Deep Learning Hyperparameter Tuning and Regularization in Python

With Code Examples

save for later

Introduction to Hyperparameter Tuning

Hyperparameter tuning is the process of optimizing the configuration of a deep learning model to improve its performance. It involves adjusting various settings that control the learning process, such as learning rate, batch size, and network architecture.

```
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
```

Swipe next →

Learning Rate Optimization

The learning rate determines the step size during gradient descent. Finding the optimal learning rate is crucial for model convergence and performance.

```
from keras.callbacks import LearningRateScheduler

def lr_schedule(epoch):
    return 0.001 * (0.1 ** int(epoch / 10))

model.fit(X_train, y_train, epochs=50, callbacks=
[LearningRateScheduler(lr_schedule)])
```

save for later

Batch Size Tuning

Batch size affects both the model's learning dynamics and computational efficiency. Smaller batch sizes can lead to more frequent updates, while larger ones can provide more stable gradients.

```
batch_sizes = [32, 64, 128, 256]
for batch_size in batch_sizes:
    history = model.fit(X_train, y_train, epochs=10,
batch_size=batch_size, validation_split=0.2)
    print(f"Batch size: {batch_size}, Validation accuracy:
{history.history['val_accuracy'][-1]}")
```

Swipe next ⟶

Dropout Regularization

Dropout is a regularization technique that prevents overfitting by randomly setting a fraction of input units to 0 during training.

```
from keras.layers import Dropout

model = Sequential([
    Dense(64, activation='relu', input_shape=(10,)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])
```

save for later

L1 and L2 Regularization

L1 and L2 regularization add penalties to the loss function based on the model's weights, encouraging simpler models and preventing overfitting.

```
from keras.regularizers import l1, l2, l1_l2

model = Sequential([
    Dense(64, activation='relu', kernel_regularizer=l1(0.01),
input_shape=(10,)),
    Dense(32, activation='relu', kernel_regularizer=l2(0.01)),
    Dense(1, activation='sigmoid', kernel_regularizer=l1_l2(l1=0.01,
l2=0.01))
])
```

Swipe next ⟶

follow for more

Early Stopping

Early stopping prevents overfitting by halting training when the validation loss stops improving.

```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
model.fit(X_train, y_train, epochs=100, validation_split=0.2, callbacks=
[early_stopping])
```

Swipe next ⟶

save for later

Grid Search for Hyperparameter Tuning

Grid search exhaustively searches through a predefined set of hyperparameters to find the best combination.

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier

def create_model(units=64, dropout_rate=0.5):
    model = Sequential([
        Dense(units, activation='relu', input_shape=(10,)),
        Dropout(dropout_rate),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model)
param_grid = {'units': [32, 64, 128], 'dropout_rate': [0.3, 0.5, 0.7]}
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3)
grid_result = grid.fit(X_train, y_train)
```

Swipe next ⟶

follow for more

Random Search for Hyperparameter Tuning

Random search samples hyperparameters from defined distributions, often finding good configurations more efficiently than grid search.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform, randint

param_distributions = {
    'units': randint(32, 256),
    'dropout_rate': uniform(0.1, 0.5),
    'batch_size': randint(16, 128),
    'epochs': randint(10, 100)
}

random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_distributions, n_iter=20, cv=3)
random_search_result = random_search.fit(X_train, y_train)
```

Swipe next →

save for later

Bayesian Optimization

Bayesian optimization uses probabilistic models to guide the search for optimal hyperparameters, balancing exploration and exploitation.

```
from skopt import BayesSearchCV
from skopt.space import Real, Integer

search_spaces = {
    'units': Integer(32, 256),
    'dropout_rate': Real(0.1, 0.5),
    'batch_size': Integer(16, 128),
    'epochs': Integer(10, 100)
}

bayes_search = BayesSearchCV(estimator=model,
search_spaces=search_spaces, n_iter=20, cv=3)
bayes_search_result = bayes_search.fit(X_train, y_train)
```

Swipe next →

Cross-Validation for Hyperparameter Tuning

Cross-validation helps assess the model's performance across different data splits, providing a more robust estimate of hyperparameter effectiveness.

```
from sklearn.model_selection import cross_val_score

def create_model(units=64, dropout_rate=0.5):
    model = Sequential([
        Dense(units, activation='relu', input_shape=(10,)),
        Dropout(dropout_rate),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=
['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model)
scores = cross_val_score(model, X, y, cv=5)
print(f"Mean accuracy: {scores.mean()}, Standard deviation:
{scores.std()}")
```

save for later

Learning Rate Scheduler

A learning rate scheduler dynamically adjusts the learning rate during training, potentially improving convergence and final performance.

```
from keras.callbacks import LearningRateScheduler
import math

def step_decay(epoch):
    initial_lr = 0.1
    drop = 0.5
    epochs_drop = 10.0
    lr = initial_lr * math.pow(drop, math.floor((1 + epoch) /
epochs_drop))
    return lr

lr_scheduler = LearningRateScheduler(step_decay)
model.fit(X_train, y_train, epochs=50, callbacks=[lr_scheduler])
```

follow for more

Hyperparameter Tuning with Keras Tuner

Keras Tuner is a library that helps automate the process of hyperparameter tuning for Keras models.

```
import keras_tuner as kt

def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int('units', min_value=32, max_value=512,
step=32),
                    activation='relu', input_shape=(10,)))
    model.add(Dropout(hp.Float('dropout', min_value=0.0, max_value=0.5,
step=0.1)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer=Adam(hp.Float('learning_rate', min_value=1e-
4, max_value=1e-2, sampling='log')),
                  loss='binary_crossentropy', metrics=['accuracy'])
    return model

tuner = kt.Hyperband(build_model, objective='val_accuracy',
max_epochs=50, factor=3, directory='my_dir', project_name='intro_to_kt')
tuner.search(X_train, y_train, epochs=50, validation_split=0.2)
best_model = tuner.get_best_models(num_models=1)[0]
```

Swipe next →

save for later

Model Checkpointing

Model checkpointing saves the best model during training, ensuring you retain the optimal weights even if training continues past the best point.

```
from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',
save_best_only=True, mode='min')
model.fit(X_train, y_train, epochs=100, validation_split=0.2, callbacks=
[checkpoint])

# Load the best model
best_model = load_model('best_model.h5')
```

Visualizing Hyperparameter Tuning Results

Visualizing the results of hyperparameter tuning can provide insights into the impact of different configurations on model performance.

```
import matplotlib.pyplot as plt
import seaborn as sns

results = pd.DataFrame(random_search.cv_results_)
plt.figure(figsize=(12, 8))
sns.scatterplot(x='param_dropout_rate', y='mean_test_score',
hue='param_units', data=results)
plt.title('Hyperparameter Tuning Results')
plt.xlabel('Dropout Rate')
plt.ylabel('Mean Test Score')
plt.show()
```

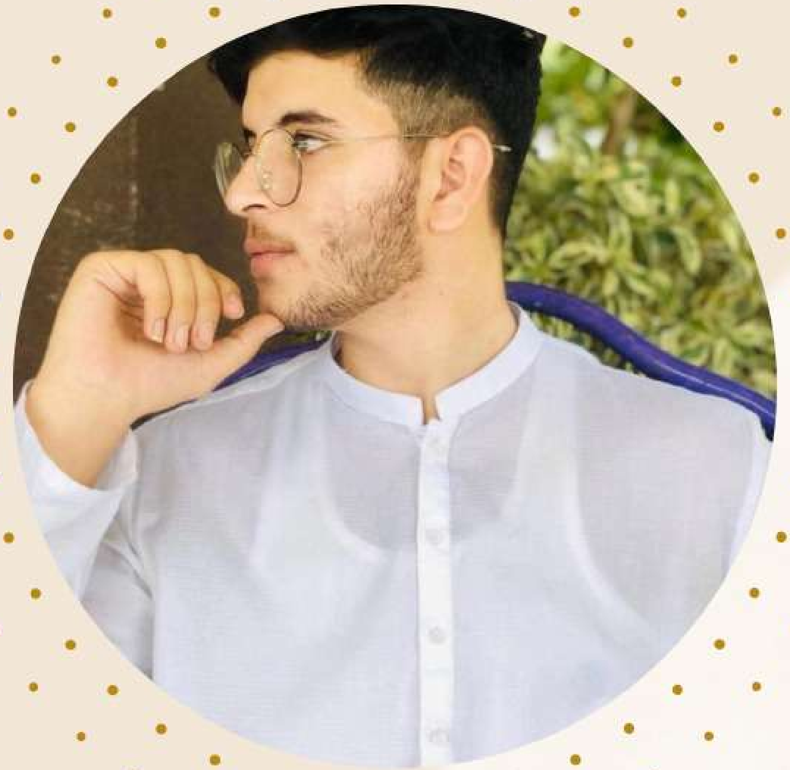
save for later

Additional Resources

- "Random Search for Hyper-Parameter Optimization" by Bergstra and Bengio (2012) ArXiv: https://arxiv.org/abs/1212.5701
- "Practical Bayesian Optimization of Machine Learning Algorithms" by Snoek, Larochelle, and Adams (2012) ArXiv: https://arxiv.org/abs/1206.2944
- "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization" by Li et al. (2017) ArXiv: https://arxiv.org/abs/1603.06560

Data scientist

&ML Engineer



Follow For More Data Science Content