TensorFlow

Hands-on TensorFlow 2.0

Josh Gordon (@random_forests)
Amit Patankar (@av8ramit24)

At SciPy Tokyo

TensorFlow

Slides from today

bit.ly/scipy-slides

Hands-on workshop

You will need

- A laptop with an internet connection.
- There's nothing to install in advance.

Agenda

- Beginner exercises in the first half
- Advanced examples in the second

Deep Learning is a huge space (our goal is not to cover everything, just to get you started).

TensorFlow

An open source Deep Learning library

- Released by Google in 2015
- >1800 contributors worldwide

TensorFlow 2.0 (we'll use this today!)

- **Easier to use**
- Code styles for beginners and experts
- Alpha released in March, 2019

The person is riding a surfboard in the waves.

Exercises

Exercises

- Installing TF2 and using Colab
- Linear regression
- MNIST (with Keras Sequential)
- MNIST (with Keras Subclassing)
- Structured data

Advanced

- Deep Dream
- Neural machine translation
- Image Colorization

Topics

For beginners and experts

- Keras Sequential
- Keras Subclassing
- Built-in vs custom training loops

Under the hood

- AutoGraph and tf.function
- TF2 vs TF1

Beyond Hello World

- Interlingual representations

Learning more

- Book recommendations

Exercise 1

Linear regression in TensorFlow 2.0

Exercise 1

Goals

- Install TensorFlow 2.0
- Introduce Colab
- **Introduce ingredients** (predict, loss, improve, repeat)

Visit

bit.ly/tf-ws1

Why is Python popular for scientific computing?

Ballpark benchmarks

About how much slower is Python than C?

Ballpark benchmarks

About how much slower is Python than C?

- Multiplying matrices: +/- 100X
- 6 seconds vs. 10 minutes
- Running vs. flying (6 MPH and 600 MPH)

Python is a great choice for scientific computing

- Why?

Ballpark benchmarks

About how much slower is Python than C?

- Multiplying matrices: +/- 100X
- 6 seconds vs. 10 minutes
- Running vs. flying (6 MPH and 600 MPH)

Python is a great choice for scientific computing

- Why?

NumPy

- C performance, Python ease of use

TensorFlow is basically

NumPy

+ GPU / TPU support
+ AutoDiff
+ Utilities to help you write neural networks (layers, optimizers)

TensorFlow

- A C++ engine to accelerate code written in Python.
- **Bonus**: compiled to a graph that can run on devices **without a Python interpreter** **(phones, web browsers)**

You can use TF 2.0 like NumPy

```
import tensorflow as tf # Assuming TF 2.0 is installed

a = tf.constant([[1, 2],[3, 4]])
b = tf.matmul(a, a)

print(b)
# tf.Tensor( [[ 7 10] [15 22]], shape=(2, 2), dtype=int32)

print(type(b.numpy()))
# <class 'numpy.ndarray'>
```

TensorFlow

For beginners and experts

For beginners

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

TF 1.x

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

TF 2.0

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Keras and tf.keras

In my view, the **clearest Deep Learning library** that exists today.

- For fast prototyping, advanced research, and production.

keras.io = reference implementation

- `import keras`

tf.keras = **TensorFlow's implementation** (a superset, built-in to TF, no need to install Keras separately)

- `from tensorflow import keras`

Exercise 2

Fashion MNIST in TensorFlow 2.0

Exercise 2

Goals

- Learn about the Sequential API
- Train a simple image classifier

Visit

bit.ly/tf-ws4

playground.tensorflow.org

For experts

```
class MyModel(tf.keras.Model):
  def __init__(self, num_classes=10):
    super(MyModel, self).__init__(name='my_model')
    self.dense_1 = layers.Dense(32, activation='relu')
    self.dense_2 = layers.Dense(num_classes, activation='sigmoid')

  def call(self, inputs):
    # Define your forward pass here,
    x = self.dense_1(inputs)
    return self.dense_2(x)
```

What’s the difference?

Symbolic vs Imperative APIs

Symbolic (Keras Sequential)

- Your model is a graph of layers
- Any graph you compile will run
- **TensorFlow helps you debug** by catching errors at **compile time**

Symbolic vs Imperative APIs

Symbolic (Keras Sequential)

- Your model is a graph of layers
- Any graph you compile will run
- **TensorFlow helps you debug** by catching errors at **compile time**

Imperative (Keras Subclassing)

- Your model is Python bytecode
- Complete flexibility and control
- Harder to debug / **harder to maintain**

Use a built-in training loop...

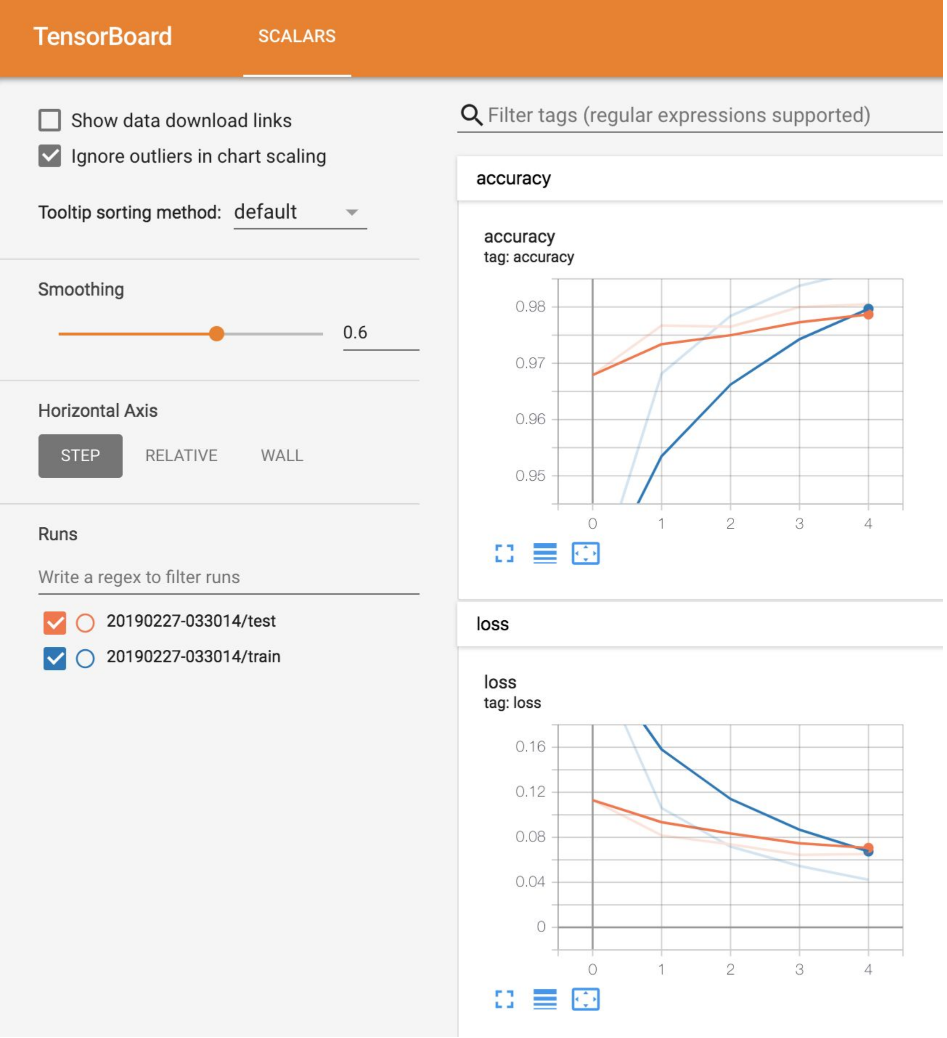
```
model.fit(x_train, y_train, epochs=5)
```

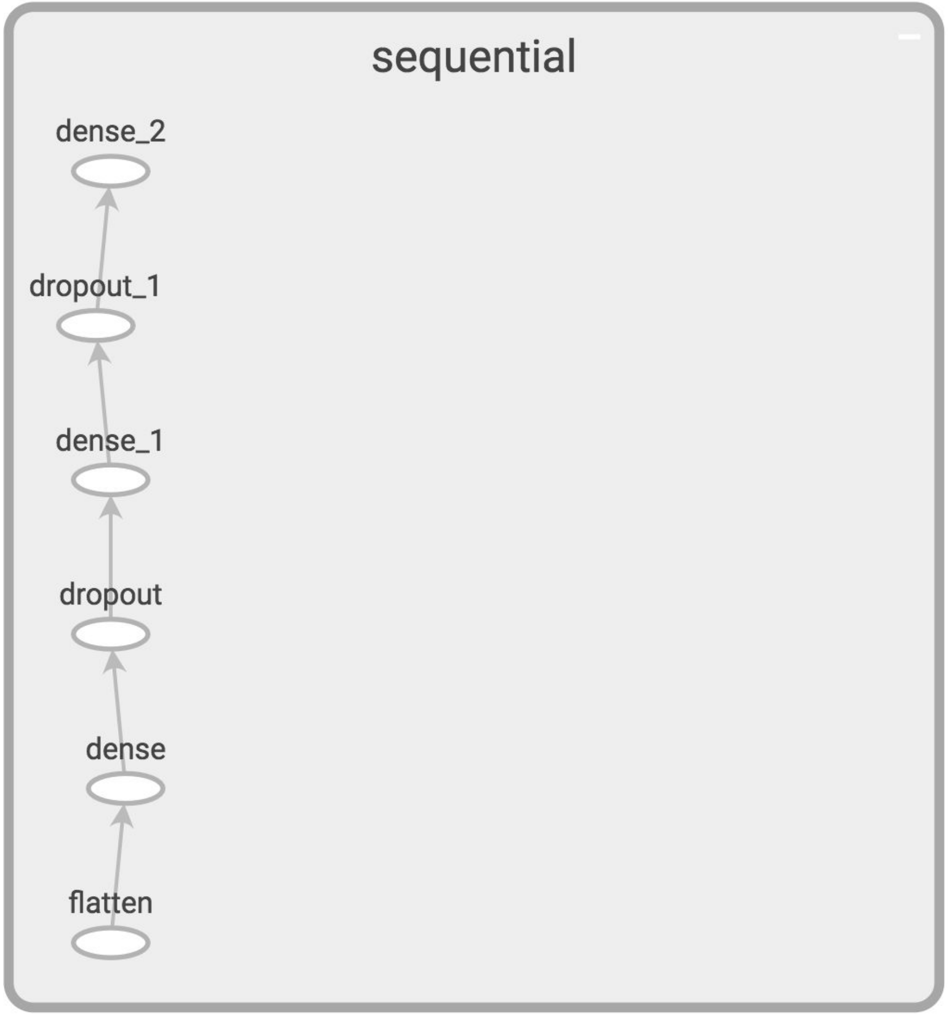
Or define your own

```
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)

grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

TensorBoard

```
tb_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir)

model.fit(
    x_train, y_train, epochs=5,
    validation_data=[x_test, y_test],
    callbacks=[tb_callback])
```


TensorBoard
SCALARS
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
20190227-033014/test
20190227-033014/train
Filter tags (regular expressions supported)
accuracy
accuracy
tag: accuracy
0.98
0.97
0.96
0.95
0
1
2
3
4
loss
loss
tag: loss
0.16
0.12
0.08
0.04
0
0
1
2
3
4


sequential
dense_2
dropout_1
dense_1
dropout
dense
flatten

Exercise 3

Keras model subclassing and TensorBoard

Exercise 3

Goals

- Learn about the Subclassing API
- See TensorBoard running in the browser

Visit

bit.ly/tf-ws3

Note: you may need to replace tf-nightly with !pip install -q tensorflow-gpu==2.0.0-alpha0

How to fix the TensorBoard example

```
!pip install -q tf-nightly-2.0-preview
!pip install -q tensorboard==1.13.0
%load_ext tensorboard.notebook
```

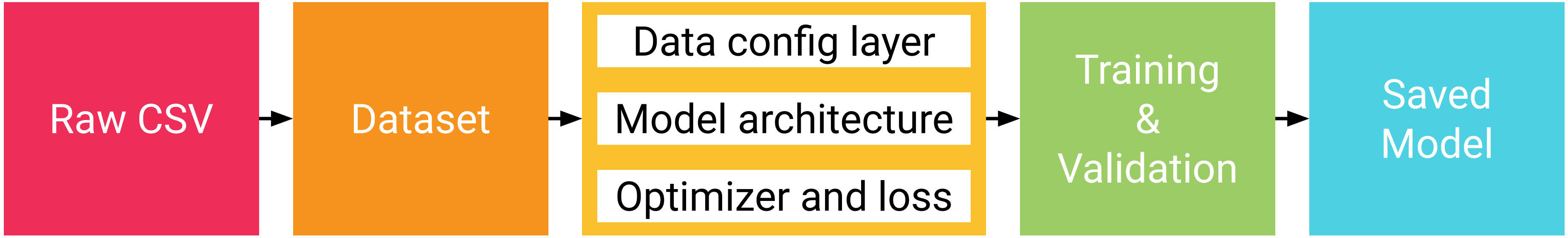
bit.ly/tf-ws3

(Thanks Amit!)

TensorFlow

Structured data

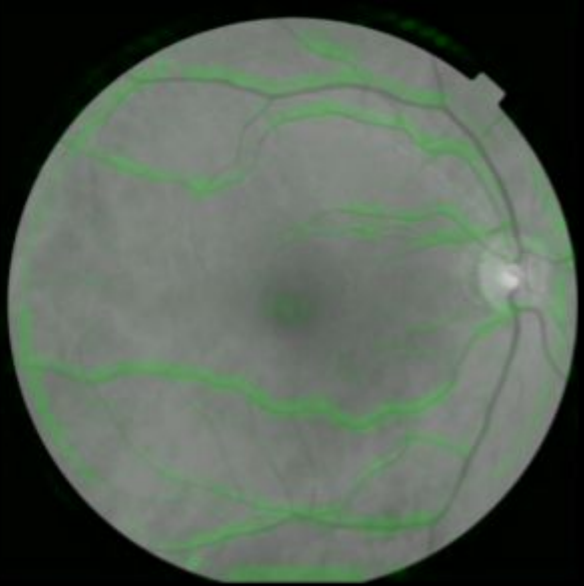
Multi-stage Process



Python

```
interestingNumbers = {
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
}

largest = 0
for (kind, numbers) in interestingNumbers.items:
    for x in numbers:
          if x > largest:
             largest = x

print(largest)
```

Image of retina



Blood pressure predictions focus on blood vessels

Our Dataset

Data: Heart Disease [V.A. Medical Center]

Task: Binary Classification (Healthy/Heart Disease)

Number of examples: ~300

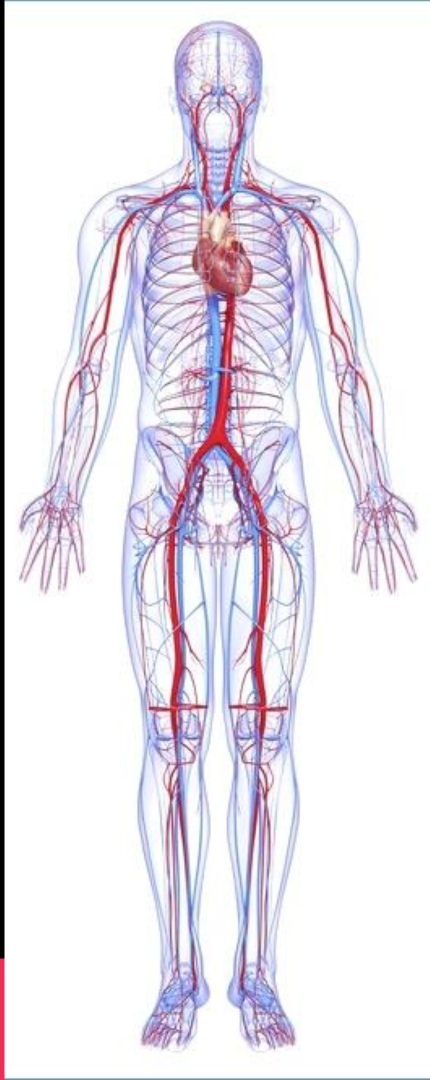
Features:

- **Real:** *Age, Blood Pressure, Cholesterol*
- **Categorical - Int:** *Gender, EKG Results*
- **Categorical - String:** *Thallium heart scan*

Our Features Explained

1. **cp:** Chest pain type
2. **trestbps:** Resting blood pressure
3. **chol:** Serum cholesterol
4. **fbs:** Blood sugar > 120
5. **restecg:** Type of EKG result
6. **thalach:** Max heart rate achieved
7. **exang:** Exercise induced angina
8. **oldpeak:** ST depression (exercise induced)
9. **slope:** Slope of peak ST segment
10. **ca:** # of vessels colored by fluoroscopy
11. **thal:** Thallium heart scan results

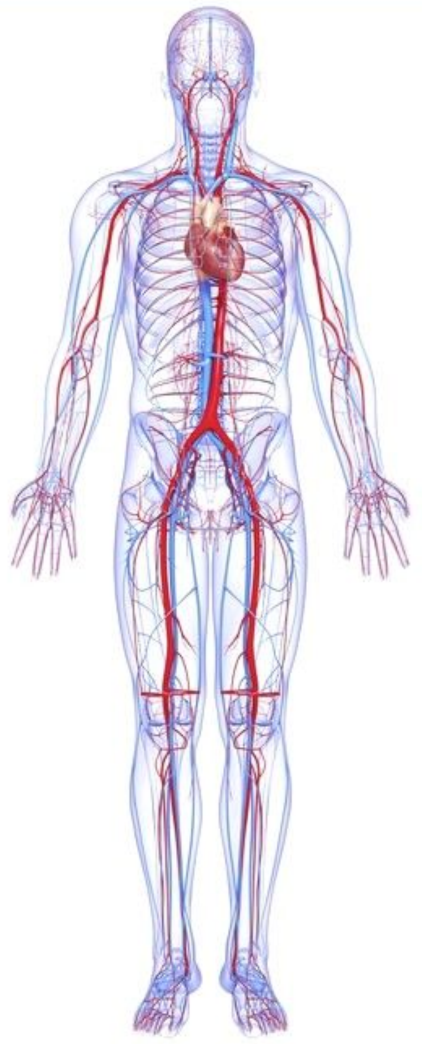
Raw CSV



The Data

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	1	145	233	1	2	150	0	2.3	3	0	fixed	0
67	1	4	160	286	0	2	108	1	1.5	2	3	normal	1
67	1	4	120	229	0	2	129	1	2.6	2	2	reversible	0
37	1	3	130	250	0	0	187	0	3.5	3	0	normal	0
41	0	2	130	204	0	2	172	0	1.4	1	0	normal	0
56	1	2	120	236	0	0	178	0	0.8	1	0	normal	0
62	0	4	140	268	0	2	160	0	3.6	3	2	normal	1

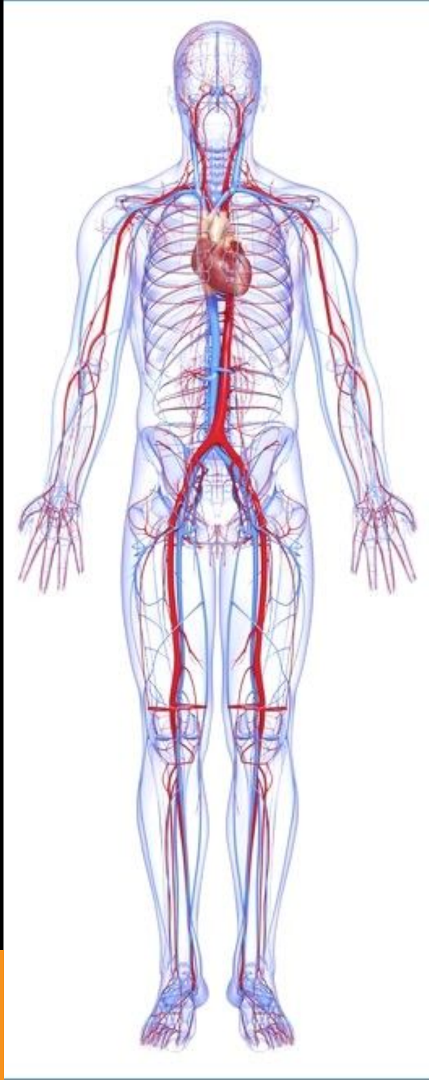
Raw CSV



Loading data

```
# Match CSV data types by column
defaults = [tf.int32, tf.int32 ... tf.string, tf.float32]

dataset = tf.contrib.data.CsvDataset(
            ['heart.csv.train'], defaults, header=True)
```

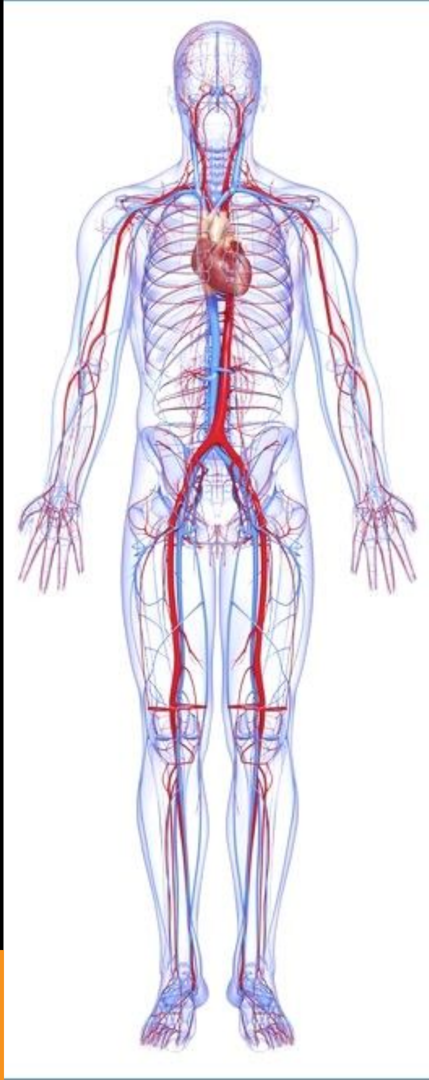
Dataset



Loading data

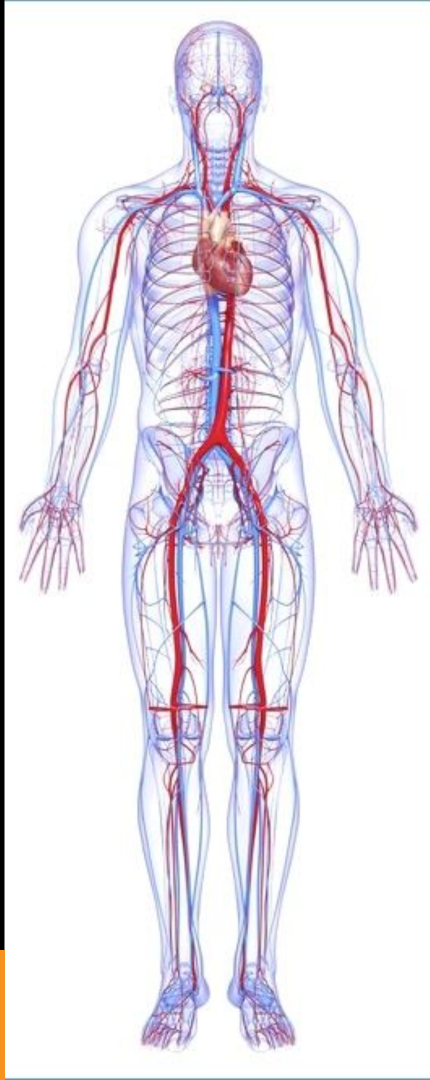
```
print(list(dataset.take(1)))

[(<tf.Tensor: id=188, shape=(), dtype=int32, numpy=63>,
  <tf.Tensor: id=189, shape=(), dtype=int32, numpy=1>,
  <tf.Tensor: id=190, shape=(), dtype=int32, numpy=145>, ...
  <tf.Tensor: id=191, shape=(), dtype=string, numpy='fixed'>,
  <tf.Tensor: id=192, shape=(), dtype=float32, numpy=1>)]
```

Dataset



Parsing data

```
def _parse_csv_row(*vals):
    # Format each row for the model input



    return features, labels
```

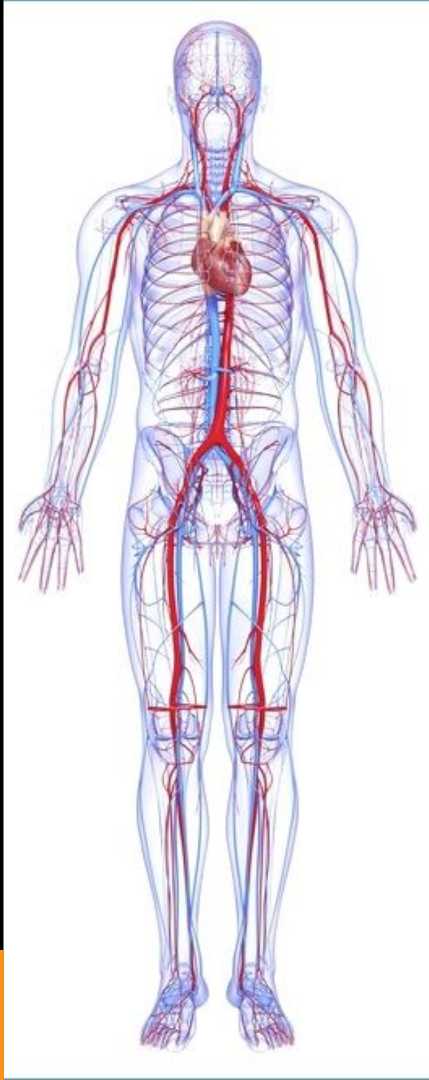


Dataset

Parsing data

```
col_names = ['age', 'sex', 'cp', 'trestbps'...]

def _parse_csv_row(*vals):
    # Format each row for the model input
    # Element of val is a tensor
    features = dict(zip(col_names, vals[:-1]))
    labels =  vals[-1]

    return features, labels
```

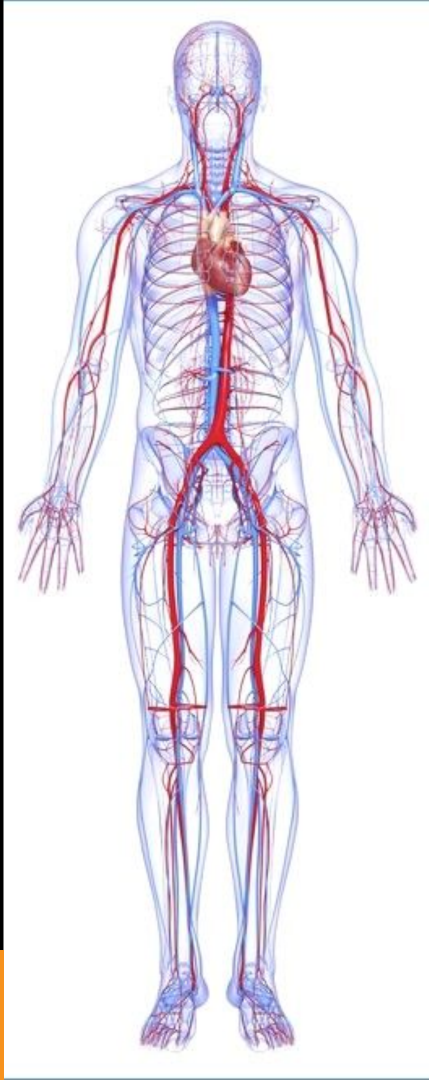
Dataset



Parsing data

```
dataset = dataset.shuffle(TRAINING_SIZE)
dataset = dataset.map(_parse_csv_row).batch(BATCH_SIZE)
```

Dataset

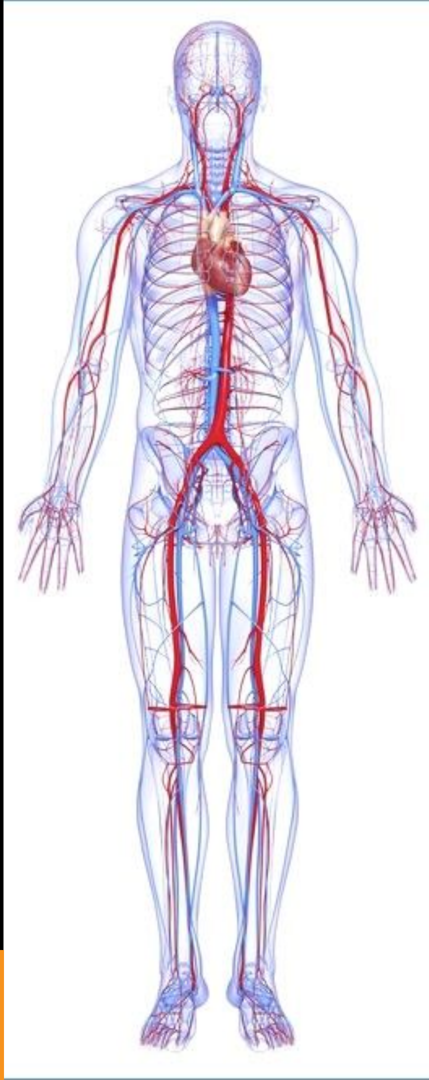


Parsing data

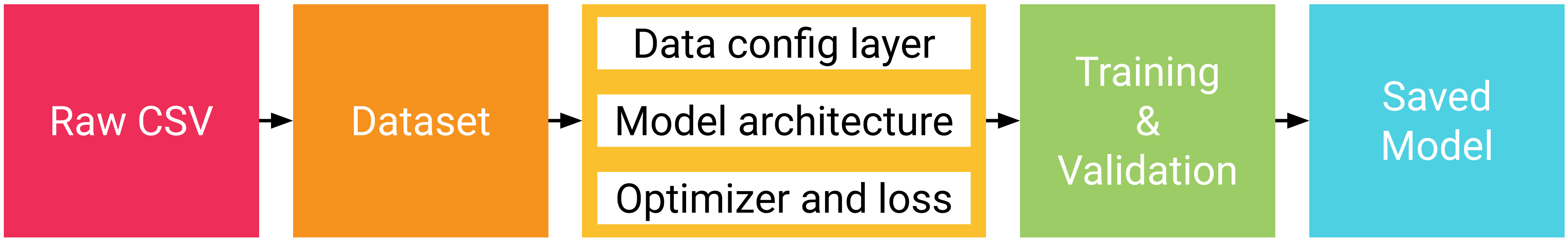
```
dataset = dataset.shuffle(TRAINING_SIZE)
dataset = dataset.map(_parse_csv_row).batch(BATCH_SIZE)
print(list(dataset.take(1)))
```

```
({'age': <tf.Tensor: shape=(64,), dtype=int32,
     array([47, ... 77, 32, 56])>,
   ...
  'thal': <tf.Tensor: shape=(64,),dtype=string
     array([['reversible', ... 'normal'])>},

 <tf.Tensor: shape=(64,), dtype=float64,
     array([0, 0, 1, ... 1, 0, 1])>)
```

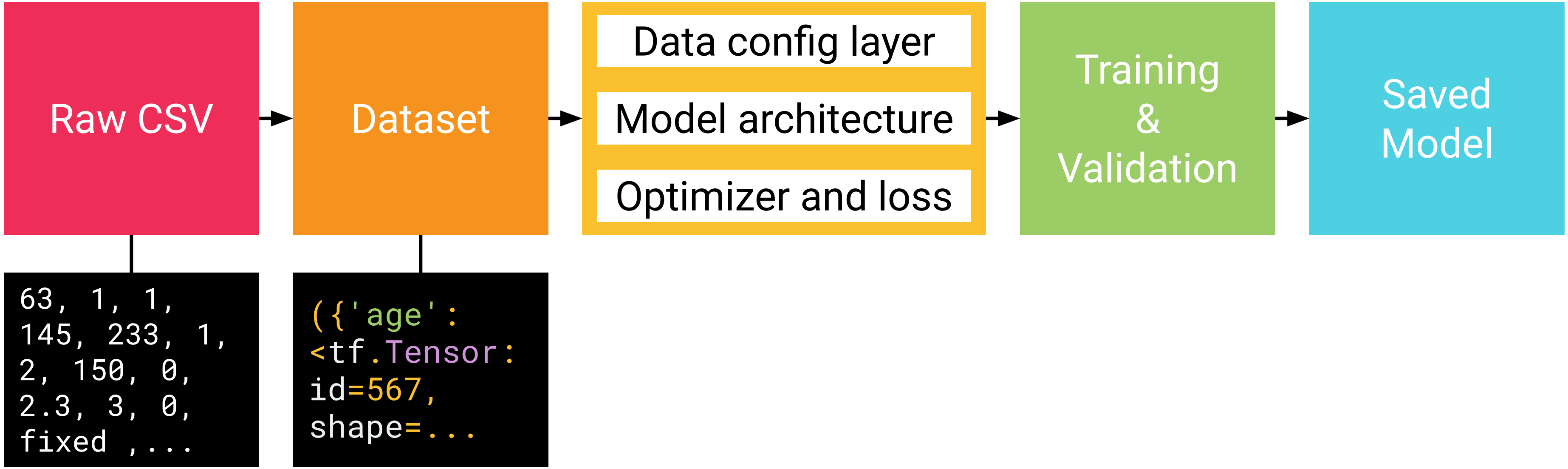
Dataset



Multi-stage Process



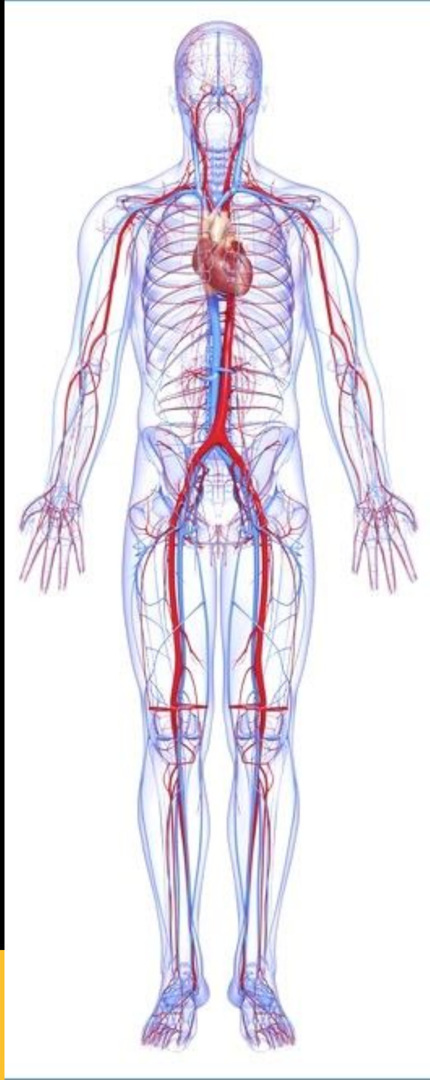
Multi-stage Process



Feature columns

- Numeric columns
  - Age, income, weight
- Bucketized columns
  - Decades, Age in ranges
- Categorical identity columns
  - Gender (0/1)
- Categorical vocabulary column
  - Countries (USA, Canada, Mexico)
- Hashed column
  - Object names
- Crossed column
  - Age along with gender

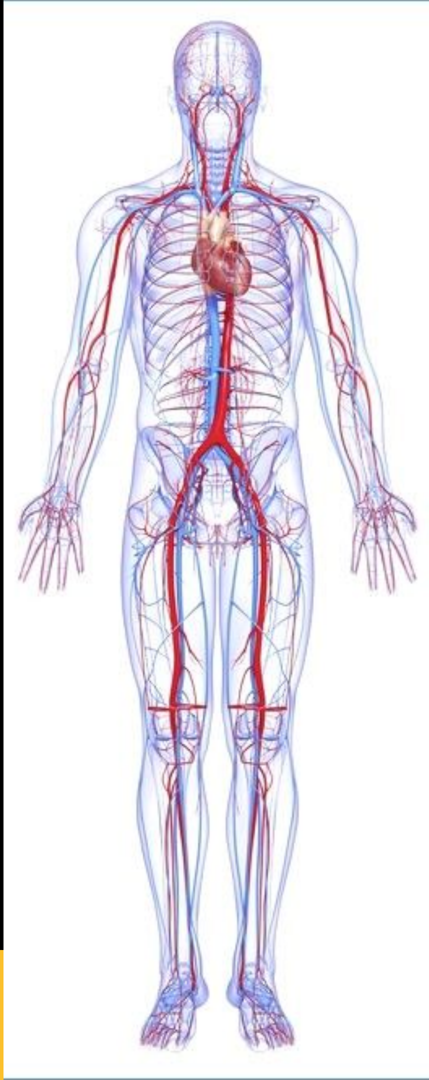
Model



Defining categorical features

```
# thal / string / 1 of 3 values
vocab = ['normal', 'fixed', 'reversible']
thal_cc = tf.keras.feature_column.
  categorical_column_with_vocabulary_list(
  'thal', vocabulary_list=vocab)
```

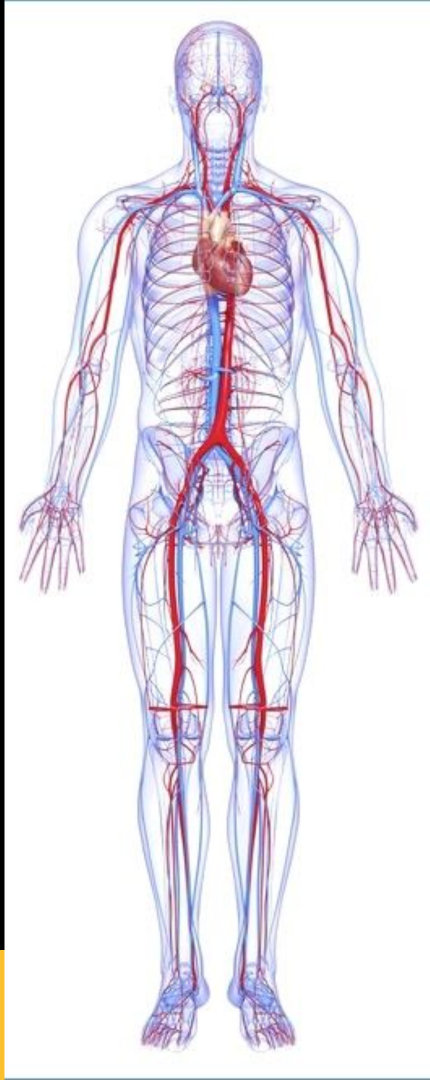
Model



Defining categorical features

```
# thal / string / 1 of 3 values
vocab = ['normal', 'fixed', 'reversible']
thal_cc = tf.keras.feature_column.
  categorical_column_with_vocabulary_list(
  'thal', vocabulary_list=vocab)
thal_embedding = tf.keras.feature_column.
  embedding_column(thal_cc, dimension=3)
```

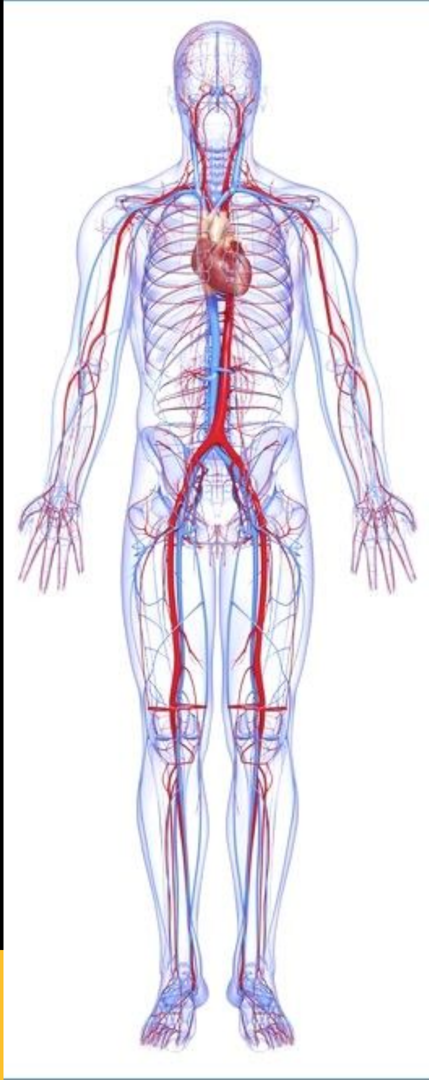
Model



Defining numeric features

```
# age / real integers
age_nc = tf.keras.feature_column.
  numeric_column('age')
```

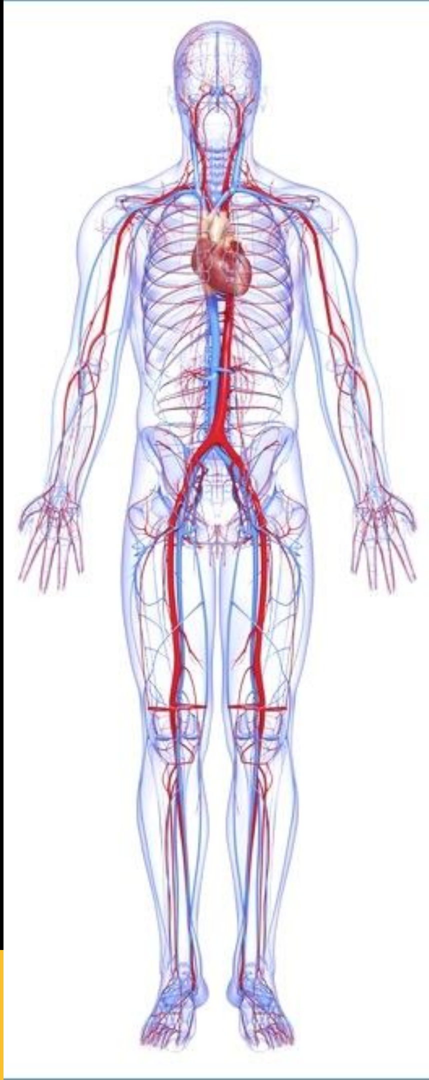
Model



Defining feature columns

```
feature_columns = [age_nc, sex_embedding,...
, thal_embedding]
```

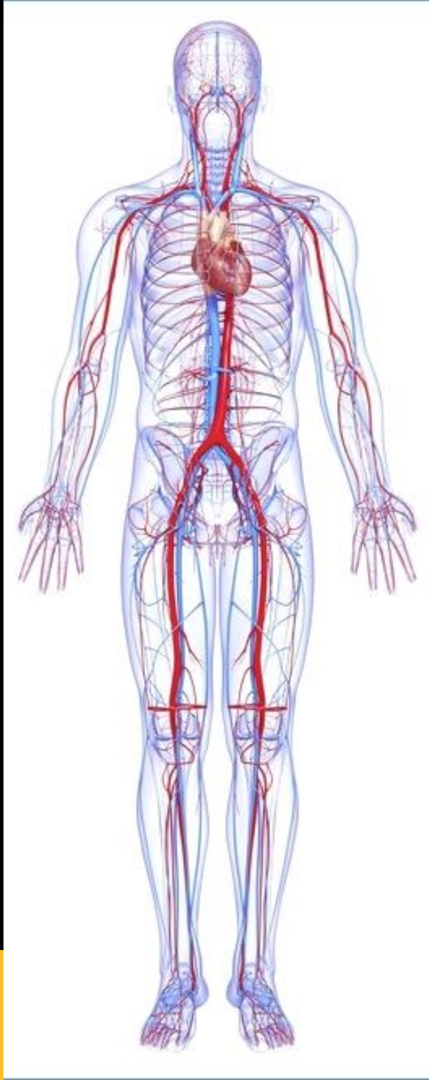
Model



Defining feature columns

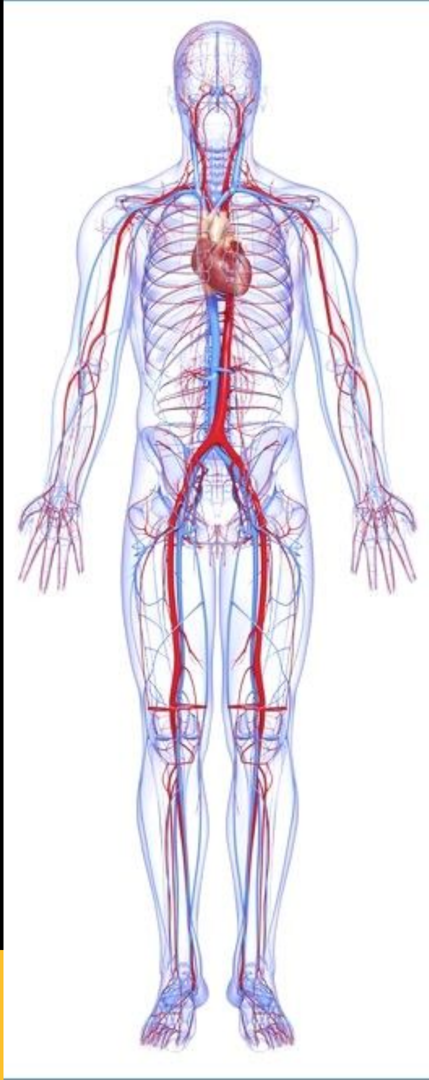
```
feature_columns = [age_nc, sex_embedding,...
, thal_embedding]

feature_layer = tf.python.feature_column.
  FeatureLayer(feature_columns)
```

Model



Building a model

```
model = tf.keras.Sequential([
    feature_layer,
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(64, activation=tf.nn.relu),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

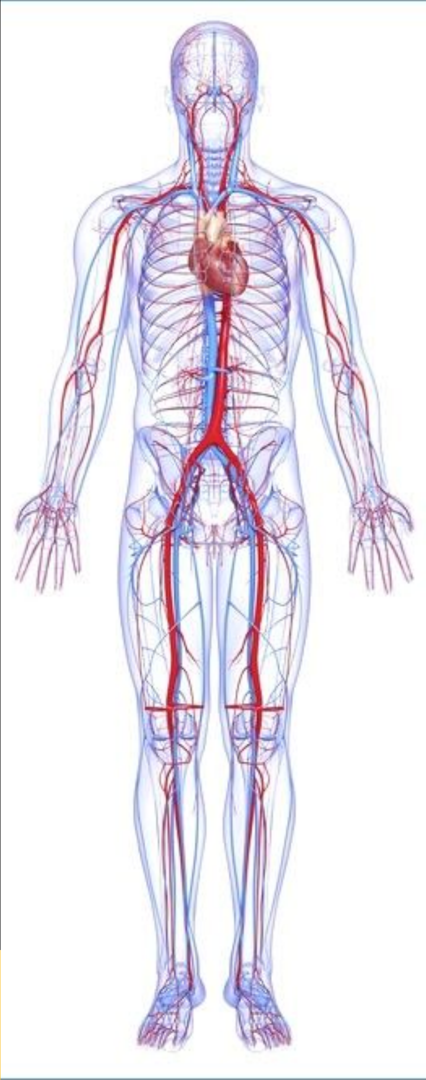


Model

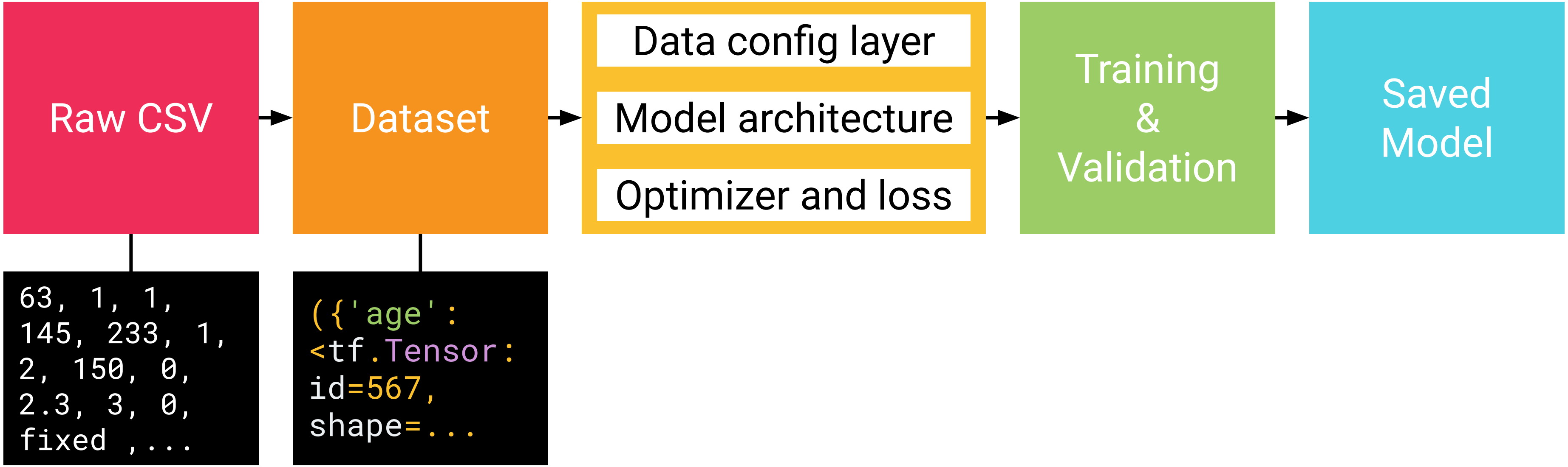
Building a model

```
model.compile(
    optimizer=tf.train.AdamOptimizer(),
    loss=tf.keras.losses.binary_crossentropy,
    metrics=['accuracy'])
```

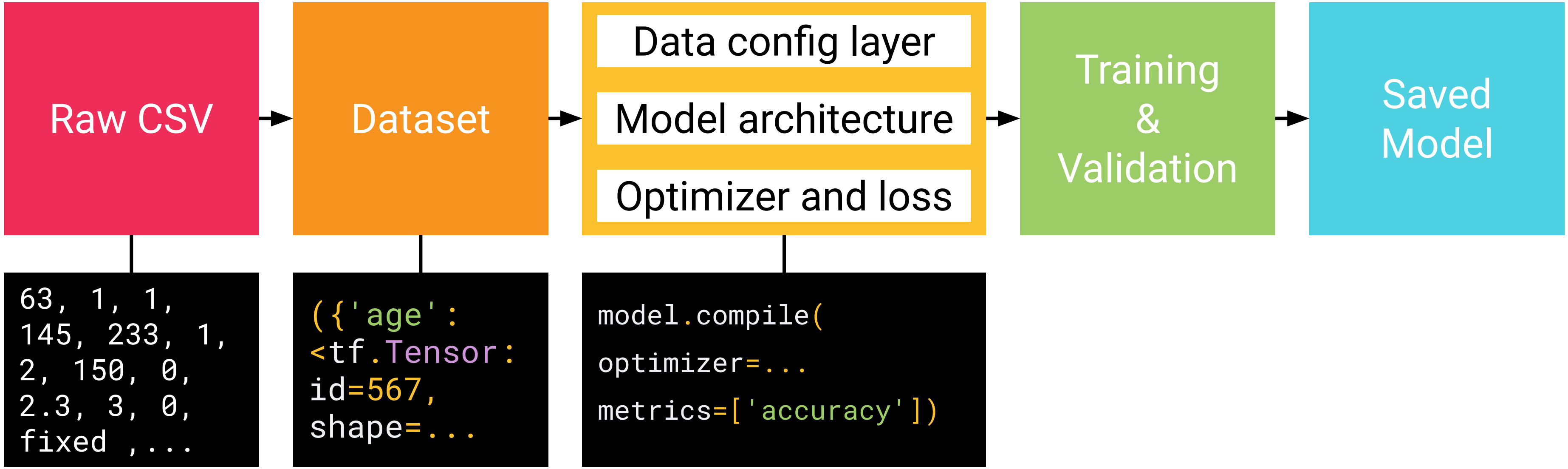
Model



Multi-stage Process



Multi-stage Process

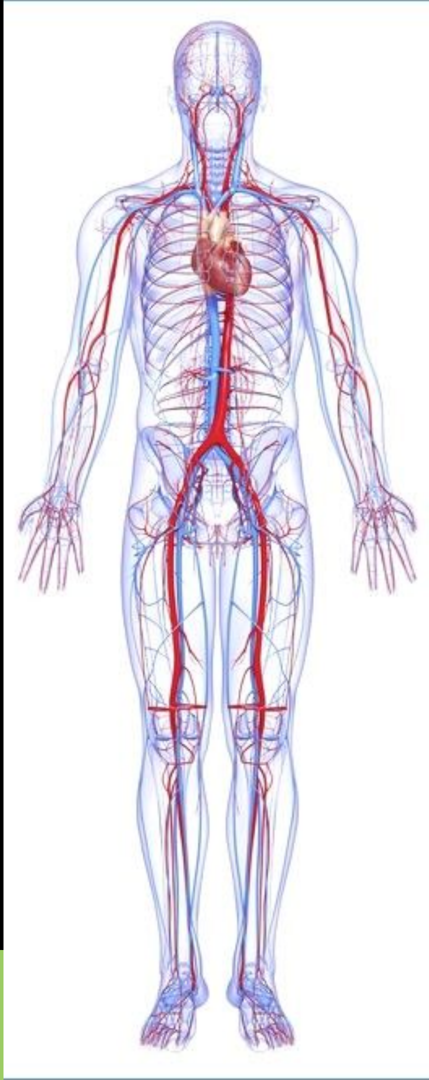


Training a model

```
for epoch in range(1,21):
    # Print epoch, check validation metrics
    model.fit(dataset,
        steps_per_epoch=TRAINING_SIZE/BATCH_SIZE)
```

```
Epoch 20:
8/8 [====================] - 1s 41ms/step -
loss: 0.3475 - acc: 0.8433
```

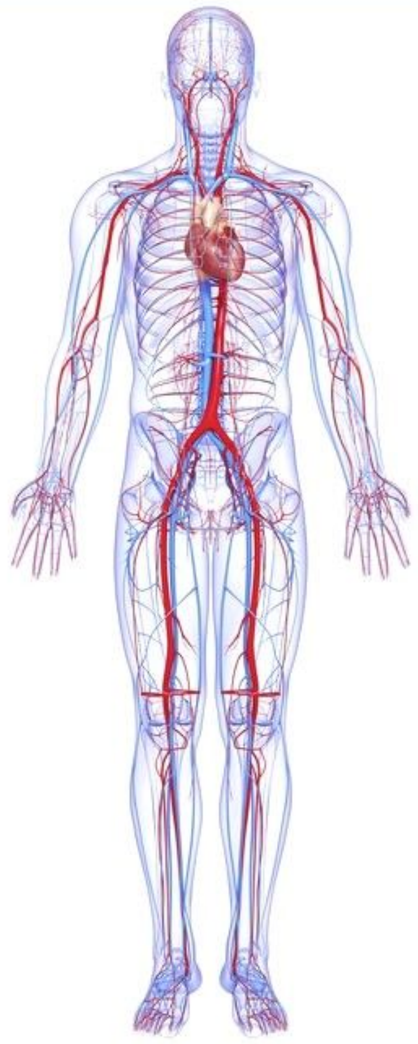
Training
&
Validation



Validating our model

```
test_ds = tf.contrib.data.CsvDataset(
            ['heart.csv.test'], defaults)
test_ds = test_ds.map(_parse_csv_row).batch(50)

loss, accuracy = model.evaluate(test_ds, steps=1)
print(“Loss: {}\nAccuracy: {}”.format(loss,
accuracy)
```

Training
&
Validation

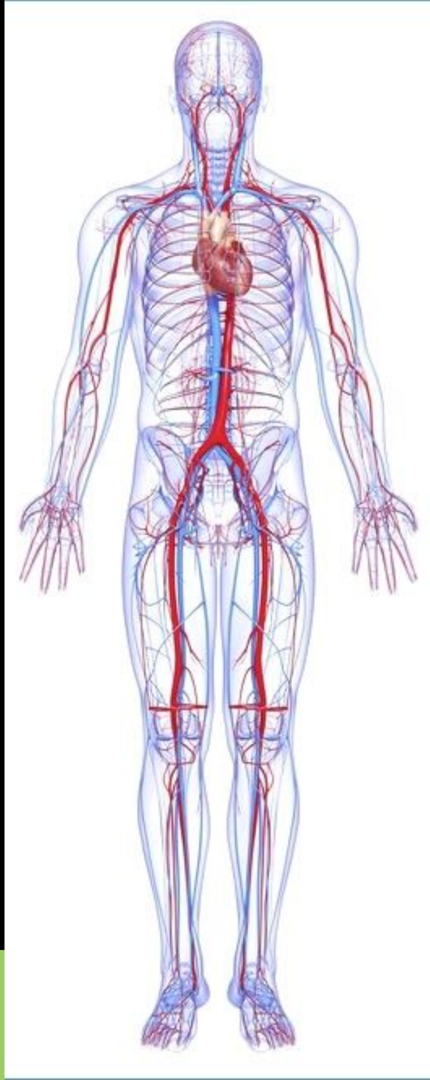


Validating our model

```
test_ds = tf.contrib.data.CsvDataset(
              ['heart.csv.test'], defaults)
test_ds = test_ds.map(_parse_csv_row).batch(50)
```

```
loss, accuracy = model.evaluate(test_ds, steps=1)
print(“Loss: {}\nAccuracy: {}”.format(loss,
accuracy)
```

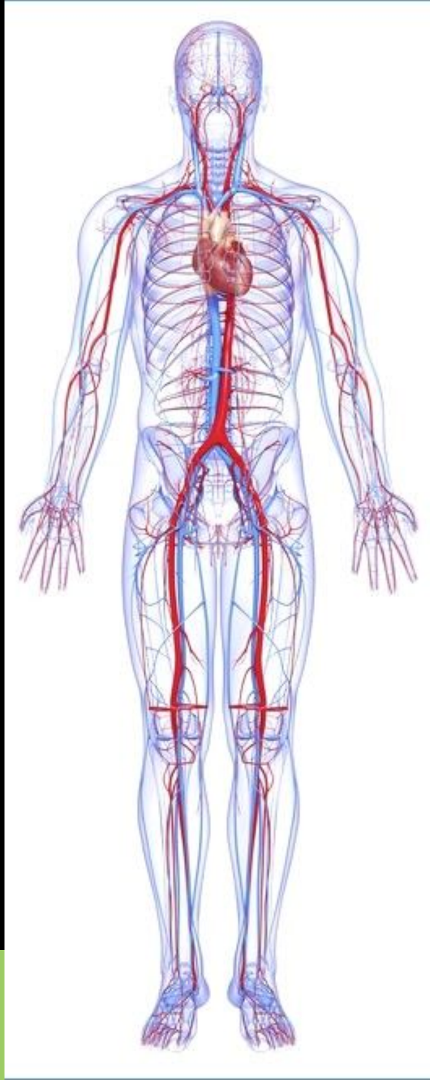
Training
&
Validation



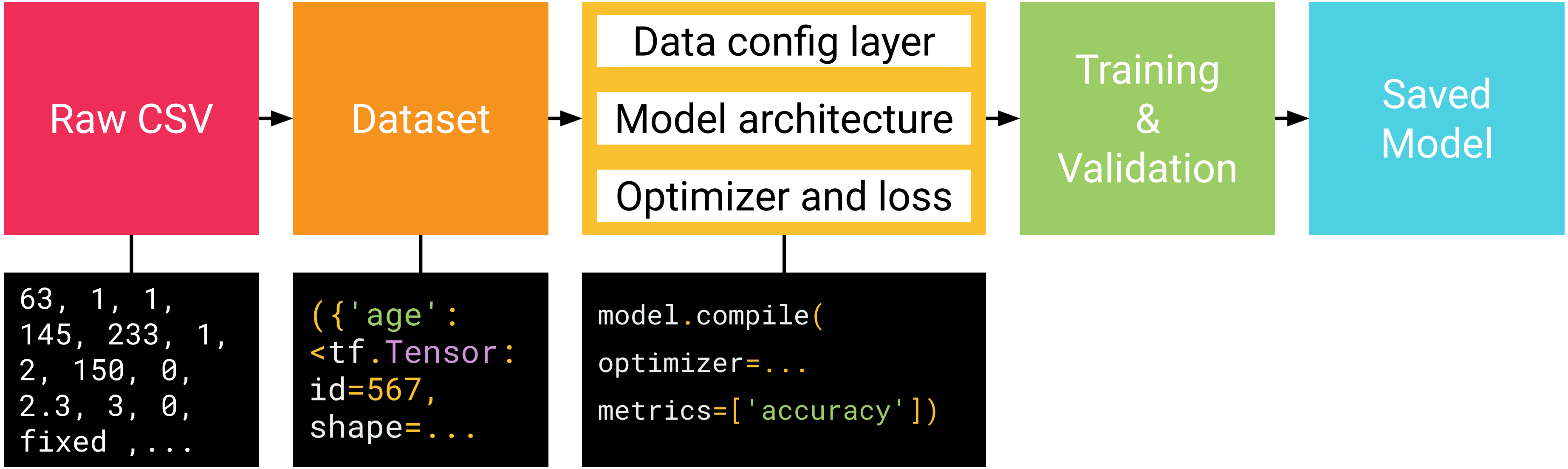
Validating our model

```
test_ds = tf.contrib.data.CsvDataset(
           ['heart.csv.test'], defaults)
test_ds = test_ds.map(_parse_csv_row).batch(50)

loss, accuracy = model.evaluate(test_ds, steps=1)
print(“Loss: {}\nAccuracy: {}”.format(loss,
accuracy)

Loss: 0.4622
Accuracy: 0.8400
```

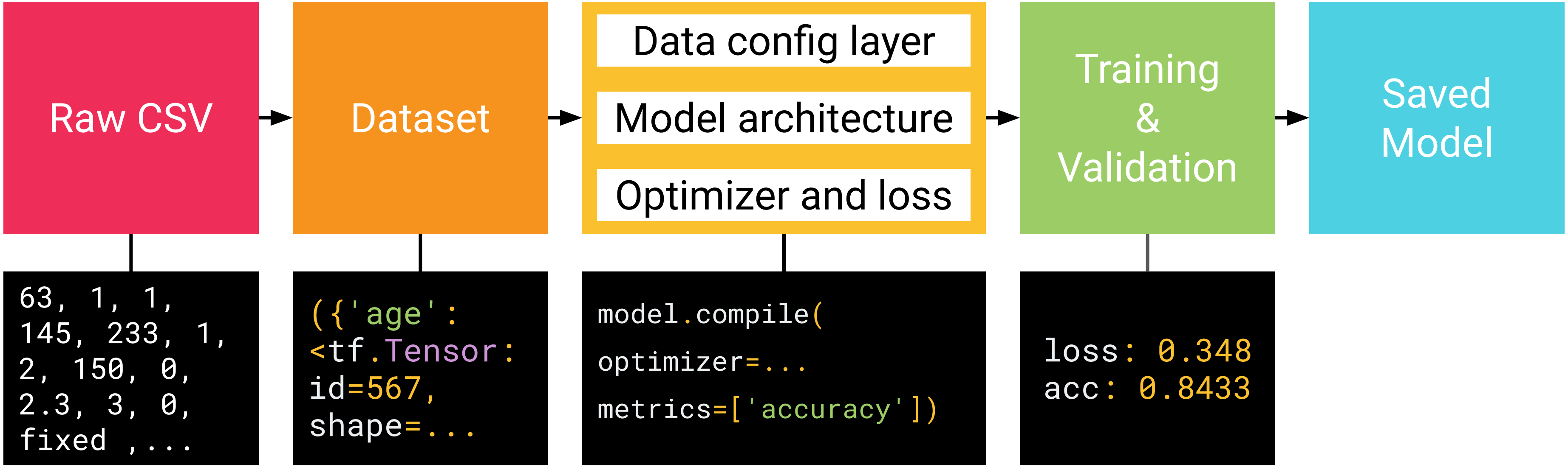
Training
&
Validation



Multi-stage Process



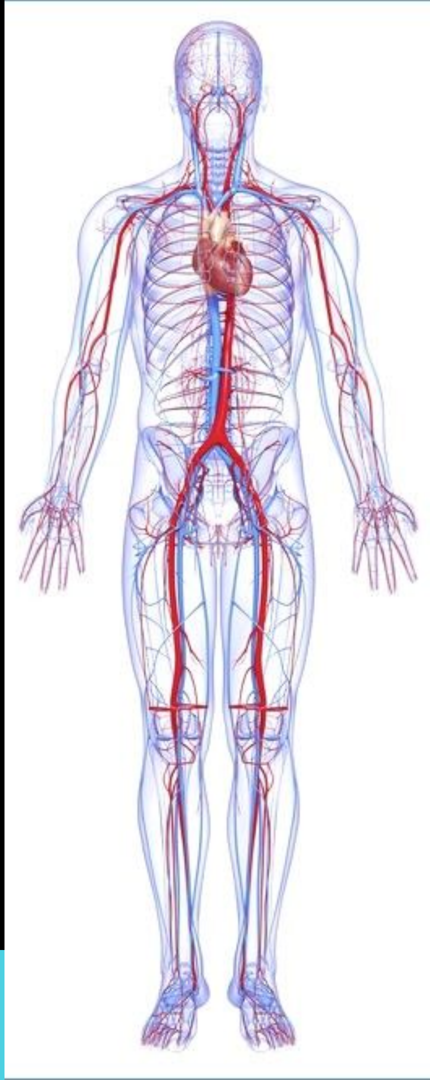
Multi-stage Process



Export to SavedModel

```
export_dir = tf.contrib.saved_model.
  save_keras_model(model, 'keras_nn')
```

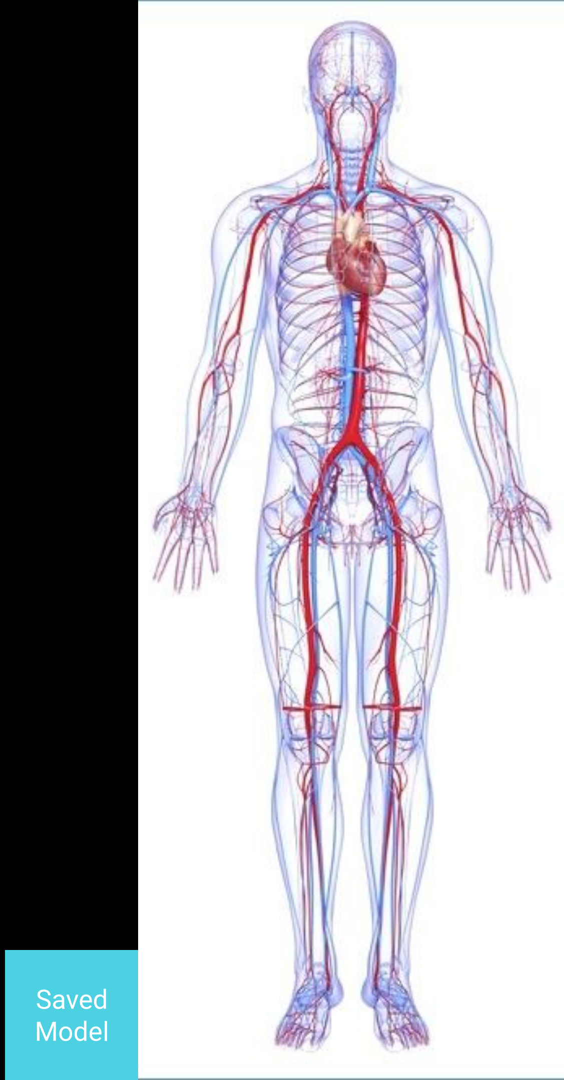
```
keras_nn/
  1536162174/
    saved_model
    variables/
    assets/
```



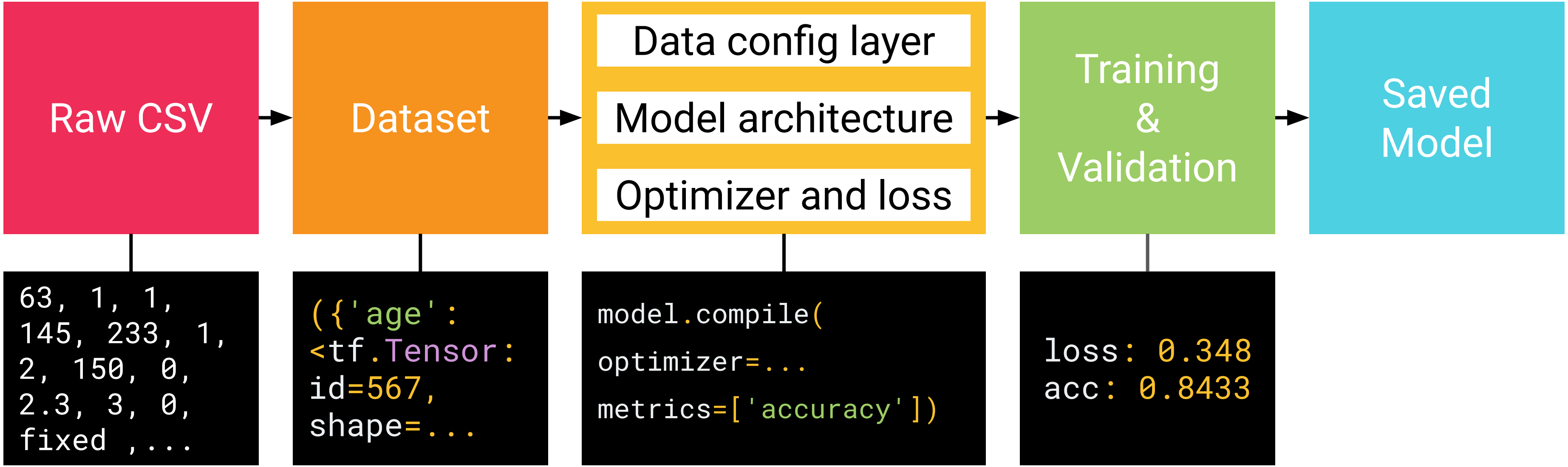
Saved Model

Restore from SavedModel

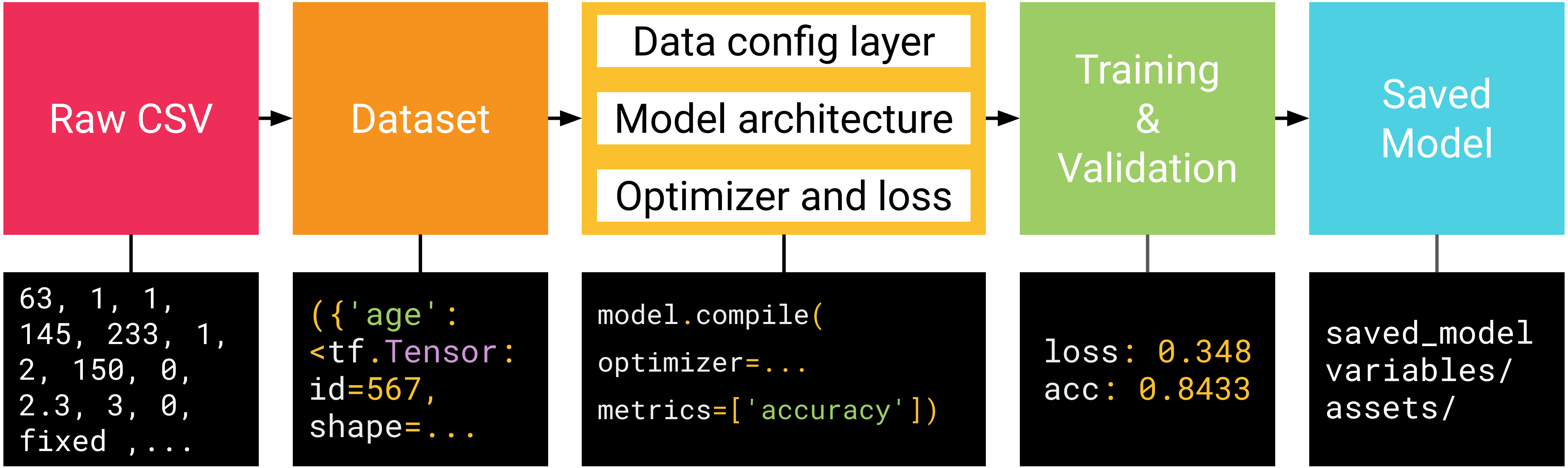
```
restored_model = tf.contrib.saved_model.
  load_keras_model(export_dir)
```



Multi-stage Process



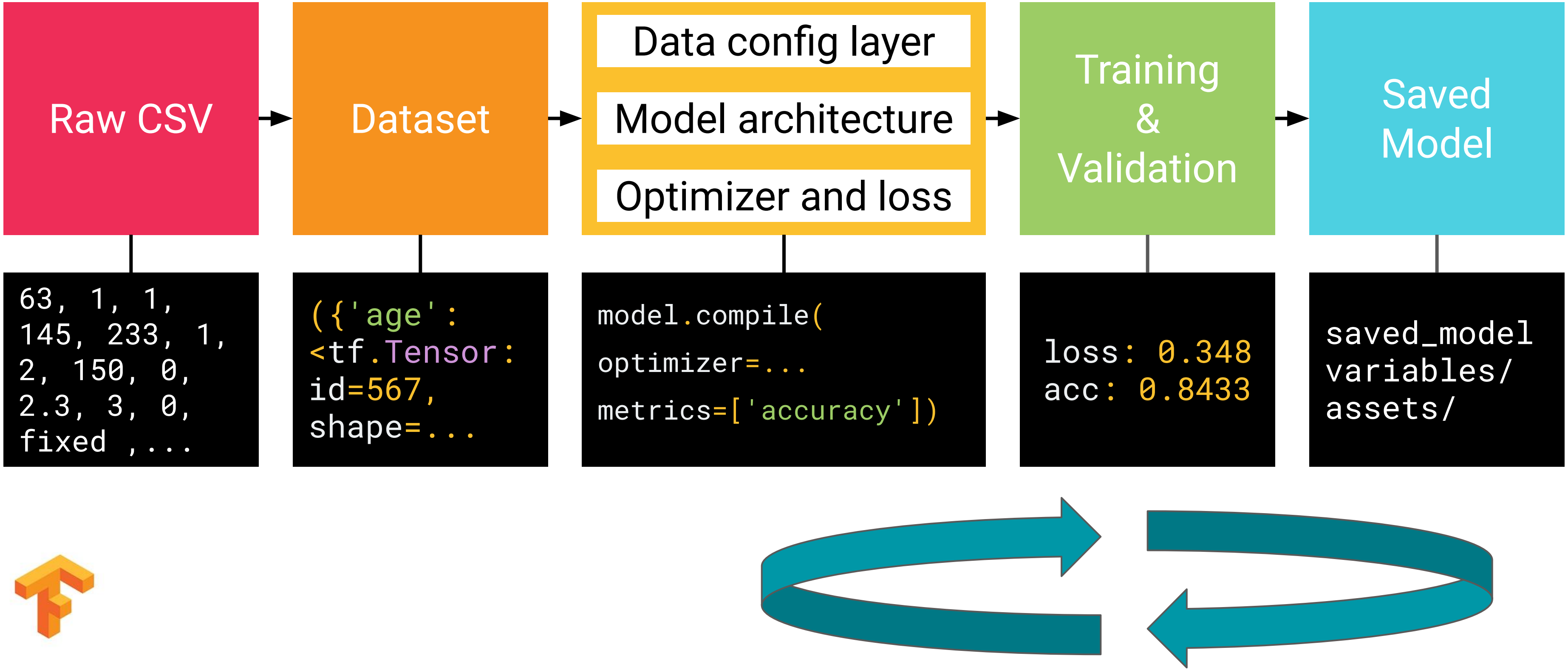
Multi-stage Process



Conclusions

- Prototype with **Eager**.
- Preprocess with **Datasets**.
- Transform with **Feature Columns**.
- Build with **Keras**.
- Package with **SavedModel**.

Multi-stage Process



Exercise 4

Structured data

Exercise 4

Goals

- Start from raw data
- Create a model

Visit

bit.ly/tf-ws4a

Facets pair-code.github.io/facets/

Beyond Hello World

A few of my favorites

- Machine Translation
- Image Captioning (the decoder is similar!)
- DCGan and Pix2Pix

The docs are code

Tutorials on tf.org/alpha are

- Backed by a Jupyter Notebook
- Can be run directly in Colab

They automatically

- Install the right TensorFlow version
- Download a dataset
- Train a model
- Show you the result

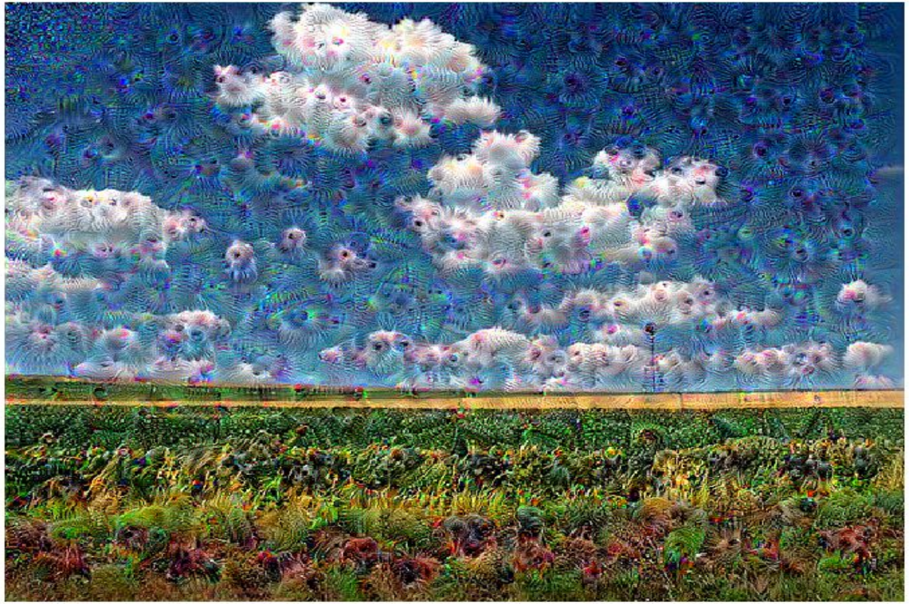
TensorFlow > Learn > TensorFlow Core > TF 2.0 Alpha

Image Captioning with Attention

Run in Google Colab | View source on GitHub

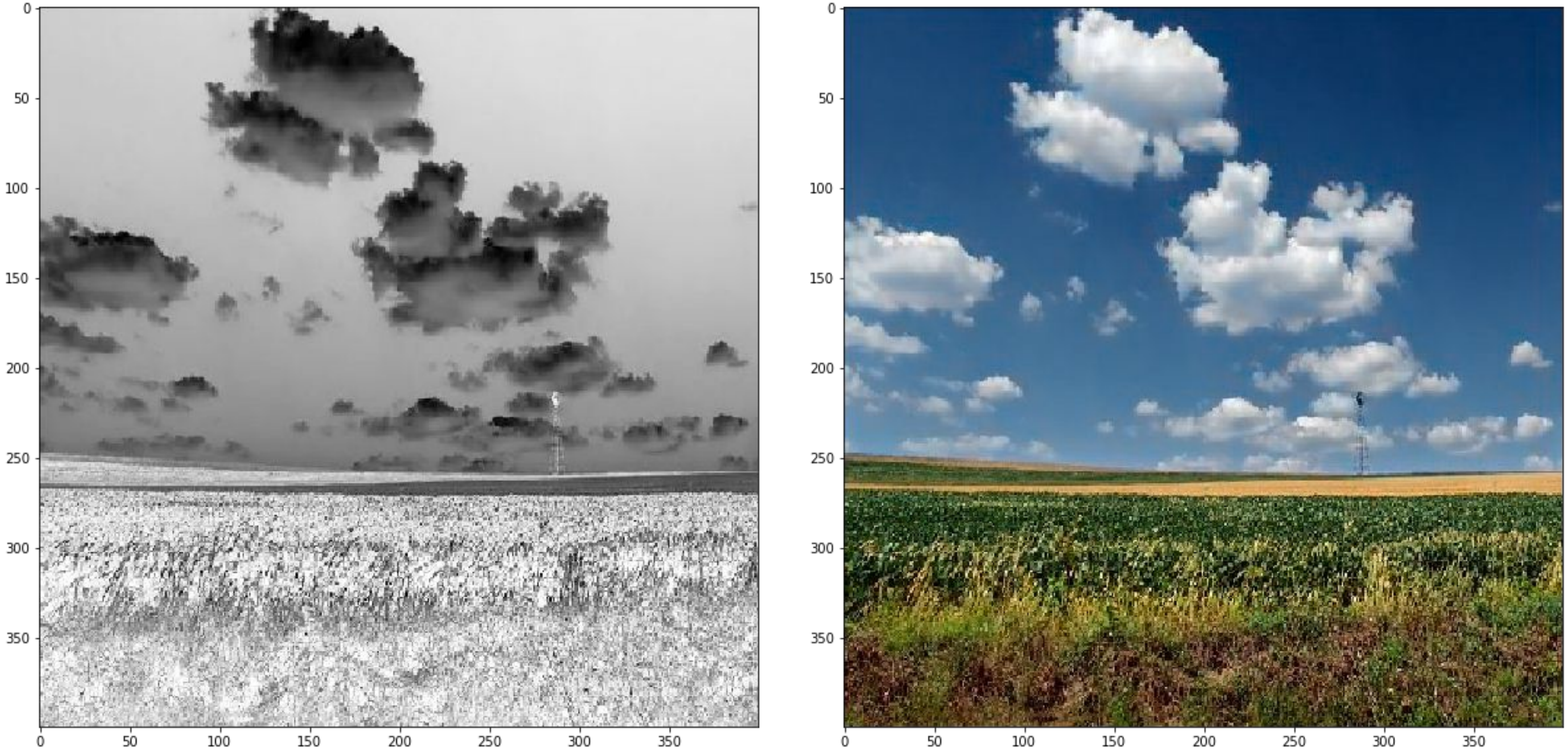
Given an image like the below, our goal is to generate a caption, such as "a surfer riding on a wave".

tensorflow.org/alpha/tutorials/text/image_captioning



https://github.com/random-forests/applied-dl/blob/master/examples/9-deep-dream-minimal.ipynb

Code walkthrough



https://github.com/random-forests/applied-dl/blob/master/examples/9-image-colorization.ipynb

Exercise 6

Deep Dream

Is anyone bilingual? Trilingual?

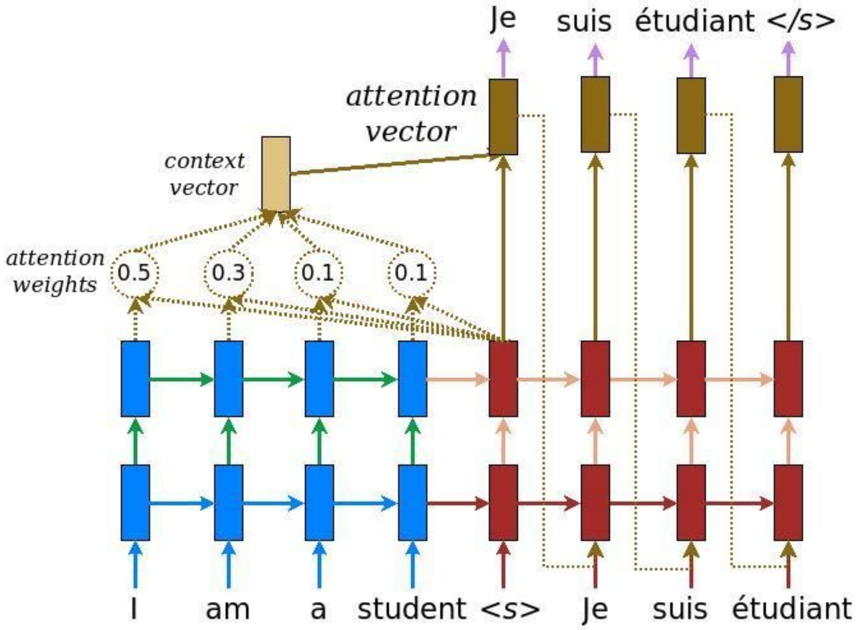
When translating, do you...

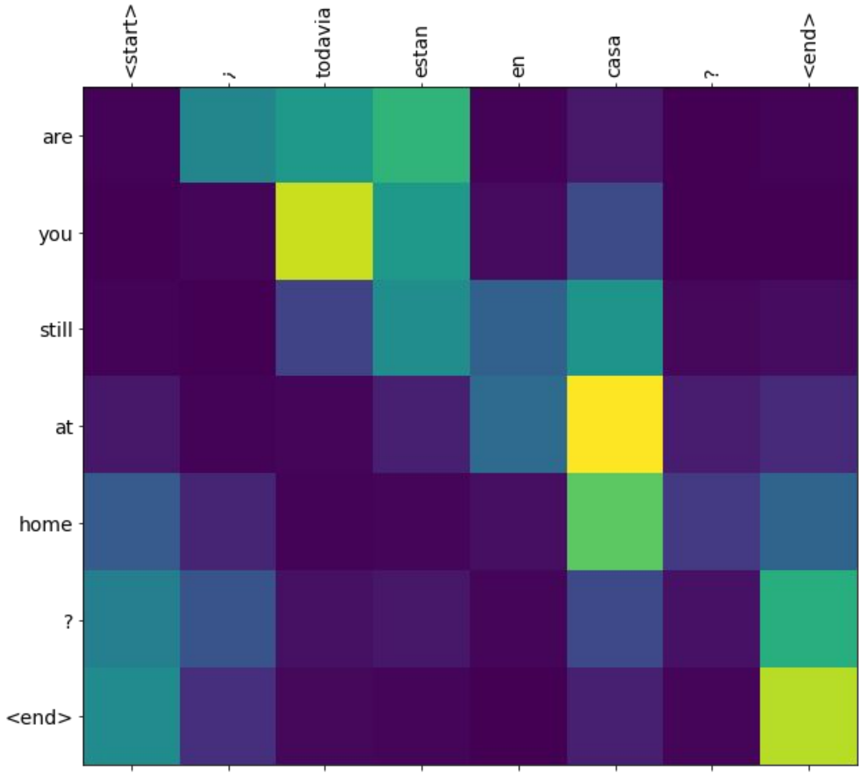
- Go directly from source -> target
- Or, go from source -> **intermediate representation** -> target.

Machine translation tutorials

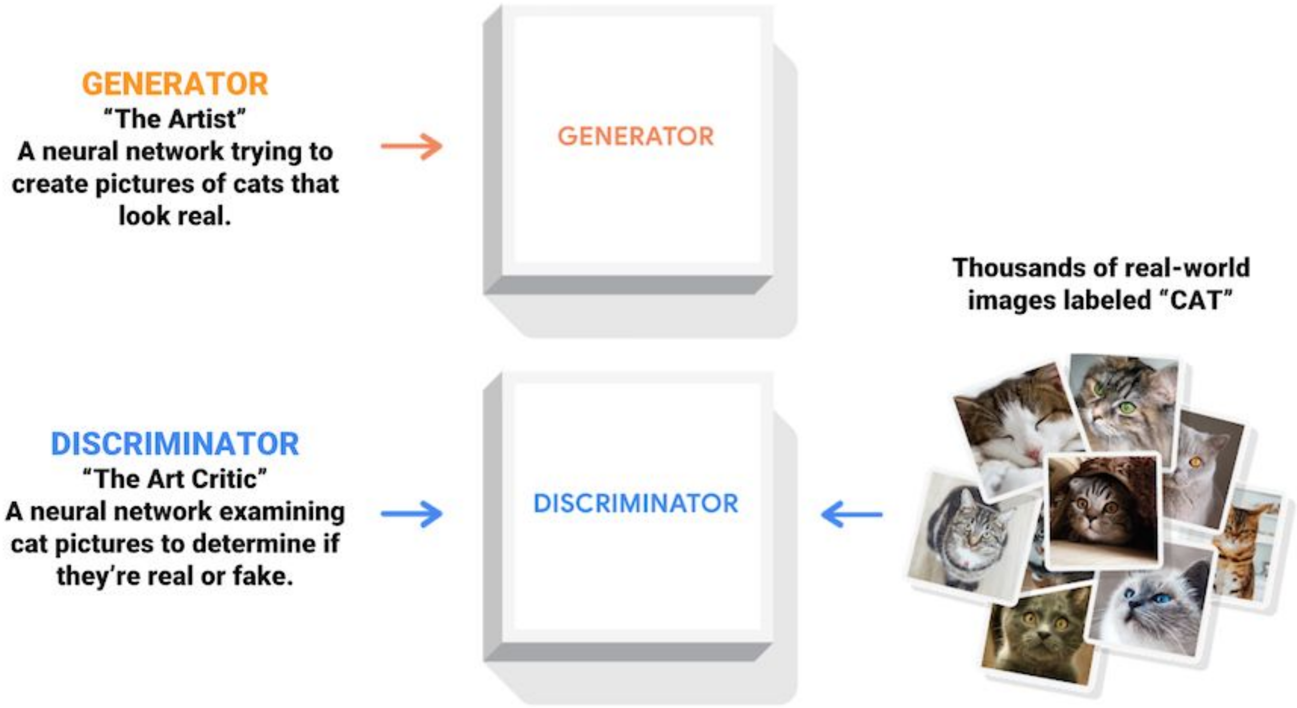
- Hello world (seq2seq), trains in about a minute.
- Neural Machine Translation with Attention
- Transformer

P.S., isn't 2019 cool? It's **amazing** this is possible.

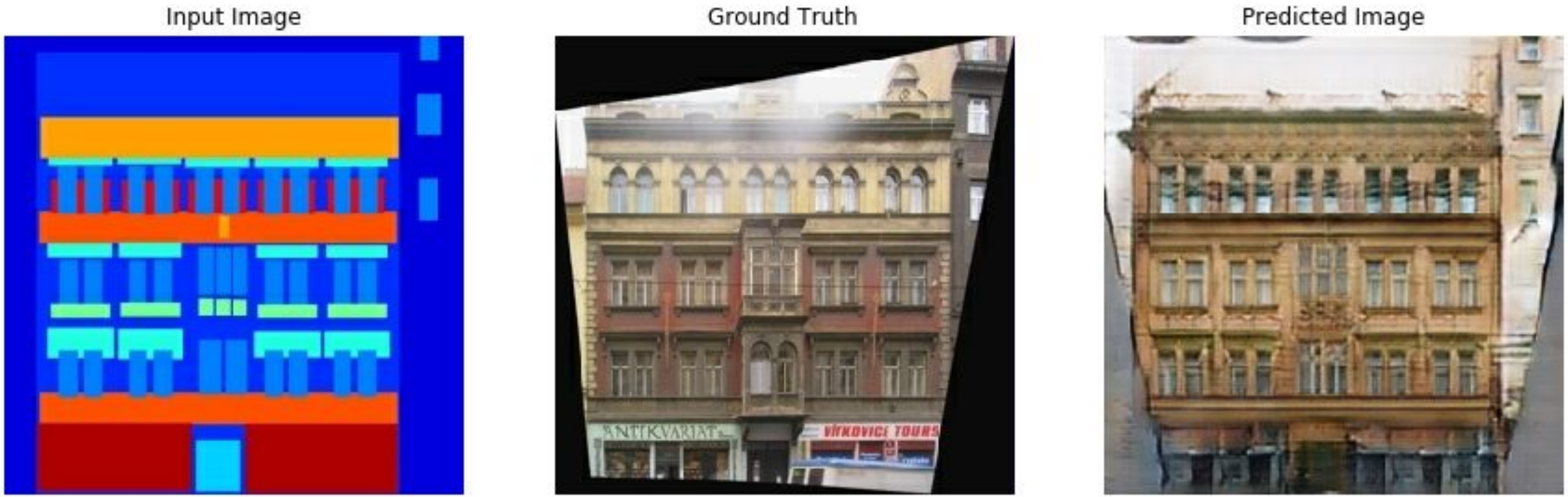




https://www.tensorflow.org/alpha/tutorials/sequences/nmt_with_attention



https://www.tensorflow.org/alpha/tutorials/generative/dcgan



https://www.tensorflow.org/alpha/tutorials/generative/pix2pix

Prediction Caption: the person is riding a surfboard in the ocean <end>

https://www.tensorflow.org/alpha/tutorials/sequences/image_captioning

Exercise 7

Seq2Seq

Exercise 6

Goals

- Use a pretrained CNN
- Extract intermediate activations
- Compute gradients w.r.t. an image

Visit

[bit.ly/tf-ws6](bit.ly/tf-ws6)

Exercise 7

Goals

- Train an English to Spanish model, just for fun
- Learn about encoder / decoders

Visit

bit.ly/minimal-nmt

Under the hood

Let's make this faster

```
lstm_cell = tf.keras.layers.LSTMCell(10)



def fn(input, state):
  return lstm_cell(input, state)

input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2
lstm_cell(input, state); fn(input, state) # warm up

# benchmark
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
```

Let’s make this faster

```
lstm_cell = tf.keras.layers.LSTMCell(10)

@tf.function
def fn(input, state):
  return lstm_cell(input, state)

input = tf.zeros([10, 10]); state = [tf.zeros([10, 10])] * 2
lstm_cell(input, state); fn(input, state) # warm up

# benchmark
timeit.timeit(lambda: lstm_cell(input, state), number=10) # 0.03
timeit.timeit(lambda: fn(input, state), number=10) # 0.004
```

AutoGraph makes this possible

```python
@tf.function
def f(x):
  while tf.reduce_sum(x) > 1:
    x = tf.tanh(x)
  return x

# you never need to run this (unless curious)
print(tf.autograph.to_code(f))
```

Generated code

```
def tf__f(x):
  def loop_test(x_1):
    with ag__.function_scope('loop_test'):
      return ag__.gt(tf.reduce_sum(x_1), 1)
  def loop_body(x_1):
    with ag__.function_scope('loop_body'):
      with ag__.utils.control_dependency_on_returns(tf.print(x_1)):
        tf_1, x = ag__.utils.alias_tensors(tf, x_1)
        x = tf_1.tanh(x)
        return x,
  x = ag__.while_stmt(loop_test, loop_body, (x,), (tf,))
  return x
```

Going big: tf.distribute.Strategy

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(64, input_shape=[10]),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Going big: Multi-GPU

```
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
  model = tf.keras.models.Sequential([
      tf.keras.layers.Dense(64, input_shape=[10]),
      tf.keras.layers.Dense(64, activation='relu'),
      tf.keras.layers.Dense(10, activation='softmax')])

  model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

What’s different between TF1 and TF2?

Removed

- session.run
- tf.control_dependencies
- tf.global_variables_initializer
- tf.cond, tf.while_loop

Added

- tf.function, AutoGraph

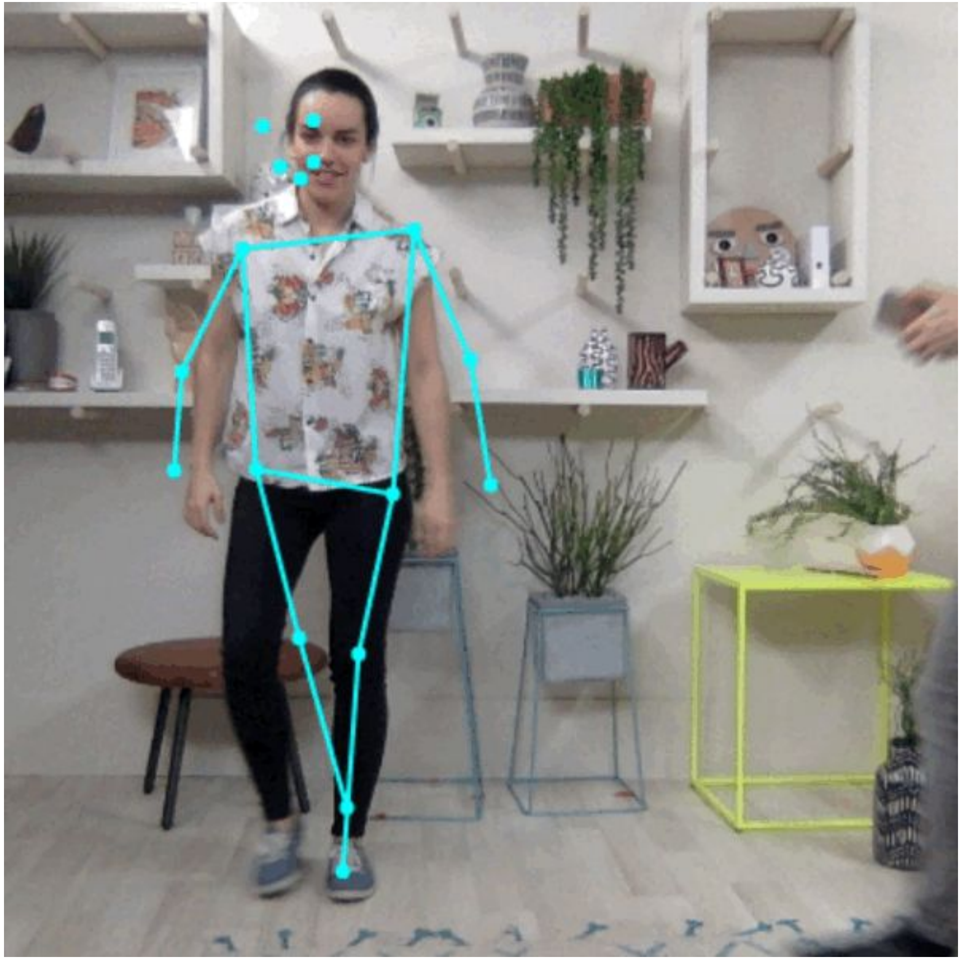
TensorFlow

TensorFlow.js

Demo #1

PoseNet

PoseNet



bit.ly/pose-net

Demo #2

BodyPix

BodyPix

bit.ly/body-pix

TensorFlow

Learning more

Learn more

Tutorials and guides

- tensorflow.org/alpha

Books

- Deep Learning with Python
- Hands-On Machine Learning with Scikit-Learn and TensorFlow (version 2.0 is almost ready)

Courses

- Intro to Deep Learning (MIT)
- Convolutional Neural Networks for Visual Recognition (Stanford)

tf.thanks!

Josh Gordon (twitter.com/random_forests)

TensorFlow

Extras

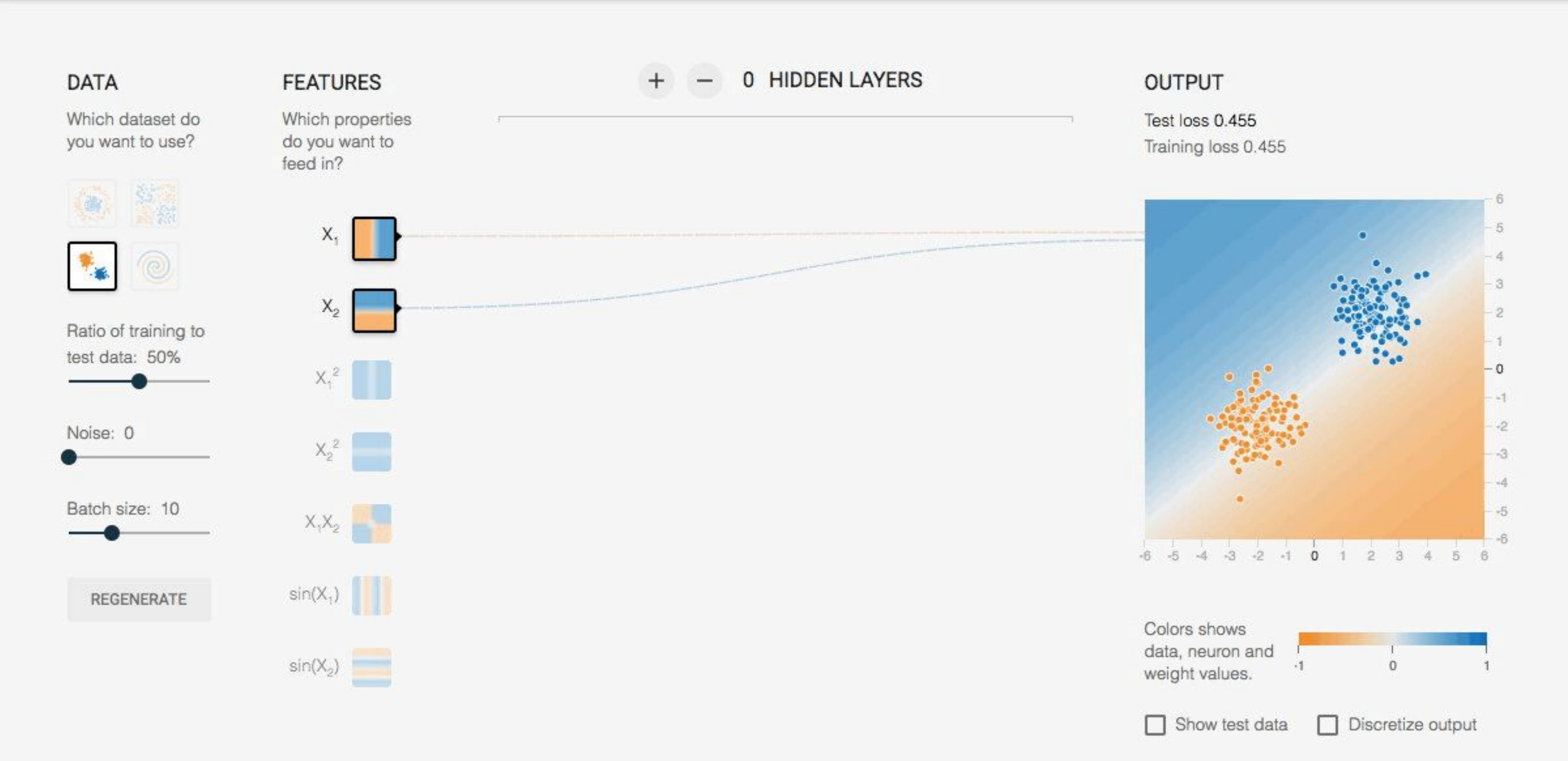
Reminders

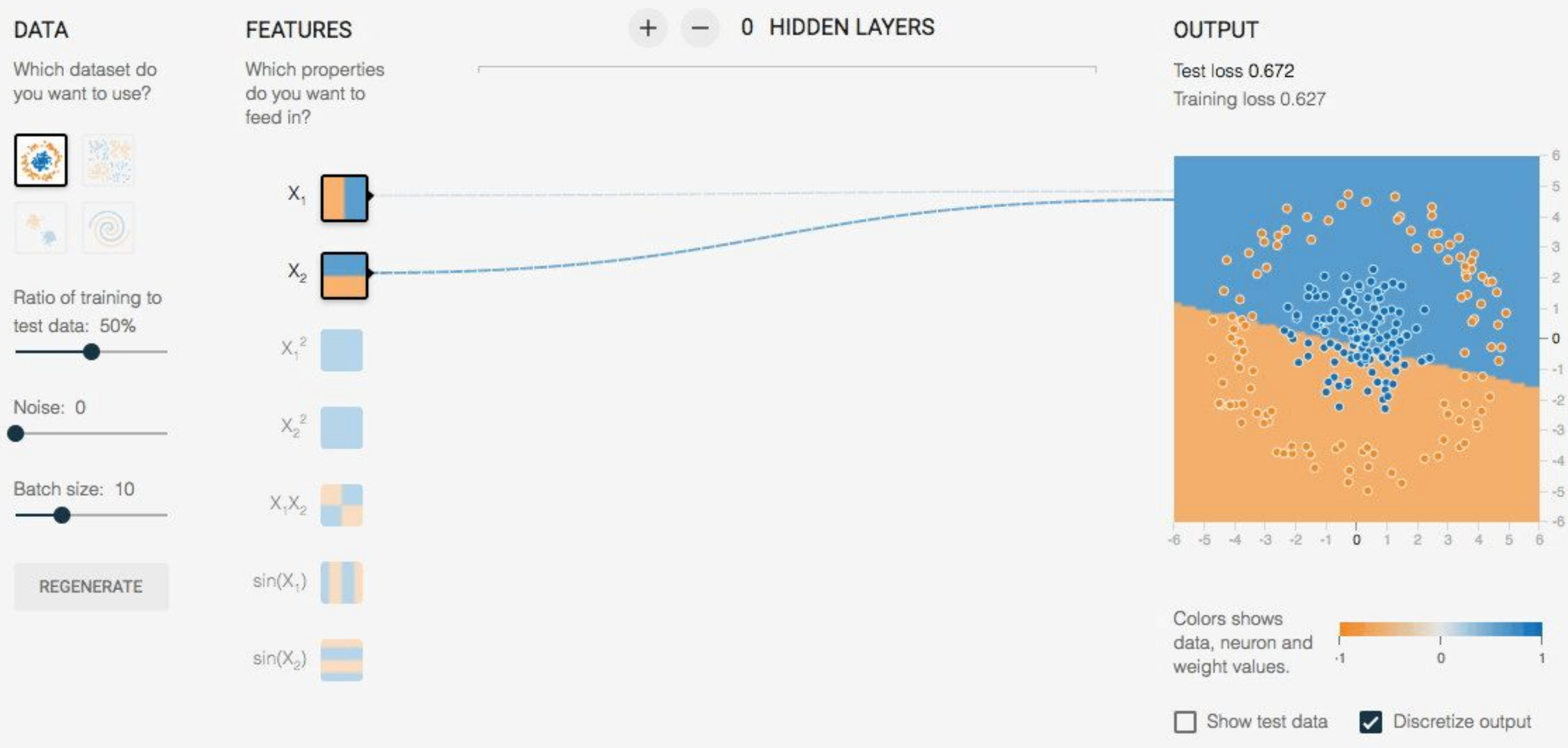
- Deep learning as compression
- Interlingual representations

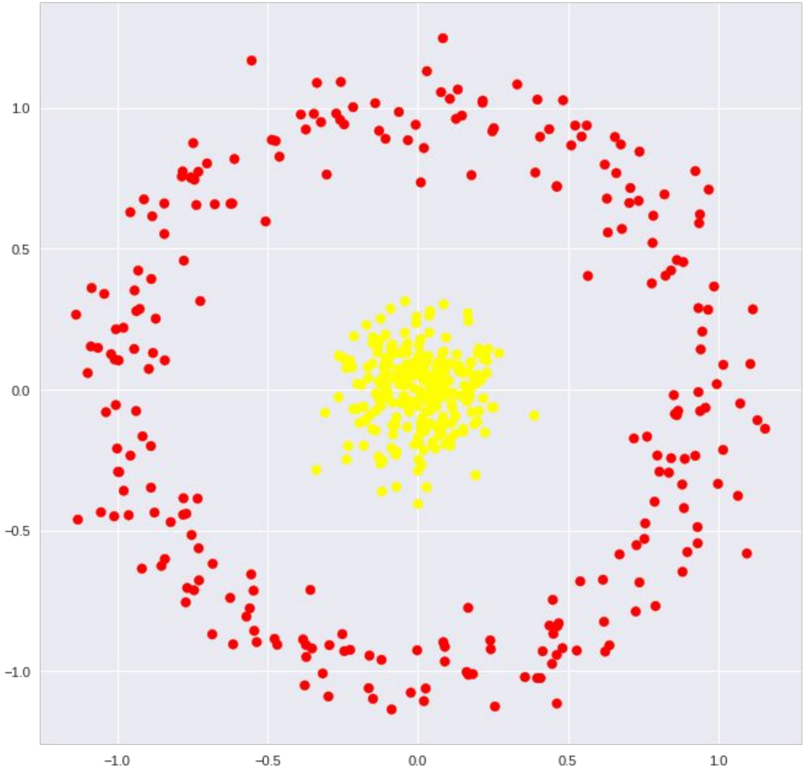
What's Deep Learning?

Representation learning

Automatic feature engineering


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
+ – 0 HIDDEN LAYERS
OUTPUT
Test loss 0.455
Training loss 0.455
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
+ − 0 HIDDEN LAYERS
OUTPUT
Test loss 0.672
Training loss 0.627
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output

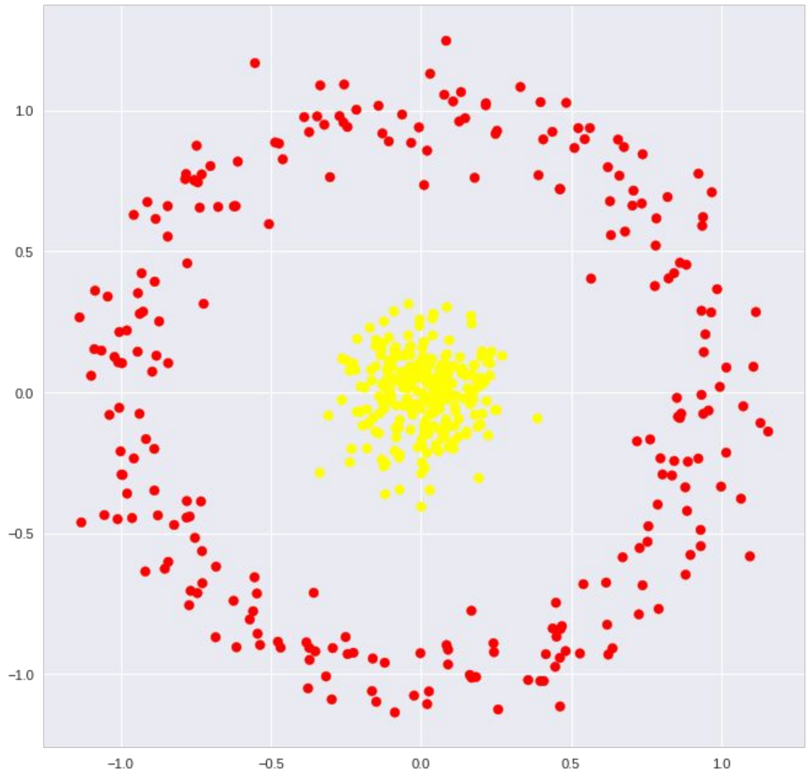


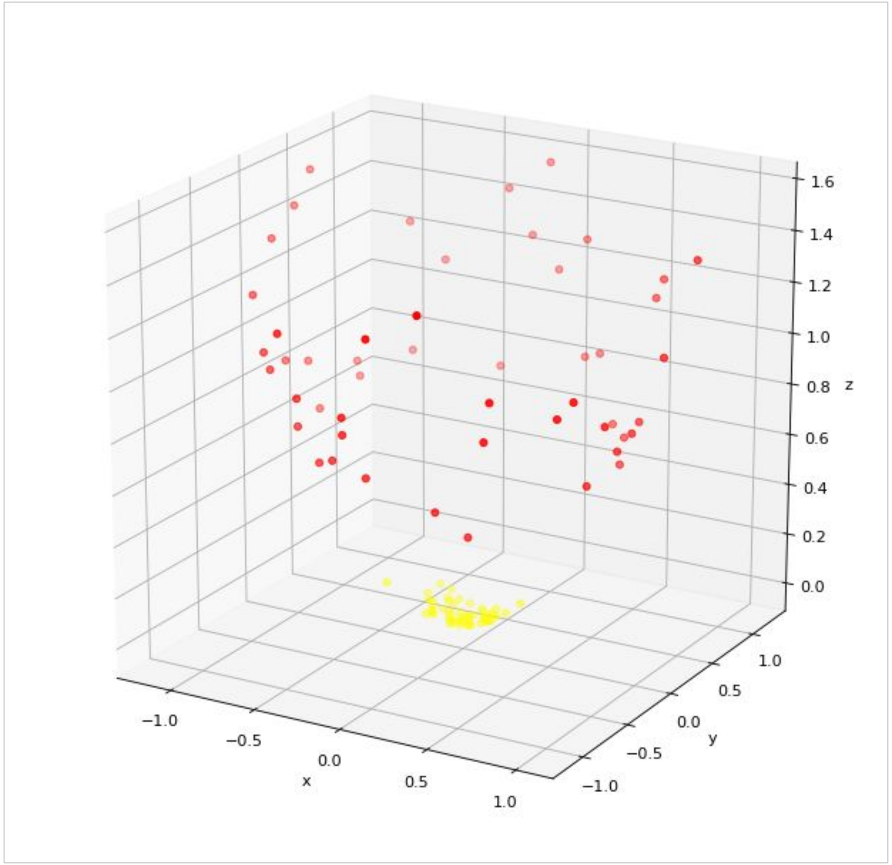
Feature engineering. What if we add a new feature:

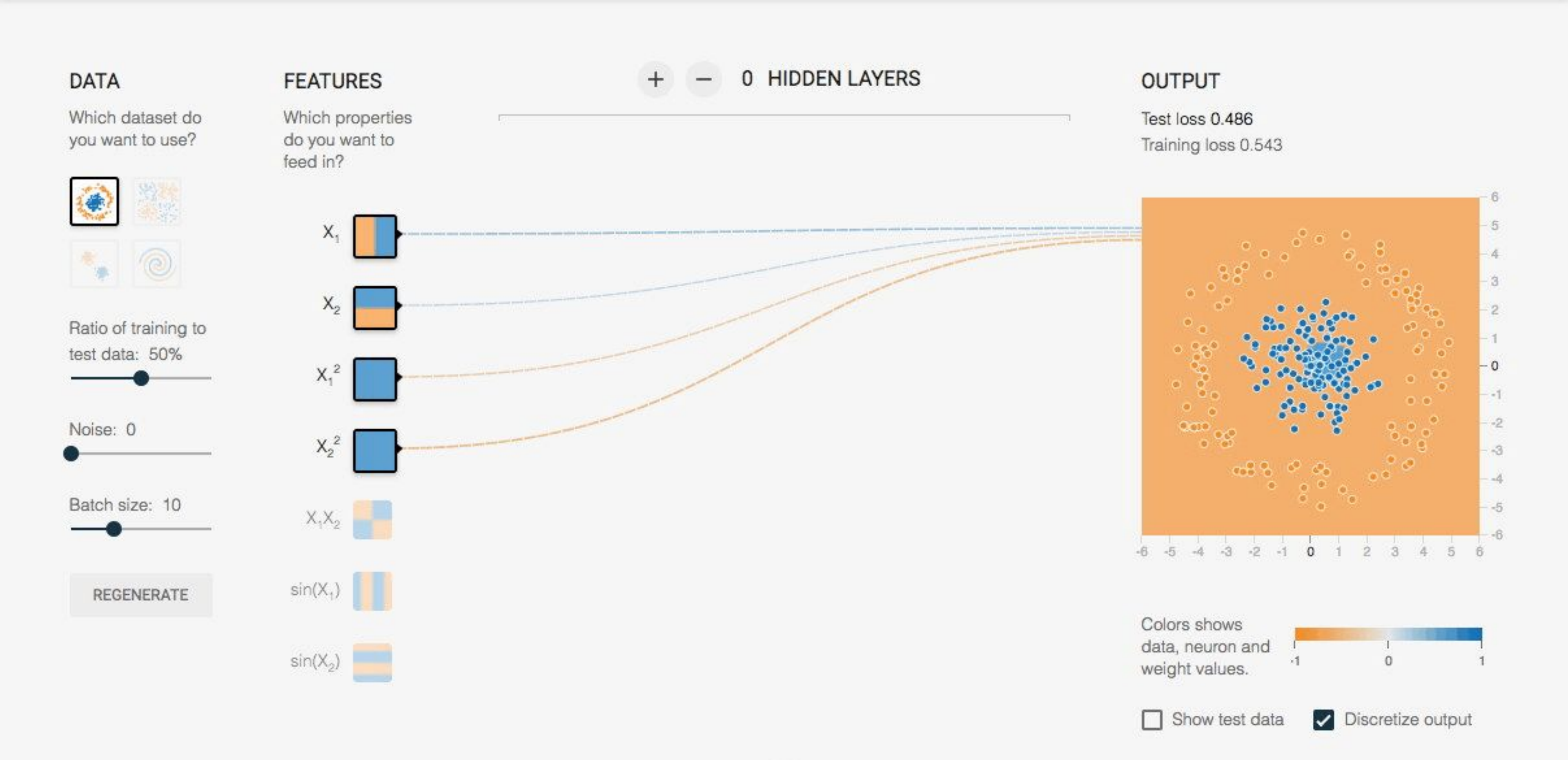
`z = x^2 + y^2.`

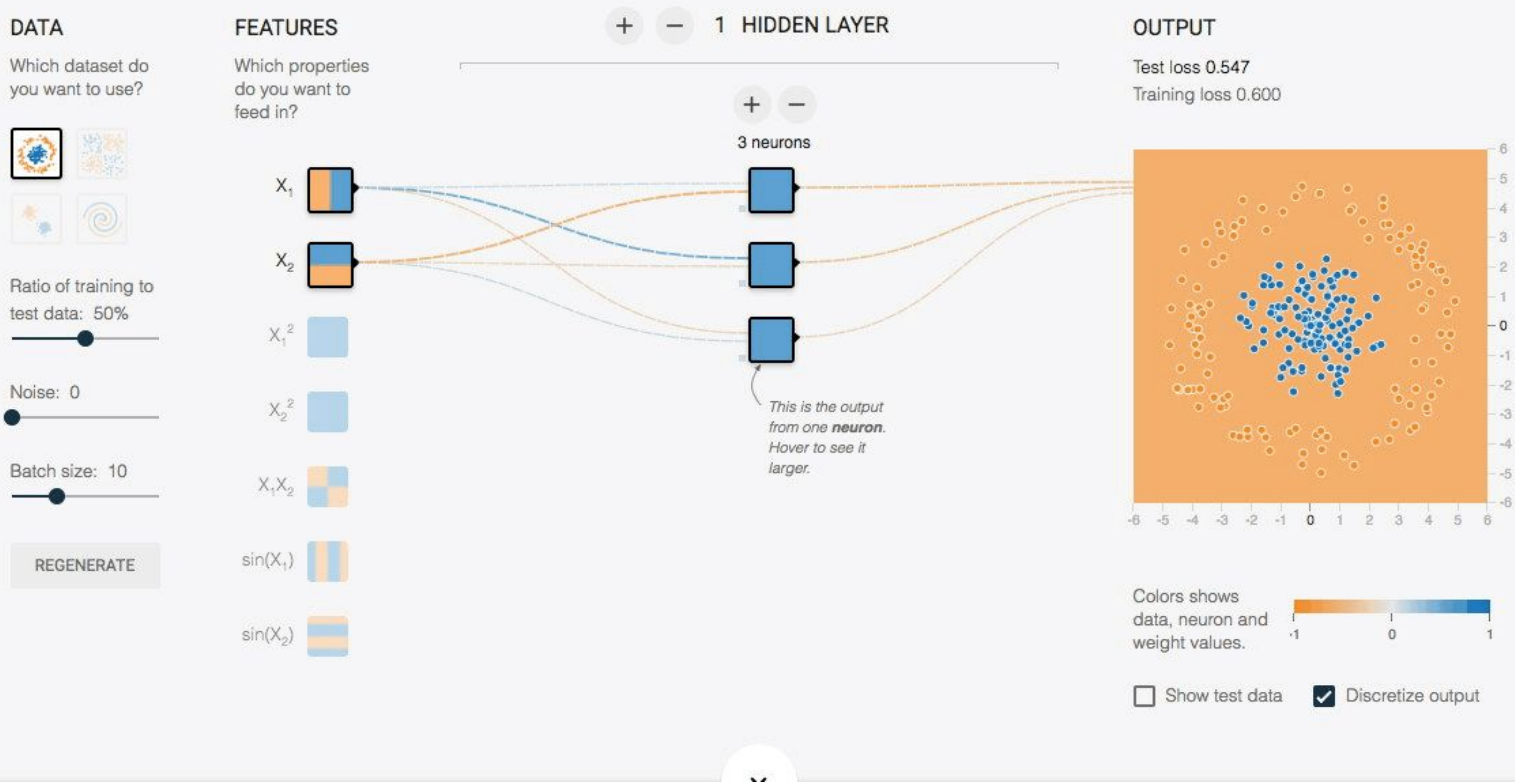
Intuition

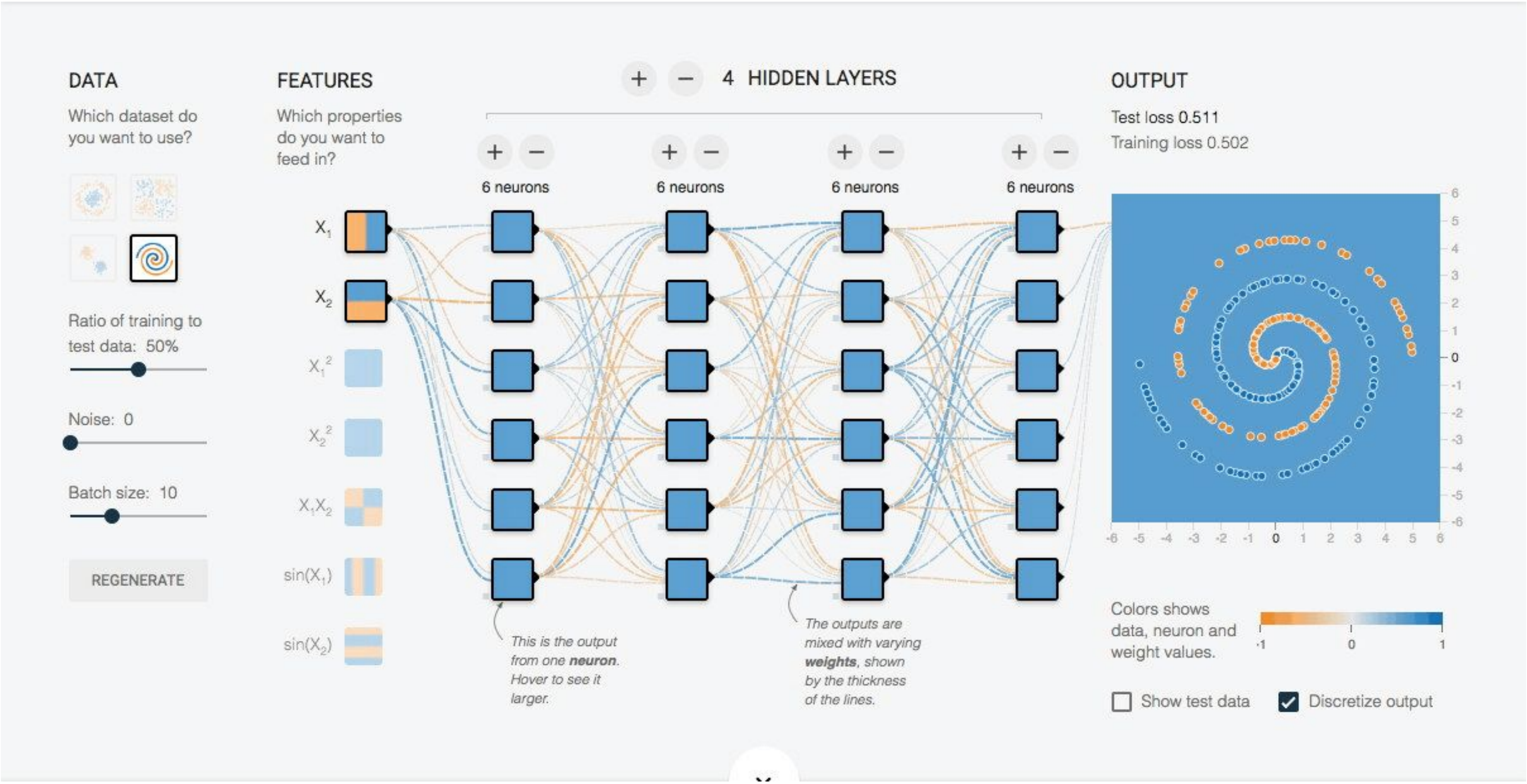
- All values of *z* will be positive.
- Yellow points are closer to the origin…
- So sum of their squared coords will be lower than red!


1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
z
−1.0
−0.5
0.0
0.5
1.0
x
1.0
0.5
0.0
−0.5
−1.0
y


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1²
X2²
X1X2
sin(X1)
sin(X2)
0 HIDDEN LAYERS
OUTPUT
Test loss 0.486
Training loss 0.543
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
+ − 1 HIDDEN LAYER
+ −
3 neurons
This is the output from one **neuron**. Hover to see it larger.
OUTPUT
Test loss 0.547
Training loss 0.600
Colors shows data, neuron and weight values.
-1 0 1
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
+ − 4 HIDDEN LAYERS
6 neurons
6 neurons
6 neurons
6 neurons
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
OUTPUT
Test loss 0.511
Training loss 0.502
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output