# VIR lab 2

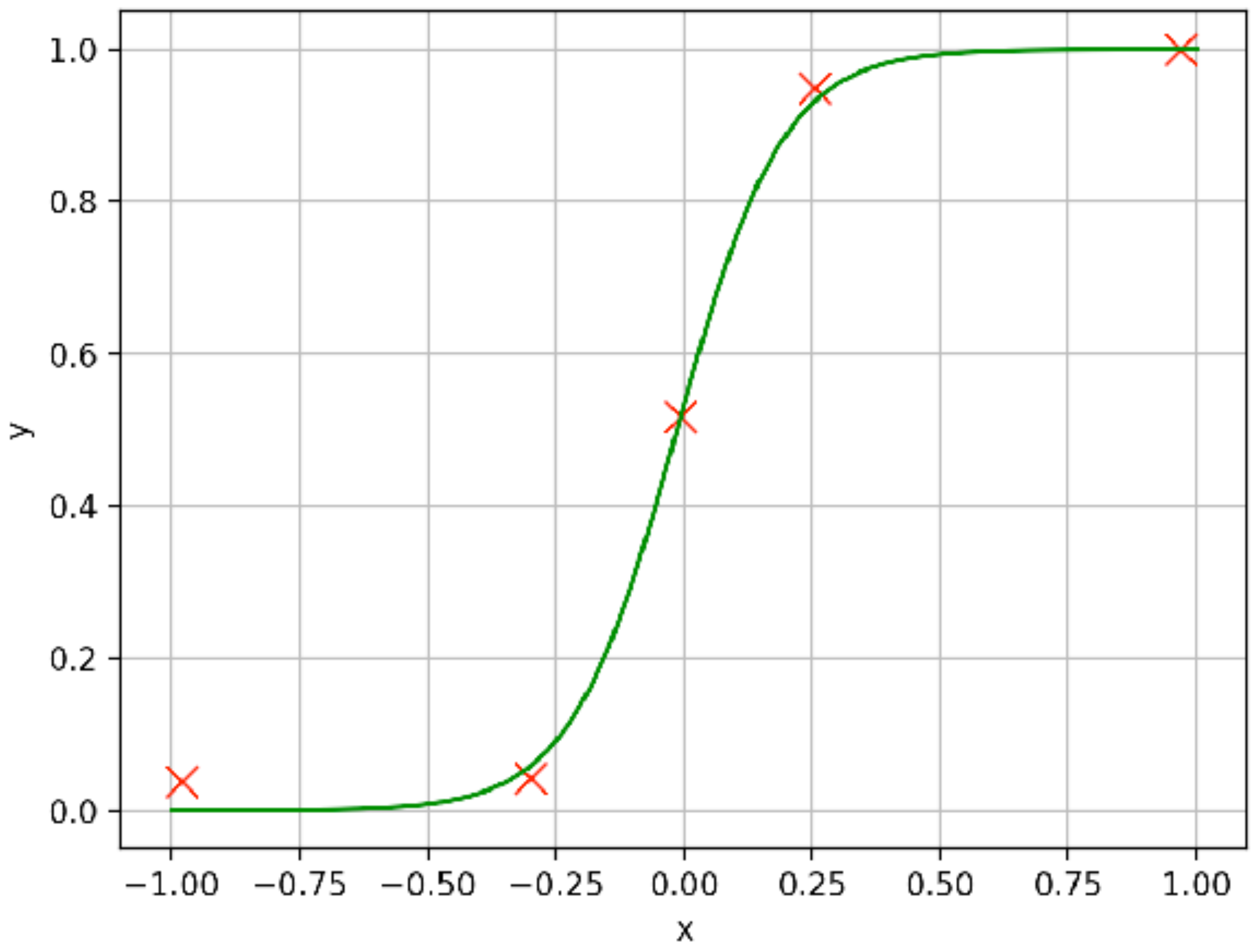
## Non-linear regression and computational graphs

**Karel Zimmermann**
**Czech Technical University in Prague**
**Faculty of Electrical Engineering, Department of Cybernetics**

# Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

```
pts = np.load('pts.npy')
```



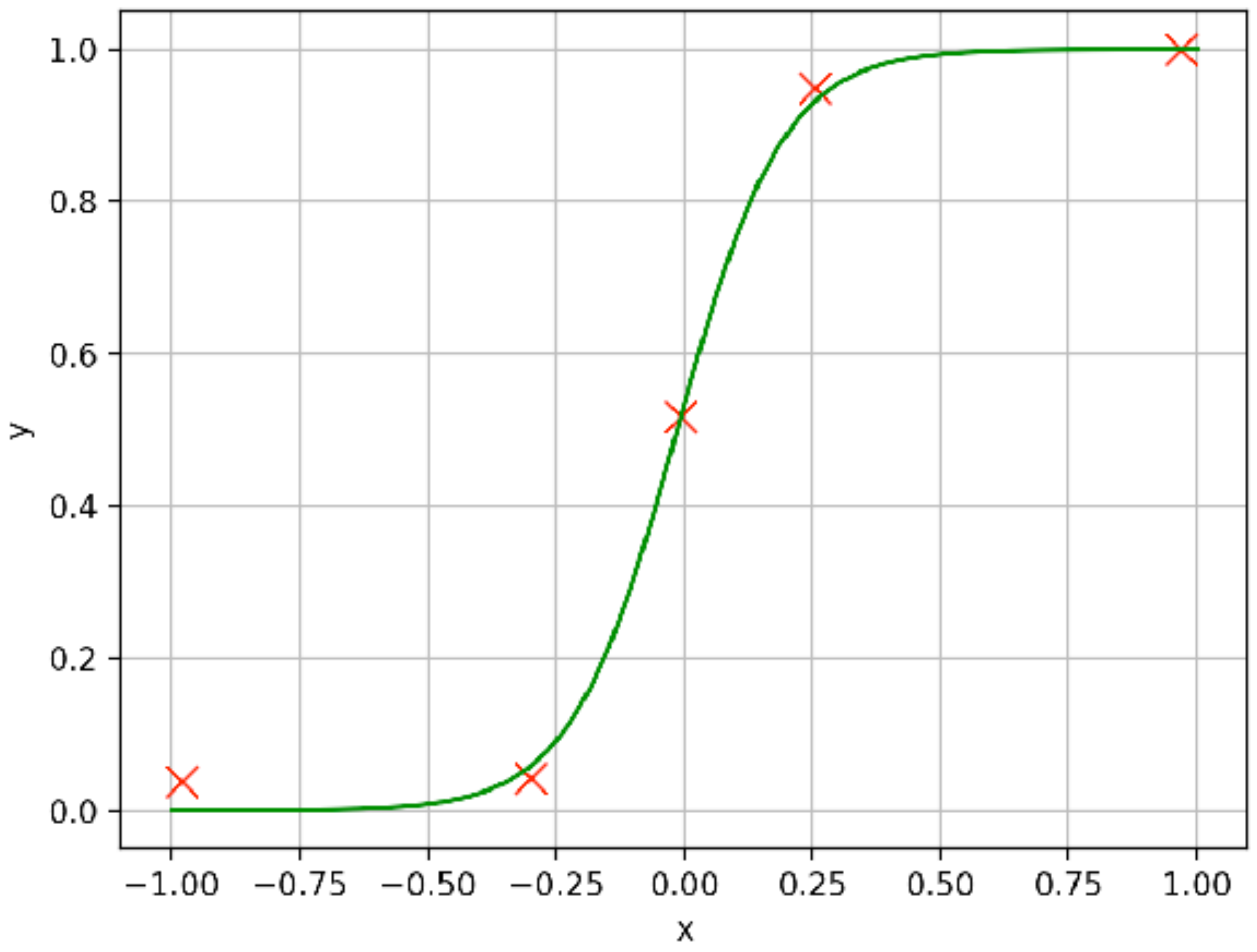
# Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

```
pts = np.load('pts.npy')
```

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

```
for i in range(30):
    f = …       # use np.exp()
```



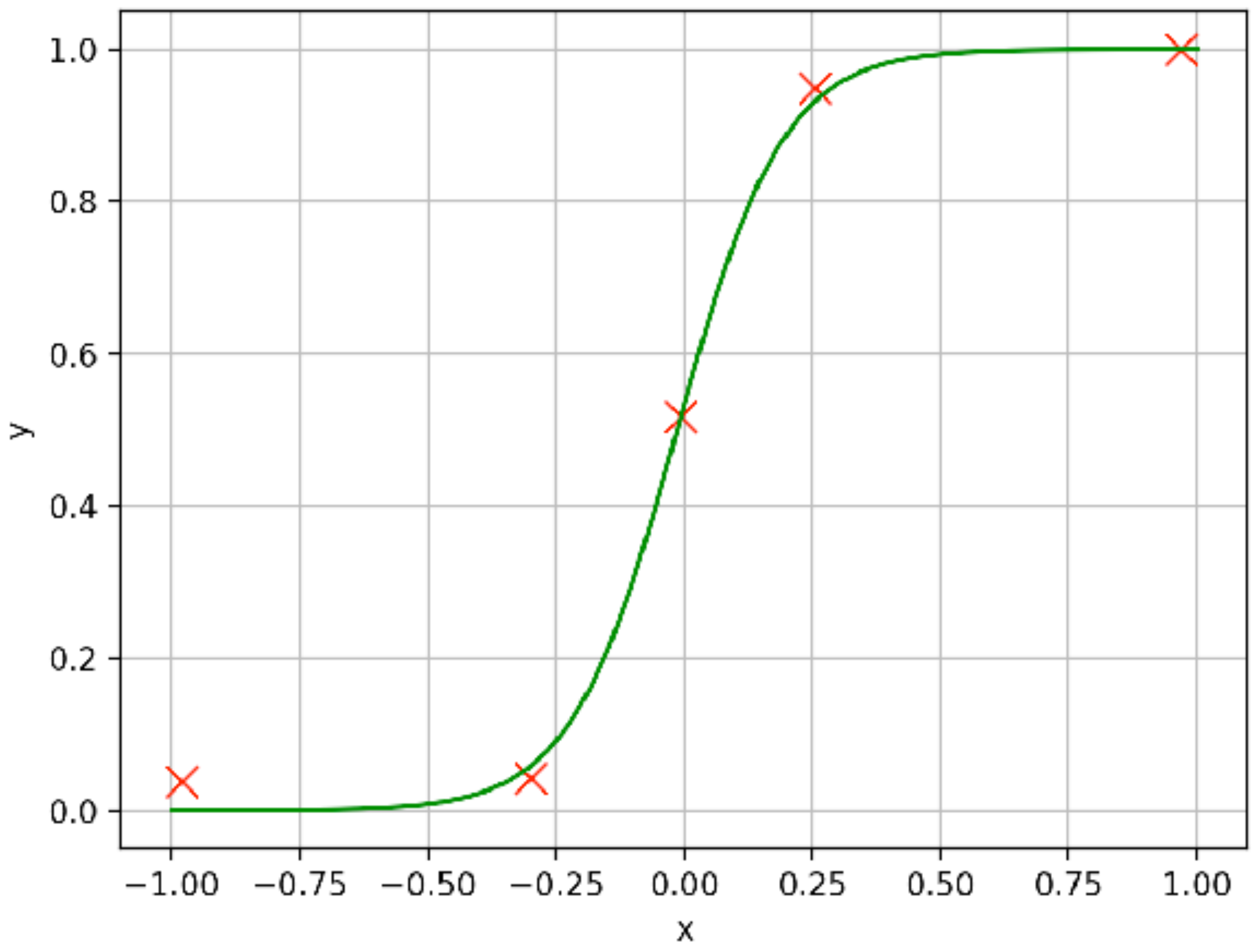
# Assignment

$$\mathcal{D} = \{\mathbf{x}_1, y_1 \dots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

$$\mathscr{L}(\mathbf{w}) = \sum_i \left(f(\mathbf{x}_i, \mathbf{w}) - y_i\right)^2$$

```
pts = np.load('pts.npy')

for i in range(30):
    f = …     # use np.exp()
    loss = … # use np.sum()
```



# Assignment

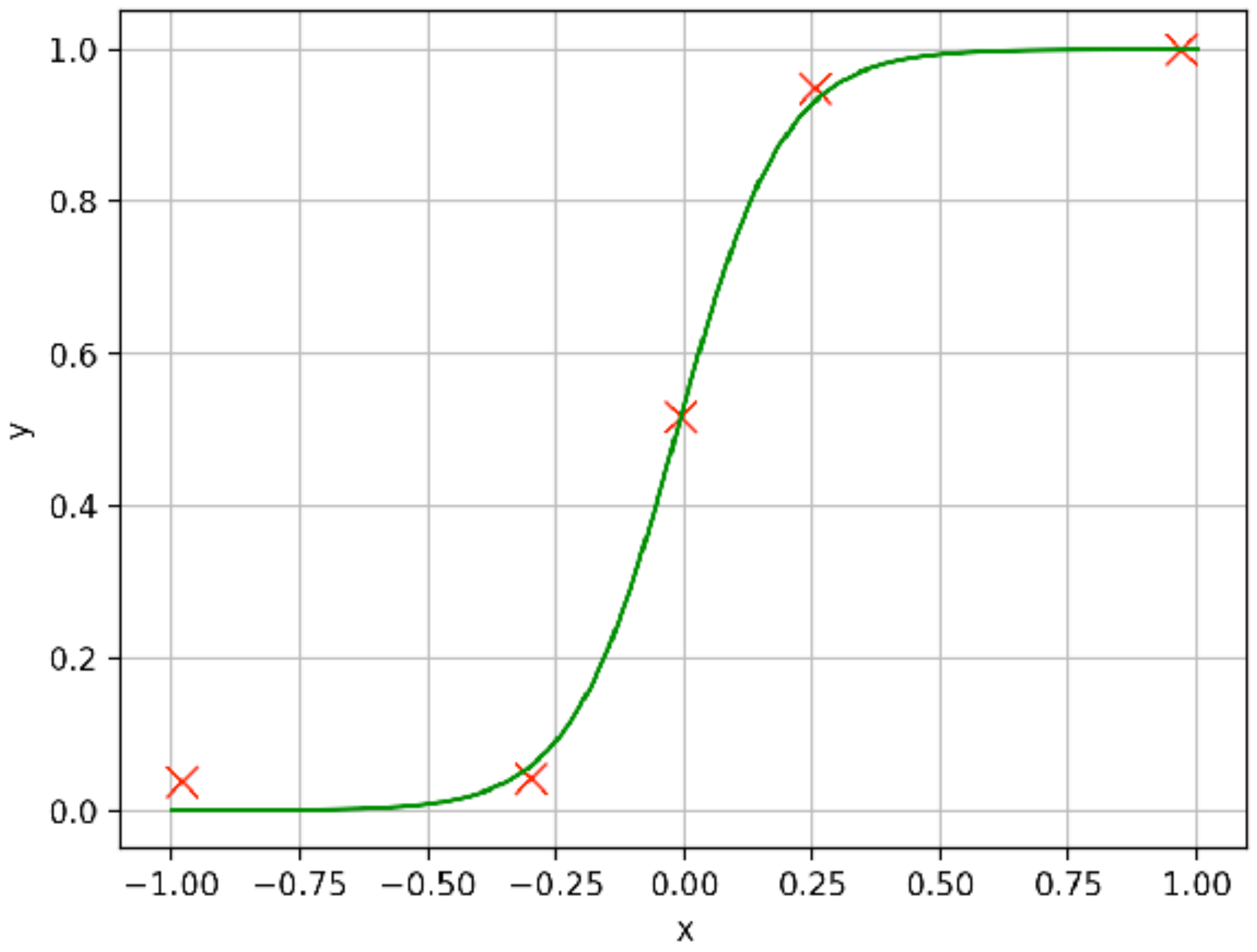
$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

$$\mathscr{L}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w})$$

```
pts = np.load('pts.npy')

for i in range(30):
    f = …      # use np.exp()
    loss = … # use np.sum()

    grad = … # compute analytically
    w -= learning_rate * grad
```



# Assignment

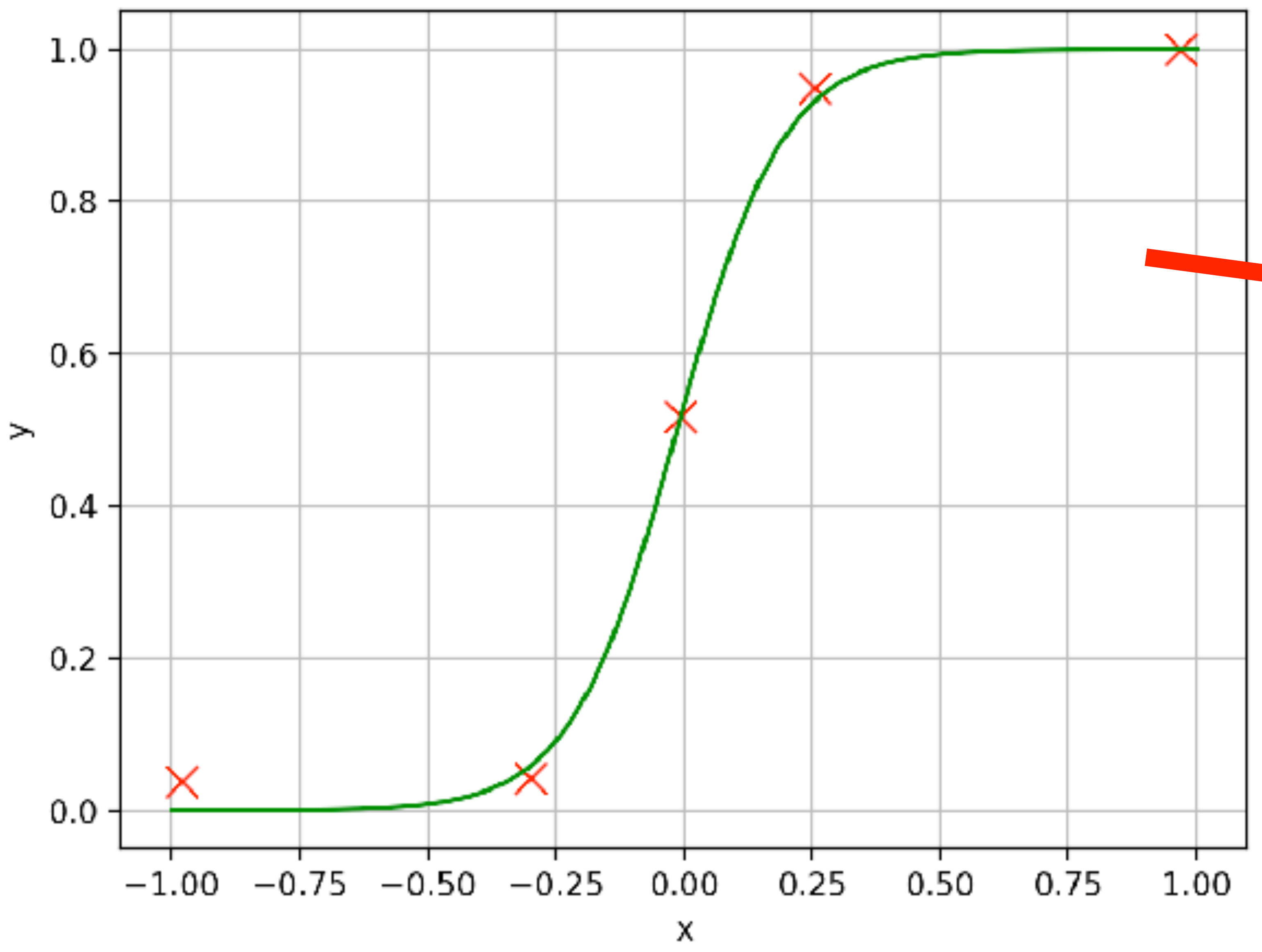
$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

$$\mathscr{L}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w})$$

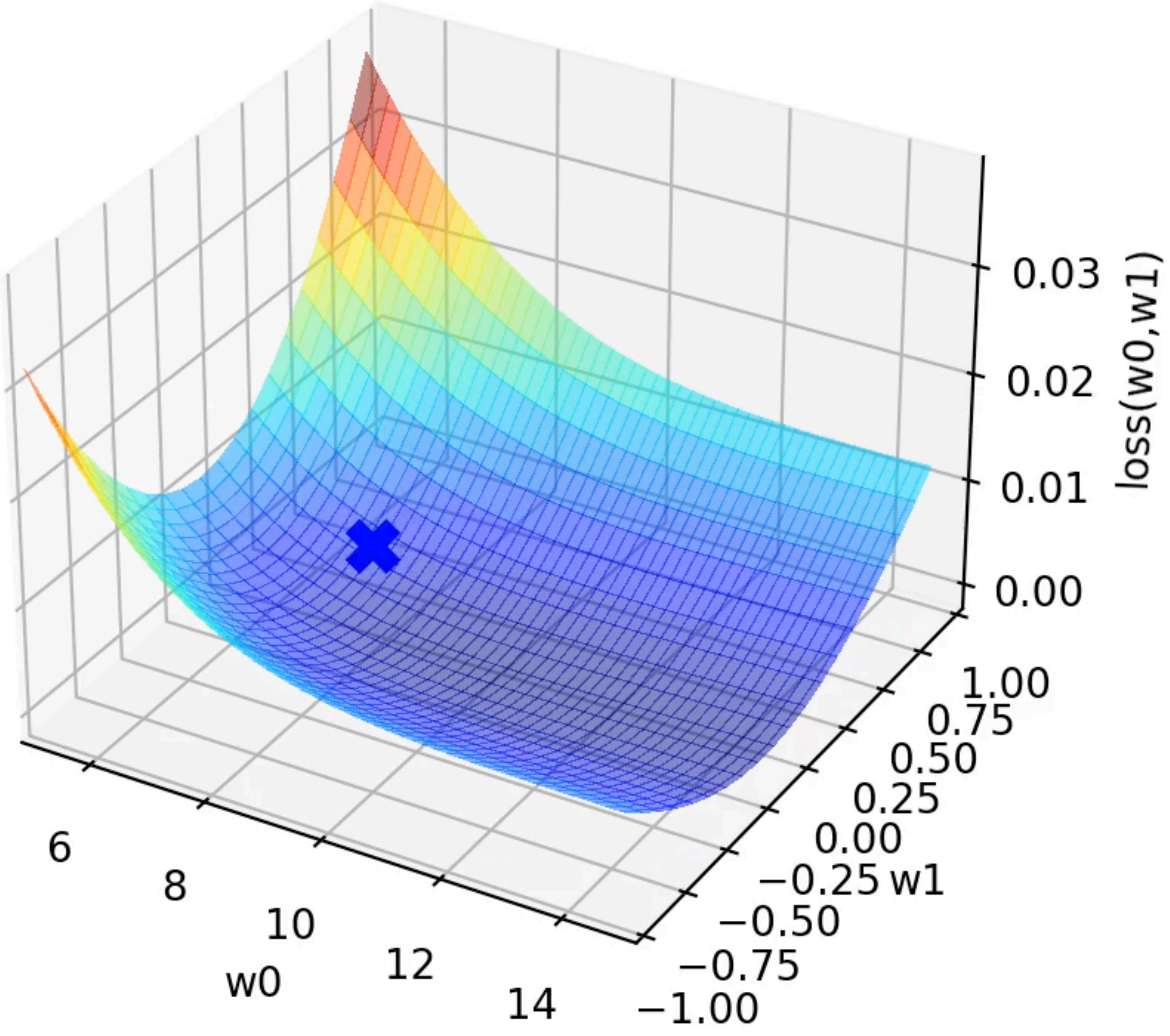
```
pts = np.load('pts.npy')

for i in range(30):
    f = …      # use np.exp()
    loss = … # use np.sum()

    grad = … # compute analytically
    w -= learning_rate * grad

    # visualize result
```



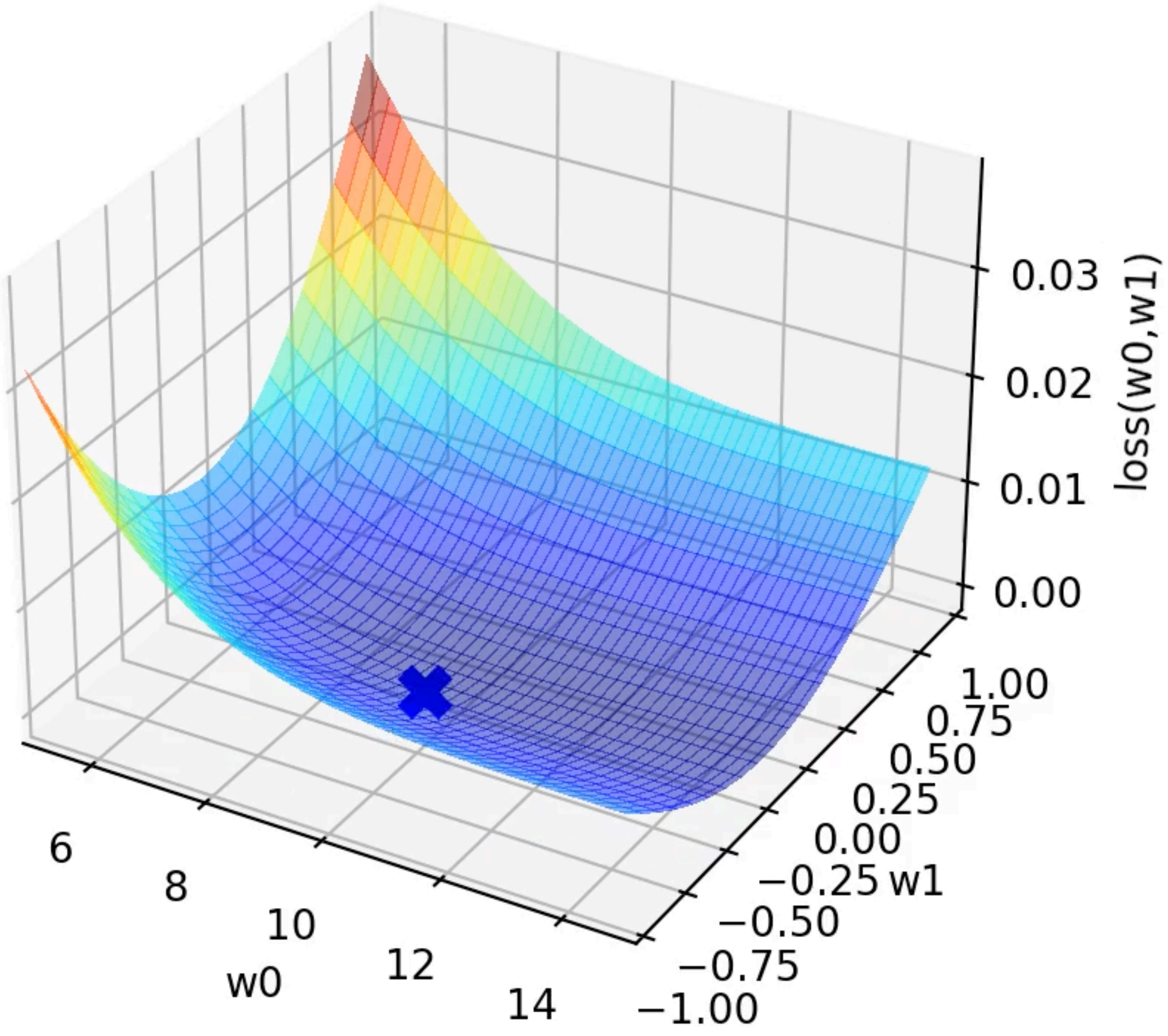
# Assignment

- Derive grad of sigmoid
- Download the template a fill the stuff in
    - Try different learning rates in order to get best convergence
    - Try different initializations of w

- What is the reasonable learning rate?



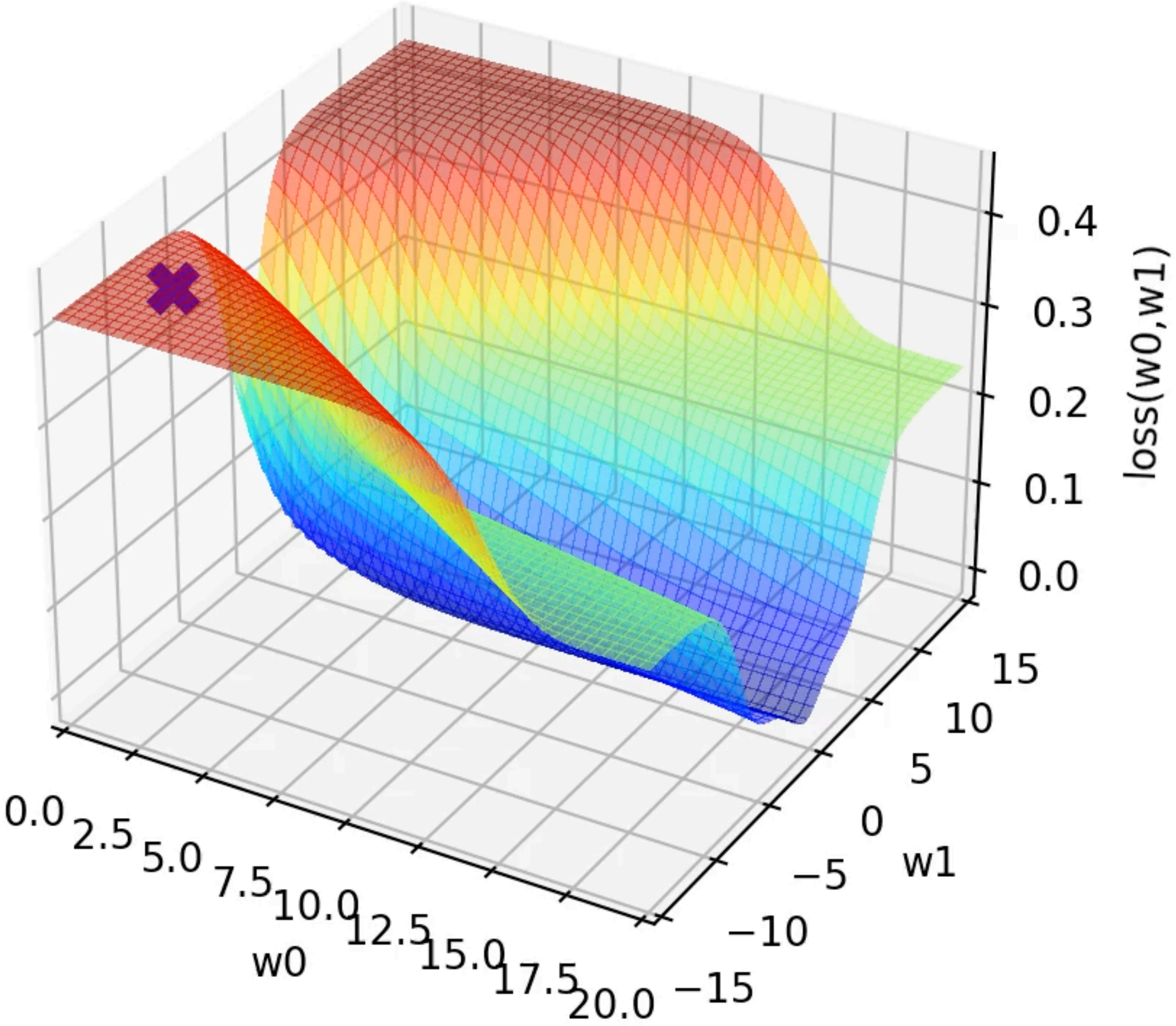
LSQ
Too small learning rate in w0-dim



LSQ
Too big learning rate in w1-dim

Should not we use different learning rate along different axis w0,w1?

- What is the reasonable learning rate?



Distant initialization (landscape at larger scale is suprisingly wild)

# Discussion

- Is it linear least squares?
- Is the loss quadratic?
- How many steps required for full Newton method for quadratic loss?

- Non-linear least squares, GD, SGD
- mean / sum
- landscape w0/w1

- Computational graph + backprop (is it DAG?)
- Jacobian and Vector-Jacobian-Product (whiteboard examples + discussion)
- PyTorch framework for now prohibited ;-)

# Pytorch

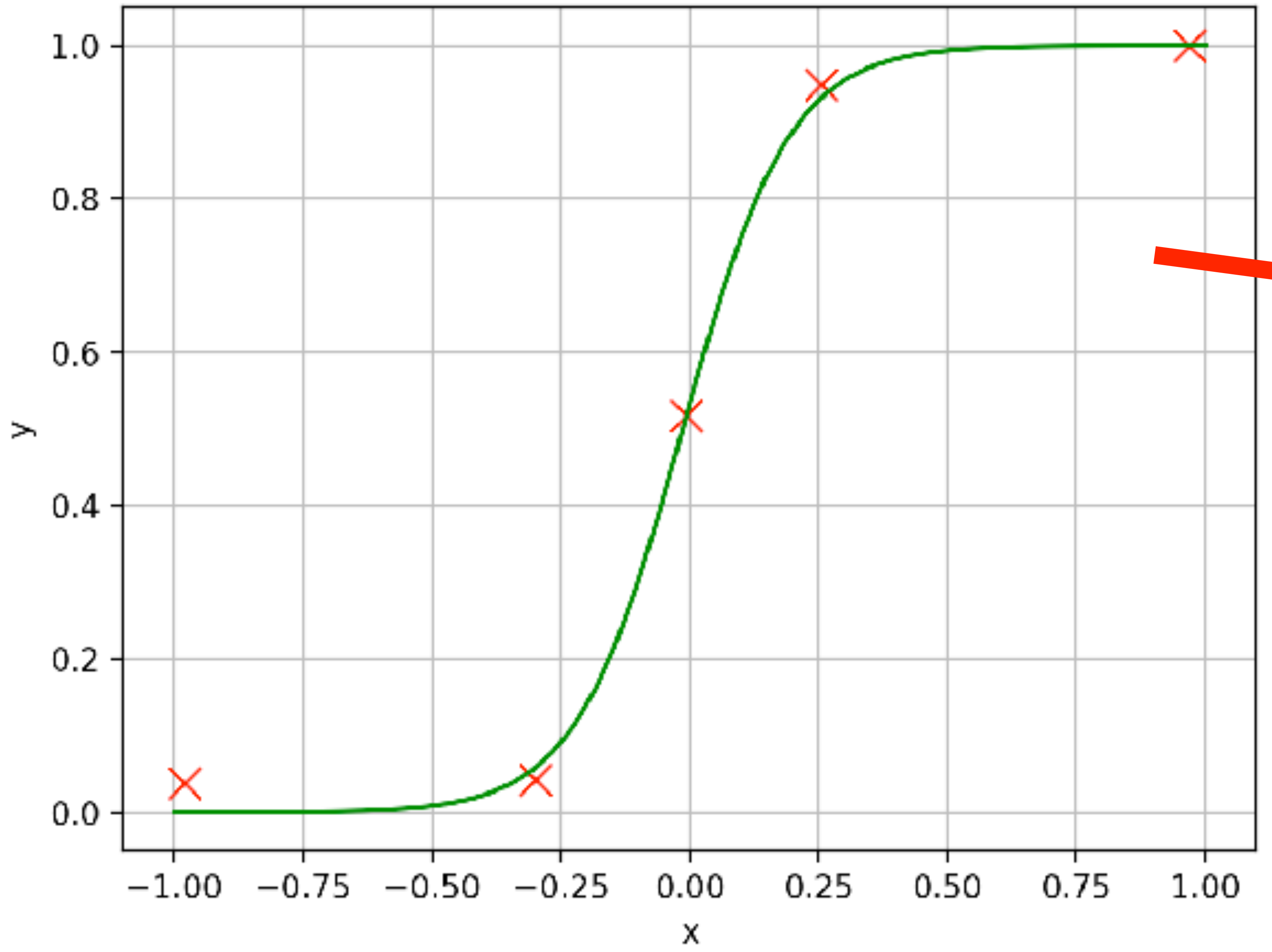
$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

$$f(\mathbf{x}_i, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w}_0 * \mathbf{x}_i + \mathbf{w}_1)}}$$

$$\mathscr{L}(\mathbf{w}) = \sum_i (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w})$$

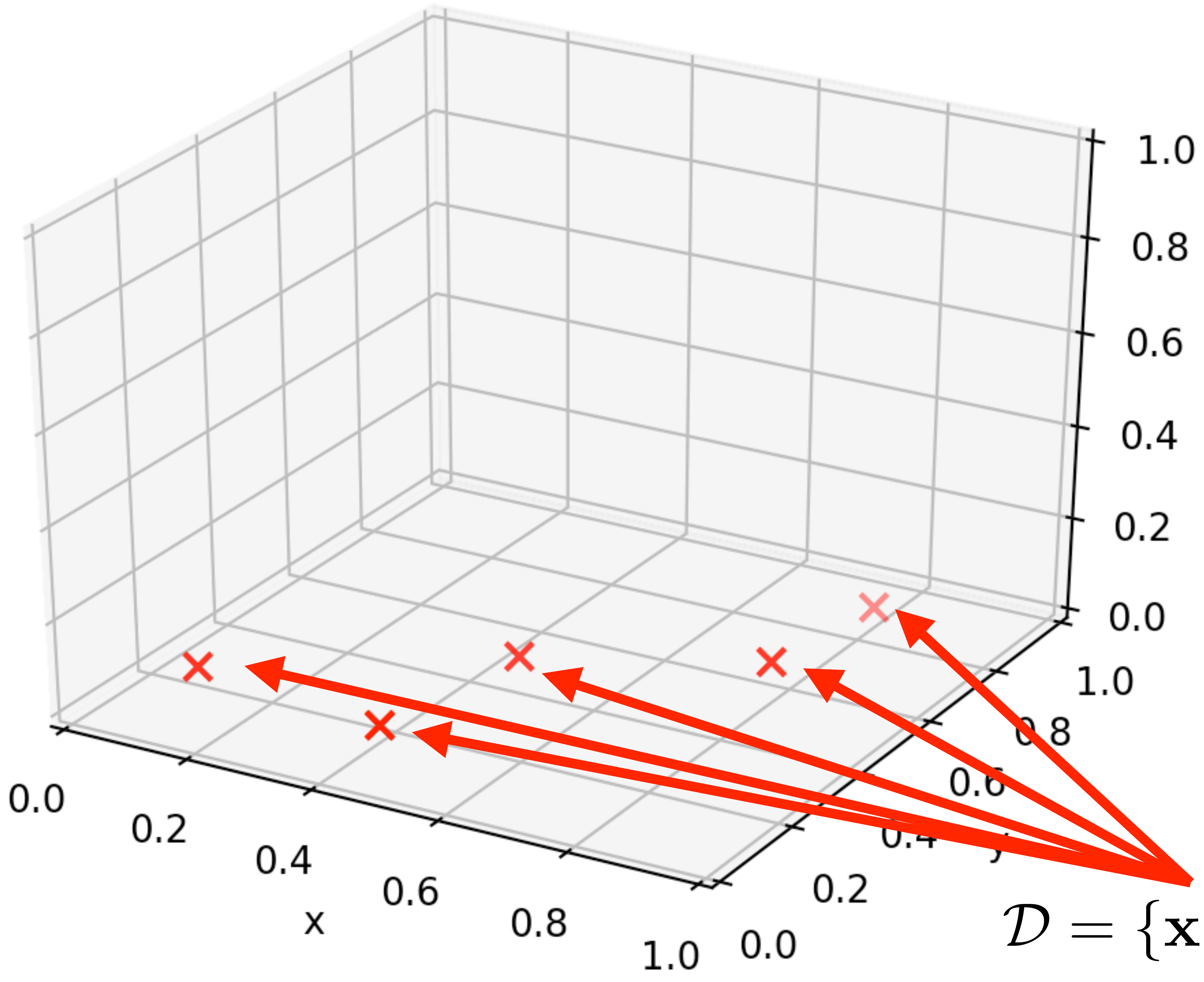
```
pts = np.load('pts.npy')

for i in range(30):
    f = …      # use torch.exp()
    loss = … # use torch.sum()

   grad = torch.autograd.grad(loss, w)
    w -= learning_rate * grad

   # visualize result
```



# Discussion

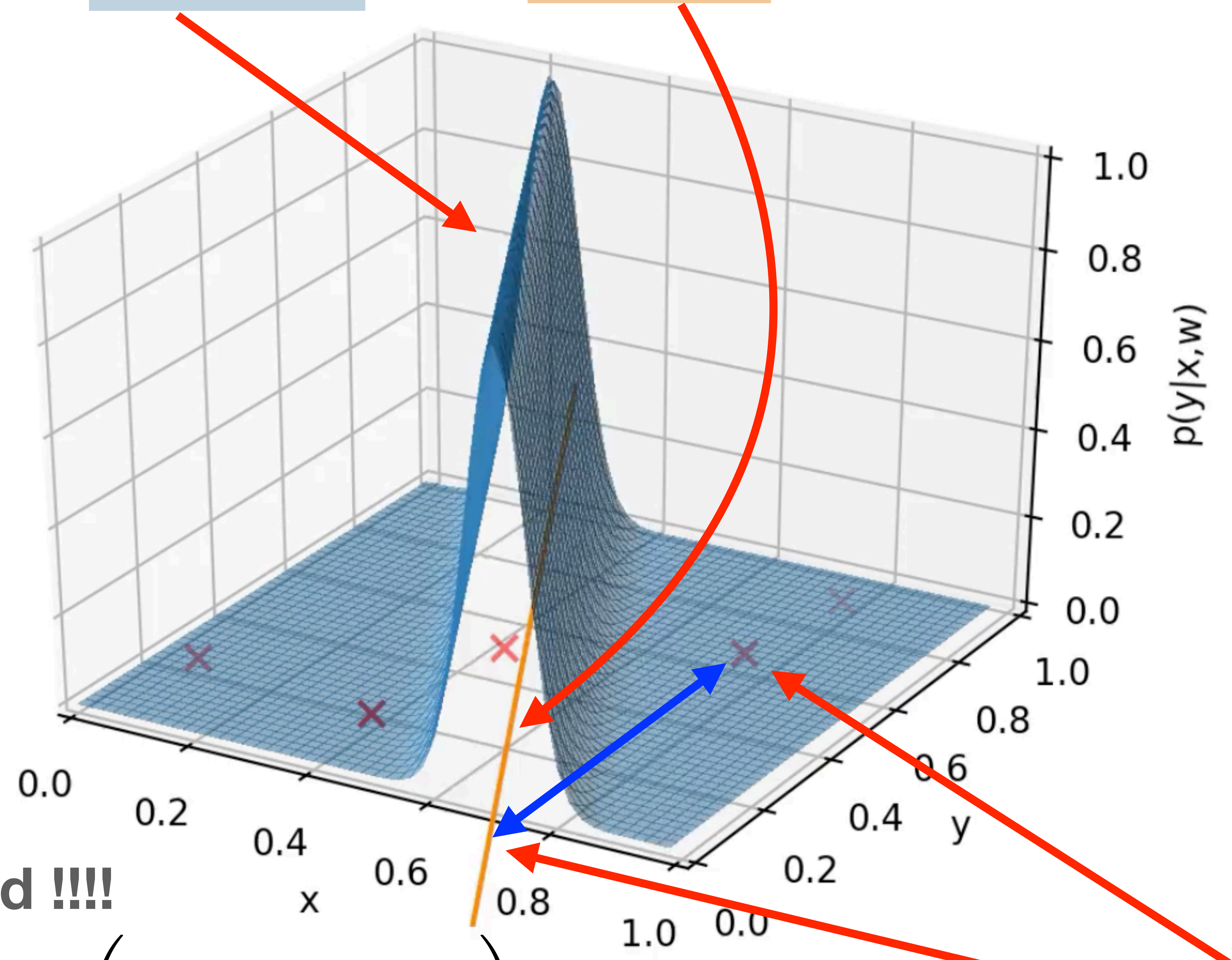
- Why the hell should I use the L2-norm????
- The simplest justification is MLE approach.

$$p(y|\mathbf{x},\mathbf{w}) \sim \mathcal{N}_y(w_1 x + w_0,\ \sigma^2)$$



$$\mathcal{D} = \{\mathbf{x}_1, y_1 \ldots \mathbf{x}_N, y_N\}$$

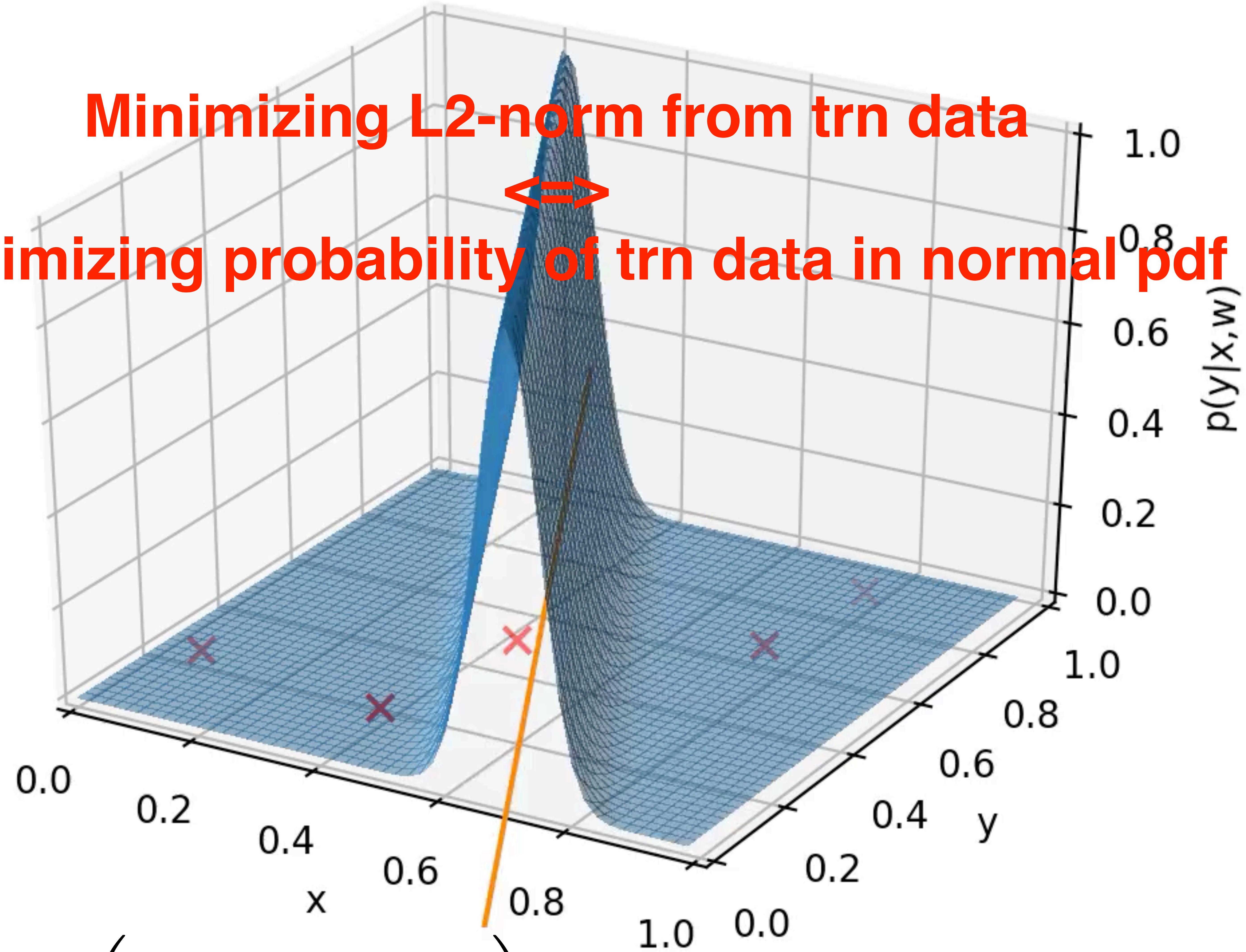
$$p(y|\mathbf{x},\mathbf{w}) \sim \mathcal{N}_y(w_1 x + w_0,\ \sigma^2)$$



**Derive on blackboard !!!!**

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left(\prod_i p(y_i|\mathbf{x}_i,\mathbf{w})\right) = \arg\min_{\mathbf{w}} \sum_i (w_1 x_i + w_0 - y_i)^2$$

$$p(y|\mathbf{x},\mathbf{w}) \sim \mathcal{N}_y(w_1 x + w_0,\ \sigma^2)$$



**Minimizing L2-norm from trn data**

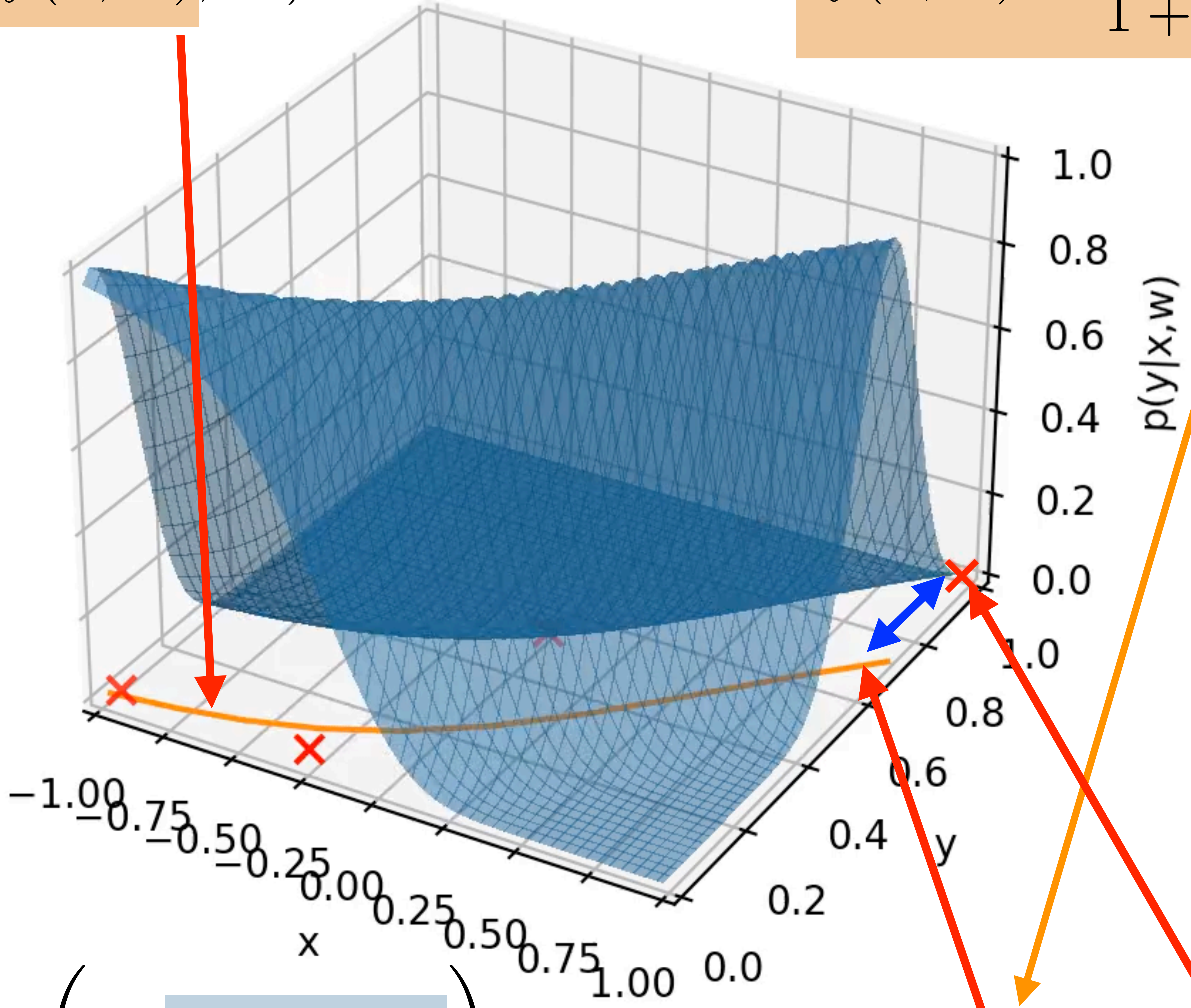
**<=>**

**Maximizing probability of trn data in normal pdf**

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left( \prod_i p(y_i|\mathbf{x}_i,\mathbf{w}) \right) = \arg\min_{\mathbf{w}} \sum_i (w_1 x_i + w_0 - y_i)^2$$

$$p(y|\mathbf{x},\mathbf{w}) \sim \mathcal{N}_y(f(\mathbf{x},\mathbf{w}),\sigma^2)$$

$$f(\mathbf{x},\mathbf{w}) = \frac{1}{1+e^{-(\mathbf{w}_0*\mathbf{x}+\mathbf{w}_1)}}$$



$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \left(\prod_i p(y_i|\mathbf{x}_i,\mathbf{w})\right) = \arg\min_{\mathbf{w}} \sum_i (f(\mathbf{x}_i,\mathbf{w}) - y_i)^2$$

# Assignment classification

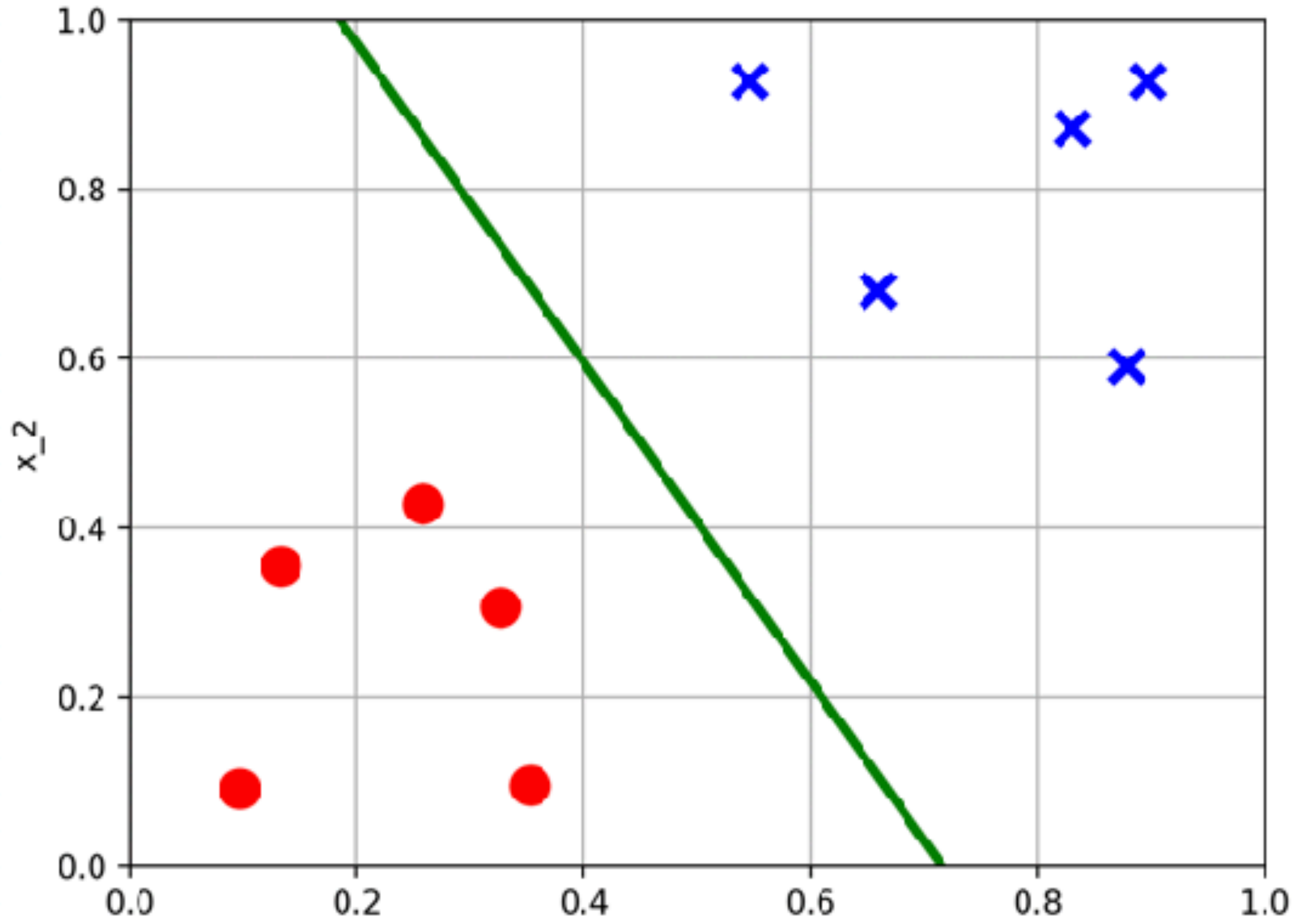
$$\mathcal{D} = \{\mathbf{x}_1, y_1 \dots \mathbf{x}_N, y_N\}$$

$$p = \sigma(x_0 w_0 + x_1 w_1 + w_2)$$

$$\mathscr{L}(\mathbf{w}) = \sum_i - y_i \log(p) + (1 - y_i)\log(1 - p)$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathscr{L}(\mathbf{w})$$

```
x = np.load(‘data_x.npy')
y = np.load(‘data_y.npy')

for i in range(30):
    p = …     # fill in
    loss = … # fill in

    grad = … # compute analytically
    w -= learning_rate * grad

    # visualize result
```



# Discussion

- What is the classification error?
- Can I directly optimize the classification error?
- Are there multiple decision boundaries that minimize classification error?
- How does the error correspond to the loss?
-

# Discussion

- Is dkt model anyhow better than
    - a fully connected neural network?
    - or linear function?

# Summary

- Multi-dimensional non-linear regression tackled by pure gradient descent
- Convergence issues of pure gradient approach
- Vector-Jacobian-Product
- Maximum likelihood justification of L2 norm
- Appropriate choice of architecture (linear, dkt, convNet,…)