# 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!)

$$y = w_0 + w_1 x$$

residual is the error between the predicted value and the actual value for a particular sample

- 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.

mean squared error is the kind of sum of the square of those residuals, the sum of the difference between the predicted value and the actual value breast cancer tumor: what we really want from our model is like how many patients live or how many patients die or how much, how well our diagnosis performs. it should be detecting and essentially giving us good performance on the true positives and the two negatives.But that isn't the information that we use to ultimately train the model. we can't really necessarily optimise those numbers. We need to pick an error metric, the likes of mean squared error or the log loss that we saw with logistic regression that we can actually do some calculus on.

## RECAP

calculus to calculate the slope or we can actually calculate the slope and tune our weights to optimise the performance

from trained mocel

- 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

So this is known as the L2 norm in Euclidean geometry.You can also have an L3 norm which is cubed, an L1 norm which just takes the absolute value and it goes off. You can go L4 norm and so on.But the L2 norm is by far and away not only machine learning, but in all sorts of data science where you're doing distance metrics, you know, and comparing anything.The L2 norm will be the most common, but they all amount to the same thing.It's just measuring the difference, the distance between the Y predict and what's actually there.If we minimise this, then we manage to minimise our error loss because this distance , if it's zero is essentially what we want is that there's no difference across all samples from I going up from 1 all the way up to M.

## 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). =fitness 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

$$y = w_1 x$$ a line where the Y intercepted 0 at the origin.

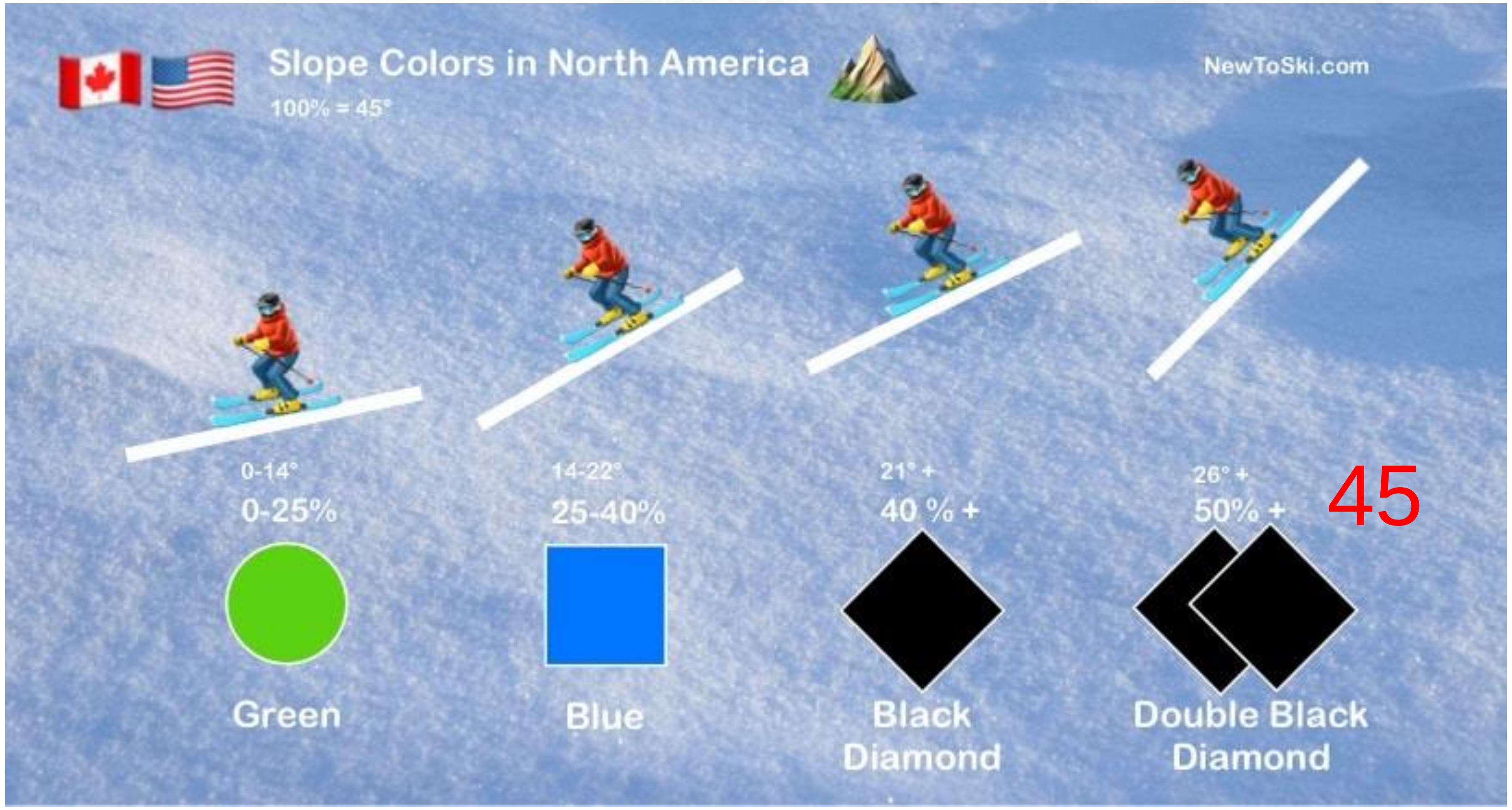
- So that the mean squared error (loss) is:

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

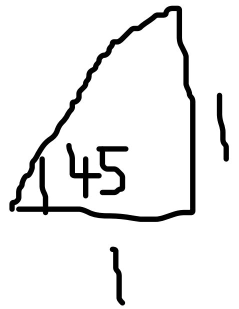
- To train the parameters - we change $w_1$ until the loss function is at its minimum. So how do we find the minimum? this is where the calculus comes in

Calculus, I think is the union of differentiation and integration. 'm more interested in is differentiation or finding the derivative is the term that's often used to do a differentiation.

## A LOOK AT SLOPE



# SLOPE



50% rise: for every one kilometre you drive, you've gone up 500 metres. So that is 500 / 1000, which is 1 kilometre is 1 / 2 or a slope of nought .5, sometimes expressed as 50% and say a 10% gradient.

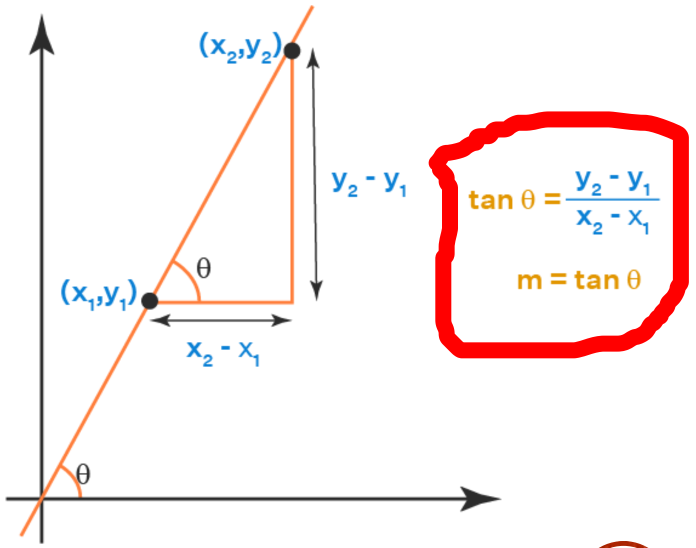
10% gradient, it'd be a gradient of naught .1, which would be for every kilometre that you're run, you go up 100 metres.

100% slope:If you had gone up four and you'd also ran four, 4 / 4 again would give you a slope of one or 100%, which means that you're going up at 45°.

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

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

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

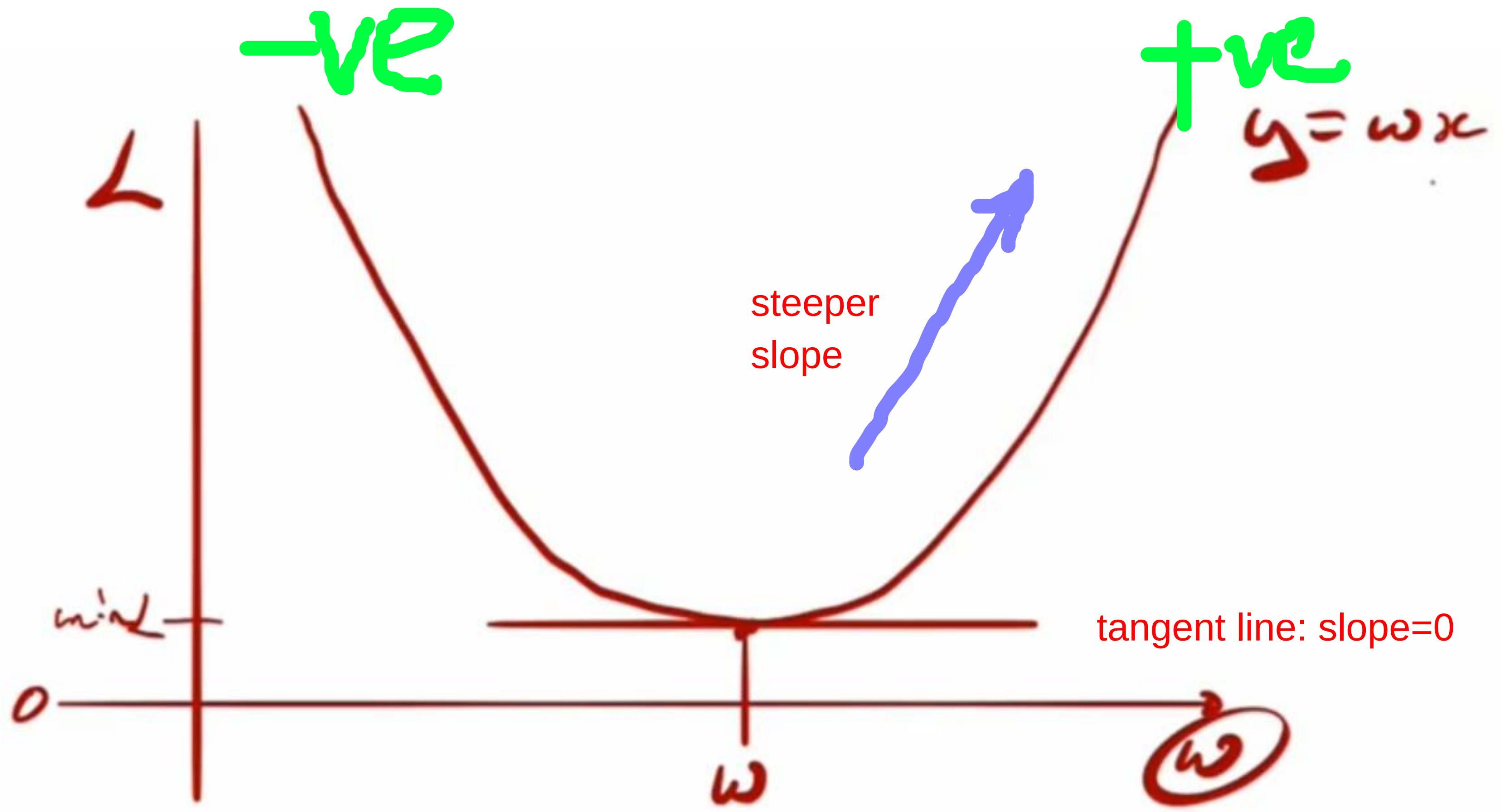


the slope of a curve is always changing.

Curve generated by the function Mean squared error.

# SLOPE OF A CURVE

if the slope is increasing, you're going to get a positive number.
But if the slope is decreasing, this number right here is going to be a negative number.
If you take Y 2 -, y one and Y one is bigger than Y2, then Y 2 -, y one is going to be a negative number.So a line sloping down the way is going to give you a negative slope -50% or something like that -2 or -1



optimum/minimum
gradient 0

## SLOPE OF A CURVE

calculus all about is finding the slope of the line.
calculus is all about finding the slope of not just lines, but of complex shapes, complex curves like cubic equations, you know, squared equations, different polynomials essentially.
Also trigonometric equations like Cos, sine, log, and so on.
What you're trying to do is find the slope of those curves at different points.

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

As our weights are changed, our loss function is going to get better and better and better up until to a point where we've got the optimal weights, we're picking a point that's minimising the loss.And as we keep changing it in that direction this way, the West gets bigger and bigger and bigger, our loss starts to get bigger again.So there's going to be this happy point of the curve known as the optimum or the minimum, which is where we're going to get our best value for our weight.That gives us the best performance in the model, the best performance in our linear regression, the best accuracy between our predicted values and our actual values.

his curve was rather represented by the mean squared error.As we put in values of X into our mean squared error equation, our loss, the output of the mean squared error will get smaller and smaller and smaller.And as we keep varying those terms that are mean squared error, they would start to get bigger and bigger and bigger.So we get a similar sort of U-shaped curve, but it's not X ^2, it's something a bit more complex. So let's look at taking the derivative.Instead of this being a simple derivative X ^2 with the power rule being 2X, let's look at taking the derivative of mean squared error.And to do that, I'm going to have to use the chain rule.
But this should be an insight into taking the derivative allows you that at any point on the curve, you know you put in any value of X.
We're looking for the value of X that will give us a slope of 0.

we're trying to find what is the slope of this curve with respect to changing the weights W.

## 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.

derivative of L with respect to w.

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

we're not trying to derive, you know, what's the slope at any given point of X ^2.We're trying to find what is the slope, at any given point of this equation right here. So what we need to do is do the derivative and that will give us our answer.So remember, we're minimising the loss that's we're trying to minimise this mean squared error metric here.So to derive it, the chain rule, basically what it does is if you have something inside brackets to a power, what it says is everything inside the brackets you treat as a single term and you just treat it as it will be a single term.

Hopefully we get a little smaller and smaller slope, and this is the foundation of gradient descent, changing W, varying the slope, searching for where our slope is 0, and hoping that that is where our answer is.

# 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

1st order, that just means calculating the slope of a function at some given point.

t is time step

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

- where $\alpha$ is called the learning rate and $\nabla L$ is called the gradient of the function $L$.

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

what we're doing essentially is crawling down the curve really, really slowly.We take what the weight is, modify it by a tiny bit, get a new set of weights, this weight then is our new set of weights, change it by a tiny bit, get the new set of weights,new set of weights, whatever it is, change by a tiny bit.

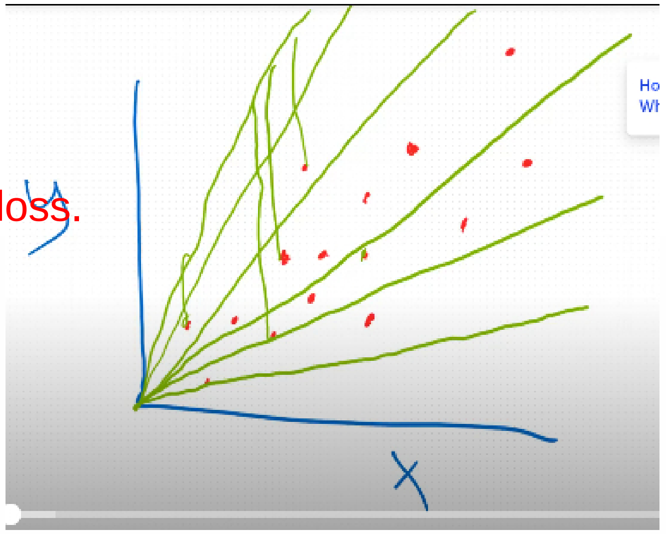
## GRADIENT DESCENT

- Sketch of the Procedure:
  1. Initialise the weights $\boldsymbol{w^0}$ to something (maybe at random, maybe to 0)
  2. Calculate the Loss function $L(\boldsymbol{w^0})$.
  3. Find the gradient of $L(\boldsymbol{w^0})$.
  4. Calculate $\boldsymbol{w^1} = \boldsymbol{w^0} - \alpha \nabla L(\boldsymbol{w^0})$
  5. Repeat the procedure for $\boldsymbol{w^2}$ then $\boldsymbol{w^3}$ 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?

The reason the slope is here is that as the slope is higher, we can take bigger jumps because typically the way the landscape is is that as the slope is higher, we can go take bigger jumps.We don't have to be like tiny little steps every time.If the slope is high, we're probably pretty far away from where the slope is 0. But the as a slope gets smaller and smaller, this term also gets smaller and smaller.So it's like a scaling term, but this is getting smaller and smaller increments all the time.Where as we get towards the solution, we're just like edge in our ways, very slightly adjusted on the way it's defined the absolute best solution that we can.So alpha is a learning rate, just a scaling term that we can change again, it's another hyper parameter that we have to optimise.
even though you see the gradient of both separately, what you do is you update both of them together.

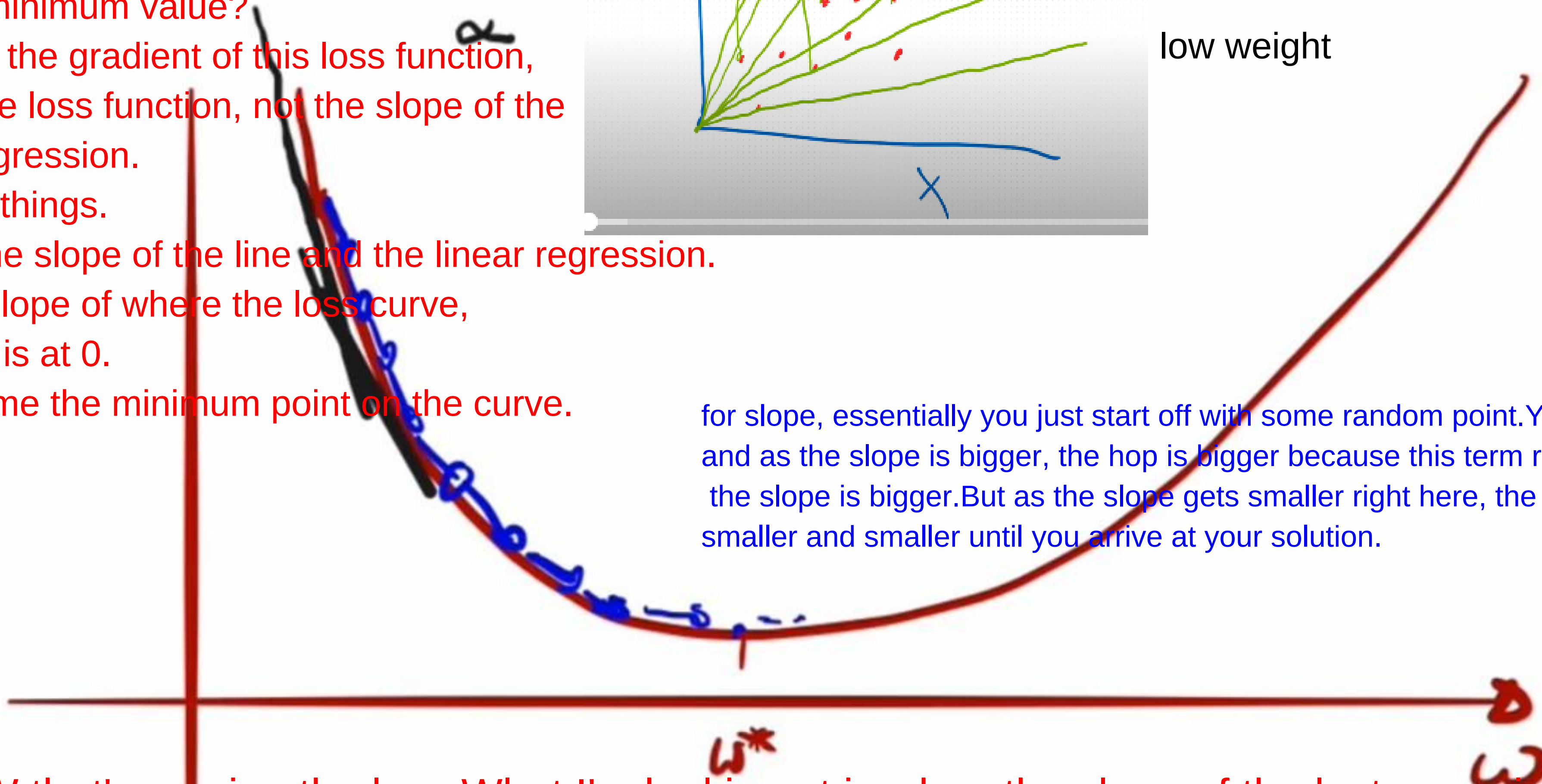
OK, what are we trying to do here?
We're trying to minimise the loss.
How do we minimise the loss?
We vary the weights and that will have an impact on the loss.
How do we find the minimum value?
We search for where the gradient of this loss function,
that is the slope of the loss function, not the slope of the
line and the linear regression.
They're two different things.
I was talking about the slope of the line and the linear regression.
But this,Finding the slope of where the loss curve,
This curve right here is at 0.
And that would give me the minimum point on the curve.

high weight



best fit, low error

low weight



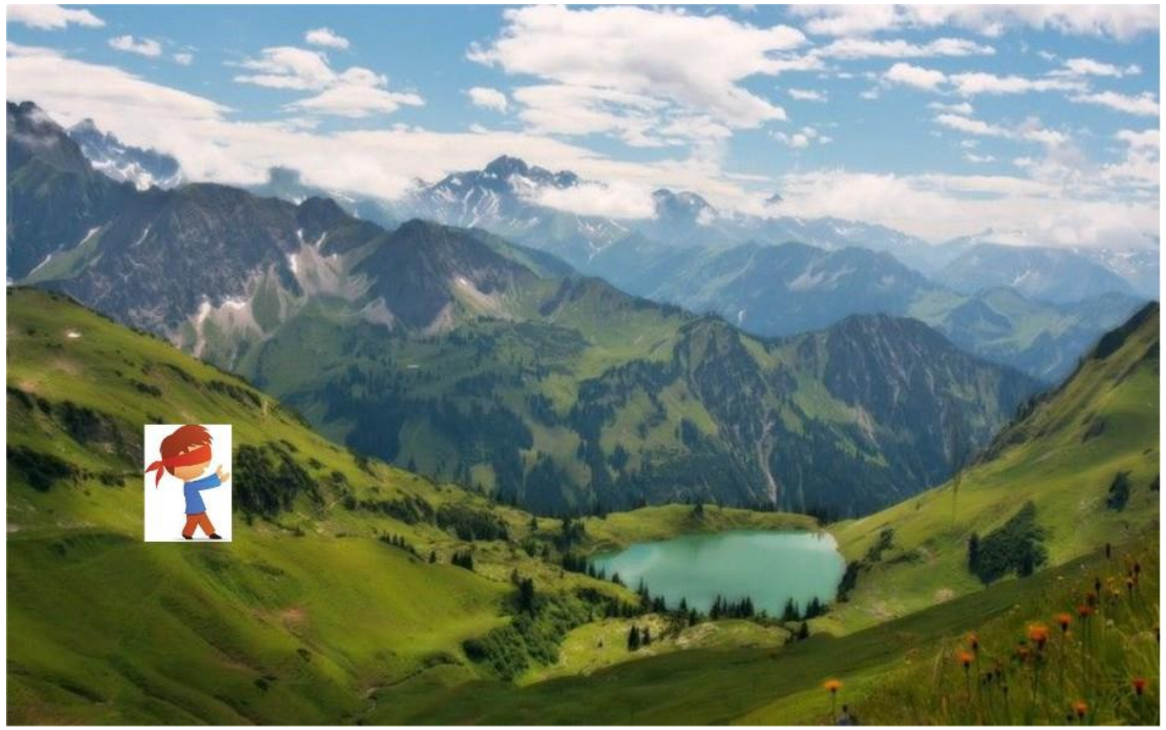
for slope, essentially you just start off with some random point.You make these hops, and as the slope is bigger, the hop is bigger because this term right here is bigger, the slope is bigger.But as the slope gets smaller right here, the hops get smaller and smaller and smaller until you arrive at your solution.

by adjusting my W that's varying the loss.What I'm looking at is when the slope of the last curve is 0 and the loss I will be able to find when the last curve is at its minimum.That is when I have the best solution, the lowest number of errors between the residuals or the lowest residual between the Y predict and the Y actual values.

## LOSS LANDSCAPE

## LOSS LANDSCAPE

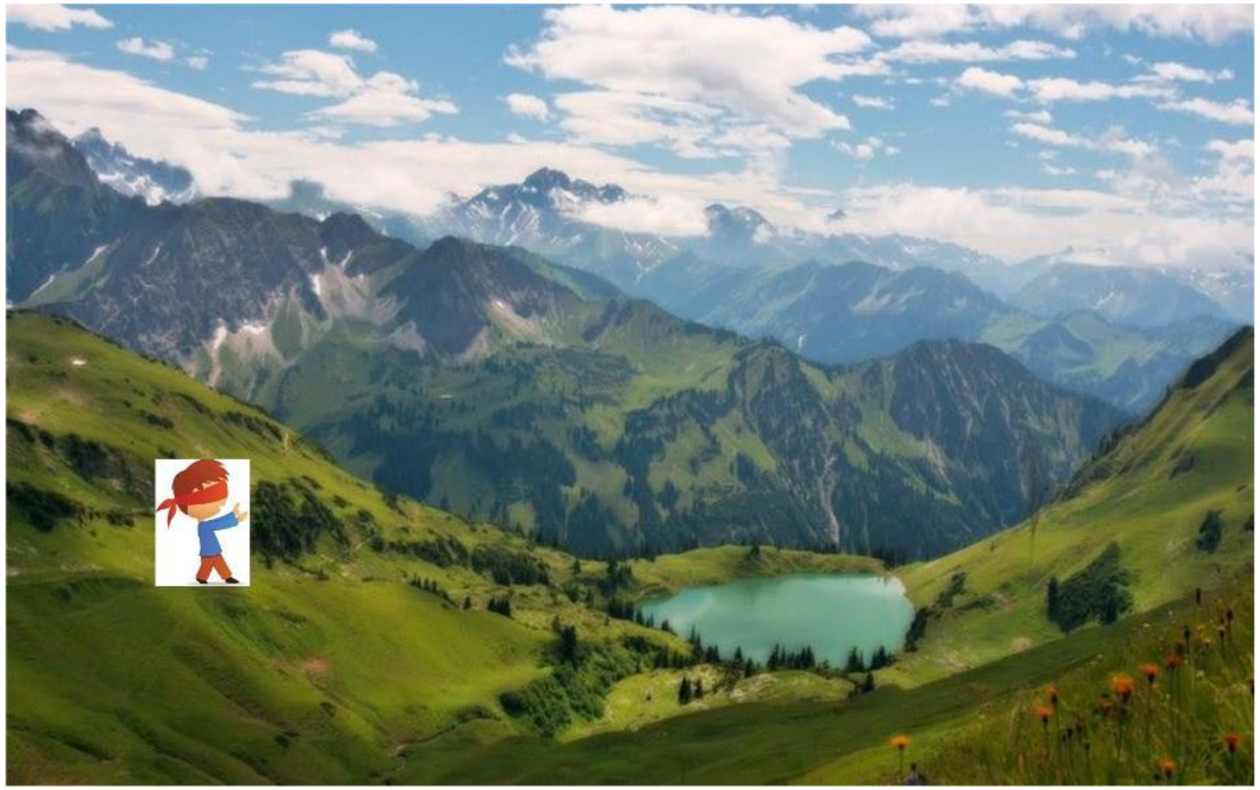
- Can tell what your loss is at any single point and trying to get to the bottom of the valley.
- Teleporting around randomly is probably not going to get the best answer.



## LOSS LANDSCAPE

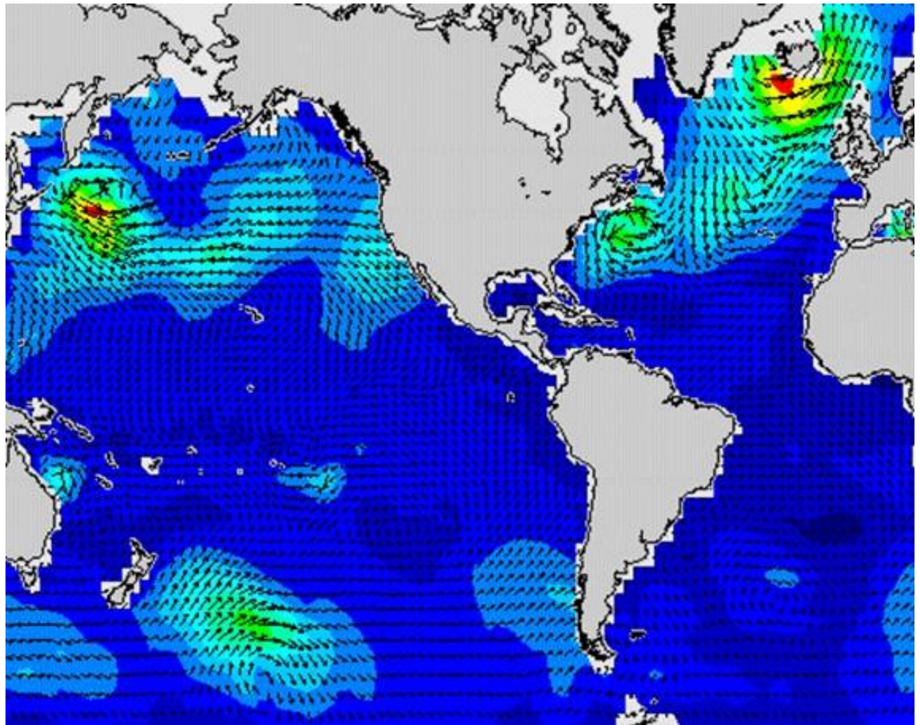
we want the GLOBAL MINIMA, not local minima

- 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.



So the loss function is the loss is actually a function of all the different weights. \having weight 0 and TuneIn weight 1 and TuneIn weight 2, they all all have an impact on the loss function.But a partial derivative will tell us what is the slope with just respect to changing weight 0.Another partial derivative is what is the slope if we're just with respect to changing weight 1.

## 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 $\frac{\partial L}{\partial w_0}$
  - 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: $\frac{dL}{dw} = \frac{1}{m}\sum_{i=1}^{m}(-x^i)(y^i - w_1 x^i)$

chain rule of mean square error

## GRADIENT

- The gradient is the direction of greatest increase the bigger the gradient, the bigger the jump we get to get.
- 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). -ve bc we want to find minima
- Note: The gradient is the vector formed by taking all the *partial derivatives*.

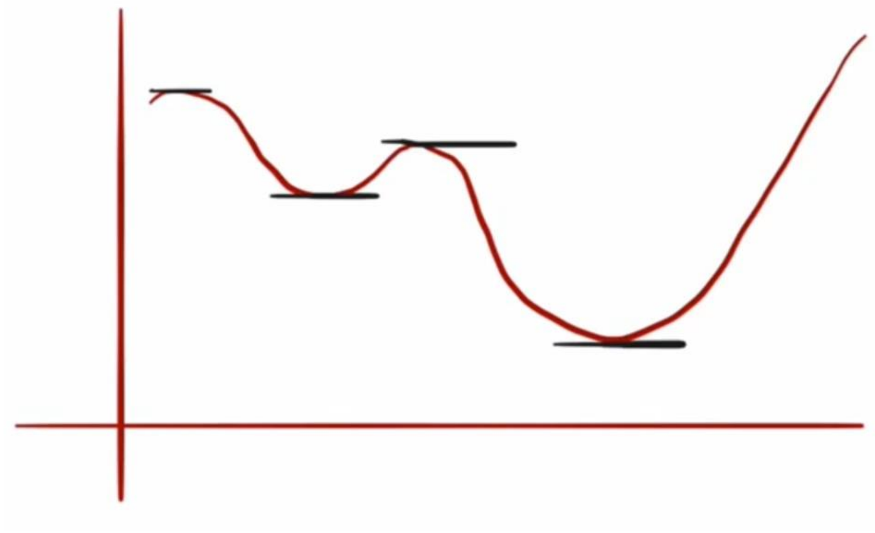
$$\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_m}$$

So the gradients then is the vector by all the different partial derivatives.So what you do is you work out the slope for each parameter.
So what happens if I change W 0?What's my slope?What happens if I just change W one and leave the rest of them all fixed?
What happens if I change W2 and so on.And this slope will tell us whether we need to, you know, increase W 0 or decrease W 0, increase W one or decrease W 1.And the overall step that we take is by moving all of these things, moving all of the weights all at the same time in one single kind of progression onto our next set of weights at the next time step, at weight t + 1, at time step t + 1.

## POTENTIAL ISSUES - MULTIMODAL

what multimodal means is that the landscape, fitness landscape or that last landscape or whatever it is isn't a nice U-shaped curve

- Of course there are going to be annoyances!



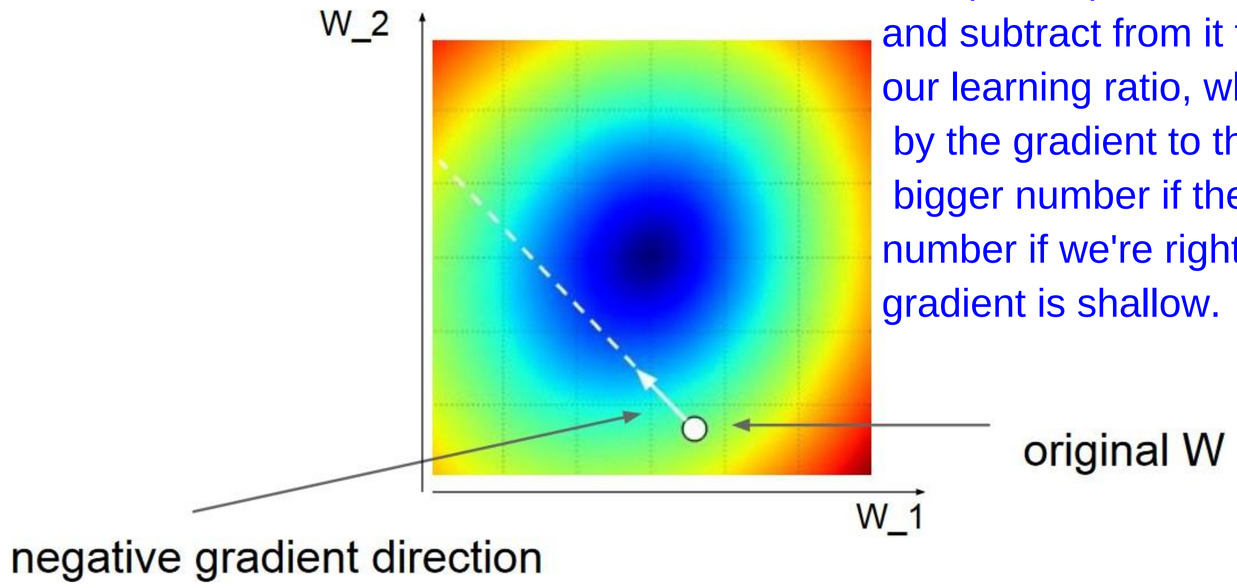
- We want the *global* minimum but we might get stuck in a *local* minimum.
- Let's assume it always works perfectly for it, because I've made it complicated enough as it is!

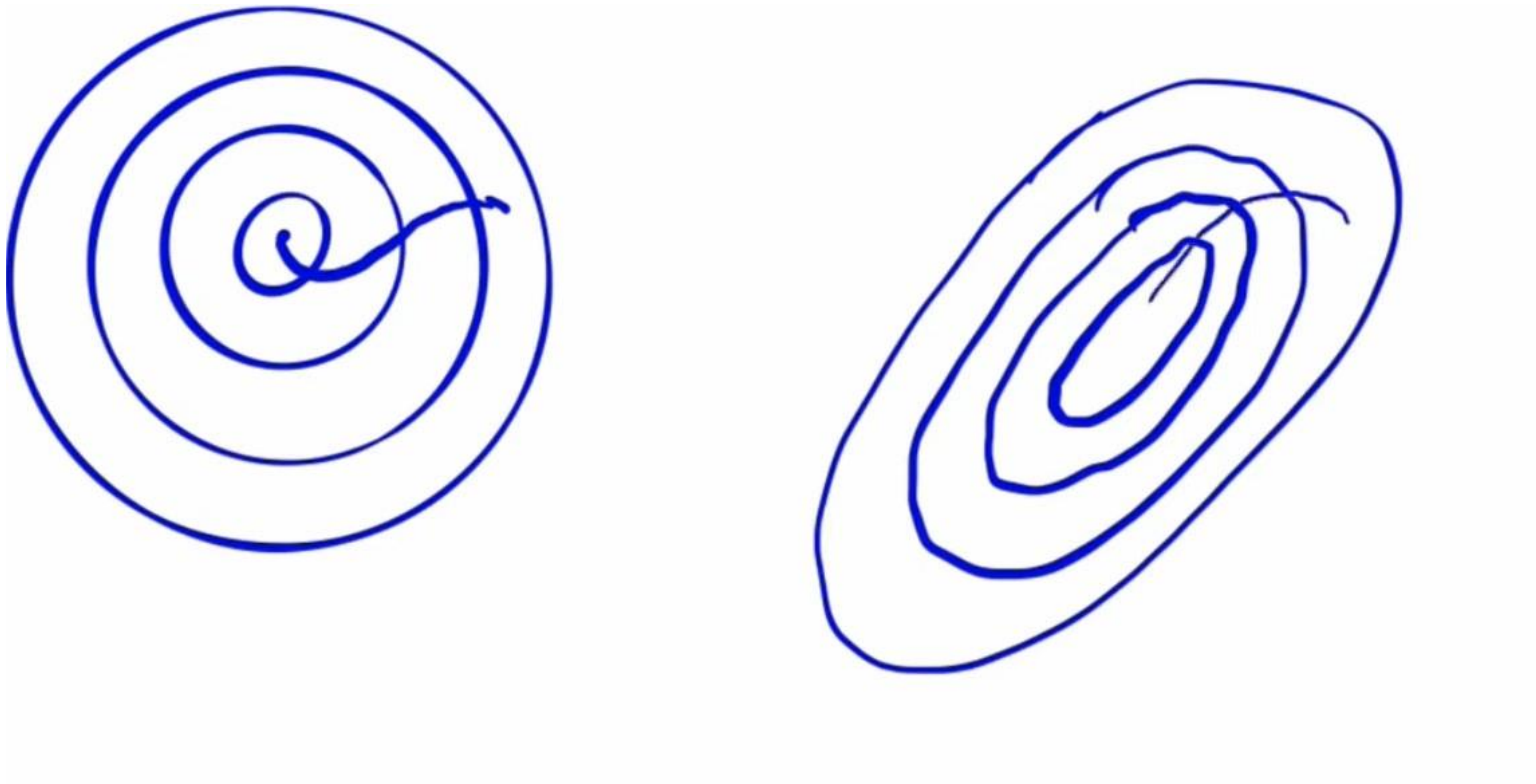
# GRADIENT DESCENT VISUALISED

```
# Vanilla Gradient Descent

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

first of all find our weights gradient, which is evaluating the gradient to the function when you're passing in the last function, passing in your data and your weights. For every single weight, it'll give you the gradient of that steps at that particular weight value. Then what we do is we figure out our new weights is a function of the step size where we take our old weights. This plus equals means take the old value of weights and subtract from it the step size, which is our alpha, our learning ratio, whatever it is, how big we're stepping by the gradient to the weights.And this is going to be a bigger number if the gradient is steep, a shallower number if we're right close to the solution where our gradient is shallow.





## SIMULTANEOUS UPDATE

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

even though we sense the gradient separately to our partial derivatives, we do a simultaneous updates. We update all the weights in one go rather than just updating the weights one at a time.
It just wouldn't make sense.So when we move from West, our weights at time step T to our weights at time step t + 1, say from one to two or two to three, we ensure that all elements of the weight vector are updated simultaneously.Now the way we do that is we have to calculate all the derivatives partially and separately, all our slopes, but then we actually make the updates all at a time before we evaluate again.

# 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

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

I've always bolded where W or X is actually a vector, where it's not just a single value, it's actually a vector of values or a matrix of values.

- then

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

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

we transpose it in order for us to be able to do a matrix multiply.

- So we want to find the vector

we want to find the West that minimises the loss function L.

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

a dot product.is essentially a matrix multiplication of of of two of two matrices or two vectors.

- that minimises the loss function

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

The loss is the same as what we've seen with mean squared error, our observed value or the data in our data set minus our predicted value, where our predicted value is weights multiplied by our features and then we're taking the squared value of all these residuals, this error across every single data point in our training set.

each step we update our weights, our W at t + 1 time plus one is equal to our all weights multiplied by the gradient and our learning rate, that's our little step size by the gradient of our weights.Remember for the gradient of the weights that we have to find the gradient of each of the weights separately.That's the partial derivative.

# GRADIENT DESCENT ON MULTIPLE VARIABLES

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

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

w superscript t is time step

w subscript is particular weight

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

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

- and

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

- etc.

even though we update them all in one go simultaneously, we have to calculate how we update them as in the magnitude of the update.We have to calculate each of them separately according to the gradients for each weight because some of the weights will behave differently. Maybe 1st guess might be close to their optimal solution and we don't need to tune them at all. maybe the slope is really shallow (small jump), while others are bigger jumps. So that's why we do these partial derivatives and we calculate each of the weights by getting the derivative with respect to each of the weights separately, leaving everything else fixed.

- The overall equation then becomes:

We adjust our weight zero.Our weight 0 becomes the old weight 0 minus the learning rate by the derivative, the slope of what weight 0 is, and that would give us a new weight 0 for the next time step.Similarly, weight 1 is the old weight 1 minus the step size by the slope of weight one at that particular point.

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

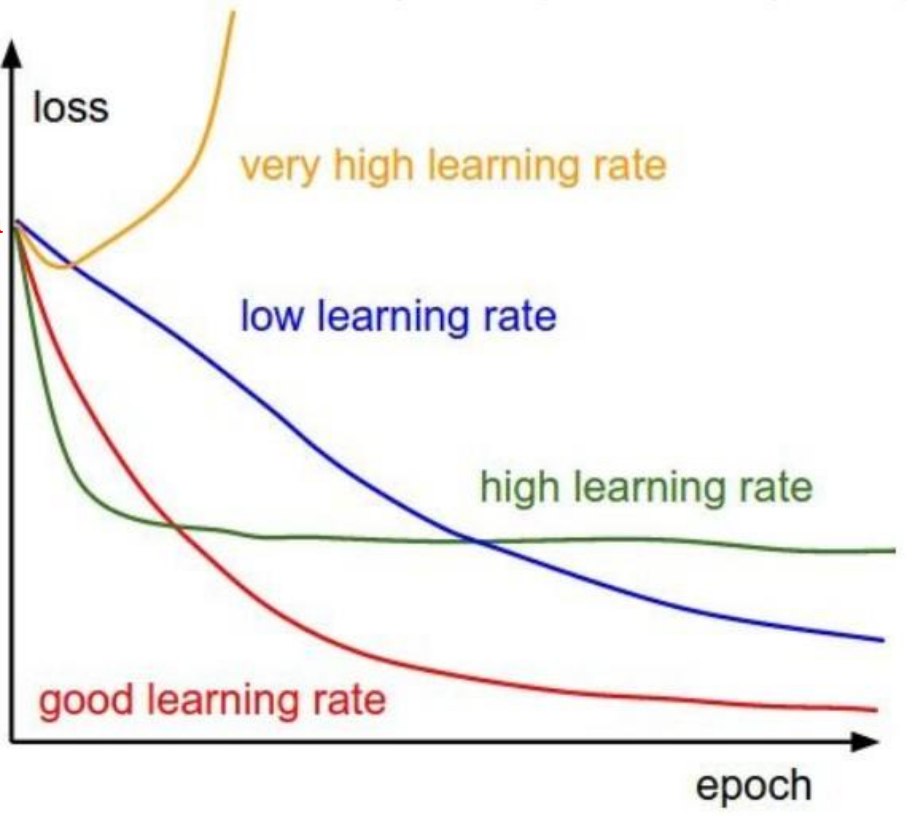
So this will give us, some of these slopes will be high, some of these slopes will be small, and even though all of the updates are calculated separately, the actual update itself is all applied simultaneously.The overall equation then to all of this gradient descent for a large or any number of variables really becomes this.

## LEARNING RATE

Our weight J at the next time step is whatever the weight J was at the previous time step minus
 the learning rate,And this derivative in the case of mean squared error (dL/dw) is i/m sigma....
Alpha learning rate, which is a parameter you just need to optimise.
Then we have to work out the the squared error over all the samples in our data set.
fx(xi) is our derivative.This has given us our slope now.
So Yi is the what's the actual value in the data set.
FW of XI, that's what the output of our model is given.(Y predict minus the Y actual value.)
xji  is the input that's multiplied by the WJ, which is the input value in our data set.

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

too high a learning rate, you know, we're jumping around too much and we'll actually get worse. Those jumps that we're making down the curve are just going to be like we're jumping over the Optima, we're just going to be like jumping over the valley and we're going to end up doing really badly.If we've got a very low learning race, it's just going to be really, really slow progress and we're just going to run out of time.High learning rate, you might get stuck in local optimal.A really good learning race for an algorithm will take you close to the Optima, like a sufficiently good Optima where you're not going to end up converging at a sub optimal solution.



## LOSS FUNCTION FOR LOGISTIC REGRESSION

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

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

- 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.

this function is convex, which means it's got a nice U shape, which means it's solvable using calculus, greatly simplified minimization where you're not likely to get caught in a local minima.So we can find the optimum parameters W that minimise the loss on the training set, and we can use the model to predict new outputs on unseen data.We want to predict outputs on unseen data.We can use gradient descent to minimise the loss.Everything stays exactly the same except for the derivative of this is different than the derivative of mean squared error.You're still going to have a chain rule and you're going to be, you know, driving this and everything is there is a derivative to be calculated of this.

# 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

if you use every single element in your training set, you can compute a really strong, robust, reliable gradient. But it takes ages. So just take a handful out of your training set, like calculate the the loss curve using that, and it just makes things a bit more efficient.

these ones are not computationally intensive as our BATCH GRADIENT DESCENT

fidelity that is like the quality of the error gradients that if you've only taken a handful of points, you might not get a really good representation of what the true gradient is of your loss landscape.You know you're going to be missing points in your loss landscape. You might be missing some important information, but you might benefit from the time that you can actually do more epochs and more iterations.

## 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.

So what batch gradient descent is is taking into account every sample in the training set so that you calculate a single loss.Then you do the update using all the techniques I showed you already, partial derivatives, working out what the gradient is for each weight, making the adjustments, doing it again for the next time step.But that every single step, if you're putting all the data through it and it's a big model, takes lots of time.

I is equal to 0 up to I is equal to M where M is the number of samples in our training set.

So we make updates based upon random observations rather than doing the whole thing if we do an up update after each randomly chosen sample.
So that's where we calculate the loss not as the summation of the loss over all elements of our data set, but just over a single randomly chosen point.
How different is the predicted value from the real value for that?Only that single point and making changes just based upon that, based upon the based upon the gradient at that single point.If we do an update after each randomly chosen sample, then the order of the samples will have a large effect on the loss function.

## STOCHASTIC GRADIENT DESCENT

It means that if you pick, a sample that's an oddball sample to begin with, you might be making changes that could be going towards overtraining territory
So it's going to be, susceptible to noise.

- 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 , where we are just picking 1 sample at a time

So what you do is you take a sample, make an update to all the weights, calculate the gradient.At that point, you know how you change the weights, what impact that has on the performance for that sample rather than for all the samples, the mean, the mean squared error across all the samples.So you take the sample, check out the gradient impacts, the performance on that predictive, the predictive power on that single sample.Change all your weights, pick another sample, do the same thing again, tune all the weights, pick another sample, see how good it is, Calculate the gradients for all the weights, tune all the weights, pick another sample.

it's susceptible to noise.Yes, pick a batch sample.You end up your weights going in the wrong direction.What's much better is this mini batch gradient descent

You do not calculate the gradients for each observation, which is SGD, but for a group or a batch of observations, which results in faster optimization. Overall you get better performance because it's not so sensitive to picking.You know, if you end up picking an outlier or something like that, due to the order of the batches and the random nature of the samples chosen, it means that the loss kind of variants and it's depending.

## 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.

So you might take 10 of the samples in your data set, calculate the error at 10 across all 10 randomly selected samples.

  - 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.

update all the weights, then do it again for another batch.

  - 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

So sometimes you can still have bad luck if you pick a random number of outliers end up in your batch.Sometimes your weights can end up going in the wrong direction and is dependent on when the training that the batch is chosen. Mostly if your batch is of sufficient size, it's going to you're going to get some good results.

# 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

an epoch is just a single update of the weights.Every single training sample is accounted for.You calculate an MSE mean squared error across every training sample.That's an epoch.You update the weights SGD with a batch size of one.You're going to update the weights for every single training set or for every single sample in your training set.And you're only going to have a single epoch done after you've the same number of updates done as there is number of samples in your training set.So that's the way the epoch works.You're going to do loads of updates per epoch with SGD and mini batches are kind of a hybrid between the two of those.So it's kind of like a for loop over the number of epochs and for each loop, loop proceeds over the training set.

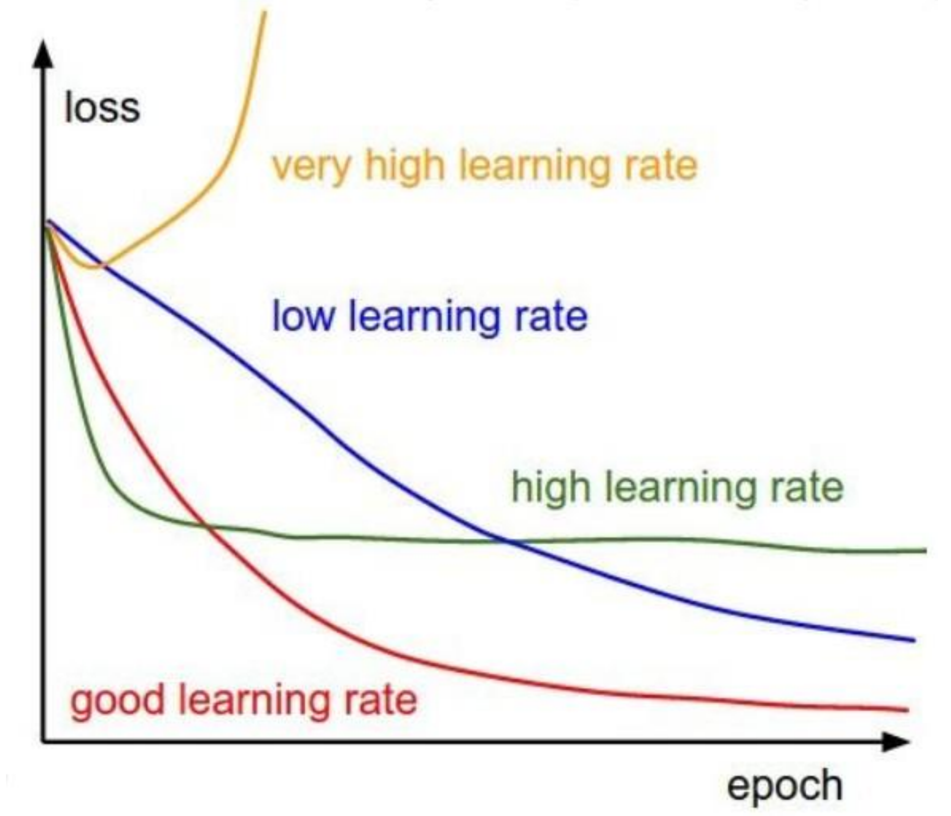
## EPOCHS

- 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

Line plots that show epochs along the X axis and the error or how good the model is, the scale of the model on the Y axis where we can see as we're stepping along down the curves, you know, for all the different, all the gradients, for all the different weights, we can see the performance of our model getting better and better and better.And we can look at that to see, have we plateaued?

## LEARNING RATE

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


loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch