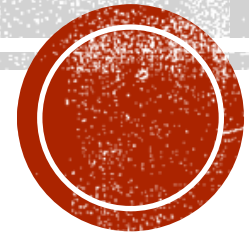# GRADIENT DESCENT

Dr. Brian Mc Ginley

TRAINING

- How do algorithms actually train? What is actually happening? Let's go to the basic linear regression again (I know there are lots of other algorithms instead but everything else follows similar patterns!)

$$y = w_0 + w_1 x$$

- Now what is happening? The "ideal" model is
  - The weights that minimise the "error".
  - What is this error? (Also called loss/residuals)
  - We need a procedure to actually train our model, to find the parameters that minimise the error.

RECAP

- All the scores: f1-score, accuracy, R^2 etc. are what we really want to be at their best. These are the things that tell us how well our model is doing. However, we can't optimise to these metrics directly.
  - Metrics on a dataset is what we care about (performance)
  - We typically cannot directly optimize for the metrics
  - Our loss/cost function should reflect the problem we are solving. We then hope it will yield models that will do well on our dataset

LOSS FUNCTION

- Firstly, what is the Loss function (error)?
  - How do we know if we have a bad model or just bad parameters?
  - This all depends on something called the Loss function (AKA the Cost function).
  - This measures the difference between what the model predicts the value should be and what the training sample actually was.
  - You may think this is straight forward but actually there are lots of ways of measuring the loss as well and choosing the wrong one can be like choosing the wrong type of model.
  - A typical example is the mean squared error:

$$L = \frac{1}{2m} \sum_{i=1}^{m} (y^i - \widehat{y^i})^2$$

  - There are variations (maybe the 2 isn't there), maybe the order after the $\sum$ is different but they amount to the same thing. If we minimise this, we minimise our Loss/Error

MEAN SQUARED ERROR

- Let's simplify our model even further just to make things look easier
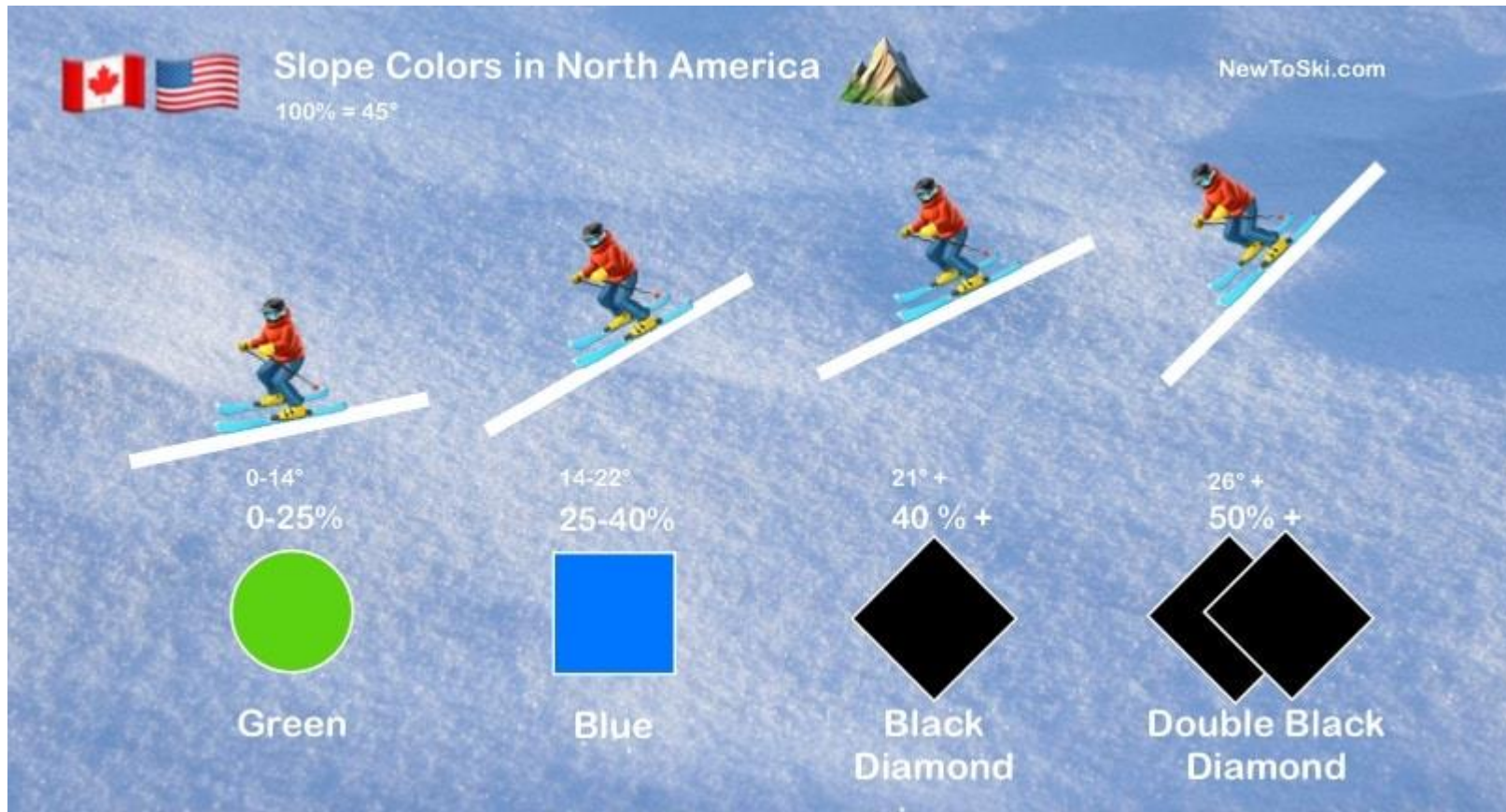
$$y = w_1 x$$

- So that the mean squared error (loss) is:

$$L = \frac{1}{2m} \sum_{i=1}^{m} (y^i - w_1 x^i)^2$$

- To train the parameters - we change w_1 until the loss function is at its minimum. So how do we find the minimum?
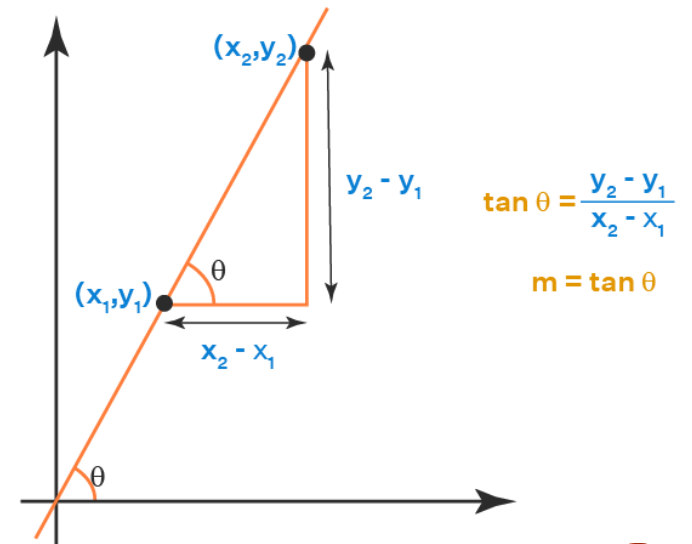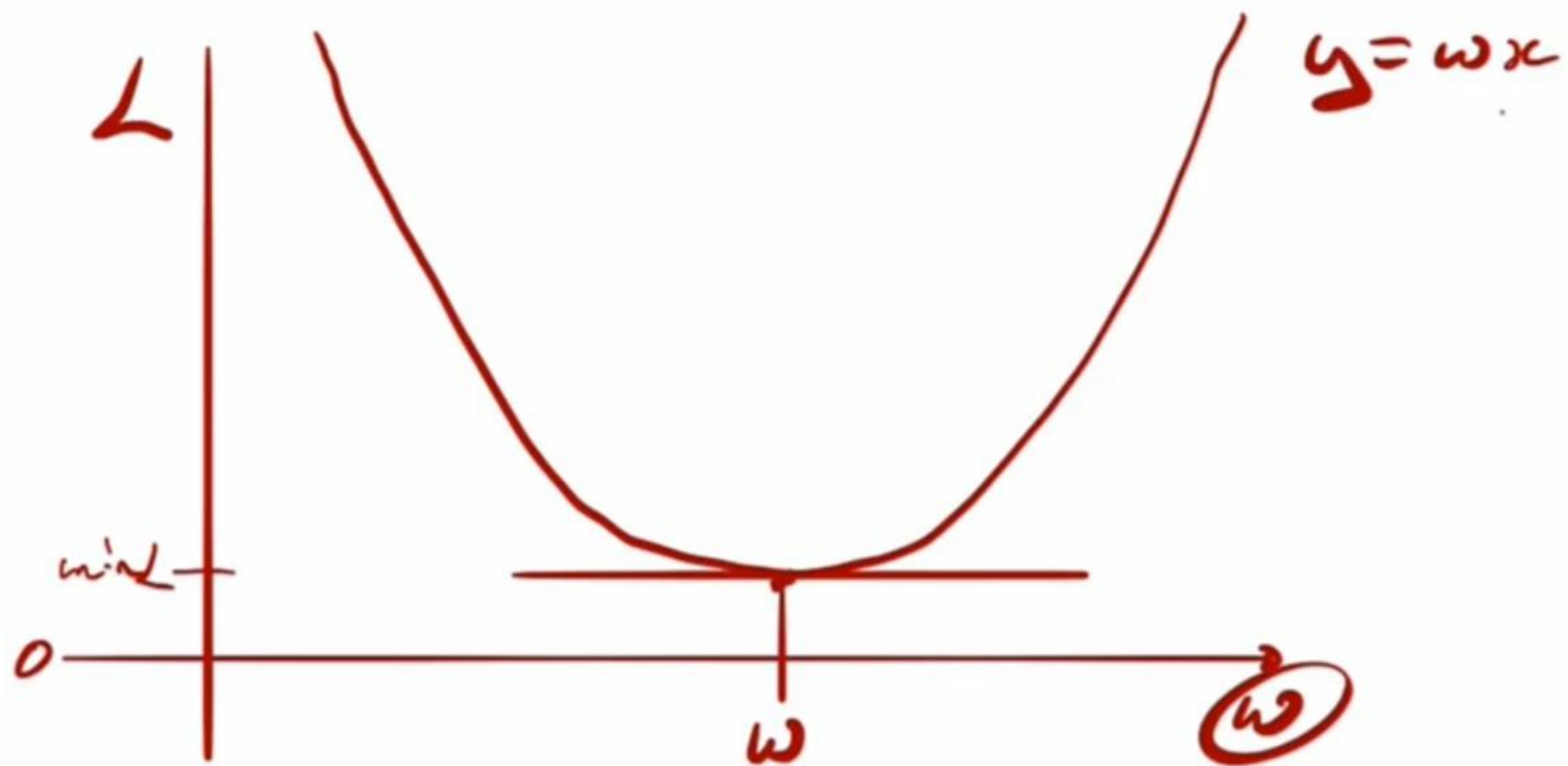
A LOOK AT SLOPE

SLOPE

$$Slope = \frac{rise}{run}$$

- Slope of a line (given 2 points on the line (x_1, y_1) & (x_2, y_2)):

$$Slope = \frac{y_2 - y_1}{x_2 - x_1}$$

SLOPE OF A CURVE

SLOPE OF A CURVE

- Calculus:
  - https://www.desmos.com/calculator/p0mxuhpx3e?lang=en

DERIVATIVE

- If we want to "optimise" something (find minimum value), we use calculus. We optimise the rate of change of L subject to the parameters:

$$L = \frac{1}{2m} \sum_{i=1}^{m} (y^i - w_1 x^i)^2$$

- So the standard method you may be familiar with is to take the derivative of the function with respect to the parameter of interest (w) and search for where the slope equals zero.

- The chain rule gives us the derivative of MSE. Remember we are minimising the **Loss** not the model.

$$\frac{dL}{dw} = \frac{1}{m} \sum_{i=1}^{m} (-x^i)(y^i - w_1 x^i)$$

GRADIENT DESCENT

- The above method is the way to go if you are dealing with small numbers of parameters and features

- E.g. simple linear regression has two parameters so relatively easy to solve.

$$y = w_0 + w_1 x$$

- But when we get into the tens of thousands of parameters it becomes computationally inefficient to do it this way.

- A very common method used is called *gradient descent* (or some variant of it).

- It is an iterative procedure that involves first order derivatives

$$\mathbf{w^{t+1}} = \mathbf{w^t} - \alpha \nabla L(\mathbf{w^t})$$

- where α is called the learning rate and ∇L is called the gradient of the function L.

$$\mathbf{w^t} = (w^{(t)}, w^{(t)}, w^{(t)}, \ldots)$$

GRADIENT DESCENT

- Sketch of the Procedure:
  1. Initialise the weights w^0 to something (maybe at random, maybe to 0)
  2. Calculate the Loss function $L(w^0)$.
  3. Find the gradient of $L(w^0)$.
  4. Calculate $w^1 = w^0 - \alpha \nabla L(w^0)$
  5. Repeat the procedure for w^2 then w^3 etc.
  6. Stop when we have reached a minimum (the result stops changing (Convergence)).

- If we pick a different Loss function, the idea is the same.

- So what is a gradient?
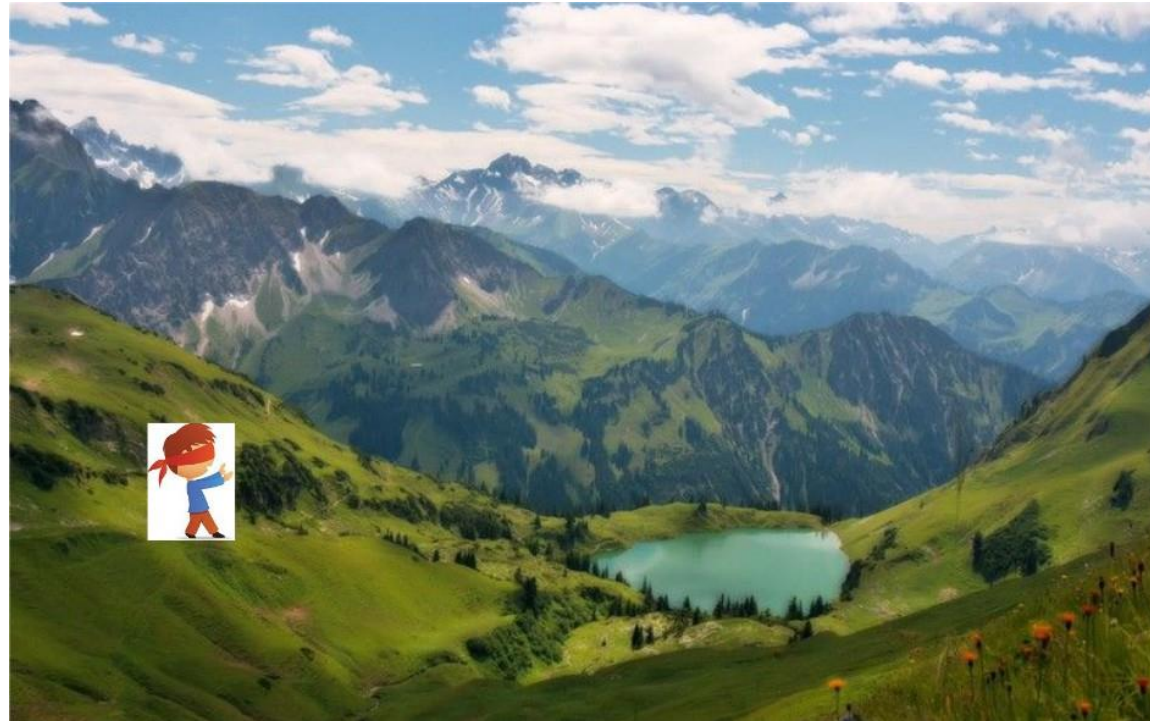
high weight

best fit, low error

low weight

α

ω^*

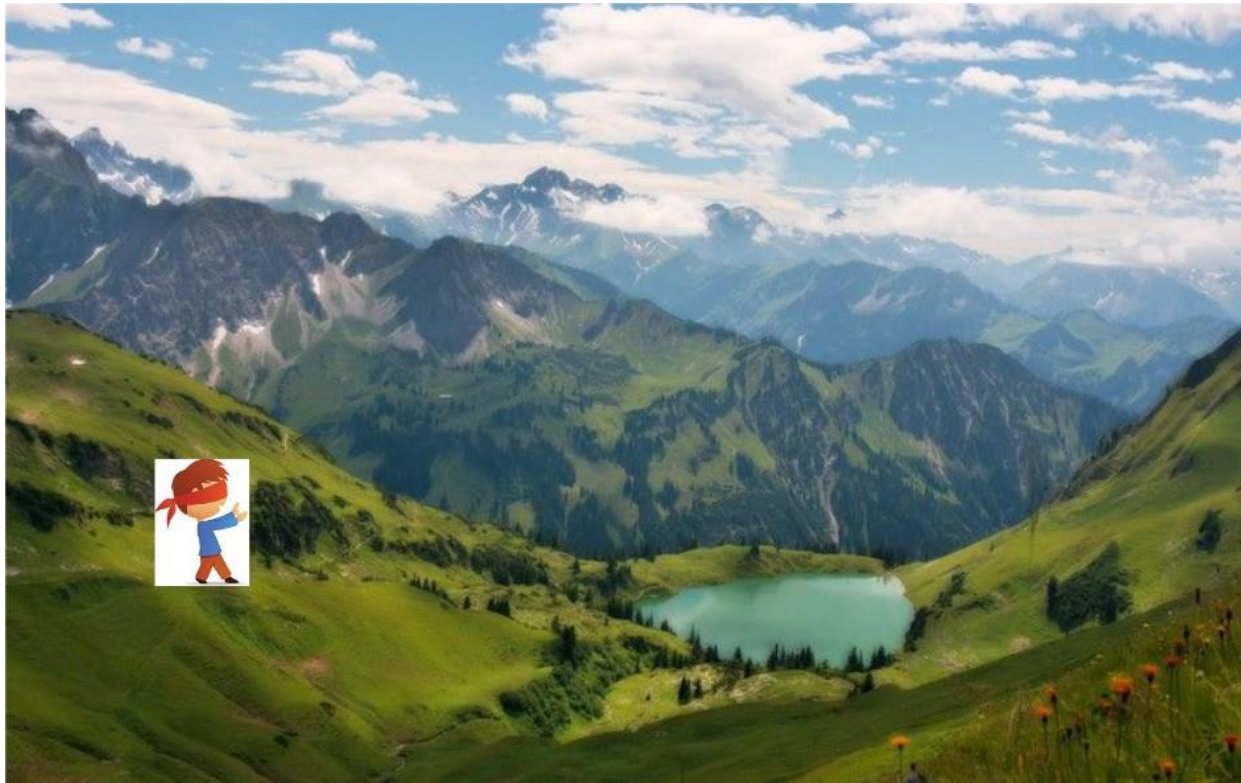ω
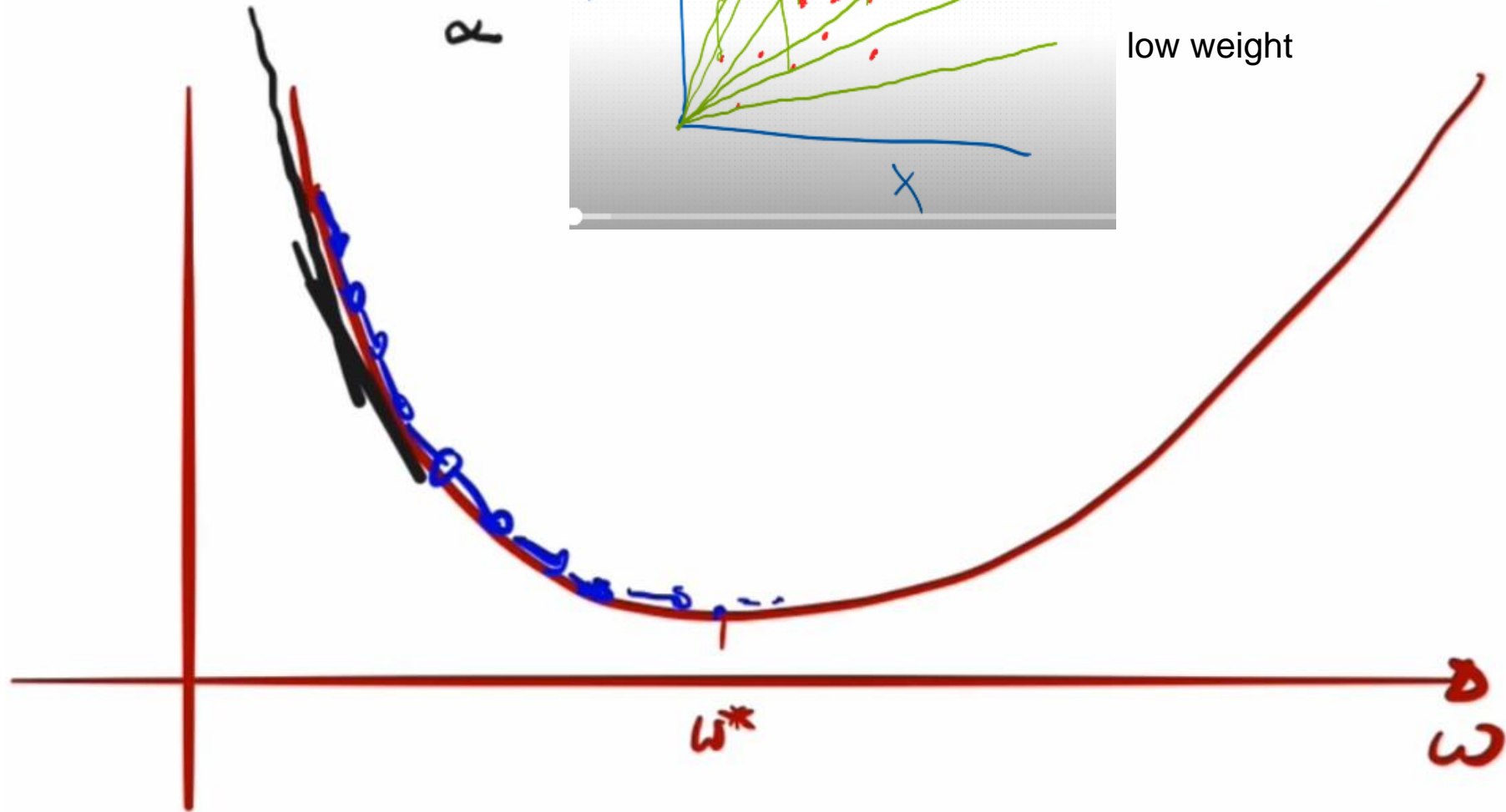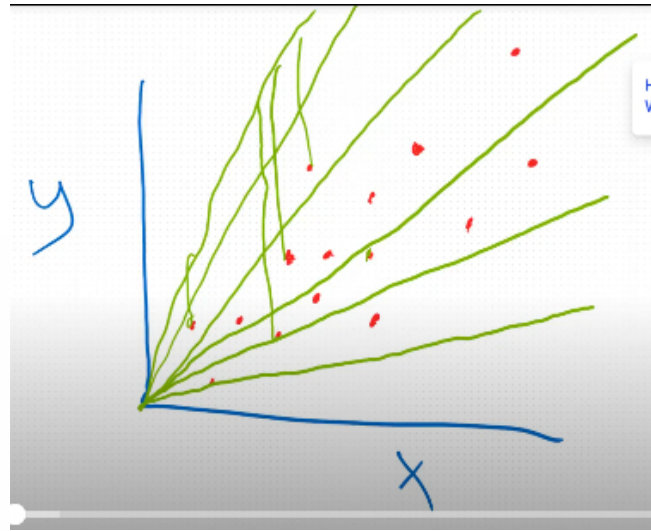
LOSS LANDSCAPE

LOSS LANDSCAPE

- Can tell what your loss is at any single point and trying to get to the bottom of the valley.

- Teleporting around randomly is probably not going to get the best answer.
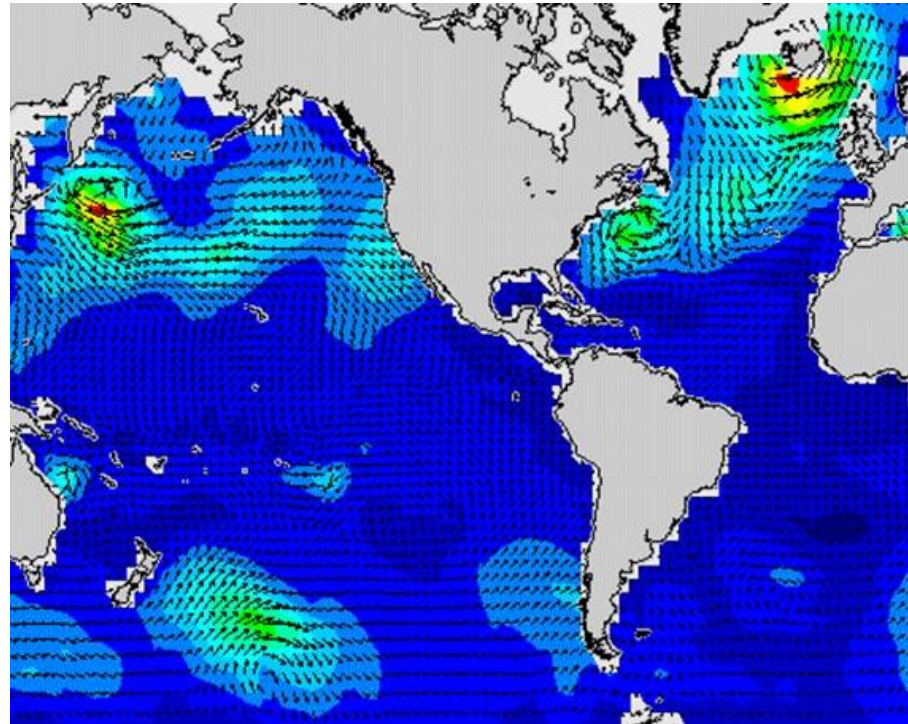
LOSS LANDSCAPE

- Compute the slope in every single direction. Then go downhill The Gradient

WAVES

- Global wave height map

- The colours (wave height) form a landscape

- Question: How can I find the biggest wave using only local information? Answer: Follow the gradient of the wave height surface.

WHAT IS A SLOPE?

- In multiple dimensions the gradient is the vector of "slopes" Our weights have multiple dimensions

- Say with (x , y), if we fix y and see how much x changes we can get the "slope" in the x direction, the amount x changes. And vice-versa.

- symbolised by $\dfrac{\partial L}{\partial w_0}$

  - is the partial derivative of the loss function with respect to w_0 - i.e. fix all the other weights and find out the slope in the w_0 direction, how much L changes in the w_0 direction.

- Remember: $\dfrac{dL}{dw} = \dfrac{1}{m} \sum_{i=1}^{m} (-x^i)(y^i - w_1 x^i)$
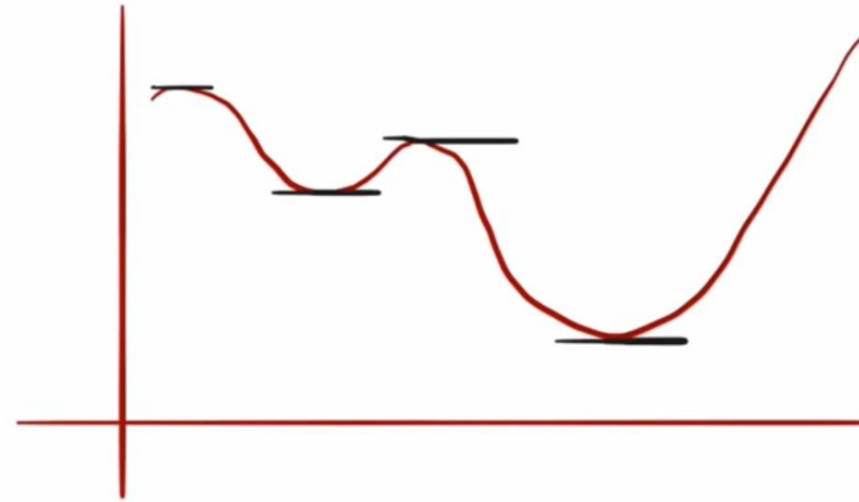
GRADIENT

- The gradient is the direction of greatest increase

- multiply it by -1 to get the direction of greatest decrease (hence $-\alpha \nabla L$ is the factor we adjust the weights in each step).

- Note: The gradient is the vector formed by taking all the *partial derivatives*.

$$\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_m}$$

POTENTIAL ISSUES - MULTIMODAL

- Of course there are going to be annoyances!

- We want the *global* minimum but we might get stuck in a *local* minimum.
- Let's assume it always works perfectly for it, because I've made it complicated enough as it is!

GRADIENT DESCENT VISUALISED
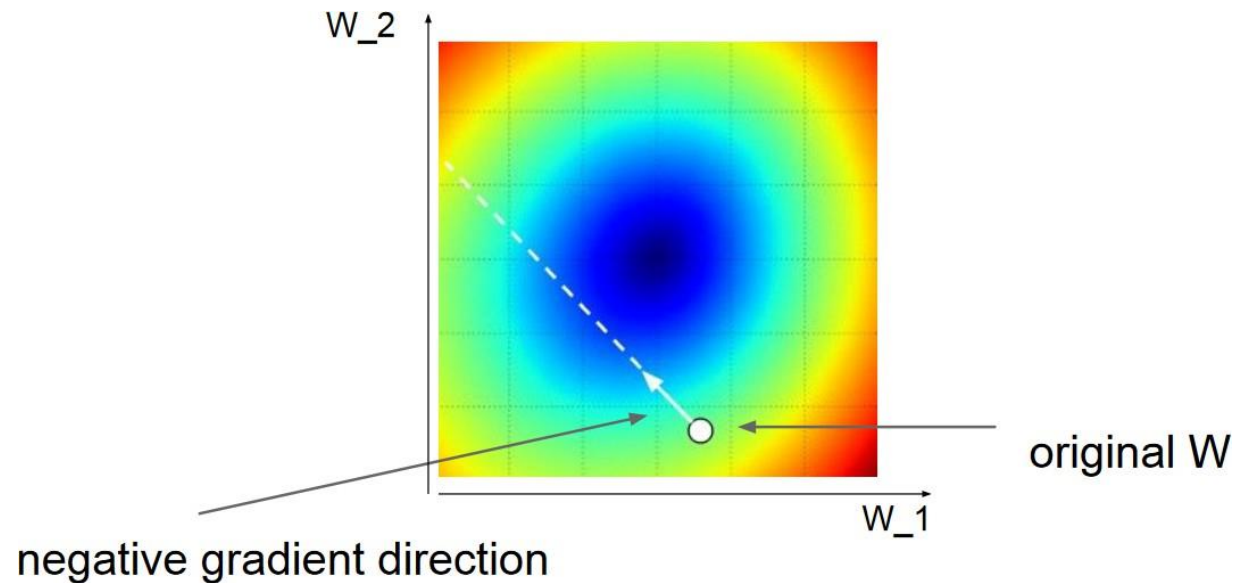
```python
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```
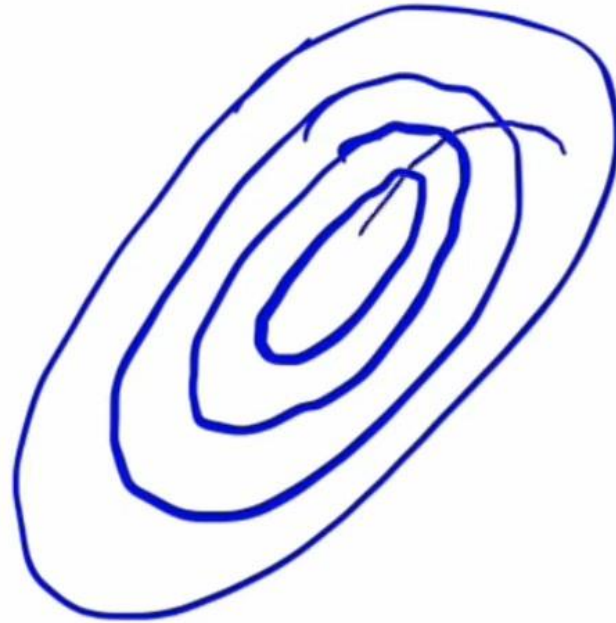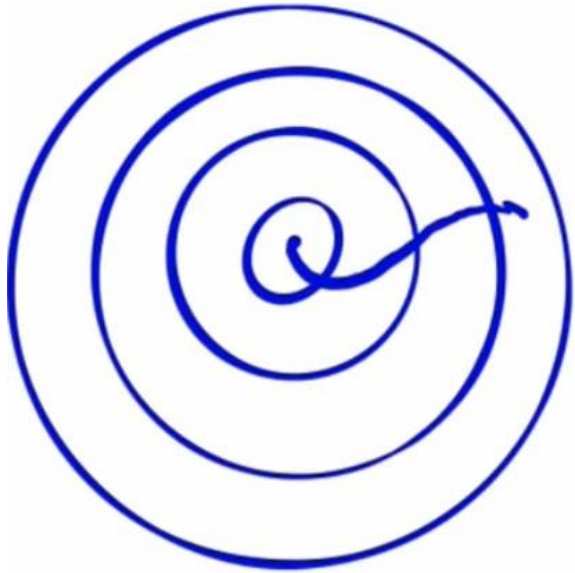
W_2

original W

W_1

negative gradient direction

SIMULTANEOUS UPDATE

- When we move from $\mathbf{w}^t \to \mathbf{w}^{t+1}$, we ensure all elements of the weight vector are updated simultaneously.

EXAMPLE OF LOTS OF WEIGHTS

- Multi-variate Regression. Each x has multiple features so let's say

$$\mathbf{x} = (1, x_1, x_2, x_3, x_4)$$

- and so for a particular sample

$$\mathbf{x^{(i)}} = (1, x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, x_4^{(i)})$$

- then

$$f_w(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 = \mathbf{w^T x}$$

- where $\mathbf{w^T}$ means the transpose of that vector. It's calculating a dot product.
- So we want to find the vector

$$\mathbf{w} = (w_0, w_1, w_2, w_3, w_4)$$

- that minimises the loss function

$$L = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - f_w(x^{(i)}))^2$$

GRADIENT DESCENT ON MULTIPLE VARIABLES

- $\mathbf{w} = (w_0, w_1, w_2, w_3, w_4)$. At each step we update $w_0, w_1, w_2, w_3, w_4,$

$$\mathbf{w^{t+1}} = \mathbf{w^t} - \alpha \nabla L(\mathbf{w^t})$$

- describes this. However, some people like to say we update each w_j at each step

- and
$$w_0 \leftarrow w_0 - \alpha \frac{\partial L}{\partial w_0}$$

$$w_1 \leftarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$
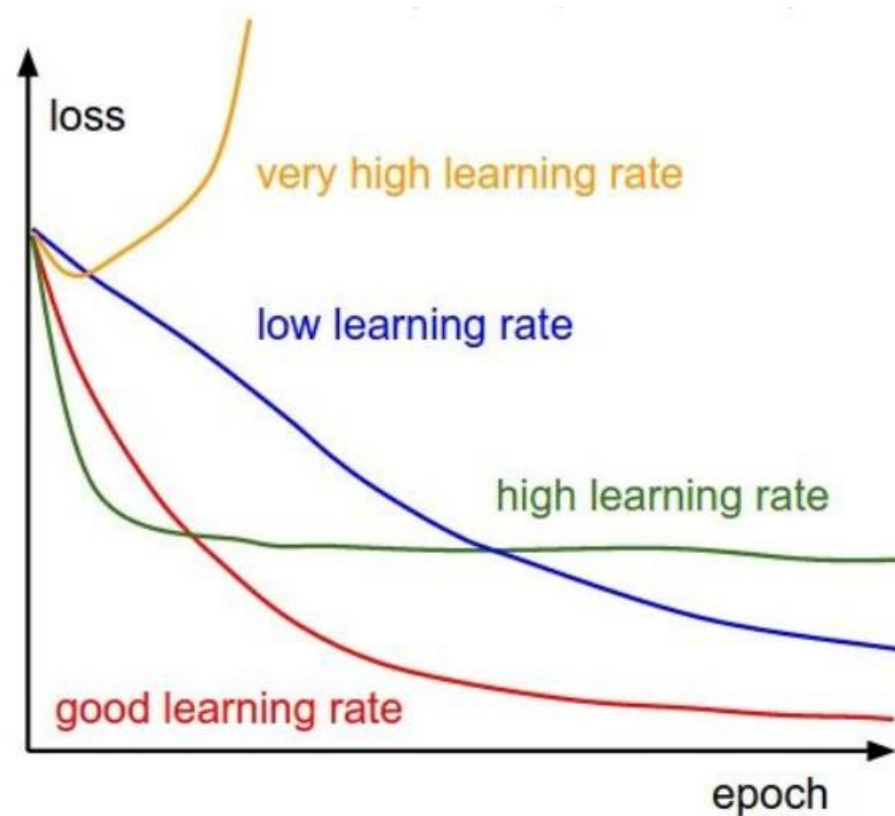
- etc.

- The overall equation then becomes:

$$w_j \leftarrow w_j - \alpha \frac{1}{m} \sum_{i=0}^{m} (x_j^{(i)})(f_w(x^{(i)}) - y^{(i)})$$

LEARNING RATE

- So this could be another hyper-parameter we may need to pick!

LOSS FUNCTION FOR LOGISTIC REGRESSION

- Recall, we use the following as our Loss Function for logistic regression (log loss)

$$L(\boldsymbol{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m} -y_i * \log\big(f_{\boldsymbol{w}}(x_i)\big) - (1 - y_i)\log(1 - f_{\boldsymbol{w}}(x_i))\right]$$

- The function is convex which greatly simplifies minimisation. We need to find the optimum parameters **w** that minimises the loss on the training set. Then we can use the model to predict new outputs on unseen data.

- Use Gradient Descent to minimise the Loss.

TYPES OF GRADIENT DESCENT

- Gradient descent can vary in terms of the number of training patterns used to calculate error; that is in turn used to update the model.

- The number of patterns used to calculate the error includes how stable the gradient is that is used to update the model.

- We will see that there is a tension in gradient descent configurations of computational efficiency and the fidelity of the error gradient.

- Variations:
  - Stochastic Gradient Descent (SGD)
  - Mini-Batch Gradient Descent

BATCH GRADIENT DESCENT

- What I've described above is known as Batch Gradient Descent. This requires:
  - Taking into account every sample in the training set to calculate a single loss and then do an update.

- Remember a sample is a single row of data. It contains inputs that are fed into the algorithm and we calculate the error for that sample by comparing the prediction to what we know is the right answer.

- For Batch Gradient Descent, the "size" of the batch is the size of the training set, the number of samples we have.

STOCHASTIC GRADIENT DESCENT

- In SGD we choose random observations.

- If we do an update after each randomly chosen sample, then the order of the samples will have a large effect on loss function and it will be a random process.

- It is called stochastic because samples are selected randomly (or shuffled) instead of as a single group, or in the order they appear in the training set

- An *Epoch* is when all samples in the training set have been used

- SGD has batch size = 1

MINI-BATCH GRADIENT DESCENT

- In practise usually an approach called Mini batch gradient descent is used. We don't want to use the full set as a batch, and also we don't want to do a sample of size 1.

- The Batch size is a hyper parameter that defines the number of samples to work through before updating the weights.
  - It uses random samples but in batches and updates after each "batch"
  - We do not calculate the gradients for each observation but for a group of observations which results in faster optimization.
  - Due to the order of the batches and the random nature of the samples chosen means the loss can have variance and is dependent on when in training that batch is chosen.
  - It should be referred to as Mini-Batch SGD but sometimes people just say SGD as a catch all for SGD and mini-batch SGD
  - Again an Epoch is when all samples in the training set have been used

OPTIMISATION METHODS

- There are lots of techniques to implement SGD and some are listed below. Some keep a consistent training rate, some change the learning rate after so many Epochs.

- SGD

- SGD with momentum

- Nesterov Momentum

- AdaGrad

- RMSProp

- Adam (Default in Sklearn's solver)

- https://www.kdnuggets.com/2019/06/gradient-descent-algorithms-cheat-sheet.html

EPOCHS

- Definition: The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

- An epoch is comprised of one or more batches.

- You can think of a for-loop over the number of epochs, where each loop proceeds over the training dataset.

- Within this for-loop, is another nested for-loop that iterates over each batch of samples, where one batch has the specified "batch size" number of samples.

- It is common to create line plots that show epochs along the x-axis as time and the error or skill of the model on the y-axis. These plots are sometimes called learning curves. These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training dataset

LEARNING RATE

- So this could be another hyper-parameter we may need to pick!