Barış Deniz Sağlam
158410

1)

a.
At around the 80th step, the validation loss stops decreasing while the training loss continues decreasing. Hence, after that step, the network starts overfitting. This inflection point in validation loss is where the training should be halted.

b. The loss graph implies an unrepresentative validation dataset. Since the validation loss is significantly lower than the training loss, it can be said that the validation dataset is easier for the network.

This can be due to various reasons. If the training and validation datasets are randomly split, then by chance, the distribution of training and validation sets might end up different. For instance, this may happen in a highly imbalanced dataset used in a classification task when the split is not stratified by class.

Another reason might be heavy data augmentation applied to the training set. If the augmentation is configured such that it distorts the data too much, then the network may fail to perform well on the training dataset, but perform relatively well on the validation dataset, in which the augmentation is not applied.

c. As both training and validation loss keep decreasing in the loss graph, the model experiences underfitting. Hence, it must be trained further, also given that it's trained only for 50 steps.
A higher learning rate may also be tried.

Barış Deniz Sağlam
158410

2)

a. $\hat{h}_t = [0.2333,\ -0.5819,\ 0.1490]$

b. $h_{<t+1>} = [0.9363,\ 0.3440,\ 0.9338]$

c. The output of reset gate : $r_t = [0.6479,\ 0.4013,\ 0.6479]$

The output of update gate: $z_t = [0.9804,\ 0.9804,\ 0.9804]$

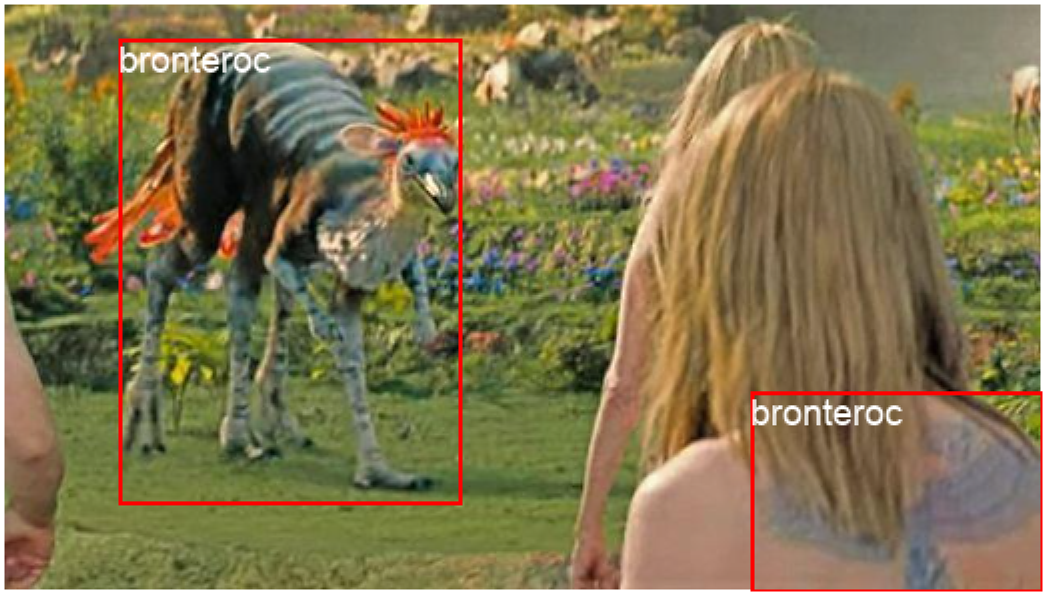
Since the output of reset gate is not close to zero, the GRU uses the long-term memory while producing the state for current time step. However, the output of update gate is close to 1; hence, the GRU doesn't pass the long-term memory inherited from the previous states and updates it with the short-term memory produced at the current time step.

The code used for calculations:

```python
import torch

def parse_weights(s):
    return [[float(i) for i in row] for row in s.split(',')]

Wr = torch.tensor(parse_weights("11000,10011,11000"))
br = torch.tensor(-1.0)
Wu = torch.tensor(parse_weights("10101,10101,10101"))
bu = torch.tensor(1.5)
Wt = torch.tensor(parse_weights("11111,00000,11111"))
bt = torch.tensor(0.4)
x = torch.tensor([+1.82, -0.21]).T
h = torch.tensor([+0.36,-1.45,+0.23]).T

rt = torch.sigmoid(Wr @ torch.cat([x,h]) + br)
zt = torch.sigmoid(Wu @ torch.cat([x,h]) + bu)
hhat = rt * h
hbar = torch.tanh(Wt @ torch.cat([x, hhat]) + bt)
hnew = (1 - zt) * h + zt * hbar
```

Barış Deniz Sağlam
158410

3)

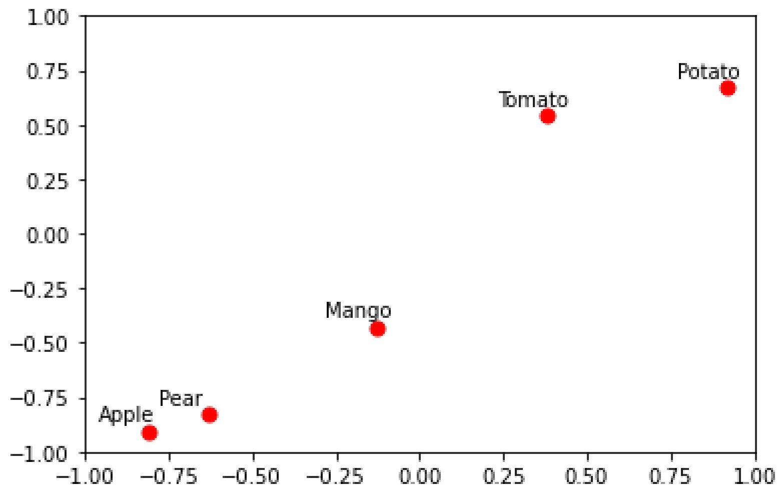


The code used to draw bounding boxes:

```python
import numpy as np
from PIL import Image, ImageDraw

def draw_bb(img, bbox, conf, labels, threshold=0.5):
    W, H = img.size
    overlay = ImageDraw.Draw(img)
    for i in range(len(conf)):
        if conf[i] <= threshold:
            continue
        x,y,w,h = bbox[i] * np.array([W, H, W, H])
        overlay.rectangle([x,y,x+w,y+h], outline ="red", fill=None,width=2)
        overlay.text((x, y), labels[i], (255,255,255))
    return img

n = 3
vocab = ["dog", "cat", "elephant", "bird", "bronteroc"]
y = [float(s) for s in "0.20 0.23 0.47 0.51 0.34 0.11 0.06 0.33 0.79 0.89
0.72 0.66 0.28 0.34 0.61 0.13 0.04 0.21 0.23 0.39".split()]
bbox_with_conf, probs = np.array(y[:15]).reshape(n, -1), np.array(y[15:])
bbox, conf = bbox_with_conf[:, :-1], bbox_with_conf[:, -1:]
label = vocab[np.argmax(probs)]
img = Image.open("./q3-image.png")
draw_bb(img, bbox, conf, [label]*n, threshold=0.6)
img
```

Barış Deniz Sağlam
158410

4)



Observing that apple, pear and mango are distant to tomato and potato, this network might be trained to classify whether a word corresponds to a vegetable or a fruit. The similarity between apple and pear; and the similarly between tomato and potato in the embedding space also support this hypothesis.