Machine Learing: Neural Networks: Optional: Linear Regression

CPSC 501: Advanced Programming Techniques
Winter 2025

Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Friday, February 21, 2025

UNIVERSITY OF CALGARY

Linear Regression

Model the linear relationship between:

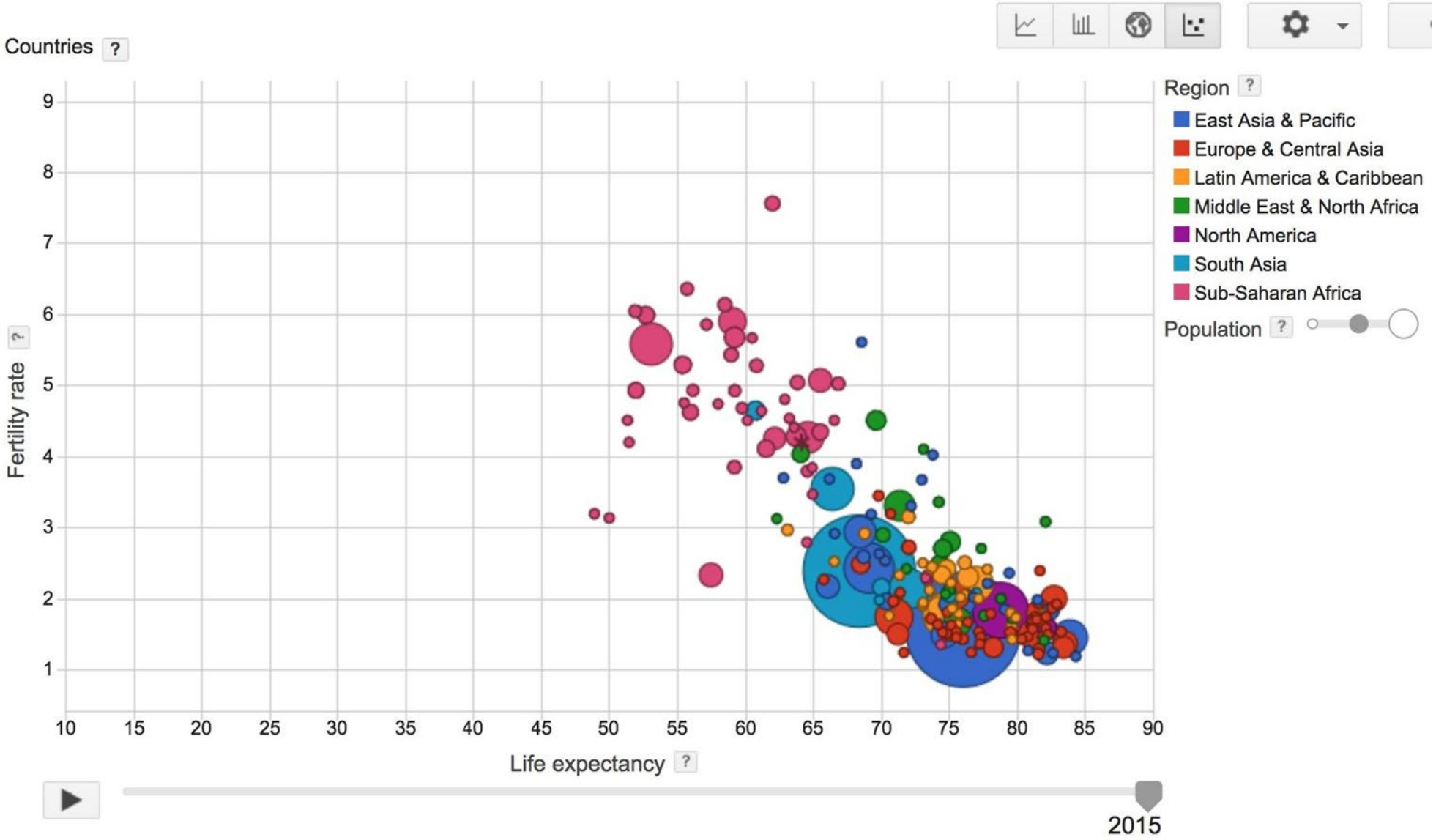
1. dependent variable Y
2. explanatory variables X

- Linear Regression
  - Finding a line of best fit for data points,

- Generally involves proposing different slopes and offsets that define line and finding one that minimizes some error function for how far off points are

World Development Indicators dataset

- X: birth rate
- Y: life expectancy
- 190 countries



[4] Visualization made by Google, based on data collected by World Bank

UNIVERSITY OF CALGARY

Data Set

- birth_life_2010.txt

Country	Birth rate	Life expectancy
Vietnam	1.822	74.828243902
Vanuatu	3.869	70.819487805
Tonga	3.911	72.150658537

…

Want

- Find a linear relationship between X and Y
- to predict Y from X

Model

- Inference: Y_predicted = w * X + b
  - This is the equation of a line (what is Y based on X)
  - W is our slope and b is an offset
- Mean squared error: $E[(y - y_{predicted})^2]$
  - How far off each value y is from where the line said it should be

Phase 1: Create Model

1. Read data file to use as X and Y
2. Create weight/bias variables, w and b
3. Inference/prediction (that there is a line that fits these)
4. Loss function (squared differential)
5. Optimizer (gradient descent)

Base Code (TF2)

```
data, n_samples = read_birth_life_data("birth_life_2010.txt")
dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))
w = tf.Variable(0.0)
b = tf.Variable(0.0)
def prediction(x):
    return x * w + b
def squared_loss(y, y_predicted):
    return (y - y_predicted) ** 2
optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)
```

Store data in tf.data.Dataset

- tf.data.Dataset.from_tensor_slices((features, labels))
  - Can create by slicing existing data in parts

- tf.data.Dataset.from_generator(gen, output_types, output_shapes)
  - Or by generating from function
  - Could make normal distribution data, or uniform, etc.

Store data in tf.data.Dataset

- tf.data.Dataset.from_tensor_slices((features, labels))

dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))

We use the method that takes file loaded **data** and splits into X/Y

Features → what we learn from, X

Labels → what we've decided the features should be called, Y

Store data in tf.data.Dataset

- tf.data.Dataset.from_tensor_slices((features, labels))

```
dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))
print(dataset.output_types)             # >> (tf.float32, tf.float32)
print(dataset.output_shapes)            # >> (TensorShape([]), TensorShape([]))
```

1. Read data file, slice into X, Y parts in a dataset

```
data, n_samples = read_birth_life_data("birth_life_2010.txt")
dataset = tf.data.Dataset.from_tensor_slices((data[:,0], data[:,1]))
```

2. Create weight and bias

```
w = tf.Variable(0.0)
b = tf.Variable(0.0)
```

3. Inference

```
def prediction(x):
    return x * w + b
```

A line based on two variables slope w, and offset b

Out input X is used to calculate some output Y

Specify loss function

```
def squared_loss(y, y_predicted):
    return (y - y_predicted) ** 2
```

Squared differential (farther away is much worse than slightly off)

Create optimizer

optimizer = tf.keras.optimizers.SGD(learning_rate=0.001)

Gradient descent is a way to minimize an objective parameterized by a model's parameters by updating the parameters in the opposite direction of the gradient of the objective function w.r.t. to the parameters.

The learning rate determines the size of the steps we take to reach a (local) minimum.

Phase 2: Train our model

1. Train model using optimizer

Base Code

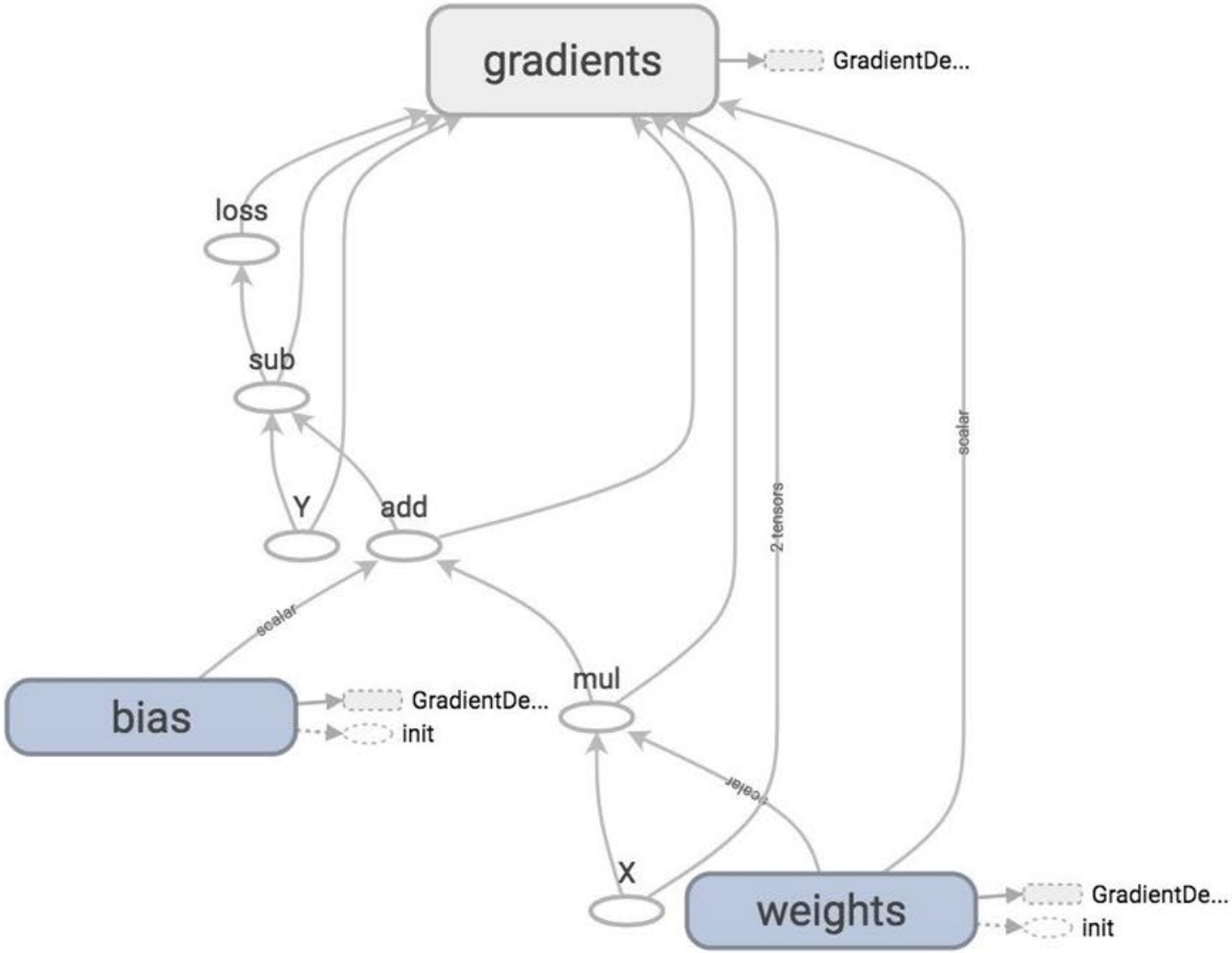
```
EPOCHS = 100
for epoch in range(EPOCHS):
  total_loss = 0.0

  for x, y in dataset:
    with tf.GradientTape() as tape:
      l = squared_loss(y, prediction(x))
      total_loss += l
    gradients = tape.gradient(l, [w,b])
    optimizer.apply_gradients(zip(gradients, [w,b]))

  print('Epoch {0}: {1}'.format(epoch, total_loss / n_samples))

print(f'w:{w.numpy()} b:{b.numpy()}')
```

1. Training

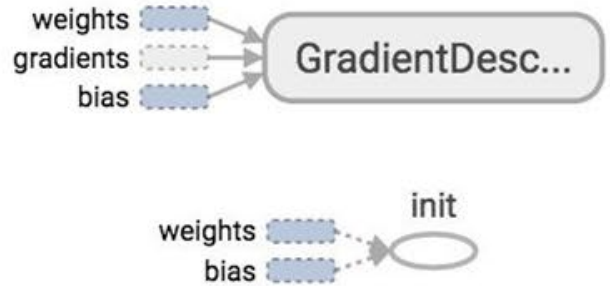
```
for x, y in dataset:
  with tf.GradientTape() as tape:
    l = squared_loss(y, prediction(x))
  gradients = tape.gradient(l, [w,b])
  optimizer.apply_gradients(zip(gradients, [w,b]))
```

For each x,y find out the error, use the error as a gradient to update w,b

Main Graph

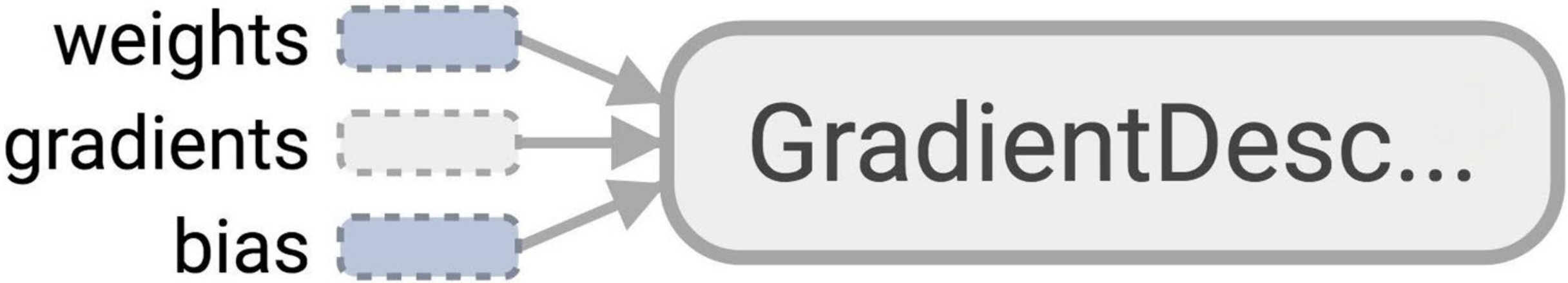


Auxiliary Nodes



Optimizer

- TensorFlow looks at all trainable variables that optimizer depends on and updates them

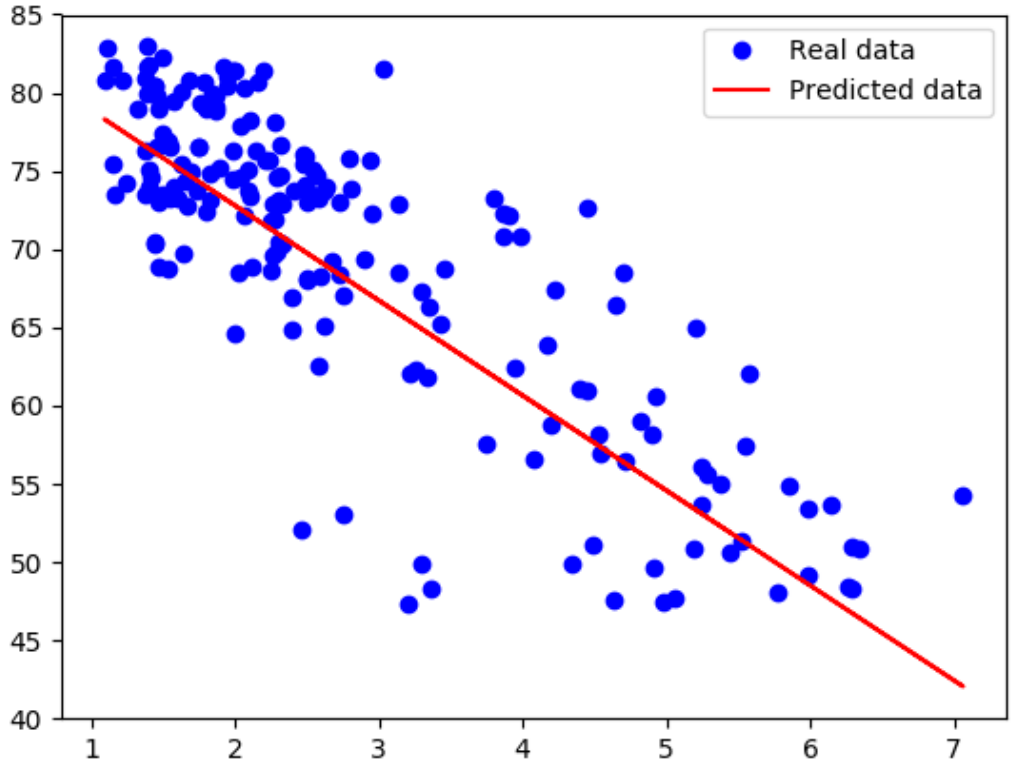


Trainable variables

- tf.Variable(initial_value=None, trainable=True,...)

  Specify if a variable should be trained or not
  By default, all variables are trainable

Plot the results with matplotlib



Same Code in Keras

```
data, n_samples = read_birth_life_data("birth_life_2010.txt")
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1,input_shape=())
  ])
EPOCHS=100
optimizer = tf.keras.optimizers.SGD(learning_rate=0.05)
model.compile(loss='mse',optimizer=optimizer)
model.fit(data[:,0],data[:,1],epochs=EPOCHS)
w = model.layers[1].get_weights()[0][0]
b = model.layers[1].get_weights()[1]
print(w, b)
```

Onward to … MNIST

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY