Improving deep networks in Keras 3 and TensorFlow

The same ideas the TensorFlow way: Keras 3 layers, `compile` / `fit`, callbacks, and `tf.GradientTape` when you want your own training loop. TensorFlow 2.21 is stable and mostly receives fixes now; Google recommends Keras 3, JAX or PyTorch for new generative-AI work, and Keras 3 code like this also runs on JAX or PyTorch by changing one line.

Open in Google Colab · part of the inPace course library. Slides © DeepLearning.AI (CC BY-SA 2.0); this notebook: inPace (CC BY-SA 4.0).

Run it top to bottom (Runtime → Run all in Colab). Each part says what to look for.

```
# Works in Google Colab as it is (TensorFlow and Keras are already installed
there).
# On your own computer:  pip install tensorflow keras scikit-learn scipy
matplotlib
# One framework per notebook: PyTorch and TensorFlow can clash when loaded into
the same Python process.
import os
os.environ["KERAS_BACKEND"] = "tensorflow"   # Keras 3 also runs on "jax" or
"torch": change it here
os.environ.setdefault("TF_CPP_MIN_LOG_LEVEL", "3")
import numpy as np, matplotlib.pyplot as plt
import tensorflow as tf, keras
keras.utils.set_random_seed(0)
print("TensorFlow", tf.__version__, "| Keras", keras.__version__, "| backend:",
keras.backend.backend(),
      "| GPU:", bool(tf.config.list_physical_devices("GPU")))
```

0. Data

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
d = load_digits()
X, Xt_, y, yt_ = train_test_split((d.data / 16.0).astype("float32"), d.target,
test_size=0.4, random_state=0)
Xd, Xte, yd, yte = train_test_split(Xt_, yt_, test_size=0.5, random_state=0)
LOSS = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
def mlp(width=256, depth=3, dropout=0.0, l2=0.0, bn=False):
    layers = [keras.Input((64,))]
    for _ in range(depth):
        layers.append(keras.layers.Dense(width,
kernel_regularizer=keras.regularizers.L2(l2) if l2 else None))
        if bn: layers.append(keras.layers.BatchNormalization())
        layers += [keras.layers.ReLU(), keras.layers.Dropout(dropout)]
    return keras.Sequential(layers + [keras.layers.Dense(10)])
```

1. Regularization on 150 examples: L2 and dropout

```
for name, kw in (("nothing", {}), ("L2 1e-3", {"l2": 1e-3}), ("dropout 0.5",
{"dropout": 0.5})):
    keras.utils.set_random_seed(0); m = mlp(512, **kw)
    m.compile(keras.optimizers.Adam(1e-3), LOSS, metrics=["accuracy"])
    h = m.fit(X[:150], y[:150], validation_data=(Xd, yd), epochs=150,
batch_size=50, verbose=0)
    plt.plot(h.history["val_loss"], label=f"{name}: dev acc
{h.history['val_accuracy'][-1]:.3f}")
plt.legend(); plt.ylim(0, 1.5); plt.title("dev loss"); plt.xlabel("epoch");
plt.show()
```

2. Optimizers

```
opts = {"SGD": lambda: keras.optimizers.SGD(0.05), "momentum": lambda:
keras.optimizers.SGD(0.05, momentum=0.9),
        "RMSprop": lambda: keras.optimizers.RMSprop(1e-3), "Adam": lambda:
keras.optimizers.Adam(1e-3),
        "AdamW": lambda: keras.optimizers.AdamW(1e-3, weight_decay=0.01)}
for name, make in opts.items():
    keras.utils.set_random_seed(0); m = mlp(); m.compile(make(), LOSS, metrics=
["accuracy"])
    h = m.fit(X, y, epochs=25, batch_size=64, verbose=0)
    plt.plot(h.history["loss"], label=f"{name}: test acc {m.evaluate(Xte, yte,
verbose=0)[1]:.3f}")
plt.yscale("log"); plt.legend(); plt.title("training loss"); plt.xlabel("epoch");
plt.show()
```

3. A schedule, batch norm and early stopping together

```
steps = 60 * int(np.ceil(len(X) / 64))
keras.utils.set_random_seed(0); m = mlp(256, 3, dropout=0.3, bn=True)
m.compile(keras.optimizers.AdamW(keras.optimizers.schedules.CosineDecay(1e-4,
steps, warmup_target=3e-3, warmup_steps=steps // 20),
                                 weight_decay=1e-2), LOSS, metrics=["accuracy"])
h = m.fit(X, y, validation_data=(Xd, yd), epochs=60, batch_size=64, verbose=0,
          callbacks=[keras.callbacks.EarlyStopping(patience=8,
restore_best_weights=True)])
print(f"stopped after {len(h.history['loss'])} epochs | test accuracy
{m.evaluate(Xte, yte, verbose=0)[1]:.3f}")
```

4. tf.data: the input pipeline

Shuffle, batch and prefetch so the GPU never waits for data.

```
ds = tf.data.Dataset.from_tensor_slices((X,
y)).shuffle(1000).batch(64).prefetch(tf.data.AUTOTUNE)
xb, yb = next(iter(ds)); print("one batch:", xb.shape, yb.shape)
```

5. Mixed precision on a GPU

In Colab with a GPU: `keras.mixed_precision.set_global_policy("mixed_float16")` before building the model. Keep the last layer in float32.

```
if tf.config.list_physical_devices("GPU"):
    keras.mixed_precision.set_global_policy("mixed_float16")
print("policy:", keras.mixed_precision.global_policy().name)
```