Deep Learning with Keras

ACTL3143 & ACTL5111 Deep Learning for Actuaries

Patrick Laub

Lecture Outline

- **California House Price Prediction**
- EDA & Baseline Model
- Our First Neural Network
- Force positive predictions
- Preprocessing
- Early Stopping

Data science always starts with the data!

The target variable is the median house value for California districts, expressed in $100,000's. This dataset was derived from the 1990 U.S. census, using one row per census block group. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people).

Dall-E's rendition of the this dataset.

Source: Scikit-learn documentation.

UNSW SYDNEY

Columns

- `MedInc` median income in block group
- `HouseAge` median house age in block group
- `AveRooms` average number of rooms per household
- `AveBedrms` average # of bedrooms per household
- `Population` block group population
- `AveOccup` average number of household members
- `Latitude` block group latitude
- `Longitude` block group longitude
- `MedHouseVal` median house value (**target**)

Source: Scikit-learn documentation.

Import the data

```
from sklearn.datasets import fetch_california_housing

features, target = fetch_california_housing(
    as_frame=True, return_X_y=True)
features
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Lon
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122
...	...	...	...	...	...	...	...	...
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.

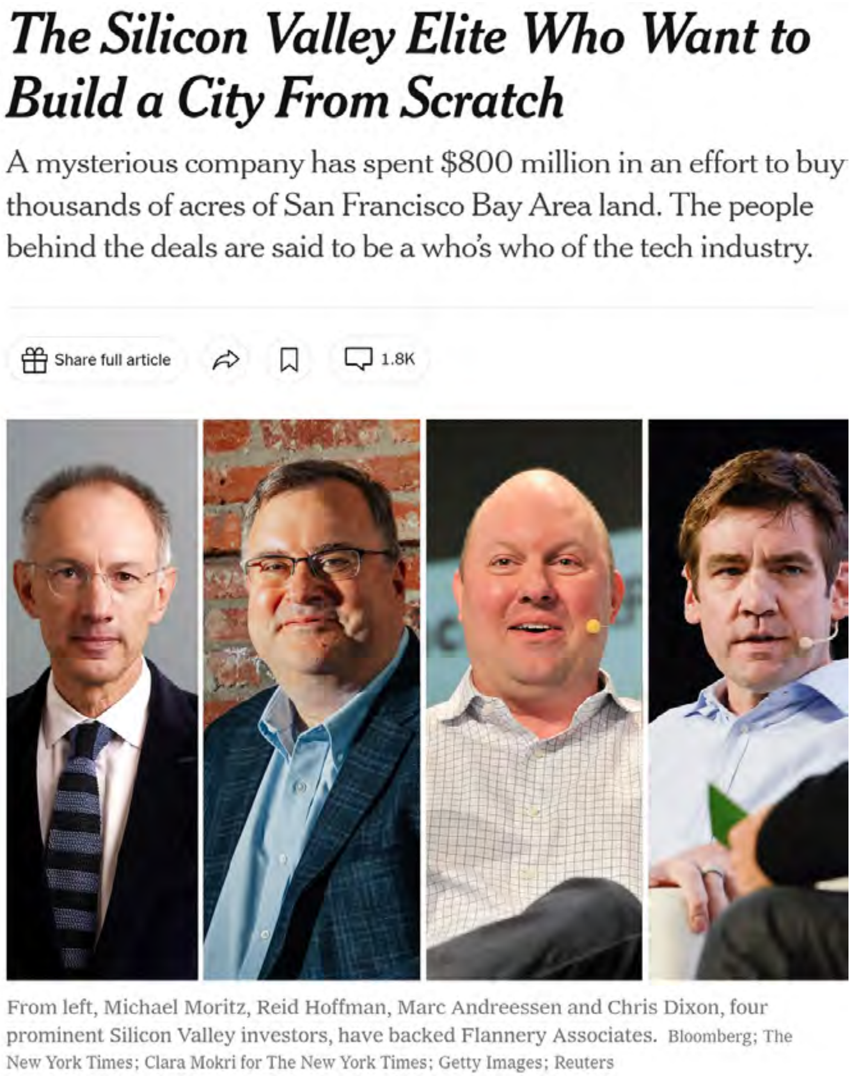
20640 rows × 8 columns

What is the target?

```
1 target
```

```
0        4.526
1        3.585
2        3.521
          ...
20637    0.923
20638    0.847
20639    0.894
Name: MedHouseVal, Length: 20640, dtype:
float64
```

Why predict this? Let's pretend we are these guys.



The Silicon Valley Elite Who Want to Build a City From Scratch

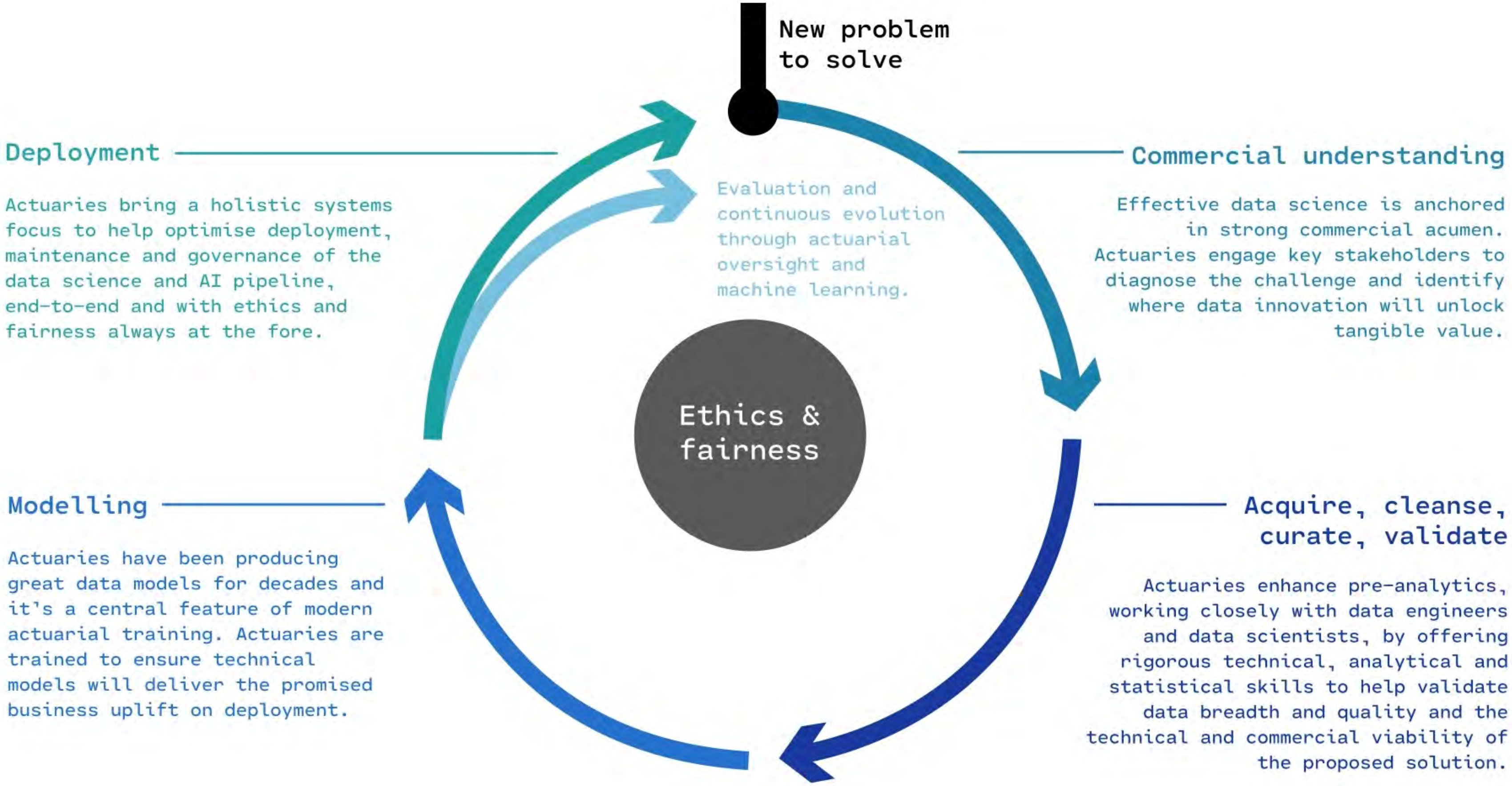
A mysterious company has spent $800 million in an effort to buy thousands of acres of San Francisco Bay Area land. The people behind the deals are said to be a who's who of the tech industry.

Share full article 1.8K

From left, Michael Moritz, Reid Hoffman, Marc Andreessen and Chris Dixon, four prominent Silicon Valley investors, have backed Flannery Associates. Bloomberg; The New York Times; Clara Mokri for The New York Times; Getty Images; Reuters

Source: Dougherty and Griffith (2023), The Silicon Valley Elite Who Want to Build a City From Scratch, New York Times.

An entire ML project



ML life cycle

Source: Actuaries Institute, Do Data Better.

Questions to answer in ML project

You fit a few models to the training set, then ask:

1. **(Selection)** Which of these models is the best?
2. **(Future Performance)** How good should we expect the final model to be on unseen data?

Set aside a fraction for a test set

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    features, target, random_state=42
)
```

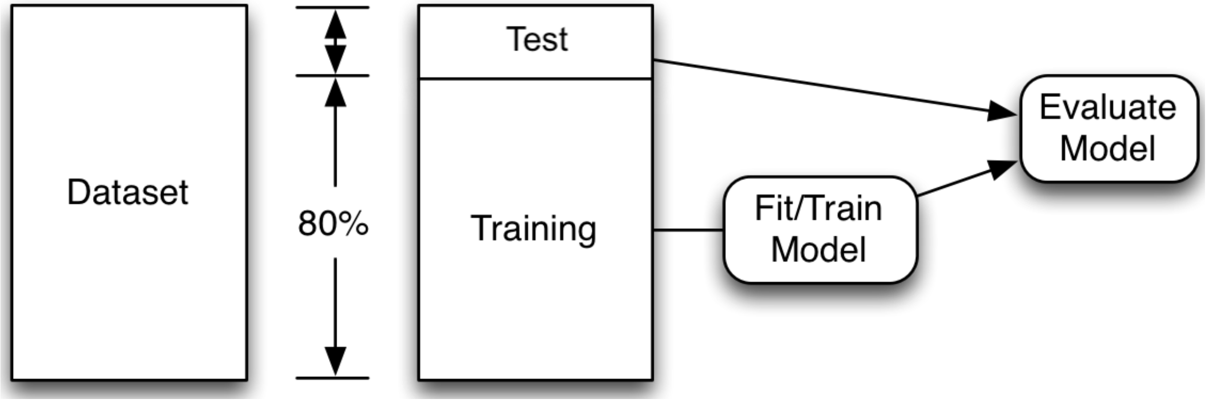
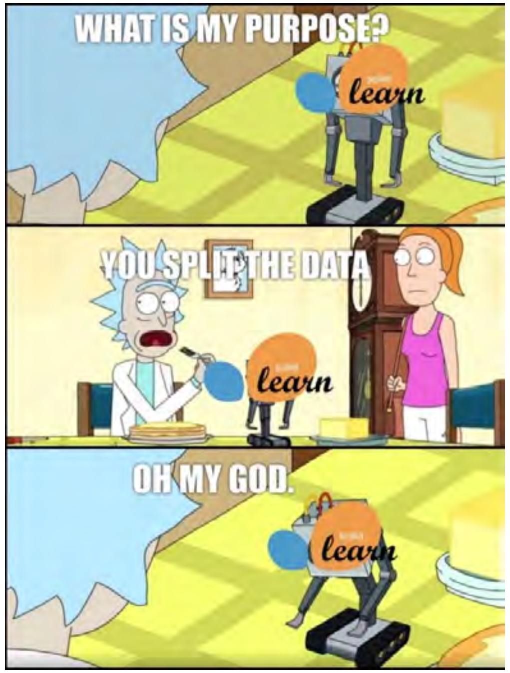


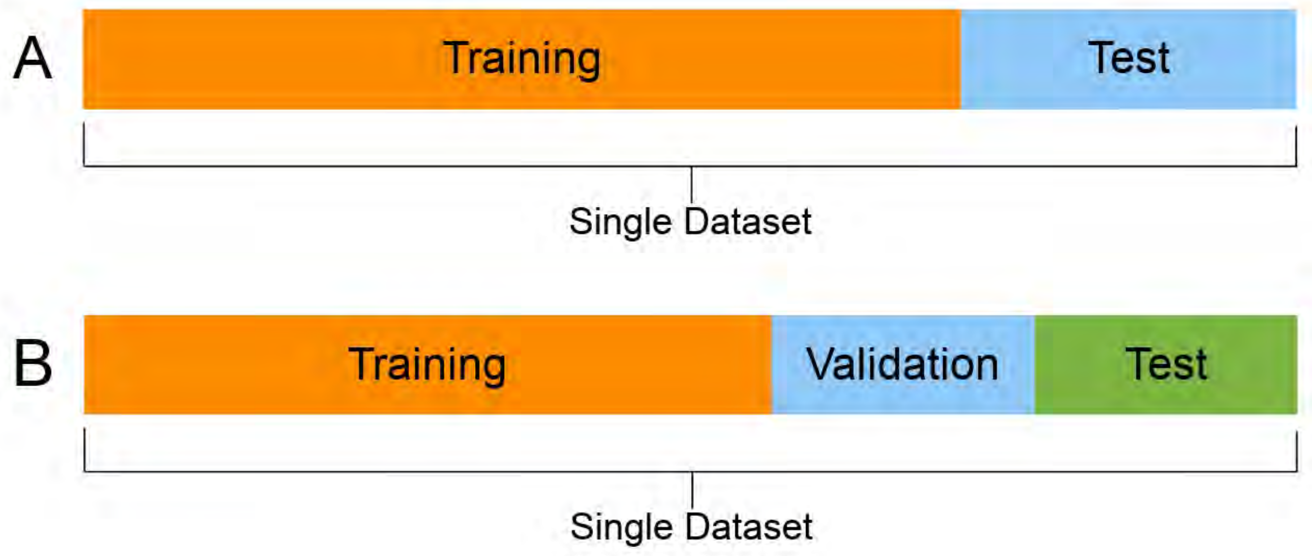
Illustration of a typical training/test split.

Note: Compare X_/y_ names, capitals & lowercase.



Our use of sklearn.

Basic ML workflow



Splitting the data.

1. For each model, fit it to the *training set*.
2. Compute the error for each model on the *validation set*.
3. Select the model with the lowest validation error.
4. Compute the error of the final model on the *test set*.

Source: Wikipedia.

Split three ways

```
# Thanks https://datascience.stackexchange.com/a/15136
X_main, X_test, y_main, y_test = train_test_split(
    features, target, test_size=0.2, random_state=1
)

# As 0.25 x 0.8 = 0.2
X_train, X_val, y_train, y_val = train_test_split(
    X_main, y_main, test_size=0.25, random_state=1
)

X_train.shape, X_val.shape, X_test.shape
```

```
((12384, 8), (4128, 8), (4128, 8))
```

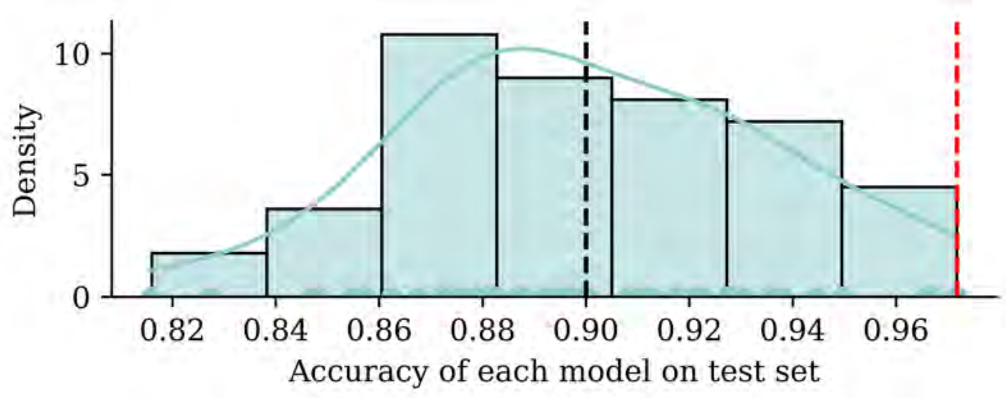
Why not use test set for both?

Thought experiment: have m classifiers: $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$.

They are just as good as each other in the long run

$$\mathbb{P}(\, f_i(\mathbf{X}) = Y \,) \; = \; 90\%, \quad \text{for } i = 1, \ldots, m.$$

Evaluate each model on the test set, some will be better than others.



Take the best, you'd think it has $\approx 98\%$ accuracy!

Lecture Outline

- California House Price Prediction
- **EDA & Baseline Model**
- Our First Neural Network
- Force positive predictions
- Preprocessing
- Early Stopping

The training set

```
X_train
```

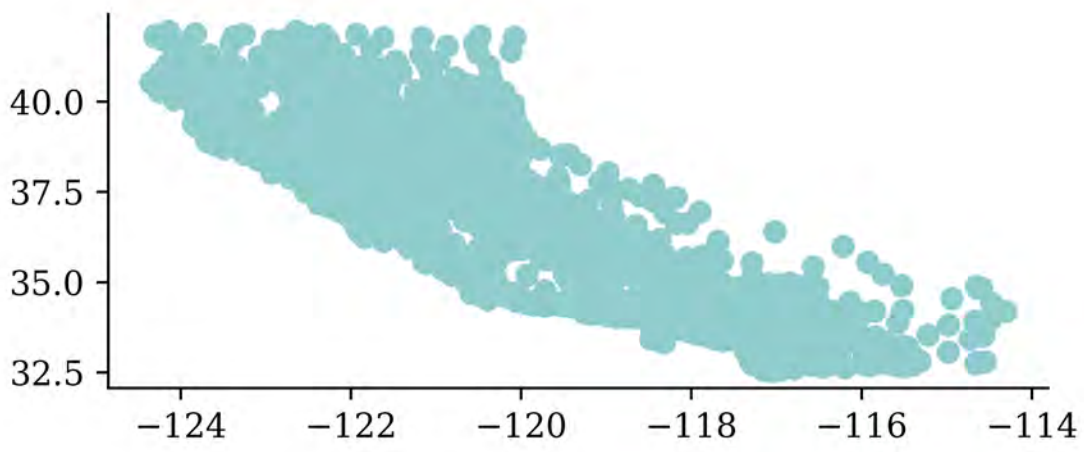
	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Long
9107	4.1573	19.0	6.162630	1.048443	1677.0	2.901384	34.63	-118.
13999	0.4999	10.0	6.740000	2.040000	108.0	2.160000	34.69	-116.
5610	2.0458	27.0	3.619048	1.062771	1723.0	3.729437	33.78	-118.
...	...	...	...	...	...	...	...	...
8539	4.0727	18.0	3.957845	1.079625	2276.0	2.665105	33.90	-118.
2155	2.3190	41.0	5.366265	1.113253	1129.0	2.720482	36.78	-119.
13351	5.5632	9.0	7.241087	0.996604	2280.0	3.870968	34.02	-117.

12384 rows × 8 columns

Location

Python's `matplotlib` package ≈ R's basic `plots`.

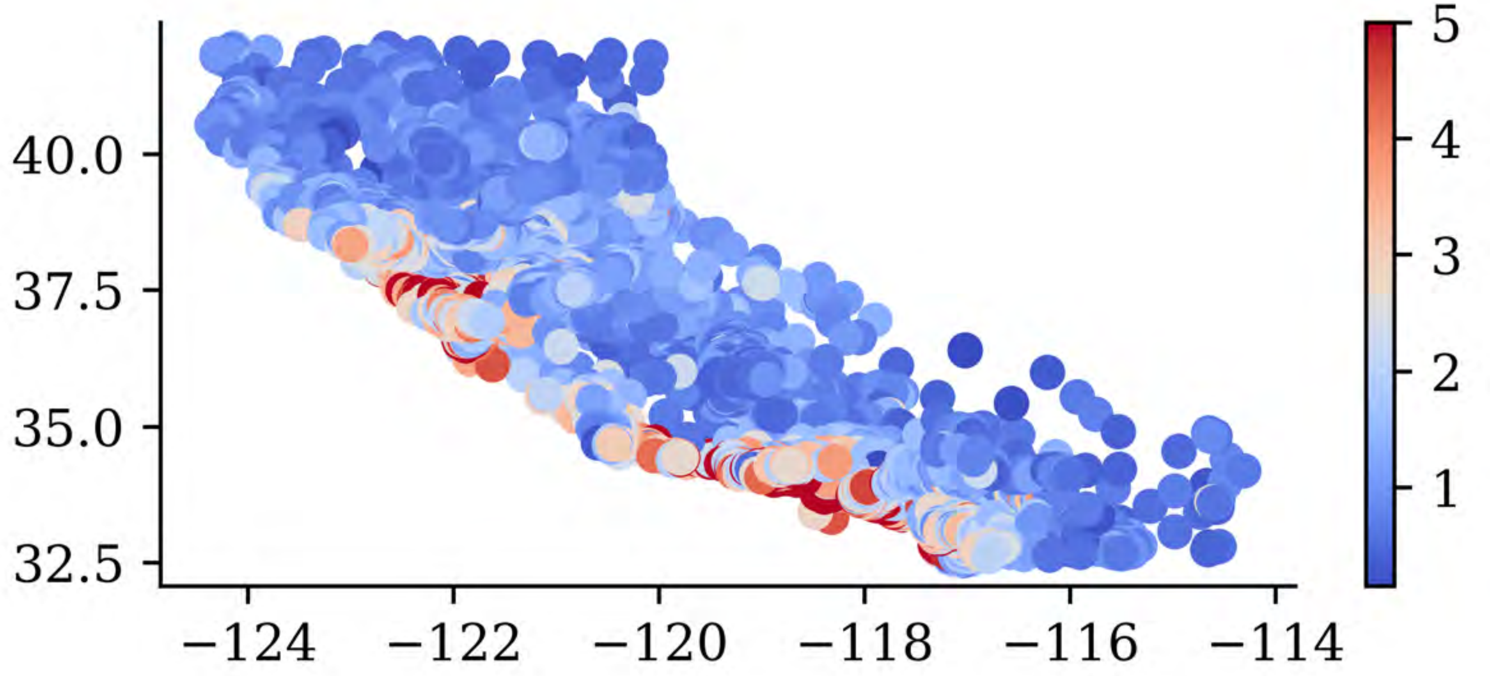
```python
import matplotlib.pyplot as plt

plt.scatter(features["Longitude"], features["Latitude"])
```



> **Note**
>
> There's no *analysis* in this EDA.

Location EDA

```python
plt.scatter(features["Longitude"], features["Latitude"], c=target, cmap="coolwarm")
plt.colorbar()
```

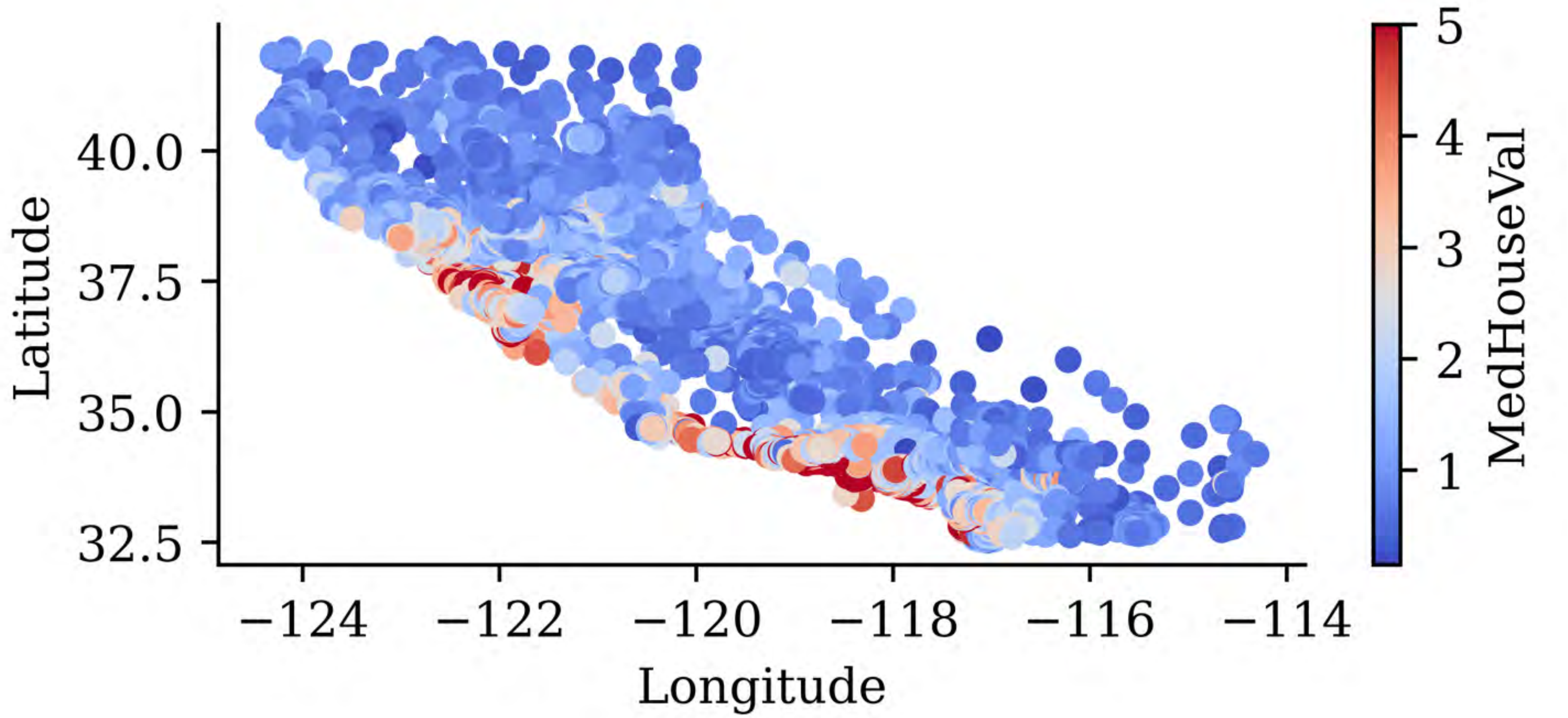


"We observe that the median house prices are higher closer to the coastline."

UNSW SYDNEY

Pandas can make plots directly

```python
both = pd.concat([features, target], axis=1)
both.plot(kind="scatter", x="Longitude", y="Latitude", c="MedHouseVal", cmap="coolwarm")
```



Features

```python
print(list(features.columns))
```

```
['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude',
'Longitude']
```

How many?

```python
num_features = len(features.columns)
num_features
```

```
8
```

Or

```python
num_features = features.shape[1]
features.shape
```

```
(20640, 8)
```

Linear Regression

$$\hat{y}_i = w_0 + \sum_{j=1}^{p} w_j x_{ij}.$$

```python
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train);
```

The w_0 is in `lr.intercept_` and the others are in

```python
print(lr.coef_)
```

```
[ 4.34267965e-01  9.88284781e-03 -9.39592954e-02  5.86373944e-01
 -1.58360948e-06 -3.59968968e-03 -4.26013498e-01 -4.41779336e-01]
```

Make some predictions

```python
X_train.head(3)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population
9107	4.1573	19.0	6.162630	1.048443	1677.0
13999	0.4999	10.0	6.740000	2.040000	108.0
5610	2.0458	27.0	3.619048	1.062771	1723.0

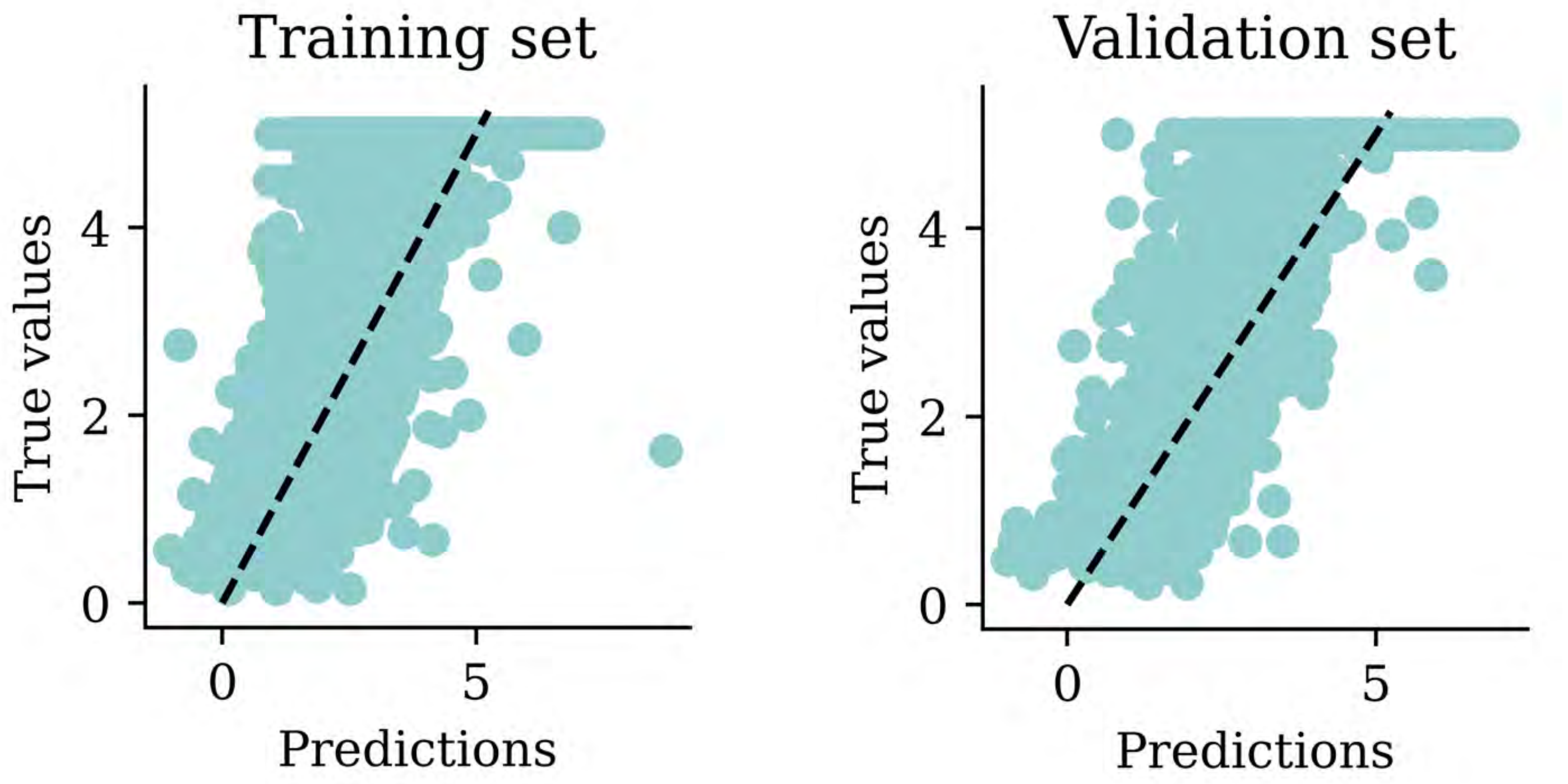
```python
y_pred = lr.predict(X_train.head(3))
y_pred
```

```
array([1.81699287, 0.0810446 , 1.62089363])
```

```python
prediction = lr.intercept_
for w_j, x_0j in zip(lr.coef_, X_train.iloc[0]):
    prediction += w_j * x_0j
prediction
```

```
1.8169928680677785
```

Plot the predictions



Calculate mean squared error

```python
import pandas as pd

y_pred = lr.predict(X_train)
df = pd.DataFrame({"Predictions": y_pred, "True values": y_train})
df["Squared Error"] = (df["Predictions"] - df["True values"]) ** 2
df.head(4)
```

	Predictions	True values	Squared Error
9107	1.816993	2.281	0.215303
13999	0.081045	0.550	0.219919
5610	1.620894	1.745	0.015402
13533	1.168949	1.199	0.000903

```python
df["Squared Error"].mean()
```

```
0.5291948207479792
```

Using `mean_squared_error`

```python
df["Squared Error"].mean()
```

```
0.5291948207479792
```

```python
from sklearn.metrics import mean_squared_error as mse

mse(y_train, y_pred)
```

```
0.5291948207479792
```

Store the results in a dictionary:

```python
mse_lr_train = mse(y_train, lr.predict(X_train))
mse_lr_val = mse(y_val, lr.predict(X_val))

mse_train = {"Linear Regression": mse_lr_train}
mse_val = {"Linear Regression": mse_lr_val}
```

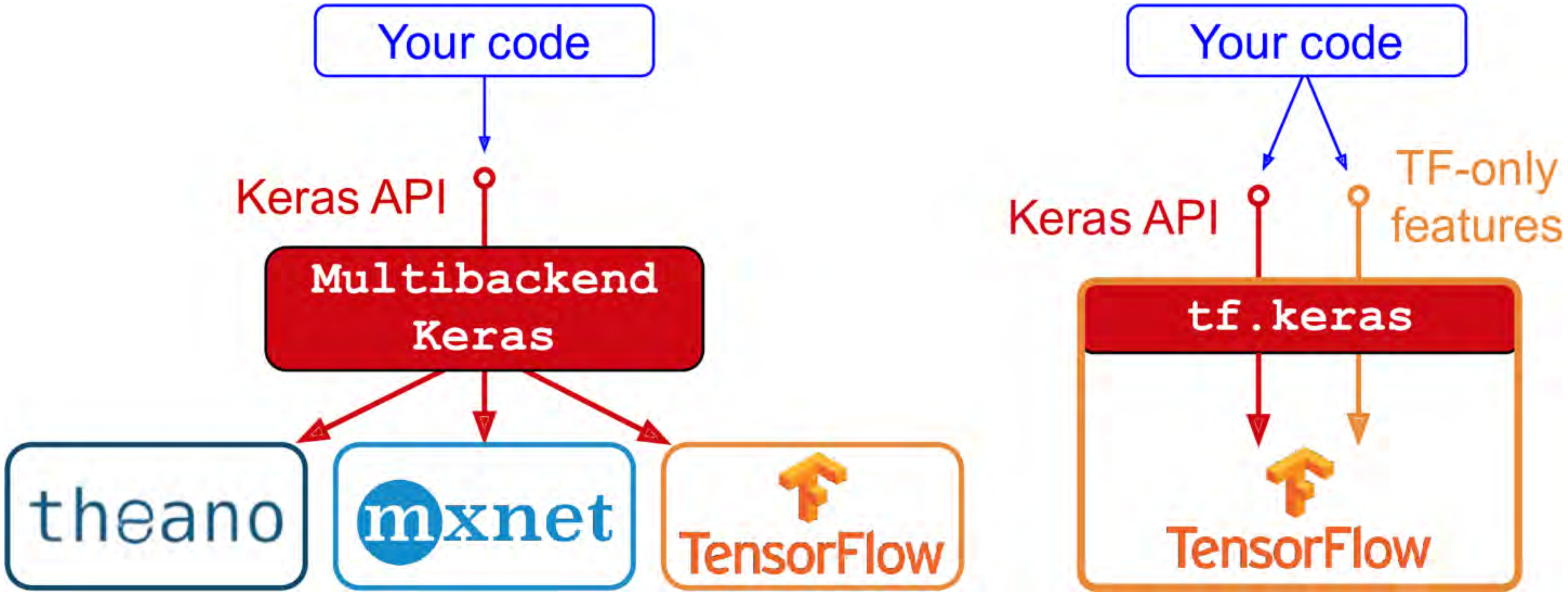
> **Tip**
>
> Thank about the units of the mean squared error. Is there a variation which is more interpretable?

Lecture Outline

- California House Price Prediction
- EDA & Baseline Model
- **Our First Neural Network**
- Force positive predictions
- Preprocessing
- Early Stopping

What are Keras and TensorFlow?

Keras is common way of specifying, training, and using neural networks. It gives a simple interface to *various backend* libraries, including Tensorflow.



Keras as a independent interface, and Keras as part of Tensorflow.

Source: Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 10-10.

Create a Keras ANN model

Decide on the architecture: a simple fully-connected network with one hidden layer with 30 neurons.

Create the model:

```python
from keras.models import Sequential
from keras.layers import Dense, Input

model = Sequential(
    [Input((num_features,)),
     Dense(30, activation="leaky_relu"),
     Dense(1, activation="leaky_relu")]
)
```

Inspect the model

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 30)	270
dense_1 (Dense)	(None, 1)	31

Total params: 301 (1.18 KB)
Trainable params: 301 (1.18 KB)
Non-trainable params: 0 (0.00 B)

The model is initialised randomly

```python
model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"
model.predict(X_val.head(3), verbose=0)
```

```
array([[-91.88699  ],
       [-57.336792 ],
       [ -1.2164348]], dtype=float32)
```

```python
model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"
model.predict(X_val.head(3), verbose=0)
```

```
array([[-63.595753],
       [-34.14082 ],
       [ 17.690414]], dtype=float32)
```

Controlling the randomness

```python
import random

random.seed(123)

model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"

display(model.predict(X_val.head(3), verbose=0))

random.seed(123)
model = Sequential([Dense(30, activation="leaky_relu"), Dense(1, activation="leaky_relu"

display(model.predict(X_val.head(3), verbose=0))
```

```
array([[ 1.3595750e+03],
       [ 8.2818079e+02],
       [-1.2993939e+00]], dtype=float32)
```

```
array([[ 1.3595750e+03],
       [ 8.2818079e+02],
       [-1.2993939e+00]], dtype=float32)
```

Fit the model

```python
random.seed(123)

model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="leaky_relu")
])

model.compile("adam", "mse")
%time hist = model.fit(X_train, y_train, epochs=5, verbose=False)
hist.history["loss"]
```

```
CPU times: user 1.04 s, sys: 97.2 ms, total: 1.14 s
Wall time: 873 ms
```

```
[18765.189453125,
 178.23837280273438,
 103.30640411376953,
 48.04053497314453,
 18.110933303833008]
```

Make predictions

```
y_pred = model.predict(X_train[:3], verbose=0)
y_pred
```

```
array([[ 0.5477159 ],
       [-1.525452  ],
       [-0.25848356]], dtype=float32)
```

Note

The `.predict` gives us a 'matrix' not a 'vector'. Calling `.flatten()` will convert it to a 'vector'.

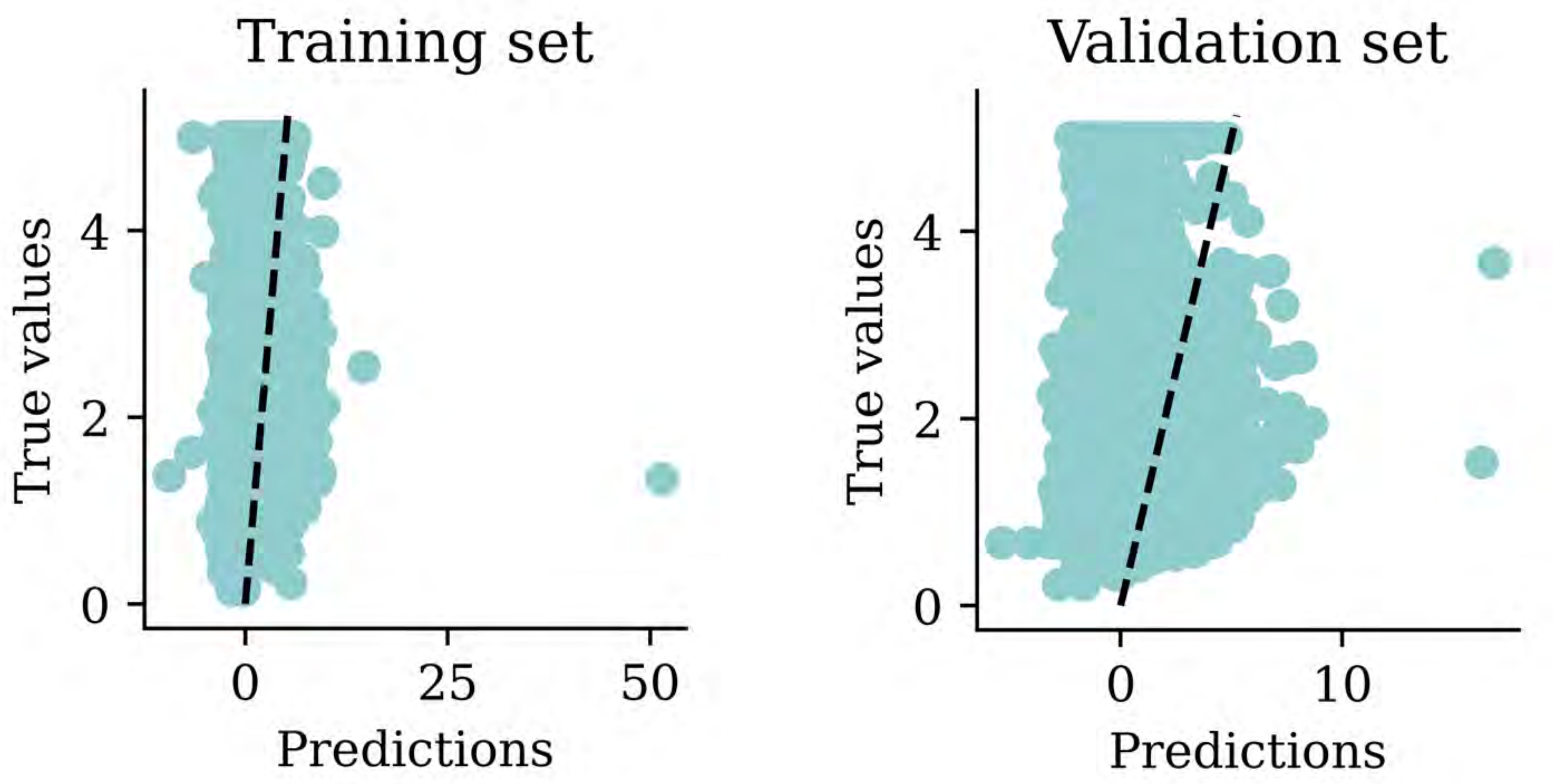
```
print(f"Original shape: {y_pred.shape}")
y_pred = y_pred.flatten()
print(f"Flattened shape: {y_pred.shape}")
y_pred
```

```
Original shape: (3, 1)
Flattened shape: (3,)
```

```
array([ 0.5477159 , -1.525452  , -0.25848356], dtype=float32)
```

UNSW SYDNEY

Plot the predictions



Assess the model

```
y_pred = model.predict(X_val, verbose=0)
mse(y_val, y_pred)
```

```
8.391657291598232
```

```
mse_train["Basic ANN"] = mse(
    y_train, model.predict(X_train, verbose=0)
)
mse_val["Basic ANN"] = mse(y_val, model.predict(X_val, verbose=0))
```

Some predictions are negative:

```
y_pred = model.predict(X_val, verbose=0)
y_pred.min(), y_pred.max()
```

```
(-5.371005, 16.863848)
```

```
y_val.min(), y_val.max()
```

```
(0.225, 5.00001)
```

Lecture Outline

- California House Price Prediction
- EDA & Baseline Model
- Our First Neural Network
- **Force positive predictions**
- Preprocessing
- Early Stopping

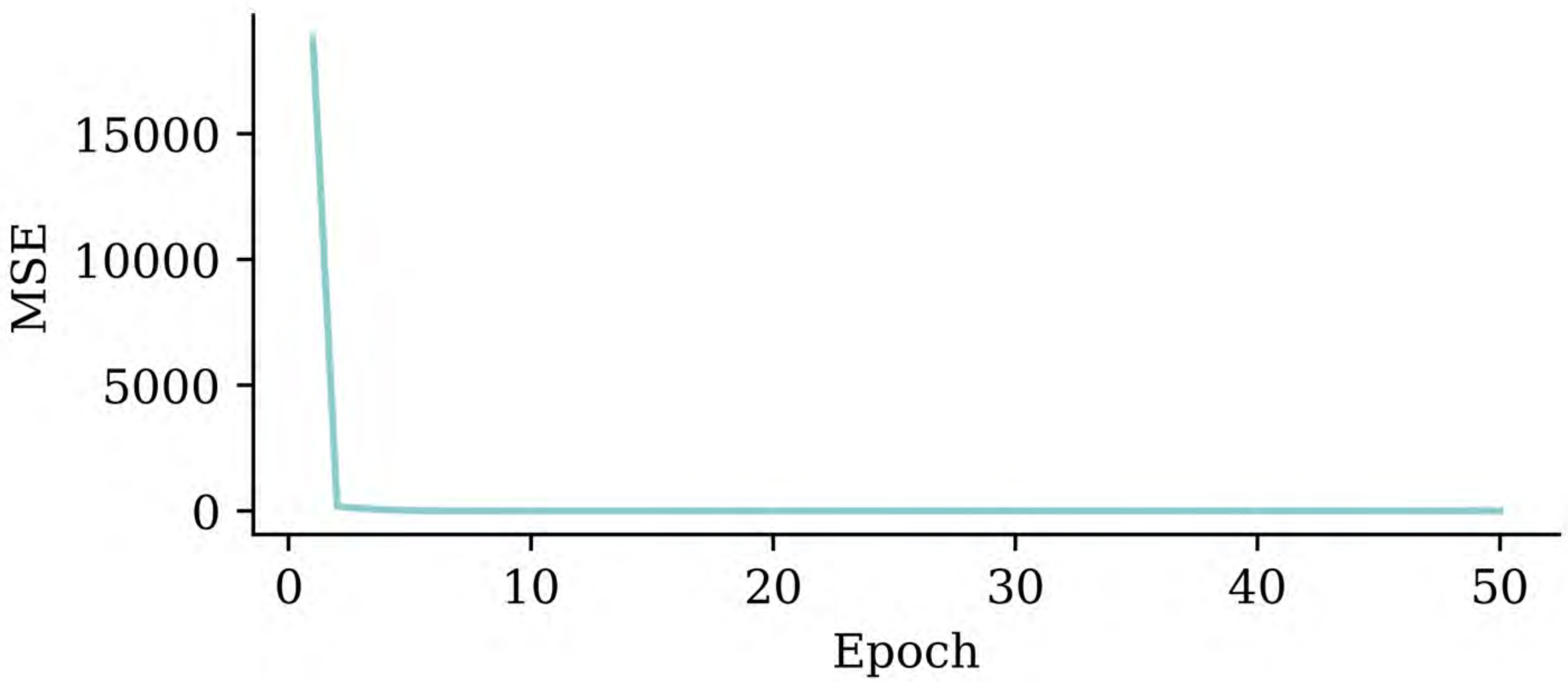
Try running for longer

```
random.seed(123)

model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="leaky_relu")
])

model.compile("adam", "mse")

%time hist = model.fit(X_train, y_train, epochs=50, verbose=False)
```

```
CPU times: user 8.4 s, sys: 711 ms, total: 9.11 s
Wall time: 6.39 s
```

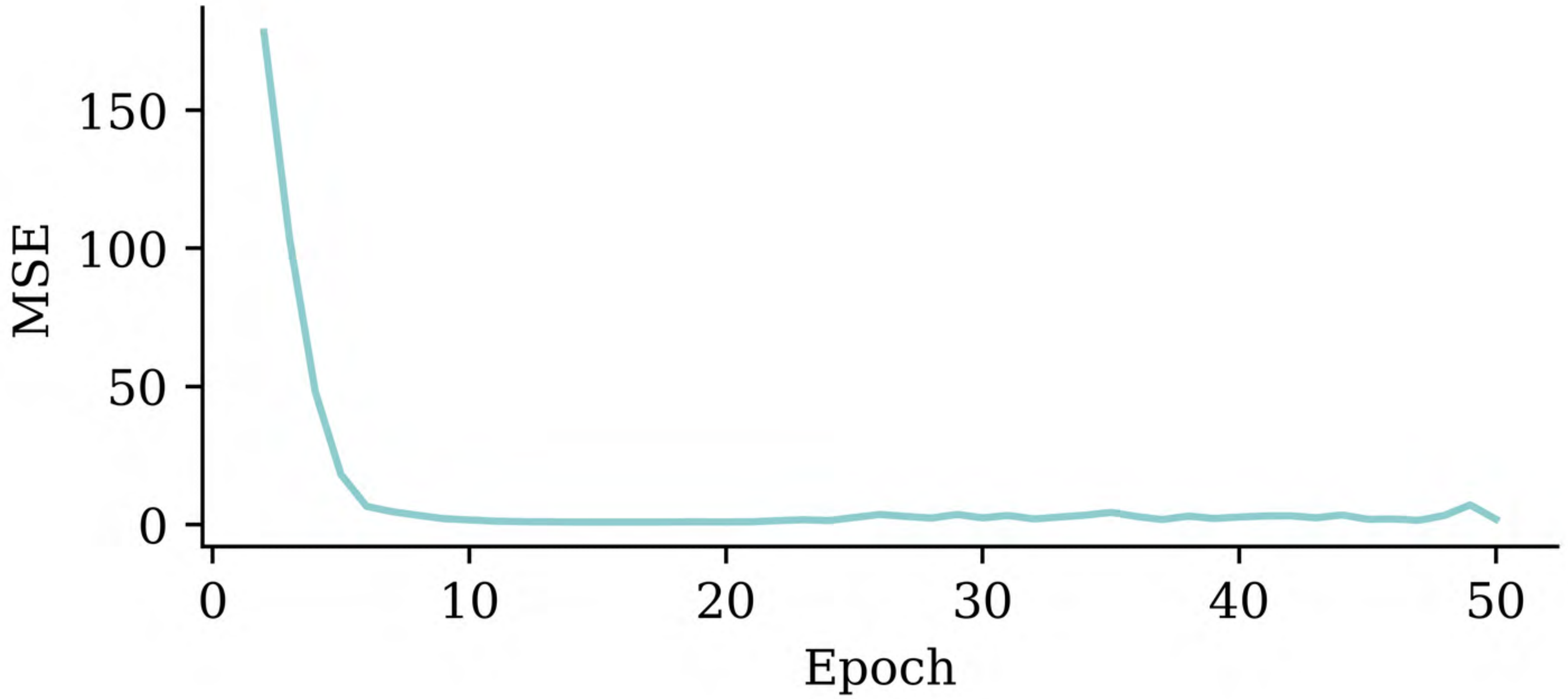
Loss curve

```python
plt.plot(range(1, 51), hist.history["loss"])
plt.xlabel("Epoch")
plt.ylabel("MSE");
```



Loss curve

```
plt.plot(range(2, 51), hist.history["loss"][1:])
plt.xlabel("Epoch")
plt.ylabel("MSE");
```

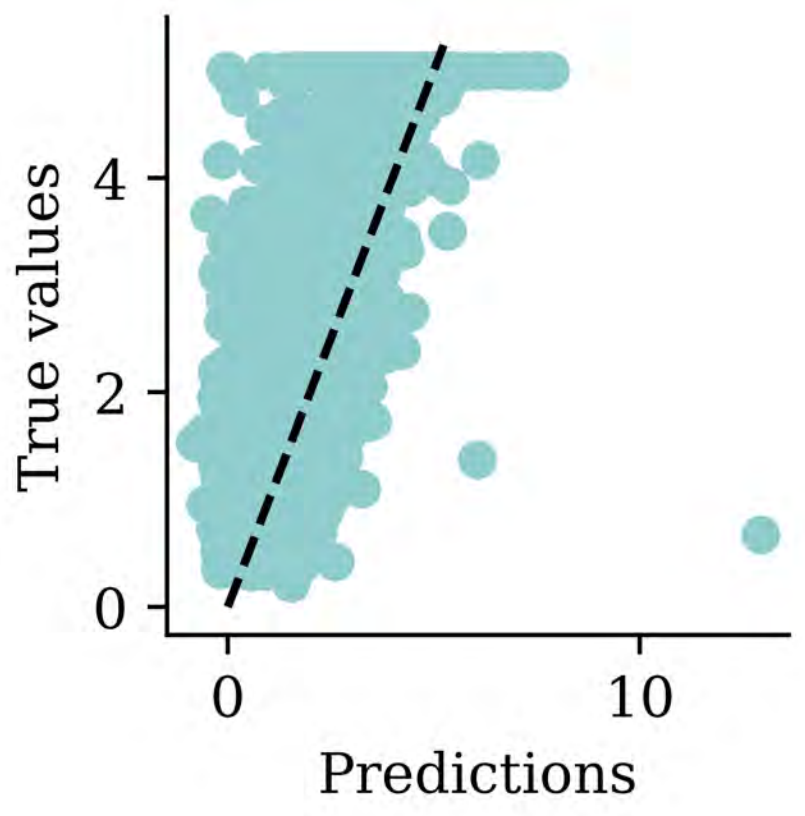


Predictions

```
y_pred = model.predict(X_val, verbose=0)
print(f"Min prediction: {y_pred.min():.2f}")
print(f"Max prediction: {y_pred.max():.2f}")
```

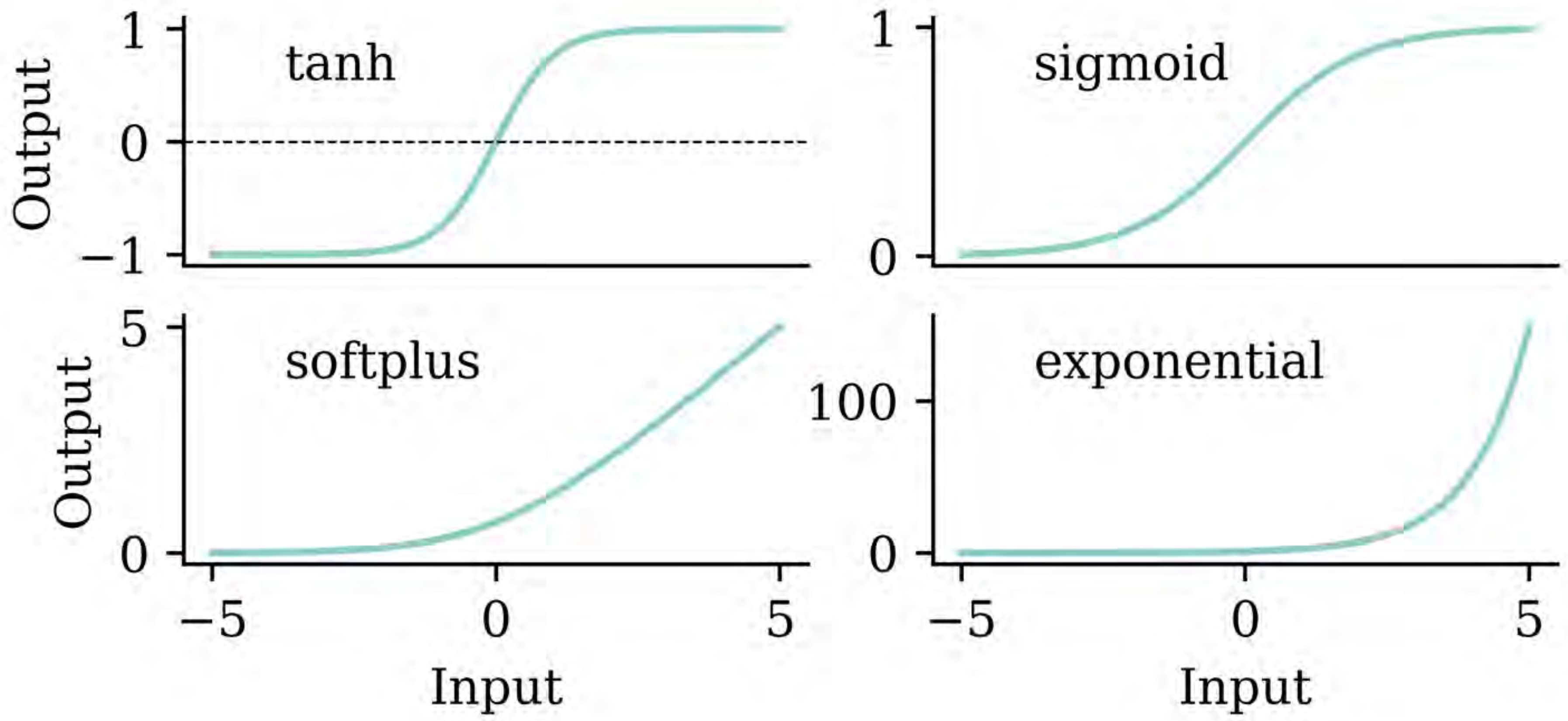
```
Min prediction: -0.79
Max prediction: 12.92
```

```
plt.scatter(y_pred, y_val)
plt.xlabel("Predictions")
plt.ylabel("True values")
add_diagonal_line()
```




```
mse_train["Long run ANN"] = mse(
    y_train, model.predict(X_train, ver
)
mse_val["Long run ANN"] = mse(y_val, mo
```

Try different activation functions

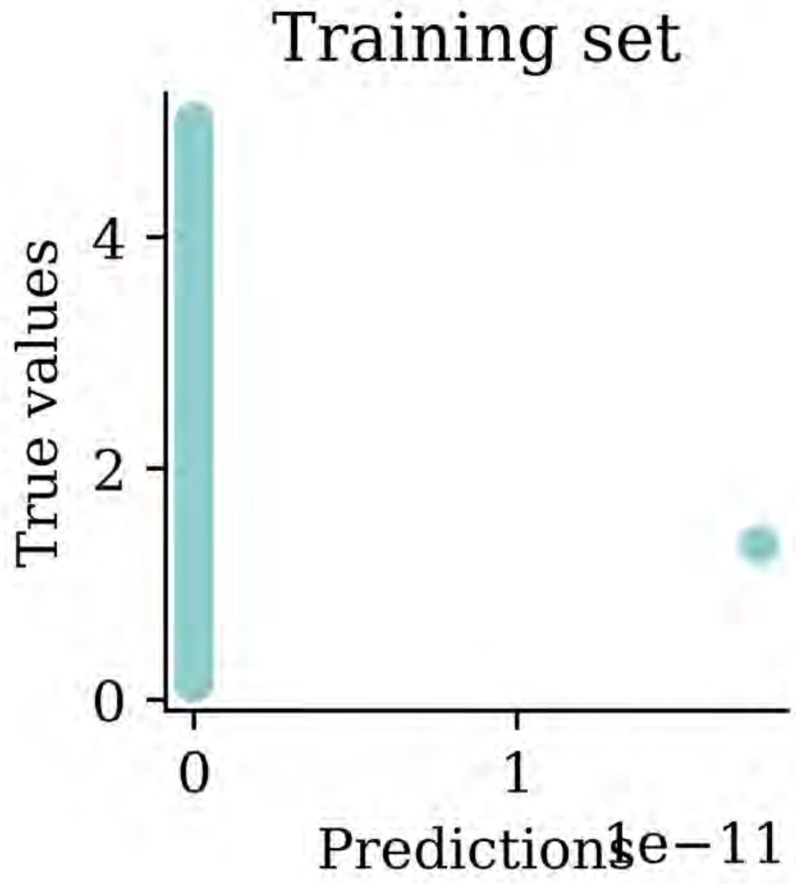


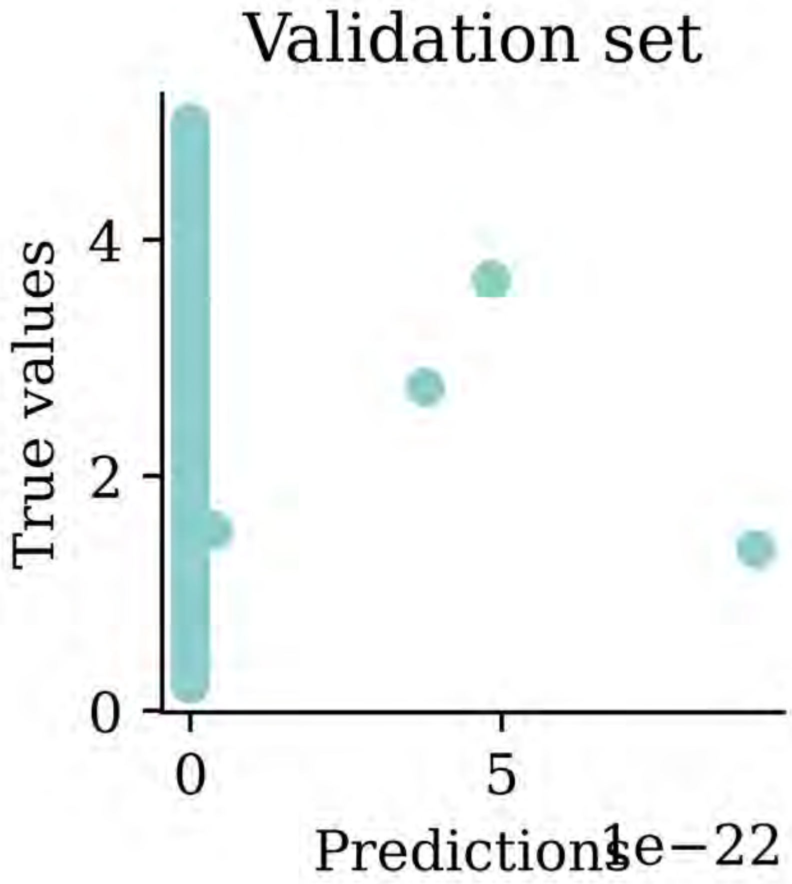
Enforce positive outputs (softplus)

```python
random.seed(123)

model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="softplus")
])

model.compile("adam", "mse")

%time hist = model.fit(X_train, y_train, epochs=50, \
    verbose=False)

import numpy as np
losses = np.round(hist.history["loss"], 2)
print(losses[:5], "...", losses[-5:])
```

```
CPU times: user 7.78 s, sys: 582 ms, total: 8.36 s
Wall time: 5.97 s
[1.856457e+04 5.640000e+00 5.640000e+00 5.640000e+00 5.640000e+00] ... [5.64 5.64 5.64 5.64
5.64]
```

Plot the predictions





Enforce positive outputs (e^x)

```
random.seed(123)

model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="exponential")
])

model.compile("adam", "mse")

%time hist = model.fit(X_train, y_train, epochs=5, verbose=False)

losses = hist.history["loss"]
print(losses)
```

```
CPU times: user 1.17 s, sys: 84.9 ms, total: 1.26 s
Wall time: 928 ms
[nan, nan, nan, nan, nan]
```

Same as transforming the target

4.1. Model

We fitted the following model:

$$\begin{aligned}\ln(\text{MEDIAN VALUE}) = {} & \text{INTERCEPT} + \beta_2\,\text{MEDIAN INCOME} + \beta_3\,\text{MEDIAN INCOME}^2 + \beta_4\,\text{MEDIAN INCOME}^3 \\ & + \beta_5 \ln(\text{MEDIAN(AGE)}) + \beta_6 \ln(\text{TOTAL ROOMS/POPULATION}) \\ & + \beta_7 \ln(\text{BEDROOMS/POPULATION}) + \beta_8 \ln(\text{POPULATION/HOUSEHOLDS}) \\ & + \beta_9 \ln(\text{HOUSEHOLDS}) \qquad (8)\end{aligned}$$

The polynomial regression used by researchers who first studied this dataset.

> **Note**
>
> Fitting $\ln(\text{Median Value})$ is mathematically identical to the `exponential` activation function in the final layer (but metrics are in different units).

Source: Pace and Barry (1997), Sparse Spatial Autoregressions, Statistics & Probability Letters.

Good to know others results

Table 1
OLS and SAR estimates for median housing prices across 20 640 California census block groups

	B_{ols}	t_{ols}	B_{sar}	t_{sar}
INTERCEPT	11.4939	275.7518	11.6637	402.5925
MEDIAN INCOME	0.4790	45.7768	0.0349	4.7104
MEDIAN INCOME^2	−0.0166	−9.4841	0.0100	8.4280
MEDIAN INCOME^3	−0.0002	−1.9157	−0.0007	−12.2444
ln(MEDIAN AGE)	0.1570	33.6123	−0.0421	−11.0942
ln(TOTAL ROOMS/POPULATION)	−0.8582	−56.1280	0.3098	24.5768
ln(BEDROOMS/POPULATION)	0.8043	38.0685	−0.1926	−11.8049
ln(POPULATION/HOUSEHOLDS)	−0.4077	−20.8762	−0.0342	−2.3582
ln(HOUSEHOLDS)	0.0477	13.0792	0.0034	1.5569
α			0.8536	
R^2	0.6078		0.8594	
Median $\|e\|$	0.2101		0.1084	
Execution time			1130 s	
Number of likelihood evaluations			10	

That basic model gets R^2 of 0.61, but their fancy model gets 0.86.

Source: Pace and Barry (1997), Sparse Spatial Autoregressions, Statistics & Probability Letters.

GPT can double-check these results

We fitted the following model:

$$\begin{aligned}\ln(\text{MEDIAN VALUE}) = {} & \text{INTERCEPT} + \beta_2\,\text{MEDIAN INCOME} + \beta_3\,\text{MEDIAN INCOME}^2 + \beta_4\,\text{MEDIAN INCOME}^3 \\ & + \beta_5 \ln(\text{MEDIAN(AGE)}) + \beta_6 \ln(\text{TOTAL ROOMS/POPULATION}) \\ & + \beta_7 \ln(\text{BEDROOMS/POPULATION}) + \beta_8 \ln(\text{POPULATION/HOUSEHOLDS}) \\ & + \beta_9 \ln(\text{HOUSEHOLDS}) \qquad (8)\end{aligned}$$

Fit the linear model from this screenshot to the data using Python, and calculate the R^2

The fitted linear model as per the screenshot yields an R^2 value of approximately 0.608. This indicates that approximately 60.8% of the variance in the log of median house value can be explained by the model.

Asking GPT to check it.

I'd previously given it the CSV of the data.

Analysis

python

Copy

```python
import numpy as np
import statsmodels.api as sm

# Prepare the data
cal_housing['MedInc2'] = cal_housing['MedInc'] ** 2
cal_housing['MedInc3'] = cal_housing['MedInc'] ** 3
cal_housing['log_HouseAge'] = np.log(cal_housing['HouseAge'])
cal_housing['log_AveRooms_Population'] = np.log(cal_housing['AveRooms'] /
cal_housing['Population'])
cal_housing['log_AveBedrms_Population'] = np.log(cal_housing['AveBedrms'] /
cal_housing['Population'])
cal_housing['log_Population_Households'] = np.log(cal_housing['Population'] /
cal_housing['AveOccup'])
cal_housing['log_AveOccup'] = np.log(cal_housing['AveOccup'])

# Define the target variable and the predictors
y = np.log(cal_housing['MedHouseVal'])
X = cal_housing[['MedInc', 'MedInc2', 'MedInc3', 'log_HouseAge',
'log_AveRooms_Population',
                 'log_AveBedrms_Population', 'log_Population_Households',
'log_AveOccup']]

# Add a constant to the model (the intercept term)
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()

# Get the R-squared value
r_squared = model.rsquared
r_squared
```

Result

```
0.6077752947111696
```

The code it wrote & ran.

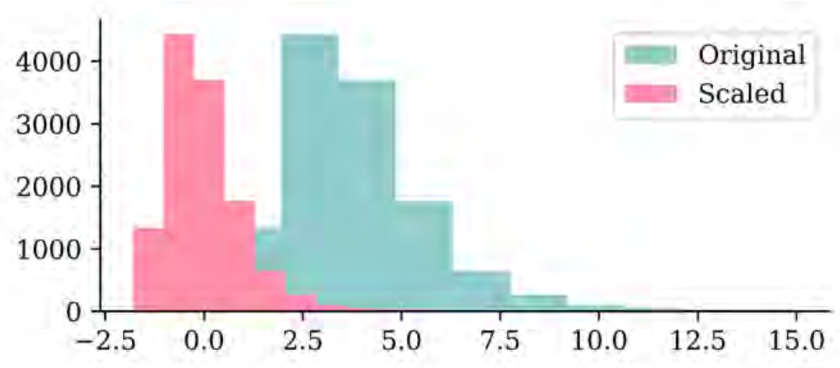
Lecture Outline

- California House Price Prediction
- EDA & Baseline Model
- Our First Neural Network
- Force positive predictions
- **Preprocessing**
- Early Stopping

Re-scaling the inputs

```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train_sc = scaler.transform(X_train)
X_val_sc = scaler.transform(X_val)
X_test_sc = scaler.transform(X_test)
```

```python
plt.hist(X_train.iloc[:, 0])
plt.hist(X_train_sc[:, 0])
plt.legend(["Original", "Scaled"]);
```



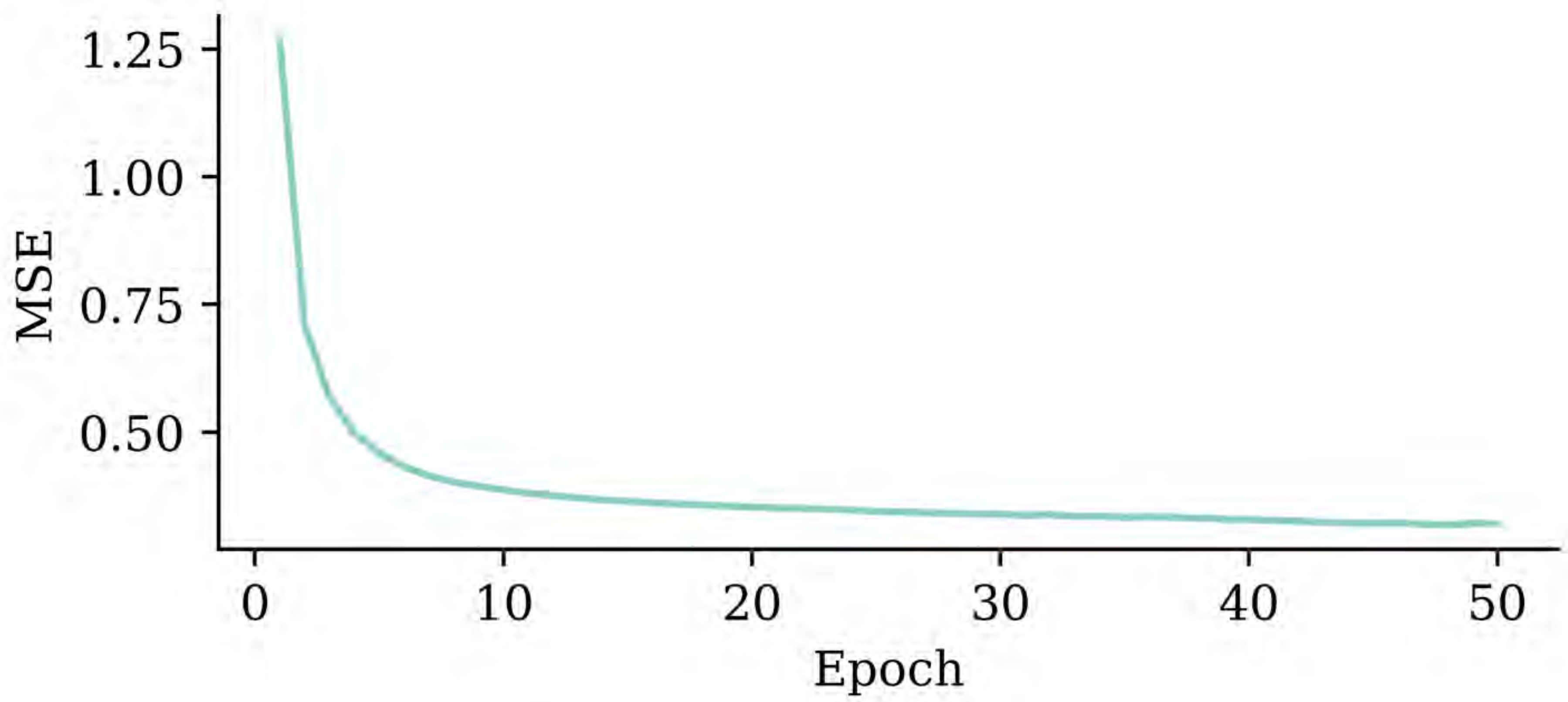
Same model with scaled inputs

```
random.seed(123)

model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="exponential")
])

model.compile("adam", "mse")

%time hist = model.fit( \
    X_train_sc, \
    y_train, \
    epochs=50, \
    verbose=False)
```

```
CPU times: user 8.31 s, sys: 634 ms, total: 8.94 s
Wall time: 6.35 s
```

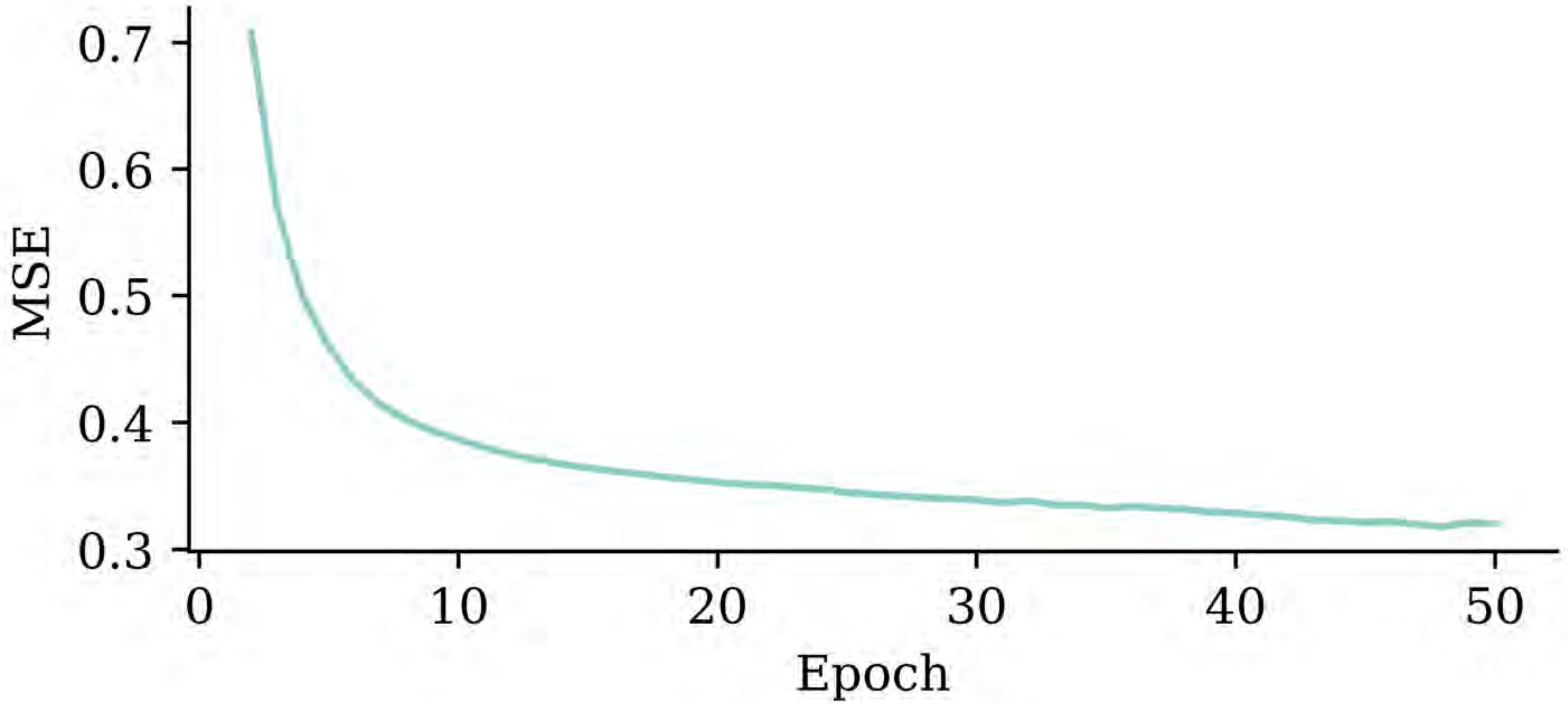
Loss curve

```python
plt.plot(range(1, 51), hist.history["loss"])
plt.xlabel("Epoch")
plt.ylabel("MSE");
```



Loss curve

```
plt.plot(range(2, 51), hist.history["loss"][1:])
plt.xlabel("Epoch")
plt.ylabel("MSE");
```



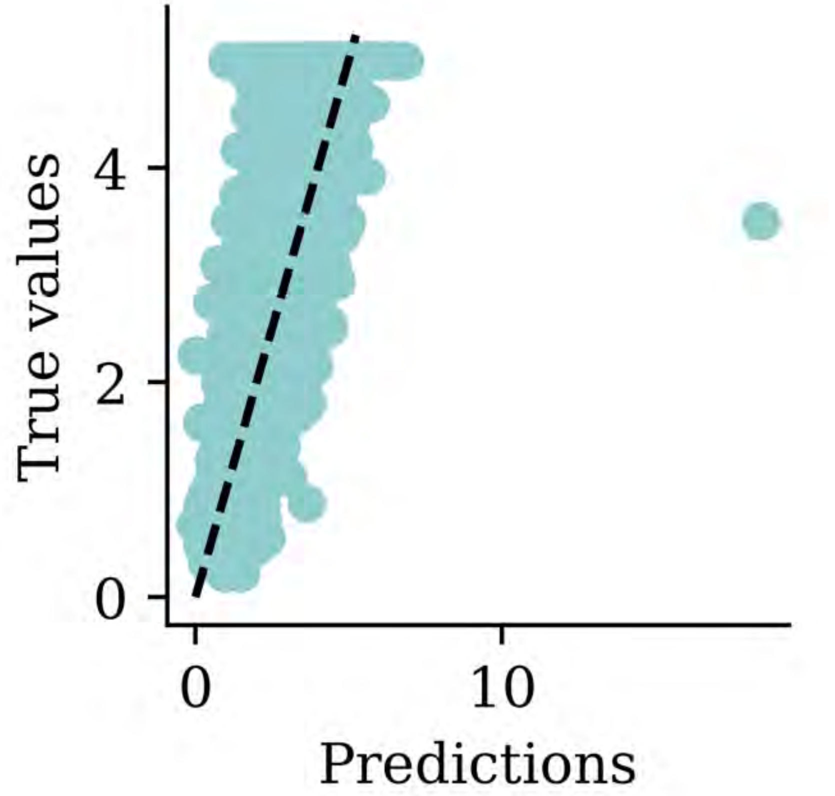
Predictions

```
y_pred = model.predict(X_val_sc, verbose=0)
print(f"Min prediction: {y_pred.min():.2f}")
print(f"Max prediction: {y_pred.max():.2f}")
```

```
Min prediction: 0.00
Max prediction: 18.45
```

```
plt.scatter(y_pred, y_val)
plt.xlabel("Predictions")
plt.ylabel("True values")
add_diagonal_line()
```

```
mse_train["Exp ANN"] = mse(
    y_train, model.predict(X_train_sc, 
)
mse_val["Exp ANN"] = mse(y_val, model.p
```



Comparing MSE (smaller is better)

On training data:

```
mse_train
```

```
{'Linear Regression': 0.5291948207479792,
 'Basic ANN': 8.374382131620425,
 'Long run ANN': 0.9770473035600079,
 'Exp ANN': 0.3182808342909683}
```

On validation data (expect *worse*, i.e. bigger):

```
mse_val
```

```
{'Linear Regression': 0.5059420205381367,
 'Basic ANN': 8.391657291598232,
 'Long run ANN': 0.9279673788287134,
 'Exp ANN': 0.36969620817676596}
```

Comparing models (train)

```
train_results = pd.DataFrame(
    {"Model": mse_train.keys(), "MSE": mse_train.values()}
)
train_results.sort_values("MSE", ascending=False)
```

	Model	MSE
1	Basic ANN	8.374382
2	Long run ANN	0.977047
0	Linear Regression	0.529195
3	Exp ANN	0.318281

UNSW SYDNEY

Comparing models (validation)

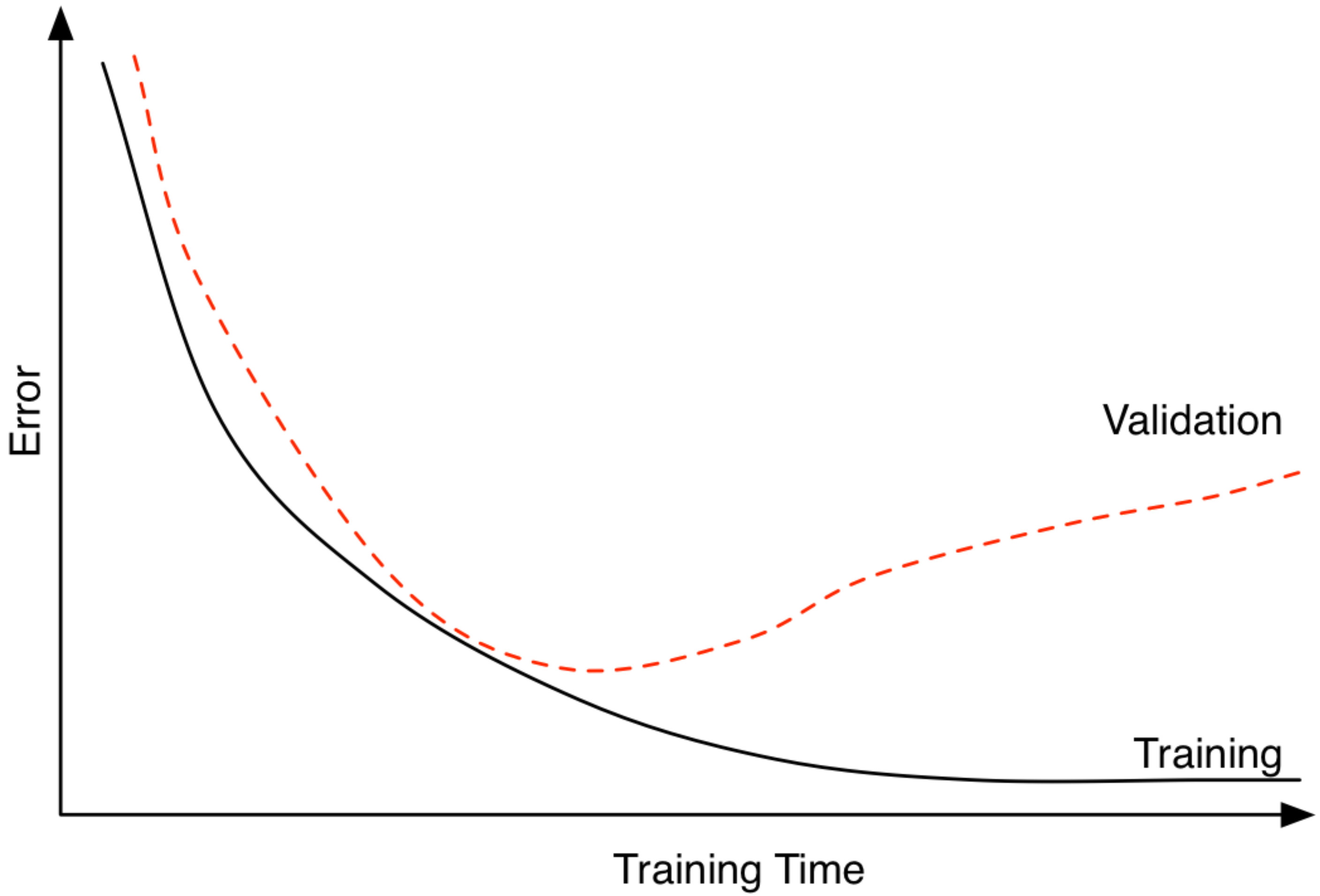
```python
val_results = pd.DataFrame(
    {"Model": mse_val.keys(), "MSE": mse_val.values()}
)
val_results.sort_values("MSE", ascending=False)
```

	Model	MSE
1	Basic ANN	8.391657
2	Long run ANN	0.927967
0	Linear Regression	0.505942
3	Exp ANN	0.369696

Lecture Outline

- California House Price Prediction
- EDA & Baseline Model
- Our First Neural Network
- Force positive predictions
- Preprocessing
- **Early Stopping**

Choosing when to stop training



Illustrative loss curves over time.

Source: Heaton (2022), Applications of Deep Learning, Part 3.4: Early Stopping.

Try early stopping

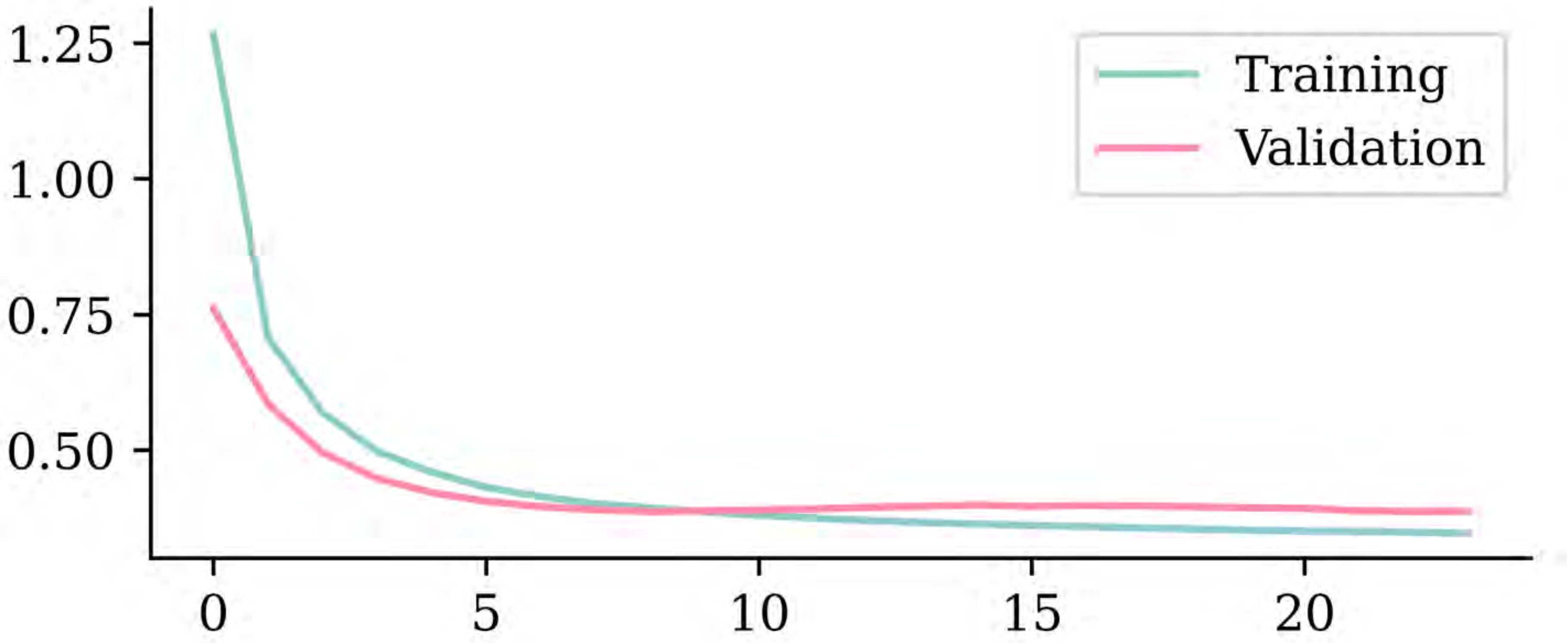
Hinton calls it a “beautiful free lunch”

```python
from keras.callbacks import EarlyStopping

random.seed(123)
model = Sequential([
    Dense(30, activation="leaky_relu"),
    Dense(1, activation="exponential")
])
model.compile("adam", "mse")

es = EarlyStopping(restore_best_weights=True, patience=15)

%time hist = model.fit(X_train_sc, y_train, epochs=1_000, \
    callbacks=[es], validation_data=(X_val_sc, y_val), verbose=False)
print(f"Keeping model at epoch #{len(hist.history['loss'])-10}.")
```

```
CPU times: user 5.52 s, sys: 410 ms, total: 5.93 s
Wall time: 4.24 s
Keeping model at epoch #14.
```

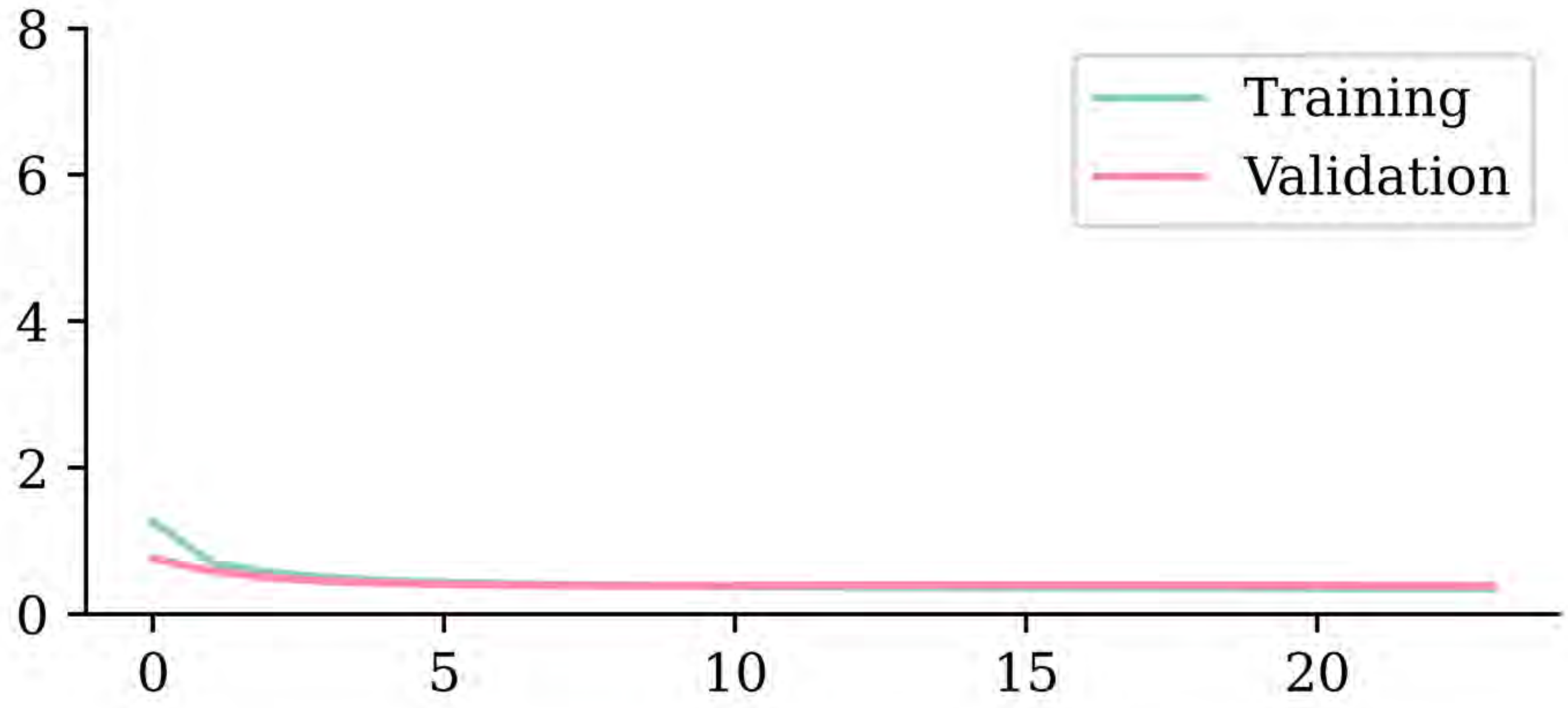
Loss curve

```
plt.plot(hist.history["loss"])
plt.plot(hist.history["val_loss"])
plt.legend(["Training", "Validation"]);
```

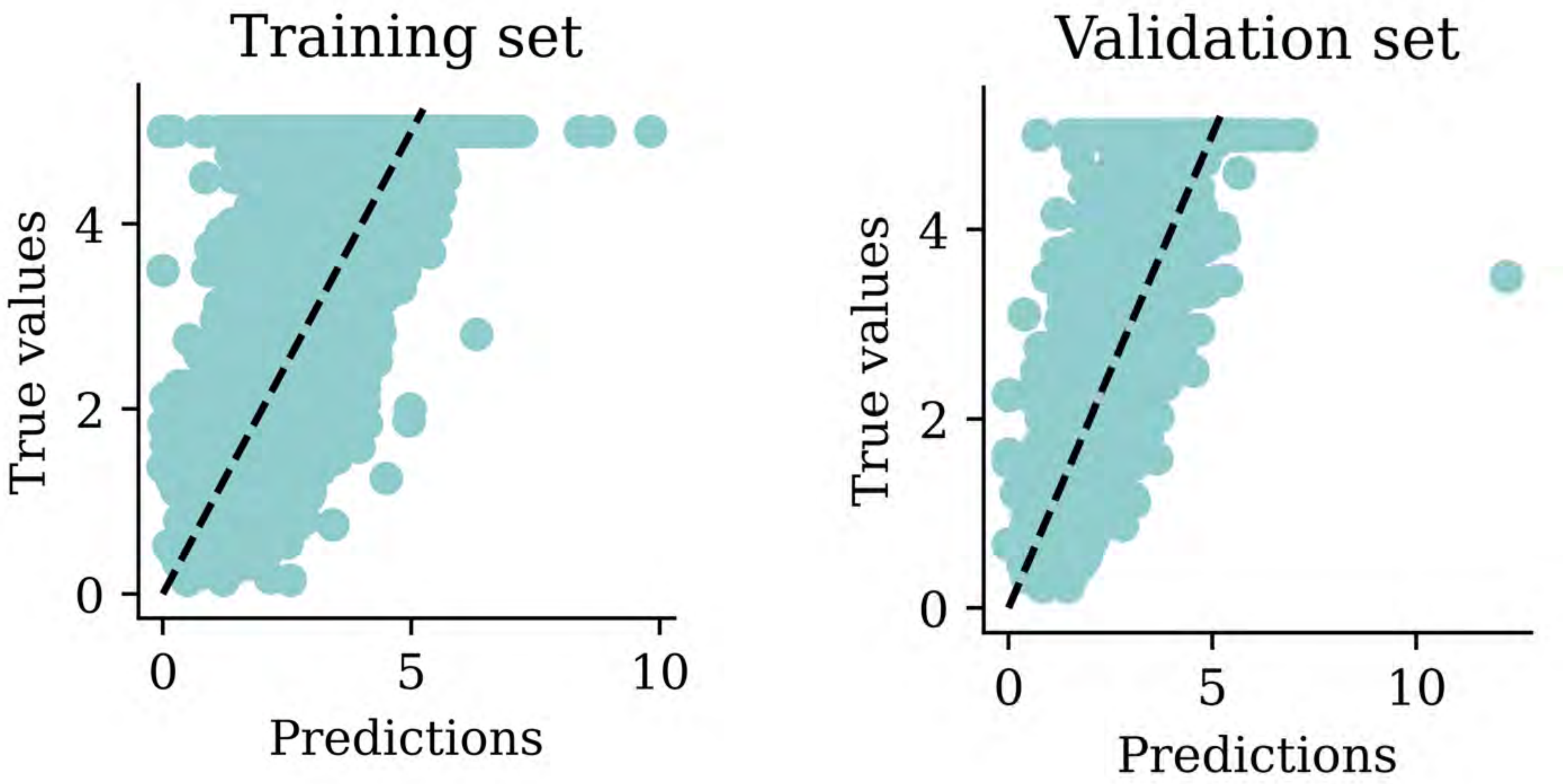


Loss curve II

```
plt.plot(hist.history["loss"])
plt.plot(hist.history["val_loss"])
plt.ylim([0, 8])
plt.legend(["Training", "Validation"]);
```



Predictions



Comparing models (validation)

	Model	MSE
1	Basic ANN	8.391657
2	Long run ANN	0.927967
0	Linear Regression	0.505942
4	Early stop ANN	0.386975
3	Exp ANN	0.369696

The test set

Evaluate *only the final/selected model* on the test set.

```
mse(y_test, model.predict(X_test_sc, verbose=0))
```

```
0.4026048522207643
```

```
model.evaluate(X_test_sc, y_test, verbose=False)
```

```
0.4026048183441162
```

Another useful callback

```python
from pathlib import Path
from keras.callbacks import ModelCheckpoint

random.seed(123)
model = Sequential(
    [Dense(30, activation="leaky_relu"), Dense(1, activation="exponential")]
)
model.compile("adam", "mse")
mc = ModelCheckpoint(
    "best-model.keras", monitor="val_loss", save_best_only=True
)
es = EarlyStopping(restore_best_weights=True, patience=5)
hist = model.fit(
    X_train_sc,
    y_train,
    epochs=100,
    validation_split=0.1,
    callbacks=[mc, es],
    verbose=False,
)
Path("best-model.keras").stat().st_size
```

19215

Package Versions

```python
from watermark import watermark
print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch
```

```
Python implementation: CPython
Python version       : 3.11.9
IPython version      : 8.24.0

keras     : 3.3.3
matplotlib: 3.9.0
numpy     : 1.26.4
pandas    : 2.2.2
seaborn   : 0.13.2
scipy     : 1.11.0
torch     : 2.0.1
tensorflow: 2.16.1
tf_keras  : 2.16.0
```

Glossary

- callbacks
- cost/loss function
- early stopping
- epoch
- Keras, Tensorflow, PyTorch
- matplotlib
- targets
- training/test split
- validation set