perceptron

August 29, 2024

1 The Perceptron

Created Aug. 2024 for the FSU Course: *Machine Learning in Physics* H. B. Prosper

The perceptron, defined by

$$y = g(x\,A^{\mathrm{T}} + b), \tag{1}$$

is the basic computational element of neural networks. The output y and and the bias b are row matrices, while the input x is a matrix consisting of one or more rows.

1.1 Tips

- Use **esc r** to disable a cell
- Use **esc y** to reactivate it
- Use **esc m** to go to markdown mode. **Markdown** is the typesetting language used in jupyter notebooks.
- In a markdown cell, double tap the mouse or glide pad (on your laptop) to go to edit mode.
- Shift + return to execute a cell (including markdown cells).
- If the equations don't typeset, try double tapping the cell again, and re-execute it.

1.2 Import modules

[1]:
```
import numpy as np
import torch
```

1.3 Input Data

The input data can consist of one or more rows. We make the data type (`dtype`) `float32` to be compatible with `PyTorch`.

[2]:
```
x = np.array([-2.0, 1.0, 4.0], dtype=np.float32)
```

1.4 Perceptron Parameters

[3]:
```
# Perceptron 1
A1 = np.array([2,-3, 1], dtype=np.float32)
b1 = -5

# Perceptron 2
```

```
A2 = np.array([1, 2, 3], dtype=np.float32)
b2 = -4
```

1.4.1 Perceptron 1: $z_1 = x\,A_1^{\mathbf{T}} + b_1$; $y_1 = g(z_1)$

[4]:
```
z1 = x @ A1.T + b1
y1 = np.maximum(0, z1)
z1, y1
```

[4]:
```
(-8.0, 0.0)
```

1.4.2 Perceptron 2: $z_2 = x\,A_2^{\mathbf{T}} + b_2$; $y_2 = g(z_2)$

[5]:
```
z2 = x @ A2.T + b2
y2 = np.maximum(0, z2)
z2, y2
```

[5]:
```
(8.0, 8.0)
```

1.5 Multi-node Perceptron

Construct a 2-node perceptron with 3 inputs.

[6]:
```
A = np.vstack([A1, A2]) # vertical stack

b = np.hstack([b1, b2]) # horizontal stack

A.shape, A, b.shape, b
```

[6]:
```
((2, 3),
 array([[ 2., -3.,  1.],
        [ 1.,  2.,  3.]], dtype=float32),
 (2,),
 array([-5, -4]))
```

Compute output of perceptron

[7]:
```
z = x @ A.T + b
y = np.maximum(0, z)
z, y
```

[7]:
```
(array([-8.,  8.]), array([0., 8.]))
```

1.6 Do the same using PyTorch

[8]:
```
# instantiate a linear function with 3 inputs and 2 outputs
linear = torch.nn.Linear(3, 2)

# instantiate a relu function that will apply itelf
# elementwise to its matrix argument.
relu = torch.nn.ReLU()

# compute output of a (3, 2) perceptron
with torch.no_grad(): # disable gradient calculation
    linear.weight.copy_(torch.tensor(A))
    linear.bias.copy_(torch.tensor(b))
    z = linear(torch.tensor(x)) # compute linear function
    y = relu(z)
z, y
```

[8]:
```
(tensor([-8.,  8.]), tensor([0., 8.]))
```

[ ]:
```

```