Machine learning challenge

1:

**Preprocessing & Feature Engineering**

All written images were denoised and normalized. Spoken data was also normalized using Cepstral Mean & Variance Normalization. Since True and False were not equally distributed in the data set, we duplicated the spoken and written data (adding random rotation too) and created an additional set for training, with roughly 50/50 T/F distribution in order to reduce bias in the model. We created a lot of other features, including PCA-based representations, Gaussian noise, robust rescaling, VAE followed by k-means clustering of the latent representations, etc., but they did not appear to improve the performance of the model.

**Learning Algorithm Tried & Parameter Tuning**

We began by testing various classifiers and exploring feature extraction methods before deciding to construct a neural network. The neural network originally took three inputs: images, processed by 2D convolutional layers; MFCC recordings, processed by 1D convolutional layers feeding into a GRU; vectors of extracted features, processed by dense feedforward layers. After conducting *very*extensive testing of various combinations of Layers and parameters, we obtained our best results without inputting a vector of extracted features and by instead relying upon preprocessed versions of the image and MFCC data sets provided. To validate the approach, we used a combination of training and validation accuracy (2000 examples from the training set were held aside for validation), as well as the training and validation losses; we also monitored the progression over Epochs, ensuring the model doesn’t stagnate and stop learning, or seem to overfit. As the features we extracted manually did not contribute to a better accuracy, we decided to focus on improving the dataset preprocessing and further hyperparameter tuning of the NN. We introduced image rotation to the written and padding to the spoken datasets. Another refinement was to duplicate the true examples in the training set until a balance of 50% true and 50% false examples was obtained, and then use this to pre-train the neural network. Training first upon a 50-50 data set (with 36 epochs) seemed to improve the speed of learning, which also allowed for more hyperparameters settings to be tested in the limited time available. Dropout was another important refinement which noticeably reduced overfitting and was kept in the final model. As a final step, we conducted additional training with increased batch size and decreased learning rate, achieving a final accuracy improvement (36 epochs).

**Discussion of the performance**

The model took ~35 minutes to train. The final version of the neural network had both high training accuracy and high validation accuracy (over 99% accuracy on both data sets). The model was able to achieve perfect accuracy on the training set when not using dropout but did so at the expense of overfitting. The final model was tuned through increasing dropout to provide a tradeoff between training and validation set accuracy, with values for the validation accuracy reaching 99.3% quite frequently. The 50-50 data set helps to improve the learning speed when pre-training, and combining it with image rotation also assists the neural networks to obtain higher accuracy. During the pre-training process, the accuracy rate highly fluctuates in the early stage and it gradually becomes stable with a smaller growth rate when it is close to 99 percent. The second phase of training allows the neural network to adjust its weights to a distribution of labels more similar to the target data set.

Thank you for your time and please feel free to ask any questions.

2:

**Our chronological thoughts:**

We were aware that the digit recognition problem on MNIST (binary digits from 0-9) can be solved already well with a fully-connected neural network comprising 2-3 hidden layers. Moreover, on this problem, a plain CNN an architecture can take the accuracy above 99%. Thus, it was very clear that for the images of the digits, the subnetwork has to be a CNN-based network.

With the speech data added to it, we were provided the mid-frequency cepstral coefficients (MFCCs) which are 2D images rather than time series data. Thus, we thought initially that using another CNN subnetwork for the MFCCs should do. However, based on discussions and some seeking around the internet, we concluded that CNNs are designed for images that have spatial dependence with the neighbors. However, when we look MFCC data, the continuity exists along the horizontal direction, however, along the vertical direction the data is independent. Hence, CNNs may not necessarily deliver the expected performance.

Since the dataset is simple, we decided to build a fully-connected sub-network for MFCCs and rigorously train the overall network. In addition, we attempted a few feature engineeering methods and parameter tunings, and details on these aspects are described below.

**Feature engineering:**

Mel-frequency cepstral coefficients (MFCCs) are a spectral representation of sound and one of the most widely used in automatic speech recognition. The signal is divided into very short frames of typically 10-25ms, where the MFCCs are a non-linear transformation of the sound spectrum in each frame.

We computed the first and second order derivatives of the MFCCs: Delta and delta-delta coefficients. The delta cepstral coefficients (rate of change) are computed from the MFCCs and the delta-delta (acceleration) cepstral coefficients are computed from the delta coefficients. They are often appended to the MFCCs in automatic speech recognition systems to add dynamic information, as the coefficients only encode static information. Adding delta coefficients has consistently been shown to improve speech recognition accuracy (Kumar, Kim, Stern, 2011). Furthermore, we used min-max normalization to scale all input onto the same scale, as normalizing the MFCCs has been shown to reduce the effect of noise contamination on the classifier (Rehr & Gerkmann, 2015). When running the models we noted that prediction error rates were higher in models that included the delta coefficients than in models that used only MFCCs, and therefore, we excluded the delta coefficients from our final model.

**Learning algorithm and parameter tuning:**

After initial research, we found that convolutional neural networks (CNNs) are often successfully used for speech classification. Therefore, we initially decided to use CNNs for this task. However, our experiments showed that using only fully-connected layers for speech encoding worked better compared to the CNNs. We suspect, that the reason for lower performance on CNN is that the data in MFCC is uncorrelated along the vertical direction, while CNN assums otherwise. We implemented a multi-input structure: The image and audio data are first given as separate inputs to separate parts of the model. The image input is fed through three sets of convolutional and max-pooling layers, followed by two dense layers. The audio input is processed by four fully-connected layers in the final model. The two parallel models are then merged by concatenation, followed by a dense layer, to produce the final binary class prediction. We further tuned the number of epochs, dropout fractions and the batch size and, found that increasing the number of epochs and reducing the dropout fractions lead to decreased error rate. 

**Class distribution**

A remarkable feature of the dataset is the extremely unbalanced class distribution, with an approximate ratio of 1:9. When fitting the model, we initially adjusted the class weights to balance out the unbalanced classes, weighting the *True* class 9 times as heavy as the *False* class. However, when evaluating the performance of the model on the test data, we noted that the class distribution is equally unbalanced, and balancing the class weights led to higher prediction error rates. Therefore, we reverted to the unbalanced class weights.

**Discussion:**

The results indicate that the lowest error rate is achieved in the last attempt, as shown in the table below. The setup of the model is one the least complex. i.e., unbalanced class weights and no feature engineering included. This contradicts the previous work by Kumar, Kim, and Stern (2011) in the sense that numerous effort on data preprocessing is not required. One reason relates to the advantage of applying the CNN algorithm, it automatically extracts the useful features. Another reason links to the complexity of the dataset in this study. The representation of number in both verbal and handwritten formats is less complex when comparing with other cases such as action words.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attempt** | **Test error rate**  **(see Codalab)** | **Epochs** | **Batch size** | **Balanced class weights** | **Dropouts** | **Delta coefficients included** |
| 1 | 0.0083 | 200 | 128 | ✔ | 0.25 | x |
| 2 | 0.0006 | 300 | 128 | x | 0.25 | x |
| 3 | 0.0096 | 300 | 128 | x | 0.25 | ✔ |
| 4 | 0.0051 | 300 | 64 | x | 0.10 | x |

**References:**

Kshitiz Kumar, Chanwoo Kim, Richard M. Stern,“Delta-Spectral cepstral coefficients for robust speech recognition”, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP*), Prague, Czech Republic, 2011, 10.1109/ICASSP.2011.5947425.

, Brisbane, QLD, Australia, 2015, 10.1109/ICASSP.2015.7177994. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*Robert Rehr, Timo Gerkmann, “Cepstral Noise Subtraction for Robust Automatic Speech Recognition“, in

3:

**The model**

We believed that the presence of two separate datasets (spoken and written) with no individual labels was the main problem of the ML challenge.  To tackle this problem, we initialized two separate models aiming to extract as much information as possible out of both datasets individually. After doing so, the two models merge into a single multilayer perceptron (MLP) which uses the combined intermediate representations of both datasets as input and ultimately classifies the examples.

We build our model using the Keras library using Tensorflow as backend. During our experiments we extensively tested the performance of two separate MLPs versus the performance of a MLP combined with a CNN. We found that the latter led to best performance. The CNN model was used to learn intermediate representations of the spoken data and the MLP learned intermediate representations of the written data. We added Dropout layers which helped reduce overfitting.

Our model is trained incrementally in multiple sessions. This allowed us to efficiently tune model parameters and change data input sizes in between training sessions while monitoring the change in performance.

**Feature engineering**

We zero-padded the spoken data in order to account for the variable frame lengths. There was a large imbalance in the labels of the training data which required you to either upsample or downsample (We found that upsampling led to better results).

We tried to supplement the MFCC features with delta and delta-delta coefficents aiming to enrich the representation of the spoken numbers. This effort did not result in additional model performance.

**Parameter tuning**

Parameter tuning included increasing and decreasing the number of layers, changing the number of nodes per layer, tweaking the optimizer and activation functions. During tuning, we roughly experienced the following:

Increasing the amount of layers and nodes vastly increased the required training time. Setting the amount of nodes too low resulted in the model getting stuck in what we believed to be a local minima.

Both the ReLu and tanh activation functions led to approximately similar model performance but ReLu is much less computationally expensive. We used the sigmoid activation function in the output layer of the multilayer perceptrion because this challenge involved a binary classification problem.

We used the ADAM optimizer with a slight decay in learning rate.

**Performance**

Our final solution achieved a 0.0059 error rate. Training our model took around 6 hours. We believe that model performance could be pushed even further by increasing the amount of epochs and/or layers/nodes. However, we did not have the hardware to make this computationally feasible.