Loss and Optimization

Ramin Zarebidoky (LiterallyTheOne)

20 Nov 2025

Loss and Optimization

Introduction

In the previous tutorial, we learned about **plotting** and **Tensorboard**. Here is the summary of the code that we have implemented so far.

```
# -------------------[ Setup ]-------------------
import os

os.environ["KERAS_BACKEND"] = "torch"

# -------------------[ Imports ]-------------------
from pathlib import Path

from matplotlib import pyplot as plt

import torch
from torch.utils.data import random_split, DataLoader

from torchvision.datasets import ImageFolder
from torchvision import transforms

import keras
from keras import layers
from keras.applications import MobileNetV2

import kagglehub

import datetime

# -------------------[ Load the data ]-------------------
path = kagglehub.dataset_download(
    "balabaskar/tom-and-jerry-image-classification")
```

```
data_path = Path(path) / "tom_and_jerry/tom_and_jerry"

trs = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ]
)

all_data = ImageFolder(data_path, transform=trs)

g1 = torch.Generator().manual_seed(20)
train_data, val_data, test_data = random_split(all_data, [0.7,
 ↪  0.2, 0.1], g1)

train_loader = DataLoader(train_data, batch_size=12,
 ↪  shuffle=True)
val_loader = DataLoader(val_data, batch_size=12, shuffle=False)
test_loader = DataLoader(test_data, batch_size=12, shuffle=False)

# -------------------[ Make the model ]-------------------
base_model = MobileNetV2(include_top=False, input_shape=(224,
 ↪  224, 3))

base_model.trainable = False

model = keras.Sequential(
    [
        layers.Input(shape=(3, 224, 224)),
        layers.Permute((2, 3, 1)),
        base_model,
        layers.Flatten(),
        layers.Dense(4, activation="softmax"),
    ]
)

model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

# -------------------[ Train the model ]-------------------
log_dir = "logs/fit/" +
 ↪  datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
```

```
tensorboard_callback =
 ↪  keras.callbacks.TensorBoard(log_dir=log_dir)

history = model.fit(
    train_loader,
    epochs=5,
    validation_data=val_loader,
    callbacks=[tensorboard_callback],
)

# -------------------[ Evaluate the model ]-------------------

loss, accuracy = model.evaluate(test_loader)

print("loss:", loss)
print("accuracy:", accuracy)

# -------------------[ Plot the training procedure
 ↪  ]-------------------

plt.figure()
plt.title("loss")
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.legend(["loss", "val_loss"])

plt.figure()
plt.title("accuracy")
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.legend(["accuracy", "val_accuracy"])

plt.show()
```

In this tutorial, we are going to learn more about **loss functions** and **optimizers** in **Keras**.

Your turn

Conclusion