Malware Classification Using Ember features and Feed Forward Neural Network

used a simple 3-layer feed-forward network with dropout and batch-norm.

At the end it can be seen that the training accuracy is 99% and validation accuracy is 92% , and testing accuracy is 92% The model is converging in the inital 5 epochs itself. since a malware sample can belongs to multiple classes at the same time. and the label with max probability is considered as true label.the validation and test accuracy around 92% is still good. I belive that if we consider top 3 labels then the accuracy can be much improved.

the corressponding confusion matrix and classification reports and plots can be seen at the end.

the architecure implemented here is very simple and more similar to the one in this link: https://towardsdatascience.com/pytorch-tabular-multiclass-classification-9f8211a123ab

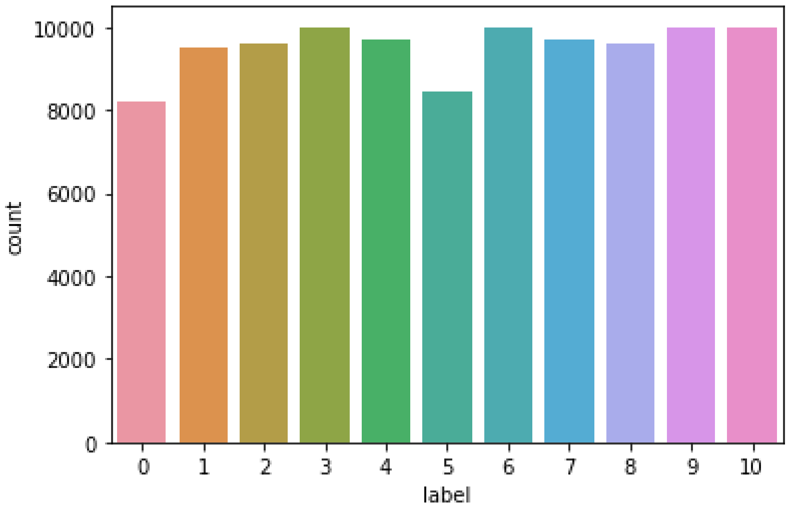
In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
```

In [2]:
```
sorel_dir = '../Dataset/sorel'
data = np.load(os.path.join(sorel_dir,'sorel_data.npz'),allow_pickle=True)
data['arr_0'].shape,data['arr_1'].shape
```

Out[2]:
```
((104746, 2381), (104746, 13))
```

In [3]:
```
df = pd.DataFrame(data=data['arr_1'][:,-1], columns=["label"])
sns.countplot(x = 'label', data=df);
```



In [4]:
```
# Split into train+val and test
X_trainval, X_test, y_trainval, y_test = train_test_split(data['arr_0'], data['arr_1

# Split train into train-val
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, test_size=
```

In [5]:
```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
X_train, y_train = np.array(X_train), np.array(y_train)
X_val, y_val = np.array(X_val), np.array(y_val)
X_test, y_test = np.array(X_test), np.array(y_test)
```

In [7]:
```
class ClassifierDataset(Dataset):
    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data
    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]
    def __len__ (self):
        return len(self.X_data)

class MulticlassClassification(nn.Module):
    def __init__(self, num_feature, num_class):
        super(MulticlassClassification, self).__init__()

        self.layer_1 = nn.Linear(num_feature, 512)
        self.layer_2 = nn.Linear(512, 128)
        self.layer_3 = nn.Linear(128, 64)
        self.layer_out = nn.Linear(64, num_class)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.2)
        self.batchnorm1 = nn.BatchNorm1d(512)
        self.batchnorm2 = nn.BatchNorm1d(128)
        self.batchnorm3 = nn.BatchNorm1d(64)

    def forward(self, x):
        x = self.layer_1(x)
        x = self.batchnorm1(x)
        x = self.relu(x)

        x = self.layer_2(x)
        x = self.batchnorm2(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.layer_3(x)
        x = self.batchnorm3(x)
        x = self.relu(x)
        x = self.dropout(x)

        x = self.layer_out(x)

        return x
```

In [8]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
values,counts = np.unique(y_train,return_counts=True)
class_weights = 1./torch.tensor(counts, dtype=torch.float)
print(class_weights)
```

```
tensor([0.0002, 0.0001, 0.0001, 0.0001, 0.0001, 0.0002, 0.0001, 0.0001, 0.0001,
        0.0001, 0.0001])
```

In [9]:
```
EPOCHS = 50
BATCH_SIZE = 16
LEARNING_RATE = 0.0007
```

```
NUM_FEATURES = 2381
NUM_CLASSES =  11
```

In [10]:

```
train_dataset = ClassifierDataset(torch.from_numpy(X_train).float(), torch.from_nump
val_dataset = ClassifierDataset(torch.from_numpy(X_val).float(), torch.from_numpy(y_
test_dataset = ClassifierDataset(torch.from_numpy(X_test).float(), torch.from_numpy(
train_loader = DataLoader(dataset=train_dataset,
                          batch_size=BATCH_SIZE,
)
val_loader = DataLoader(dataset=val_dataset, batch_size=1)
test_loader = DataLoader(dataset=test_dataset, batch_size=1)
```

In [31]:

```
model = MulticlassClassification(num_feature = NUM_FEATURES, num_class=NUM_CLASSES)
model.load_state_dict(torch.load(os.path.join(sorel_dir,'classification.pth')))
model.eval()
model.to(device)
```

Out[31]:

```
MulticlassClassification(
  (layer_1): Linear(in_features=2381, out_features=512, bias=True)
  (layer_2): Linear(in_features=512, out_features=128, bias=True)
  (layer_3): Linear(in_features=128, out_features=64, bias=True)
  (layer_out): Linear(in_features=64, out_features=11, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.2)
  (batchnorm1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (batchnorm2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (batchnorm3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
)
```

In [14]:

```
model = MulticlassClassification(num_feature = NUM_FEATURES, num_class=NUM_CLASSES)
model.to(device)

criterion = nn.CrossEntropyLoss(weight=class_weights.to(device))
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
print(model)
```

```
MulticlassClassification(
  (layer_1): Linear(in_features=2381, out_features=512, bias=True)
  (layer_2): Linear(in_features=512, out_features=128, bias=True)
  (layer_3): Linear(in_features=128, out_features=64, bias=True)
  (layer_out): Linear(in_features=64, out_features=11, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.2)
  (batchnorm1): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (batchnorm2): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
  (batchnorm3): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
)
```

In [11]:

```
def multi_acc(y_pred, y_test):
    y_pred_softmax = torch.log_softmax(y_pred, dim = 1)
    _, y_pred_tags = torch.max(y_pred_softmax, dim = 1)

    correct_pred = (y_pred_tags == y_test).float()
    acc = correct_pred.sum() / len(correct_pred)

    acc = torch.round(acc * 100)

    return (acc)
```

In [12]:

```
accuracy_stats = {
```

```
    'train': [],
    "val": []
}
loss_stats = {
    'train': [],
    "val": []
}
```

In [15]:

```
for e in tqdm(range(1, EPOCHS+1)):

    # TRAINING
    train_epoch_loss = 0
    train_epoch_acc = 0
    model.train()
    for X_train_batch, y_train_batch in train_loader:
        X_train_batch, y_train_batch = X_train_batch.to(device), y_train_batch.to(de
        optimizer.zero_grad()

        y_train_pred = model(X_train_batch)

        train_loss = criterion(y_train_pred, y_train_batch)
        train_acc =  multi_acc(y_train_pred, y_train_batch)

        train_loss.backward()
        optimizer.step()

        train_epoch_loss += train_loss.item()
        train_epoch_acc += train_acc.item()


    # VALIDATION
    with torch.no_grad():

        val_epoch_loss = 0
        val_epoch_acc = 0

        model.eval()
        for X_val_batch, y_val_batch in val_loader:
            X_val_batch, y_val_batch = X_val_batch.to(device), y_val_batch.to(device

            y_val_pred = model(X_val_batch)

            val_loss = criterion(y_val_pred, y_val_batch)
            val_acc = multi_acc(y_val_pred, y_val_batch)

            val_epoch_loss += val_loss.item()
            val_epoch_acc += val_acc.item()
    loss_stats['train'].append(train_epoch_loss/len(train_loader))
    loss_stats['val'].append(val_epoch_loss/len(val_loader))
    accuracy_stats['train'].append(train_epoch_acc/len(train_loader))
    accuracy_stats['val'].append(val_epoch_acc/len(val_loader))


    print(f'Epoch {e+0:03}: | Train Loss: {train_epoch_loss/len(train_loader):.5f} |
```

```
Epoch 001: | Train Loss: 0.41790 | Val Loss: 0.22116 | Train Acc: 88.328| Val Acc: 9
2.053
Epoch 002: | Train Loss: 0.23859 | Val Loss: 0.20268 | Train Acc: 92.360| Val Acc: 9
2.876
Epoch 003: | Train Loss: 0.19627 | Val Loss: 0.20098 | Train Acc: 93.507| Val Acc: 9
2.924
Epoch 004: | Train Loss: 0.16484 | Val Loss: 0.19252 | Train Acc: 94.409| Val Acc: 9
3.413
Epoch 005: | Train Loss: 0.14514 | Val Loss: 0.18975 | Train Acc: 94.925| Val Acc: 9
3.484
```

```
Epoch 006: | Train Loss: 0.12980 | Val Loss: 0.18957 | Train Acc: 95.459| Val Acc: 9
3.974
Epoch 007: | Train Loss: 0.11753 | Val Loss: 0.19869 | Train Acc: 95.746| Val Acc: 9
4.081
Epoch 008: | Train Loss: 0.10754 | Val Loss: 0.20689 | Train Acc: 96.138| Val Acc: 9
3.890
Epoch 009: | Train Loss: 0.10033 | Val Loss: 0.20554 | Train Acc: 96.235| Val Acc: 9
4.368
Epoch 010: | Train Loss: 0.09180 | Val Loss: 0.22072 | Train Acc: 96.574| Val Acc: 9
3.950
Epoch 011: | Train Loss: 0.08742 | Val Loss: 0.22180 | Train Acc: 96.726| Val Acc: 9
4.069
Epoch 012: | Train Loss: 0.08105 | Val Loss: 0.21773 | Train Acc: 96.926| Val Acc: 9
4.308
Epoch 013: | Train Loss: 0.07668 | Val Loss: 0.23866 | Train Acc: 97.096| Val Acc: 9
4.021
Epoch 014: | Train Loss: 0.07139 | Val Loss: 0.25575 | Train Acc: 97.274| Val Acc: 9
4.165
Epoch 015: | Train Loss: 0.06980 | Val Loss: 0.24474 | Train Acc: 97.323| Val Acc: 9
4.332
Epoch 016: | Train Loss: 0.06365 | Val Loss: 0.25575 | Train Acc: 97.488| Val Acc: 9
4.189
Epoch 017: | Train Loss: 0.06236 | Val Loss: 0.26237 | Train Acc: 97.572| Val Acc: 9
4.117
Epoch 018: | Train Loss: 0.05824 | Val Loss: 0.26963 | Train Acc: 97.734| Val Acc: 9
4.224
Epoch 019: | Train Loss: 0.05700 | Val Loss: 0.27907 | Train Acc: 97.832| Val Acc: 9
3.902
Epoch 020: | Train Loss: 0.05291 | Val Loss: 0.27230 | Train Acc: 97.986| Val Acc: 9
4.177
Epoch 021: | Train Loss: 0.05105 | Val Loss: 0.26992 | Train Acc: 98.056| Val Acc: 9
4.344
Epoch 022: | Train Loss: 0.04678 | Val Loss: 0.30489 | Train Acc: 98.168| Val Acc: 9
4.272
Epoch 023: | Train Loss: 0.04538 | Val Loss: 0.29419 | Train Acc: 98.282| Val Acc: 9
4.379
Epoch 024: | Train Loss: 0.04311 | Val Loss: 0.28303 | Train Acc: 98.388| Val Acc: 9
4.284
Epoch 025: | Train Loss: 0.04053 | Val Loss: 0.32409 | Train Acc: 98.454| Val Acc: 9
4.558
Epoch 026: | Train Loss: 0.03950 | Val Loss: 0.29567 | Train Acc: 98.534| Val Acc: 9
4.463
Epoch 027: | Train Loss: 0.03631 | Val Loss: 0.33275 | Train Acc: 98.645| Val Acc: 9
3.735
Epoch 028: | Train Loss: 0.03532 | Val Loss: 0.34068 | Train Acc: 98.706| Val Acc: 9
4.248
Epoch 029: | Train Loss: 0.03483 | Val Loss: 0.30766 | Train Acc: 98.741| Val Acc: 9
4.379
Epoch 030: | Train Loss: 0.03177 | Val Loss: 0.30858 | Train Acc: 98.842| Val Acc: 9
4.714
Epoch 031: | Train Loss: 0.03059 | Val Loss: 0.33766 | Train Acc: 98.922| Val Acc: 9
4.177
Epoch 032: | Train Loss: 0.02927 | Val Loss: 0.32903 | Train Acc: 98.989| Val Acc: 9
4.153
Epoch 033: | Train Loss: 0.02803 | Val Loss: 0.33556 | Train Acc: 98.991| Val Acc: 9
4.379
Epoch 034: | Train Loss: 0.02742 | Val Loss: 0.32791 | Train Acc: 99.058| Val Acc: 9
4.153
Epoch 035: | Train Loss: 0.02613 | Val Loss: 0.30905 | Train Acc: 99.147| Val Acc: 9
4.666
Epoch 036: | Train Loss: 0.02455 | Val Loss: 0.33800 | Train Acc: 99.123| Val Acc: 9
4.200
Epoch 037: | Train Loss: 0.02455 | Val Loss: 0.33508 | Train Acc: 99.163| Val Acc: 9
4.511
Epoch 038: | Train Loss: 0.02352 | Val Loss: 0.34711 | Train Acc: 99.238| Val Acc: 9
4.391
Epoch 039: | Train Loss: 0.02235 | Val Loss: 0.34183 | Train Acc: 99.260| Val Acc: 9
4.320
Epoch 040: | Train Loss: 0.02218 | Val Loss: 0.33779 | Train Acc: 99.251| Val Acc: 9
```

```
4.332
Epoch 041: | Train Loss: 0.02232 | Val Loss: 0.36364 | Train Acc: 99.278| Val Acc: 9
4.356
Epoch 042: | Train Loss: 0.02118 | Val Loss: 0.35037 | Train Acc: 99.327| Val Acc: 9
4.368
Epoch 043: | Train Loss: 0.01996 | Val Loss: 0.34793 | Train Acc: 99.352| Val Acc: 9
4.678
Epoch 044: | Train Loss: 0.01811 | Val Loss: 0.35122 | Train Acc: 99.388| Val Acc: 9
4.785
Epoch 045: | Train Loss: 0.01849 | Val Loss: 0.35650 | Train Acc: 99.408| Val Acc: 9
4.379
Epoch 046: | Train Loss: 0.01829 | Val Loss: 0.39512 | Train Acc: 99.383| Val Acc: 9
4.308
Epoch 047: | Train Loss: 0.01701 | Val Loss: 0.40093 | Train Acc: 99.423| Val Acc: 9
3.699
Epoch 048: | Train Loss: 0.01763 | Val Loss: 0.34270 | Train Acc: 99.420| Val Acc: 9
4.499
Epoch 049: | Train Loss: 0.01704 | Val Loss: 0.34712 | Train Acc: 99.445| Val Acc: 9
4.523
Epoch 050: | Train Loss: 0.01658 | Val Loss: 0.42581 | Train Acc: 99.480| Val Acc: 9
4.177
```

In [26]:
```python
torch.save(model.state_dict(), os.path.join(sorel_dir,'classification.pth'))
```

In [28]:
```python
os.path.join(sorel_dir,'classification.pth')
```

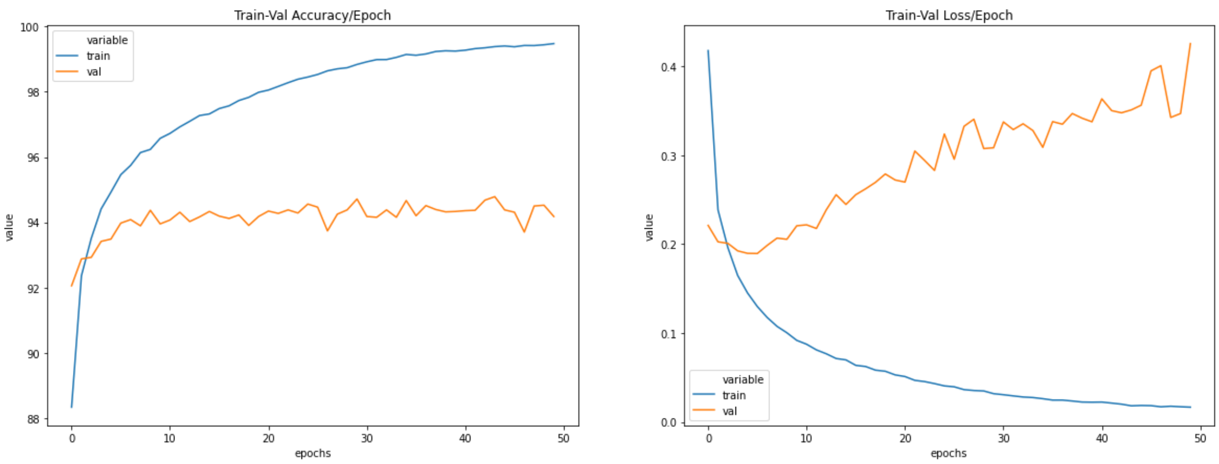
Out[28]:
```
'../Dataset/sorel/classification.pth'
```

In [30]:
```python
!ls ../Dataset/sorel/
```

```
classification.pth  sorel_label.csv        test-features.npz
db                  sorel_label_sum25.csv  validation-features.npz
meta.db             sorel_malware1.csv
sorel_data.npz      sorel_malware.csv
```

In [16]:
```python
# Create dataframes
train_val_acc_df = pd.DataFrame.from_dict(accuracy_stats).reset_index().melt(id_vars
train_val_loss_df = pd.DataFrame.from_dict(loss_stats).reset_index().melt(id_vars=['
# Plot the dataframes
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20,7))
sns.lineplot(data=train_val_acc_df, x = "epochs", y="value", hue="variable",  ax=axe
sns.lineplot(data=train_val_loss_df, x = "epochs", y="value", hue="variable", ax=axe
```

Out[16]:
```
Text(0.5, 1.0, 'Train-Val Loss/Epoch')
```



In [32]:
```python
y_pred_list = []
y_prob_list = []
with torch.no_grad():
    model.eval()
    for X_batch, _ in test_loader:
```

```
            X_batch = X_batch.to(device)
            y_test_pred = model(X_batch)
            _, y_pred_tags = torch.max(y_test_pred, dim = 1)

            y_pred_list.append(y_pred_tags.cpu().numpy())
            y_prob_list.append(y_test_pred)


    y_pred_list = [a.squeeze().tolist() for a in y_pred_list]
```

In [33]:
```
y_prob_list[0],y_pred_list[0]
```

Out[33]:
```
(tensor([[-15.1259, -23.0870, -25.0988, -23.7968, -21.3749, -21.3992, -19.6963,
          -17.8799,   5.0449, -22.6265, -18.0614]], device='cuda:0'),
 8)
```

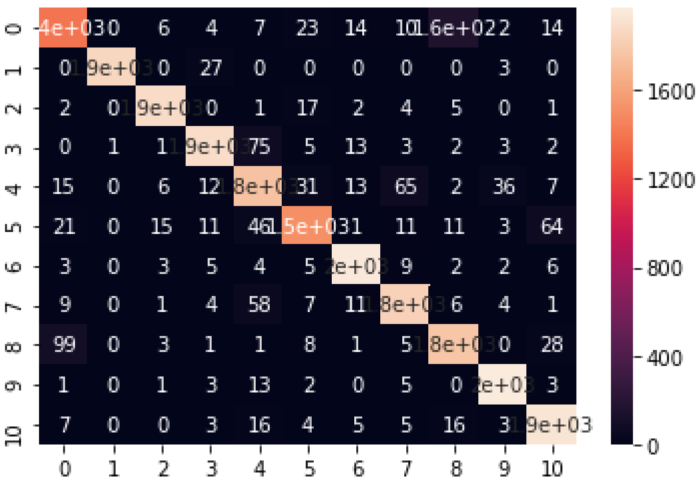
In [16]:
```
y_test.astype('long')
```

Out[16]:
```
array([ 8,  6, 10, ...,  4,  4,  7])
```

In [17]:
```
confusion_matrix_df = pd.DataFrame(confusion_matrix(y_test.astype('long'), y_pred_li

sns.heatmap(confusion_matrix_df, annot=True)
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2aed144500f0>
```



In [19]:
```
confusion_matrix_df
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	10
0	1401	0	6	4	7	23	14	10	161	2	14
1	0	1866	0	27	0	0	0	0	0	3	0
2	2	0	1889	0	1	17	2	4	5	0	1
3	0	1	1	1895	75	5	13	3	2	3	2
4	15	0	6	12	1757	31	13	65	2	36	7
5	21	0	15	11	46	1510	1	11	11	3	64
6	3	0	3	5	4	5	1959	9	2	2	6
7	9	0	1	4	58	7	11	1841	6	4	1
8	99	0	3	1	1	8	1	5	1773	0	28
9	1	0	1	3	13	2	0	5	0	1972	3

	0	1	2	3	4	5	6	7	8	9	10
10	7	0	0	3	16	4	5	5	16	3	1936

In [18]:
```
print(classification_report(y_test.astype('long'), y_pred_list))
```

```
              precision    recall  f1-score   support

           0       0.90      0.85      0.88      1642
           1       1.00      0.98      0.99      1896
           2       0.98      0.98      0.98      1921
           3       0.96      0.95      0.96      2000
           4       0.89      0.90      0.90      1944
           5       0.94      0.89      0.91      1693
           6       0.97      0.98      0.98      1998
           7       0.94      0.95      0.94      1942
           8       0.90      0.92      0.91      1919
           9       0.97      0.99      0.98      2000
          10       0.94      0.97      0.95      1995

    accuracy                           0.95     20950
   macro avg       0.94      0.94      0.94     20950
weighted avg       0.95      0.95      0.94     20950
```

In [36]:
```
ember2018 = '../Dataset/ember_zip/ember2018'
```

In [35]:
```
EPOCHS = 50
BATCH_SIZE = 16
LEARNING_RATE = 0.0007
NUM_FEATURES = 2381
NUM_CLASSES =  2
```

In [64]:
```
test_dataset = np.load(os.path.join(ember2018,'ember2018_test_data.npz'),allow_pickl
X_train,y_train = test_dataset['arr_0'],test_dataset['arr_1']
```