UNIVERSITAT POLITÈCNICA DE CATALUNYA

CIÈNCIA I ENGINYERIA DE DADES

REPORT - ASSIGNMENT 5

PRESENTED BY

Bruno Pérez
Martí Farré

SUBJECT

PROCESSAT DEL LLENGUATGE ORAL I ESCRIT

June 16, 2023

1 Introduction

In this assignment, we will work on an audio classification task. Our goal will be diagnose COVID-19 using a small dataset of breathing, cough, and voice recordings. We could choose different audio classification task.

2 Exploratory Data Analysis

The Data Analysis we have done has been mainly to understand how the code provided works, what is the input and what is the expected output. This has helped us do the lab assignment.

3 Hyperparameter optimization

In the next sections we will display the results obtained by the hyperparameter tuning and model changes. The metrics displayed are the ones obtained in the epoch with the best validation set. The modifications made are mentioned as the version name. No other modification has been made if not mentioned.

3.1 MelSpectogram

Version	Elapsed time	Loss	AUC
Baseline	886s	0.6711	64.8%
lr = 0.005	431s	0.6902	62.1%
optim SGD	3624s	0.6650	66.1%
VGG13	1246s	0.7018	65.9%
VGG16	986s	0.6680	66.3%
VGG19	1537s	0.6737	67.7%

After doing several experiments, we can see that modifying the learning rate slightly reduces the AUC in exchange of a faster execution time. We could choose this version if we prioritized getting the results faster than a little bit more accurate.

On the other hand, changing the optimizer to SGD improves the AUC, but it's a lot slower. This could be the model to choose if we need the higher accuracy, with no regards of the execution time.

The changes in the model, specially the 16 and 19 version, provide a better AUC in exchange of a slightly higher elapsed time.

3.2 HuBERT

Version	Elapsed time	Loss	AUC
Baseline	1589s	0.6285	72.4%
lr = 0.0005	1095s	0.6314	72.0%
patience = 10	2096s	0.6285	72.4%
optim SGD	598s	0.6944	48.6%
optim SGD & momentum = 0.5	596s	0.6951	49.6%

The results we get after experimenting with the HuBERT model suggests that changing the optimizer to SGD does not result in a better AUC, not even modifying the momentum.

Again, similar as the MelSpectogram model, changing the learning rate makes the code faster, but with a lower AUC. We have to decide whether we prefer a faster or a more accurate version of our code.

We have also verified that increasing our model patience does not improve the AUC, it just makes the baseline model slower.

4 Cross-validation

To improve our model accuracy we have modified the dropout value of our model. Here we can see the results.

4.1 HuBERT

Version	Elapsed time	Loss	AUC
Dropout 0	1500s	0.6574	72.4%
Dropout 0.1	1597s	0.6285	72.4%
Dropout 0.3	1783s	0.6510	70.5%
Dropout 0.5	1790s	0.6419	69.9%

We can see that increasing the dropout value lowers the model AUC, while 0.1 is the baseline value, so it stays the same. The version where the Dropout probability is 0 (so there is no Dropout) obtains the same results as using a 0.1 value, but slightly faster.

5 Weighted average of layers' hidden states

In this modifications we have used the two last layers of the model to have a different version.

The first model we have tried just uses the values of the penultimate hidden layer. The second version uses the last and penultimate layers and averages to try increasing the model output. It can be seen that there are two different second versions. On the first one we give more weight to the last layer, while in the second-second version the opposite. The implementation can be found at the end of the document.

Here can be seen the results.

Version	Elapsed time	Loss	AUC
Model 1	1350s	0.6296	71.9%
Model 2.1	1804s	0.6585	72.4%
Model 2.2	1598s	0.6370	72.3%

It can be seen no real improvement is achieved implementing this changes.

6 Conclusions

6.1 MelSpectogram conclusions

The biggest improvement we have seen in the MelSpectogram model is changing the architecture to VGG16 or VGG19. We think the model we would use is the VGG19 version, with improved accuracy, but with a slightly longer elapsed time.

6.2 HuBERT conclusions

With the HuBERT model we haven't been able to improve the baseline AUC. The hyperparameter tuning hasn't been successful, even worsening the AUC by a large margin, like changing the optimizer. The Dropout parameter changing hasn't improved the model on the AUC metric, but using a dropout probability of 0, the model trains 1:30 minutes faster. Averaging the weighted layers hidden states hasn't been successful either.

6.3 Model election

We have chosen the HuBERT model, because it outperforms the MelSpectogram version almost alwats. The modifications we have decided to do to the HuBERT model are changing the dropout value to 0. The output of the mode can be found as an attachment of the zip presented.

Modified Code

We have only modified code in the section 5. The other modifications have just been hyperparameter tuning.

Model 1

```
class HubertForAudioClassification(nn.Module):
def __init__(self, adapter_hidden_size=64):
    super().__init__()
```

```
        self.hubert = HubertModel.from_pretrained(MODEL)

        hidden_size = self.hubert.config.hidden_size

        self.adaptor = nn.Sequential(
            nn.Linear(hidden_size, adapter_hidden_size),
            nn.ReLU(True),
            nn.Dropout(0.1),
            nn.Linear(adapter_hidden_size, hidden_size),
        )

        self.classifier = nn.Sequential(
            nn.Linear(hidden_size, adapter_hidden_size),
            nn.ReLU(True),
            nn.Dropout(0.1),
            nn.Linear(adapter_hidden_size, 1),
        )

    def freeze_feature_encoder(self):
        """
        Calling this function will disable the gradient computation for the feature enc
        not be updated during training.
        """
        self.hubert.feature_extractor._freeze_parameters()

    def forward(self, x):
        # x shape: (B,E)
        x = self.hubert(x, output_hidden_states=True).hidden_states[-2]

        x = self.adaptor(x)

        # pooling
        x, _ = x.max(dim=1)

        # Mutilayer perceptron
        out = self.classifier(x)
        # out shape: (B,1)

        # Remove last dimension
        return out.squeeze(-1)
        # return shape: (B)
```

Model 2.1

```
class HubertForAudioClassification(nn.Module):
    def __init__(self, adapter_hidden_size=64):
        super().__init__()

        self.hubert = HubertModel.from_pretrained(MODEL)

        hidden_size = self.hubert.config.hidden_size

        self.adaptor = nn.Sequential(
            nn.Linear(hidden_size, adapter_hidden_size),
            nn.ReLU(True),
            nn.Dropout(0.1),
            nn.Linear(adapter_hidden_size, hidden_size),
        )

        self.classifier = nn.Sequential(
            nn.Linear(hidden_size, adapter_hidden_size),
            nn.ReLU(True),
            nn.Dropout(0.1),
            nn.Linear(adapter_hidden_size, 1),
        )

    def freeze_feature_encoder(self):
        """
        Calling this function will disable the gradient computation for the feature enc
        not be updated during training.
        """
        self.hubert.feature_extractor._freeze_parameters()

    def forward(self, x):
        # x shape: (B,E)
        outputs = self.hubert(x, output_hidden_states=True)

        x1 = outputs.hidden_states[-1]
        x2 = outputs.hidden_states[-2]

        # Compute weights
        w1 = torch.tensor(0.6)  # Weight for x1
        w2 = torch.tensor(0.4)  # Weight for x2

        # Compute weighted average
        x = w1 * x1 + w2 * x2
```

```
        x = self.adaptor(x)

        # pooling
        x, _ = x.max(dim=1)

        # Mutilayer perceptron
        out = self.classifier(x)
        # out shape: (B,1)

        # Remove last dimension
        return out.squeeze(-1)
        # return shape: (B)
```

Model 2.2

```
class HubertForAudioClassification(nn.Module):
    def __init__(self, adapter_hidden_size=64):
        super().__init__()

        self.hubert = HubertModel.from_pretrained(MODEL)

        hidden_size = self.hubert.config.hidden_size

        self.adaptor = nn.Sequential(
            nn.Linear(hidden_size, adapter_hidden_size),
            nn.ReLU(True),
            nn.Dropout(0.1),
            nn.Linear(adapter_hidden_size, hidden_size),
        )

        self.classifier = nn.Sequential(
            nn.Linear(hidden_size, adapter_hidden_size),
            nn.ReLU(True),
            nn.Dropout(0.1),
            nn.Linear(adapter_hidden_size, 1),
        )

    def freeze_feature_encoder(self):
        """
        Calling this function will disable the gradient computation for the feature enc
        not be updated during training.
        """
        self.hubert.feature_extractor._freeze_parameters()
```

```
def forward(self, x):
    # x shape: (B,E)
    outputs = self.hubert(x, output_hidden_states=True)

    x1 = outputs.hidden_states[-1]
    x2 = outputs.hidden_states[-2]

    # Compute weights
    w1 = torch.tensor(0.6)  # Weight for x1
    w2 = torch.tensor(0.4)  # Weight for x2

    # Compute weighted average
    x = w2 * x1 + w1 * x2

    x = self.adaptor(x)

    # pooling
    x, _ = x.max(dim=1)

    # Mutilayer perceptron
    out = self.classifier(x)
    # out shape: (B,1)

    # Remove last dimension
    return out.squeeze(-1)
    # return shape: (B)
```