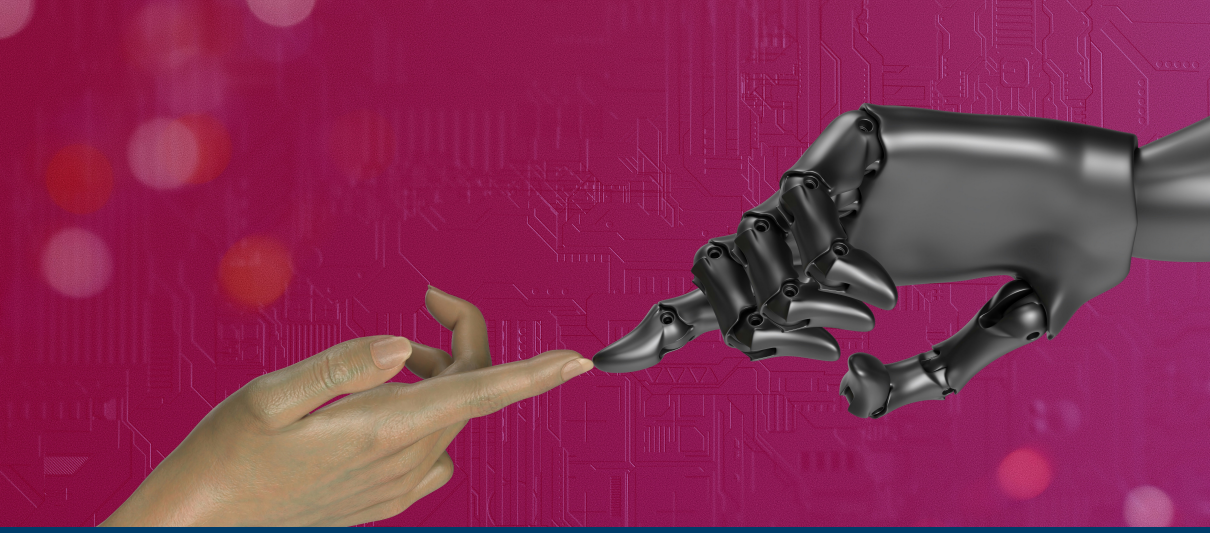


Hands-On Machine Learning

Dr. Derek Bridge | 2026/2027

UCC Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

School of Computer Science and Information Technology
Scoil na Ríomheolaíochta agus na Teicneolaíochta Faisnéise

8. Gradient Descent

In this lecture, we use the same housing data that we used in several of the previous lectures.

Gradient Descent

- **Gradient Descent** is a *generic* method for finding optimal solutions to problems that involve minimizing a loss function.
- It is a search in the model's parameter space for values of the parameters that minimize the loss function.

> start with an initial guess for the values of the parameters;
>
> **repeat**
>
> update the guess, changing the parameter values in a way that reduces the loss;
>
> **until** *convergence*;

- Ideally, it keeps searching until convergence — changes to the parameter values do not result in lower loss.
- The key to this algorithm is how to update the parameter values.

How to Update the Parameters

- To update the parameter values to reduce the loss:
  - Compute the **gradient vector**, $\nabla J(\boldsymbol{X}, \boldsymbol{y}, h)$.
    - But this points 'uphill' and we want to go 'downhill'.
    - And we want to make 'baby steps', so we use a **learning rate**, α, which is between 0 and 1.
  - So subtract α times the gradient vector from vector $\boldsymbol{w}$:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla J(\boldsymbol{X}, \boldsymbol{y}, h)$$

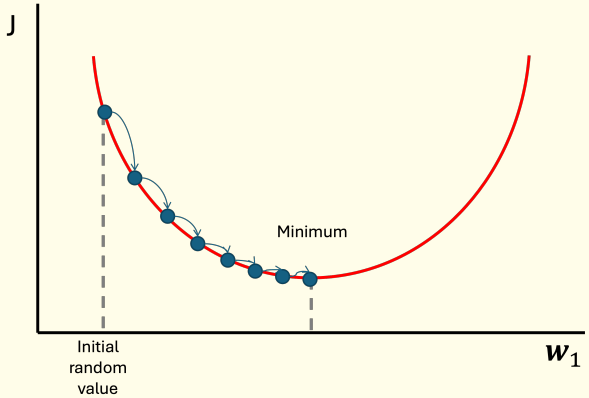
Or

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{m} \boldsymbol{X}^T (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

- (Note how this updates the parameters $\boldsymbol{w}$ simultaneously .)

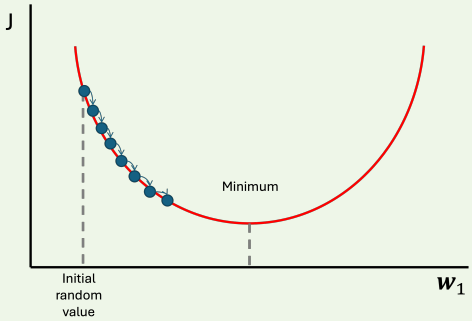
Baby Steps

- We'll use an example with a single feature/single parameter $\boldsymbol{w}_1$ in order to visualise.
- We update $\boldsymbol{w}_1$ gradually, one baby step at a time, until the algorithm converges on minimum loss.

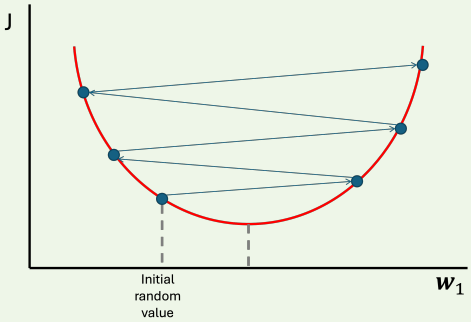


- The size of the steps is determined by the learning rate.

- If the learning rate is too small, it will take many updates until convergence:



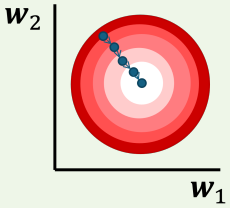
- If the learning rate is too big, the algorithm might jump across the valley — it may even end up with higher loss than before, making the next step bigger.
- This might make the algorithm **diverge**:



Scaling for Gradient Descent

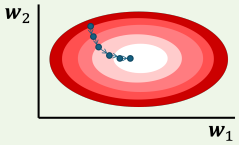
- If we are doing OLS regression using the Normal Equation, we do not need to scale the features.
- But if we are using Gradient Descent, we do need to scale the features.

- E.g. features 1 and 2 have similar ranges of values — a 'bowl':



- The algorithm goes straight towards the minimum.

- E.g. feature 1 has smaller values than feature 2 — an elongated 'bowl':



- Since feature 1 has smaller values, it takes a larger change in w_1 to affect the loss function, which is why it is elongated.
- It takes more steps to get to the minimum — steeply down but not really towards the goal, followed by a long march down a nearly flat valley.
- It makes it more difficult to choose a value for the learning rate that avoids divergence: a value that suits one feature may not suit another.

Batch Gradient Descent

In **Batch Gradient Descent**, the updates involve a calculation over the entire training set $\boldsymbol{X}$ on every iteration.

initialize $\boldsymbol{w}$ randomly;

repeat

$\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{m}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$;

until *convergence*;

- The algorithm **converges** when the parameters do not change between iterations. Equivalently, it converges when the loss doesn't change. In practice, we introduce a hyperparameter: the maximum. number of iterations.

Question

- Some people suggest a variant of Batch Gradient Descent in which the value of α is decreased over time, i.e. its value in later iterations is smaller.
- Why do they suggest this?
- And why isn't it necessary?

- Batch Gradient Descent is more suitable for smaller training sets.
- If the problem is convex, BGD is guaranteed to find the minimum (assuming the learning rate is small enough and the maximum number of iterations is high enough).

Stochastic Gradient Descent

In **Stochastic Gradient Descent**, the updates involve a calculation using a single randomly-selected training example $\boldsymbol{x}$.

initialize $\boldsymbol{w}$ randomly;

for *each epoch* **do**
  shuffle the training set;
  for *each example* $\boldsymbol{x}$ **do**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{x}(\boldsymbol{w}\boldsymbol{x} - y)$;
  end
end

- Stochastic Gradient Descent works even on huge datasets since only one example needs to be in memory in each iteration.
- But, because it is stochastic, the loss will not necessarily decrease on each iteration:
  - *On average*, the loss decreases, but in any one iteration, loss may go up or down.
  - Eventually, it will get close to the minimum (assuming a convex problem), but it will continue to go up and down a bit.
  - So, once you stop it, the parameters it finds will be close to the best, but not necessarily optimal.

Simulated Annealing

- Stochastic Gradient Descent may 'bounce' around the minimum loss.
- One solution to this is **Simulated Annealing**, where we gradually reduce the learning rate α.
  - Updates start out 'large' so you make progress.
  - But, over time, updates get smaller, allowing SGD to settle at or near the global minimum.
- The function that determines how to reduce the learning rate is called the **learning schedule**.
  - Reduce it too quickly and you may not converge on or near to the global minimum.
  - Reduce it too slowly and you may still bounce around a lot and, if stopped after too few iterations, may end up with a suboptimal solution.

Mini-Batch Gradient Descent

In **MiniBatch Gradient Descent**, the updates involve a calculation over randomly-selected subsets of the training set $\boldsymbol{X}$.

initialize $\boldsymbol{w}$ randomly;

for *each epoch* **do**
  shuffle the training set and split into mini-batches;
  for *each mini-batch* $\boldsymbol{X}_i, \boldsymbol{y}_i$ **do**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \frac{\alpha}{|\boldsymbol{X}_i|} \boldsymbol{X}_i^T (\boldsymbol{X}_i \boldsymbol{w} - \boldsymbol{y}_i)$;
  end
end

- Mini-Batch Gradient Descent is stochastic, therefore similar to SGD.
- But by working on subsets of the training set, rather than single examples, 'bouncing' may be gentler.
- It is very common when training Neural Networks.

Efficiency/scaling-up to large training sets

- Normal Equation:
  - This is linear in m, so can handle large training sets efficiently if they fit into main memory.
  - But it has to compute the inverse (or pseudo-inverse) of a $n \times n$ matrix, which takes time between quadratic and cubic in n, and so is only feasible for smallish n (up to a few thousand).
- Gradient Descent:
  - SGD scales really well to huge m.
  - All three Gradient Descent methods can handle huge n (even 100s of 1000s).

Finding the global minimum for OLS regression

- Normal Equation: guaranteed to find the global minimum.
- Gradient Descent: all a bit dependent on number of iterations, learning rate, learning schedule.

Feature scaling

- Normal Equation: scaling is not needed.
- Gradient Descent: scaling is needed.

Generality

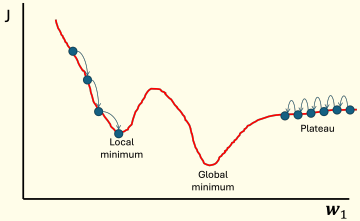
Gradient Descent is a general method, whereas the Normal Equation is only for OLS regression.

Convex Functions

- The loss function for OLS regression is convex and it has a slope that never changes abruptly.
- This gives us good 'guarantees' about reaching the minimum (depending on such things as running for long enough, using a learning rate that isn't too high, and whether we are using Batch, Mini-Batch or Stochastic Gradient Descent).
- But Gradient Descent is a generic method: you can use it to find the minima of other loss functions.

Non-Convex Functions

- Not all loss functions are convex, which can cause problems for Gradient Descent.

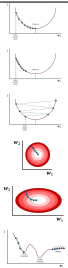


- The algorithm might converge to a local minimum, instead of the global minimum.
- It may take a long time to cross a plateau.

Gradient Descent for Non-Convex Problems

- One thing is to prefer Stochastic Gradient Descent (or Mini-Batch Gradient Descent): because of the way they 'bounce around', they might even escape a local minimum, and might even get to the global minimum.
- In this context, simulated annealing is also useful: updates start out 'large' allowing these algorithms to make progress and even escape local minima; but, over time, updates get smaller, allowing these algorithms to settle at or near the global minimum.
- But, if using simulated annealing, if you reduce the learning rate too quickly, you may still get stuck in a local minimum.

Image credits



I based these diagrams on ones to be found in A. Géron: Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (2nd edn), O'Reilly, 2019