



---

# Introduction to Deep Learning with



## TensorFlow

Sebastian Raschka

# Slides

Speaker Deck:

<https://speakerdeck.com/rasbt/introduction-to-deep-learning-with-tensorflow-at-pydata-ann-arbor>

# Code snippets

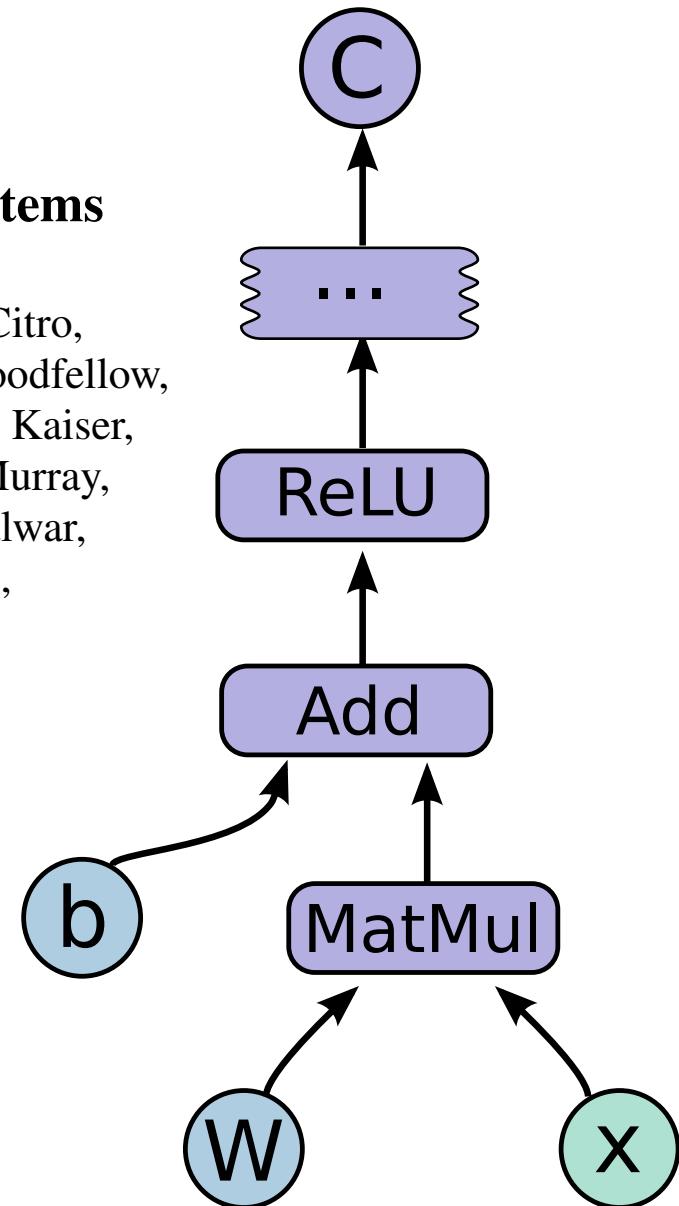
GitHub:

<https://github.com/rasbt/pydata-annarbor2017-dl-tutorial>

**TensorFlow:**  
**Large-Scale Machine Learning on Heterogeneous Distributed Systems**  
(Preliminary White Paper, November 9, 2015)

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng  
Google Research\*

<https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf>



# Tensors?

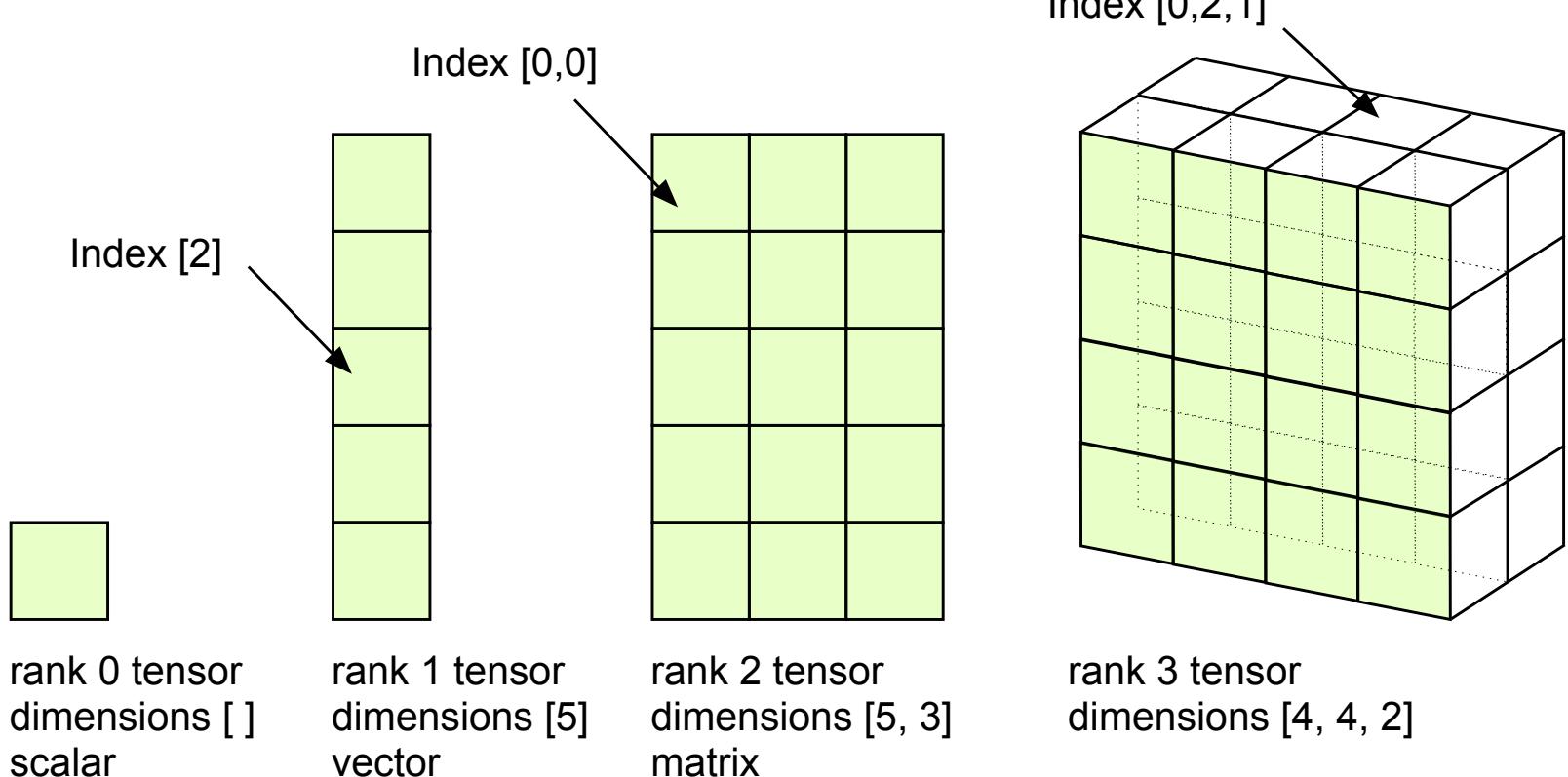
Scalar:  $\mathbb{R}$

Vector:  $\mathbb{R}^n$

Matrix:  $\mathbb{R}^n \times \mathbb{R}^m$

3-Tensor:  $\mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p$

...



[https://sebastianraschka.com/pdf/books/dlb/appendix\\_g\\_tensorflow.pdf](https://sebastianraschka.com/pdf/books/dlb/appendix_g_tensorflow.pdf)

# Installing TensorFlow

```
pip install tensorflow  
pip install tensorflow-gpu
```

<https://www.tensorflow.org/install/>

```
pip install tensorflow  
pip install tensorflow-gpu
```

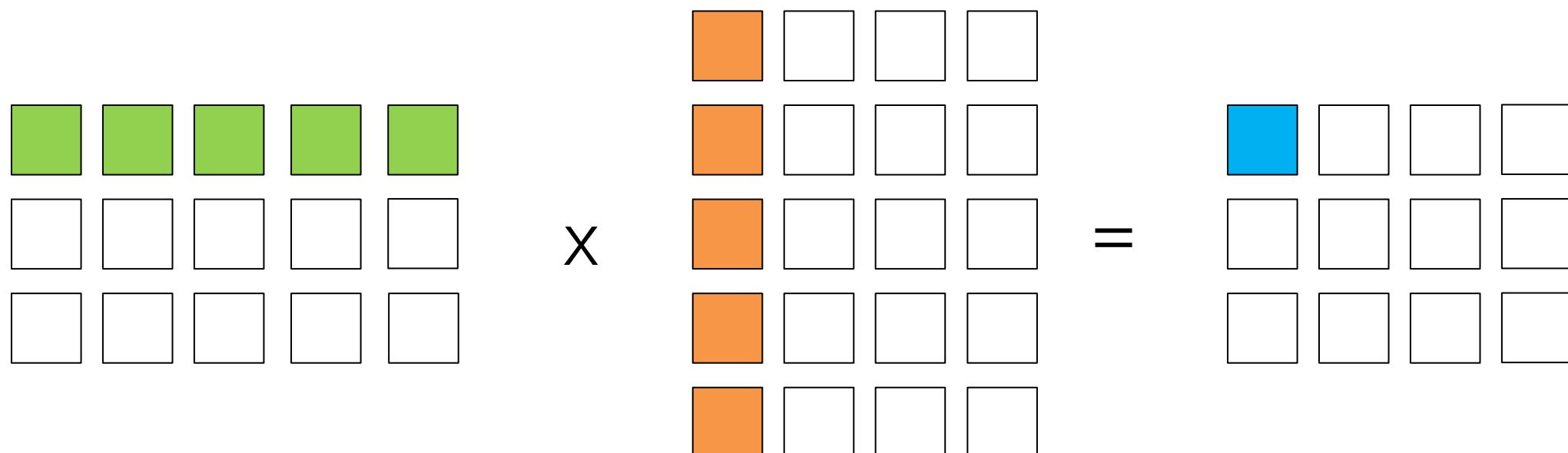
Specifications	Intel® Core™ i7-6900K Processor Extreme Ed.	NVIDIA GeForce® GTX™ 1080 Ti
Base Clock Frequency	3.2 GHz	< 1.5 GHz
Cores	8	3584
Memory Bandwidth	64 GB/s	484 GB/s
Floating-Point Calculations	409 GFLOPS	11300 GFLOPS
Cost	~ \$1000.00	~ \$700.00

Setup help:

- <https://www.tensorflow.org/install/>
- [https://sebastianraschka.com/pdf/books/dlb/appendix\\_h\\_cloud-computing.pdf](https://sebastianraschka.com/pdf/books/dlb/appendix_h_cloud-computing.pdf)

# Vectorization

```
X = np.random.random((num_train_examles, num_features))  
W = np.random.random((num_features, num_hidden))
```



```
logits = np.zeros([num_train_examples, num_hidden])

for i, row in enumerate(X): # row = training_example

    for j, col in enumerate(W.T): # col = features

        vector_dot_product = 0
        for a, b in zip(row, col):
            vector_dot_product += a*b

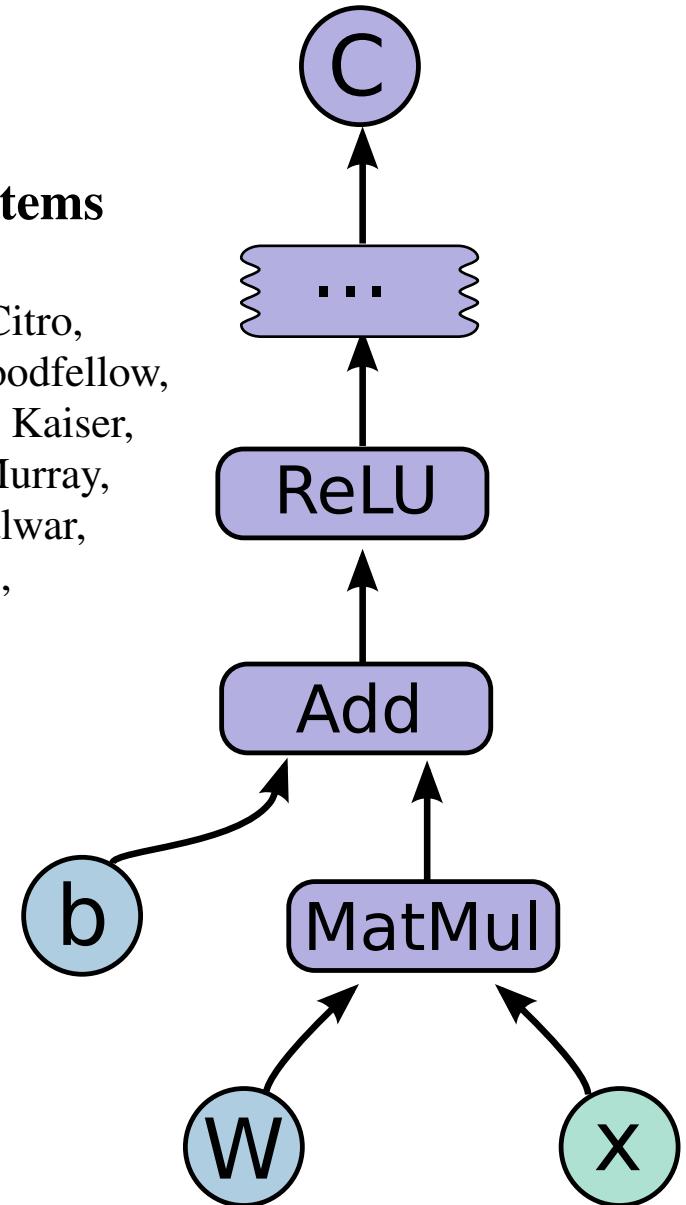
        logits[i, j] = vector_dot_product

np.allclose(logits, np.dot(X, W))
```

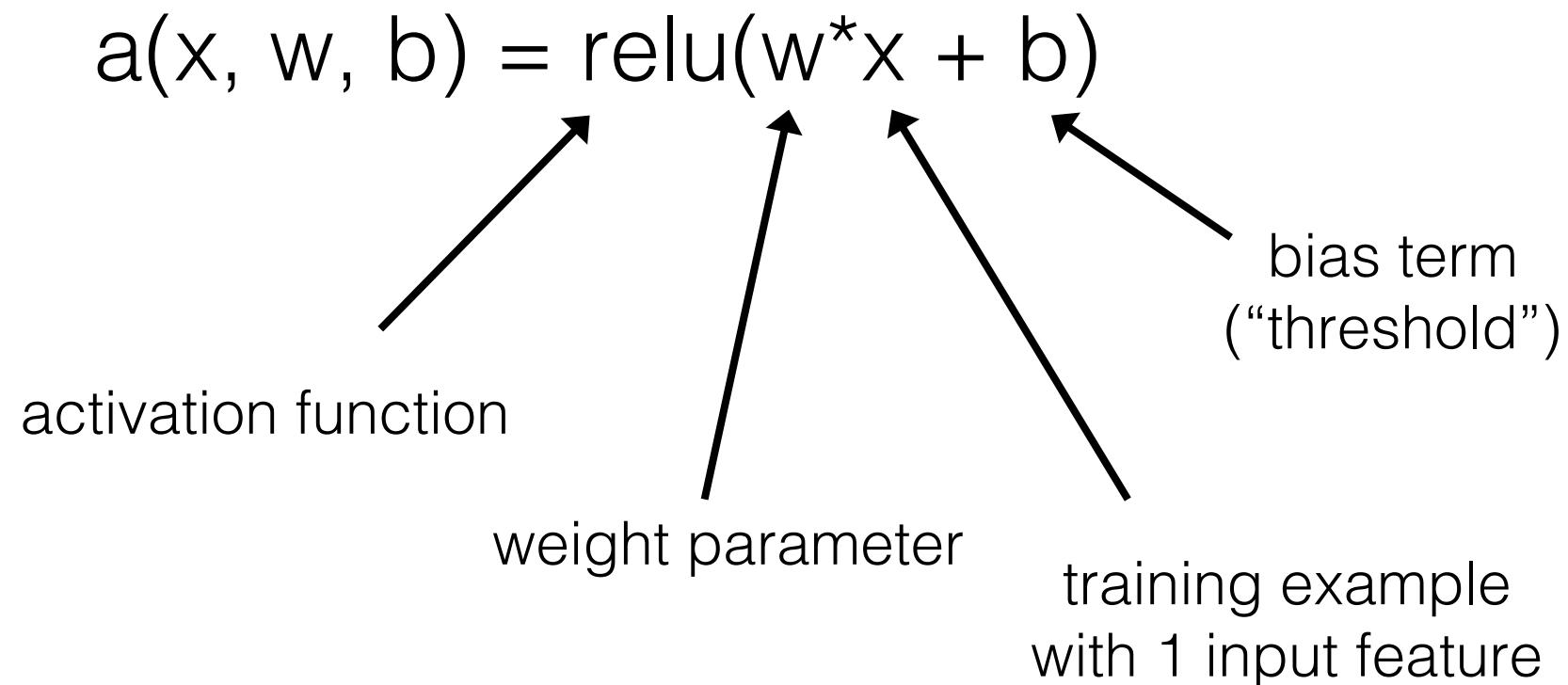
**TensorFlow:**  
**Large-Scale Machine Learning on Heterogeneous Distributed Systems**  
(Preliminary White Paper, November 9, 2015)

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng  
Google Research\*

<https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf>



# Computation Graphs

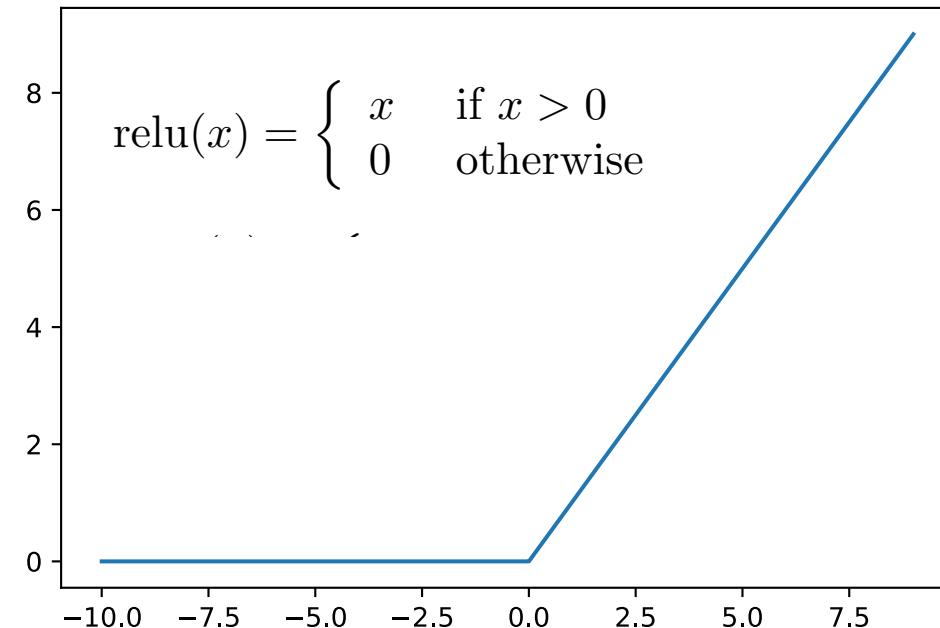


# REctified Linear Unit

```
import matplotlib.pyplot as plt
import numpy as np

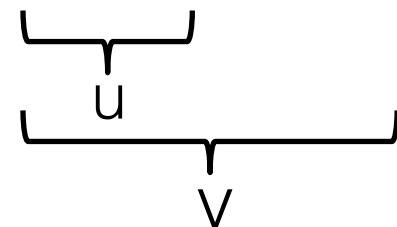
def relu(x):
    # max(0, x)
    return x * (x > 0)

x = np.arange(-10, 10)
plt.plot(x, relu(x))
```



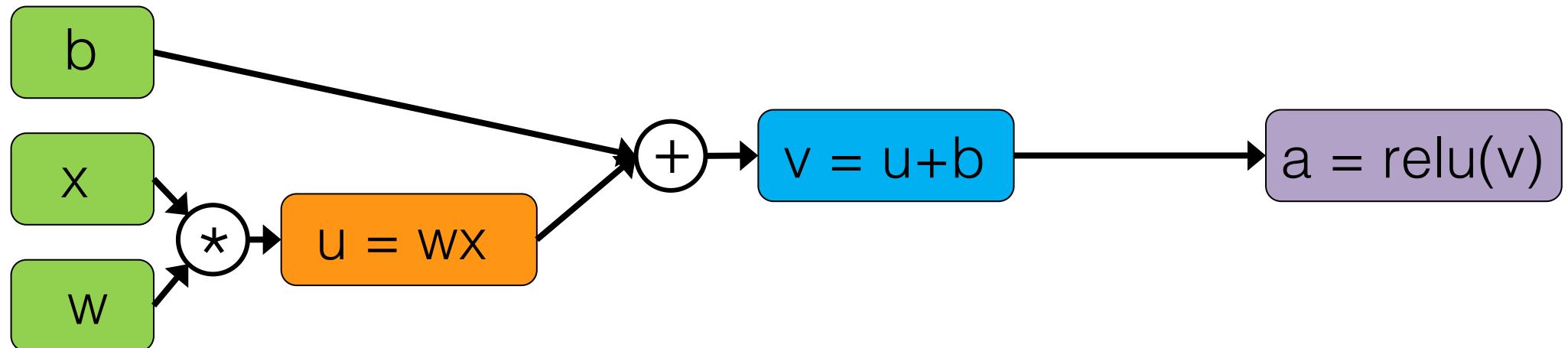
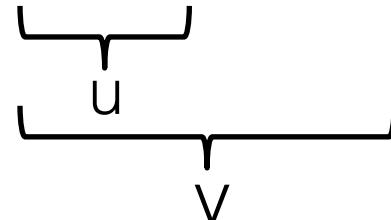
# Computation Graphs

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



# Computation Graphs

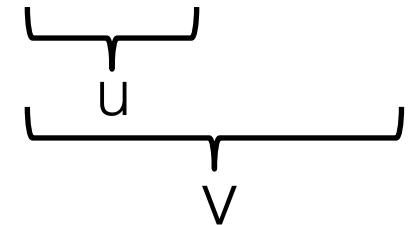
$$a(x, w, b) = \text{relu}(w^*x + b)$$



# Computation Graphs

```
import tensorflow as tf
g = tf.Graph()
with g.as_default() as g:
    x = tf.placeholder(dtype=tf.float32, shape=None, name='x')
    w = tf.Variable(initial_value=2, dtype=tf.float32, name='w')
    b = tf.Variable(initial_value=1, dtype=tf.float32, name='b')
    u = x * w
    v = u + b
    a = tf.nn.relu(v)
init_op = tf.global_variables_initializer()
print(x, w, b, u, v, a)
```

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



# Computation Graphs

```
import tensorflow as tf

g = tf.Graph()
with g.as_default() as g:

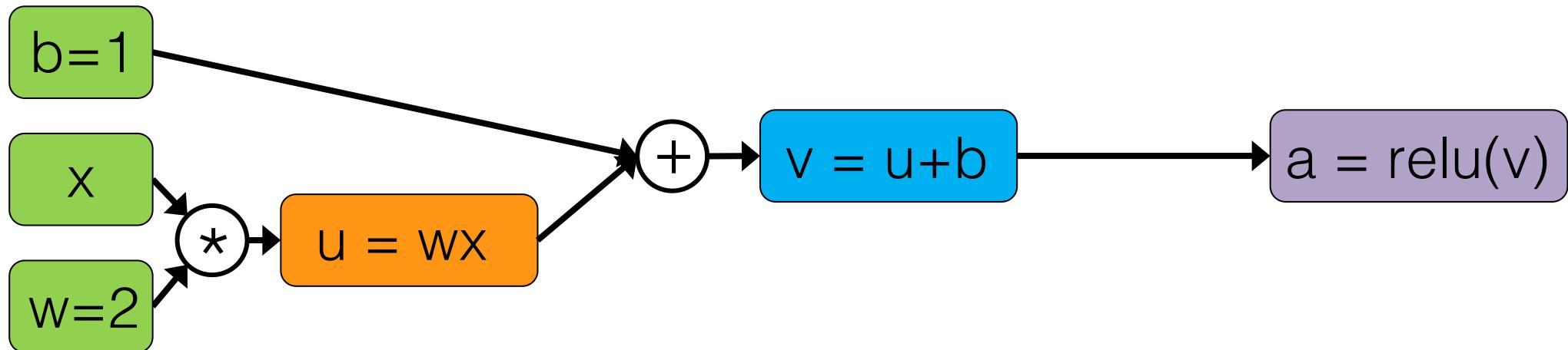
    x = tf.placeholder(dtype=tf.float32, shape=None, name='x')
    w = tf.Variable(initial_value=2, dtype=tf.float32, name='w')
    b = tf.Variable(initial_value=1, dtype=tf.float32, name='b')

    u = x * w
    v = u + b
    a = tf.nn.relu(v)

print(x, w, b, u, v, a)

Tensor("x:0", dtype=float32) <tf.Variable 'w:0' shape=() dtype=float32_ref> <tf.Variable 'b:0' shape=() dtype=float32_ref> Tensor("mul:0", dtype=float32) Tensor("add:0", dtype=float32) Tensor("Relu:0", dtype=float32)
```

# Computation Graphs



```
with tf.Session(graph=g) as sess:  
    sess.run(init_op)  
    b_res = sess.run('b:0')  
  
print(b_res)
```

1.0

# TensorBoard

```
with tf.Session(graph=g) as sess:  
    sess.run(init_op)  
    file_writer = tf.summary.FileWriter(logdir='logs/graph-1', graph=g)
```

In your terminal

```
$ pip install tensorboard  
$ tensorboard --logdir logs/graph-1
```

Fit to screen

Download PNG

Run

(1)

Session

runs (0)

Upload

Choose File

Trace inputs

Color Structure

Device

colors same substructure

unique substructure

Graph (\* = expandable)

Namespace\*

OpNode

Unconnected series\*

Connected series\*

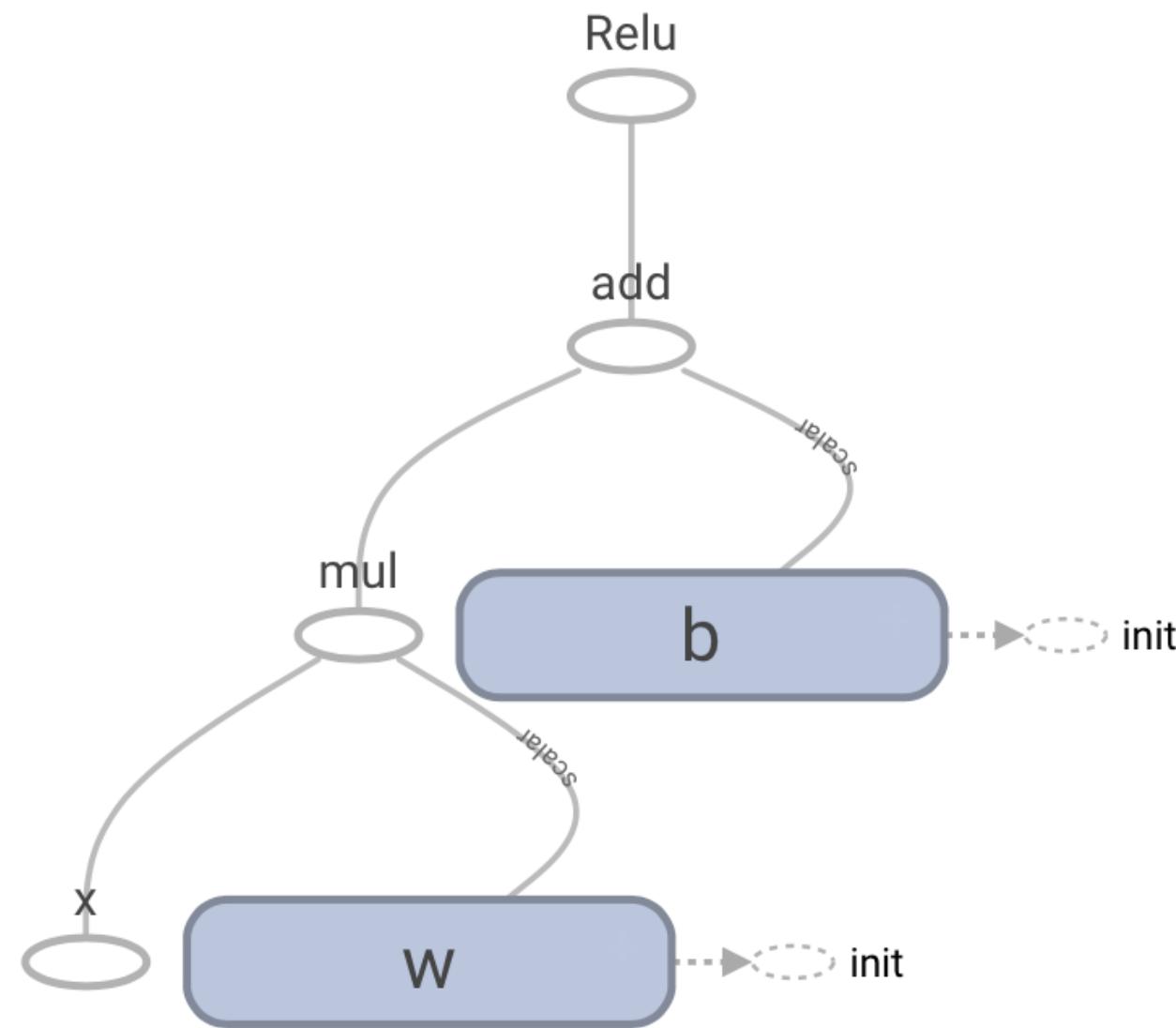
Constant

Summary

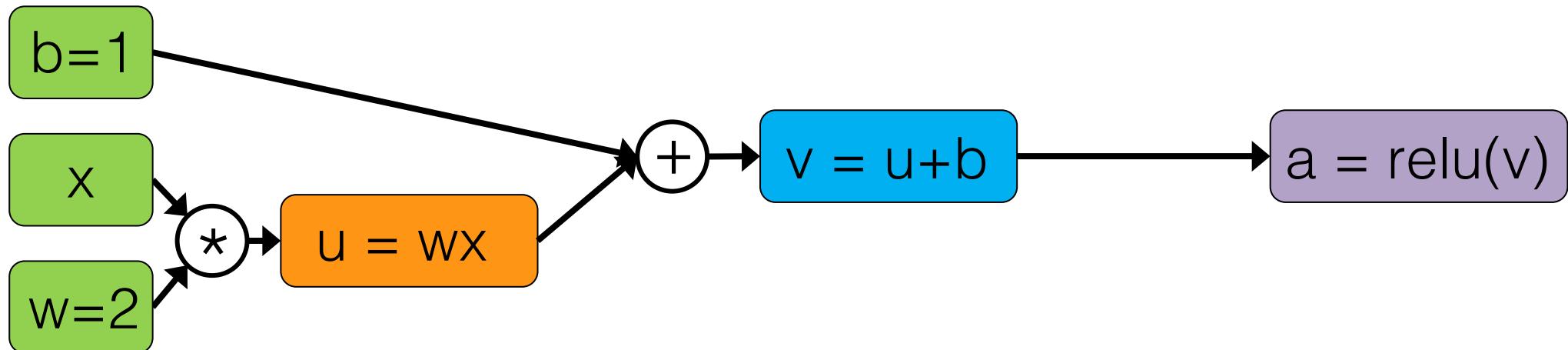
Dataflow edge

Control dependency edge

Reference edge



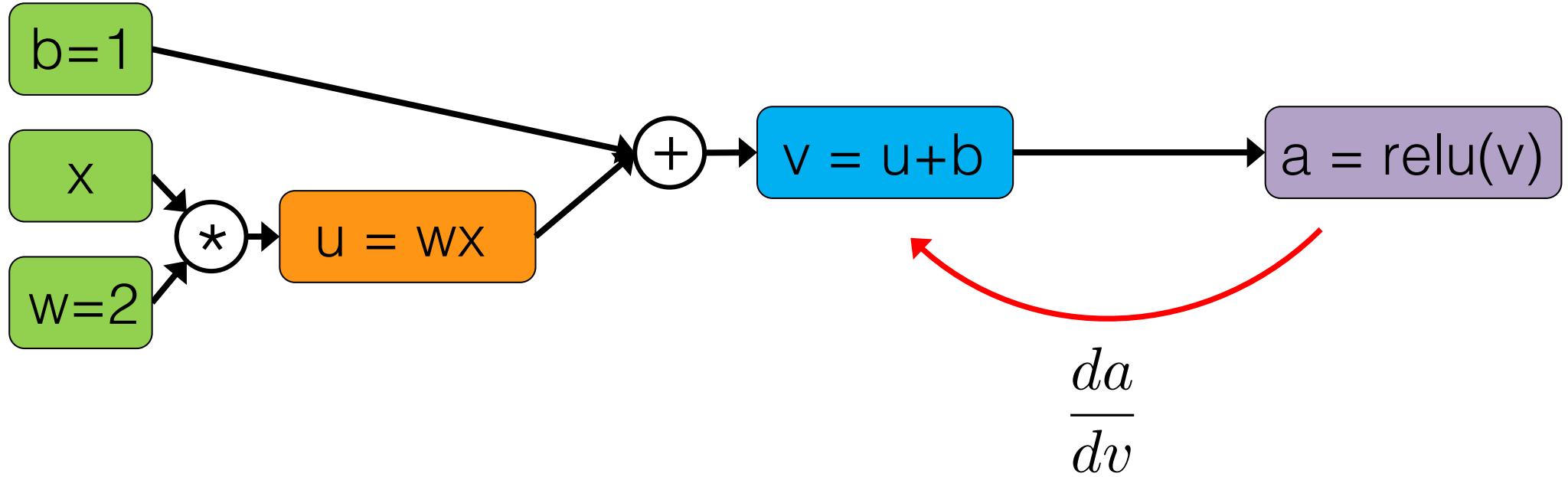
# Computation Graphs



```
with tf.Session(graph=g) as sess:  
    sess.run(tf.global_variables_initializer())  
    u_res, v_res, a_res = sess.run([u, v, a], feed_dict={'x:0': 3})  
  
print(u_res, v_res, a_res)
```

6.0, 6.0 7.0

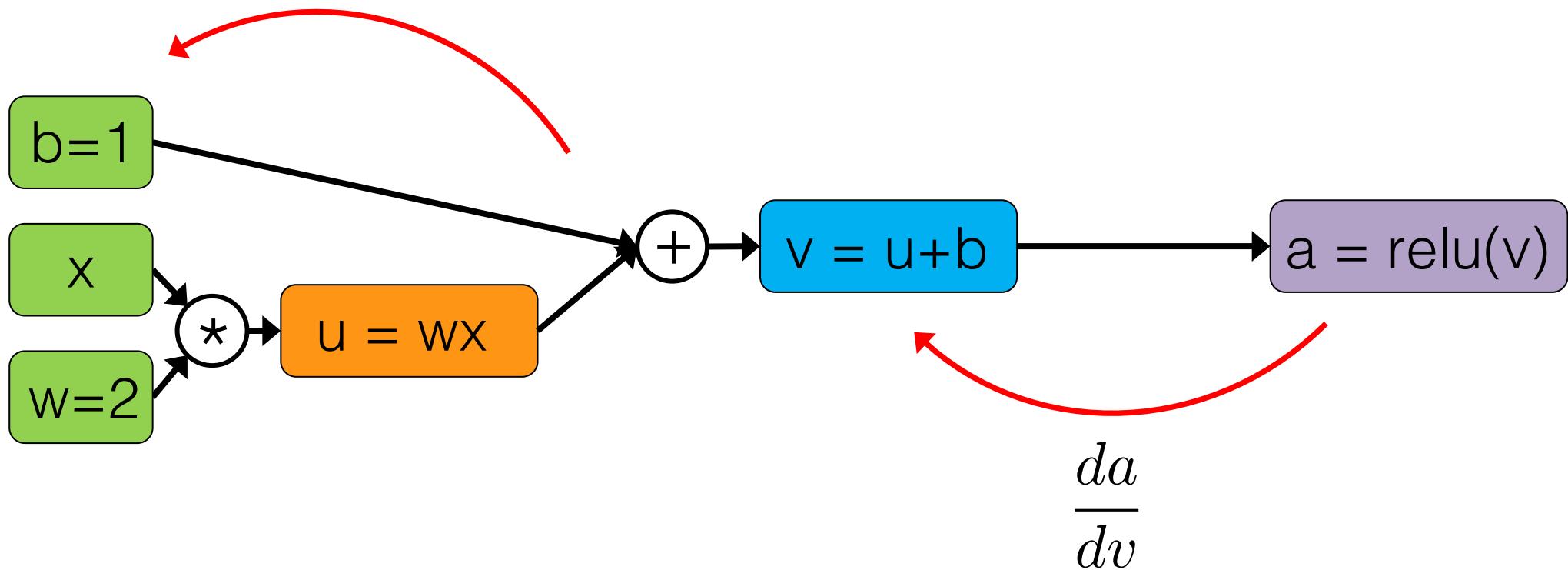
# Computation Graphs and Derivatives



Calculus refresher:

[https://sebastianraschka.com/pdf/books/dlb/appendix\\_d\\_calculus.pdf](https://sebastianraschka.com/pdf/books/dlb/appendix_d_calculus.pdf)

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

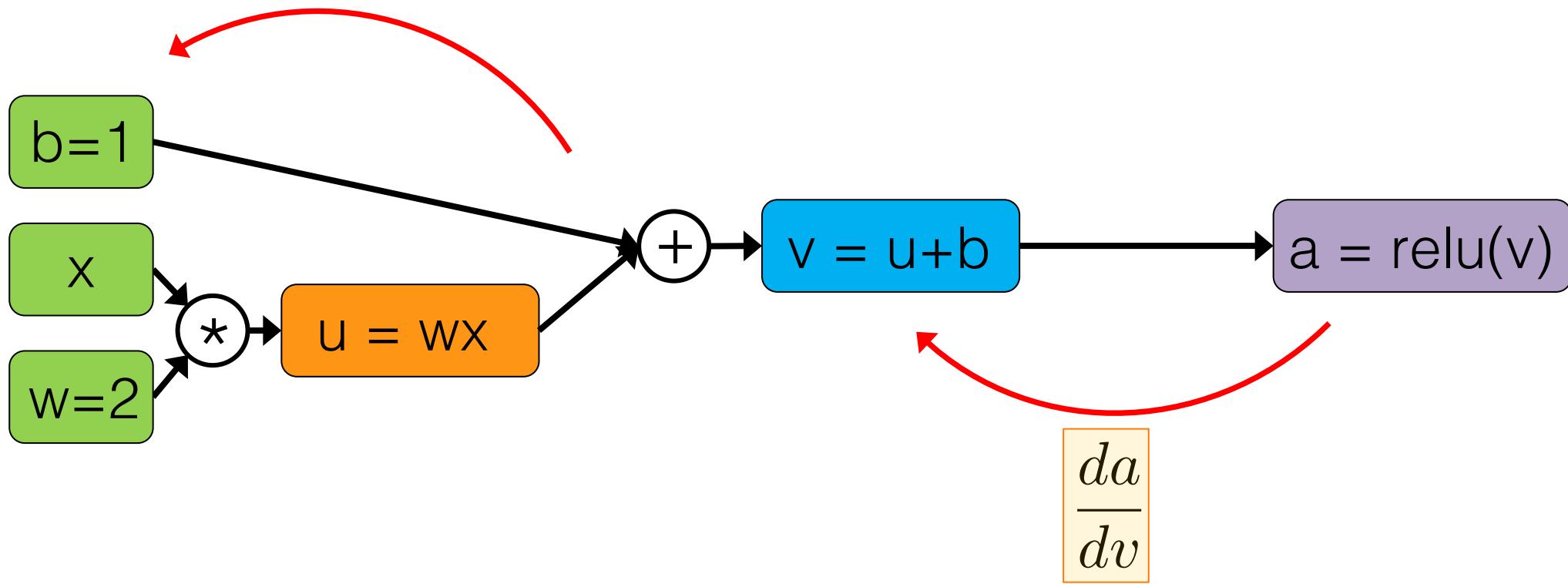


# Chain Rule

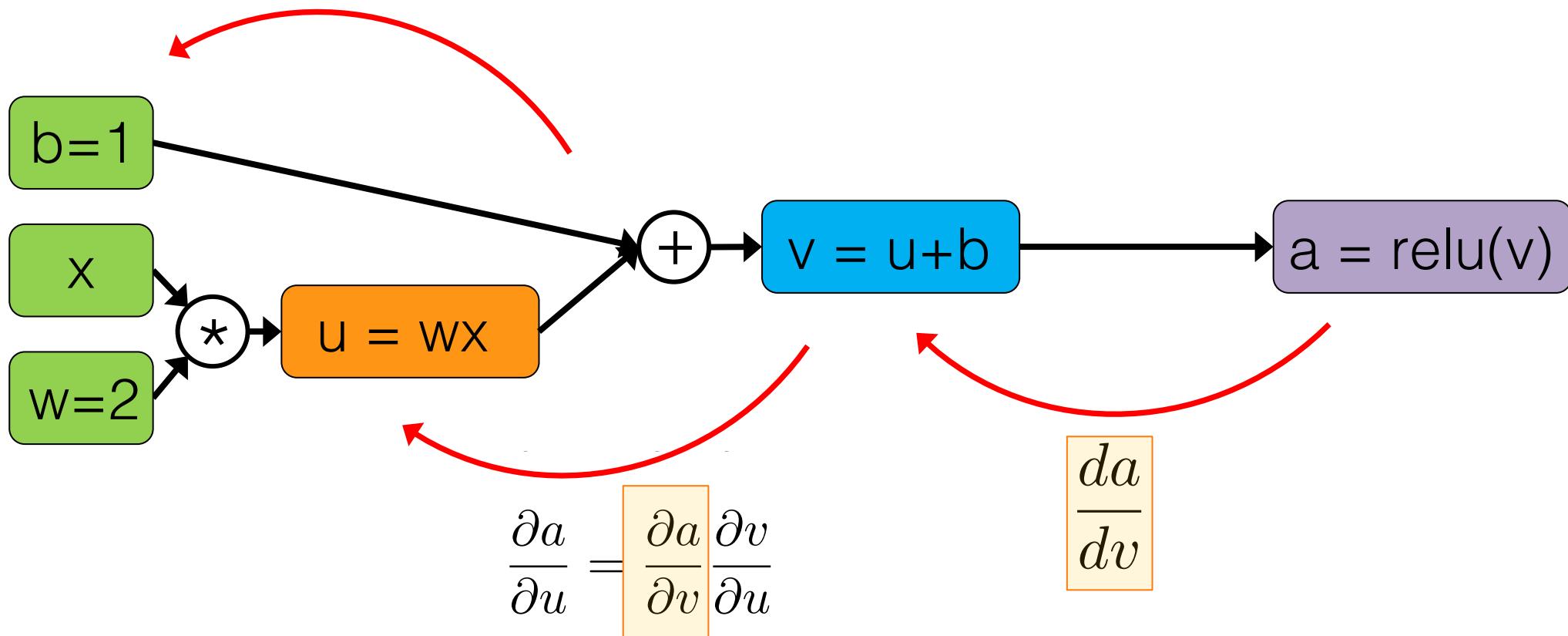
$$f(g(x))$$

$$\frac{d}{dx} [f(g(x))] = \frac{df}{dg} \cdot \frac{dg}{dx}$$

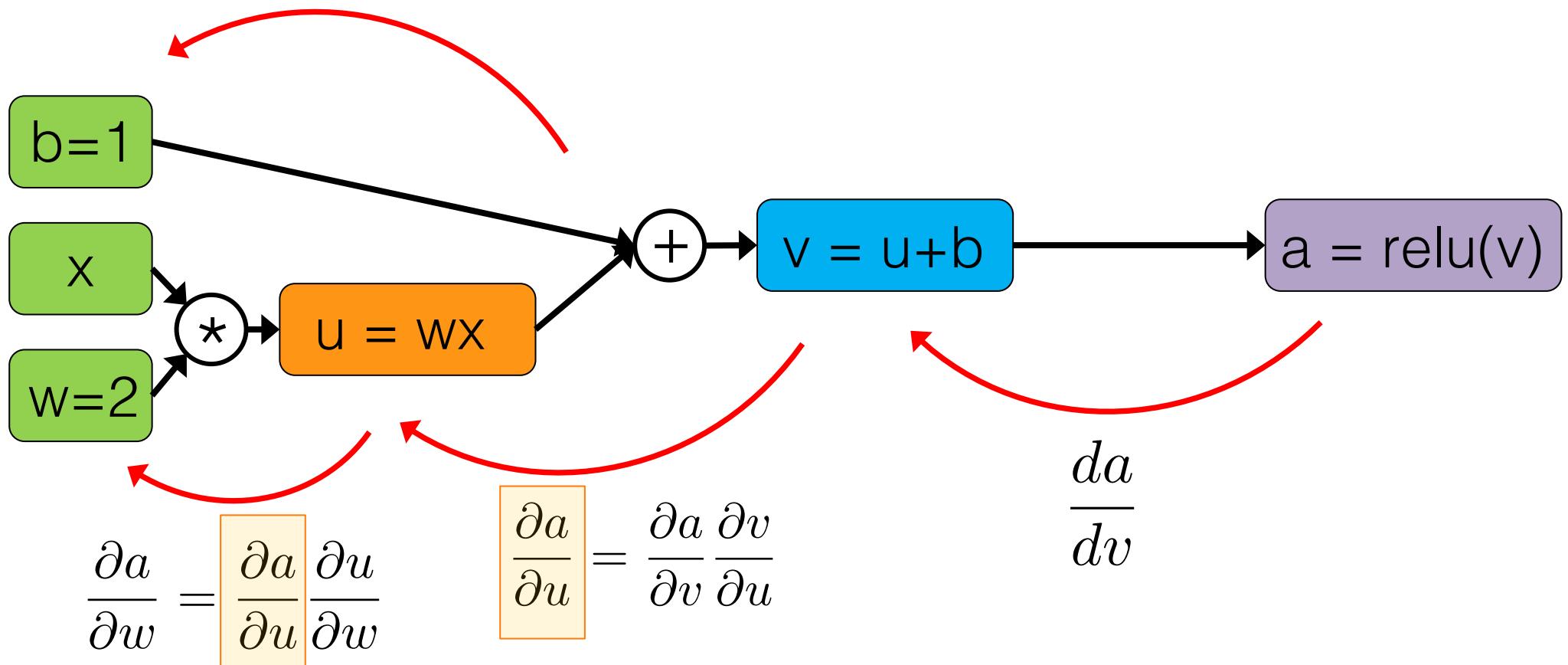
$$\frac{\partial a}{\partial b} = \boxed{\frac{\partial a}{\partial v}} \frac{\partial v}{\partial b}$$



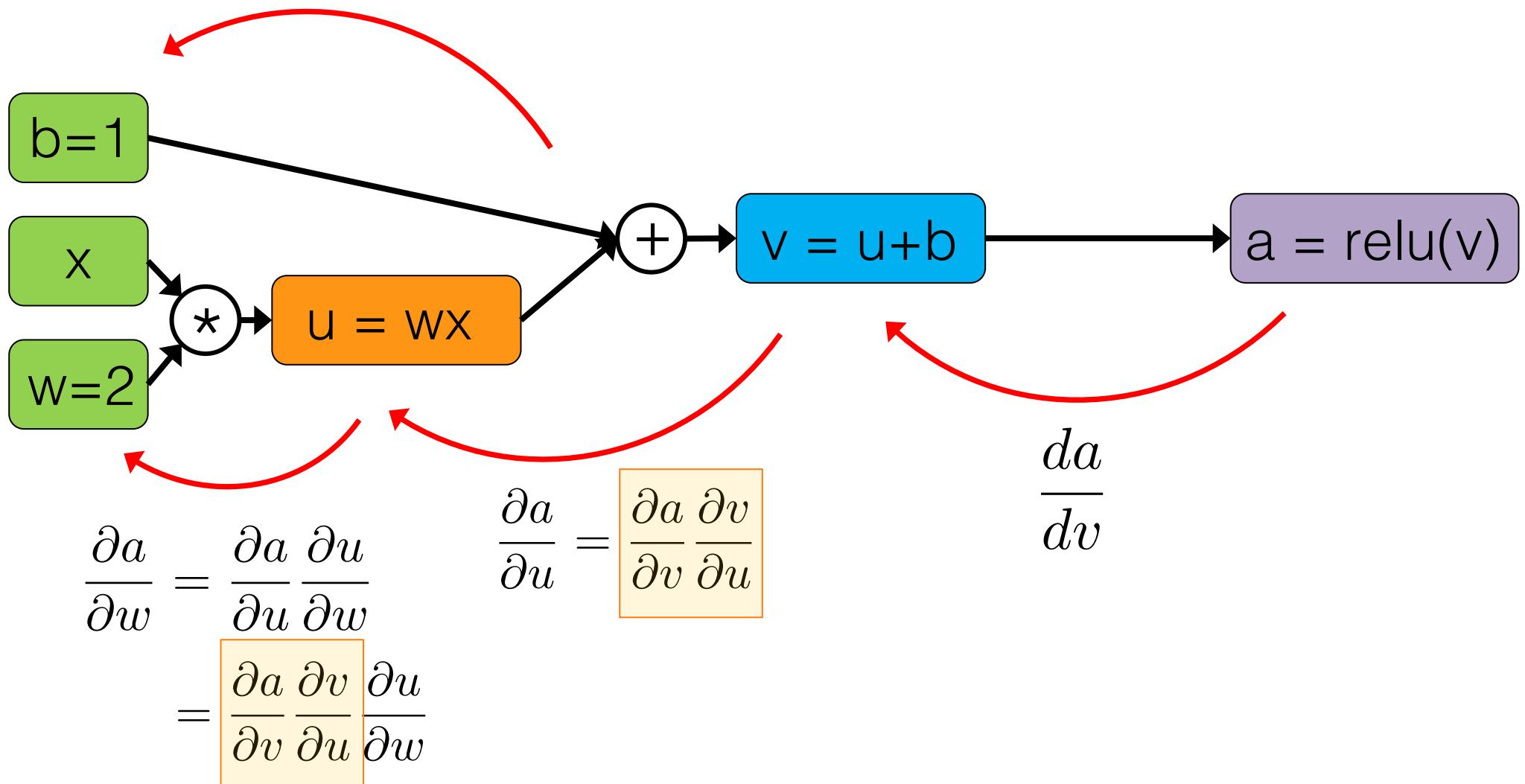
$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial b}$$



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

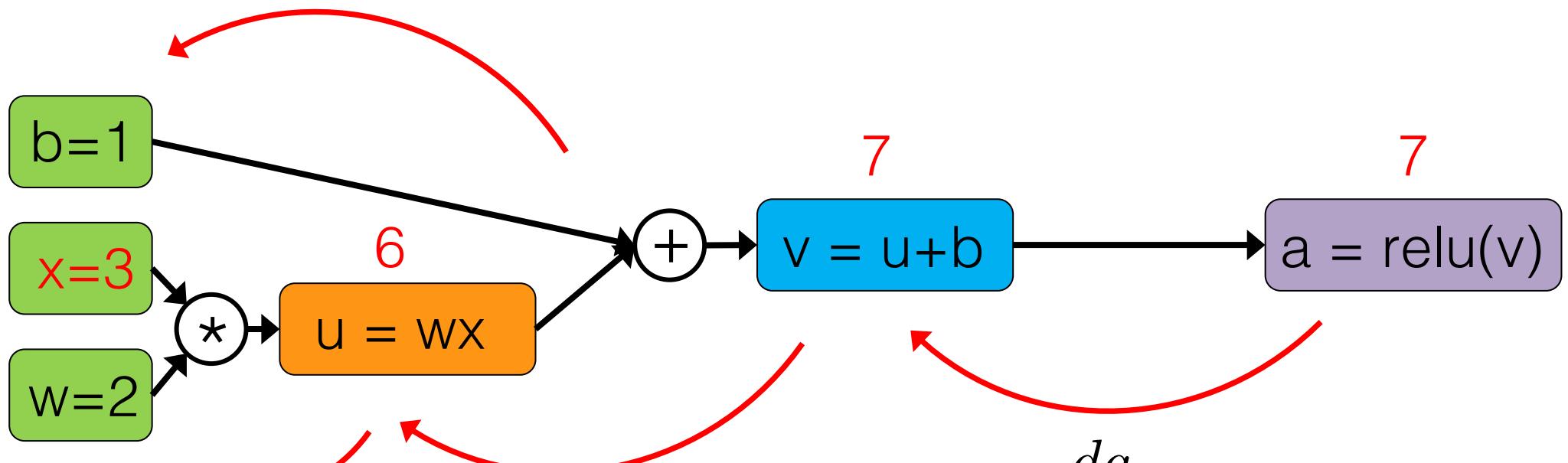


$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial b}$$



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

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

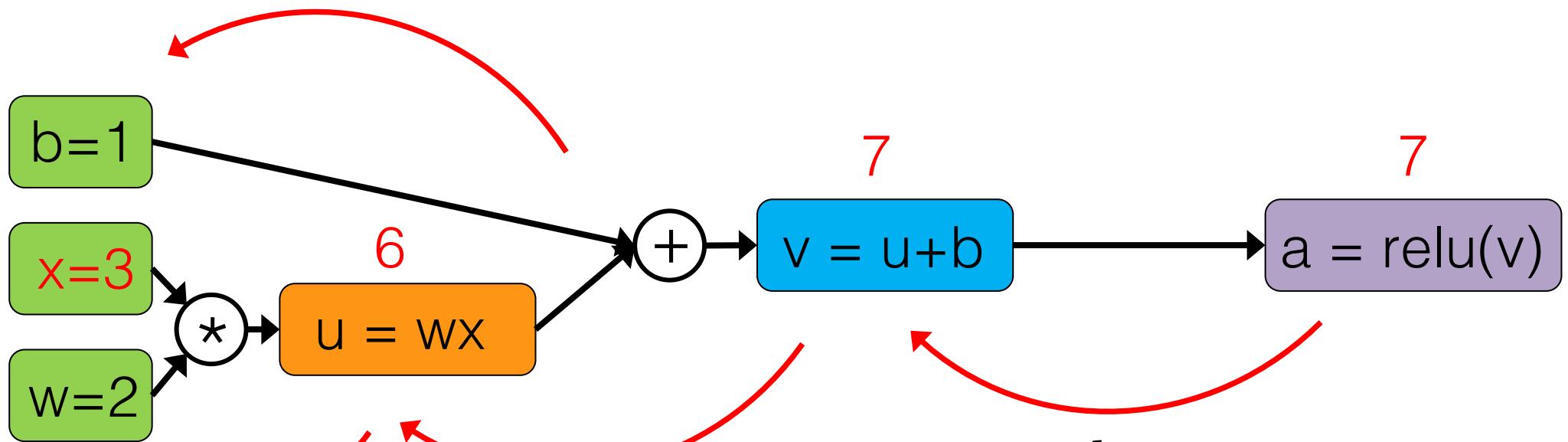


$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

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

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

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

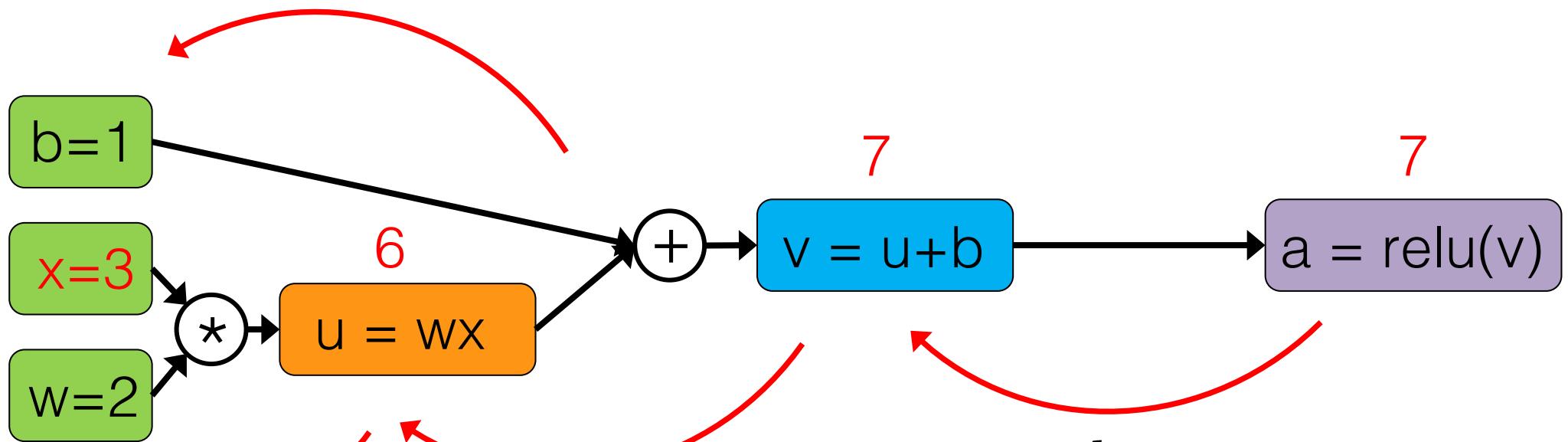


$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

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

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

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



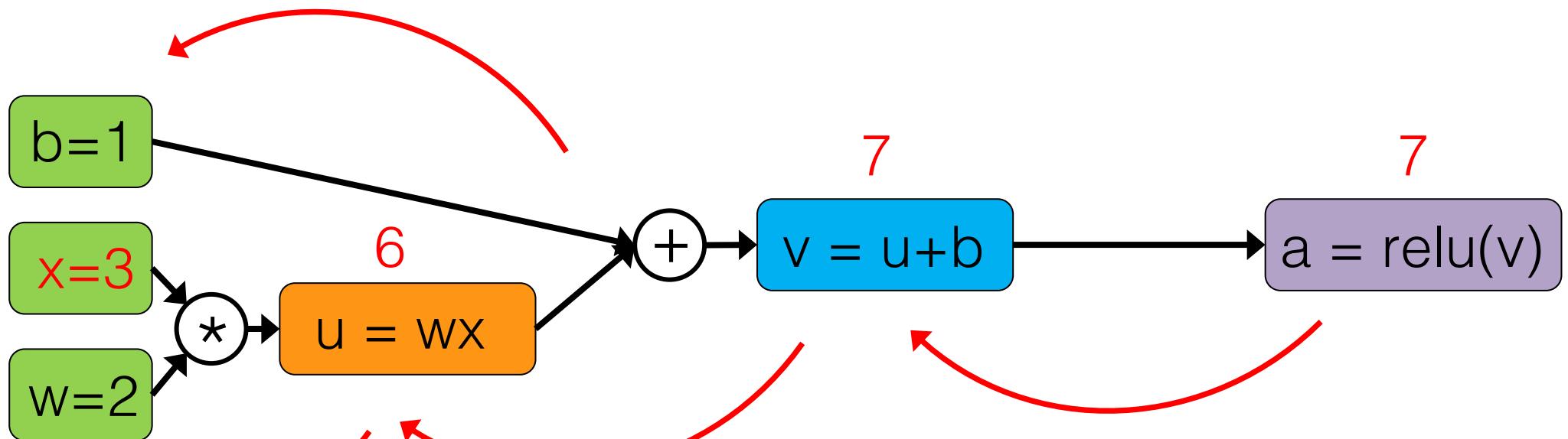
$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

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

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

$$\text{relu}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

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

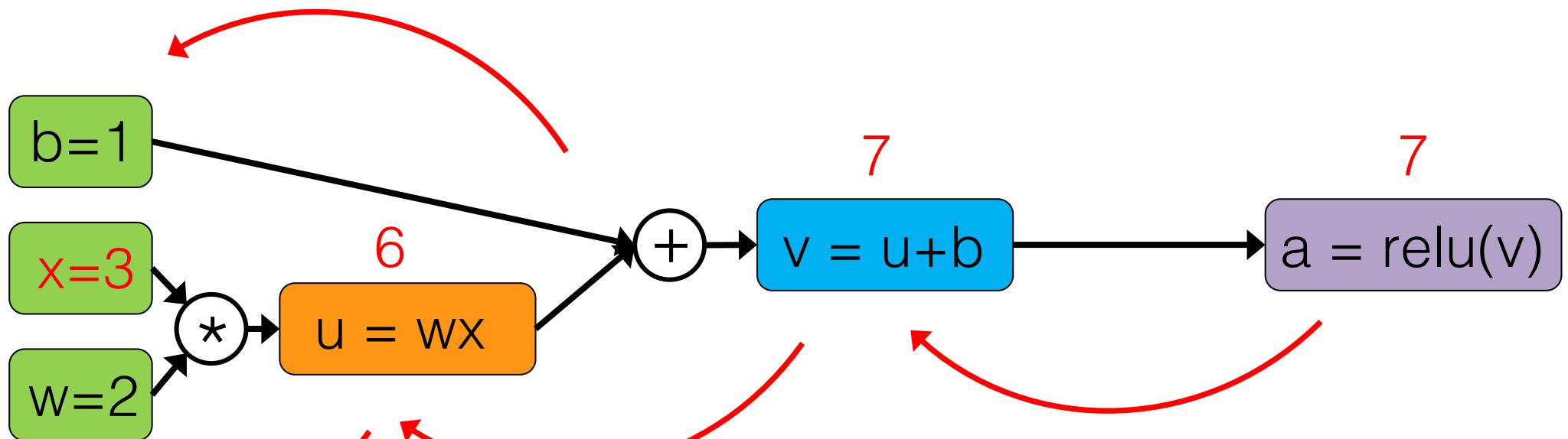


$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

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

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

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



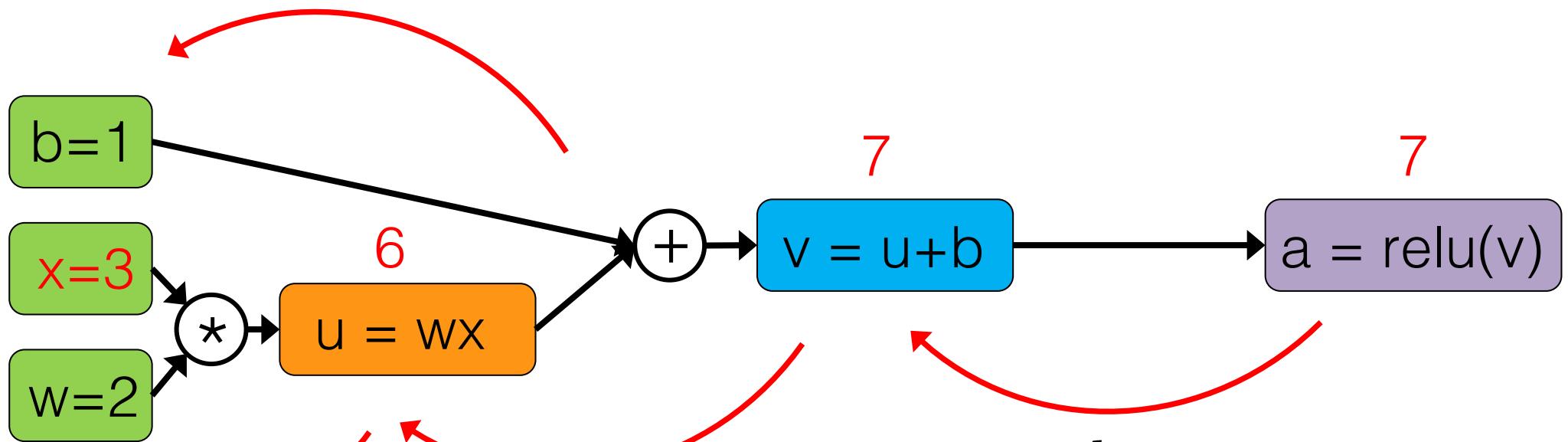
$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

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

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

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

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



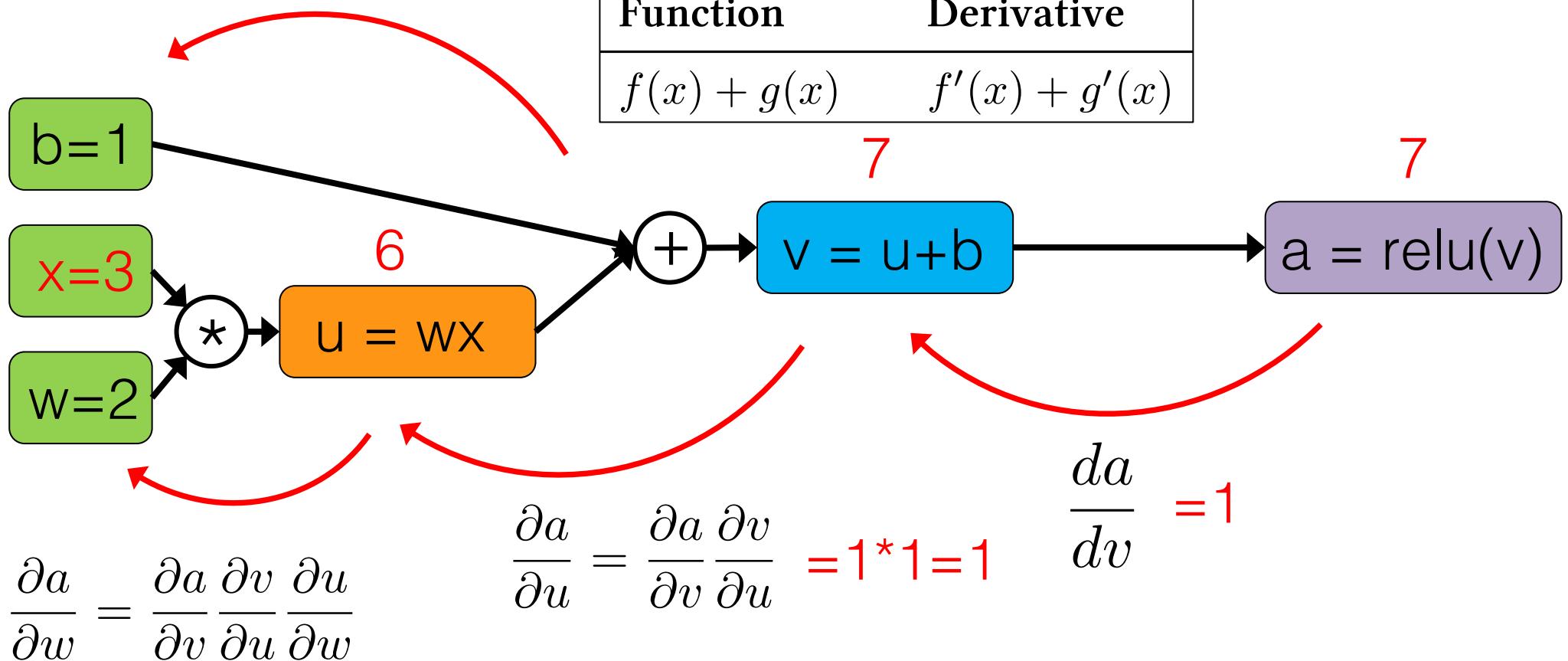
$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

$$\frac{\partial a}{\partial u} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} = 1 * 1 = 1$$

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

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

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

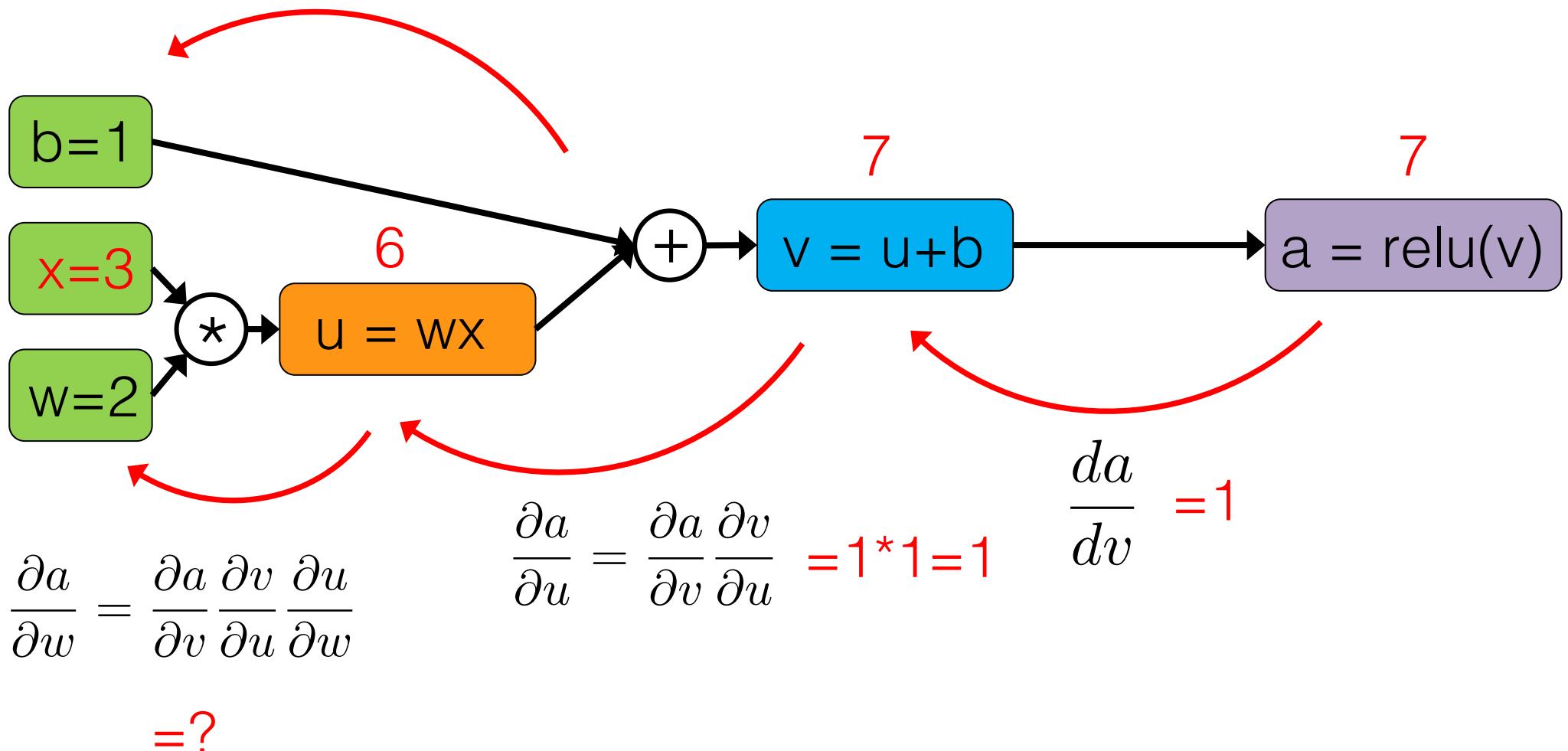


$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} \frac{\partial u}{\partial w}$$

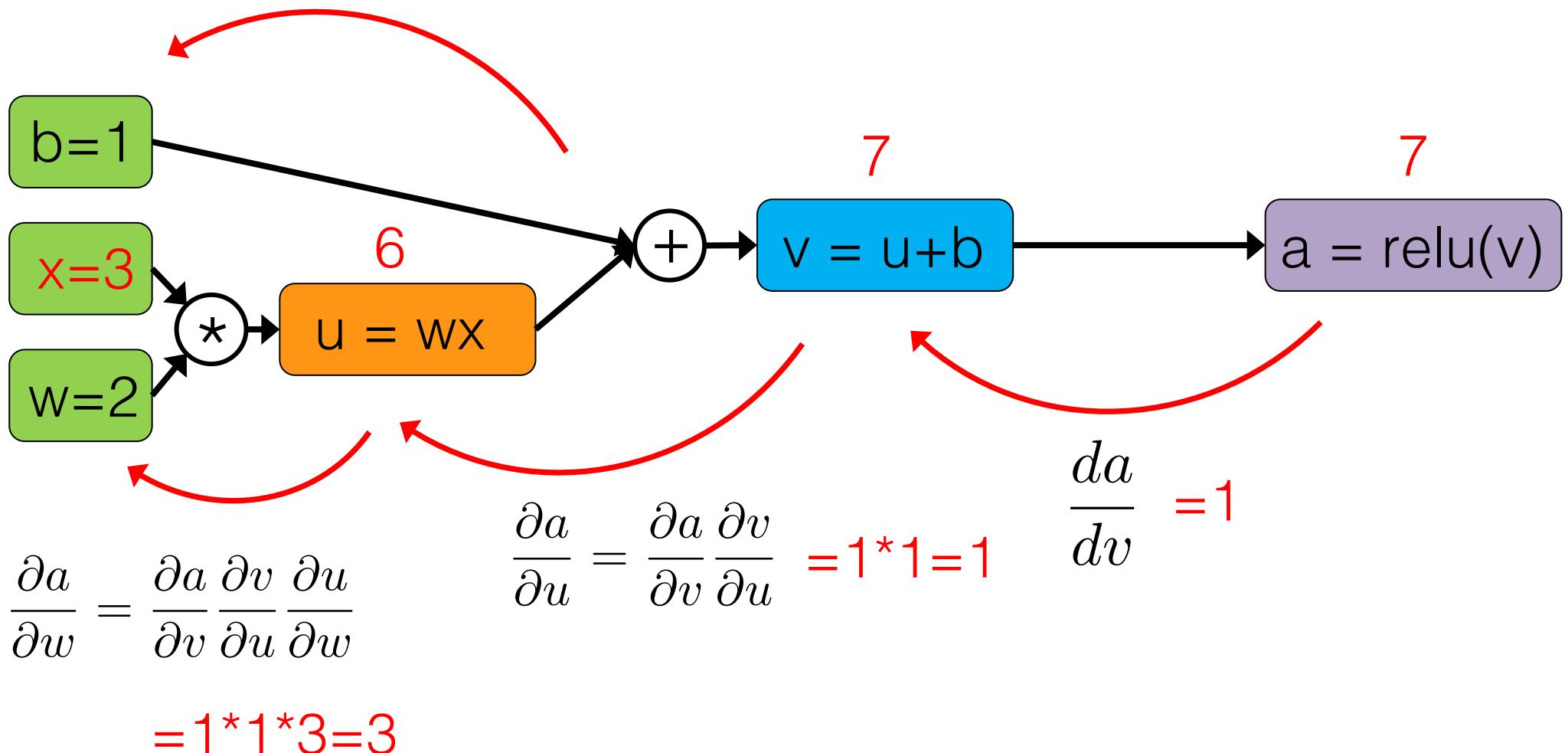
$$\frac{\partial a}{\partial u} = \frac{\partial a}{\partial v} \frac{\partial v}{\partial u} = 1 * 1 = 1$$

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

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



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



```
with g.as_default() as g:  
    d_a_w = tf.gradients(a, w)  
    d_b_w = tf.gradients(a, b)  
  
with tf.Session(graph=g) as sess:  
    sess.run(init_op)  
    dw, db = sess.run([d_a_w, d_b_w], feed_dict={'x:0': 3})  
  
print(dw, db)
```

[3.0] [1.0]

```
g = tf.Graph()
with g.as_default() as g:

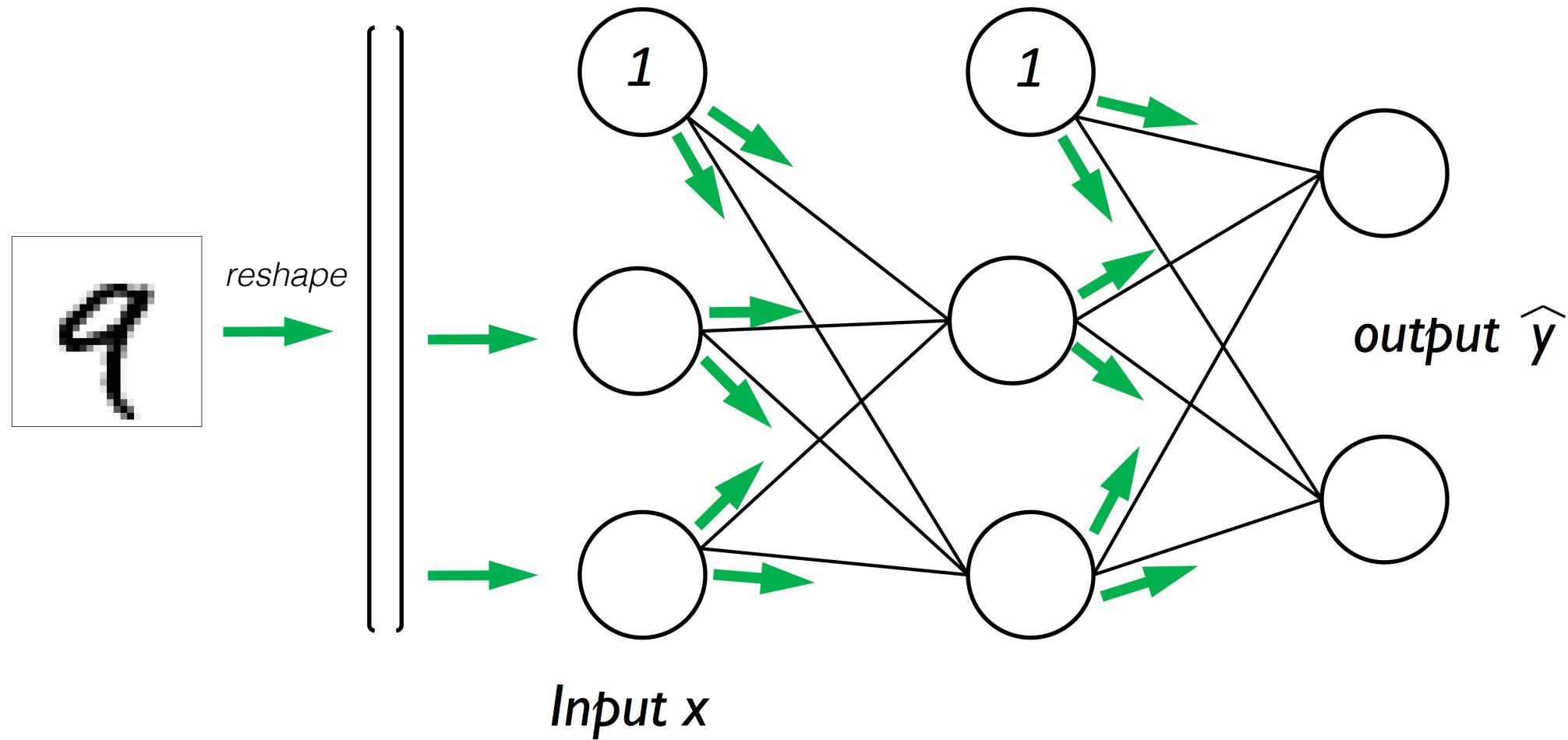
    x = tf.placeholder(dtype=tf.float32, shape=None, name='x')
    w = tf.Variable(initial_value=2, dtype=tf.float32, name='w')
    b = tf.Variable(initial_value=1, dtype=tf.float32, name='b')

    u = x * w
    v = u + b
    a = tf.nn.relu(v)

    d_a_w = tf.gradients(a, w)
    d_b_w = tf.gradients(a, b)

with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())
    res = sess.run([d_a_w, d_b_w], feed_dict={'x:0': 3})
```

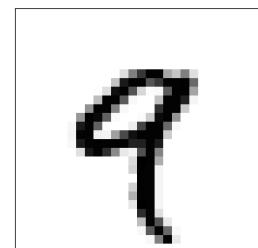
# Multilayer Perceptron – Forward Pass



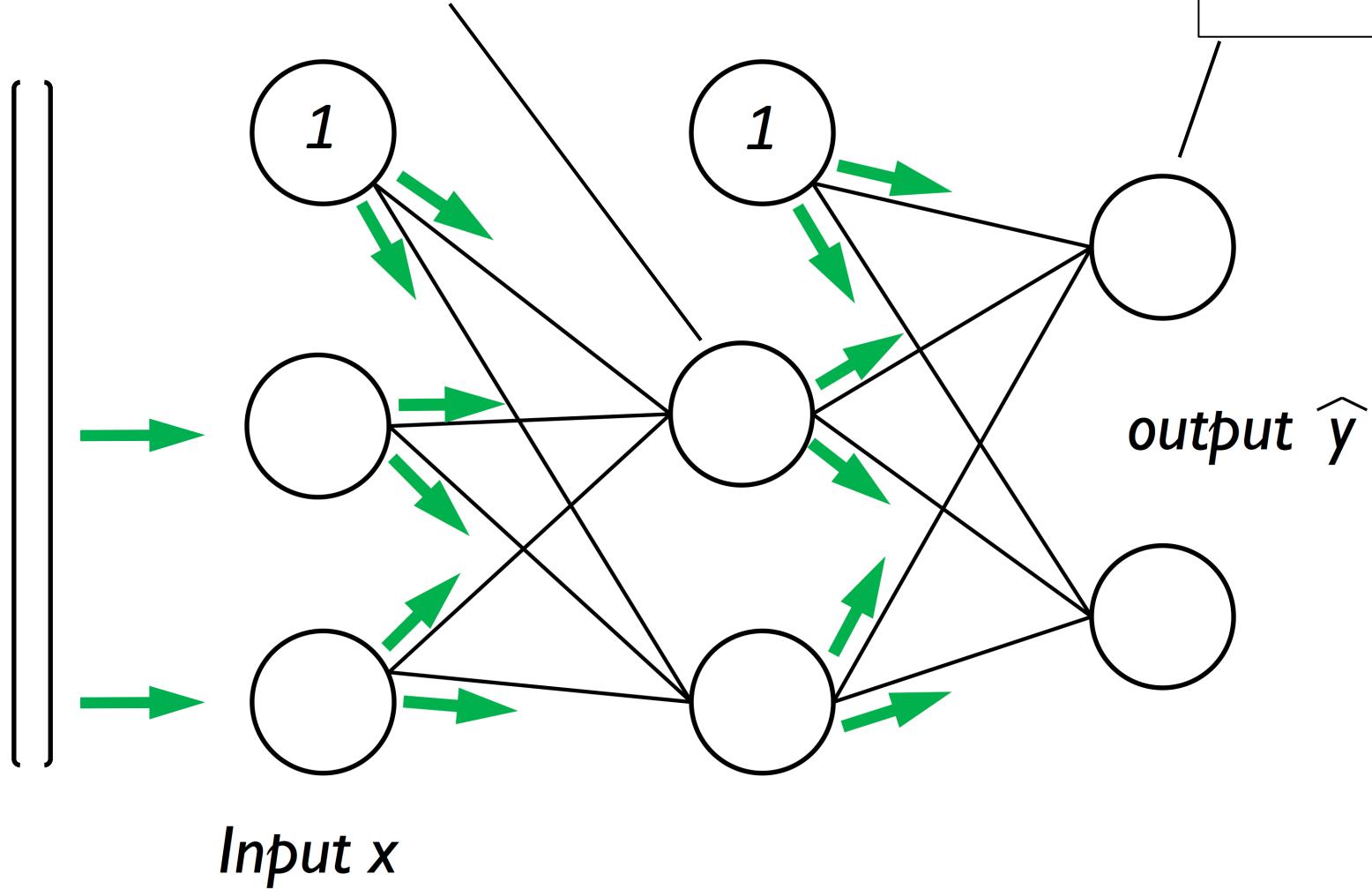
$$z_1^{(h)} = a_0^{(in)} w_{0,1}^{(h)} + a_1^{(in)} w_{1,1}^{(h)} + \dots + a_m^{(in)} w_{m,1}^{(h)}$$

$$a_1^{(h)} = \text{sigmoid}(z_1^{(h)})$$

$$a_1^{(o)} = \text{softmax}(z_1^{(o)})$$



reshape



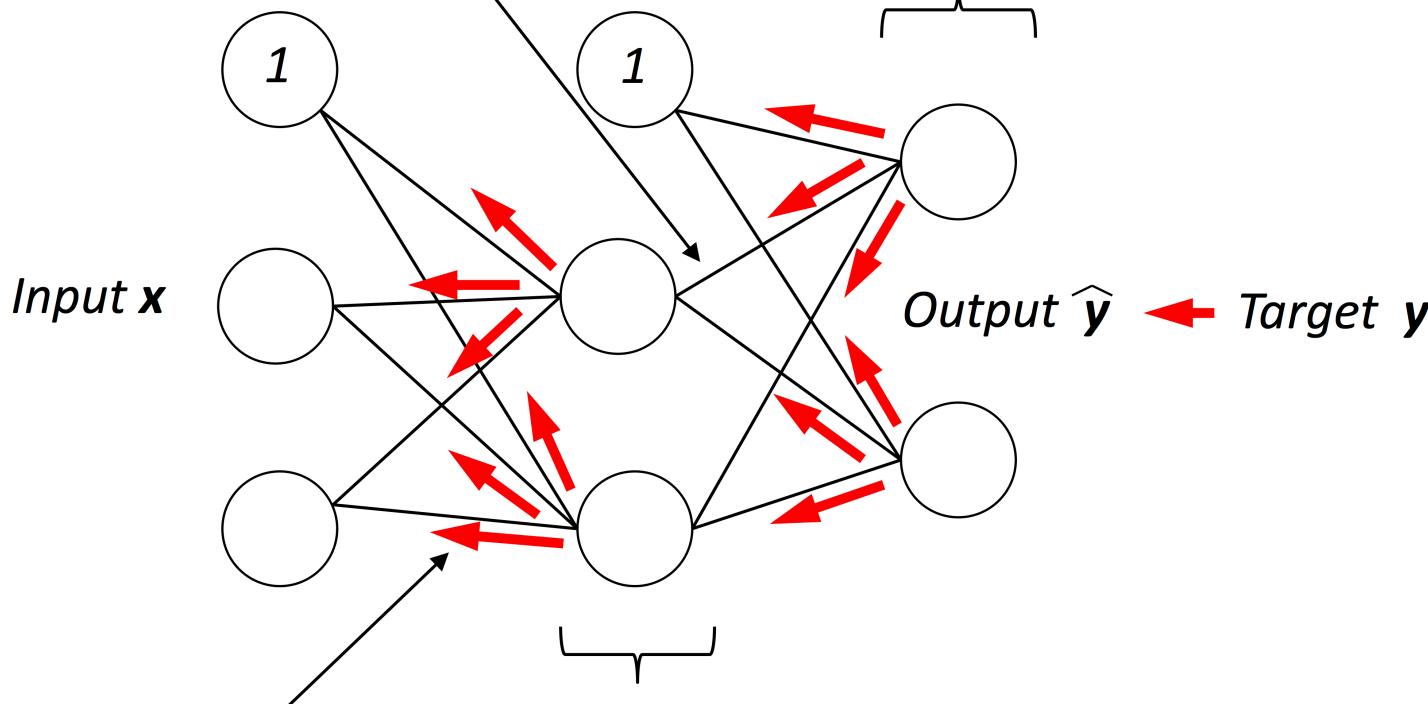
# Multilayer Perceptron – Backpropagation

Compute the gradient:

$$\frac{\partial}{\partial w_{i,j}^{(o)}} J(\mathbf{W}) = a_j^{(h)} \delta_i^{(o)}$$

Error term of the output layer:

$$\boldsymbol{\delta}^{(o)} = \mathbf{a}^{(o)} - \mathbf{y}$$



$$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(o)} (\mathbf{W}^{(o)})^T \odot \frac{\partial \phi(z^{(h)})}{\partial z^{(h)}}$$

Compute the gradient:

$$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\mathbf{W}) = a_j^{(in)} \delta_i^{(h)}$$

As implemented in  
<https://github.com/rasbt/pydata-ann Arbor2017-dl-tutorial/blob/master/code.ipynb>

TensorFlow makes implementing  
neural nets very convenient!

```

# Loss
loss = tf.nn.softmax_cross_entropy_with_logits(logits=out_z, labels=tf_y)
cost = tf.reduce_mean(loss, name='cost')

# input/output dim: [n_samples, n_classlabels]
sigma_out = (out_act - tf_y) / batch_size

# input/output dim: [n_samples, n_hidden_1]
softmax_derivative_h1 = h1_act * (1. - h1_act)

# input dim: [n_samples, n_classlabels] dot [n_classlabels, n_hidden]
# output dim: [n_samples, n_hidden]
sigma_h = (tf.matmul(sigma_out, tf.transpose(weights['out']))) *
softmax_derivative_h1

# input dim: [n_features, n_samples] dot [n_samples, n_hidden]
# output dim: [n_features, n_hidden]
grad_w_h1 = tf.matmul(tf.transpose(tf_x), sigma_h)
grad_b_h1 = tf.reduce_sum(sigma_h, axis=0)

# input dim: [n_hidden, n_samples] dot [n_samples, n_classlabels]
# output dim: [n_hidden, n_classlabels]
grad_w_out = tf.matmul(tf.transpose(h1_act), sigma_out)
grad_b_out = tf.reduce_sum(sigma_out, axis=0)

# Update weights
upd_w_1 = tf.assign(weights['h1'], weights['h1'] - learning_rate * grad_w_h1)
upd_b_1 = tf.assign(biases['b1'], biases['b1'] - learning_rate * grad_b_h1)
upd_w_out = tf.assign(weights['out'], weights['out'] - learning_rate * grad_w_out)
upd_b_out = tf.assign(biases['out'], biases['out'] - learning_rate * grad_b_out)

train = tf.group(upd_w_1, upd_b_1, upd_w_out, upd_b_out, name='train')

```

(very)  
low-level  
backprop

```
#####
### TRAINING & EVALUATION
#####

with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())

    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = mnist.train.num_examples // batch_size

        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            _, c = sess.run(['train', 'cost:0'], feed_dict={'features:0': batch_x,
                                                'targets:0': batch_y})
```

# low-level backprop

```
# Loss
loss = tf.nn.softmax_cross_entropy_with_logits(logits=out_z, labels=tf_y)
cost = tf.reduce_mean(loss, name='cost')

#####
# Backpropagation
#####

# Get Gradients
dc_dw_out, dc_db_out = tf.gradients(cost, [weights['out'], biases['out']])
dc_dw_1, dc_db_1 = tf.gradients(cost, [weights['h1'], biases['b1']])

# Update Weights
upd_w_1 = tf.assign(weights['h1'], weights['h1'] - learning_rate * dc_dw_1)
upd_b_1 = tf.assign(biases['b1'], biases['b1'] - learning_rate * dc_db_1)
upd_w_out = tf.assign(weights['out'], weights['out'] - learning_rate * dc_dw_out)
upd_b_out = tf.assign(biases['out'], biases['out'] - learning_rate * dc_db_out)

train = tf.group(upd_w_1, upd_b_1, upd_w_out, upd_b_out, name='train')
```

## “convenient” backprop

```
# Loss
loss = tf.nn.softmax_cross_entropy_with_logits(logits=out_z, labels=tf_y)
cost = tf.reduce_mean(loss, name='cost')

#####
# Backpropagation
#####

optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
train = optimizer.minimize(cost, name='train')
```

```
#####
### TRAINING & EVALUATION
#####

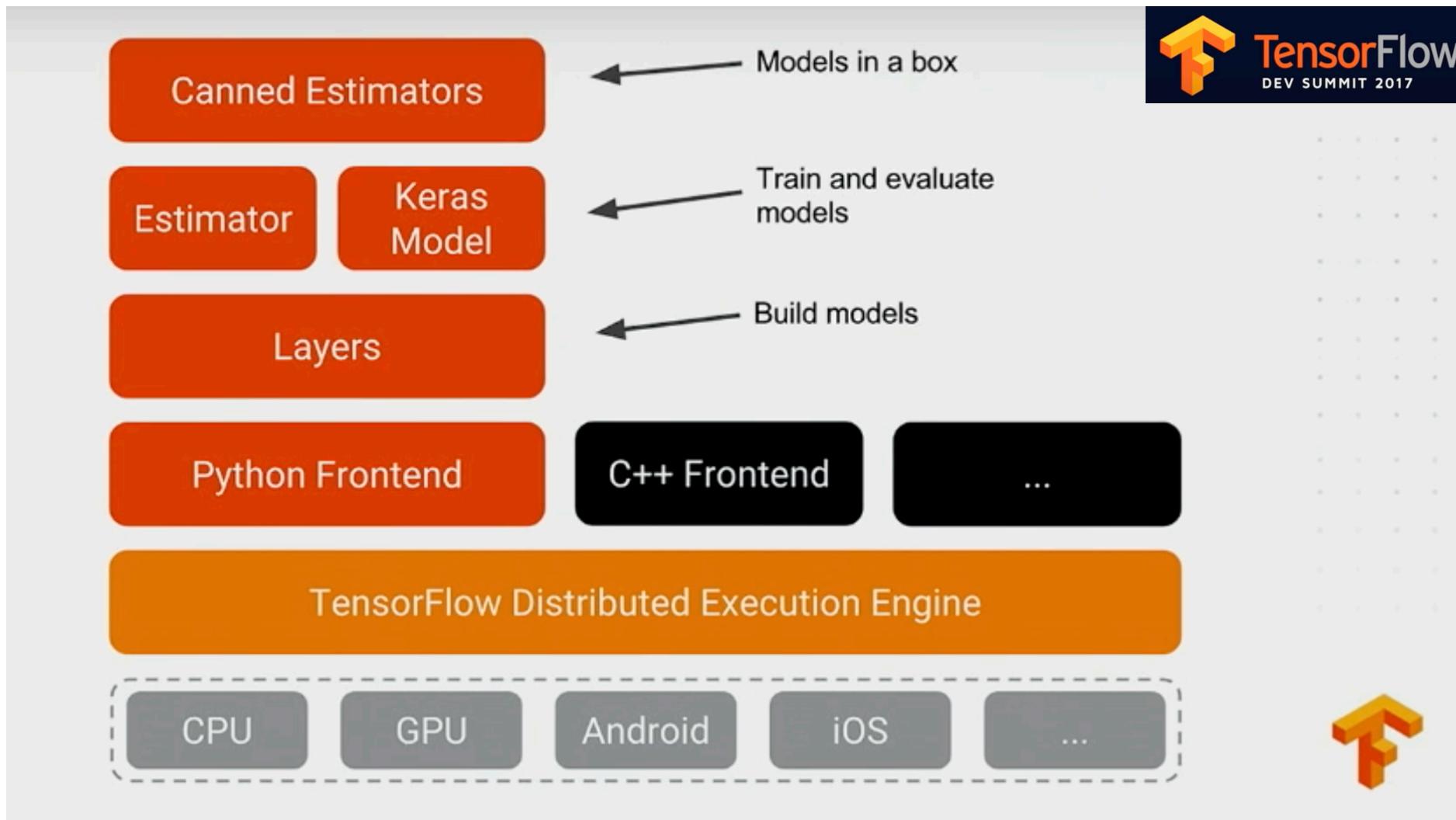
with tf.Session(graph=g) as sess:
    sess.run(tf.global_variables_initializer())

    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = mnist.train.num_examples // batch_size

        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size)
            _, c = sess.run(['train', 'cost:0'], feed_dict={'features:0': batch_x,
                                                'targets:0': batch_y})
```

# TensorFlow Layers





Link to the talk: <https://www.youtube.com/watch?v=t64ortpgS-E>

Estimator Documentation: <https://www.tensorflow.org/extend/estimators>

# Defining your wrapper functions manually

```
def fully_connected(input_tensor, output_nodes,
                    activation=None, seed=None,
                    name='fully_connected'):

    with tf.variable_scope(name):
        input_nodes = input_tensor.get_shape().as_list()[1]
        weights = tf.Variable(tf.truncated_normal(shape=(input_nodes,
                                                       output_nodes),
                                                   mean=0.0,
                                                   stddev=0.01,
                                                   dtype=tf.float32,
                                                   seed=seed),
                              name='weights')
        biases = tf.Variable(tf.zeros(shape=[output_nodes]), name='biases')

        act = tf.matmul(input_tensor, weights) + biases
        if activation is not None:
            act = activation(act)

    return act
```

# Using tensorflow.layers

```
g = tf.Graph()
with g.as_default():

    # Input data
    tf_x = tf.placeholder(tf.float32, [None, n_input], name='features')
    tf_y = tf.placeholder(tf.float32, [None, n_classes], name='targets')

    # Multilayer perceptron
    layer_1 = tf.layers.dense(tf_x, n_hidden_1,
                            activation=tf.nn.relu,
                            kernel_initializer=tf.truncated_normal_initializer(stddev=0.1))
    layer_2 = tf.layers.dense(layer_1, n_hidden_2,
                            activation=tf.nn.relu,
                            kernel_initializer=tf.truncated_normal_initializer(stddev=0.1))
    out_layer = tf.layers.dense(layer_2, n_classes, activation=None)
```

# Feeding Data into the Graph

## From Python via placeholders

```
sess.run(..., feed_dict={'x:0': ..., 'y:0': ..., ...})
```

- Python pickle
- NumPy .npz archives ([https://github.com/rasbt/deep-learning-book/blob/master/code/model\\_zoo/image-data-chunking-npz.ipynb](https://github.com/rasbt/deep-learning-book/blob/master/code/model_zoo/image-data-chunking-npz.ipynb))
- HDF5 ([https://github.com/rasbt/deep-learning-book/blob/master/code/model\\_zoo/image-data-chunking-hdf5.ipynb](https://github.com/rasbt/deep-learning-book/blob/master/code/model_zoo/image-data-chunking-hdf5.ipynb))
- CSV
- ...

## Using input pipelines and queues

- Reading data from TFRecords files ([https://github.com/rasbt/deep-learning-book/blob/master/code/model\\_zoo/tfrecords.ipynb](https://github.com/rasbt/deep-learning-book/blob/master/code/model_zoo/tfrecords.ipynb))
- Queues for loading raw images ([https://github.com/rasbt/deep-learning-book/blob/master/code/model\\_zoo/file-queues.ipynb](https://github.com/rasbt/deep-learning-book/blob/master/code/model_zoo/file-queues.ipynb))

More info: [https://www.tensorflow.org/programmers\\_guide/reading\\_data](https://www.tensorflow.org/programmers_guide/reading_data)



Name: Sebastian Raschka

Title: An Introduction to Deep Learning with TensorFlow

### Abstract

=====

In this tutorial, you will learn how to use the open-source TensorFlow library for deep learning. What's so great about TensorFlow is that it allows us to work with multi-dimensional arrays and train deep neural network very efficiently by utilizing GPU resources.

In this introduction to TensorFlow, you will learn how to define computational graphs and how to execute them in a Python runtime environment. After implementing backpropagation for a simple multi-layer perceptron, we will talk about TensorFlow's convenience features for optimization and the new layers API to construct more complex deep learning architectures more compactly. Finally, we will implement a General Adversarial Networks architecture to see how we can access and update variables from different network graphs and scopes -- the entry point for inventing and experimenting with novel architectures in our real-world applications and research.



PyData Aug 2017

Name: Sebastian Raschka

Title: An Introduction to Deep Learning with TensorFlow

## Abstract

=====

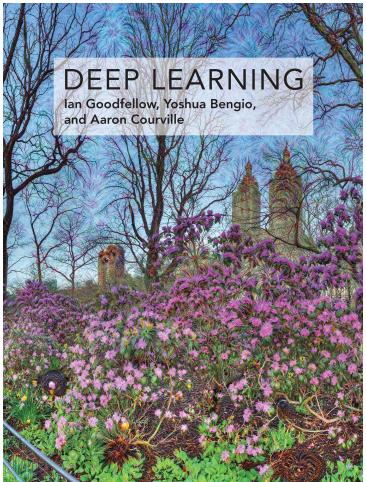
In this tutorial, you will learn how to use the open-source TensorFlow library for deep learning. What's so great about TensorFlow is that it allows us to work with multi-dimensional arrays and train deep neural network very efficiently by utilizing GPU resources.

In this introduction to TensorFlow, you will learn how to define computational graphs and how to execute them in a Python runtime environment. After implementing backpropagation for a simple multi-layer perceptron, we will talk about TensorFlow's convenience features for optimization and the new layers API to construct more complex deep learning architectures more compactly. Finally, we will implement a General Adversarial Networks architecture to see how we can access and update variables from different network graphs and scopes -- the entry point for inventing and experimenting with novel architectures in our real-world applications and research.

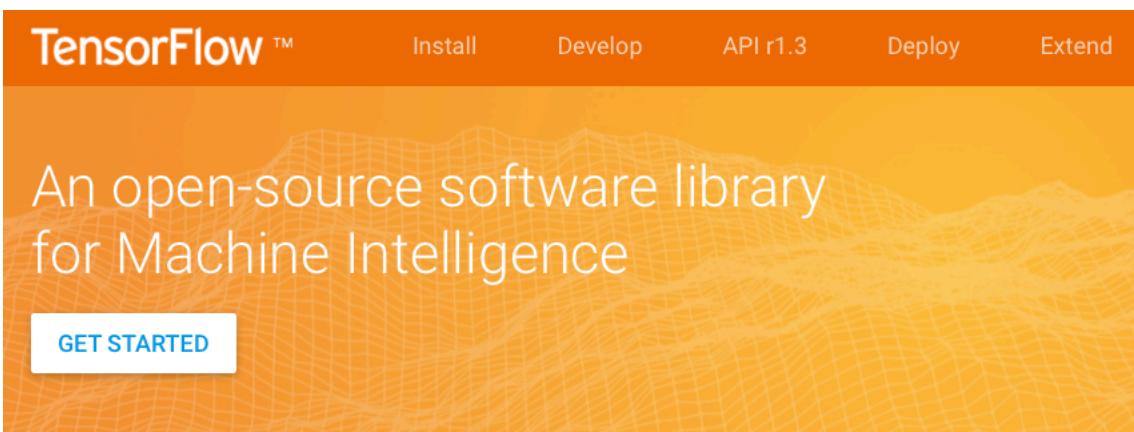
## Code snippets

GitHub: <https://github.com/rasbt/pydata-annarbor2017-dl-tutorial>

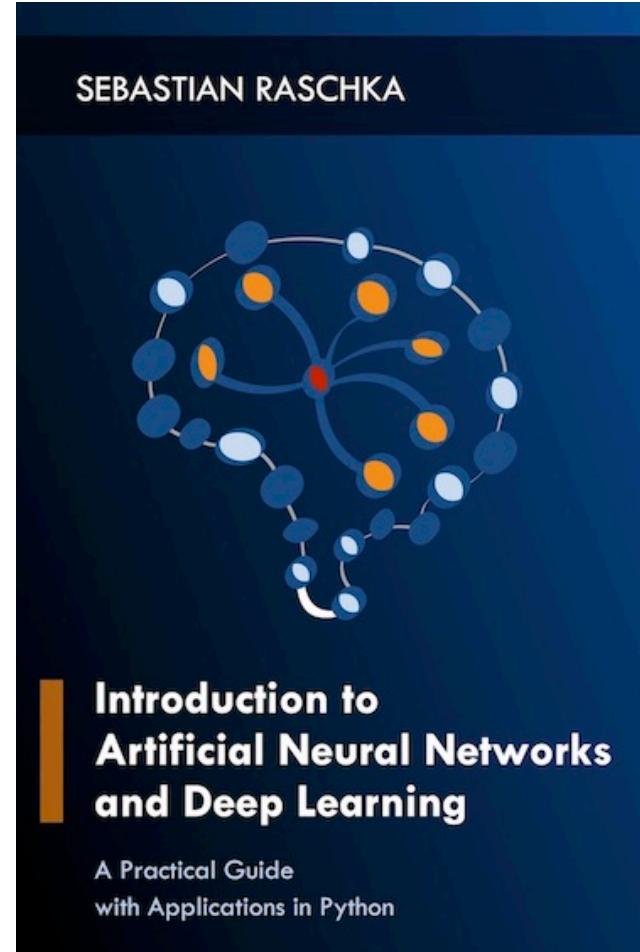
# Useful (and Free) Resources



<http://www.deeplearningbook.org>



<https://www.tensorflow.org>



<https://github.com/rasbt/deep-learning-book>

# One More Thing!

The book cover features a dark blue background with a faint, large profile of a person's head on the right side. At the top left, the title "Python Machine Learning" and subtitle "- Second Edition" are displayed in white. Below the title, there is a brief description of the book's content, followed by a bulleted list of "Things you will learn". On the right side, the authors' names, "Sebastian Raschka & Vahid Mirjalili", are listed above the title. The central part of the cover has a white rectangular area containing the title and authors' names. At the bottom, there is a "Packt" logo, the price (\$49.99 US / £32.99 UK), a barcode, and the ISBN number.

**Python Machine Learning**  
- Second Edition

Machine learning is eating the software world, and now deep learning is extending machine learning. This second edition of Sebastian Raschka's Python Machine Learning is thoroughly updated using the latest, most mature, and most efficient Python open source libraries, so that you can understand and work at the cutting-edge of machine learning, neural networks, and deep learning.

This best-selling book has been extended and modernized to now include the popular TensorFlow deep learning library, and on top, it introduces the popular Keras library as an intuitive API layer that makes creating neural networks even more convenient. The scikit-learn code has also been fully updated to include recent improvements and additions to this versatile machine learning library. The result is a new edition of this classic book at the cutting edge of machine learning.

If you're new to machine learning, you'll find that this classic book offers the practical knowledge and rich techniques you need to create and contribute to machine learning, deep learning, and modern data analysis. Raschka and Mirjalili introduce you to machine learning and deep learning algorithms, and show you how to apply

Things you will learn:

- Use the key frameworks for data science, machine learning, and deep learning
- Ask new questions of your data through machine learning models and neural networks
- Work with the most powerful Python open source libraries in machine learning
- Implement deep neural networks using the TensorFlow and Keras deep learning libraries
- Embed your machine learning model in accessible web applications
- Predict continuous target outcomes using regression analysis
- Uncover hidden patterns and structures in data with clustering
- Analyze images using deep learning techniques
- Use sentiment analysis to delve deeper into textual and social media data

EXPERT INSIGHT

**Sebastian Raschka  
& Vahid Mirjalili**

**Python Machine Learning - Second Edition**  
Sebastian Raschka, Vahid Mirjalili

Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow

**Second Edition**

Packt

<https://www.amazon.com/Python-Machine-Learning-scikit-learn-TensorFlow/dp/1787125939/>

# Thanks for attending!

## Slides

Speaker Deck:

<https://speakerdeck.com/rasbt/introduction-to-deep-learning-with-tensorflow-at-pydata-ann-arbor>

## Code snippets

GitHub:

<https://github.com/rasbt/pydata-annarbor2017-dl-tutorial>

## Contact:

- E-mail: [mail@sebastianraschka.com](mailto:mail@sebastianraschka.com)
- Website: <http://sebastianraschka.com>
- Twitter: [@rasbt](https://twitter.com/rasbt)
- GitHub: [rasbt](https://github.com/rasbt)

# Thanks for attending!

Slides

Speaker Deck:

<https://speakerdeck.com/rasbt/introduction-to-deep-learning-with-tensorflow-at-pydata-ann-arbor>

Questions?

Code snippets

GitHub:

<https://github.com/rasbt/pydata-annarbor2017-dl-tutorial>

Contact:

- E-mail: [mail@sebastianraschka.com](mailto:mail@sebastianraschka.com)
- Website: <http://sebastianraschka.com>
- Twitter: [@rasbt](https://twitter.com/rasbt)
- GitHub: [rasbt](https://github.com/rasbt)