

# Neural Network Regression with



TensorFlow

# Where can you get help?

- Follow along with the code



Introduction to Regression with Neural Networks in TensorFlow Tutorial  
There are many definitions for a [regression problem](#) but in our case, we're going to simplify it to be: predicting a number.  
For example, you might want to:

- Predict the selling price of houses given information about them (such as number of rooms, size, number of bathrooms).
- Predict the coordinates of a bounding box of an item in an image.
- Predict the cost of medical insurance for an individual given their demographics (age, sex, gender, race).

In this notebook, we're going to set the foundations for how you can take a sample of inputs (this is your data), build a neural network to discover patterns in those inputs and then make a prediction (in the form of a number) based on those inputs.

What we're going to cover  
Specifically, we're going to go through doing the following with TensorFlow:

- Architecture of a regression model
- Input shapes and output shapes
  - x: features/data (inputs)
  - y: labels (outputs)

"If in doubt, run the code"

- Try it for yourself

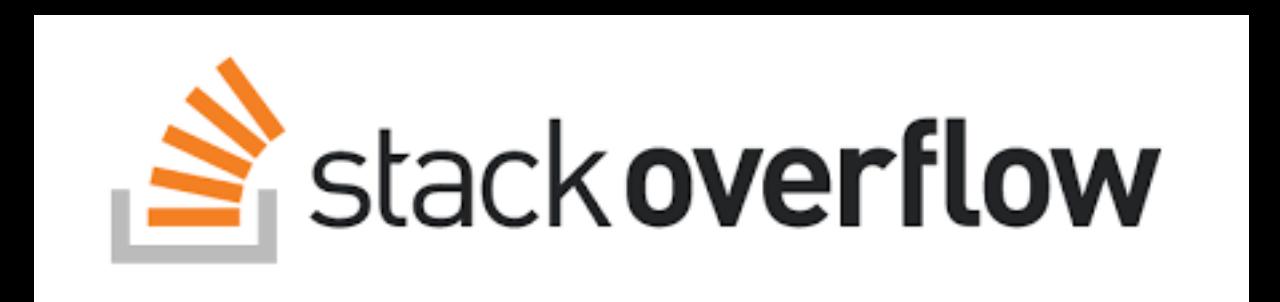


- Press SHIFT + CMD + SPACE to read the docstring

```
house_info = tf.constant(["bedroom", "bathroom", "garage"])
#tf_export('constant', v1=[])
def constant(value, dtype=None, shape=None,
            name='Const')
```

Creates a constant tensor from a tensor-like object.  
Note: All eager `Tensor` values are immutable (in contrast to `tfVariable`). There is nothing especially constant about the value returned from `tf.constant`. This function is not fundamentally different from `tf.convert_to_tensor`. The name `tf.constant` comes from the symbolic APIs (like `tf.data` or `keras` functional models) where the value is embedded in a `Const` node in the `tf.Graph`. `tf.constant` is useful for asserting that the value can be embedded that way.

- Search for it



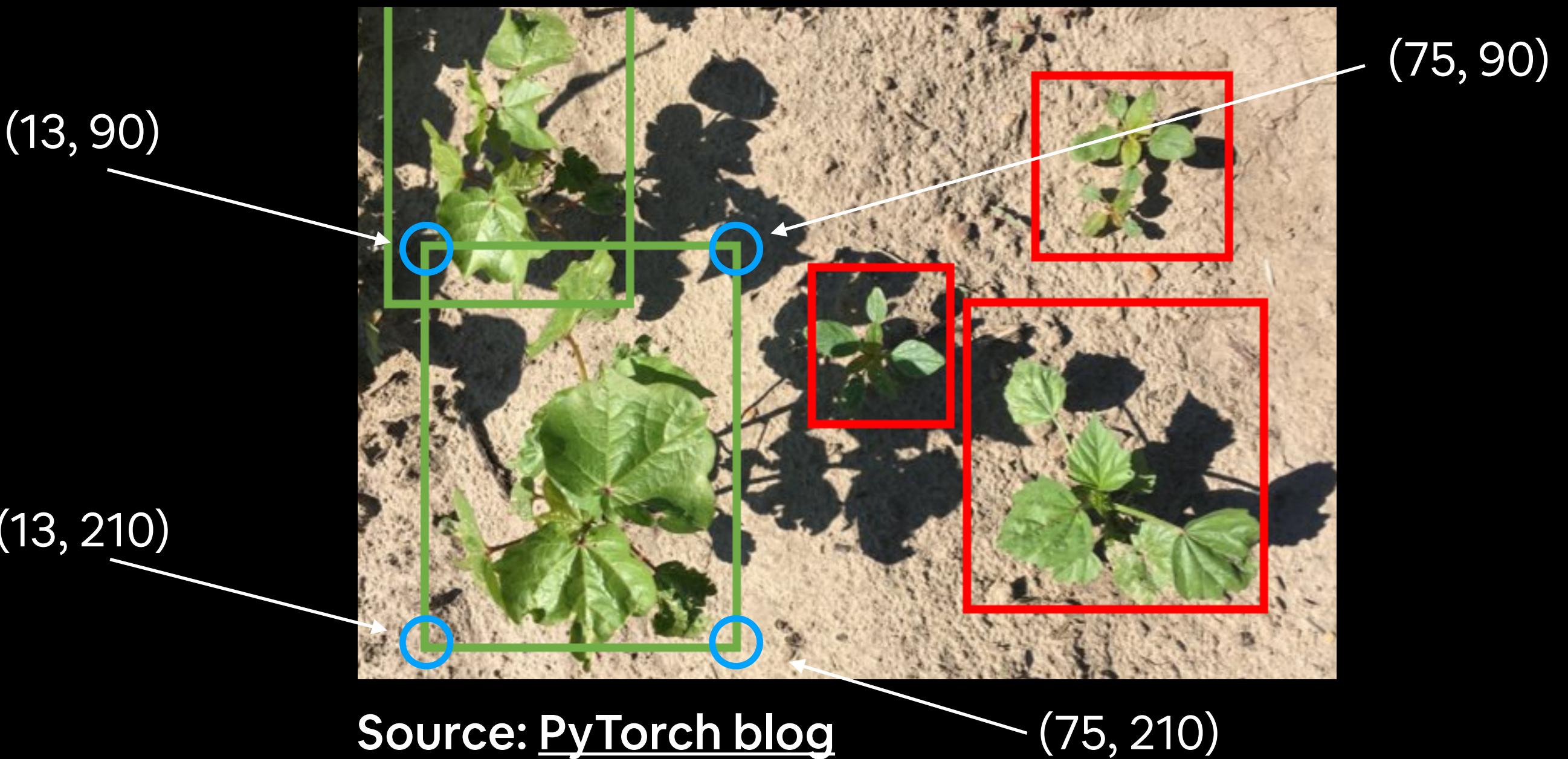
- Try again



- Ask (don't forget the Discord chat!)

“What is a regression  
problem?”

# Example regression problems



- “How much will this house sell for?”
- “How many people will buy this app?”
- “How much will my health insurance be?”
- “How much should I save each week for fuel?”



# What we're going to cover

(broadly)

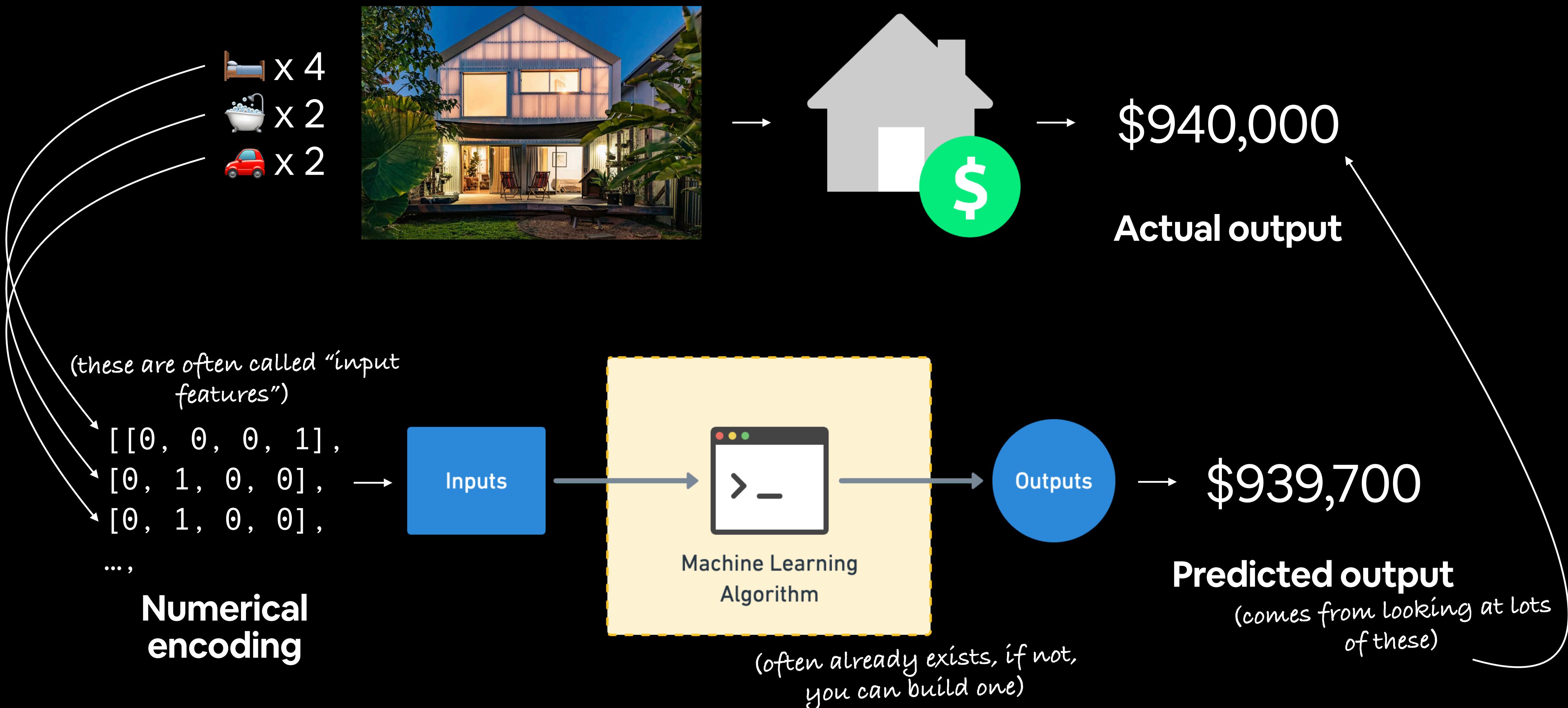
- Architecture of a neural network regression model
- Input shapes and output shapes of a **regression** model (features and labels)
- Creating custom data to view and fit
- Steps in modelling
  - Creating a model, compiling a model, fitting a model, evaluating a model
  - Different **regression** evaluation methods
  - Saving and loading models

(we'll be cooking up lots of code!)

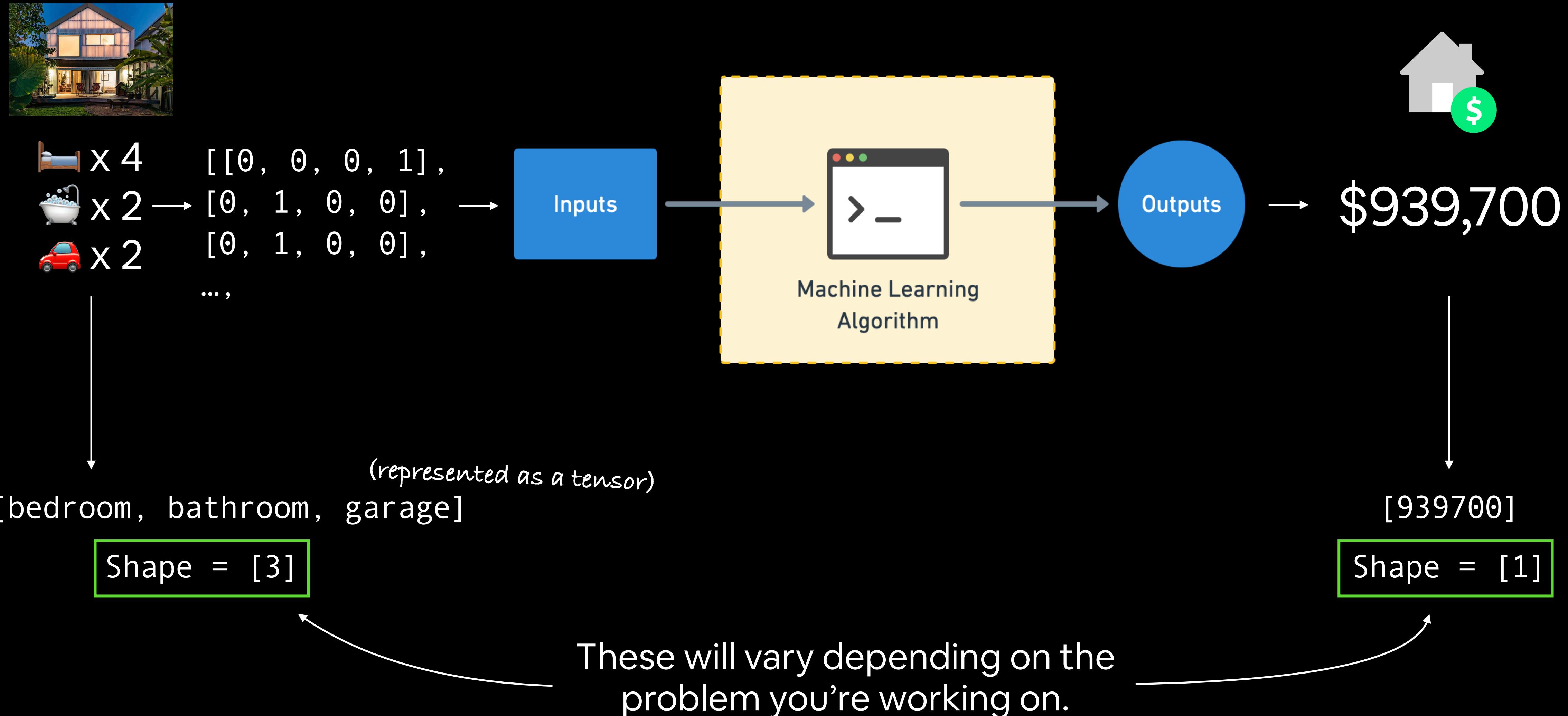
**How:**



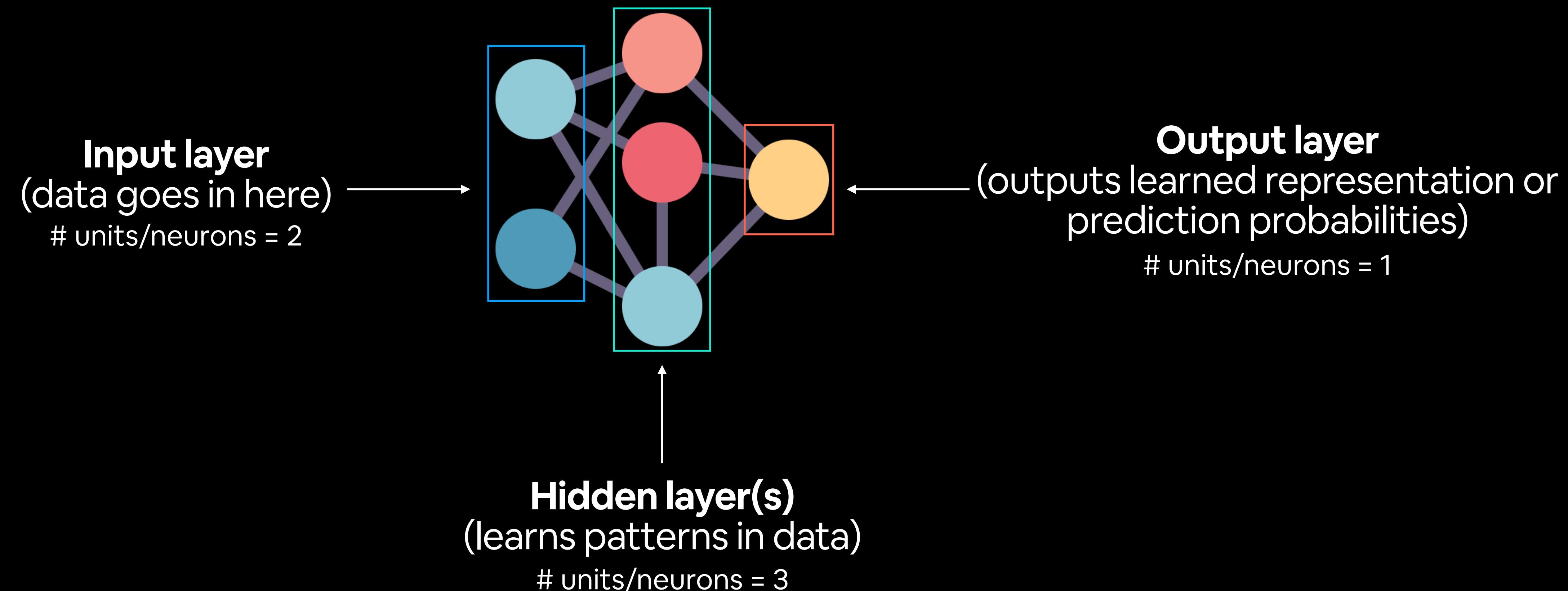
# Regression inputs and outputs



# Input and output shapes



# Anatomy of Neural Networks



Note: “patterns” is an arbitrary term, you’ll often hear “embedding”, “weights”, “feature representation”, “feature vectors” all referring to similar things.

(typical)

# Architecture of a regression model

Hyperparameter	Typical value
Input layer shape	Same shape as number of features (e.g. 3 for # bedrooms, # bathrooms, # car spaces in housing price prediction)
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited
Neurons per hidden layer	Problem specific, generally 10 to 100
Output layer shape	Same shape as desired prediction shape (e.g. 1 for house price)
Hidden activation	Usually <a href="#">ReLU</a> (rectified linear unit)
Output activation	None, ReLU, logistic/tanh
Loss function	<a href="#">MSE</a> (mean square error) or <a href="#">MAE</a> (mean absolute error)/Huber (combination of MAE/MSE) if outliers
Optimizer	<a href="#">SGD</a> (stochastic gradient descent), <a href="#">Adam</a>

← Also called  
“output neurons”

Source: Adapted from page 293 of [Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book](#) by Aurélien Géron



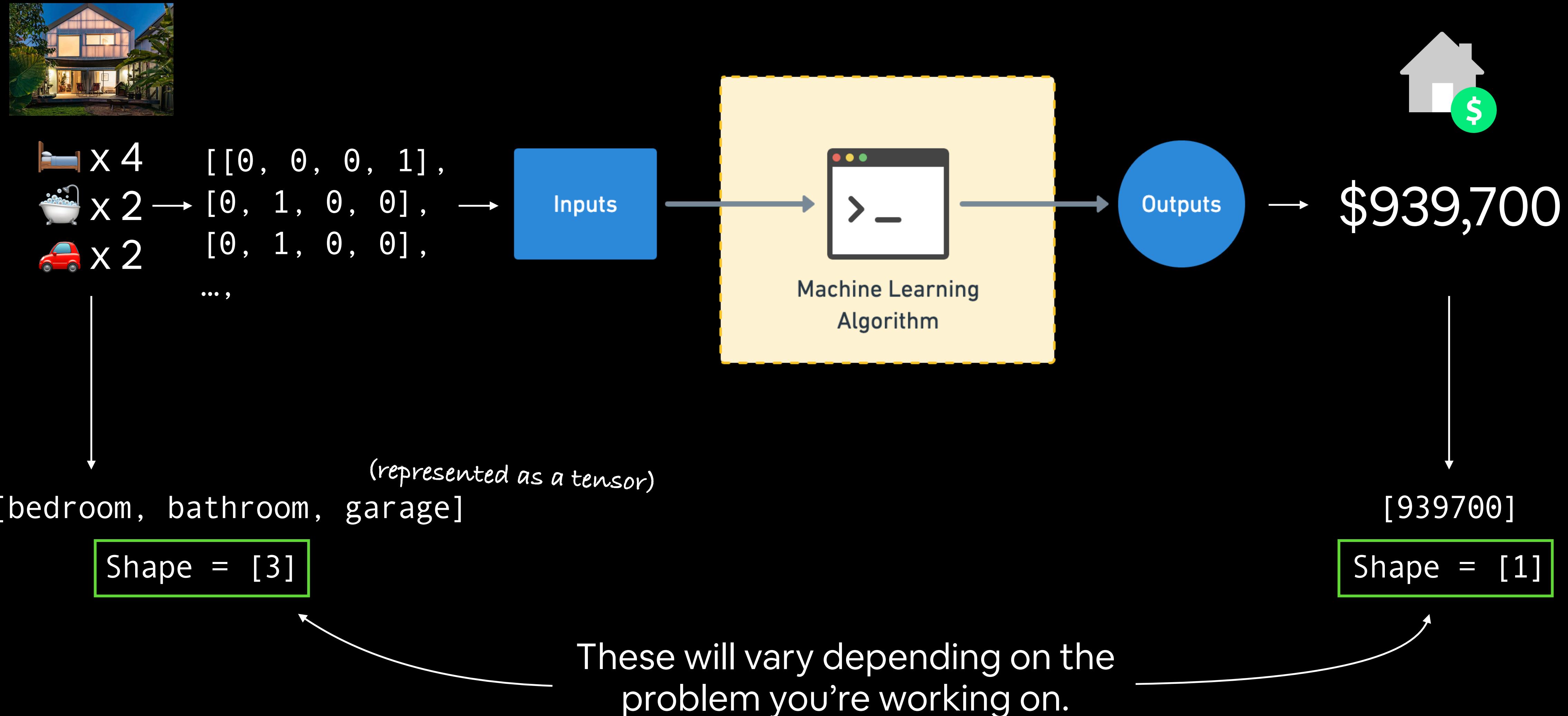
```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.Input(shape=(3,)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(1, activation=None)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["mae"])

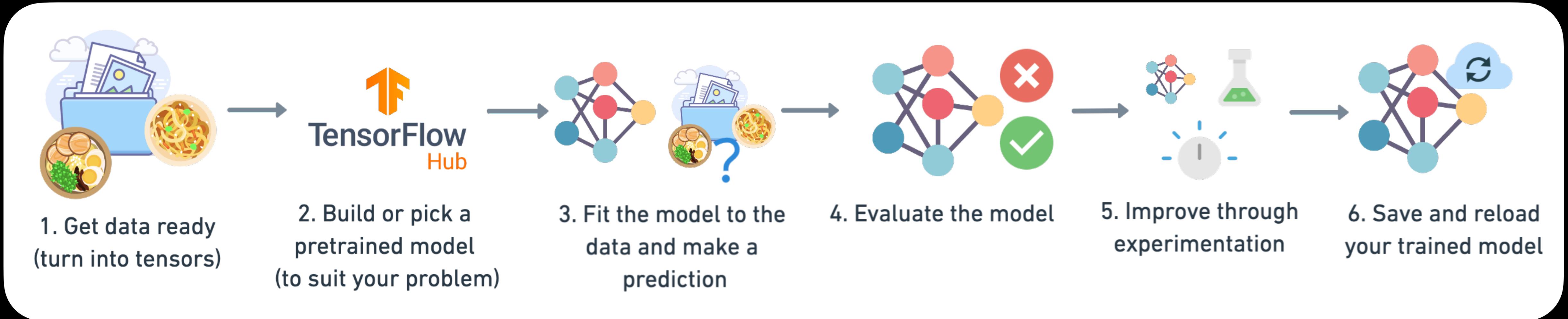
# 3. Fit the model
model.fit(X_train, y_train, epochs=100)
```

→ \$940,000

# Input and output shapes



# Steps in modelling with TensorFlow



```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
              optimizer=tf.keras.optimizers.SGD(),
              metrics=[ "mae" ])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

# Steps in modelling with TensorFlow

```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
               optimizer=tf.keras.optimizers.SGD(),
               metrics=[ "mae" ])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)
```

1. Construct or import a pretrained model relevant to your problem
2. Compile the model (prepare it to be used with data)
  - **Loss** — how wrong your model's predictions are compared to the truth labels (you want to minimise this).
  - **Optimizer** — how your model should update its internal patterns to better its predictions.
  - **Metrics** — human interpretable values for how well your model is doing.
3. Fit the model to the training data so it can discover patterns
  - **Epochs** — how many times the model will go through all of the training examples.
4. Evaluate the model on the test data (how reliable are our model's predictions?)

# Improving a model

(from a model's perspective)

```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
               optimizer=tf.keras.optimizers.SGD(),
               metrics=[ "mae" ])

# 3. Fit the model
model.fit(x_train_subset, y_train_subset, epochs=5)
```

Smaller model

```
# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(1)

])

# 2. Compile the model
model.compile(loss=tf.keras.losses.mae,
               optimizer=tf.keras.optimizers.Adam(lr=0.0001),
               metrics=[ "mae" ])

# 3. Fit the model
model.fit(x_train_full, y_train_full, epochs=100)
```

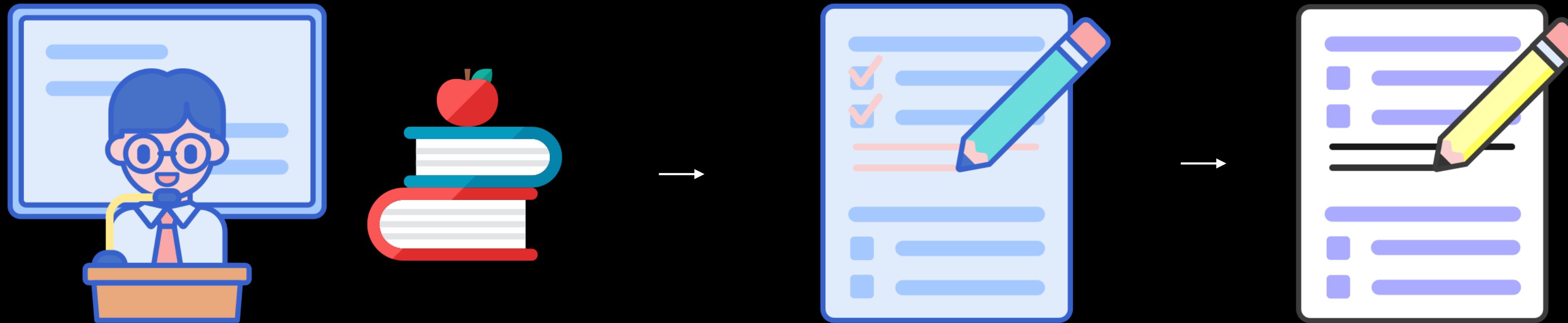
Larger model

## Common ways to improve a deep model:

- Adding layers
- Increase the number of hidden units
- Change the activation functions
- Change the optimization function
- Change the learning rate      (because you can alter each of these, they're hyperparameters)
- Fitting on more data
- Fitting for longer

# Three datasets

(possibly the most important  
concept in machine learning...)



**Course materials  
(training set)**

**Practice exam  
(validation set)**

**Final exam  
(test set)**

**Generalization**

The ability for a machine learning model to perform well on data it hasn't seen before.

(some common)

# Regression evaluation metrics

Metric Name	Metric Forumla	TensorFlow code	When to use
Mean absolute error (MAE)	$\text{MAE} = \frac{\sum_{i=1}^n  y_i - x_i }{n}$	<code>tf.keras.losses.MAE()</code> or <code>tf.metrics.mean_absolute_error()</code>	As a great starter metric for any regression problem.
Mean square error (MSE)	$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$	<code>tf.keras.losses.MSE()</code> or <code>tf.metrics.mean_squared_error()</code>	When larger errors are more significant than smaller errors.
Huber	$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for }  y - f(x)  \leq \delta, \\ \delta  y - f(x)  - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$	<code>tf.keras.losses.Huber()</code>	Combination of MSE and MAE. Less sensitive to outliers than MSE.

# The machine learning explorer's motto

“Visualize, visualize, visualize”



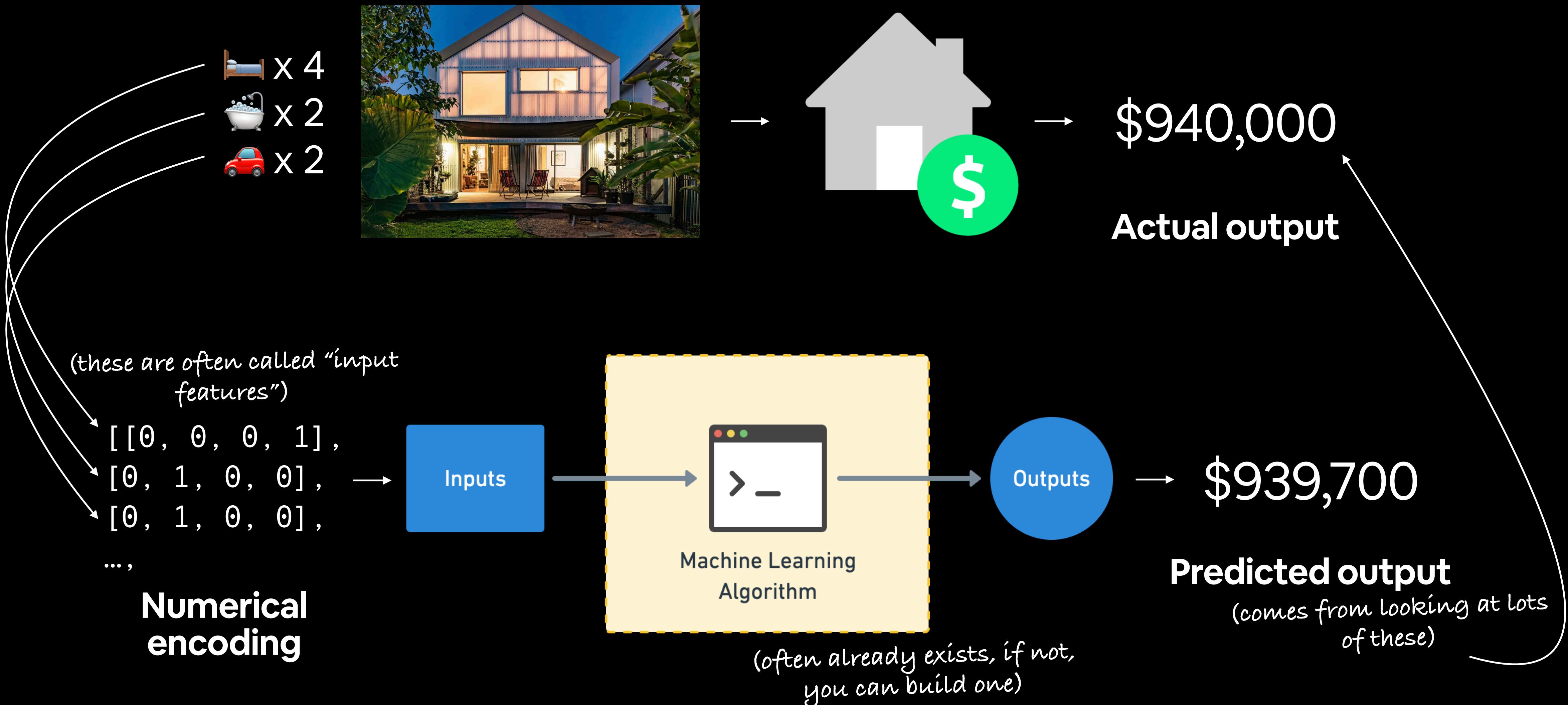
# The machine learning practitioner's motto

“Experiment, experiment, experiment”

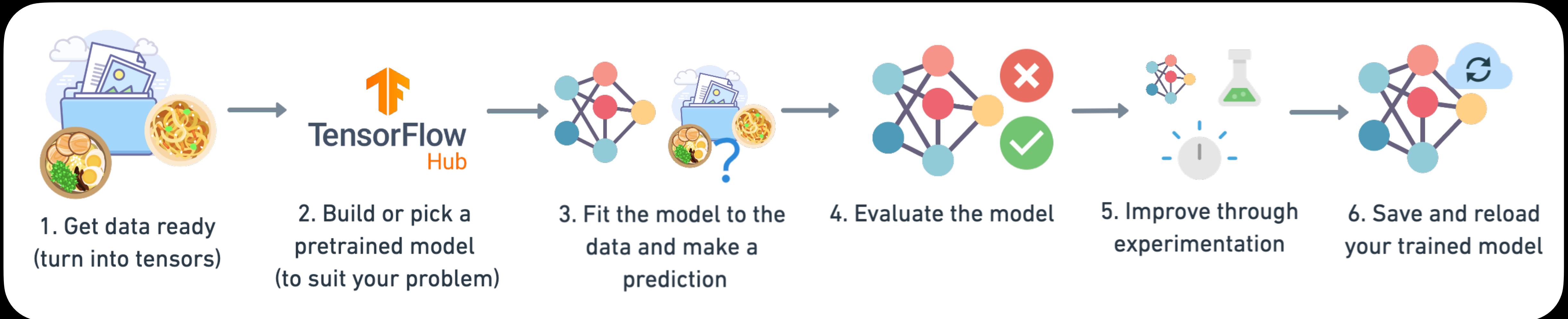


*(try lots of things and see what  
tastes good)*

# Regression inputs and outputs



# Steps in modelling with TensorFlow



- ↓
1. Turn all data into numbers (neural networks can't handle strings)
  2. Make sure all of your tensors are the right shape
  3. Scale features (normalize or standardize, neural networks tend to prefer normalization)

# Feature scaling

Scaling type	What it does	Scikit-Learn Function	When to use
Scale (also referred to as normalisation)	Converts all values to between 0 and 1 whilst preserving the original distribution.	MinMaxScaler	Use as default scaler with neural networks.
Standardization	Removes the mean and divides each value by the standard deviation.	StandardScaler	Transform a feature to have close to normal distribution (caution: this reduces the effect of outliers).

**Source:** Adapted from Jeff Hale's [Scale, Standardize, or Normalize with Scikit-Learn](#) article.