Using Neural Networks to Differentiate Genuine from Posed Anger

Nutthadech Banditakkarakul
Research School of Computer Science
Australian National University
Acton ACT 2601 Australia
u4995458@anu.edu.au

Abstract. This paper details the process of training neural networks for a classification task: differentiating genuine from posed anger using pupillary data. The first part focuses on using summary statistics data with feedforward neural networks and a feature selection technique to make the classifications. The second moves on to using times series data with gated recurrent units to further enhance the classification performance. Our experiments demonstrate the efficacy of both feedforward neural networks and gated recurrent units in extracting useful patterns from pupillary data. Our results supports the claim that pupillary response pattern can be used to reliably differentiate genuine from posed anger.

Keywords: Feedforward Neural Network, Feature Importance, Gated Recurrent Units, Anger Prediction.

1 Introduction

In 2017, [1] documented that humans may actually be more capable than they realize at determining other's anger veracity. In their study, they tracked the pupillary responses of 22 participants who are shown 10 videos of people genuinely angry and 10 videos of people just acting angry. They use the statistics of these pupillary response data to train an ensemble of machine learning classifiers and achieved a 95% test accuracy in differentiating genuine from posed anger. Concurrently, these same participants were asked to record their own classification of the same videos (but without access to their own pupillary or other physiological response data). It turned out that the participants' conscious effort to differentiate genuine from posed anger only achieved a mere 60% accuracy. Their study exemplified the ability of machine learning to unearth valuable information from data that humans are capable of generating but not able to utilize. We feel that the concept of combining human (subconscious) intelligence with machine intelligence is very intriguing.

In this paper, we aim to replicate the findings of [1] regarding the usefulness of pupillary response patterns to differentiate genuine from posed anger. However, we will focus more on using neural networks and conduct experiments using both the summary statistics and the actual time series of the pupillary response. For the summary statistics dataset, we use feedforward neural networks, combined with a feature selection technique proposed by [2] to make the classifications. For the time series dataset, we utilize a type of recurrent neural network called gated recurrent units to further improve the classification performance. We describe our datasets in Section 2 and explain our methodologies in Section 3. In Section 4, we details all techniques used in achieving our final classification results. Finally, we analyze and compare our findings with that of [1] in Section 5.

2 Dataset

Summary Statistics. This dataset is slightly smaller than that used by [1]. It consists of 400 samples collected from 20 participants. There are six features and one target in our dataset. The features are time domain statistics of participant's pupillary responses (particularly pupil sizes) while they watched the videos: (i) mean, (ii) standard deviation, (iii) 1st difference, (iv) 2nd difference, (v) 1st principal component and (vi) 2nd principal component. These features all take numerical values between 0 and 1. The target consists of two classes: genuine and posed, which are perfectly balanced (200 samples of each). The genuine anger videos are clips of live news reporting or documentaries, whereas the acted anger videos are cuts from movies that are not famous [1].

Time Series. This dataset is the raw data that the first dataset used to calculate the time domain statistics of participant's pupillary responses. In other words, it is the actual time series data. For each video that each participant watched, both left and right pupil diameters were recorded at every 1/60th second. The longest series is 186 time-steps, whereas the shortest is 60. That is, the length of these series vary from one to about three seconds. This dataset contains only 390 samples, with 196 samples belonging to genuine class and 194 belonging to posed class.

3 Methods

3.1 Training Process

The dataset is first shuffled to ensure that the models are exposed to a good mix of samples; this has the effect of maximizing the information content during training [3]. For summary statistics dataset, the data is split using 5-fold cross validation. K-fold cross validation is used instead of a simple train-test split because our dataset is small. K-fold cross validation does not waste training data and allows for a more reliable estimate of the model's generalization ability [4]. For time series dataset, we resort to simple train-test split due to time constraints. The split ratio is the same as that of summary statistics dataset: 20% in test set.

Cross entropy loss function is used when optimizing our neural networks. This is a suitable choice given that our model predicts whether a person in the video is genuinely angry or not, outputting a probability of a sample belonging to each class. Cross entropy loss measures the differences between two probability distributions [5]. The loss increases as the predicted probability diverges from the actual class and decreases to zero when the model makes a perfect prediction.

Pack_padded_sequence function offered by Pytorch is used when training our gated recurrent units on time series dataset. This function allows training to be done in batch while also ensuring that the zeros from zero padding are not used by the model [6]. This is a subtle but crucial step to prevent data leakage: without pack_padded_sequence, the model can simply use the number of zeros at the end of each time series as video ID (because each video has a different length) and make predictions accordingly.

3.2 Performance Evaluation

Model performances are evaluated using the accuracy score calculated from the test set. For K-fold cross validation, the accuracy is simply averaged. The accuracy score measures the ratio of correct predictions to total predictions. This is an appropriate evaluation metric for our classification task, where the target consists of two balanced classes. Our choice of model evaluation metric also aligns with that used by [1].

To increase the reliability of performance evaluation, performance should be averaged across several runs of experiments with different weight initializations. However, our model performance are evaluated using just one run due to time constraints.

3.3 Benchmark Model

A benchmark model is set so that we can compare subsequent results with it. We start with training a feedforward neural network on the summary statistics data. Using network topology and hyper-parameter values below, we achieve a mean train and test accuracies of 79.9% and 69.8%, respectively.

Table 1. Network topology and hyperparameter values of benchmark model

Input Neurons	Hidden Neurons	Output Neurons	Loss Function	Activation Function	Optimizer	Learning Rate
6	12	2	Cross Entropy[1]	Sigmoid	Adam	0.01

At this stage, the network topology and hyperparameters choices are simply made based on initial guesses. In the following sections, we conduct various experiments to enhance the classification performance of our model. Based on quantifiable evidence, we make informed decisions to alter (or keep) these initial data and model choices.

4 Experiments

4.1 Feedforward neural network with summary statistics dataset

Preprocessing Data. Upon a more careful observation of the data, we found that many features are not normally distributed and that their magnitudes are vastly different. To fix these problems, we applied square root transformation followed by normalization on the data. The former helps reduce the skewness of the data distribution, whereas the

[1] It should be noted that, in Pytorch, Softmax function is automatically applied to the output neurons when cross entropy loss function is used [6].

latter ensure similar magnitudes between the data. This leads to a more symmetric loss surface which promotes faster learning and reduces the chance of the optimization algorithm getting stuck in a local minima [7]. Without other network topology or hyperparameter adjustment, these data transformation and scaling methods already improve the model performance materially. The mean train and test accuracies jump to 97.3% and 82.3%, respectively.

Changing Neural Network Topology. A high mean train accuracy of 97.3% suggests that 12 hidden neurons may already be sufficient for the model to discover useful patterns. Therefore, we tried reducing the number of hidden neurons. Using 11 hidden neurons, mean test accuracy improves while mean train accuracy drops. However, we ultimately care more about our model performance on unseen data. So, using 11 hidden neurons makes a sensible choice. The mean test accuracy improves slightly – now at 83.2%.

Tuning Hyperparameters. We tried experimenting several hyperparameter values but it turns out changing the activation function is the only thing the helps. Using Tanh instead of Sigmoid, mean test accuracy rises to 85%. This is not unexpected as Tanh has got a steeper derivative and is symmetric around the origin, making the learning process more efficient for the neural network [3].

Dropping Insignificant Features. We utilize a technique called magnitude measure Q, proposed by [2] to rank the importance of each feature. This Q measure takes weight matrices from the trained neural network as inputs and produces a numerical score in the range of 0 and 1 for each feature[2]. Keeping everything else constant, we drop the two least important features (lowest Q scores). This leads to a final boost in the model performance with mean test accuracy now at 88.5%. Q score also reveals that, by far, '1st Principal Component' is the most useful feature for our classification task, whereas 'Difference 2' is the least useful.

The final feedforward model architecture and the features used are shown in Table 2 and 3. An increase in classification performance after each step in the model development process is also summarized in Table 4.

Table 2. Our final feedforward neural network model

Input Neurons	Hidden Neurons	Output Neurons	Loss Function	Activation Function	Optimizer	Learning Rate
4	11	2	Cross Entropy	Tanh	Adam	0.01

Table 3. Final features used and their Q scores

Feature (Most to Least Important)	Mean Q	Used in Final Feedforward Model
1st Principal Component	0.4356	Yes
2nd Principal Component	0.1589	Yes
Standard Deviation	0.1148	Yes
Difference1	0.1140	Yes
Mean	0.0958	No
Difference2	0.0809	No

Table 4. Improvement in classification accuracy after each step in the model development process

	Benchmark Model	After Preprocessing Data	After Changing Network Topology	After Tuning Hyperparameters	After Dropping Insignificant Features
Mean test accuracy	69.8%	82.3%	83.2%	85.0%	**88.5%**
	Increment:	+12.5%	+0.9%	+1.8%	+3.5%

While all steps in training feedforward neural networks contribute to achieving a good result, it is worth noting that the improvements come mostly from preprocessing data and dropping out insignificant features. This highlights the importance of having the right inputs along with proper data representation over perfect network topology and hyperparameter values.

4.2 Gated recurrent units with time series dataset

[2] Formula of Q measure is provided in the Appendix.

In this section, we move on to enhancing classification performance using gated recurrent units with the time series data of pupillary response.

Cleaning data. The time series data requires cleaning as the occasional blink of pupil returns a pupil size of zero. Following the approach of [1], we corrected these value using linear interpolation between the earlier and later nonzero values. We do this for both left and right pupil data, and then use the average of the two for our classification task.

Choosing types of neural networks. Time series data are naturally suit to recurrent neural networks. These type of networks have loops in its architecture, allowing it to pass previous information forward [8]. In other words, these networks have sequential memory that allows it to recognize sequential patterns. Additionally, recurrent neural networks can process inputs of any length.

Two of the most widely used recurrent neural networks are long short-term memory (LSTM) and gated recurrent units (GRU). Through the use of gating mechanism, both LSTM and GRU are able to overcome the vanishing gradient problem that Tanh units (plain vanilla RNN) suffer [8]. Intuitively, the gates modulate the flow of information: they learn which data in the sequence is important to keep or throw away.

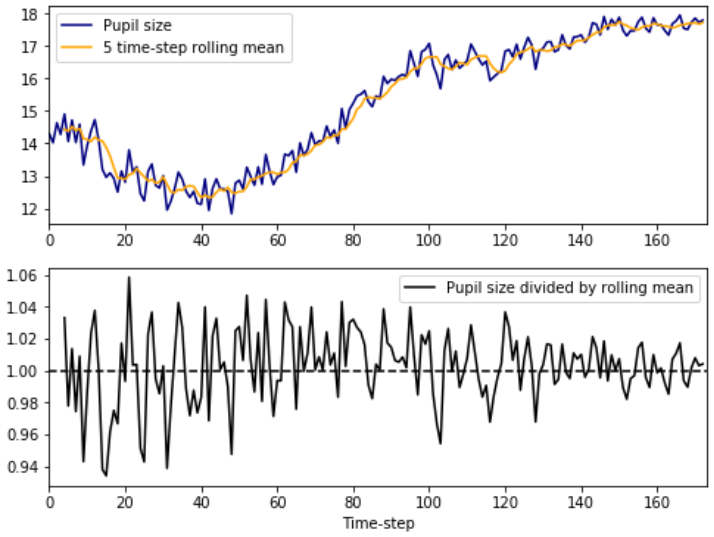
The key difference between LSTM and GRU is that LSTM has three gates (forget, input and output), whereas GRU has only two (reset and update). And unlike LSTM, GRU does not have cell states. Indeed, GRU performs less calculations and are faster to train than LSTM. Yet, it has been shown in various applications to achieve performance on-par with that of LSTM [9]. Given GRU's well-rounded strengths, we choose to implement it in our experiments.

As we are performing a binary classification task, a final linear layer is added at the end of our GRU. This linear layer takes the last hidden states of GRU as inputs and returns a probability of a sample belonging to each class.

Changing Network Topology and Tuning Hyperparameters. Using the cleaned time series data as described above, the model really struggles to beat the initial benchmark. For our first attempt, we use a hidden size of 40 and a learning rate of 0.01, this gives a test accuracy of only 55.1%. After performing a grid search, the model performance improved to 64.1%. This was achieved with the same hidden size but a smaller learning rate of 0.001. Unlike the previous section, the learning rate here plays an influential role in helping the model achieve a better result. This is more in line with the consensus that learning rate is one of the most, if not the most important hyperparameter [7].

Representing data differently. Leveraging on our finding from previous section that having a proper data representation can significantly boost model performance, we come up with a different way to represent our time series data. Instead of using the cleaned time series of pupil size, we calculate a five time-step rolling mean and divide the pupil size by the rolling average. This ratio captures the change in pupil size as compared to its average size from previous five time-step. A ratio of more than one means the pupil size has been increasing. The opposite is true when the ratio is smaller than one. Using this ratio, the data is automatically normalized.

Figure 1. A visual example of the new data representation we use (the bottom subplot).



With this new data representation, the model performance is boosted significantly. For a hidden size of 40 and a learning rate of 0.001, our GRU can now achieve 92.3% test accuracy, beating the initial benchmark as well as the best feedforward neural network model. Again, our experiment shows that having a proper data representation is vital.

Table 5. GRU classification performance improves significantly when the new data representation is used.

Using raw pupil data		*Learning Rate*		
		0.1	0.01	0.001
Hidden Size	30	51.3%	53.8%	61.5%
	40	51.3%	55.1%	64.1%
	50	55.1%	60.3%	62.8%

Using new data representation		*Learning Rate*		
		0.1	0.01	0.001
Hidden Size	30	57.7%	60.3%	85.9%
	40	67.9%	69.2%	**92.3%**
	50	52.6%	62.8%	88.5%

Incorporating the most useful feature from previous section. To utilize the finding of feature importance according to Q measure from the previous section, we try to incorporate '1st principal component' feature into our best GRU model by adding it as an extra input at the final linear layer[3]. However, it turns out that doing so does not help increase the model classification performance further. It appears that our best GRU model has already learned, from time series data, the information that '1st principal component' feature is capturing.

Applying Q Measure. We also try to directly apply Q measure at the linear layer of our best GRU model (still the one with hidden size of 40 and learning rate of 0.001 with no added '1st principal component' feature). As with our previous section, the aim is to find insignificant features and drop them. Unfortunately, dropping the feature with lowest Q score did not increase model performance like it did previously: the test accuracy falls to 87.2%. In addition, we try dropping features with the second, third and fourth lowest Q score, none of which yields a fruitful outcome. This seems to suggest that either Q measure is ineffective in this case, or all features are actually somewhat useful.

Adding a dropout layer before the linear layer. As a final attempt to improve the performance of our GRU model, we try adding a dropout layer before the linear layer using a dropout rate of 0.1 and 0.2. The rational is to randomly drop neurons during training in order to prevent overfitting [10]. Unfortunately, in our experiments, adding a dropout layer lead to either a lower test set performance or on-par performance with longer training time.

We settle with a final GRU model (Table 6 below) which achieves a 92.3% test accuracy. Like in the previous section, the biggest performance contributor comes from ensuring proper input data representation – a step that gives as much as 28.2% (=92.3-64.1) increase in the test accuracy.

Table 6. Our final GRU model

Input Neurons	**Hidden Neurons**	**Output Neurons**	**Loss Function**	**Optimizer**	**Learning Rate**
1	40	2	Cross Entropy	Adam	0.001

5 Results and Discussion

Our final model (Table 6) achieves a test accuracy of 92.3%, which is slightly lower than 95% reported by [1]. Reasons for the discrepancy may include: the slightly bigger dataset, the use of other features and/or the effect of ensemble learning. Regardless of the reasons, our results supports the claim that pupillary response data can be used to reliably differentiate genuine from posed anger. Equally, we demonstrated a successful use of both feedforward neural network and gated recurrent units for this classification problem.

Unlike the study of [1], we also provide a full ranking (Table 3) to show which features are the most important for differentiating genuine from posed anger for the summary statistics dataset. By far, '1st principal component' is the most important feature: its Q score stands out from that of all other features and not using it in the feedforward model causes a huge 25.3% drop in the classification accuracy.

Moving onto time series data, we also try to utilize Q measure again in two ways. First is an indirect use: we try to incorporate '1st principal component' into our gated recurrent unit model by adding it as an extra input at the final linear layer. Second is a direct use: we apply Q measure on the final linear layer of the gated recurrent unit in an attempt to find the least useful features and drop them. Unfortunately, both approaches fail to improve our gated recurrent unit model performance. Either Q measure is ineffective in this case, or all the final hidden state features are actually somewhat useful. All in all, Q measure is clearly useful for our feedforward neural network and the summary

[3] So, the input size of the final linear layer will be hidden size + 1.

statistics dataset; our results for this part outperform that presented by [2]. However, the same cannot be concluded for Q measure experiments relating to the gated recurrent unit and the time series dataset.

When comparing both our final feedforward neural network and gated recurrent unit model with our initial benchmark, we observe a marked improvement in our model performance (Table 4 and 5). The former beats the benchmark by 18.7%, whereas the latter beats the benchmark by 22.5%. While there are many steps during the training process that contribute to achieving good results, the improvements come mostly from preprocessing data and correctly dropping out insignificant features. This highlights the importance of having the right inputs along with proper data representation over perfect network topology and hyperparameter values.

6 Conclusion and Future Work

In this paper, we replicated the classification task found in [1] and our results strongly support their claim that pupillary response patterns can be used to differentiate genuine from posed anger. Equally, we showed that neural networks are useful for such tasks. Our final feedforward model achieved a mean test accuracy of 88.5% with the summary statistics dataset, whereas our final gated recurrent unit model achieved a test accuracy of 92.3% with the time series dataset. Using Q measure, we provide extra insights into the importance of each feature used in this classification task for the summary statistic datasets. We discovered that '1st Principal Component' is by far the single most useful feature for differentiating genuine from posed anger, whereas 'Difference2' is the least useful. For time series dataset, Q measure turns out to be less beneficial.

We recognize that these results are subject to some limitations. First, number of samples in our dataset is not big. If more data (both more participants and more videos) can be collected in the future, this could increase the reliability and accuracy of neural network results. Second, our results are based on one run of experiment due to time constraints. It would be better to get an access to a GPU and do more than one run of experiments to get more reliable results under a shorter amount of time. On a different note, it might also be worthwhile spending more time to come up with better data representation techniques, especially for the time series dataset. Alternatively, it might also worth experimenting other advanced deep learning techniques such as long-short term memory and/or attention models and compare the results with that of gated recurrent units. Finally, Q measure is only one of many techniques that can be utilized for feature selection in neural networks. It would be interesting to see whether other techniques such as functional measures [2] and permutation importance [11] can give comparable or superior results.

References

1. Chen, L., Gedeon, T.D., Hossain, M.Z., Caldwell S.: Are you really Angry? Detecting emotion veracity as a proposed tool for interaction. Proceeding of the 29th Australian Conference on Human-Computer Interaction, 412-416 (2017)
2. Gedeon, T.D.: Data mining of inputs: Analyzing magnitude and functional measures. International Journal of Neural Systems, 8(2), 209-218 (1997)
3. LeCun, Y., Bottou, L., Orr, G.B., Muller, K.: Efficient BackProp. Springer (1998)
4. Kohavi, R.: A Study of Cross-Validation and Boostrap for Accuracy Estimation and Model Selection. Proceedings of the 14th International Joint Conference on Artificial Intelligence, 1137-1143 (1995)
5. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press (2012)
6. Pytorch Documentation: https://pytorch.org/docs/stable/index.html. Last Accessed: 3 May 2020.
7. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press (1995)
8. Lipton, Z., Berkowitz J.: A Critical Review of Recurrent Neural Networks for Sequence Learning. arXiv: 1506.00019v4 (2015)
9. Chung J., Gulcehre C., Cho K., Bengio Y.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modelling. arXiv: 1412.3555 (2014)
10. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 1929-1958 (2014)
11. Altmann, A., Toloski, L., Sander, O., Lengauer, T.: Permutation Importance: a corrected feature importance measure. Bioinformatics, 26(10), 1340-1347 (2010)
12. Hossain, M.Z., Gedeon, T.D.: Classifying Posed and Real Smiles from Observers' Peripheral Physiology. 11th International Conference on Pervasive Computing Technologies for Healthcare, EAI Conference Series, Barcelona, Spain, 1-4 (2017)
13. Gedeon, T.D.: Indicators of Input Contributions: Analyzing the Weight matrix. Proceedings Australia New Zealand International Conference on Intelligent Information Systems, Adelaide, 166-169 (1996)

Appendix

Q Measure Formula [2]. First, calculate the contribution of an input to a hidden neuron. That is: take the absolute value of weight of an input neuron to a hidden neuron and divide it by the sum of absolute value of all input neuron weights going to that same hidden neuron:

$$P_{ij} = \frac{|w_{ij}|}{\sum_{p=1}^{ni} |w_{pj}|} \tag{1}$$

Second, calculate the contribution of a hidden neuron to an output neuron. That is: take the absolute value of weight of a hidden neuron to an output neuron and divide it by the sum of absolute value of all hidden neuron weights going to that same output neuron:

$$P_{jk} = \frac{|w_{jk}|}{\sum_{r=1}^{nh} |w_{rk}|} \tag{2}$$

Finally, we arrive at the contribution of an input neuron to an output neuron, which is the product of the above two equations sum over all hidden neurons:

$$Q_{ik} = \sum_{r=1}^{nh} (P_{ir} \times P_{rk}) \tag{3}$$

Q measure ranges between 0 and 1, with higher value indicating more importance.