



# STAT 453: Introduction to Deep Learning and Generative Models

---

Ben Lengerich

Lecture 06: Automatic Differentiation with PyTorch

September 22, 2025



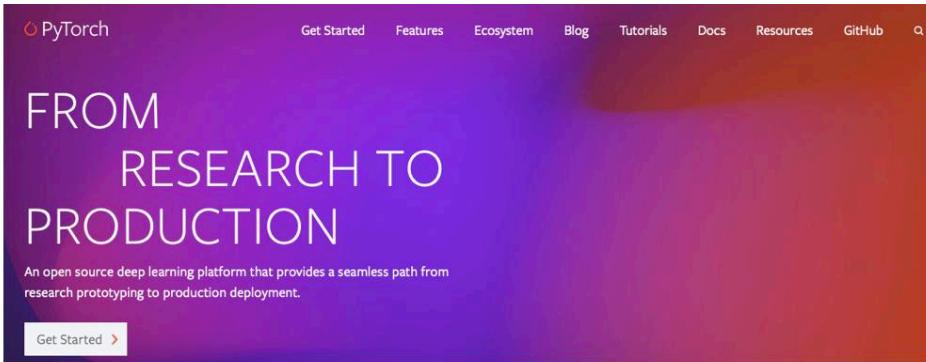
# Today: Computing partial derivatives with PyTorch

---

1. PyTorch Resources
2. Computation Graphs
3. Automatic Differentiation in PyTorch
4. A Closer Look at the PyTorch API



# PyTorch



<https://pytorch.org/>

## At a Glance:

- Based on Torch 7, which was based on Lua and inspired by Lush
- PyTorch started in 2016
- Focuses on flexibility and minimizing cognitive overhead
- Dynamic nature of autograd API inspired by Chainer
- Core features
  - **Automatic differentiation**
  - **Dynamic computation graphs**
  - NumPy integration
- written in C++ and CUDA (CUDA is like C++ for the GPU)
- Python is the usability glue



# Installing PyTorch

## Recommendation for Laptop (e.g., MacBook)

PyTorch Build	Stable (1.7.1)		Preview (Nightly)		
Your OS	Linux	Mac	Windows		
Package	Conda	Pip	LibTorch	Source	
Language	Python		C++ / Java		
CUDA	9.2	10.1	10.2	11.0	None
Run this Command:	<b>NOTE:</b> Python 3.9 users will need to add '-c=conda-forge' for installation conda install pytorch torchvision torchaudio -c pytorch				

## Recommendation for Desktop (Linux) with GPU

PyTorch Build	Stable (1.7.1)		Preview (Nightly)		
Your OS	Linux	Mac	Windows		
Package	Conda	Pip	LibTorch	Source	
Language	Python		C++ / Java		
CUDA	9.2	10.1	10.2	11.0	None
Run this Command:	<b>NOTE:</b> Python 3.9 users will need to add '-c=conda-forge' for installation conda install pytorch torchvision torchaudio cudatoolkit=11.0 -c pytorch				

<https://pytorch.org/>

And don't forget that you import PyTorch as "import torch," not "import pytorch" :)

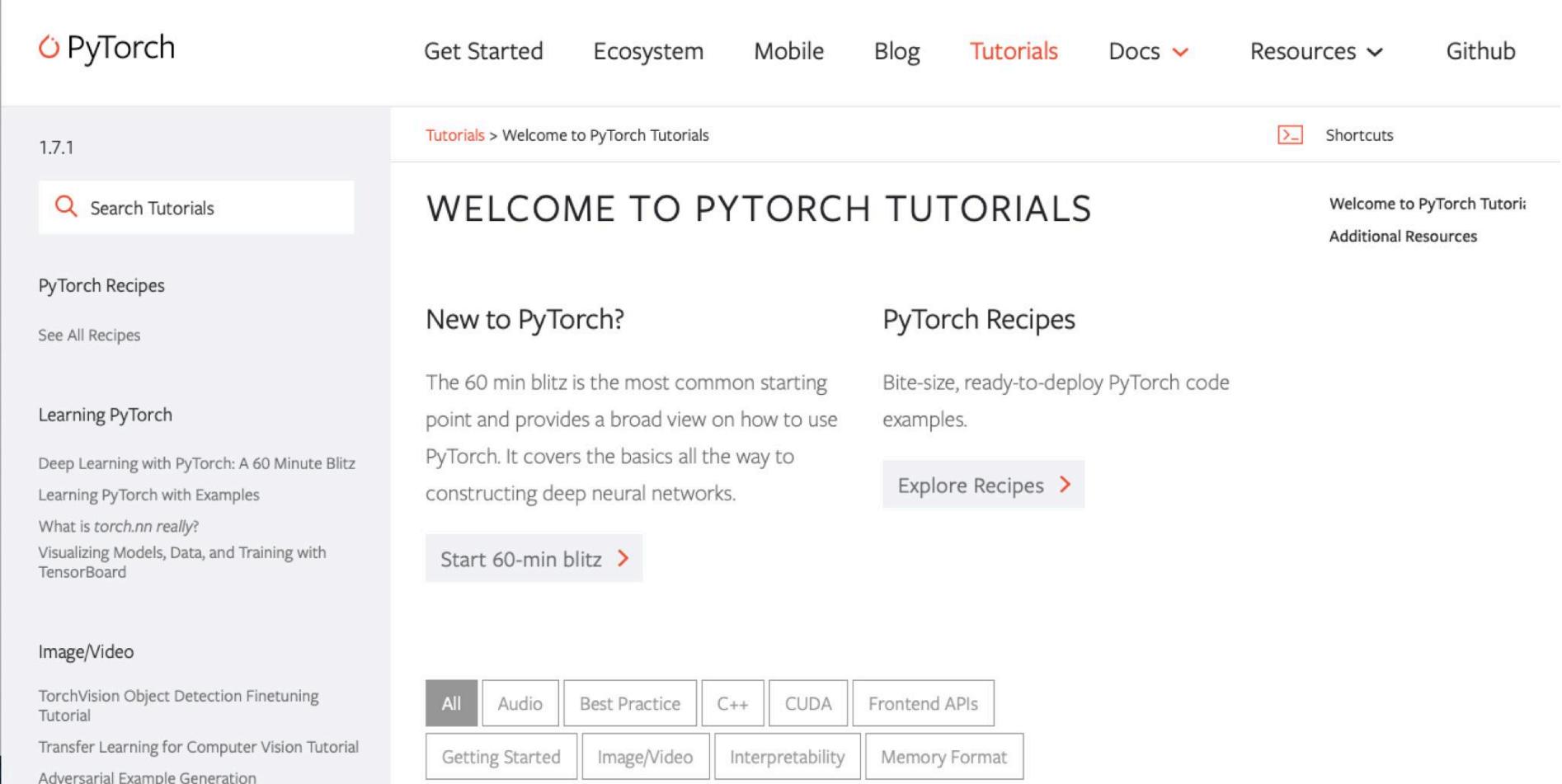
```
[In [1]: import torch

[In [2]: torch.__version__
Out[2]: '1.7.0'

In [3]: ]
```



# Many useful tutorials (recommend you read some)

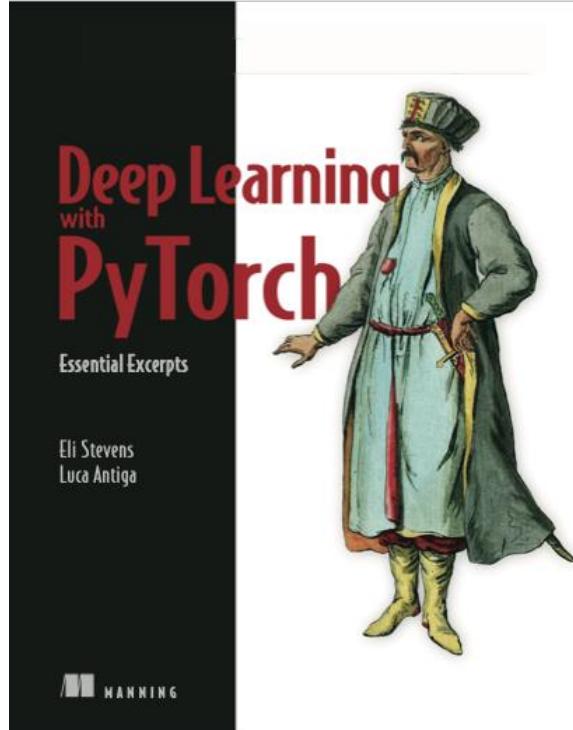


The screenshot shows the PyTorch Tutorials homepage. At the top, there is a navigation bar with links: Get Started, Ecosystem, Mobile, Blog, Tutorials (which is highlighted in red), Docs, Resources, and Github. Below the navigation bar, the version 1.7.1 is displayed. On the left side, there is a sidebar with sections for PyTorch Recipes (Search Tutorials, See All Recipes), Learning PyTorch (Deep Learning with PyTorch: A 60 Minute Blitz, Learning PyTorch with Examples, What is torch.nn really?, Visualizing Models, Data, and Training with TensorBoard), and Image/Video (TorchVision Object Detection Finetuning Tutorial, Transfer Learning for Computer Vision Tutorial, Adversarial Example Generation). The main content area features a "WELCOME TO PYTORCH TUTORIALS" heading, a "New to PyTorch?" section with a "Start 60-min blitz >" button, a "PyTorch Recipes" section with a "Explore Recipes >" button, and a footer with categories: All, Audio, Best Practice, C++, CUDA, Frontend APIs, Getting Started, Image/Video, Interpretability, and Memory Format.

<https://pytorch.org/tutorials/>



# Other resources



PyTorch

Do you want live notifications when people reply to your posts? [Enable Notifications](#)

all categories > all Latest New (47) Unread (104) Top Categories + New Topic

Topic	Replies	Views	Activity
Using MSELoss instead of CrossEntropy for Ordinal Regression/Classification	2	83	1h
Optimizer.load_state_dict() weird behaviour with Adam optimizer	7	2.0k	1h
Is there a way to train 3 dataloaders using multiprocessing?	0	11	2h
Getting different feature vectors from frozen layers after training	5	86	2h
Libtorch_cuda.so is too large (>2GB)	22	346	2h
Undo pruning - How to 'unmask' pruned weights	0	8	2h
If input.dim() == 2 and bias is not None: AttributeError: 'tuple' object has no attribute 'dim'	3	41	2h
Export unsupported/compound ops to ONNX	0	9	2h

<https://discuss.pytorch.org>

And...

Ask ChatGPT/Claude if your PyTorch code is not working 😊



# Today: Computing partial derivatives with PyTorch

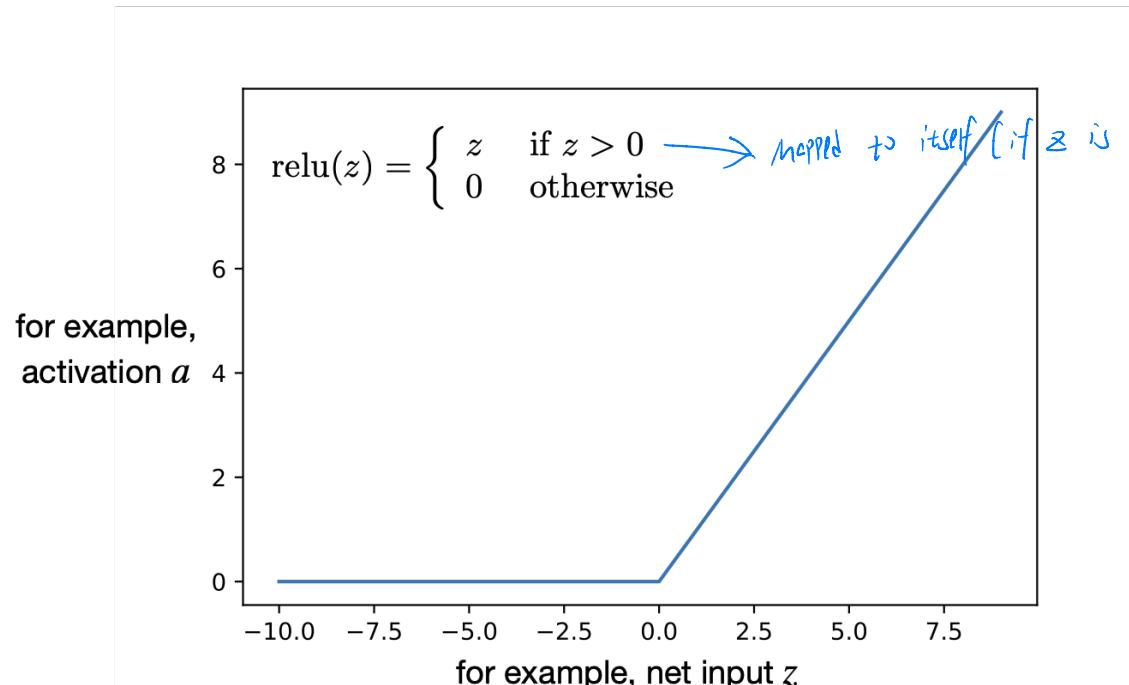
---

1. PyTorch Resources
2. Computation Graphs
3. Automatic Differentiation in PyTorch
4. A Closer Look at the PyTorch API

# Computation graphs: ReLU

Suppose we have the following activation function:

$$a(x, w, b) = \text{relu}(w \cdot x + b)$$



**ReLU = Rectified Linear Unit**

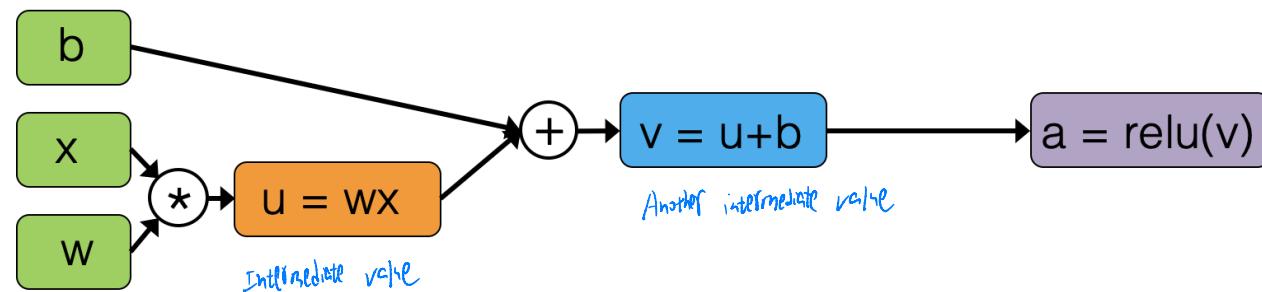
(prob. the most commonly used activation function in DL)

# Computation graphs: ReLU

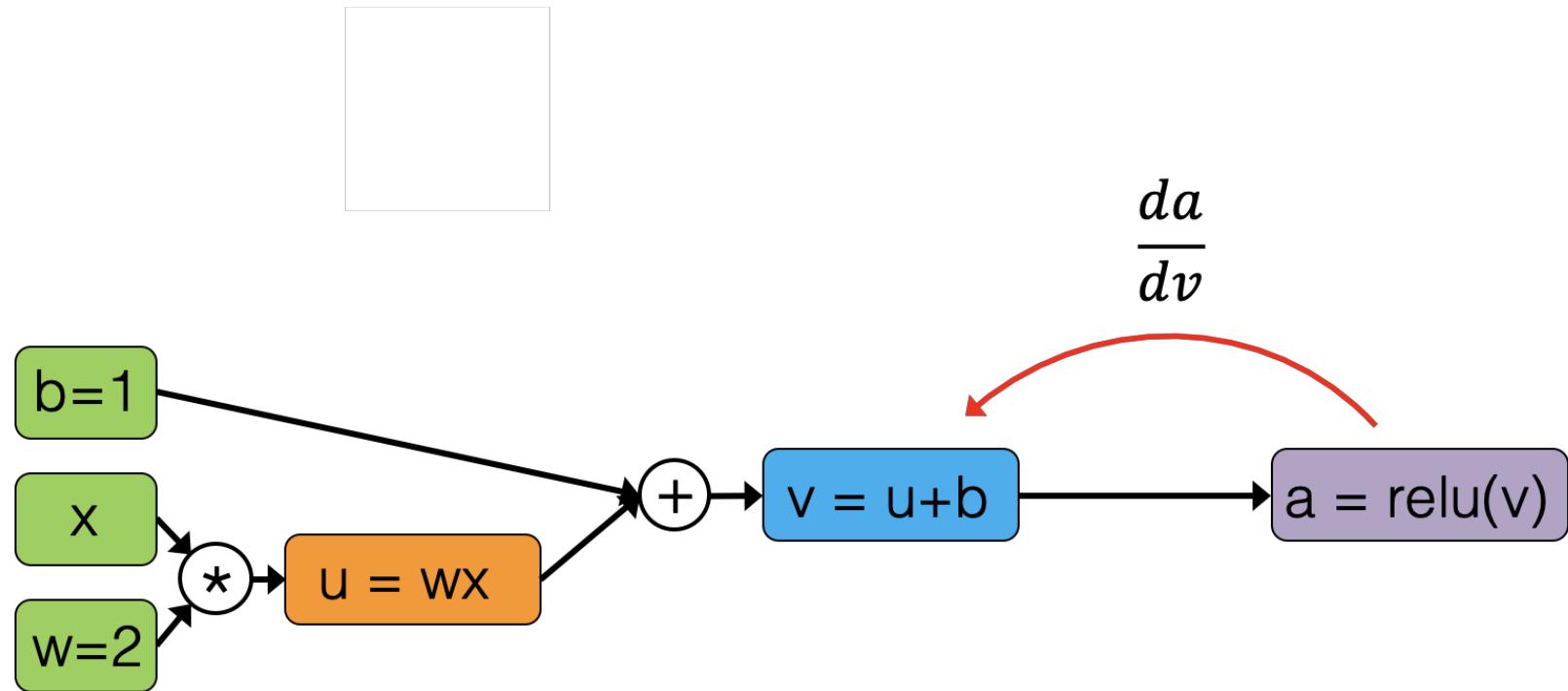
↓  
Directed

$$a(x, w, b) = \text{relu}(w \cdot x + b)$$

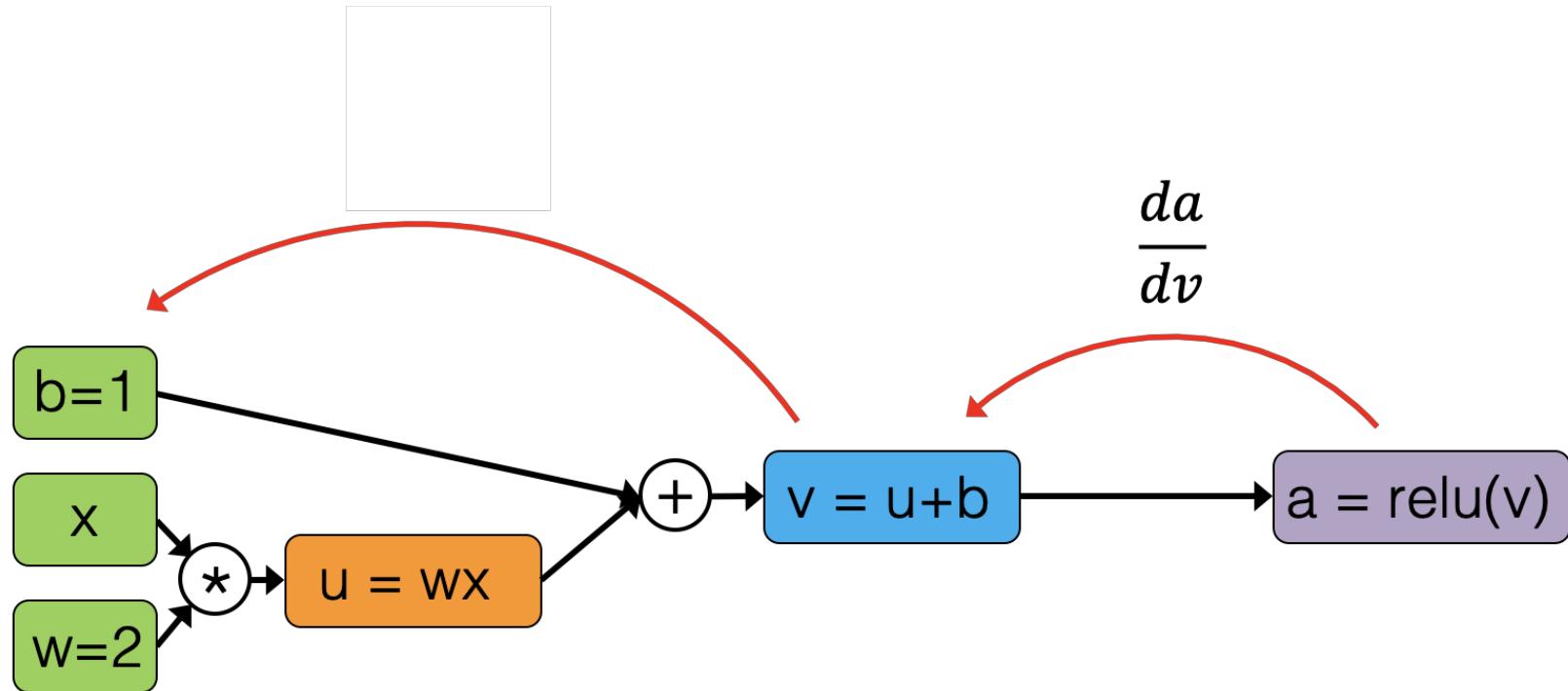
$$\begin{matrix} & u \\ & \downarrow \\ v & \end{matrix}$$



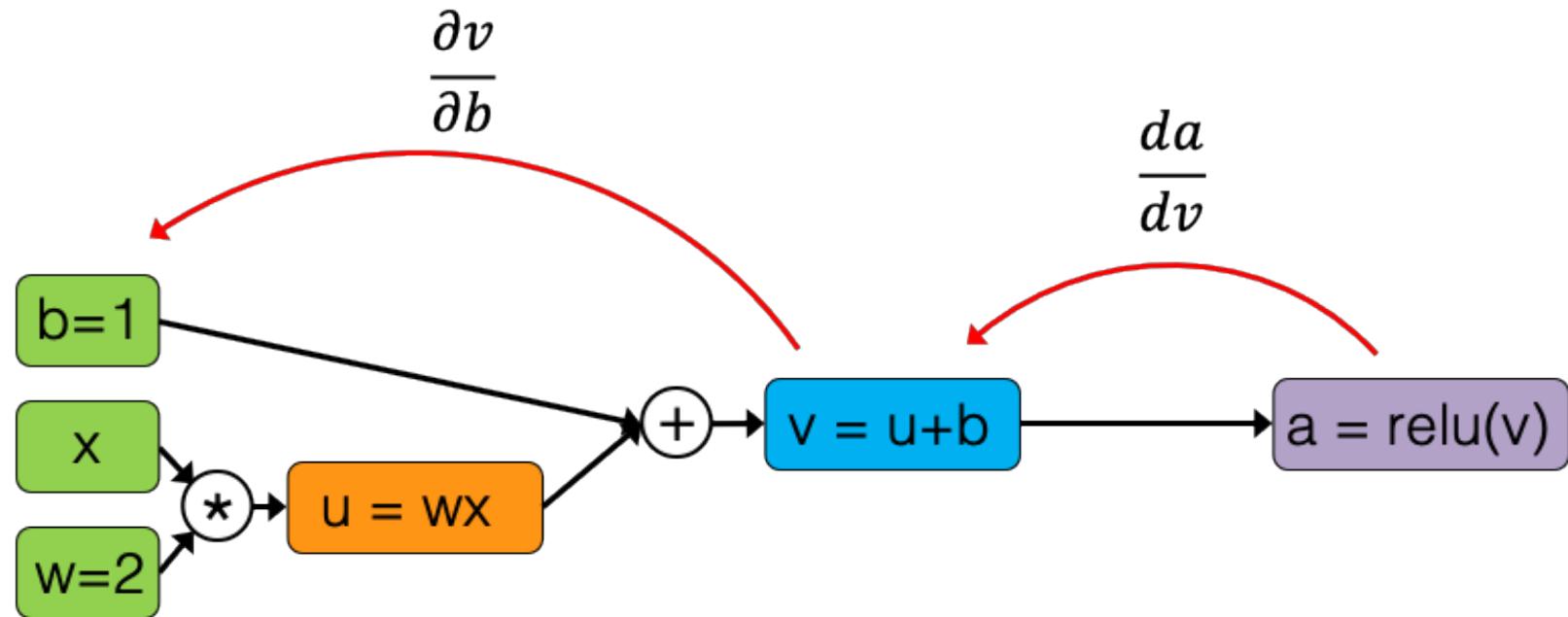
# Computation graphs: ReLU



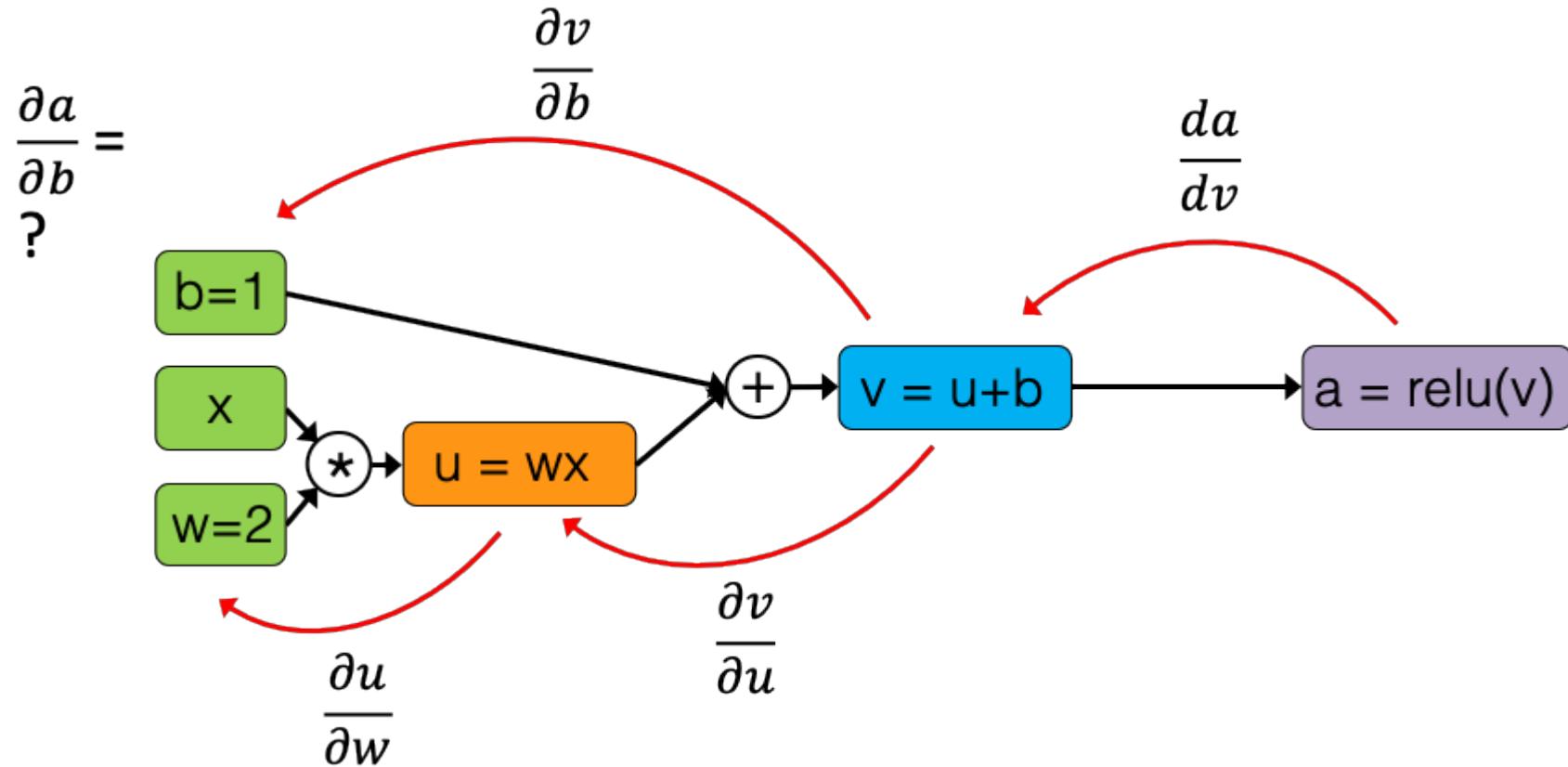
# Computation graphs: ReLU



# Computation graphs: ReLU

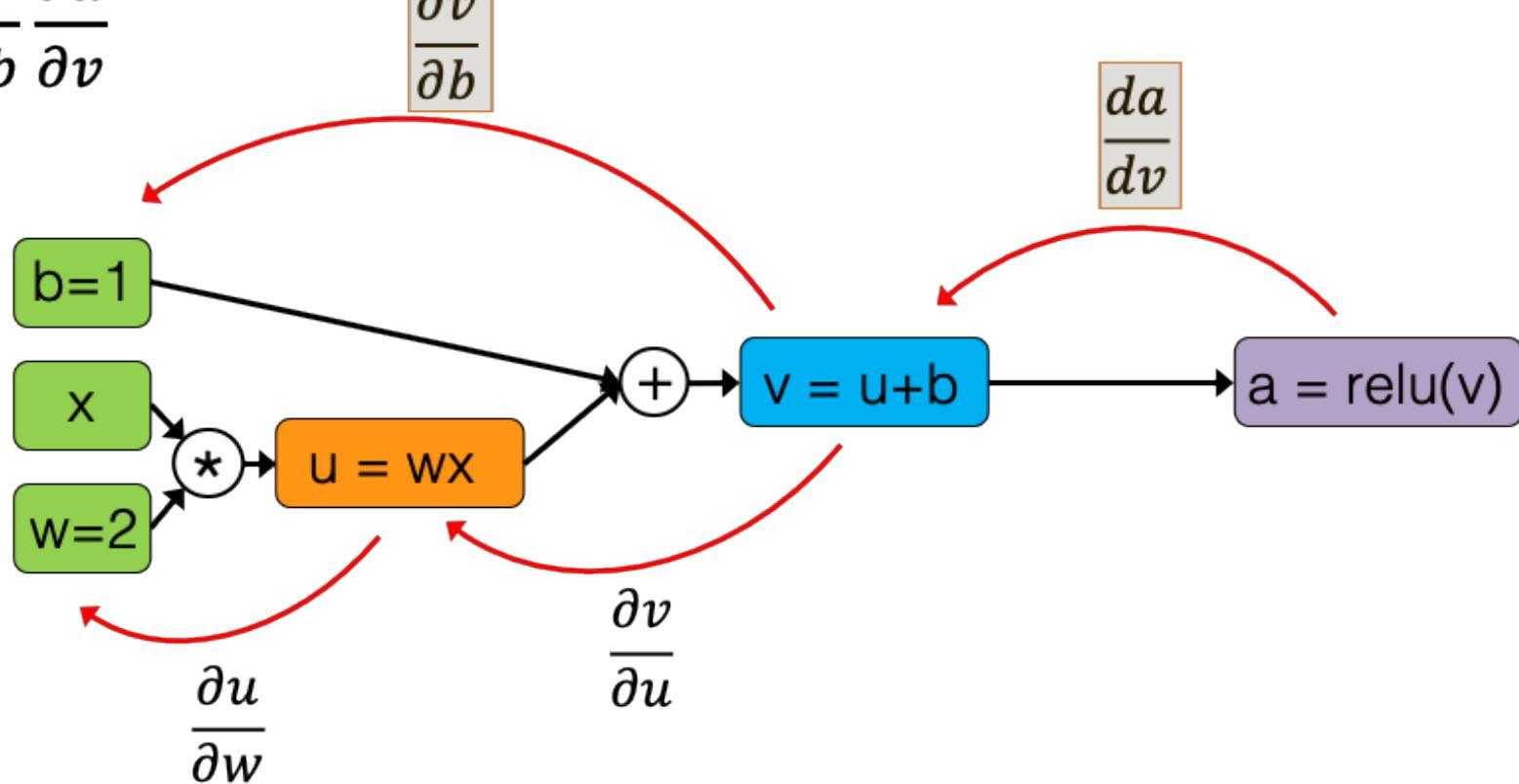


# Computation graphs: ReLU



# Computation graphs: ReLU

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v}$$



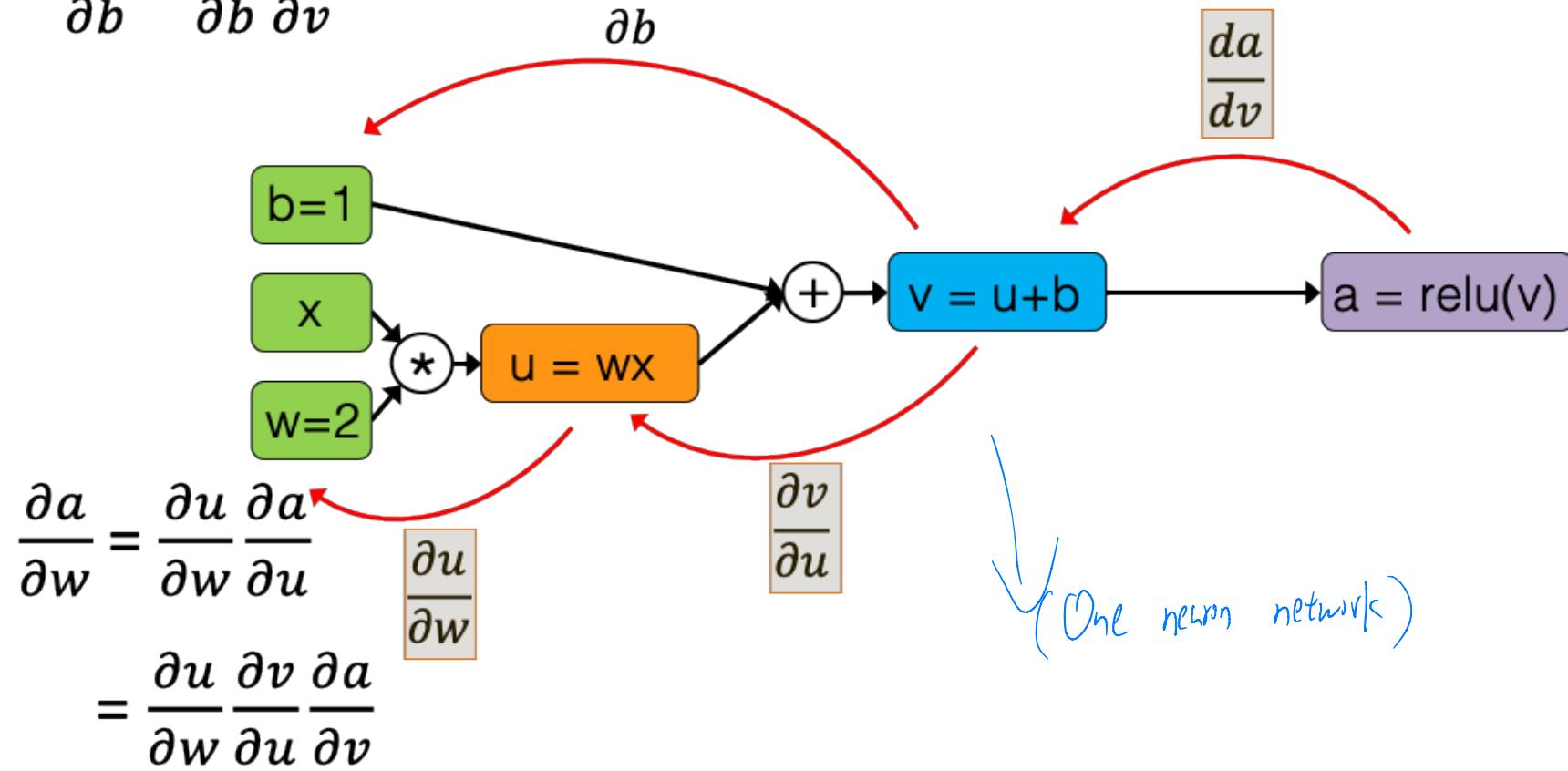
# Computation graphs: ReLU

---

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v}$$

$$\frac{\partial v}{\partial b}$$

$$\frac{da}{dv}$$



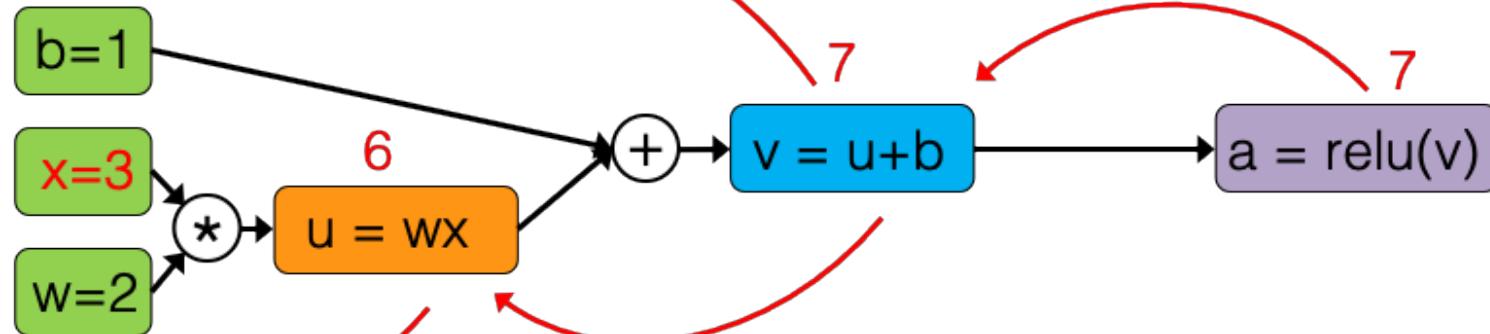
# Computation graphs: ReLU

---

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v}$$

$$\frac{\partial v}{\partial b}$$

$$\frac{da}{dv}$$



$$\frac{\partial a}{\partial w} = \frac{\partial u}{\partial w} \frac{\partial a}{\partial u}$$

$$\frac{\partial u}{\partial w}$$

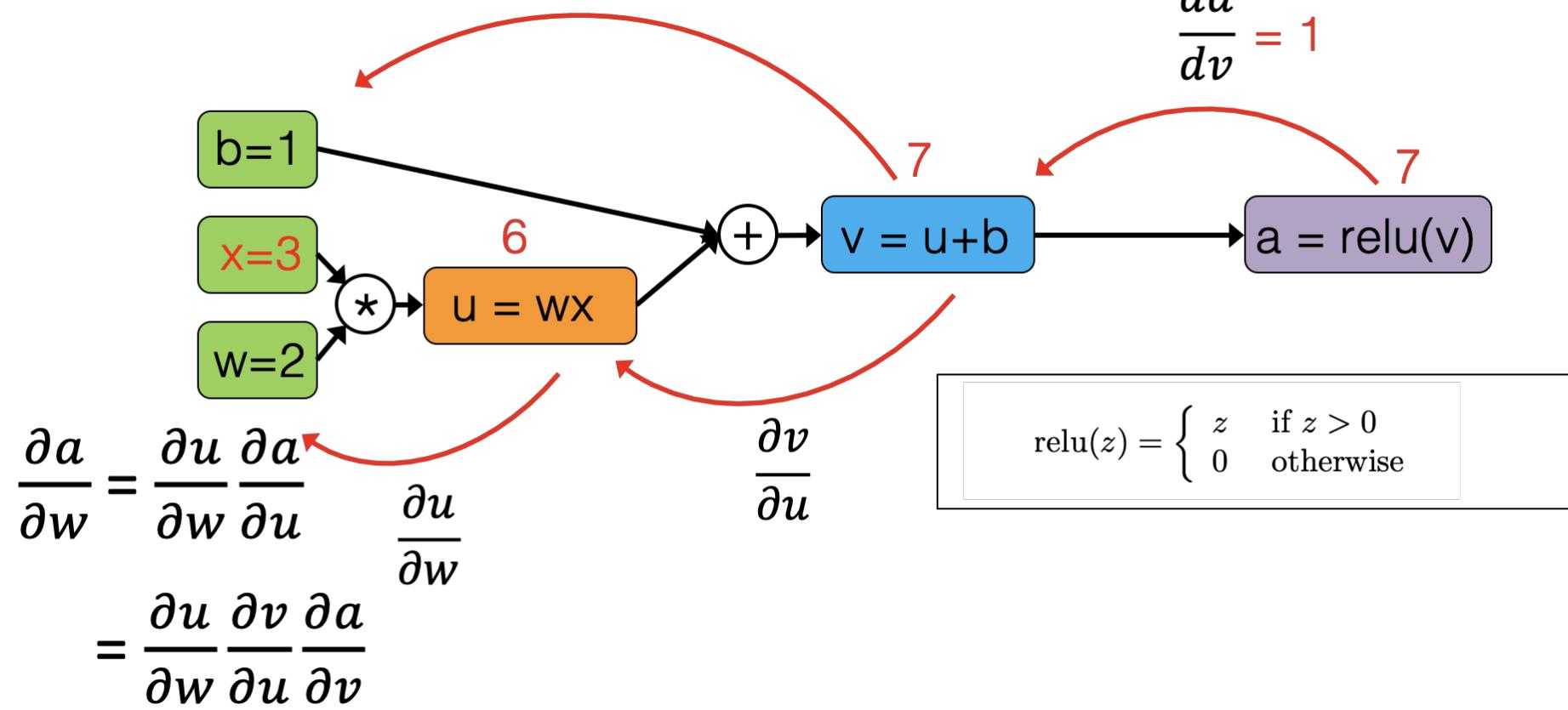
$$= \frac{\partial u}{\partial w} \frac{\partial v}{\partial u} \frac{\partial a}{\partial v}$$

# Computation graphs: ReLU

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v}$$

$$\frac{\partial v}{\partial b}$$

$$\frac{da}{dv} = 1$$

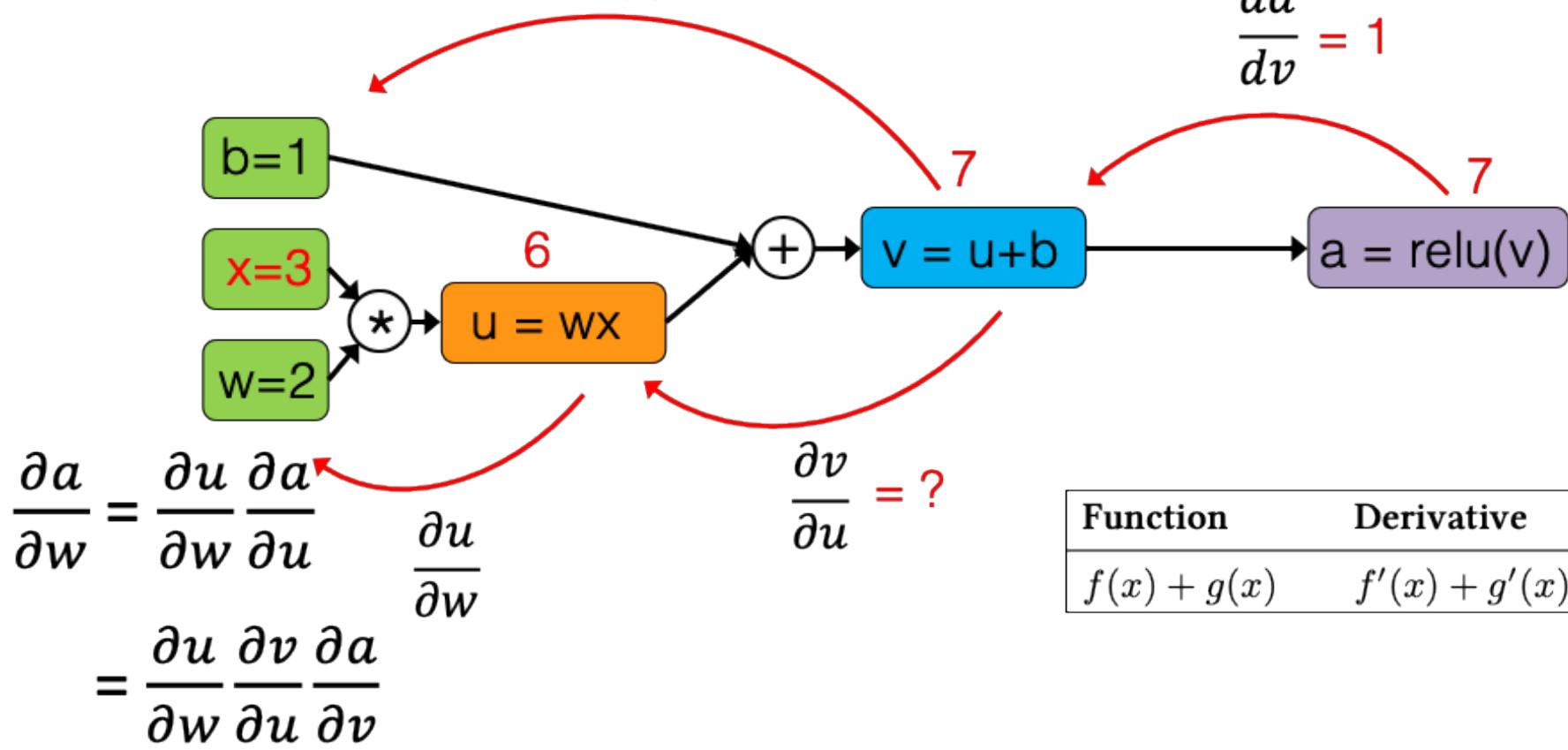


# Computation graphs: ReLU

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v}$$

$$\frac{\partial v}{\partial b} = ?$$

$$\frac{da}{dv} = 1$$



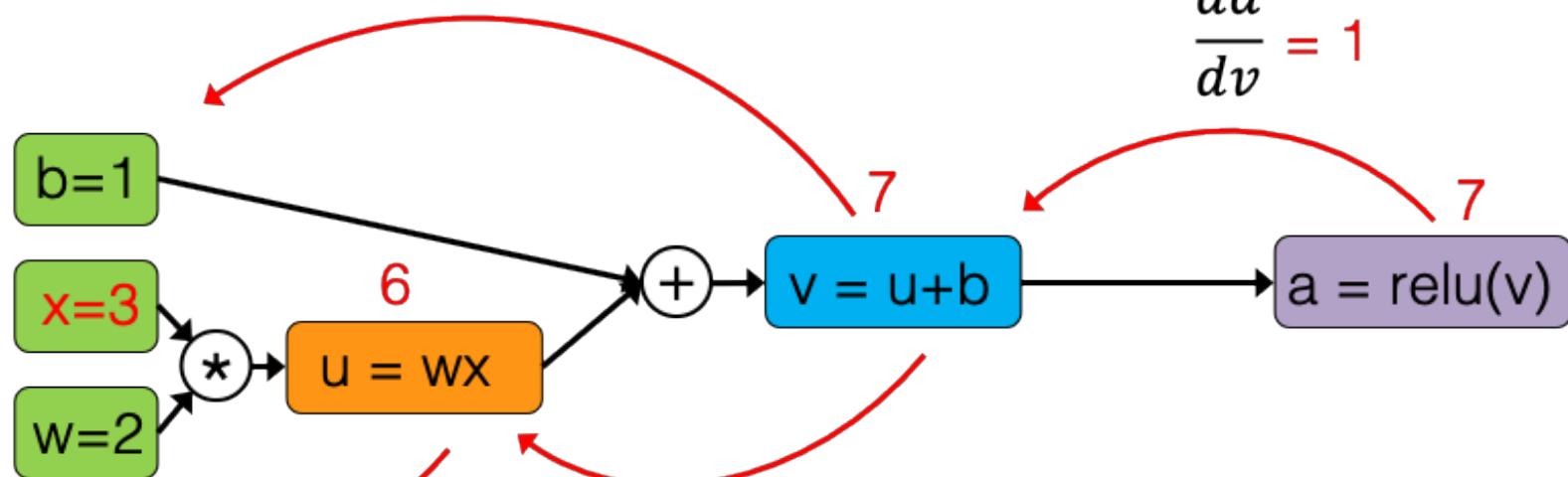
Function	Derivative
$f(x) + g(x)$	$f'(x) + g'(x)$

# Computation graphs: ReLU

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v}$$

$$\frac{\partial v}{\partial b} = 1$$

$$\frac{da}{dv} = 1$$



$$\frac{\partial a}{\partial w} = \frac{\partial u}{\partial w} \frac{\partial a}{\partial u}$$

$$\frac{\partial u}{\partial w} = ?$$

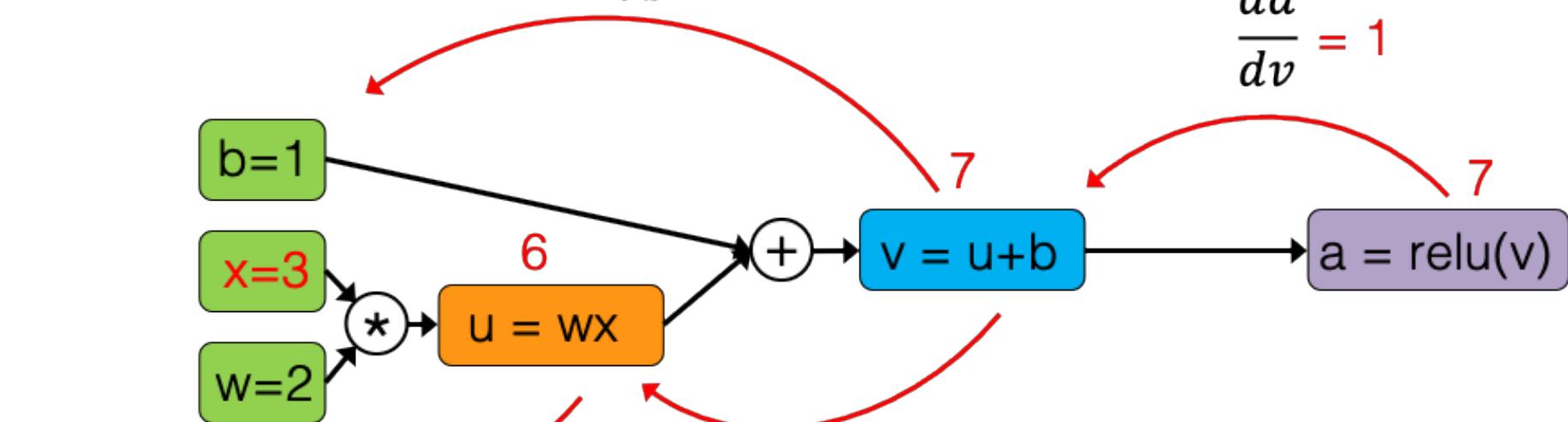
$$= \frac{\partial u}{\partial w} \frac{\partial v}{\partial u} \frac{\partial a}{\partial v}$$

# Computation graphs: ReLU

$$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b} \frac{\partial a}{\partial v} = 1$$

$$\frac{\partial v}{\partial b} = 1$$

$$\frac{da}{dv} = 1$$



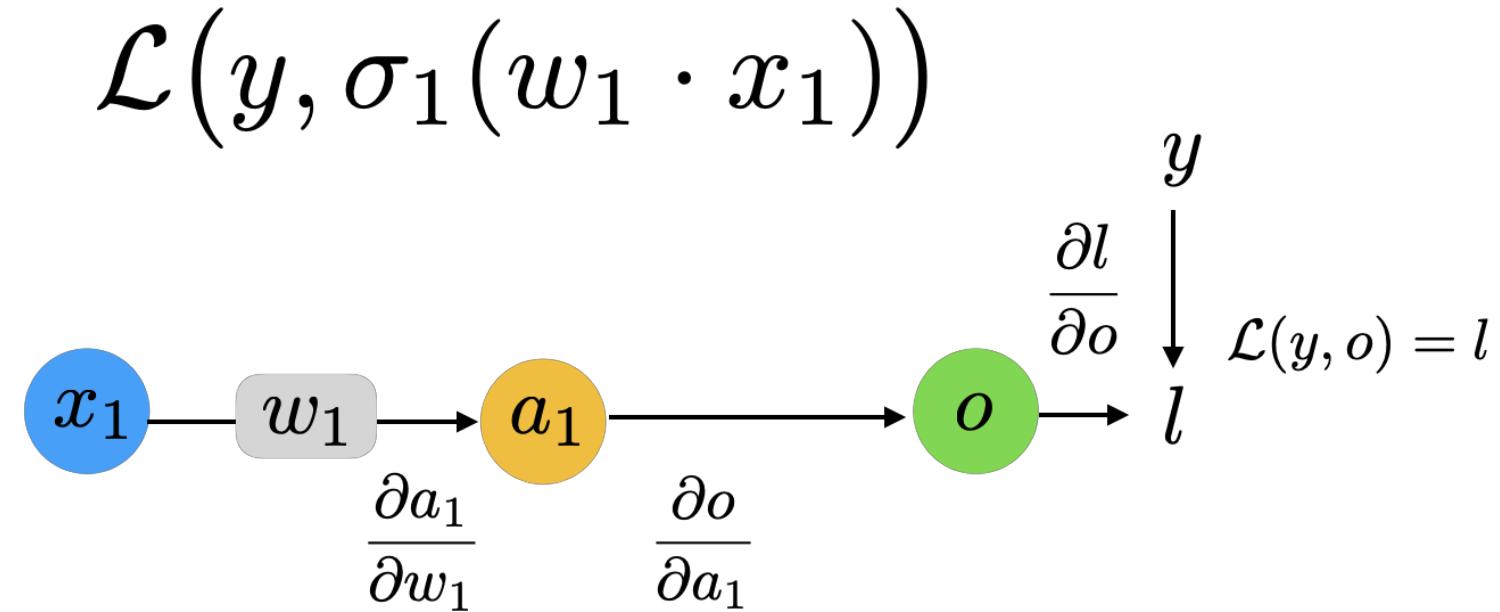
$$\begin{aligned}\frac{\partial a}{\partial w} &= \frac{\partial u}{\partial w} \frac{\partial a}{\partial u} & \frac{\partial u}{\partial w} &= 3 \\ &= \frac{\partial u}{\partial w} \frac{\partial v}{\partial u} \frac{\partial a}{\partial v} & &= 3 * 1 * 1 = 3\end{aligned}$$



- Some more computation graphs

# Computation graphs: Single-path

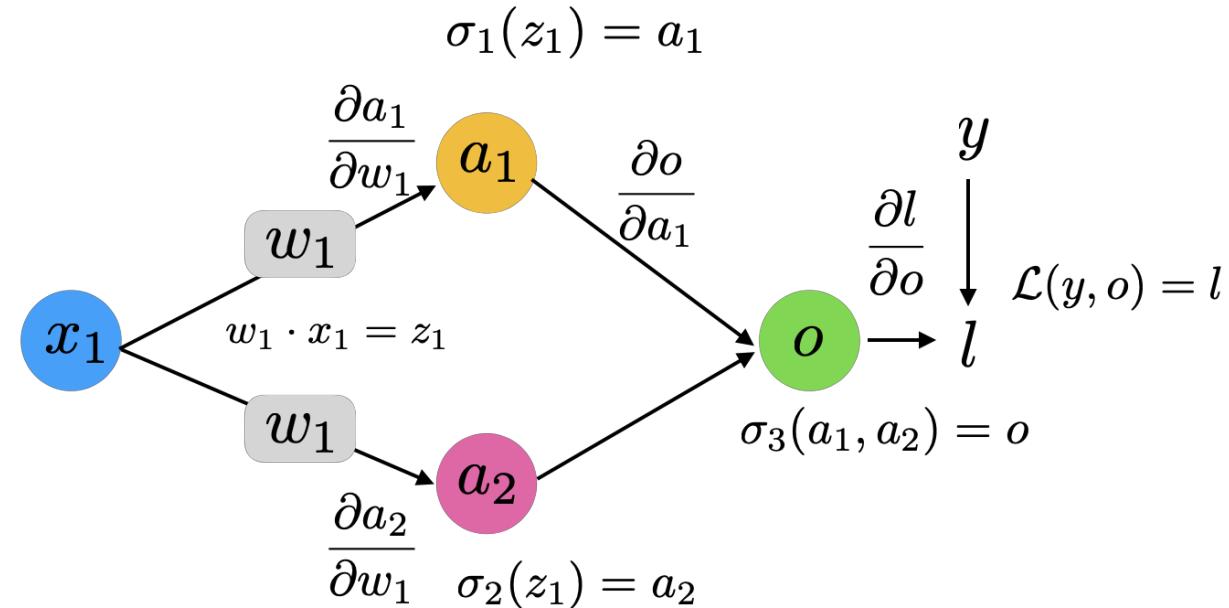
---



$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} \quad (\text{univariate chain rule})$$

# Computation graphs: Weight-Sharing

$$\mathcal{L}(y, \sigma_3[\sigma_1(w_1 \cdot x_1), \sigma_2(w_1 \cdot x_1)])$$



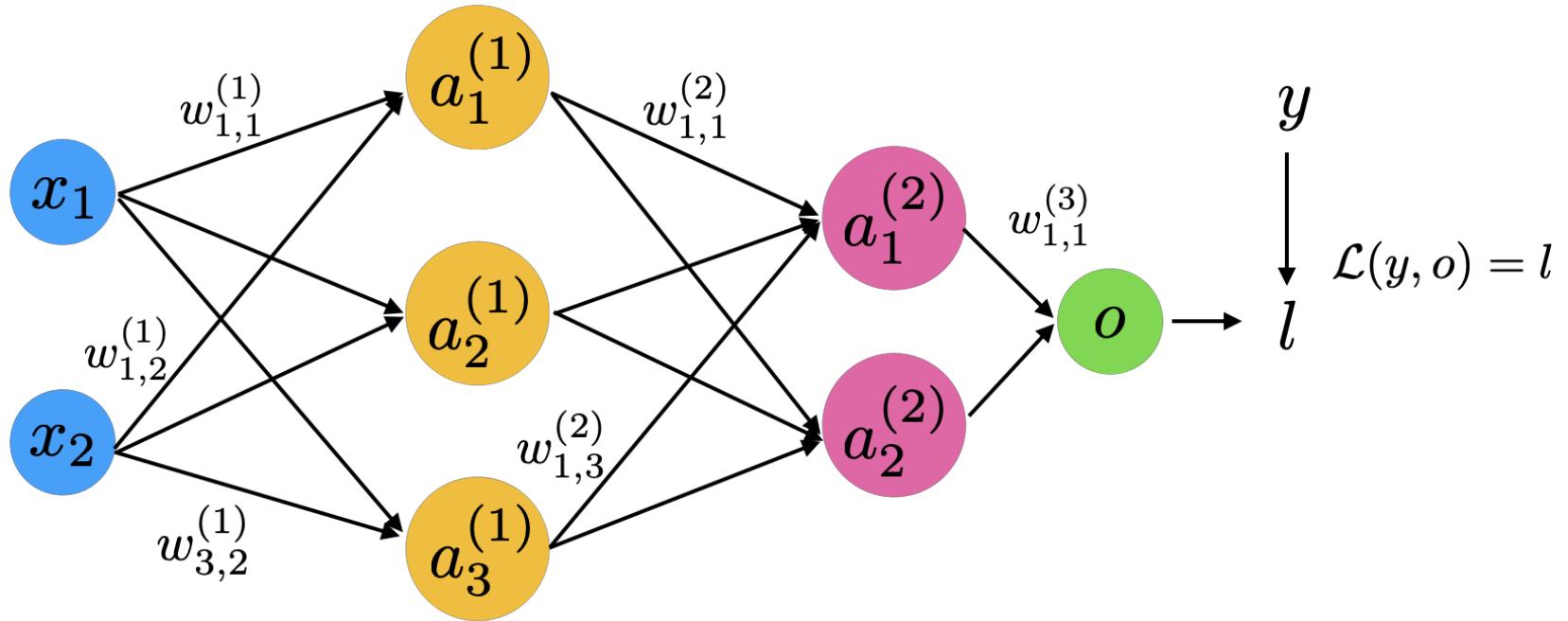
Upper path

$$\frac{\partial l}{\partial w_1} = \underbrace{\frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}}_{\text{Upper path}} + \underbrace{\frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}}_{\text{Lower path}} \quad (\text{multivariable chain rule})$$

Lower path

# Computation graphs: Fully-Connected Layer

---



$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

$$+ \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$



# Today: Computing partial derivatives with PyTorch

---

1. PyTorch Resources
2. Computation Graphs
- 3. Automatic Differentiation in PyTorch**
4. A Closer Look at the PyTorch API



# Automatic Differentiation in PyTorch

---

- An example:

<https://github.com/rasbt/stat453-deep-learning-ss21/tree/master/L06/code/pytorch-autograd.ipynb>



# Today: Computing partial derivatives with PyTorch

---

1. PyTorch Resources
2. Computation Graphs
3. Automatic Differentiation in PyTorch
4. **A Closer Look at the PyTorch API**



# PyTorch Usage: Step 1 (Definition)

```
class MultilayerPerceptron(torch.nn.Module): ←  
  
    def __init__(self, num_features, num_classes):  
        super(MultilayerPerceptron, self).__init__()  
  
        ### 1st hidden layer  
        self.linear_1 = torch.nn.Linear(num_feat, num_h1)  
  
        ### 2nd hidden layer  
        self.linear_2 = torch.nn.Linear(num_h1, num_h2)  
  
        ### Output layer  
        self.linear_out = torch.nn.Linear(num_h2, num_classes)  
  
    def forward(self, x):  
        out = self.linear_1(x)  
        out = F.relu(out)  
        out = self.linear_2(out)  
        out = F.relu(out)  
        logits = self.linear_out(out)  
        probas = F.log_softmax(logits, dim=1)  
        return logits, probas
```

Backward will be inferred automatically if we use the `nn.Module` class!

Define model parameters that will be instantiated when created an object of this class

Define how and it what order the model parameters should be used in the forward pass



# PyTorch Usage: Step 2 (Creation)

```
torch.manual_seed(random_seed)
model = MultilayerPerceptron(num_features=num_features,
                             num_classes=num_classes) | Instantiate model
                                         (creates the model parameters)

model = model.to(device)

optimizer = torch.optim.SGD(model.parameters(),
                           lr=learning_rate) | Define an optimization method
```



# PyTorch Usage: Step 3 (Training)

```
for epoch in range(num_epochs):
    model.train()
    for batch_idx, (features, targets) in enumerate(train_loader):

        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)

        ### FORWARD AND BACK PROP
        logits, probas = model(features)
        cost = F.cross_entropy(probas, targets)
        optimizer.zero_grad()

        cost.backward()

        ### UPDATE MODEL PARAMETERS
        optimizer.step()

        model.eval()
        with torch.no_grad():
            # compute accuracy
```

Run for a specified number of epochs

Iterate over minibatches in epoch

If your model is on the GPU, data should also be on the GPU

y = model(x) calls \_\_call\_\_ and then .forward(), where some extra stuff is done in \_\_call\_\_;  
don't run y = model.forward(x) directly

Gradients at each leaf node are accumulated under the .grad attribute, not just stored. This is why we have to zero them before each backward pass



# PyTorch Usage: Step 3 (Training)

```
for epoch in range(num_epochs):
    model.train()
    for batch_idx, (features, targets) in enumerate(train_loader):

        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)

        ### FORWARD AND BACK PROP
        logits, probas = model(features)           ← This will run the forward() method
        loss = F.cross_entropy(logits, targets)      ← Define a loss function to optimize
        optimizer.zero_grad()                      ← Set the gradient to zero
                                                    (could be non-zero from a previous forward pass)

        loss.backward()

        ### UPDATE MODEL PARAMETERS
        optimizer.step()

        model.eval()
        with torch.no_grad():
            # compute accuracy
```

Compute the gradients, the backward is automatically constructed by "autograd" based on the forward() method and the loss function

Use the gradients to update the weights according to the optimization method (defined on the previous slide)  
E.g., for SGD,  $w := w + \text{learning\_rate} \times \text{gradient}$



# PyTorch Usage: Step 3 (Training)

```
for epoch in range(num_epochs):
    model.train()
    for batch_idx, (features, targets) in enumerate(train_loader):

        features = features.view(-1, 28*28).to(device)
        targets = targets.to(device)

        ### FORWARD AND BACK PROP
        logits, probas = model(features)
        loss = F.cross_entropy(logits, targets)
        optimizer.zero_grad()

        loss.backward()

        ### UPDATE MODEL PARAMETERS
        optimizer.step()

        model.eval()
        with torch.no_grad():
            # compute accuracy
```

For evaluation, set the model to eval mode (will be relevant later when we use DropOut or BatchNorm)

This prevents the computation graph for backpropagation from automatically being build in the background to save memory



# Simple “print” statements don’t work for debugging

```
[7]: model.net

[7]: Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): ReLU(inplace)
    (4): Linear(in_features=256, out_features=10, bias=True)
)

[ ]: If we want to get the output from the 2nd layer during the forward pass, we can register a hook as follows:

[8]: outputs = []
def hook(module, input, output):
    outputs.append(output)

model.net[2].register_forward_hook(hook)

[8]: <torch.utils.hooks.RemovableHandle at 0x7f659c6685c0>

Now, if we call the model on some inputs, it will save the intermediate results in the "outputs" list:

[9]: _ = model(features)

print(outputs)

[tensor([[0.5341, 1.0513, 2.3542, ..., 0.0000, 0.0000, 0.0000],
       [0.0000, 0.6676, 0.6620, ..., 0.0000, 0.0000, 2.4056],
       [1.1520, 0.0000, 0.0000, ..., 2.5860, 0.8992, 0.9642],
       ...,
       [0.0000, 0.1076, 0.0000, ..., 1.8367, 0.0000, 2.5203],
       [0.5415, 0.0000, 0.0000, ..., 2.7968, 0.8244, 1.6335],
       [1.0710, 0.9805, 3.0103, ..., 0.0000, 0.0000, 0.0000]],
      device='cuda:3', grad_fn=<ThresholdBackward1>)]
```

Questions?

