The Figure 3 shows a part of the training set's data after this step.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | \ |
|---|---|---|---|---|---|---|---|---|
| 21 | -0.073688 | 0.766312 | -0.365839 | -0.354157 | -0.125458 | 0.748789 | 0.110683 | |
| 23 | -0.036165 | -0.703900 | 0.797962 | -0.030212 | -0.060189 | -0.379244 | 0.916599 | |
| 27 | -0.044448 | -0.530453 | 0.624059 | -0.570240 | 0.217557 | 0.720541 | 0.670717 | |
| 29 | -0.088847 | -0.692923 | -0.585665 | 0.357857 | 0.484181 | -1.368021 | 1.244642 | |
| 32 | -0.132233 | -0.861214 | -1.098239 | 0.033736 | 0.743431 | -0.948875 | -0.464413 | |
| 42 | -0.053483 | -0.697878 | 0.431025 | -1.397657 | -0.726993 | -0.299768 | | |

**Fig. 3.** The result of the second step

### 2.3 Build the neural network

A simple and basic neural network uses sigmoid as its activation function, Figure 4 shows this function.

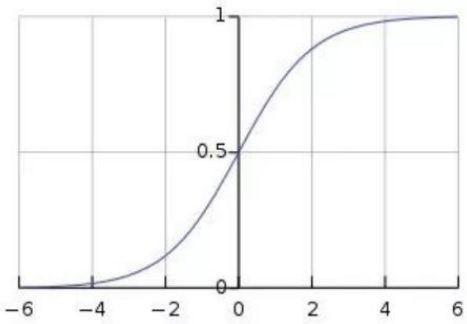



**Fig. 4.** The sigmoid function

The hyper parameters are set as follows:
input_size = 10
hidden_size = 50
num_classes = 7
num_epochs = 500
batch_size = 10
learning_rate = 0.1

Then, set the network’s initial parameters, forward functions and backward functions. Calculate its loss in each batch of training and update its weight until it converges. The Figure 5 shows the structure of the simple neural network.

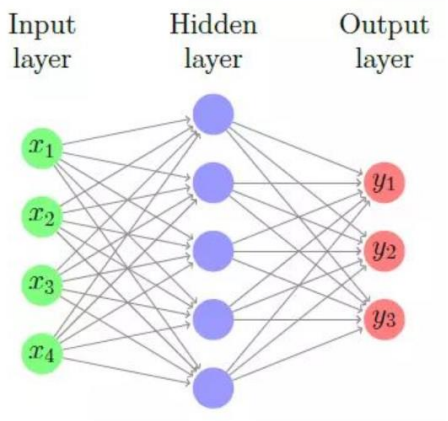



**Fig. 5.** structure of the simple neural network.

The Figure 6 shows the structure of the Bi-directional neural network.

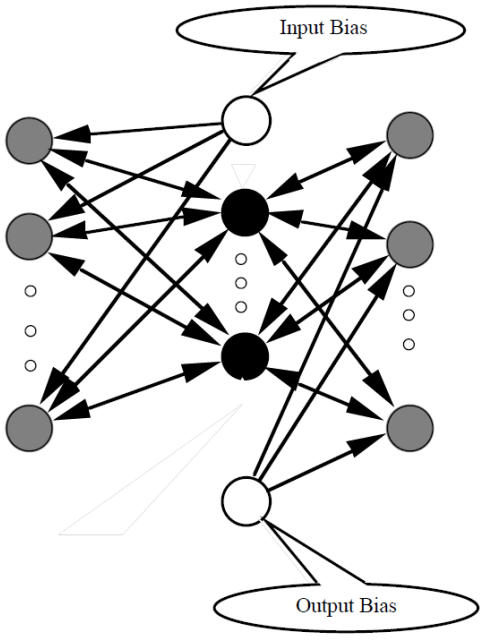



**Fig. 6.** the structure of the Bi-directional neural network

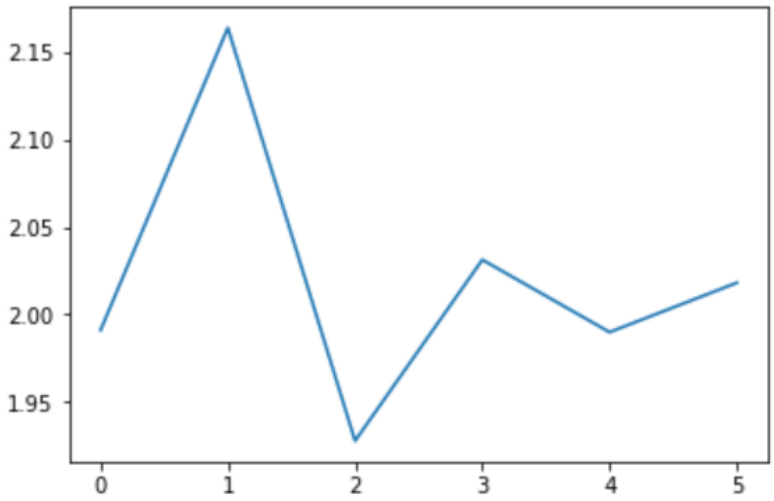
I gave a part of my code as follows:

```
)]: # #data load dataset and delete the first row and col
import xlrd
import numpy as np
import scipy
filename='SFEW.xlsx'
dataset = [ ]
workbook = xlrd.open_workbook(filename)
table=workbook.sheets()[0]
for row in range(table.nrows):
    dataset.append(table.row_values(row))

data=np.array(dataset)
data=np.delete(data,0,axis=0)
data=np.delete(data,0,axis=1)

data=pd.DataFrame(data)
cols=list(data)

cols.insert(10,cols.pop(cols.index(0)))
data=data.loc[:,cols]
print(data)
```

The rest you can get from the zip file I uploaded.

## 3 Results and Discussion

The loss of a simple neural network is shown as Figure 5



The testing accuracy is around 20.98%. The result gave in the paper I cited is 74%. The result I got from the simple neural network was quite bad. That indicates Bidirectional neural networks' performance is better than simple neural networks.

## 4 Conclusion and Future work

I have shown how a basic ANN trains the model and compared the results with that of the bidirectional neural networks. Bidirectional neural networks' performance is better than simple neural networks. I believe that the Bidirectional neural network has a wider application. In the future, I will adjust the neurons of the hidden layer. I want to explore whether it will improve the accuracy of the bidirectional network.

## Reference

Dhall, A., Goecke, R., Lucey, S., & Gedeon, T. (2011, November). Static facial expressions in tough conditions: Data, evaluation protocol and benchmark. In 1st IEEE International Workshop on Benchmarking Facial Image Analysis Technologies BeFIT, ICCV2011.

Nejad, A. F., & Gedeon, T. D. (1995, November). Bidirectional neural networks and class prototypes. In *Neural Networks, 1995. Proceedings., IEEE International Conference on* (Vol. 3, pp. 1322-1327). IEEE.