Lecture 21 - Machine Learning Models

Two types of machine learning

Supervised

An example is Email spam filtering.

Start with some labeled data that has some “features”

```
        < Features, Lable >
```

A set of email that is or is not spam. Probably start with a sizable set.

You train the “Model” to classify based on the existing set.

Then…

You give it new data that the model has never seen and let it identify if it is or is not spam.

```
        < Features, ?? >
```

The Model is then used to generate the Label.

Key terms : Features, Labels, Model

Key Idea - you don’t need to know what the “features” are.

Key Idea - you don’t need to write the “program” that performs the classification.

This is much more of a “science” than traditional programming.

Non-Supervised

This requires a constant evaluation function. How do you know when you are more successful.

We will get back to this.

How a “machine” learns

Example of square feet v.s. price of a house

Price	Value
$120	good
$136	maybe
$142	too much
$110	something is wrong

What is “Loss” - L-Squred, Minimization of Loss

Linear regression

Key Terms: Linear Regression, L-Squared, Minimization of Loss

Linear Model Example - Gradient Decent

Gradient Decent - this has a “tunable” learning rate parameter.

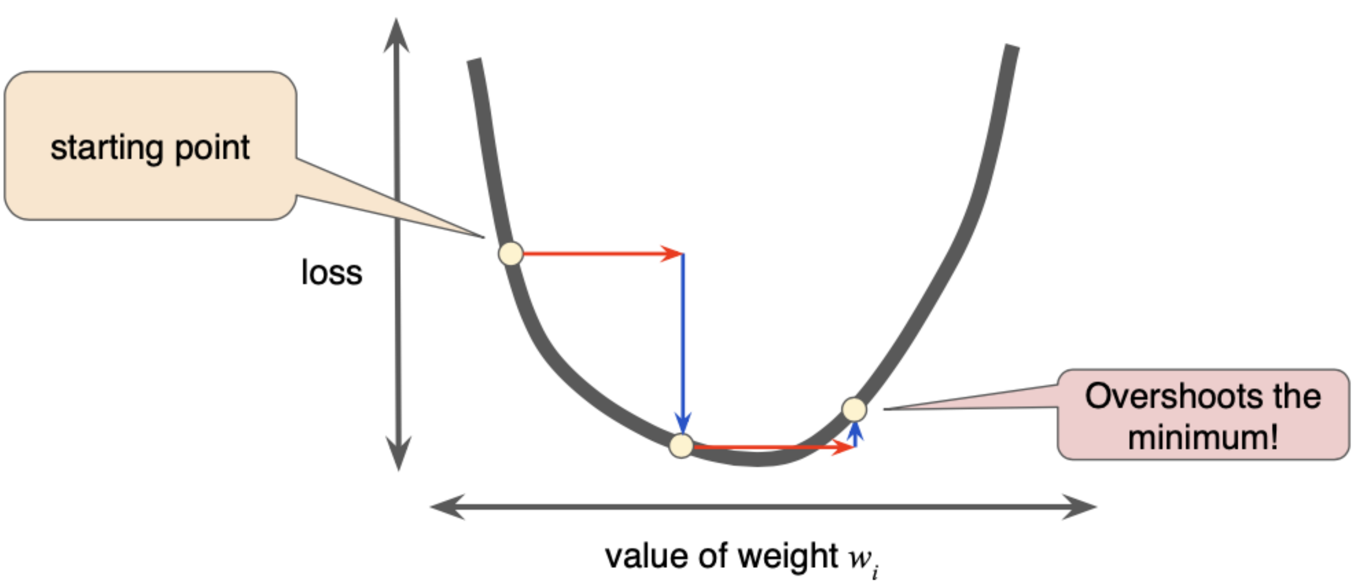
Gradient Decent combines a direction and a scale for each move. This is multiplied by the *learning rate*.

Given a gradient of 5.5 and a learning rate of 0.02 the algorithm will pick the next point as `5.5*0.02` or .11 away from the current point.

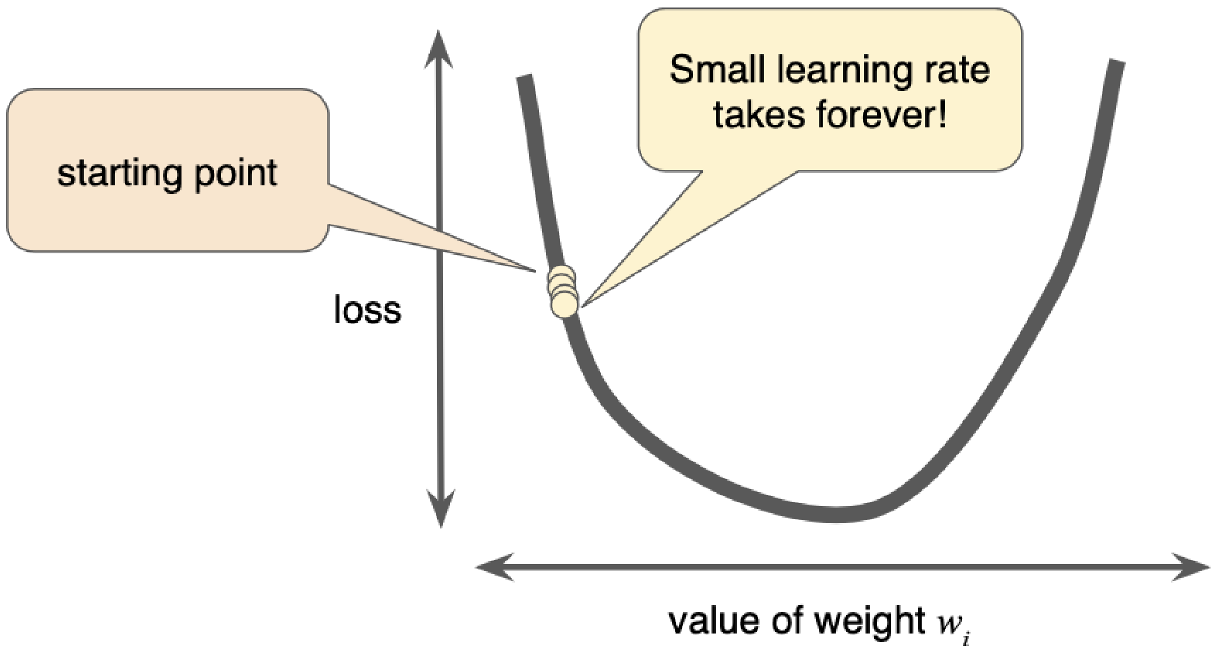
This is a “goldylox” type solution.

Example of Little Moves

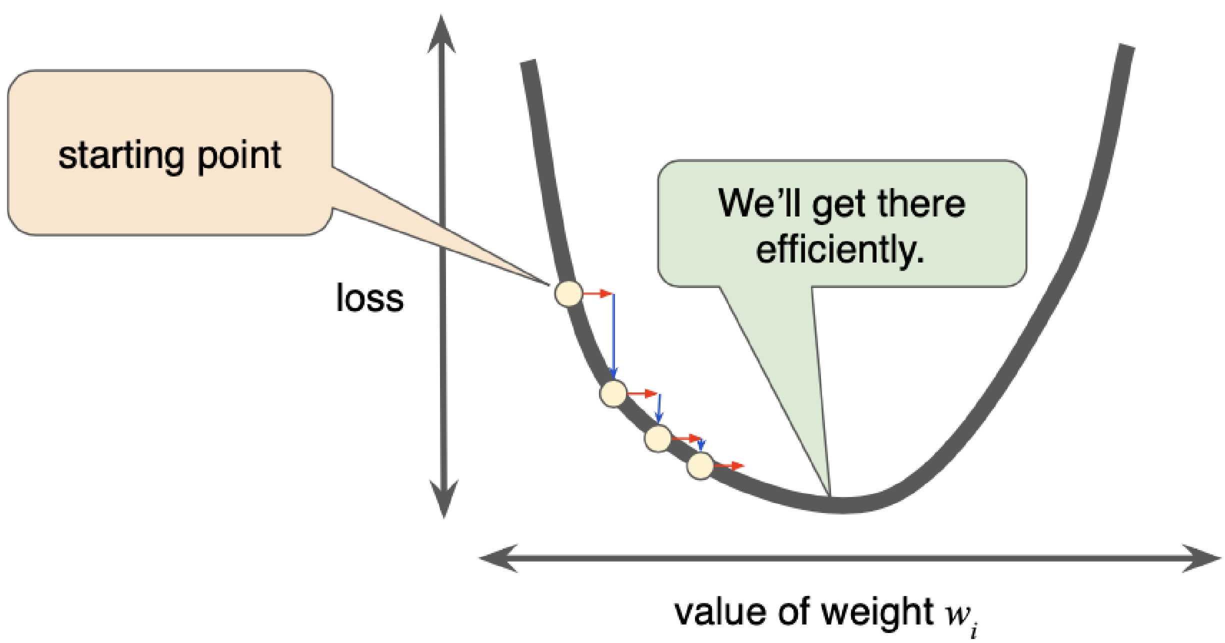
Example of Too Big



Example of Too Small



Example of Just Right



Nothing says that you get the “right” solution either.

Example - Interactive 3d.

https://blog.skz.dev/gradient-descent

There is math in this - The ideal learning rate in 2D is 1/(f(x))" - 2nd derivative of the function. In more dimensions it is the second partial derivative. The problem is the number of variables. Google base search has 400,000 variables - so that is 400k dimensions. The practical answer is to use some guess work based on “field knowledge” and run a set of tests.

This runs into the problem - if I have 10,000,000 input images and I guess it takes forever to train. There is good reason to believe that you can guess/train to figure out the “learning rate” on 10 to 1000 for a data size and then use this on the full set of data.

Over fitting

In the real world.

https://danluu.com/car-safety/

Learn Based on a Linear Model

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

from datetime import datetime
from packaging import version

import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np

print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >= 2,
    "This notebook requires TensorFlow 2.0 or above."

########################################################################34

data_size = 1000
# 80% of the data is for training.
train_pct = 0.8

train_size = int(data_size * train_pct)

# Create some input data between -1 and 1 and randomize it.
x = np.linspace(-1, 1, data_size)
np.random.shuffle(x)

# Generate the output data.
# y = 0.5x + 2 + noise
y = 0.5 * x + 2 + np.random.normal(0, 0.05, (data_size, ))
```

```
# Split into test and train pairs.
x_train, y_train = x[:train_size], y[:train_size]
x_test, y_test = x[train_size:], y[train_size:]

# print ( x_train )
plt.plot(x_train, y_train, 'o', color='black')
plt.show()

# input("Press Enter to continue...")
# exit(0)

#######################################################################33

logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

model = keras.models.Sequential([
    keras.layers.Dense(16, input_dim=1),
    keras.layers.Dense(1),
])

model.compile(
    loss='mse', # keras.losses.mean_squared_error
    optimizer=keras.optimizers.SGD(lr=0.2),
)

print("Training ... With default parameters, this takes less than 10 seconds.")
training_history = model.fit(
    x_train, # input
    y_train, # output
    batch_size=train_size,
    verbose=0, # Suppress chatty output; use Tensorboard instead
    epochs=100,
    validation_data=(x_test, y_test),
    callbacks=[tensorboard_callback],
)

print("Average test loss: ", np.average(training_history.history['loss']))

## Save our trained model
model.save ( "./save/model.h5" )
```

A runner that lets us predict

```

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf
from tensorflow import keras

model = keras.models.load_model ( "./save/model.h5" )

x = float(input("Enter a number> "))
pv = model.predict([x])

print ( "Input {x} Output {y}".format ( x=x, y=pv ) )

```