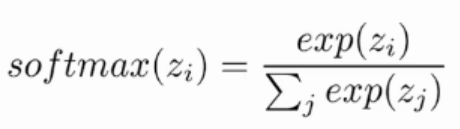
1. Standard Scalar:
   1. Standard scalar is done to resize the distribution so that the mean becomes 0 and SD becomes 1.
   2. The idea behind the StandardScaler is that variables that are measured at different scales do not contribute equally to the fit of the model and the learning function of the model and could end up creating a bias.
   3. So, to deal with this potential problem, we need to standardize the data (μ = 0, σ = 1) that is typically used before we integrate it into the machine learning model.
2. Different methods for time series: https://davidbetancourt.net/deep-learning-for-high-dimensional-time-series/
3. The features are highly correlated as seen this result. It shows the correlations where |correlation between features|>0.3.
4. Although PCA yielded good results, it doesn't matter if the columns will be jumbled, we will get the same result.
5. A bias has to be added to the columns towards the previous columns as they are dependent (time series). [As seen in the above plot, the data with shuffled columns also yields similar accuracy scores as PCA doesn't consider the order (order matters here as the data is a time series) which is flawed for this particular dataset.  
     
   So, we need to consider the bias of a column towards its previous columns.]
6. LSTMs are a type of recurrent neural network that specifically addresses the issue of vanishing gradients and is designed to capture long-term dependencies in sequential data. They have a more complex architecture with memory cells and gating mechanisms, making them well-suited for tasks involving long-term dependencies. Traditional RNNs, on the other hand, are more flexible in terms of architecture but can struggle with long-term dependencies due to the vanishing gradient problem.
7. Loss function for multiclass classification: sparse\_categorical\_crossentropy [**FILL IN DETAILS**]
8. softmax as an activation function.
   1. The sigmoid activation function gives the value between 0 and 1. The probability that the data point belongs to class 1 does not take into account the probability of the other classes.
   2. Similar to the sigmoid activation function the SoftMax function returns the probability of each class. Here is the equation for the SoftMax activation function. 
9. ~~After the prediction, the inverse transform of the standard scalar has to be applied (which was used to reduce bias) to get the actual y\_predicted~~.
10. The bias was taken care of using the dropout probabilities and so I removed the standard scalar.
11. The 4 parameters calculated for testing the model:
    1. Accuracy: It measures the overall correctness of the predicted labels compared to the true labels. It is calculated as the ratio of the total number of correctly predicted labels to the total number of samples. A higher accuracy indicates a better performance of the model.
    2. Precision: It measures the ability of the model to correctly predict positive samples (i.e., true positives) out of the total predicted positive samples (i.e., true positives + false positives). It is calculated as the ratio of true positives to the sum of true positives and false positives. Precision indicates how well the model avoids false positives.
    3. Recall (also known as Sensitivity or True Positive Rate): It measures the ability of the model to correctly predict positive samples (i.e., true positives) out of the total actual positive samples (i.e., true positives + false negatives). It is calculated as the ratio of true positives to the sum of true positives and false negatives. Recall indicates how well the model avoids false negatives.
    4. F1-score: It is the harmonic mean of precision and recall, and provides a balance between precision and recall. It is calculated as 2 times the product of precision and recall divided by the sum of precision and recall. F1-score is a single value that combines both precision and recall, and is often used when both false positives and false negatives are important.

These evaluation metrics are commonly used to assess the performance of classification models and provide insights into different aspects of model performance, such as overall accuracy, ability to avoid false positives, ability to avoid false negatives, and balance between precision and recall.

1. Different approaches for rime series classification: <https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification/>
2. The simulation of causality or creating the bias towards the previous data is working as expected as shuffling of the columns did not yield the same result as the unshuffled data. The accuracy is very low (94% compared to 5.3% in this case) meaning that a dependency has been created between the columns.
3. My ideas: Give the series as a vector [Check by iterating through various sizes of vector i.e., divide the whole vector into 2, 3 ,... n(number of columns)]. Did not work as arrays are given for training and so the doesn’t matter the size of the vector, everything is converted to arrays and each is broken into individual features (which is the data itself!).
4. Used Scikit-learn Dynamic Time Warping SVM and it doesn’t stop running. [Link for reference: <https://effectiveml.com/dynamic-time-warping-for-sequence-classification> ]
5. some research questions for time series classification:
   1. Can deep learning models outperform traditional machine learning models for time series classification tasks? Yes
   2. How can time series data be effectively preprocessed and transformed to improve classification accuracy? Check this – No need as NN takes care of it.
   3. Can transfer learning be used to improve time series classification accuracy, and if so, what transfer learning strategies are most effective? Check this
   4. How can uncertainty in time series classification models be quantified and communicated to end-users?
   5. How can time series classification models be made robust to missing or incomplete data? Check this
   6. How can time series classification models be adapted to handle streaming or online data?
   7. Can ensemble methods improve the accuracy and robustness of time series classification models? Computing id better and doesn’t consider dependecies but NN performs better.
   8. How can interpretability of time series classification models be improved, particularly for deep learning models?
   9. What impact do different feature selection techniques have on the performance of time series classification models?
   10. How can the class imbalance problem in time series classification be effectively addressed?
6. NN advantages:
   1. Scalability: Deep neural networks can scale to very large datasets and can be parallelized across multiple processors or GPUs. This is achieved through mini-batch training, where the training data is divided into small batches and the weights and biases are updated based on the average error across the batch. This allows the model to be trained on large datasets that may not fit in memory, and can significantly speed up training and inference.
   2. Scalability: Deep neural networks can scale to very large datasets and can be parallelized across multiple processors or GPUs. This is achieved through mini-batch training, where the training data is divided into small batches and the weights and biases are updated based on the average error across the batch. This allows the model to be trained on large datasets that may not fit in memory, and can significantly speed up training and inference.
   3. End-to-end training: Deep neural networks can be trained end-to-end, meaning the model can learn to extract features and make predictions in a single step. This is achieved through a process of backpropagation, where the error between the network's output and the true output is propagated backwards through the network to adjust the weights and biases. End-to-end training avoids the need for manual feature engineering, which can be time-consuming and may miss important features. It also allows the model to learn the most relevant features for the task, rather than relying on pre-defined features.
   4. End-to-end training: Deep neural networks can be trained end-to-end, meaning the model can learn to extract features and make predictions in a single step. This is achieved through a process of backpropagation, where the error between the network's output and the true output is propagated backwards through the network to adjust the weights and biases. End-to-end training avoids the need for manual feature engineering, which can be time-consuming and may miss important features. It also allows the model to learn the most relevant features for the task, rather than relying on pre-defined features.