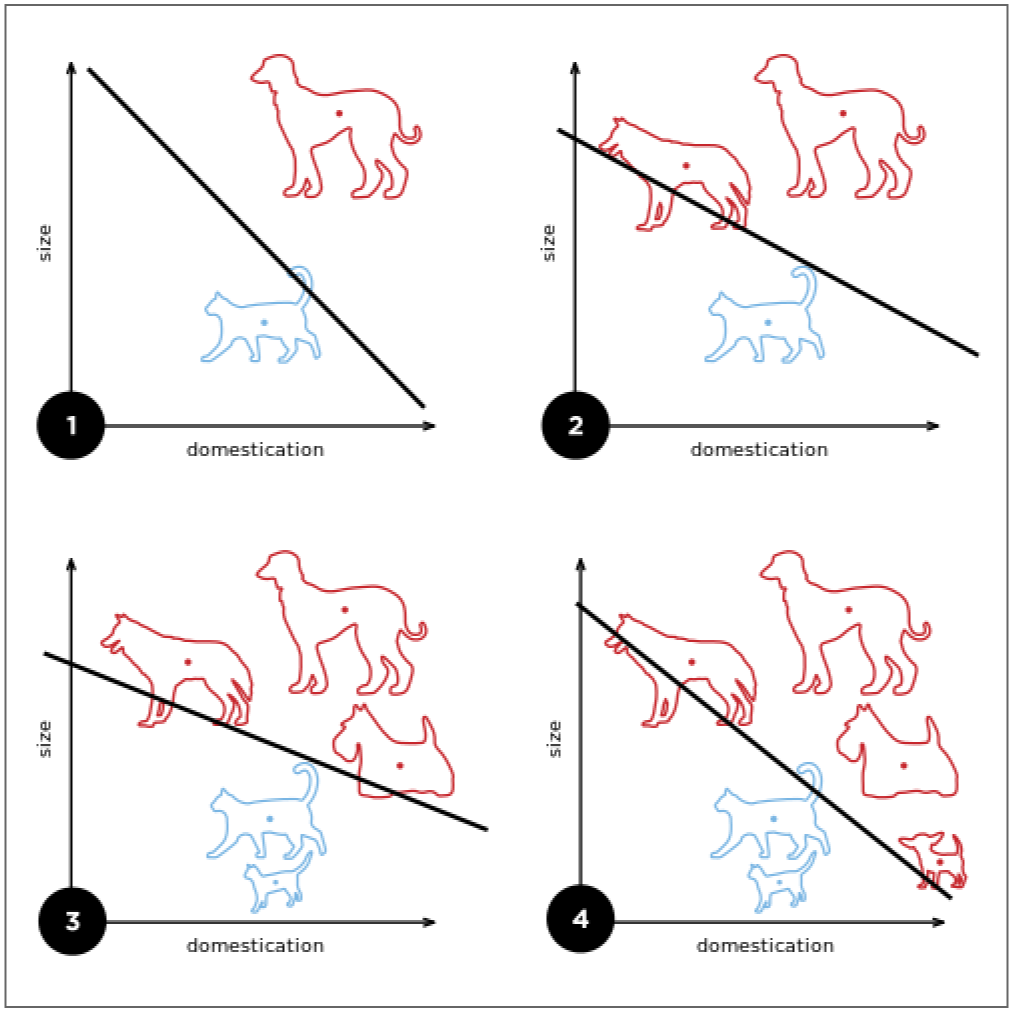
Perceptron


1
size
domestication
2
size
domestication
3
size
domestication
4
size
domestication

Second derivative

What is the role of second derivative in optimization?

Second derivative

How do the following functions differ?

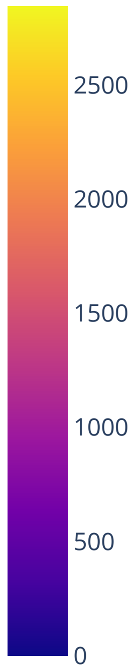
$$f(x,y) = 2x^2 + 4y^2 - xy - 6x - 8y + 6$$

$$f(x,y) = -2x^2 - 4y^2 - xy - 6x - 8y + 6$$

$$f(x,y) = 2x^2 - 4y^2 - xy - 6x - 8y + 6$$

$$f(x,y) = 2x^2 + 4y^2 - xy - 6x - 8y + 6$$

```
In [3]: def f(x, y):
            return 2*x**2 + 4*y**2 - x*y - 6*x - 8*y  + 6

        plot_surface(f)
```


2500
2000
1500
1000
500
0

$$f(x, y) = -2x^2 - 4y^2 - xy - 6x - 8y + 6$$

```
In [4]: def f(x, y): return - 2*x**2 - 4*y**2 - x*y - 6*x - 8*y  + 6
        plot_surface(f)
```

0
−500
−1000
−1500
−2000
−2500
−3000

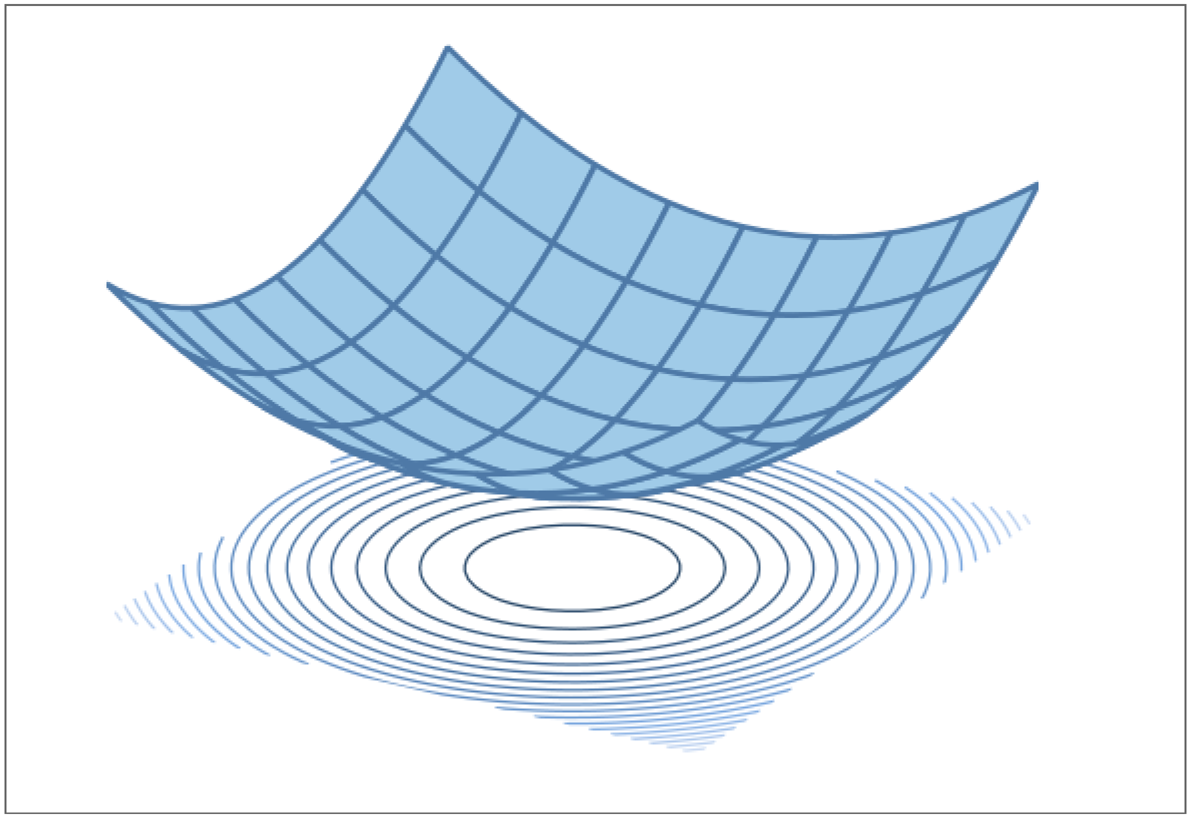
$$f(x,) = 2x^2 - 4y^2 - xy - 6x - 8y + 6$$

```
In [5]: def f(x, y): return  2*x**2 - 4*y**2 - x*y - 6*x - 8*y  + 6
        plot_surface(f)
```

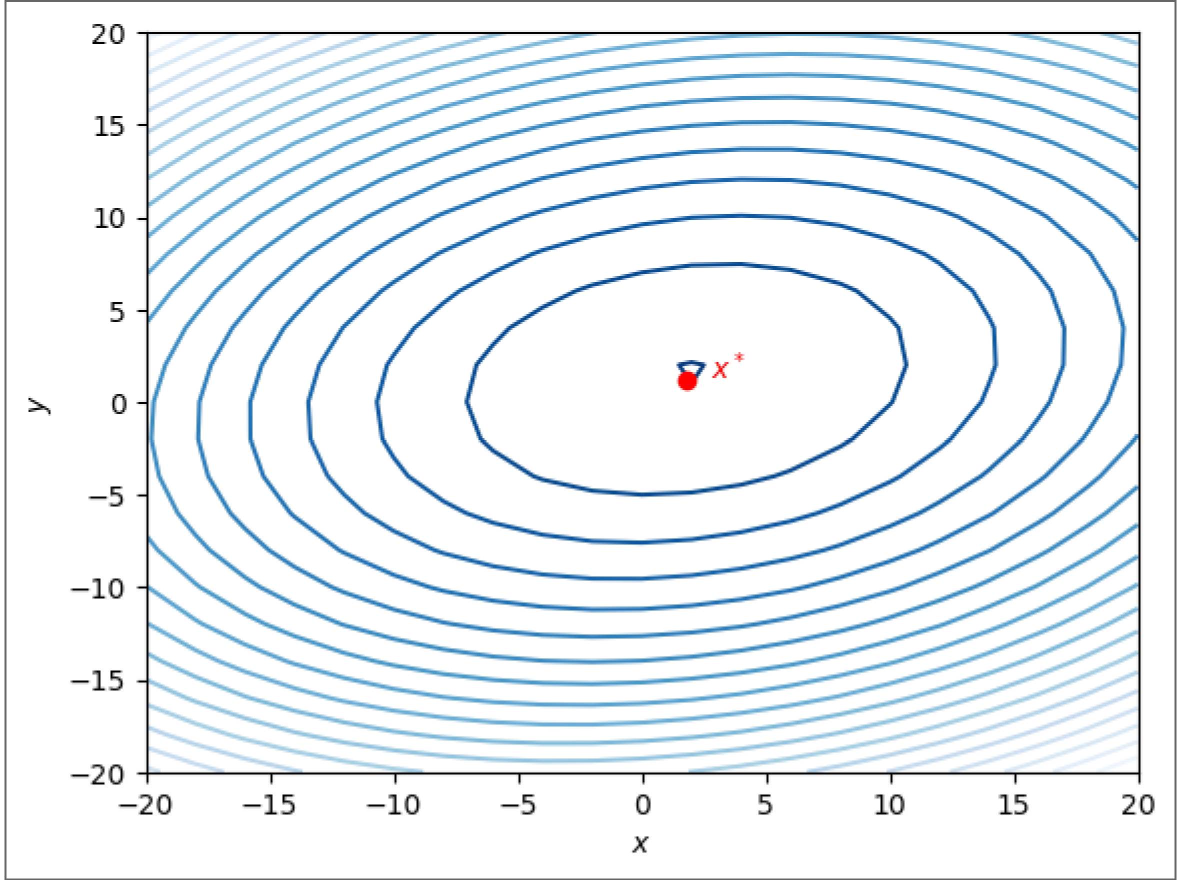
500
0
−500
−1000
−1500

Hessians

Contour Plots



```
In [7]: def f(x, y): return  2*x**2 + 4*y**2 - x*y - 6*x - 8*y  + 6
        plot_contour(f)
```

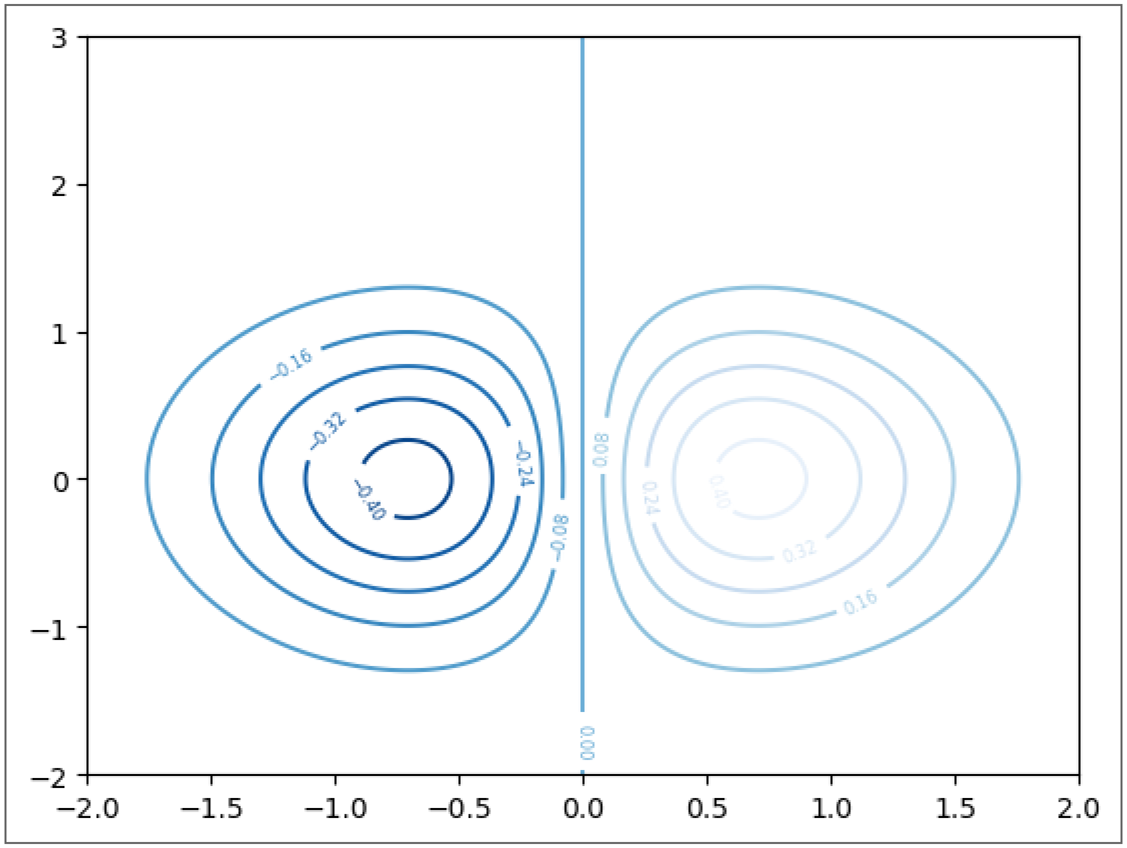


x^*
x
y
−20
−15
−10
−5
0
5
10
15
20

But how about other kinds of functions say:

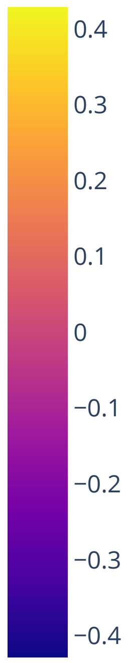
$$\arg\min_x f(x) = x\exp(-(x^2+y^2))$$

```
In [9]: def f(x,y): return  x * np.exp(-(x**2 + y**2))
        plot_contour(f)
```



```
In [10]: def plot_surface_3d(func):
             x, y = np.mgrid[-2:2:201j,
                    -2:3:201j]
    f = func(x,y)
    fig = go.Figure(data=[go.Surface(z=f, x=x, y=y,
                                     contours = {
                                         "x": {"start": -2, "end": 2, "size": 0.2},
                                         "z": {"start": -2, "end": 2, "size": 0.2}
                                     },
                                     )])
    fig.update_traces(contours_z=dict(show=True, usecolormap=True, project_z=True))
    fig.show()
```

```
In [11]: plot_surface_3d(f)
```

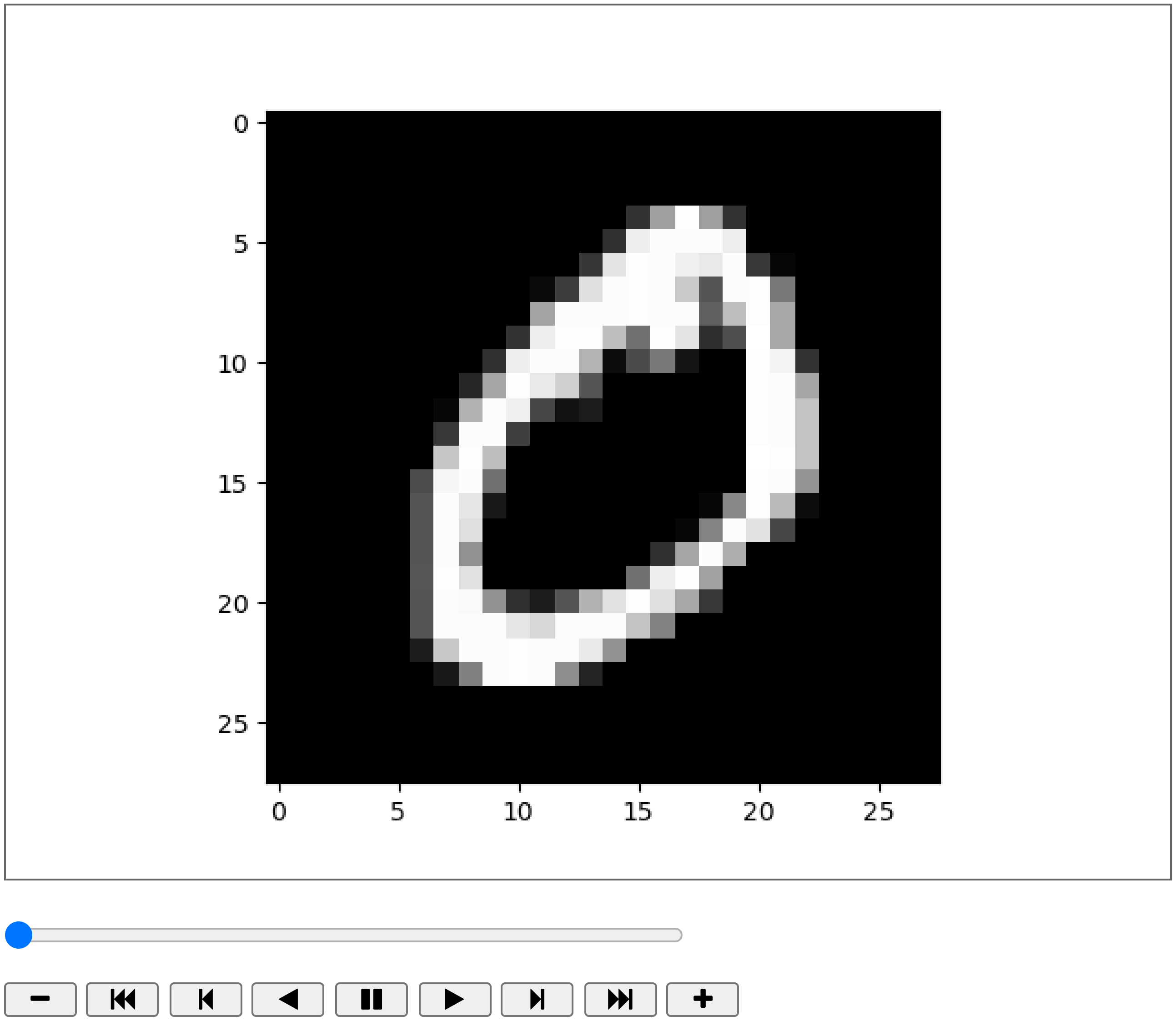


Taylor series

Taylor series vectorized

```
In [15]: zero_images_anim
```

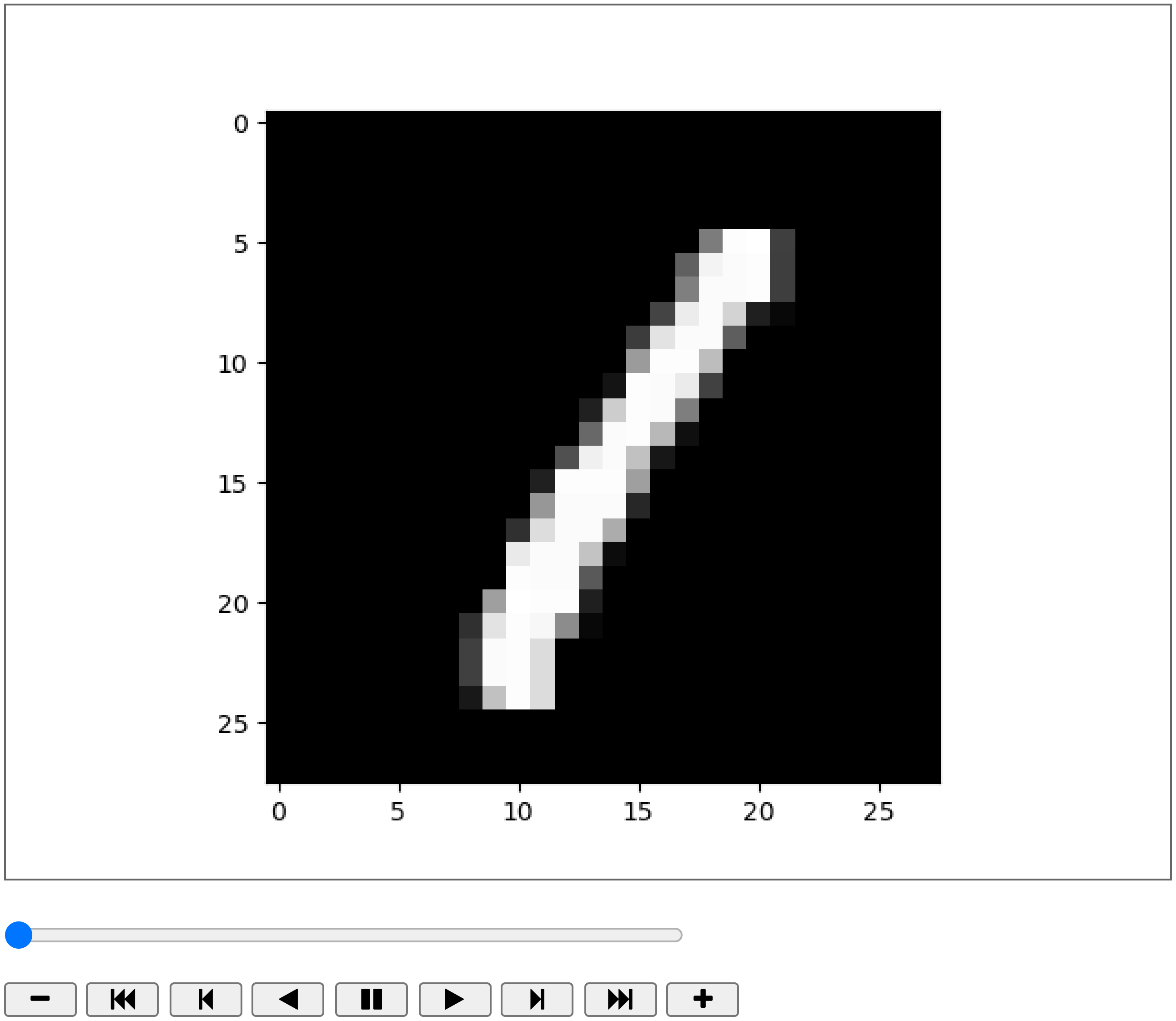
```
Out[15]:
```



[✓ Once] Loop Reflect

```
In [17]: one_images_anim
```

```
Out[17]:
```



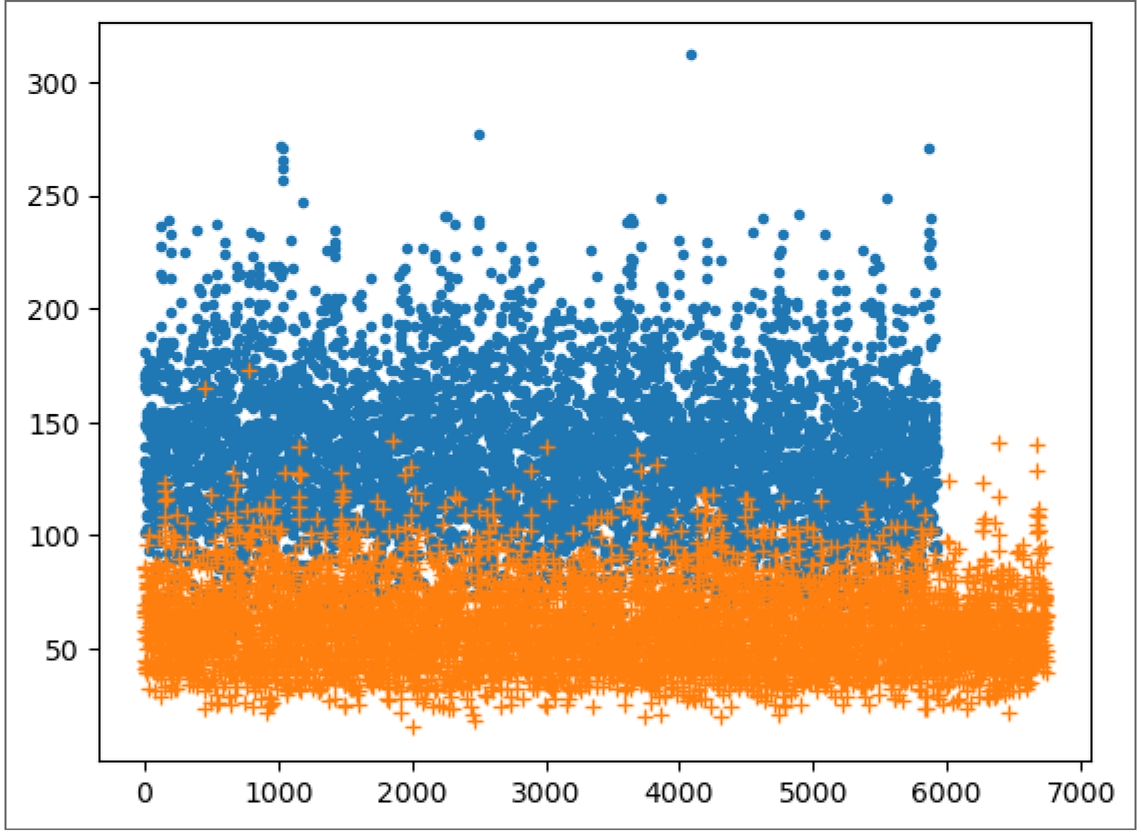
[✓ Once] Loop Reflect

What is a feature

Any property of data sample that helps with the task.

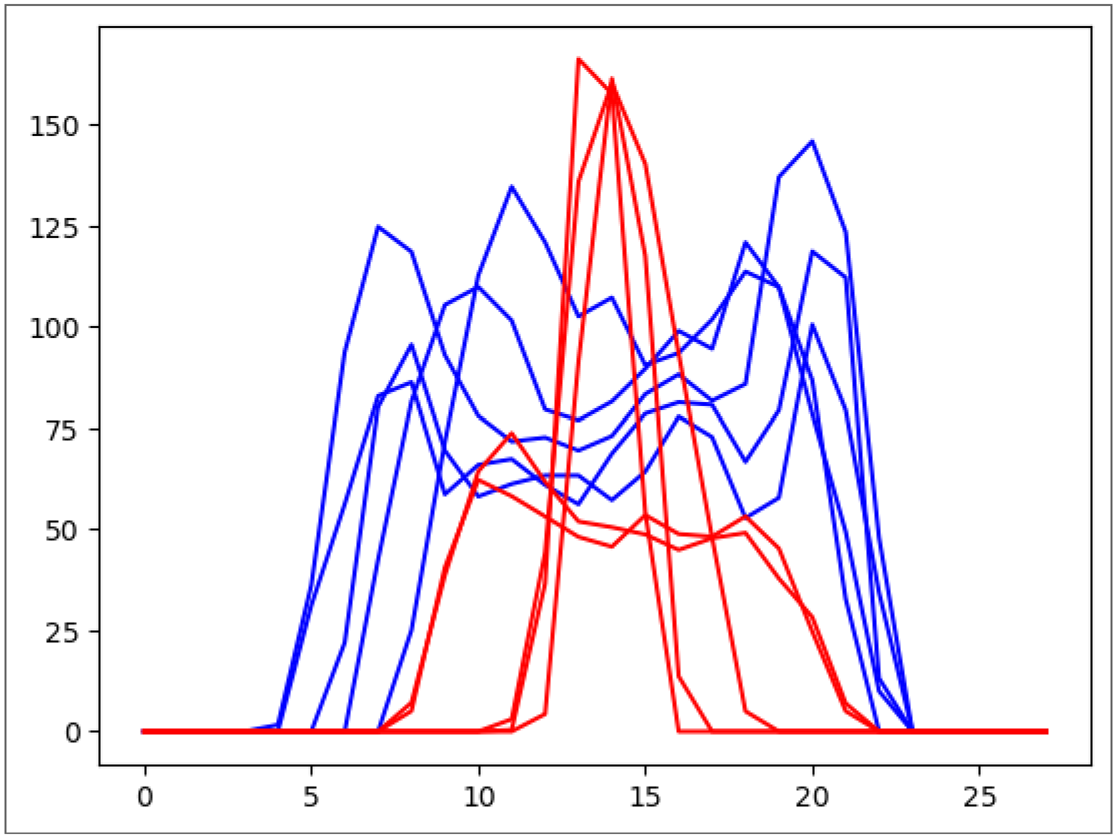
```
In [18]: def feature_n_pxls(imgs):
             n, *shape = imgs.shape
    return np.sum(imgs[:, :, :].reshape(n, -1) > 128, axis=1)

n_pxls_zero_images = feature_n_pxls(zero_images)
n_pxls_one_images = feature_n_pxls(one_images)
fig, ax = plt.subplots()
ax.plot(n_pxls_zero_images, '.')
ax.plot(n_pxls_one_images, '+')
```

```
Out[18]:

[<matplotlib.lines.Line2D at 0x7f838d398490>]
```


300
250
200
150
100
50
0
1000
2000
3000
4000
5000
6000
7000

In [19]:

```
fig, ax = plt.subplots()
        for i in range(5):
    ax.plot(zero_images[i].mean(axis=0), 'b-')
for i in range(5):
    plt.plot(one_images[i].mean(axis=0), 'r-')
```



```
In [20]: wts = zero_images[0].mean(axis=0)
         mean = (np.arange(wts.shape[0]) * wts).sum() / np.sum(wts)
var = ((np.arange(wts.shape[0]) - mean)**2 * wts).sum() / np.sum(wts)
var
```

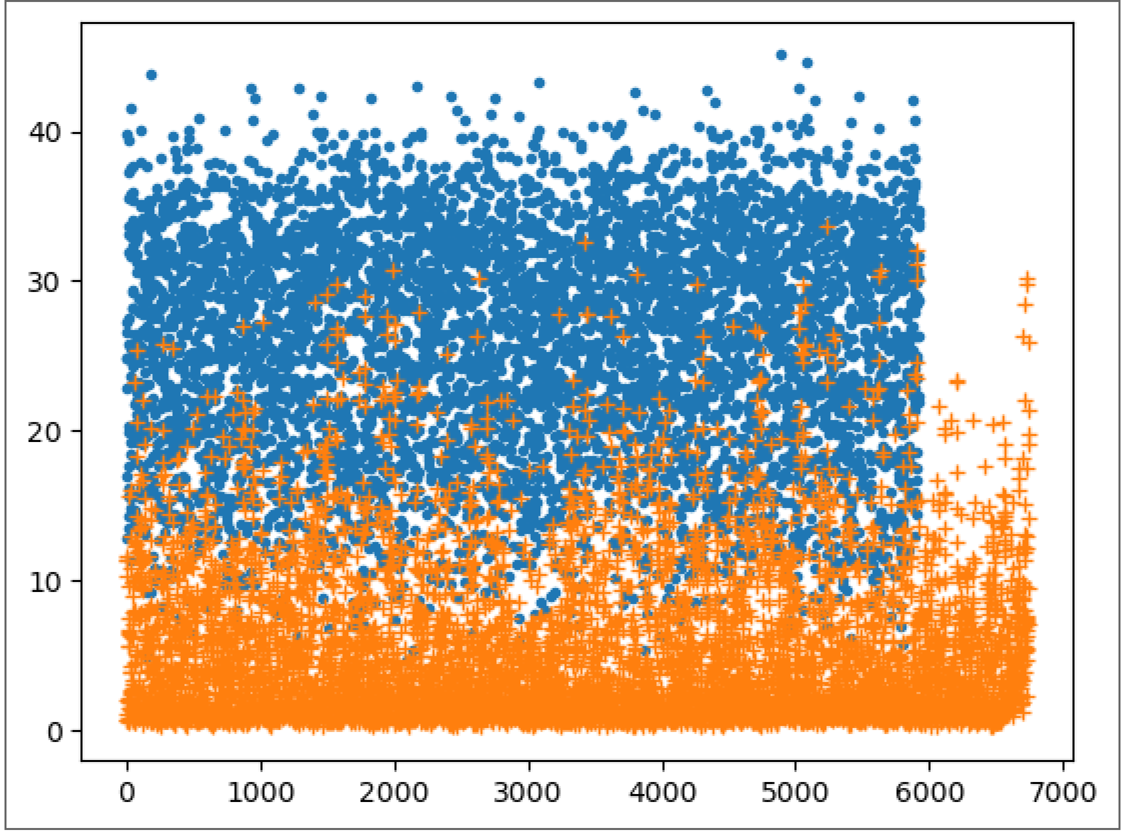
```
Out[20]:

22.811061800377757
```

```
In [21]: def feature_y_var(img):
             wts = img.mean(axis=0)
    mean = (np.arange(wts.shape[0]) * wts).sum() / np.sum(wts)
    var = ((np.arange(wts.shape[0]) - mean)**2 * wts).sum() / np.sum(wts)
    return var
feature_y_var(zero_images[0]), feature_y_var(one_images[0])
```

```
Out[21]:

(22.811061800377757, 11.384958735403274)
```

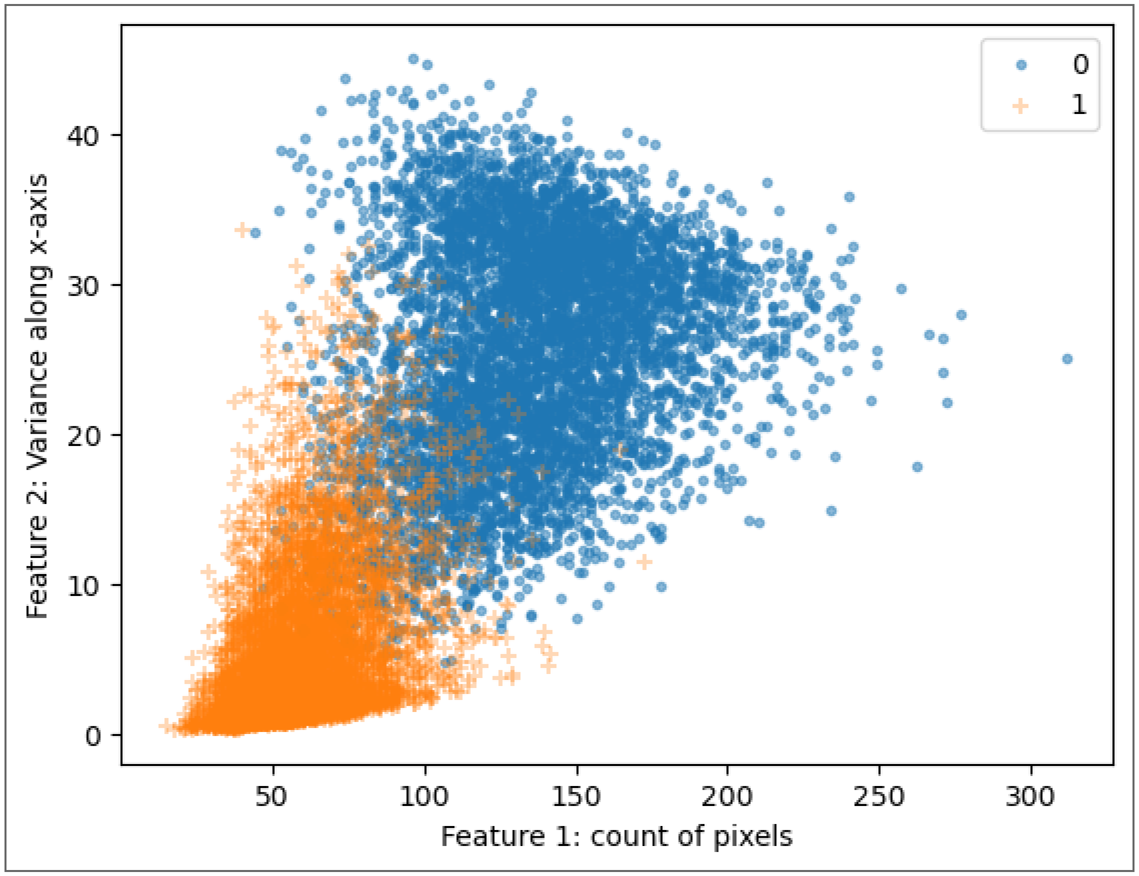
```
In [22]: def feature_y_var(imgs):
             wts = imgs.mean(axis=-2)
    arange = np.arange(wts.shape[-1])
    mean = (arange * wts).sum(axis=-1) / wts.sum(axis=-1)
    mean = mean[:, np.newaxis]
    var = ((arange - mean)**2 * wts).sum(axis=-1) / wts.sum(axis=-1)
    return var

fig, ax = plt.subplots()
ax.plot(feature_y_var(zero_images), '.')
ax.plot(feature_y_var(one_images), '+')
```

```
Out[22]:

[<matplotlib.lines.Line2D at 0x7f838d58a1a0>]
```



0
10
20
30
40
0
1000
2000
3000
4000
5000
6000
7000

```
In [24]: fig, ax = plt.subplots()
         draw_features(ax, zero_features, one_features)
plt.show()
```



```
In [27]: def error(X, Y, bfm):
             # YOUR CODE HERE
    raise NotImplementedError()

def grad_error(Xw, Yw, bfm):
    # YOUR CODE HERE
```

```python
    raise NotImplementedError()

def train(X, Y, lr = 0.1):
    # YOUR CODE HERE
    raise NotImplementedError()

OPTIMAL_BFM, list_of_bfms, list_of_errors = train(X, Y)
fig, ax = plt.subplots()
ax.plot(list_of_errors)
ax.set_xlabel('t')
ax.set_ylabel('loss')
plt.show()
```

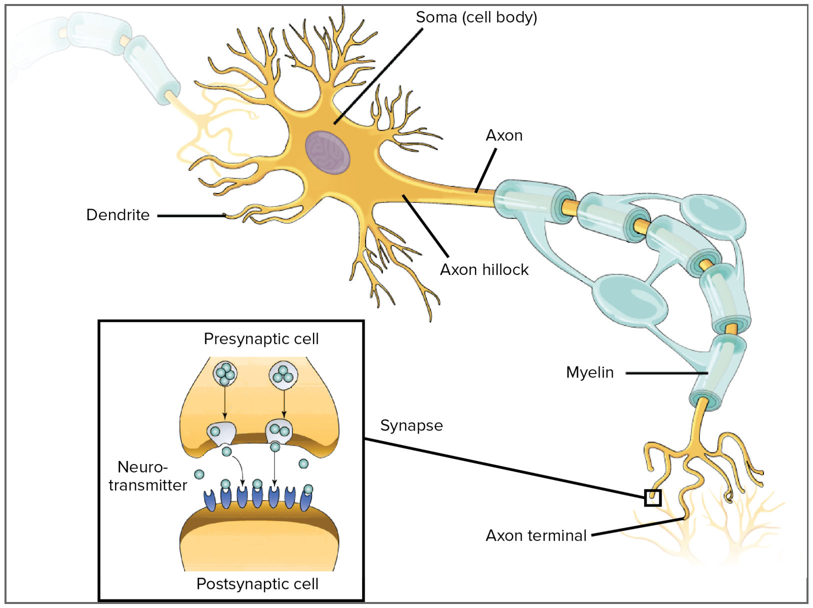
```
---------------------------------------------------------------------------
-----
NotImplementedError                       Traceback (most recent call
last)
Cell In[27], line 13
      9 def train(X, Y, lr = 0.1):
     10     # YOUR CODE HERE
     11     raise NotImplementedError()
---> 13 OPTIMAL_BFM, list_of_bfms, list_of_errors = train(X, Y)
     14 fig, ax = plt.subplots()
     15 ax.plot(list_of_errors)

Cell In[27], line 11, in train(X, Y, lr)
      9 def train(X, Y, lr = 0.1):
     10     # YOUR CODE HERE
---> 11     raise NotImplementedError()

NotImplementedError:
```

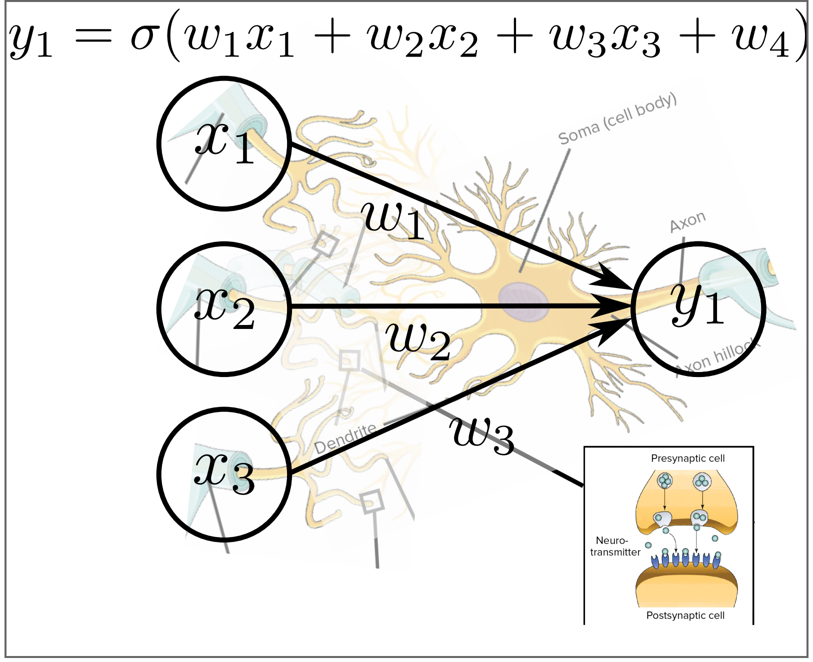
Single Layer Neural Networks

Read Chapter 2 and 3 of UDL Book

Notes Single Layer Neural Networks are simplest kind of neural networks. But before we dive into single layer neural networks, may be we should focus on the name *neural* networks. The name neural networks comes from biological neurons.

Similarities between Artificial neuron and Biological neuron





$$y_1 = \sigma(w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4)$$

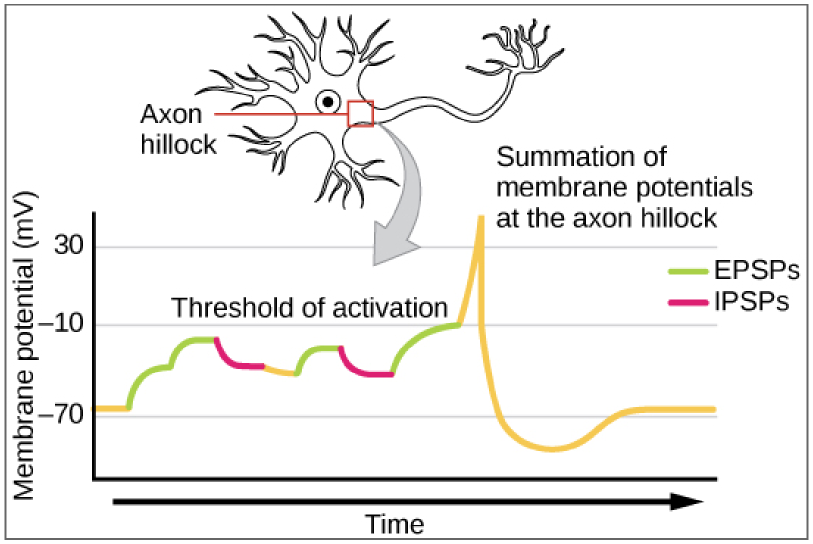
1. The excitation or firing of a biological neuron can be equated to a high positive value of units (x_1, x_2, x_3) in artificial neurons.
2. The synapse in biological neuron determines which input excitations will have excitatory or inhibitary impact on output excitations. Synapses can strengthen, weaken, disconnect or form new connections during biological learning. Similarly to excitatory synapes, positive weights can cause positive input values to contribute to positive output values.
3. Usually multiple excitatory inputs are required excite the output neuron.

References:

1. **https://openstax.org/books/biology/pages/35-2-how-neurons-communicate**

Differences

1. Biological neuron is all or None
2. Biological neuron has a time component



notes

1. The activity of the biological neuron is an "all-or-none" process. Articial activations are typically continuous range. Even when sigmoid or softmax nonlinearities.
2. Biological neuron has time dynamics. The input activations are integrated over time.