TC 5033

Deep Learning

Fully Connected Deep Neural Networks using PyTorch

José Antonio Cantoral Ceballos, Ph.D.

Team Members:

- A01200230 - Armando Bringas Corpus

Activity 2a: Implementing a FC for ASL Dataset using PyTorch

- Objective

  The primary aim of this activity is to transition from using Numpy for network implementation to utilizing PyTorch, a powerful deep learning framework. You will be replicating the work you did for the ASL dataset in Activity 1b, but this time, you'll implement a your multi layer FC model using PyTorch.

- Instructions

  Review Previous Work: Begin by reviewing your Numpy-based Fully Connected Network for the ASL dataset from Activity 1b. Note the architecture, hyperparameters, and performance metrics for comparison.

  Introduce PyTorch: If you're new to PyTorch, take some time to familiarize yourself with its basic operations and syntax. You can consult the official documentation or follow online tutorials.

Prepare the ASL Dataset: As before, download and preprocess the Kaggle ASL dataset.

Implement the Network: Design your network architecture tailored for the ASL dataset. Pay special attention to PyTorch modules like nn.Linear() and nn.ReLU().

Train the Model: Implement the training loop, making use of PyTorch's autograd to handle backpropagation. Monitor metrics like loss and accuracy as the model trains.

Analyze and Document: In Markdown cells, discuss the architecture choices, any differences in performance between the Numpy and PyTorch implementations, and insights gained from using a deep learning framework like PyTorch.

```
'2.1.0'
```

```
True
```

```
NVIDIA GeForce GTX 1650
(7, 5)
_CudaDeviceProperties(name='NVIDIA GeForce GTX 1650', major=7, minor=5, total_memory=4095MB,
multi_processor_count=14)
```

Always a good idea to explore the data

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776
0	3	107	118	127	134	139	143	146	150	153	...	207	207
1	6	155	157	156	156	156	157	156	158	158	...	69	149
2	2	187	188	188	187	187	186	187	188	187	...	202	201
3	2	211	211	212	212	211	210	211	210	210	...	235	234
4	12	164	167	170	172	176	179	180	184	185	...	92	105

5 rows × 785 columns

Get training label data

```
(27455, 784)
(27455,)
```

```
(7172, 784) (7172,)
```

```
numpy.ndarray
```

```
(3586, 784) (3586,)
(3586, 784) (3586,)
```

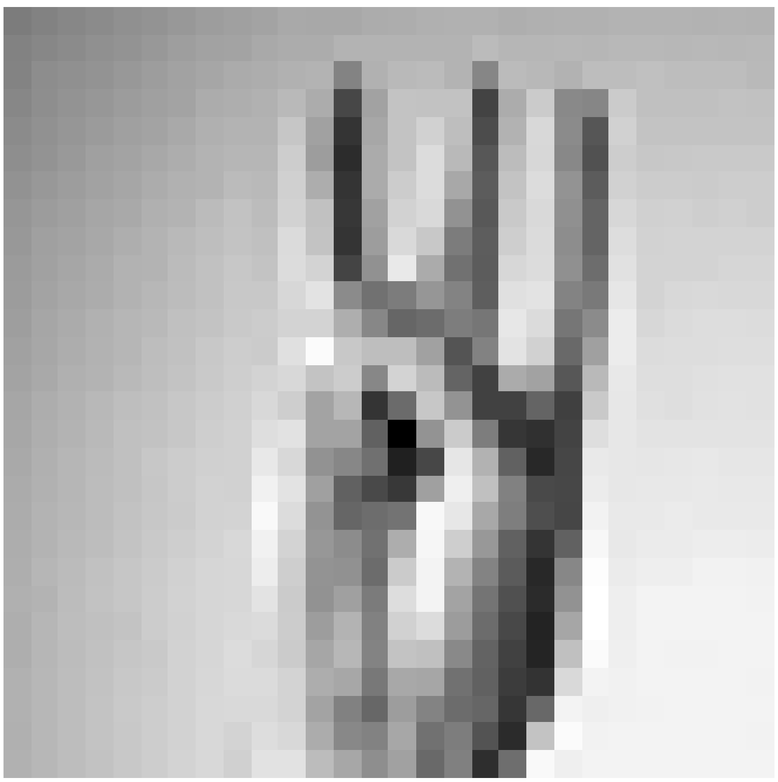
```
24
```

Normalise the data

```
(3.6268384e-06, 0.99999946)
```

```
numpy.ndarray
```

```
The sampled image represents a: w
```



The model

$$z^1 = W^1 X + b^1$$

$$a^1 = ReLU(z^1)$$

$$z^2 = W^2 a^1 + b^2$$

$$\hat{y} = \frac{e^{z^2{}_k}}{\sum_j e^{z_j}}$$

$$\mathcal{L}(\hat{y}^i, y^i) = -y^i \ln(\hat{y}^i) = -\ln(\hat{y}^i)$$

$$\mathcal{J}(w,b) = \frac{1}{num\_samples} \sum_{i=1}^{num\_samples} -\ln(\hat{y}^i)$$

Create minibatches

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54

```
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
```

Now the PyTorch part

```
cuda
```

Accuracy

Training Loop

Model using Sequential

Changing model hyperparameters to verify if accuracy can be improved

```
Epoch:0, train cost: 0.861163, val cost: 0.696276, train acc: 0.7290, val acc: 0.781372, lr: 0.004000
Epoch:20, train cost: 0.110764, val cost: 1.814120, train acc: 0.9760, val acc: 0.802844, lr: 0.004000
Epoch:40, train cost: 0.103240, val cost: 2.020840, train acc: 0.9790, val acc: 0.801729, lr: 0.004000
Epoch:60, train cost: 0.090984, val cost: 2.568904, train acc: 0.9816, val acc: 0.795594, lr: 0.004000
Epoch:80, train cost: 0.163161, val cost: 2.652196, train acc: 0.9732, val acc: 0.803681, lr: 0.004000
```

```
0.8218070273284998
```

```
The sampled image represents a: h
```

```
The predicted value is: h
```