The Fundamentals of Modern Deep Learning with PyTorch

2. Understanding the PyTorch API (9:30 - 10:15 am)

Lightning AI
Creators of PyTorch Lightning

PyCon US · 24
By the community, for the community

What is PyTorch?

What is PyTorch?

PyTorch is free and open-source software

https://pytorch.org/

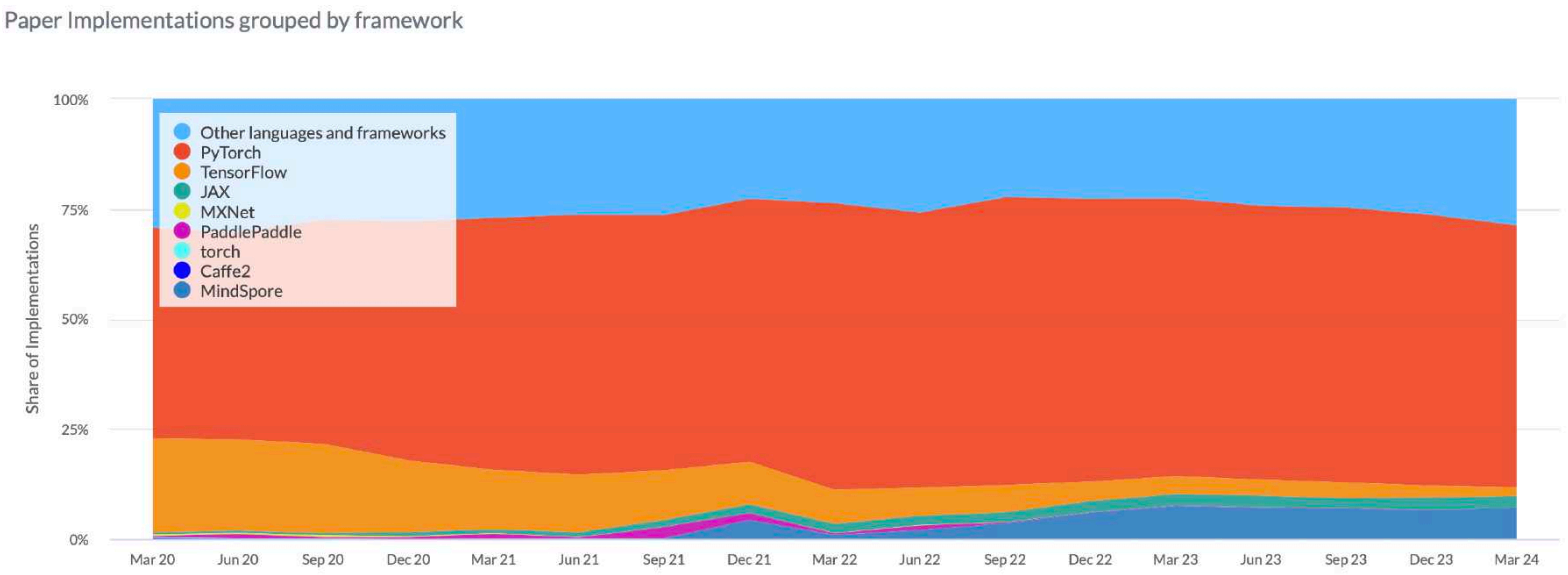
1 Tensor library

2 Automatic differentiation engine

What is PyTorch?

3 Deep learning library

PyTorch is widely used



https://paperswithcode.com/trends

What are tensors?

1 Tensor library

2 Automatic differentiation engine

What is PyTorch?

3 Deep learning library

Tensor library

Mathematically: a generalization of vectors, matrices, etc.

Computationally: a data container

Matrix
(rank-2 tensor)

```
a = torch.tensor([[1., 2., 3.],
                  [2., 3., 4.]])
a.shape
```

```
torch.Size([2, 3])
```

Tensor library

Tensor library == array library

Tensor library

torch.tensor ≈ numpy.array

+ GPU support

+ automatic differentiation support

1 Tensor library

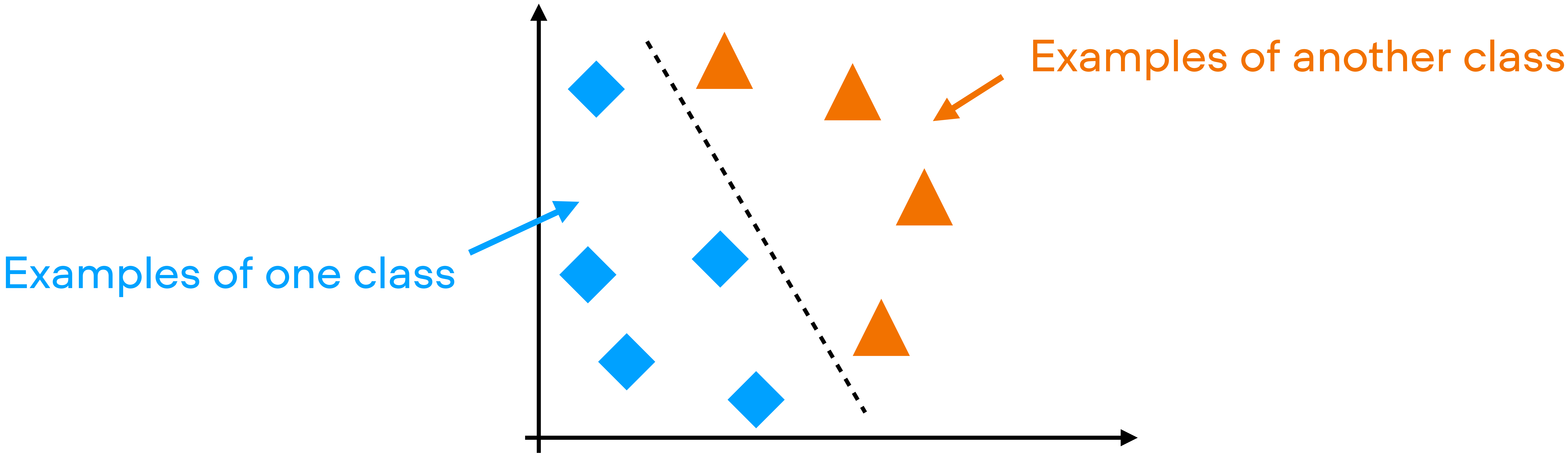
2 Automatic differentiation engine

What is PyTorch?

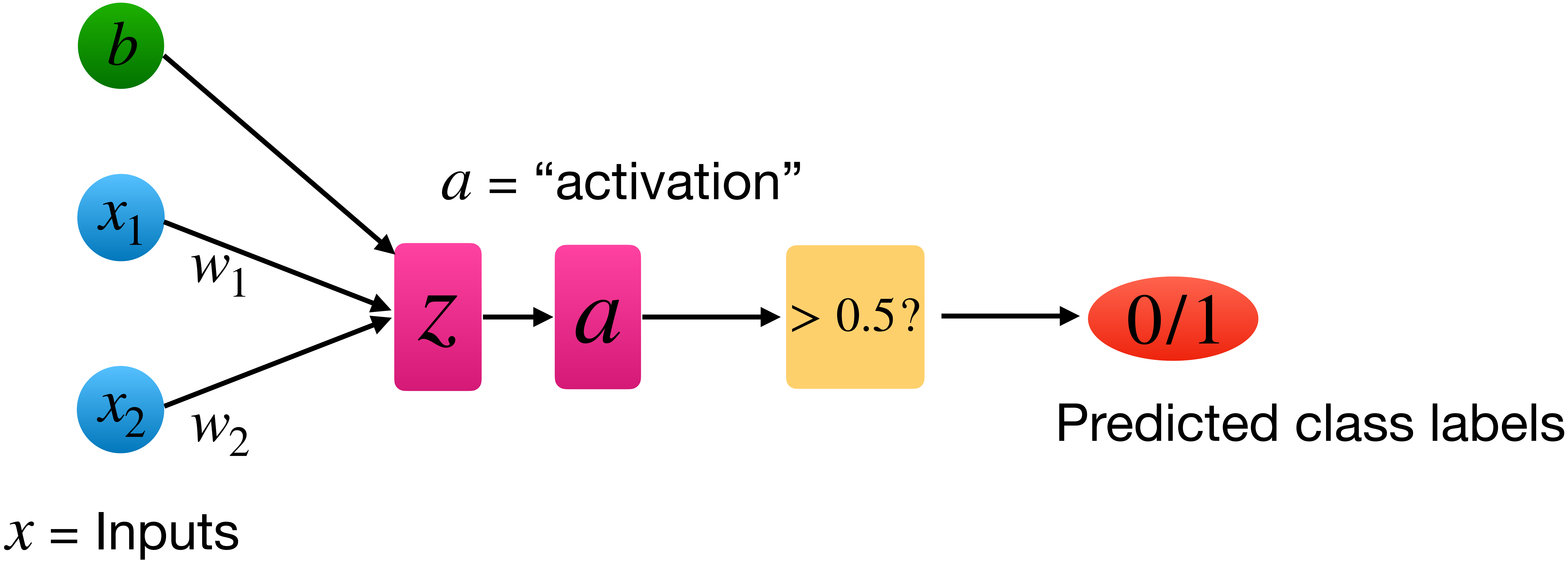
3 Deep learning library

A linear classifier

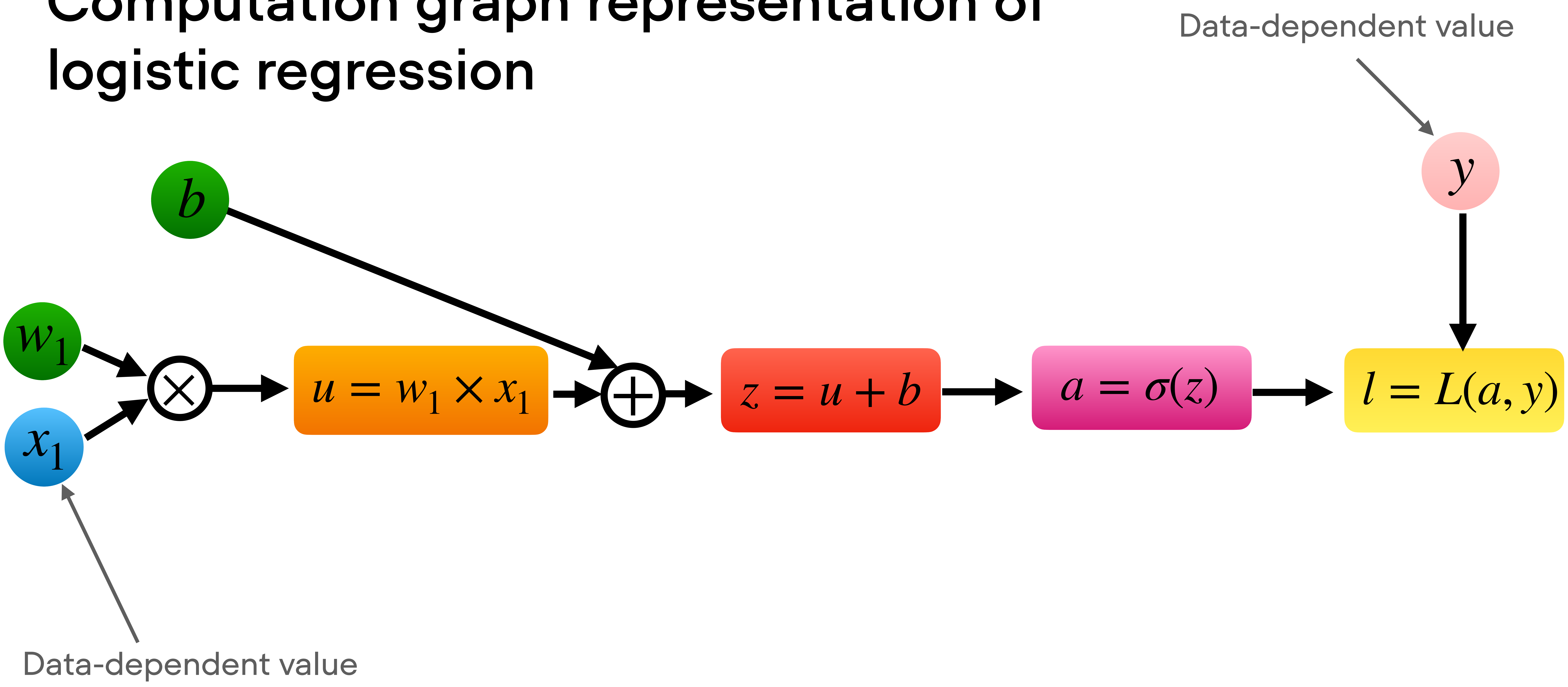
(e.g., Logistic regression)

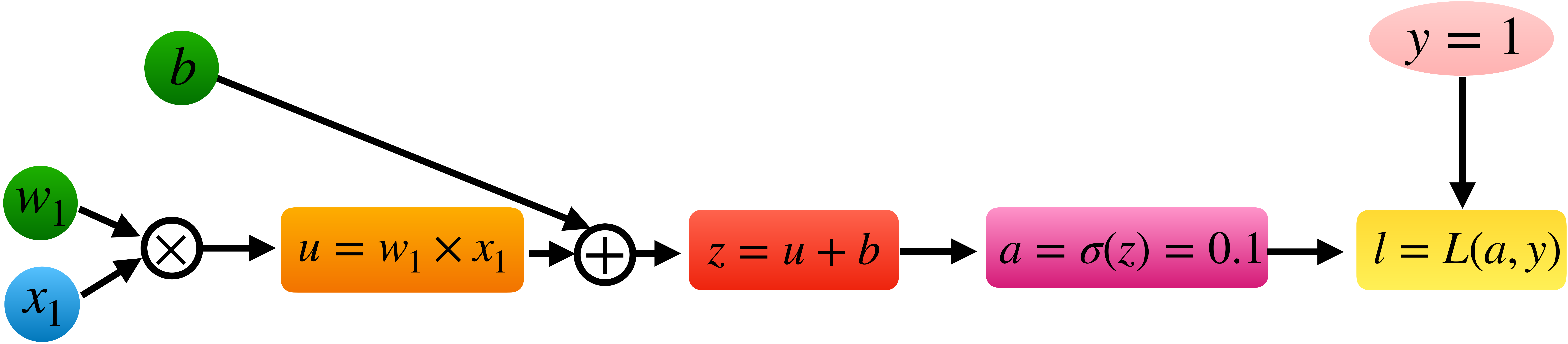


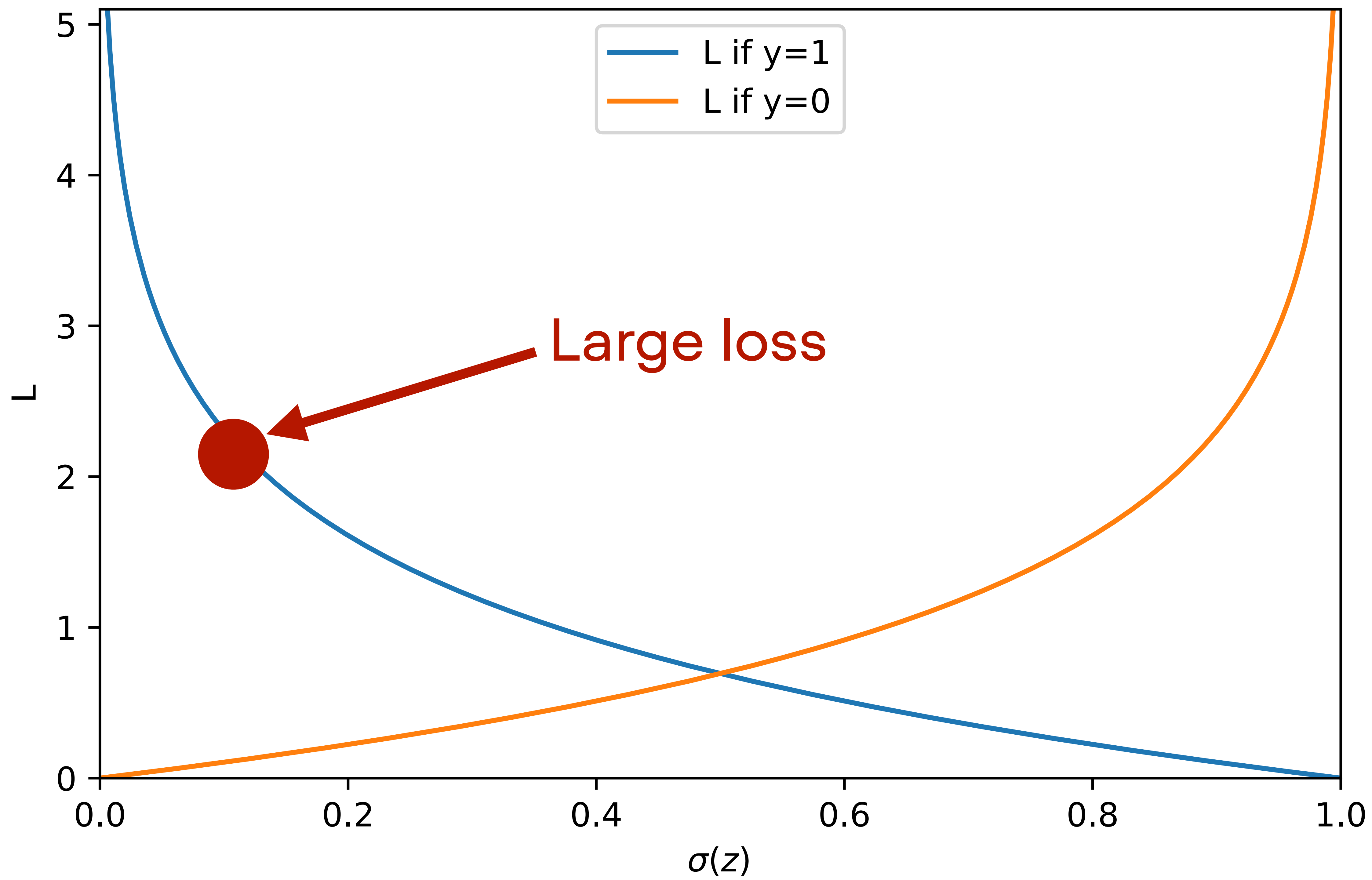
Logistic regression under the hood

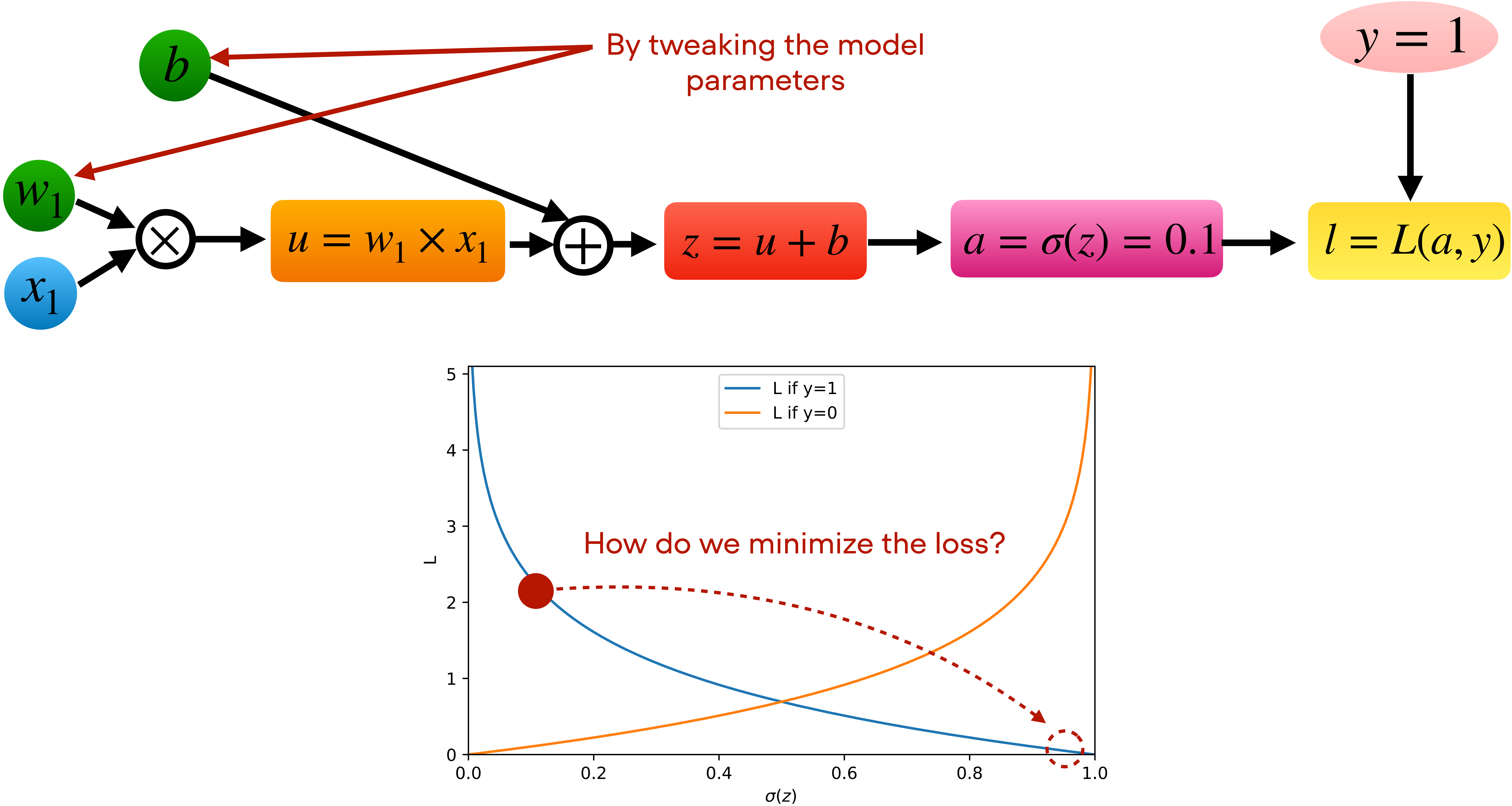


Computation graph representation of logistic regression



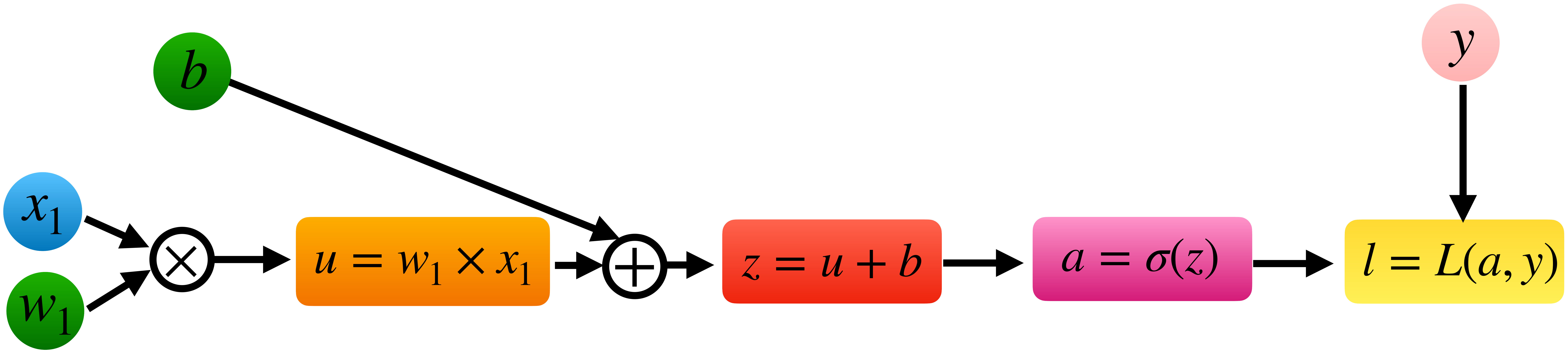

b
w_1
x_1
$u = w_1 \times x_1$
$z = u + b$
$a = \sigma(z) = 0.1$
$y = 1$
$l = L(a, y)$


L if y=1
L if y=0
Large loss
L
$\sigma(z)$
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0


By tweaking the model parameters
b
w_1
x_1
$u = w_1 \times x_1$
$z = u + b$
$a = \sigma(z) = 0.1$
$y = 1$
$l = L(a, y)$
How do we minimize the loss?
L if y=1
L if y=0
L
$\sigma(z)$
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5

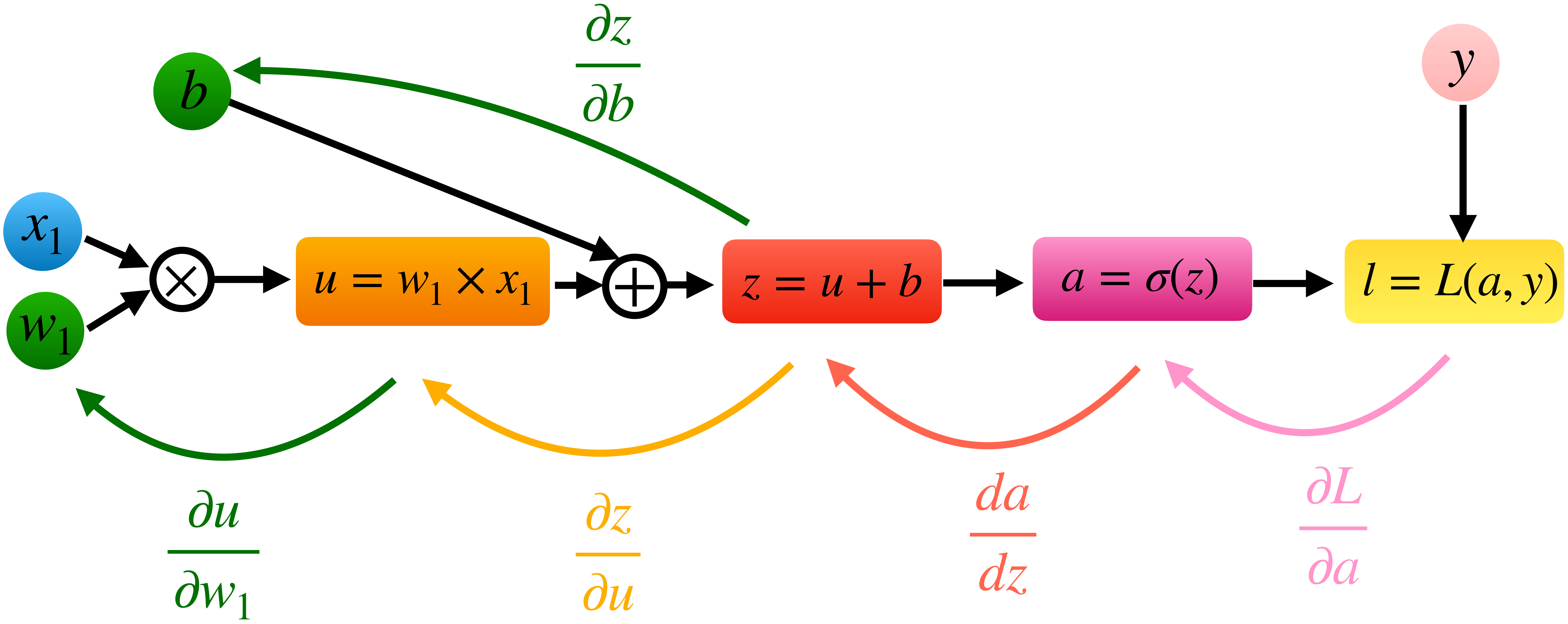
Chain rule:

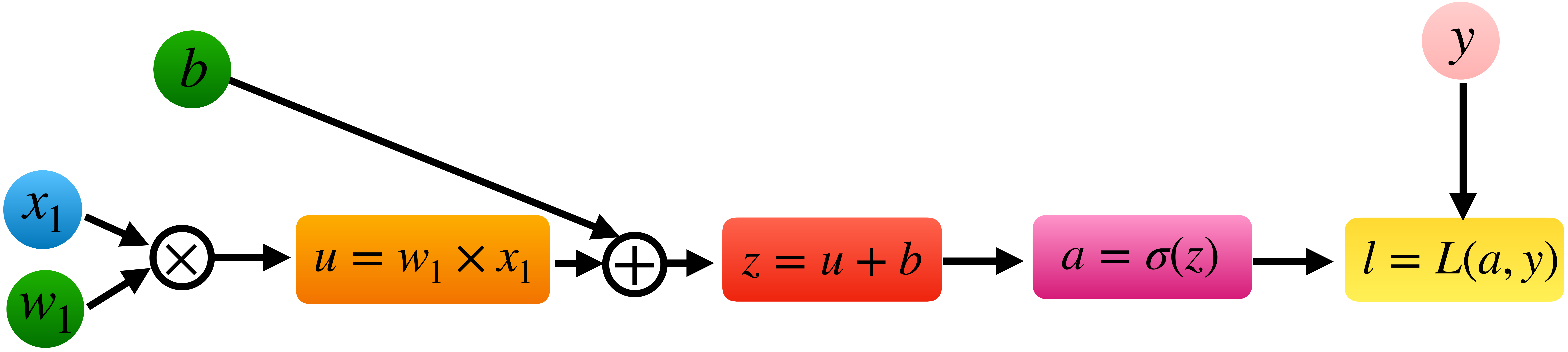
Calculate partial derivatives (gradients) to tweak model parameters)



$$\frac{\partial L}{\partial w_1} = \frac{\partial u}{\partial w_1} \times \frac{\partial z}{\partial u} \times \frac{da}{dz} \times \frac{\partial L}{\partial a}$$

Chain Rule


$\frac{\partial z}{\partial b}$
b
y
x_1
w_1
$u = w_1 \times x_1$
$z = u + b$
$a = \sigma(z)$
$l = L(a, y)$
$\frac{\partial u}{\partial w_1}$
$\frac{\partial z}{\partial u}$
$\frac{da}{dz}$
$\frac{\partial L}{\partial a}$



$$\frac{\partial L}{\partial w_1} =$$

```python
from torch.autograd import grad

grad_L_w1 = grad(l, w_1, retain_graph=True)
print(grad_L_w1)
```

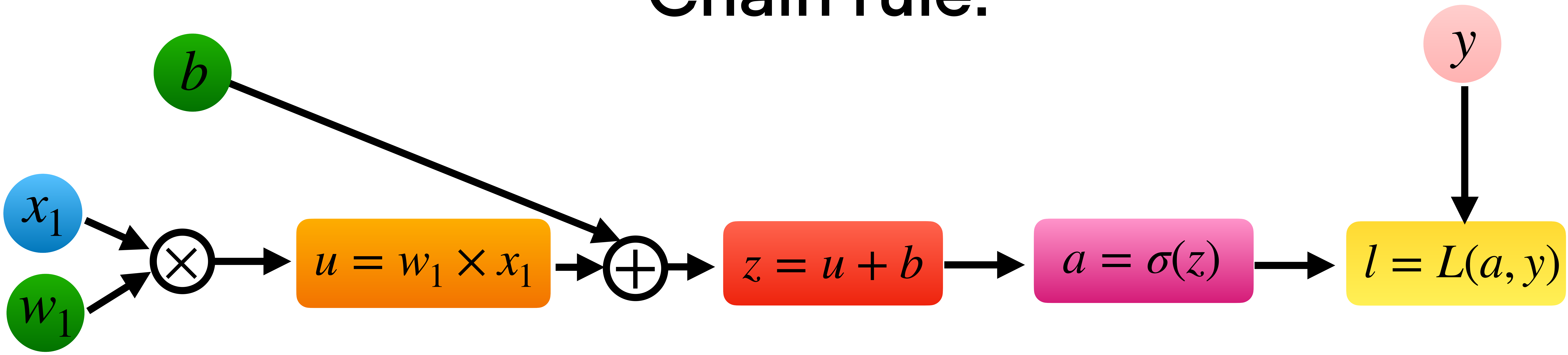
```
(tensor([-0.4987]),)
```

$$\frac{\partial L}{\partial b} =$$

```python
grad_L_b = grad(l, b, retain_graph=True)
print(grad_L_b)
```

```
(tensor([-0.4054]),)
```

Chain rule:



$$\frac{\partial L}{\partial w_1} = \frac{\partial u}{\partial w_1} \times \frac{\partial z}{\partial u} \times \frac{da}{dz} \times \frac{\partial L}{\partial a}$$

$$\frac{\partial L}{\partial w_1} =$$

```
from torch.autograd import grad

grad_L_w1 = grad(l, w_1, retain_graph=True)
print(grad_L_w1)
```

```
(tensor([-0.4987]),)
```

$$\frac{\partial L}{\partial w_1} =$$

```python
from torch.autograd import grad

grad_L_w1 = grad(l, w_1, retain_graph=True)
print(grad_L_w1)
```

```
(tensor([-0.4987]),)
```

Or even easier: use .backward()

$$\frac{\partial L}{\partial w_1} =$$

```python
l.backward()
```

```python
w_1.grad
```

```
tensor([-0.4987])
```

1 Tensor library

2 Automatic differentiation engine

What is PyTorch?

3 Deep learning library

1) Define the Model

Inherit many convenience features

```python
import torch

class MyClassifier(torch.nn.Module):
    def __init__(self):
        # define model parameters

    def forward(self, x):
        # define how the model
        # produces outputs

        return outputs
```

2) Train the Model

```
model = MyClassifier()
optimizer = torch.optim.SGD(...)

for epoch in range(num_epochs):
    for x, y in train_dataloader:

        # forward pass
        outputs = model(x)
        loss = loss_fn(outputs, y)

        # backward pass (backpropagation)
        optimizer.zero_grad()
        loss.backward()

        # update model parameters
        optimizer.step()
```

2) Train the Model

Instantiate the classifier and optimizer

```python
model = MyClassifier()
optimizer = torch.optim.SGD(...)

for epoch in range(num_epochs):
    for x, y in train_dataloader:

        # forward pass
        outputs = model(x)
        loss = loss_fn(outputs, y)

        # backward pass (backpropagation)
        optimizer.zero_grad()
        loss.backward()

        # update model parameters
        optimizer.step()
```

2) Train the Model

```python
model = MyClassifier()
optimizer = torch.optim.SGD(...)

for epoch in range(num_epochs):
    for x, y in train_dataloader:

        # forward pass
        outputs = model(x)
        loss = loss_fn(outputs, y)

        # backward pass (backpropagation)
        optimizer.zero_grad()
        loss.backward()

        # update model parameters
        optimizer.step()
```

Iterate over the training dataset

2) Train the Model

Predict outputs and compute the loss

```
model = MyClassifier()
optimizer = torch.optim.SGD(...)

for epoch in range(num_epochs):
    for x, y in train_dataloader:

        # forward pass
        outputs = model(x)
        loss = loss_fn(outputs, y)

        # backward pass (backpropagation)
        optimizer.zero_grad()
        loss.backward()

        # update model parameters
        optimizer.step()
```

2) Train the Model

```python
model = MyClassifier()
optimizer = torch.optim.SGD(...)

for epoch in range(num_epochs):
    for x, y in train_dataloader:

        # forward pass
        outputs = model(x)
        loss = loss_fn(outputs, y)

        # backward pass (backpropagation)
        optimizer.zero_grad()
        loss.backward()

        # update model parameters
        optimizer.step()
```

Reset and calculate gradients

2) Train the Model

```python
model = MyClassifier()
optimizer = torch.optim.SGD(...)

for epoch in range(num_epochs):
    for x, y in train_dataloader:

        # forward pass
        outputs = model(x)
        loss = loss_fn(outputs, y)

        # backward pass (backpropagation)
        optimizer.zero_grad()
        loss.backward()

        # update model parameters
        optimizer.step()
```

Update model parameters

Exercise

Implementing a logistic regression classifier in PyTorch

6) The training loop

```python
import torch.nn.functional as F


torch.manual_seed(1)
model = LogisticRegression(num_features=2, num_classes=2)
optimizer = torch.optim.SGD(model.parameters(), lr=0.5)

num_epochs = 20

for epoch in range(num_epochs):

    model = model.train()
    for batch_idx, (features, class_labels) in enumerate(train_loader):


        ###############################
        ### Complete the training loop
        ###
        ### Your code below
        ###############################

        outputs = # ?????

        loss = F.cross_entropy(outputs, class_labels)

        # ?????
        # ?????
        # ?????

        ################################
        ## No changes necessary below
        ################################

        ### LOGGING
        print(f'Epoch: {epoch+1:03d}/{num_epochs:03d}'
              f' | Batch {batch_idx:03d}/{len(train_loader):03d}'
              f' | Loss: {loss:.2f}')
```