nn

March 16, 2023

1 Neural Networks

1.1 Introduction

popular

1.1.1 Setup

neurons w/ weights w (+ biases b) and nonlinearity/activation ϕ

$\phi(\sum_i x_i w_i + b_i)$

In layers w/ weights $W \in \mathbb{R}^{n_l \times n_{l+1}}$ and biases $b_l \in \mathbb{R}^{n_k}$ w/ n_l neurons in layer l:

$$\phi(W_l x_l + b_l)$$

(abuse of notation w/ ϕ)

(if input points are $x \in \mathbb{R}^d$, then $n_l = d$)

Do this for all layers to get some output values in your final layer (*forward pass*)

set initial weights W_l randomly

Tons of different shapes/types of NNs

split data into train and test (80/20ish is good)

1.1.2 Backpropagation

Loss $L(y)$ is a function of the output y and the target t, e.g.:

$$L(y) = (t - j)^2$$

Calculate derivative wrt each weight $D_n = \frac{\partial L(y)}{\partial w_n}$ and use gradient descent to update weights:

$$w_n \leftarrow w_n - \eta D_n$$

for learning rate $\eta > 0$

[ ]:
```
from sklearn.datasets import load_iris
X, y = load_iris(as_frame = True, return_X_y=True)
```

[ ]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```python
import tensorflow as tf
train = tf.data.Dataset.from_tensor_slices((X_train, y_train))
test = tf.data.Dataset.from_tensor_slices((X_test, y_test))
```

[ ]:
```python
train = train.repeat(20).shuffle(1000).batch(32)
test = test.batch(1)
```

[ ]:
```python
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation=tf.nn.relu),   # hidden layer
    # tf.keras.layers.Dense(10, activation=tf.nn.relu),   # hidden layer
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(3, activation=tf.nn.softmax)  # output layer
])

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

model.fit(
    train,
    validation_data=test,
    epochs=10,
)
```

```
Epoch 1/10
75/75 [==============================] - 1s 3ms/step - loss: 1.2829 - accuracy:
0.3292 - val_loss: 1.1317 - val_accuracy: 0.3667
Epoch 2/10
75/75 [==============================] - 0s 2ms/step - loss: 1.1326 - accuracy:
0.2912 - val_loss: 1.0770 - val_accuracy: 0.3333
Epoch 3/10
75/75 [==============================] - 0s 2ms/step - loss: 1.0597 - accuracy:
0.3508 - val_loss: 1.0327 - val_accuracy: 0.4333
Epoch 4/10
75/75 [==============================] - 0s 2ms/step - loss: 1.0013 - accuracy:
0.5467 - val_loss: 0.9714 - val_accuracy: 0.6333
Epoch 5/10
75/75 [==============================] - 0s 2ms/step - loss: 0.9382 - accuracy:
0.6058 - val_loss: 0.9125 - val_accuracy: 0.6333
Epoch 6/10
75/75 [==============================] - 0s 2ms/step - loss: 0.8886 - accuracy:
0.6029 - val_loss: 0.8520 - val_accuracy: 0.6333
Epoch 7/10
```

```
75/75 [==============================] - 0s 3ms/step - loss: 0.8368 - accuracy:
0.6071 - val_loss: 0.7963 - val_accuracy: 0.6333
Epoch 8/10
75/75 [==============================] - 0s 2ms/step - loss: 0.8034 - accuracy:
0.6075 - val_loss: 0.7507 - val_accuracy: 0.6333
Epoch 9/10
75/75 [==============================] - 0s 2ms/step - loss: 0.7824 - accuracy:
0.6029 - val_loss: 0.7095 - val_accuracy: 0.6333
Epoch 10/10
75/75 [==============================] - 0s 3ms/step - loss: 0.7626 - accuracy:
0.6017 - val_loss: 0.6747 - val_accuracy: 0.6333
```

```
[ ]: <keras.callbacks.History at 0x7f00181a6740>
```

```
[ ]: for pred_dict, expected in zip(predictions, ["setosa", "versicolor",␣
     ↪"virginica"]):
         predicted_index = pred_dict.argmax()
         predicted = load_iris().target_names[predicted_index]
         probability = pred_dict.max()
         tick_cross = " " if predicted == expected else " "
         print(f"{tick_cross} Prediction is '{predicted}' ({100 * probability:.
     ↪1f}%), expected '{expected}'")
```

```
Prediction is 'setosa' (100.0%), expected 'setosa'
Prediction is 'versicolor' (99.8%), expected 'versicolor'
Prediction is 'virginica' (91.9%), expected 'virginica'
```

1.2 Convolutional Neural Networks (CNNs)

1.2.1 Image Kernel Convolutions

images are matrices of pixel values use kernel to convolve over image to get new image (using padding at edges maybe so output image is same size as input image—e.g. zero padding (add a border of zeros) or mirror padding (add a border of identical pixels to the edge pixels))

e.g. for kernel w and image w/ pixel coords $f(x, y)$ we get pixel value $g(x, y)$ where:

$$g(x, y) = w * f(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx, dy) f(x - dx, y - dy)$$

e.g. for a 3x3 kernel:

$$w = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

(^^directional edge detection kernel (I think?))

1.2.2 CNNs

Convolutional Layers We'll make an NN learn the convolution kernels for us! (i.e. learn the weights $w_{x,y}(dx, dy)$—i.e. the weights of the kernel depend on the pixels being convolved over.) And we can stack these layers to get more complex kernels.

Pooling Layers We can also use pooling layers to reduce the size of the image (e.g. max pooling). These just take a window of pixels and output the max value (or average value or something), meaning we can reduce the size of the image without losing too much information (downsampling).

Fully Connected Layers (FC/Dense Layers) Fully connected layers are just like the ones we've seen before (i.e. in non-convolution-land), but we flatten the image first (i.e. we take the image and turn it into a vector of pixel values).

[ ]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(
        filters=16,
        kernel_size=5,
        padding="same",
        activation=tf.nn.relu
    ),
    tf.keras.layers.MaxPool2D((2, 2), (2, 2), padding="same"),
    tf.keras.layers.Conv2D(
        filters=32,
        kernel_size=5,
        padding="same",
        activation=tf.nn.relu
    ),
    tf.keras.layers.MaxPool2D((2, 2), (2, 2), padding="same"),
    tf.keras.layers.Conv2D(
        filters=64,
        kernel_size=5,
        padding="same",
        activation=tf.nn.relu
    ),
    tf.keras.layers.MaxPool2D((2, 2), (2, 2), padding="same"),
    tf.keras.layers.Conv2D(
        filters=128,
        kernel_size=5,
        padding="same",
        activation=tf.nn.relu
    ),
    tf.keras.layers.MaxPool2D((2, 2), (2, 2), padding="same"),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

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

[ ]:
```
import tensorflow_datasets as tfds

ds_train, ds_test = tfds.load(
    "mnist",
    split=["train", "test"],
    as_supervised=True,
)
```

[ ]:
```
ds_train.element_spec
```

[ ]:
```
(TensorSpec(shape=(28, 28, 1), dtype=tf.uint8, name=None),
 TensorSpec(shape=(), dtype=tf.int64, name=None))
```

[ ]:
```
def normalize_img(image, label):
    return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(normalize_img)

ds_train = ds_train.shuffle(1000)
ds_train = ds_train.batch(128)

ds_test = ds_test.map(normalize_img)
ds_test = ds_test.batch(128)
```

[ ]:
```
model.fit(
    ds_train,
    validation_data=ds_test,
    epochs=20,
)
```

```
Epoch 1/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0046 -
accuracy: 0.9987 - val_loss: 0.0366 - val_accuracy: 0.9935
Epoch 2/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0040 -
accuracy: 0.9988 - val_loss: 0.0478 - val_accuracy: 0.9932
Epoch 3/20
469/469 [==============================] - 20s 42ms/step - loss: 0.0031 -
accuracy: 0.9992 - val_loss: 0.0496 - val_accuracy: 0.9928
Epoch 4/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0036 -
accuracy: 0.9990 - val_loss: 0.0611 - val_accuracy: 0.9921
```

```
Epoch 5/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0035 -
accuracy: 0.9990 - val_loss: 0.0441 - val_accuracy: 0.9925
Epoch 6/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0036 -
accuracy: 0.9990 - val_loss: 0.0510 - val_accuracy: 0.9925
Epoch 7/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0023 -
accuracy: 0.9994 - val_loss: 0.0825 - val_accuracy: 0.9900
Epoch 8/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0023 -
accuracy: 0.9994 - val_loss: 0.0604 - val_accuracy: 0.9918
Epoch 9/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0034 -
accuracy: 0.9993 - val_loss: 0.0536 - val_accuracy: 0.9928
Epoch 10/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0023 -
accuracy: 0.9995 - val_loss: 0.0600 - val_accuracy: 0.9919
Epoch 11/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0022 -
accuracy: 0.9995 - val_loss: 0.0515 - val_accuracy: 0.9928
Epoch 12/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0030 -
accuracy: 0.9993 - val_loss: 0.0688 - val_accuracy: 0.9926
Epoch 13/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0019 -
accuracy: 0.9995 - val_loss: 0.0709 - val_accuracy: 0.9923
Epoch 14/20
469/469 [==============================] - 19s 40ms/step - loss: 0.0021 -
accuracy: 0.9995 - val_loss: 0.0554 - val_accuracy: 0.9933
Epoch 15/20
469/469 [==============================] - 19s 40ms/step - loss: 0.0027 -
accuracy: 0.9994 - val_loss: 0.0607 - val_accuracy: 0.9932
Epoch 16/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0020 -
accuracy: 0.9995 - val_loss: 0.0698 - val_accuracy: 0.9920
Epoch 17/20
469/469 [==============================] - 19s 40ms/step - loss: 0.0018 -
accuracy: 0.9995 - val_loss: 0.0553 - val_accuracy: 0.9943
Epoch 18/20
469/469 [==============================] - 19s 40ms/step - loss: 0.0019 -
accuracy: 0.9995 - val_loss: 0.0640 - val_accuracy: 0.9934
Epoch 19/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0013 -
accuracy: 0.9997 - val_loss: 0.0624 - val_accuracy: 0.9926
Epoch 20/20
469/469 [==============================] - 19s 41ms/step - loss: 0.0022 -
accuracy: 0.9996 - val_loss: 0.0649 - val_accuracy: 0.9923
```

[ ]: <keras.callbacks.History at 0x7eff9b45e920>

[ ]:
```
from urllib.request import urlretrieve

for i in list(range(1,10)) + ["dog"]:
    urlretrieve(f"https://github.com/milliams/intro_deep_learning/raw/master/
↪{i}.png", f"{i}.png")
```

[ ]:
```
import numpy as np
from skimage.io import imread

images = []
for i in list(range(1,10)) + ["dog"]:
    images.append(np.array(imread(f"{i}.png")/255.0, dtype="float32"))
images = np.array(images)[:,:,:,np.newaxis]
images.shape
```

[ ]: (10, 28, 28, 1)

[ ]:
```
probabilities = model.predict(images)
```

```
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 32ms/step
```

[ ]:
```
truths = list(range(1, 10)) + ["dog"]

table = []
for truth, probs in zip(truths, probabilities):
    prediction = probs.argmax()
    if truth == 'dog':
        print(f"{truth}. CNN thinks it's a {prediction} ({probs[prediction]*100:
↪.1f}%)")
    else:
        print(f"{truth} at {probs[truth]*100:4.1f}%. CNN thinks it's a␣
↪{prediction} ({probs[prediction]*100:4.1f}%)")
    table.append((truth, probs))
```

```
1 at 51.3%. CNN thinks it's a 1 (51.3%)
2 at 84.8%. CNN thinks it's a 2 (84.8%)
3 at 91.2%. CNN thinks it's a 3 (91.2%)
4 at  0.1%. CNN thinks it's a 5 (41.7%)
5 at 100.0%. CNN thinks it's a 5 (100.0%)
6 at  0.0%. CNN thinks it's a 3 (99.9%)
7 at 99.7%. CNN thinks it's a 7 (99.7%)
8 at  4.7%. CNN thinks it's a 1 (21.8%)
9 at 15.2%. CNN thinks it's a 8 (64.3%)
dog. CNN thinks it's a 8 (17.0%)
```

1.2.3 Data Augmentation

add inveted images to training data to make the NN more robust to different images (could also do rotated images, &c.)

[ ]:
```python
def invert_img(image, label):
    return 1.-image, label


ds_train = ds_train.map(normalize_img)
ds_train = ds_train.concatenate(ds_train.map(invert_img))  # new line
ds_train = ds_train.shuffle(1000)
ds_train = ds_train.batch(128)

ds_test = ds_test.map(normalize_img)
ds_test = ds_test.concatenate(ds_test.map(invert_img))  # new line
ds_test = ds_test.batch(128)

model.fit(
    ds_train,
    validation_data=ds_test,
    epochs=2,
)
```

```
Epoch 1/2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[169], line 14
     11 ds_test = ds_test.concatenate(ds_test.map(invert_img))  # new line
     12 ds_test = ds_test.batch(128)
---> 14 model.fit(
     15     ds_train,
     16     validation_data=ds_test,
     17     epochs=2,
     18 )

File ~/Documents/Compass/SC2/lec7/.conda/lib/python3.10/site-packages/keras/
 ↪utils/traceback_utils.py:70, in filter_traceback.<locals>.error_handler(*args␣
 ↪**kwargs)
     67     filtered_tb = _process_traceback_frames(e.__traceback__)
     68     # To get the full stack trace, call:
     69     # `tf.debugging.disable_traceback_filtering()`
---> 70     raise e.with_traceback(filtered_tb) from None
     71 finally:
     72     del filtered_tb

File /tmp/__autograph_generated_filec1w5pftg.py:15, in outer_factory.<locals>.
 ↪inner_factory.<locals>.tf__train_function(iterator)
```

```
     13 try:
     14     do_return = True
---> 15     retval_ = ag__.converted_call(ag__.ld(step_function), (ag__.
↪ld(self), ag__.ld(iterator)), None, fscope)
     16 except:
     17     do_return = False

ValueError: in user code:

    File "/home/dg22309/Documents/Compass/SC2/lec7/.conda/lib/python3.10/
↪site-packages/keras/engine/training.py", line 1249, in train_function  *
        return step_function(self, iterator)
    File "/home/dg22309/Documents/Compass/SC2/lec7/.conda/lib/python3.10/
↪site-packages/keras/engine/training.py", line 1233, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/home/dg22309/Documents/Compass/SC2/lec7/.conda/lib/python3.10/
↪site-packages/keras/engine/training.py", line 1222, in run_step  **
        outputs = model.train_step(data)
    File "/home/dg22309/Documents/Compass/SC2/lec7/.conda/lib/python3.10/
↪site-packages/keras/engine/training.py", line 1023, in train_step
        y_pred = self(x, training=True)
    File "/home/dg22309/Documents/Compass/SC2/lec7/.conda/lib/python3.10/
↪site-packages/keras/utils/traceback_utils.py", line 70, in error_handler
        raise e.with_traceback(filtered_tb) from None

    ValueError: Exception encountered when calling layer 'sequential_24' (type␣
↪Sequential).

    Cannot iterate over a shape with unknown rank.

    Call arguments received by layer 'sequential_24' (type Sequential):
      • inputs=tf.Tensor(shape=<unknown>, dtype=float32)
      • training=True
      • mask=None
```

[ ]:
```
probabilities = model.predict(images)
```

[ ]:
```
truths = list(range(1, 10)) + ["dog"]

table = []
for truth, probs in zip(truths, probabilities):
    prediction = probs.argmax()
    if truth == 'dog':
        print(f"{truth}. CNN thinks it's a {prediction} ({probs[prediction]*100:
↪.1f}%)")
```

```
    else:
        print(f"{truth} at {probs[truth]*100:4.1f}%. CNN thinks it's a␣
↪{prediction} ({probs[prediction]*100:4.1f}%)")
    table.append((truth, probs))
```