Evaluation of Artificial Neural Networks

Brandon Marquez Salazar

I. Introduction

Nowadays in machine learning, learning algorithms are fundamental part to focus on. Those algorithms can help to reduce overfitting and improve generalization. This assignment will focus on different strategies for training algorithms.

II. Core Concepts and Methods

For machine learning, training methods were developed to enhance the generalization capabilities of the model. Each training method was thought focusing on different problems and requirements.

A. Gradient Descent

This is the main training method, was proposed doing a gradient to update the weights of each layer. There are some gradients of this method.

- Batch Gradient Descent Here, the dataset is completely used for each update.
- Stochastic Gradient Descent Here, each sample is used for each update. This can increase the convergence speed for large datasets.
- Mini-batch Gradient Descent Here, a small subset of the dataset is used for each update. It can be the one between the batch and the stochastic gradient descent.

B. Adaptive Algorithms

Adaptive algorithms starts to use momentum and learning rate to improve the convergence speed, trying to avoid local minima.

- Adaptive Gradient Descent (AdaGrad) Is a gradient descent method that uses the learning rate to adapt the step size. This can help low weighed features to be better included in the final model.
- RMSProp It's an AdaGrad adaptation which intends to slow the learning rate decay.
- Adam or Adaptive Moment Estimation It's a modification og AdaGrad that uses momentum and learning rate to improve the convergence speed. This solution is the most used. Often offering a good behaviour in most cases.

III. Applications and Discussion

In this section, a little demonstration of Mini-batch Gradient Descent and Adam will be shown. Also, it'll be implemented a RMSProp wirh Mini-batch Gradient Descent. The reason is that, both are a more complete approach for each main training method.

A. Testing Mini-batch Gradient Descent and Adam

For this experiment it'll be used sklearn.

```
!pip install numpy
!pip install scikit-learn
```

1) Importing modules and libraries: After installing the libraries, it's needed to import the required modules and libraries.

```
from sklearn.datasets         import load_iris
from sklearn.neural_network   import MLPClassifier
from sklearn.metrics          import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
import random
```

2) Loading the dataset: As in the previous report, the iris dataset will be used.

```
x, y = load_iris(return_X_y=True)
```

3) Creating the MLP with a 3-layer 10-nodes ANN: Here, the MLP will be implemented using `Adam` and with a batch of 32.

```
mlp = MLPClassifier(
  hidden_layer_sizes=(10, 10, 10),
  solver='adam',
  batch_size=32,
  max_iter=500,
  random_state=random.randint(0, 20000)
)
```

4) K-fold cross-validation: A `k-fold` cross-validation will be implemented to evaluate the model.

```
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=1)
kf_accuracies = []

# Perform K-fold cross-validation
for train_index, test_index in kf.split(x):
    x_train_fold, x_test_fold = \
      x[train_index], x[test_index]
    y_train_fold, y_test_fold = \
      y[train_index], y[test_index]

    # Train the model
    mlp.fit(x_train_fold, y_train_fold)

    # Evaluate the model
    y_pred = mlp.predict(x_test_fold)
    accuracy = accuracy_score(y_test_fold, y_pred)
```

```
    kf_accuracies.append(accuracy)

# Calculate the average accuracy
kf_accuracy = sum(kf_accuracies) / len(kf_accuracies)
print(f"K-fold Cross-Validation Accuracy with "
    f"Adam and Mini-Batch Gradient Descent: {kf_accuracy}")
```

```
K-fold Cross-Validation Accuracy with Adam and Mini-
Batch Gradient Descent: 0.9666666666666666
```

B. Testing RMSProp with Mini-batch Gradient Descent

Here, tensorflow will be needed.

```
!pip install numpy
!pip install scikit-learn
!pip install tensorflow
```

C. Importing modules and libraries

After installing the libraries, it's needed to import the required modules and libraries.

```
import tensorflow as tf
from   tensorflow.keras.models     import Sequential
from   tensorflow.keras.layers     import Dense
from   tensorflow.keras.optimizers import RMSprop
from   sklearn.datasets            import load_iris
from   sklearn.model_selection     import KFold
from   sklearn.preprocessing       import StandardScaler
from   sklearn.metrics             import accuracy_score
import random
```

1) Loading the dataset: Again, the iris dataset will be used.

```
x, y = load_iris(return_X_y=True)

# Feature standardization
scaler = StandardScaler()
x = scaler.fit_transform(x)
```

2) Creating the MLP with a 3-layer 10-nodes ANN: Here, the MLP will have a final softmax layer with 3 nodes, one for each class. This implementation will be more complex than the previous one.

```
def create_model():
    model = Sequential()
    model.add(Dense(10, input_dim=x.shape[1],
      activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    optimizer = RMSprop(learning_rate=0.001)
    model.compile(
      optimizer=optimizer,
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy']
    )
    return model
```

3) K-fold cross-validation: For this one, the **k-fold** cross-validation will be used too.

```
# Define K-fold cross-validation
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=1)
kf_accuracies = []

# Perform K-fold cross-validation
for train_index, test_index in kf.split(x):
    x_train_fold, x_test_fold = \
      x[train_index], x[test_index]
    y_train_fold, y_test_fold = \
      y[train_index], y[test_index]

    # Create and train the model
    model = create_model()
    model.fit(
      x_train_fold,
      y_train_fold,
      epochs=50,
      batch_size=32,
      verbose=0
    )

    # Evaluate the model
    y_pred = model.predict(x_test_fold)
    y_pred_classes = tf.argmax(y_pred, axis=1)
    accuracy = accuracy_score(y_test_fold, y_pred_classes)
    kf_accuracies.append(accuracy)
```

```
# Calculate the average accuracy
kf_accuracy = sum(kf_accuracies) / len(kf_accuracies)
print(f"K-fold Cross-Validation Accuracy with "+
    f"RMSprop and Mini-Batch Gradient Descent: {kf_accuracy}")
```

```
K-fold Cross-Validation Accuracy with RMSprop and Mini-
Batch Gradient Descent: 0.8066666666666666
```

IV. Conclusion

As seen before, the **Adam** approach is more robust than the **RMSprop** one. With the same iterations and epochs, the **Adam** approach has a higher accuracy, allowing also to train the model faster. Those approaches can show the power of learning methods to train the model, and the why were created.

References

[1] M. Ekman. *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow*. Pearson Education, 2021. ISBN: 9780137470297. URL: https://books.google.com.mx/books?id=wNnPEAAAQBAJ.

[2] S.S. Haykin. *Neural Networks and Learning Machines*. Neural networks and learning machines v. 10. Prentice Hall, 2009. ISBN: 9780131471399. URL: https://books.google.com.mx/books?id=K7P36lKzI_QC.

[3] Christopher M. *Pattern Recognition and Machine Learning*. en. 1st ed. Information Science and Statistics. New York, NY: Springer, Aug. 2006.

[4] Gireen Naidu, Tranos Zuva, and Elias Mmbongeni Sibanda. "A Review of Evaluation Metrics in Machine Learning Algorithms". In: *Artificial Intelligence Application in Networks and Systems*. Springer International Publishing, 2023, pp. 15–25. ISBN: 9783031353147. DOI: 10.1007/978-3-031-35314-7_2. URL: http://dx.doi.org/10.1007/978-3-031-35314-7_2.

[5] quarto. *Guide*. URL: https://quarto.org/docs/guide/.

[6] quarto. *PDF Options*. URL: https://quarto.org/docs/reference/formats/pdf.html.

[7] quarto. *Welcome Page*. URL: https://quarto.org.

[8] scikit-learn. *MLPClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[9] Parul Singh. *A Must-Read History of Artificial Intelligence*. 2020. URL: https://cyfuture.com/blog/history-of-artificial-intelligence/.