mlp

January 19, 2024

1 Multilayer perceptron (MLP)

In this notebook, we'll walk through the steps required to train your own multilayer perceptron on the CIFAR dataset

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras import layers, models, optimizers, utils, datasets
from utils import display
```

1.1 0. Parameters

[ ]:
```
NUM_CLASSES = 10
```

1.2 1. Prepare the Data

[ ]:
```
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
```

[ ]:
```
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

y_train = utils.to_categorical(y_train, NUM_CLASSES)
y_test = utils.to_categorical(y_test, NUM_CLASSES)
```

[ ]:
```
display(x_train[:10])
print(y_train[:10])
```

```
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
```

1.3 2. Build the model

[ ]:

```python
input_layer = layers.Input((32, 32, 3))

x = layers.Flatten()(input_layer)
x = layers.Dense(200, activation="relu")(x)
x = layers.Dense(150, activation="relu")(x)

output_layer = layers.Dense(NUM_CLASSES, activation="softmax")(x)

model = models.Model(input_layer, output_layer)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0


_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 flatten (Flatten)           (None, 3072)              0

 dense (Dense)               (None, 200)               614600

 dense_1 (Dense)             (None, 150)               30150

 dense_2 (Dense)             (None, 10)                1510

=================================================================
Total params: 646260 (2.47 MB)
Trainable params: 646260 (2.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1.4 3. Train the model

[ ]:

```
opt = optimizers.Adam(learning_rate=0.0005)
model.compile(
    loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"]
)
```

[ ]:

```
model.fit(x_train, y_train, batch_size=32, epochs=10, shuffle=True)
```

```
Epoch 1/10
1563/1563 [==============================] - 18s 11ms/step - loss: 1.8566 -
accuracy: 0.3329
Epoch 2/10
1563/1563 [==============================] - 16s 10ms/step - loss: 1.6687 -
accuracy: 0.4014
Epoch 3/10
1563/1563 [==============================] - 17s 11ms/step - loss: 1.5935 -
accuracy: 0.4334
Epoch 4/10
1563/1563 [==============================] - 28s 18ms/step - loss: 1.5374 -
accuracy: 0.4493
Epoch 5/10
1563/1563 [==============================] - 20s 13ms/step - loss: 1.4979 -
accuracy: 0.4672
Epoch 6/10
1563/1563 [==============================] - 23s 15ms/step - loss: 1.4604 -
accuracy: 0.4801
Epoch 7/10
1563/1563 [==============================] - 19s 12ms/step - loss: 1.4343 -
accuracy: 0.4886
Epoch 8/10
1563/1563 [==============================] - 19s 12ms/step - loss: 1.4114 -
accuracy: 0.4987
Epoch 9/10
1563/1563 [==============================] - 19s 12ms/step - loss: 1.3876 -
accuracy: 0.5060
Epoch 10/10
1563/1563 [==============================] - 18s 12ms/step - loss: 1.3698 -
accuracy: 0.5132
```

```
[ ]: <keras.src.callbacks.History at 0x2c0806af2b0>
```

1.5 4. Evaluation

[ ]:

```
model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 1.4829 -
accuracy: 0.4732
```

[ ]:

```
[1.4829447269439697, 0.4731999933719635]
```

[ ]:

```
CLASSES = np.array(
    [
        "airplane",
        "automobile",
        "bird",
        "cat",
        "deer",
        "dog",
        "frog",
        "horse",
        "ship",
        "truck",
    ]
)

preds = model.predict(x_test)
preds_single = CLASSES[np.argmax(preds, axis=-1)]
actual_single = CLASSES[np.argmax(y_test, axis=-1)]
```

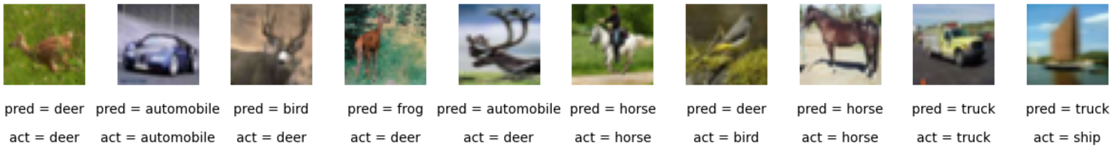
```
313/313 [==============================] - 1s 2ms/step
```

[ ]:

```
n_to_show = 10
indices = np.random.choice(range(len(x_test)), n_to_show)

fig = plt.figure(figsize=(15, 3))
fig.subplots_adjust(hspace=0.4, wspace=0.4)

for i, idx in enumerate(indices):
    img = x_test[idx]
    ax = fig.add_subplot(1, n_to_show, i + 1)
    ax.axis("off")
    ax.text(
        0.5,
        -0.35,
        "pred = " + str(preds_single[idx]),
        fontsize=10,
        ha="center",
        transform=ax.transAxes,
    )
    ax.text(
        0.5,
        -0.7,
        "act = " + str(actual_single[idx]),
        fontsize=10,
        ha="center",
        transform=ax.transAxes,
    )
    ax.imshow(img)
```



pred = deer act = deer

pred = automobile act = automobile

pred = bird act = deer

pred = frog act = deer

pred = automobile act = deer

pred = horse act = horse

pred = deer act = bird

pred = horse act = horse

pred = truck act = truck

pred = truck act = ship