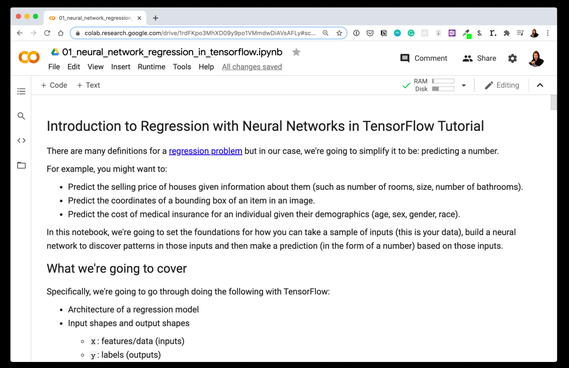
Neural Network Regression with

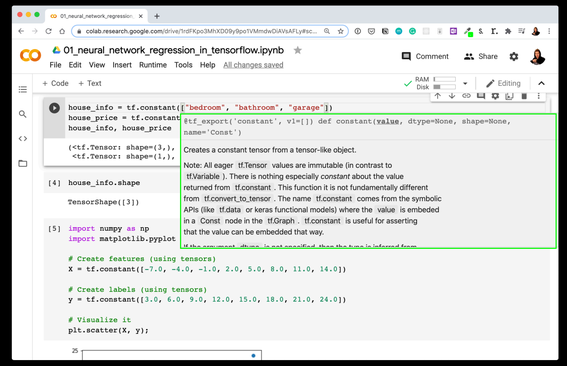
TensorFlow

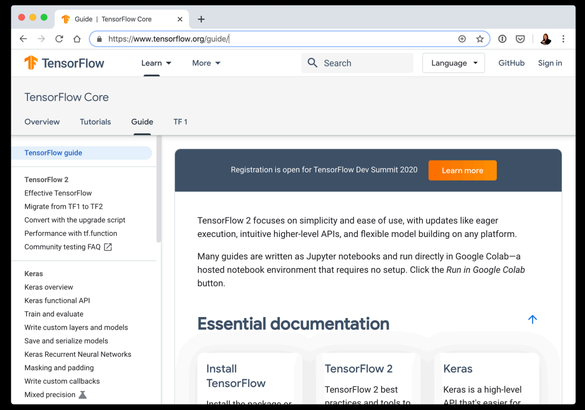
Where can you get help?

- Follow along with the code
- Try it for yourself
- Press SHIFT + CMD + SPACE to read the docstring
- Search for it
- Try again
- Ask (don't forget the Discord chat!)



"If in doubt, run the code"



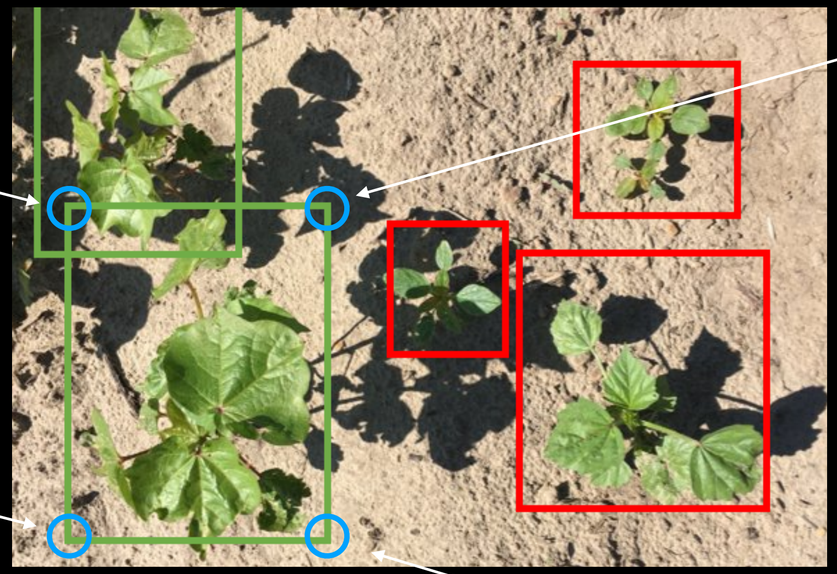


regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome' or 'response' variable) and one or more independent variables (often called 'predictors' or 'features').

“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?”



Source: PyTorch blog

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

🛏️ x 4
🛁 x 2
🚗 x 2

$940,000

Actual output

(these are often called "input features")

```
[[0, 0, 0, 1],
[0, 1, 0, 0],
[0, 1, 0, 0],
...,
```

Numerical encoding

Inputs

Machine Learning Algorithm

(often already exists, if not, you can build one)

Outputs

$939,700

Predicted output

(comes from looking at lots of these)

Input and output shapes

🛏️ x 4
🛁 x 2
🚗 x 2

```
[[0, 0, 0, 1],
[0, 1, 0, 0],
[0, 1, 0, 0],
…,
```

Inputs → Machine Learning Algorithm → Outputs → $939,700

(represented as a tensor)

`[bedroom, bathroom, garage]`

`Shape = [3]`

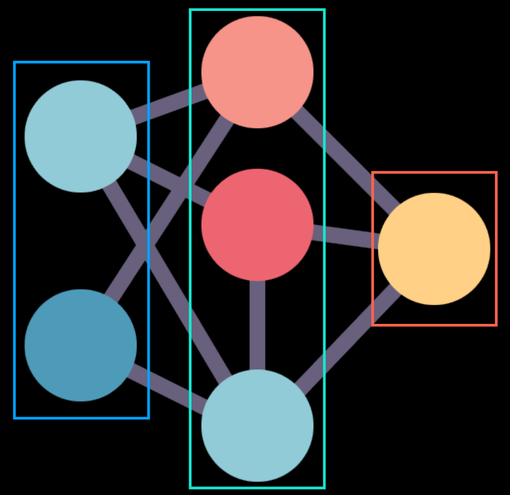
`[939700]`

`Shape = [1]`

These will vary depending on the problem you're working on.

Anatomy of Neural Networks

Input layer
(data goes in here)
units/neurons = 2



Output layer
(outputs learned representation or prediction probabilities)
units/neurons = 1

Hidden layer(s)
(learns patterns in data)
units/neurons = 3

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	See according to the colours	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 ReLU (rectified linear unit)
Output activation		None, ReLU, logistic/tanh
Loss function		MSE (mean square error) or MAE (mean absolute error)/Huber (combination of MAE/MSE) if outliers
Optimizer		SGD (stochastic gradient descent), Adam

← Also called "output neurons"

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

🛏 x 4
🛁 x 2
🚗 x 2

→

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

# 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

🛏️ x 4
🛁 x 2
🚗 x 2

```
[[0, 0, 0, 1],
[0, 1, 0, 0],
[0, 1, 0, 0],
…,
```

Inputs → Machine Learning Algorithm → Outputs → $939,700

(represented as a tensor)

`[bedroom, bathroom, garage]`

`Shape = [3]`

`[939700]`

`Shape = [1]`

These will vary depending on the problem you're working on.

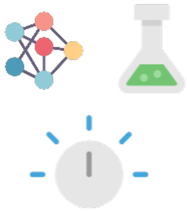
Steps in modelling with TensorFlow

1. Get data ready (turn into tensors)

2. Build or pick a pretrained model (to suit your problem)

3. Fit the model to the data and make a prediction

4. Evaluate the model



5. Improve through experimentation

6. Save and reload your trained model

```python
# 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
- Fitting on more data
- Fitting for longer

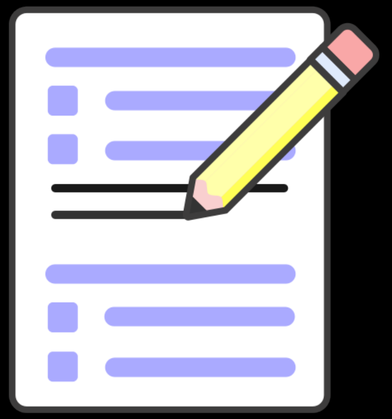
(because you can alter each of these, they're hyperparameters)

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)	difference between prediction and label
	$\text{MAE} = \frac{\sum_{i=1}^{n} \lvert y_i - x_i \rvert}{n}$
		`tf.keras.losses.MAE( )`
		or
		`tf.metrics.mean_absolute_error( )`
			As a great starter metric for any regression problem.

Mean square error (MSE)	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$	`tf.keras.losses.MSE( )`
		or
		`tf.metrics.mean_square_error( )`
			When larger errors are more significant than smaller errors.

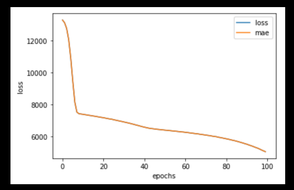
Huber	$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \textbf{for } \lvert y - f(x) \rvert \le \delta, \\ \delta \lvert y - f(x) \rvert - \frac{1}{2}\delta^2 & \textbf{otherwise.} \end{cases}$	`tf.keras.losses.Huber( )`	Combination of MSE and MAE. Less sensitive to outliers than MSE.

The machine learning explorer's motto

"Visualize, visualize, visualize"

Data

Model



Training

Predictions

It's a good idea to visualize these as often as possible.

The machine learning practitioner's motto

"Experiment, experiment, experiment"


(try lots of things and see what tastes good)

Regression inputs and outputs

🛏️ x 4
🛁 x 2
🚗 x 2

$940,000

Actual output

(these are often called "input features")

```
[[0, 0, 0, 1],
[0, 1, 0, 0],
[0, 1, 0, 0],
...,
```

Numerical encoding

Inputs

Machine Learning Algorithm

(often already exists, if not, you can build one)

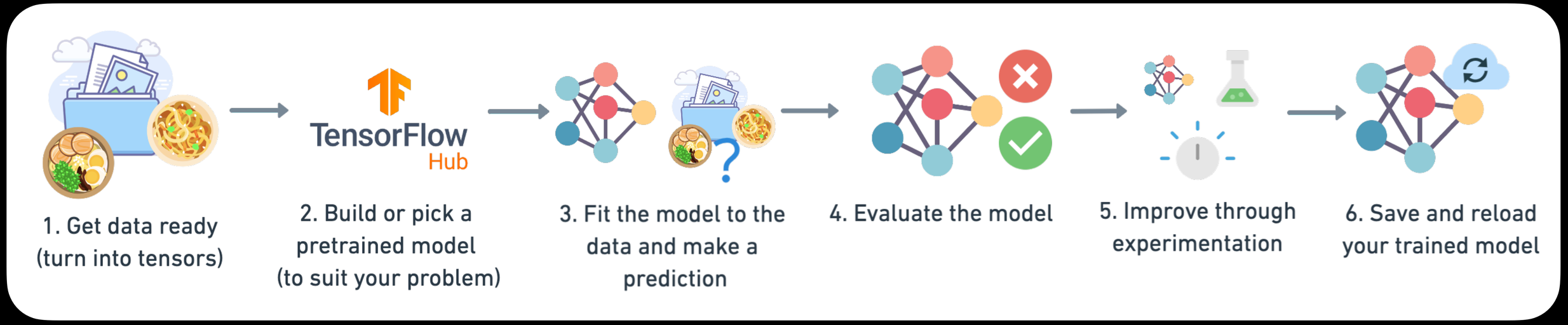
Outputs

$939,700

Predicted output

(comes from looking at lots of these)

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.