optimisers

June 11, 2025

[1]:
```python
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import numpy as np
```

[3]:
```python
df = pd.read_csv('titanic.csv')
```

[5]:
```python
df = df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)
```

[7]:
```python
df["Age"].fillna(df["Age"].median(), inplace=True)
df["Fare"].fillna(df["Fare"].median(), inplace=True)
df["Embarked"].fillna(df["Embarked"].mode()[0], inplace=True)
```

```
C:\Users\sajee\AppData\Local\Temp\ipykernel_20180\2319829390.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df["Age"].fillna(df["Age"].median(), inplace=True)
C:\Users\sajee\AppData\Local\Temp\ipykernel_20180\2319829390.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

```
  df["Fare"].fillna(df["Fare"].median(), inplace=True)
C:\Users\sajee\AppData\Local\Temp\ipykernel_20180\2319829390.py:3:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df["Embarked"].fillna(df["Embarked"].mode()[0], inplace=True)
```

[9]:
```
df["Sex"] = df["Sex"].map({"male": 0, "female": 1})
embarked_dummies = pd.get_dummies(df["Embarked"], prefix="Embarked",␣
 ↪drop_first=True)
df = pd.concat([df, embarked_dummies], axis=1).drop(columns=["Embarked"])
```

[11]:
```
df["Age"] = (df["Age"] - df["Age"].mean()) / df["Age"].std()
df["Fare"] = (df["Fare"] - df["Fare"].mean()) / df["Fare"].std()
```

[13]:
```
X = df.drop(columns=["Survived"]).values.astype(np.float32)
y = keras.utils.to_categorical(df["Survived"].values, num_classes=2)
```

[15]:
```
split_idx = int(0.8 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]
```

[17]:
```
def create_mlp(input_shape):
    model = keras.Sequential([
        keras.layers.Dense(64, activation="relu", input_shape=(input_shape,)),
        keras.layers.Dense(32, activation="relu"),
        keras.layers.Dense(2, activation="softmax")  # Multi-class␣
 ↪classification
    ])
    return model
```

[19]:
```
optimizers = {
    "SGD": keras.optimizers.SGD(learning_rate=0.01),
    "Momentum": keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
    "NAG": keras.optimizers.SGD(learning_rate=0.01, momentum=0.9,␣
 ↪nesterov=True),
    "Adam": keras.optimizers.Adam(learning_rate=0.001),
    "RMSprop": keras.optimizers.RMSprop(learning_rate=0.001),
    "Adagrad": keras.optimizers.Adagrad(learning_rate=0.01),
```

```
    "Adadelta": keras.optimizers.Adadelta(learning_rate=1.0),
    "Nadam": keras.optimizers.Nadam(learning_rate=0.001),
}
```

[21]:
```
def get_lr_schedule(schedule_type):
    if schedule_type == "StepDecay":
        return keras.optimizers.schedules.PiecewiseConstantDecay([5, 10], [0.
↪01, 0.005, 0.001])
    elif schedule_type == "ExponentialDecay":
        return keras.optimizers.schedules.
↪ExponentialDecay(initial_learning_rate=0.01, decay_steps=1000, decay_rate=0.
↪96)
    elif schedule_type == "PiecewiseConstantDecay":
        return keras.optimizers.schedules.PiecewiseConstantDecay([5000, 10000],␣
↪[0.01, 0.005, 0.001])
    elif schedule_type == "CosineDecay":
        return keras.optimizers.schedules.CosineDecay(initial_learning_rate=0.
↪01, decay_steps=10000, alpha=0.1)
    return 0.01
```

[23]:
```
results = {}
for opt_name, optimizer in optimizers.items():
    for schedule_name in ["StepDecay", "ExponentialDecay",␣
↪"PiecewiseConstantDecay", "CosineDecay"]:
        print(f"Training with {opt_name} + {schedule_name}...")


        model = create_mlp(X_train.shape[1])


        model.compile(
            optimizer=keras.optimizers.
↪Adam(learning_rate=get_lr_schedule(schedule_name)),
            loss="categorical_crossentropy",
            metrics=["accuracy"]
        )


        history = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=15, batch_size=32, verbose=0)


        final_accuracy = history.history["val_accuracy"][-1]
        results[f"{opt_name} + {schedule_name}"] = final_accuracy
```

```python
print("\nFinal Accuracy Results:")
for key, value in results.items():
    print(f"{key}: {value:.4f}")
```

```
Training with SGD + StepDecay…

C:\Users\sajee\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Training with SGD + ExponentialDecay…
Training with SGD + PiecewiseConstantDecay…
Training with SGD + CosineDecay…
Training with Momentum + StepDecay…
Training with Momentum + ExponentialDecay…
Training with Momentum + PiecewiseConstantDecay…
Training with Momentum + CosineDecay…
Training with NAG + StepDecay…
Training with NAG + ExponentialDecay…
Training with NAG + PiecewiseConstantDecay…
Training with NAG + CosineDecay…
Training with Adam + StepDecay…
Training with Adam + ExponentialDecay…
Training with Adam + PiecewiseConstantDecay…
Training with Adam + CosineDecay…
Training with RMSprop + StepDecay…
Training with RMSprop + ExponentialDecay…
Training with RMSprop + PiecewiseConstantDecay…
Training with RMSprop + CosineDecay…
Training with Adagrad + StepDecay…
Training with Adagrad + ExponentialDecay…
Training with Adagrad + PiecewiseConstantDecay…
Training with Adagrad + CosineDecay…
Training with Adadelta + StepDecay…
Training with Adadelta + ExponentialDecay…
Training with Adadelta + PiecewiseConstantDecay…
Training with Adadelta + CosineDecay…
Training with Nadam + StepDecay…
Training with Nadam + ExponentialDecay…
Training with Nadam + PiecewiseConstantDecay…
Training with Nadam + CosineDecay…

Final Accuracy Results:
SGD + StepDecay: 0.8715
SGD + ExponentialDecay: 0.8827
SGD + PiecewiseConstantDecay: 0.8715
SGD + CosineDecay: 0.8659
```

```
Momentum + StepDecay: 0.8771
Momentum + ExponentialDecay: 0.8771
Momentum + PiecewiseConstantDecay: 0.8603
Momentum + CosineDecay: 0.8492
NAG + StepDecay: 0.8603
NAG + ExponentialDecay: 0.8771
NAG + PiecewiseConstantDecay: 0.8827
NAG + CosineDecay: 0.8492
Adam + StepDecay: 0.8771
Adam + ExponentialDecay: 0.8715
Adam + PiecewiseConstantDecay: 0.8659
Adam + CosineDecay: 0.8603
RMSprop + StepDecay: 0.8547
RMSprop + ExponentialDecay: 0.8603
RMSprop + PiecewiseConstantDecay: 0.8715
RMSprop + CosineDecay: 0.8659
Adagrad + StepDecay: 0.8715
Adagrad + ExponentialDecay: 0.8883
Adagrad + PiecewiseConstantDecay: 0.8436
Adagrad + CosineDecay: 0.8603
Adadelta + StepDecay: 0.8659
Adadelta + ExponentialDecay: 0.8492
Adadelta + PiecewiseConstantDecay: 0.8603
Adadelta + CosineDecay: 0.8771
Nadam + StepDecay: 0.8547
Nadam + ExponentialDecay: 0.8492
Nadam + PiecewiseConstantDecay: 0.8771
Nadam + CosineDecay: 0.8547
```

[ ]: