ANSWERS

8.4 A tiny network

Learning · Robot club · about 30 min

What this lesson is about

Weights, a loss, and watching it fall: training in the browser.

- Numbers in, numbers out
- Untrained
- Training
- Why the centring matters

Questions

7 marks in all

1. What does a trained network keep, so the data can be thrown away? [1 mark]

- ☐ **A** Its weights
- ☐ **B** Every sample it was shown
- ☐ **C** Just the list of labels
- ☐ **D** The loss from the last epoch

Answer: A. Nearest neighbour keeps the data; a network keeps the numbers it learned from the data.

2. What does this program print? [1 mark]

```
labels = sorted(["open", "wall-ahead", "gap-left", "gap-right"])
row = [0] * len(labels)
row[labels.index("open")] = 1
print(labels)
print(row)
```

Answer:

```
['gap-left', 'gap-right', 'open', 'wall-ahead']
[0, 0, 1, 0]
```

Sorted alphabetically, open is third, so its one-hot row has a 1 in position 2 and 0 everywhere else.

3. What does this program print? [1 mark]

```
for reading in [400, 30, 10]:
    print((reading - 30) / 20)
```

Answer:

```
18.5
0.0
-1.0
```

Subtract 30 and divide by 20: 30 becomes 0 and 10 becomes -1, but the out-of-range 400 still becomes 18.5.

4. What does this program print? [1 mark]

```
inputs = [1.0, -0.5]
weights = [2.0, 4.0]
bias = 0.5
total = sum(i * w for i, w in zip(inputs, weights)) + bias
print(total)
```

Answer:

```
0.5
```

Multiply and add: 1.0 x 2.0 + (-0.5) x 4.0 + 0.5 = 2 - 2 + 0.5 = 0.5. A network does this, then squashes, then repeats.

5. Put one epoch of training in order. [1 mark]

Number the lines 1 to 4 to put them in the right order.

___ `Step: move every weight a little`

___ `Measure how wrong they are: the loss`

___ `Backward: work out the gradients`

___ `Forward: work out the outputs from the inputs`

Answer:

```
Forward: work out the outputs from the inputs
Measure how wrong they are: the loss
Backward: work out the gradients
Step: move every weight a little
```

Forward, measure, backward, step, then repeat for the next epoch.

6. The learning rate `lr` is set far too big. What do you see? [1 mark]

- ☐ **A** The loss bounces about and never settles
- ☐ **B** The loss creeps down very slowly
- ☐ **C** The loss drops to zero at once
- ☐ **D** Training is skipped

Answer: A. Too big overshoots every step; too small creeps. It is the gain from Module 3 again.

7. Scaling the inputs as `X / 100` instead of `(X - 30) / 20` leaves the network stuck at about 88%. Why? [1 mark]

- ☐ **A** The inputs are no longer centred near zero, which gives the network worse numbers to work with
- ☐ **B** Dividing by 100 deletes some samples
- ☐ **C** The labels stop being one-hot
- ☐ **D** The network needs more hidden numbers

Answer: A. Same network, same data, worse numbers. Getting inputs into a sensible range is not a detail.

The task: a tiny network

Train the network on `DATA`, printing `epoch <n>: loss <value>` every 50 epochs, then print `accuracy: <n>%` and get it above 90. Do not drive.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()
DATA = [
    ([54, 52, 51, 51, 51, 51, 52, 54, 54, 52, 51, 51, 51, 51, 52, 54], 'open'),
    ([16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16], 'wall-ahead'),
    ([36, 47, 46, 46, 46, 16, 16, 16, 36, 47, 46, 46, 46, 16, 16, 16], 'gap-left'),
    ([16, 16, 16, 46, 46, 46, 47, 36, 16, 16, 16, 46, 46, 46, 47, 36], 'gap-right'),
    ([400, 400, 61, 61, 61, 61, 400, 400, 96, 62, 61, 61, 61, 61, 62, 64], 'open'),
    ([52, 50, 49, 49, 49, 49, 50, 52, 51, 50, 49, 49, 49, 49, 50, 51], 'open'),
    ([39, 38, 37, 37, 37, 37, 38, 39, 39, 38, 37, 37, 37, 37, 38, 39], 'open'),
    ([400, 400, 61, 61, 61, 61, 400, 400, 64, 62, 61, 61, 61, 61, 62, 64], 'open'),
    ([52, 50, 49, 49, 49, 49, 50, 52, 51, 50, 49, 49, 49, 49, 50, 51], 'open'),
    ([39, 38, 37, 37, 37, 37, 38, 39, 39, 38, 37, 37, 37, 37, 38, 39], 'open'),
    ([400, 400, 61, 61, 61, 61, 400, 400, 64, 62, 61, 61, 61, 61, 62, 96], 'open'),
    ([52, 50, 49, 49, 49, 49, 50, 52, 51, 50, 49, 49, 49, 49, 50, 51], 'open'),
    ([39, 38, 37, 37, 37, 37, 38, 39, 39, 38, 37, 37, 37, 37, 38, 39], 'open'),
    ([26, 25, 25, 25, 25, 25, 25, 26, 26, 25, 25, 25, 25, 25, 25, 26], 'wall-ahead'),
    ([18, 17, 17, 17, 17, 17, 17, 18, 17, 17, 17, 17, 17, 17, 17, 17], 'wall-ahead'),
    ([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 'wall-ahead'),
    ([26, 25, 25, 25, 25, 25, 25, 26, 26, 25, 25, 25, 25, 25, 25, 26], 'wall-ahead'),
    ([18, 17, 17, 17, 17, 17, 17, 18, 17, 17, 17, 17, 17, 17, 17, 17], 'wall-ahead'),
    ([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 'wall-ahead'),
    ([26, 25, 25, 25, 25, 25, 25, 26, 26, 25, 25, 25, 25, 25, 25, 26], 'wall-ahead'),
    ([18, 17, 17, 17, 17, 17, 17, 18, 17, 17, 17, 17, 17, 17, 17, 17], 'wall-ahead'),
    ([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 'wall-ahead'),
    ([24, 34, 55, 59, 59, 59, 29, 30, 24, 33, 55, 59, 59, 59, 29, 30], 'gap-left'),
    ([24, 34, 51, 51, 51, 51, 25, 22, 24, 33, 51, 51, 51, 51, 25, 22], 'gap-left'),
    ([24, 34, 43, 43, 43, 43, 44, 18, 24, 33, 43, 43, 43, 43, 44, 18], 'gap-left'),
    ([42, 58, 59, 59, 29, 29, 29, 30, 42, 58, 59, 59, 29, 29, 29, 30], 'gap-left'),
    ([42, 52, 51, 51, 21, 21, 21, 22, 42, 52, 51, 51, 21, 21, 21, 22], 'gap-left'),
    ([42, 44, 43, 43, 13, 13, 13, 13, 42, 44, 43, 43, 13, 13, 13, 13], 'gap-left'),
    ([30, 29, 29, 29, 59, 59, 58, 42, 30, 29, 29, 29, 59, 59, 58, 42], 'gap-right'),
    ([22, 21, 21, 21, 51, 51, 52, 42, 22, 21, 21, 21, 51, 51, 52, 42], 'gap-right'),
    ([13, 13, 13, 13, 43, 43, 44, 42, 13, 13, 13, 13, 43, 43, 44, 42], 'gap-right'),
    ([30, 29, 59, 59, 59, 55, 34, 24, 30, 29, 59, 59, 59, 55, 33, 24], 'gap-right'),
    ([22, 25, 51, 51, 51, 51, 34, 24, 22, 25, 51, 51, 51, 51, 33, 24], 'gap-right'),
    ([18, 44, 43, 43, 43, 43, 34, 24, 18, 44, 43, 43, 43, 43, 33, 24], 'gap-right'),
]
import numpy as np
X = (np.array([f for f, _ in DATA], dtype=float) - 30) / 20
print('samples:', X.shape)
```

The hint students can ask for: Train the network on DATA, printing the loss every 50 epochs, then print the training accuracy. Do not drive.

A solution

```
from bugbot import *
connect()
DATA = [
    ([54, 52, 51, 51, 51, 51, 52, 54, 54, 52, 51, 51, 51, 51, 52, 54], 'open'),
```

```
    ([16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16], 'wall-ahead'),
    ([36, 47, 46, 46, 46, 16, 16, 16, 36, 47, 46, 46, 46, 16, 16, 16], 'gap-left'),
    ([16, 16, 16, 46, 46, 46, 47, 36, 16, 16, 16, 46, 46, 46, 47, 36], 'gap-right'),
    ([400, 400, 61, 61, 61, 61, 400, 400, 96, 62, 61, 61, 61, 61, 62, 64], 'open'),
    ([52, 50, 49, 49, 49, 49, 50, 52, 51, 50, 49, 49, 49, 49, 50, 51], 'open'),
    ([39, 38, 37, 37, 37, 37, 38, 39, 39, 38, 37, 37, 37, 37, 38, 39], 'open'),
    ([400, 400, 61, 61, 61, 61, 400, 400, 64, 62, 61, 61, 61, 61, 62, 64], 'open'),
    ([52, 50, 49, 49, 49, 49, 50, 52, 51, 50, 49, 49, 49, 49, 50, 51], 'open'),
    ([39, 38, 37, 37, 37, 37, 38, 39, 39, 38, 37, 37, 37, 37, 38, 39], 'open'),
    ([400, 400, 61, 61, 61, 61, 400, 400, 64, 62, 61, 61, 61, 61, 62, 96], 'open'),
    ([52, 50, 49, 49, 49, 49, 50, 52, 51, 50, 49, 49, 49, 49, 50, 51], 'open'),
    ([39, 38, 37, 37, 37, 37, 38, 39, 39, 38, 37, 37, 37, 37, 38, 39], 'open'),
    ([26, 25, 25, 25, 25, 25, 25, 26, 26, 25, 25, 25, 25, 25, 25, 26], 'wall-ahead'),
    ([18, 17, 17, 17, 17, 17, 17, 18, 17, 17, 17, 17, 17, 17, 17, 17], 'wall-ahead'),
    ([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 'wall-ahead'),
    ([26, 25, 25, 25, 25, 25, 25, 26, 26, 25, 25, 25, 25, 25, 25, 26], 'wall-ahead'),
    ([18, 17, 17, 17, 17, 17, 17, 18, 17, 17, 17, 17, 17, 17, 17, 17], 'wall-ahead'),
    ([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 'wall-ahead'),
    ([26, 25, 25, 25, 25, 25, 25, 26, 26, 25, 25, 25, 25, 25, 25, 26], 'wall-ahead'),
    ([18, 17, 17, 17, 17, 17, 17, 18, 17, 17, 17, 17, 17, 17, 17, 17], 'wall-ahead'),
    ([9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9], 'wall-ahead'),
    ([24, 34, 55, 59, 59, 59, 29, 30, 24, 33, 55, 59, 59, 59, 29, 30], 'gap-left'),
    ([24, 34, 51, 51, 51, 51, 25, 22, 24, 33, 51, 51, 51, 51, 25, 22], 'gap-left'),
    ([24, 34, 43, 43, 43, 43, 44, 18, 24, 33, 43, 43, 43, 43, 44, 18], 'gap-left'),
    ([42, 58, 59, 59, 29, 29, 29, 30, 42, 58, 59, 59, 29, 29, 29, 30], 'gap-left'),
    ([42, 52, 51, 51, 21, 21, 21, 22, 42, 52, 51, 51, 21, 21, 21, 22], 'gap-left'),
    ([42, 44, 43, 43, 13, 13, 13, 13, 42, 44, 43, 43, 13, 13, 13, 13], 'gap-left'),
    ([30, 29, 29, 29, 59, 59, 58, 42, 30, 29, 29, 29, 59, 59, 58, 42], 'gap-right'),
    ([22, 21, 21, 21, 51, 51, 52, 42, 22, 21, 21, 21, 51, 51, 52, 42], 'gap-right'),
    ([13, 13, 13, 13, 43, 43, 44, 42, 13, 13, 13, 13, 43, 43, 44, 42], 'gap-right'),
    ([30, 29, 59, 59, 59, 55, 34, 24, 30, 29, 59, 59, 59, 55, 33, 24], 'gap-right'),
    ([22, 25, 51, 51, 51, 51, 34, 24, 22, 25, 51, 51, 51, 51, 33, 24], 'gap-right'),
    ([18, 44, 43, 43, 43, 43, 34, 24, 18, 44, 43, 43, 43, 43, 33, 24], 'gap-right'),
]
import numpy as np
np.random.seed(1)
labels = sorted(set(l for _, l in DATA))
X = (np.array([f for f, _ in DATA], dtype=float) - 30) / 20      # 16 readings, centred
near 0 and about -1..+1
Y = np.zeros((len(DATA), len(labels)))                            # one-hot: a 1 in the
column of the right label
for i, (_, l) in enumerate(DATA):
    Y[i, labels.index(l)] = 1
W1 = np.random.randn(16, 8) * 0.5; b1 = np.zeros(8)              # 16 inputs -> 8 hidden
W2 = np.random.randn(8, len(labels)) * 0.5; b2 = np.zeros(len(labels))   # 8 hidden -> 4
outputs
lr = 0.5
for epoch in range(1, 301):
    H = np.tanh(X @ W1 + b1)                                      # forward
    Z = H @ W2 + b2
    P = np.exp(Z - Z.max(axis=1, keepdims=True)); P /= P.sum(axis=1, keepdims=True)   #
softmax
    loss = -np.mean(np.log(P[np.arange(len(DATA)), Y.argmax(axis=1)] + 1e-9))
    dZ = (P - Y) / len(DATA)                                      # backward
    dW2 = H.T @ dZ; db2 = dZ.sum(axis=0)
    dH = dZ @ W2.T * (1 - H ** 2)
    dW1 = X.T @ dH; db1 = dH.sum(axis=0)
    W1 -= lr * dW1; b1 -= lr * db1; W2 -= lr * dW2; b2 -= lr * db2
    if epoch % 50 == 0:
```

```
        print(f"epoch {epoch}: loss {loss:.3f}")
pred = np.tanh(X @ W1 + b1) @ W2 + b2
acc = np.mean(pred.argmax(axis=1) == Y.argmax(axis=1))
print(f"accuracy: {round(acc * 100)}%")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.