Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Deep Learning y Sistemas inteligentes - Sección 20
Jeyner David Arango Ruíz - 201106
Oscar Jose Mendez Tello - 20402
Cristian Fernando Laynez Bachez - 201281

Construcciones de la mejor red neuronal posible por medio de técnicas de optimización

Resultados obtenidos

Primer modelo:

```
# Make predictions.
mlp_pred = mlp.predict(X_test)

# CV score
mlp_cv=cross_val_score(mlp, X_train, y_train, cv=10).mean()
```

```
# Accuracy: 1 is perfect prediction.
print('Accuracy: %.4f' % mlp.score(X_test, y_test))
# Cross-Validation accuracy
print('Cross-validation accuracy: %0.4f' % mlp_cv)
# Precision
print('Precision: %.4f' % precision_score(y_test, mlp_pred))
# Recall
print('Recall: %.4f' % recall_score(y_test, mlp_pred))
# f1 score: best value at 1 (perfect precision and recall) and worst at 0.
print('F1 score: %.4f' % f1_score(y_test, mlp_pred))
```

```
Accuracy: 0.8260
Cross-validation accuracy: 0.8174
Precision: 0.6399
Recall: 0.4028
F1 score: 0.4944
```

Segundo modelo:

Epoch and Batch Optimization

In [27]:
```
seed = 777
tf.random.set_seed(seed)
# define the grid search parameters
batch_size = [40, 50, 60, 80, 100, 150, 200]
epochs = [10, 25, 30, 50]
param_grid = dict(batch_size=batch_size, epochs=epochs)
param_grid
```

Out[27]:
```
{'batch_size': [40, 50, 60, 80, 100, 150, 200], 'epochs': [10, 25, 30, 50]}
```

Grid Search

In [28]:
```
gridSearch_grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)
gridSearch_result = gridSearch_grid.fit(X, y)
```

In [29]:
```
# gridSearch results
print("Best: %f using %s" % (gridSearch_result.best_score_, gridSearch_result.best_para
```

```
Best: 0.778642 using {'batch_size': 50, 'epochs': 10}
```

Tercer modelo:

Random Search

In [17]:
```
randomSearch_batch_size =np.arange(20, 501)
randomSearch_epochs = np.arange(80,201)
randomSearch_param_grid = dict(batch_size=randomSearch_batch_size, epochs=randomSearch_epochs)
```

In [20]:
```
RandomizedSearch_grid = RandomizedSearchCV(estimator=model, param_distributions=randomSearch_param_grid, n_iter
RandomizedSearch_result = RandomizedSearch_grid.fit(X, y)
```

In [21]:
```
# gridSearch results
print("Best: %f in Random Search using %s" % (RandomizedSearch_result.best_score_, RandomizedSearch_result.best
```

```
Best: 0.780110 in Random Search using {'epochs': 127, 'batch_size': 378}
```

Ideas para mejorar la red

- Primer modelo: Para esta red neuronal se optó por utilizar el clasificador Multi-layer perceptron. Se realizó una división de los datos en conjuntos de entrenamiento y prueba, el 20% es para el conjunto de pruebas. Se utilizó “Standard Scaler” para estandarizar las características en los conjuntos de entrenamiento y prueba. Esto es importante para que tengan una media de cero y una desviación estándar de uno, el cual facilita el entrenamiento de la red neuronal. La red cuenta con dos capas, la primera tiene 12 neuronas y la segunda 5 neuronas. Se insertó un número máximo de iteraciones, se utilizó control de aleatoriedad para controlar los pesos y los otros procesos aleatorios durante el entrenamiento. Se mezclan todos los datos para generalizar mejor, y se realiza la validación cruzada con 10 pliegues en los datos de entrenamiento.

- Segundo modelo: Para este escenario se utilizó una arquitectura secuencial. Tiene una capa densa con 20 neuronas. Se realiza una búsqueda de hiper parámetros para encontrar la mejor combinación posible de hiper parámetros entre el tamaño de lote y el número de épocas, estos valores se guardaron en un diccionario.
- Tercer modelo: Para este escenario se optó por una búsqueda aleatoria de hiper parámetros, prácticamente se utilizó el mismo escenario del segundo modelo pero con la diferencia que se optó por usar una búsqueda aleatoria.

Discusión

El objetivo de esta práctica es construir la mejor red neuronal posible según las técnicas de optimización de modificación de hiper parámetros. Se emplearon tres modelos de redes neuronales utilizando diversas técnicas de optimización. Para el primer modelo se obtuvo un accuracy de 0.81, el segundo modelo se obtuvo un accuracy de 0.77 y el tercero un accuracy de 0.78.

Los resultados de los tres modelos demuestran lo importante que es la configuración de los hiper parámetros. En el primer modelo logró la precisión más alta, esto podría deberse a su arquitectura específica y su validación cruzada. Por otro lado, el segundo y tercer modelos exploraron diferentes enfoques de búsqueda de hiper parámetros y aún lograron un rendimiento respetable. Estos resultados sugieren que la elección de la arquitectura y los hiper parámetros es crucial en el diseño de redes neuronales, y diferentes enfoques de optimización pueden llevar a resultados competitivos.

Conclusiones

- La optimización de modelos es un proceso crítico para mejorar el rendimiento y la generalización de los modelos.
- La estandarización de características es esencial para preparar los datos antes de alimentarlos a modelos de redes neuronales. Ayuda a normalizar las características y facilita el entrenamiento.
- La elección de la arquitectura de redes neuronales desempeña un papel fundamental en el rendimiento del modelo.

Repositorio

https://github.com/jeyner777/DeepLearning_Lab04.git