ECE 1508: Applied Deep Learning

Chapter 1: Preliminaries

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

Function Optimization

The model training always reduces to an optimization problem

$$\mathbf{w}^\star = \operatorname*{argmin}_{\mathbf{w}} \hat{R}(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}} \frac{1}{I} \sum_{i=1}^{I} \mathcal{L}\left(f_{\mathbf{h}}\left(\boldsymbol{x}_i | \mathbf{w}\right), y_i\right) \qquad \text{(Training)}$$

Let's recall each component of this optimization problem

- $f_{\mathbf{h}}(\cdot|\mathbf{w})$ model with hyperparameters $\mathbf{h}$ and learnable parameters $\mathbf{w}$
  - ↳ *in DL, it is the input-output relation of a neural network whose architecture is specified by* $\mathbf{h}$ *and whose weights and biases are collected in* $\mathbf{w}$
- $\boldsymbol{x}_i$ is a data-point with label y_i, and I is size of dataset
- $\mathcal{L}$ is the loss function

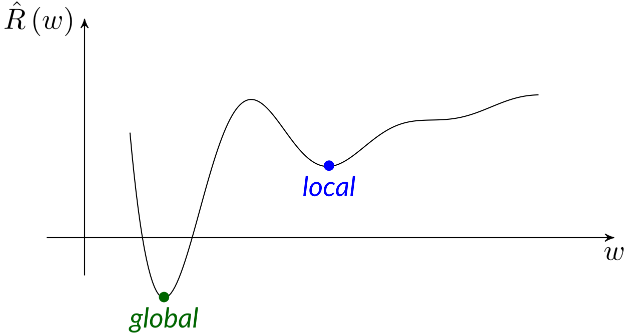
No matter what we choose, at the end of the day we need to solve

$$\min_{\mathbf{w}} \hat{R}(\mathbf{w})$$

Function Optimization

In general, the empirical risk $\hat{R}(\mathbf{w})$ can have local and global minima

Let's take a look at a simple visual case with only one parameter, i.e., $w \in \mathbb{R}$



We are happy if we get the global; but, many times getting to a local is enough!

Function Optimization

+ *Why is it a big problem? We could grid w and search for the grid with smallest empirical risk. We then find it with a good accuracy!*
– For only one parameter yes! But, we have seen deep neural networks. They have too many neurons, and hence too many parameters!

Say for an accurate approximation with only one parameter, we need G grids

If we have D parameters, i.e., $\mathbf{w} \in \mathbb{R}^D$, we need

G^D grids

to get an approximation with the same accuracy!

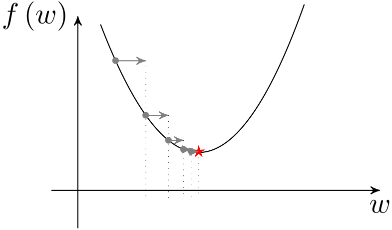
For practical neural networks with $D = 10^5$, this is impossible!

we need to have an optimization algorithm with feasible complexity

Optimization Algorithms $\equiv$ *Optimizer*

We look for an optimization algorithm, *or as ML people call it "an optimizer"*

- *it starts from an initial point and moves for some steps*
- *in each step, it moves towards where the empirical risk is minimized*
- *it moves for a feasible number of steps*



Optimization Algorithms $\equiv$ *Optimizer*

Let's clear things up: we are looking for an iterative approach as below

1: Initiate at some $\mathbf{w}^{(0)} \in \mathbb{R}^D$ and deviation $\Delta = +\infty$
2: Choose some small ϵ
3: **while** $\Delta > \epsilon$ **do**
4: Determine a vector $\boldsymbol{\mu}^{(t)} \in \mathbb{R}^D$ based on $\hat{R}(\mathbf{w})$ $\leftarrow$ $\boxed{\text{we need to figure out}}$
5: Update weights as $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} + \boldsymbol{\mu}^{(t)}$
6: Update the deviation $\Delta = |\hat{R}(\mathbf{w}^{(t)}) - \hat{R}(\mathbf{w}^{(t-1)})|$
7: **end while**

We would like to have following properties

↳ *most of the time the empirical risk reduces in line 5*

↳ *the algorithm stops after a feasible number of iterations*

Optimization Algorithms

Initiate at some $\mathbf{w}^{(0)} \in \mathbb{R}^D$ and deviation $\Delta = +\infty$
Choose some small ϵ, and set $t = 1$
while $\Delta > \epsilon$ **do**
  Determine a vector $\boldsymbol{\mu}^{(t)} \in \mathbb{R}^D$ based on $\hat{R}(\mathbf{w})$ $\leftarrow$ we need to figure out
  Update weights as $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} + \boldsymbol{\mu}^{(t)}$
  Update the deviation $\Delta = |\hat{R}(\mathbf{w}^{(t)}) - \hat{R}(\mathbf{w}^{(t-1)})|$
end while

We are going to get what we want, if we set

$\boldsymbol{\mu}^{(t)}$ *to be proportional to the negative of gradient at* $\mathbf{w}^{(t-1)}$

This is what we call the *gradient descent algorithm*. *But, before we start with this algorithm, let's recap some basic notions of calculus!*

Review: *Derivative of a Function*

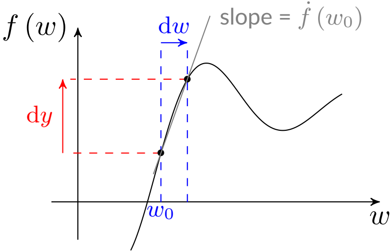
Derivative of one-dimensional function $f(w)$ at point $w = w_0$ is defined as

$$\dot{f}(w_0) = \frac{\mathrm{d}}{\mathrm{d}w} f(w_0) = f'(w_0) = \lim_{\delta \to 0} \frac{f(w_0 + \delta) - f(w_0)}{\delta}$$

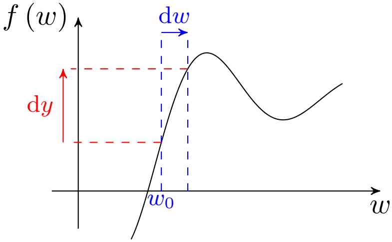
This definition is intuitively interpreted as follows:

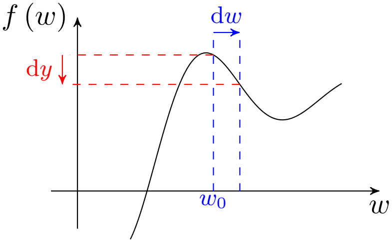
Let $y = f(w)$. If we vary w around w_0 with a tiny step $\mathrm{d}w$; then,

$$\text{Variation of } y = \mathrm{d}y = \dot{f}(w_0)\,\mathrm{d}w$$



Review: *Derivative of a Function*





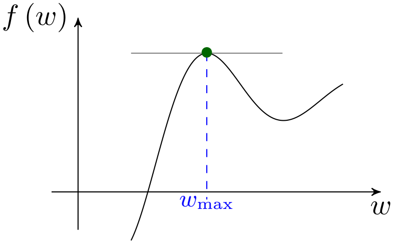
The derivative represents the slope of function

- $\dot{f}(w_0) > 0$ means *increasing* w *will increase* $y = f(w)$
- $\dot{f}(w_0) < 0$ means *increasing* w *will decrease* $y = f(w)$

So, we could also say: *the derivative shows the moving direction on* w*-axis towards which the function increases*; or alternatively, *its negative is the direction that function decreases*

Review: *Derivative of a Function*

When do we have the derivative equal to zero? Either we are at a *maximum*



Starting before the maximum,

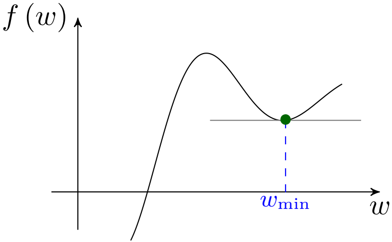
- The derivative is *first positive* and gradually *reduces to zero*
- As we pass the maximum the derivative gets *more and more negative*

So around the maximum as we increase w, the derivative reduces

$$\ddot{f}(w_{\max}) = \frac{\mathrm{d}^2}{\mathrm{d}w^2} f(w_{\max}) = f''(w_{\max}) < 0$$

Review: *Derivative of a Function*

When do we have the derivative equal to zero? Either we are at a *minimum*



Starting before the minimum,

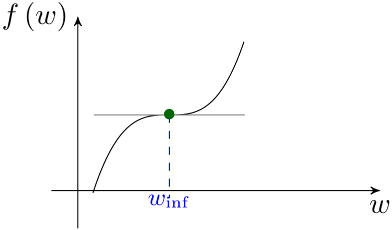
- The derivative is *first negative* and gradually *increases to zero*
- As we pass the minimum the derivative gets *more and more positive*

So around the maximum as we increase w, the derivative reduces

$$\ddot{f}(w_{\min}) = \frac{\mathrm{d}^2}{\mathrm{d}w^2} f(w_{\min}) = f''(w_{\min}) > 0$$

Review: *Derivative of a Function*

When do we have the derivative equal to zero? Either we are at an *inflection*



Starting before the inflection point,

- The derivative is *first positive* and gradually *decreases to zero*
- As we pass the inflection point the derivative gets *again positive*

So around the inflection point, the second derivative *changes sign*

$$\ddot{f}(w_{\text{inf}}) = \frac{\mathrm{d}^2}{\mathrm{d}w^2} f(w_{\text{inf}}) = f''(w_{\text{inf}}) = 0$$

Review: *Gradient of a Function*

+ *What about multi-variable functions, e.g., $f(\mathbf{w})$ for $\mathbf{w} = [w_1, \ldots, w_N]$?*
– We can take derivative with respect to each variable, i.e.,

$$\dot{f}_n(\mathbf{w}_0) = \frac{\partial}{\partial w_n} f(\mathbf{w}_0)$$

This is what we call *partial derivative*

Partial derivative n represents the same thing: *slope in direction of w_n*

Let $y = f(\mathbf{w})$. If we vary $\mathbf{w}$ around $\mathbf{w}_0$ in N-dimensional space with

$$\mathrm{d}\mathbf{w} = [\mathrm{d}w_1, \ldots, \mathrm{d}w_N]$$

whose entries are very tiny; then, the variation of y is

$$\mathrm{d}y = \dot{f}_1(\mathbf{w}_0)\,\mathrm{d}w_1 + \ldots + \dot{f}_N(\mathbf{w}_0)\,\mathrm{d}w_N = \sum_{n=1}^{N} \dot{f}_n(\mathbf{w}_0)\,\mathrm{d}w_n$$

Review: *Gradient of a Function*

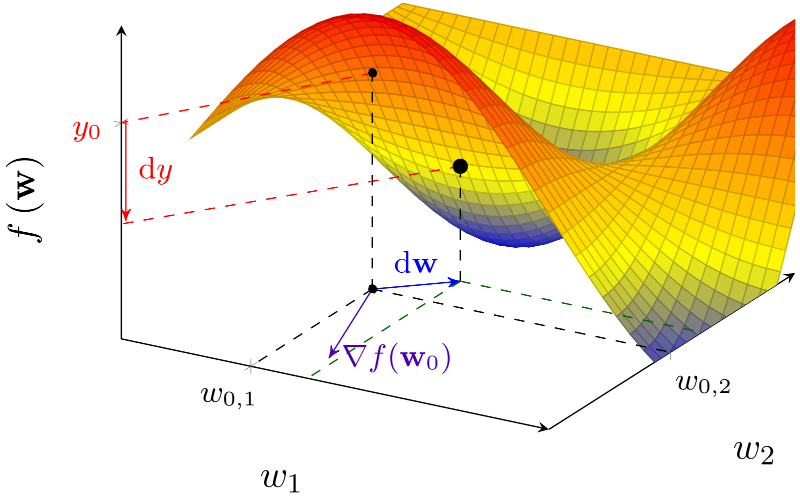
We can use inner-product to represent $\mathrm{d}y$ *compactly*

$$\mathrm{d}y = \sum_{n=1}^{N} \dot{f}_n\left(\mathbf{w}_0\right) \mathrm{d}w_n = \underbrace{\left[\dot{f}_1\left(\mathbf{w}_0\right) \quad \dots \quad \dot{f}_N\left(\mathbf{w}_0\right)\right]}_{\nabla f(\mathbf{w}_0)^{\mathsf{T}}} \begin{bmatrix} \mathrm{d}w_1 \\ \vdots \\ \mathrm{d}w_N \end{bmatrix}$$

$$= \nabla f\left(\mathbf{w}_0\right)^{\mathsf{T}} \mathrm{d}\mathbf{w}$$

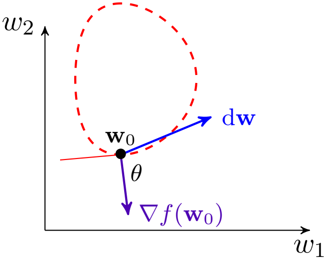
We call $\nabla f\left(\mathbf{w}_0\right)$ *the gradient of* $f\left(\cdot\right)$ *at* $\mathbf{w} = \mathbf{w}_0$

Review: *Gradient of a Function*



Review: *Gradient of a Function*

Let's get to the $\mathbf{w}$*-plane:* the gradient is perpendicular to the contour level



The variation of y is the inner product of these two vectors

$$\mathrm{d}y = \nabla f\left(\mathbf{w}_0\right)^{\mathsf{T}} \mathrm{d}\mathbf{w} = \|\nabla f\left(\mathbf{w}_0\right)\| \|\mathrm{d}\mathbf{w}\| \cos\left(\theta\right)$$

where $\|\cdot\|$ is the Euclidean norm, i.e., $\|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2}$

Review: *Gradient of a Function*

Say we move with a tiny step of fixed size: so we have

$$\|\mathrm{d}\mathbf{w}\| = \epsilon$$

for some small ϵ

+ *How can we move, such that y maximally increases?*
- Well, we need $\theta = 0$ meaning that

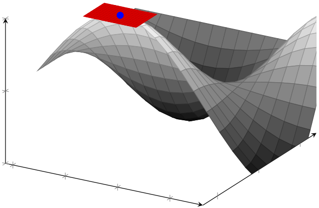
we should move in the direction of gradient

Alternatively, the function decreases maximally if $\theta = \pi$ or

we move in the direction of negative gradient

Review: *Gradient of a Function*

When do we have zero gradient? Either when we are at a maximum

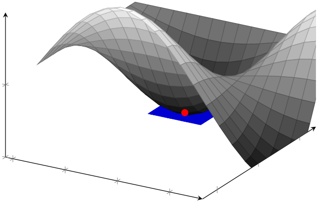


We can again relate it to the *second order derivatives* of the function

at maximum Hessian matrix is negative definite

Review: *Gradient of a Function*

When do we have zero gradient? or when we are at a minimum

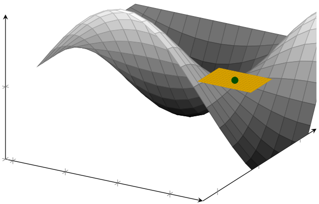


We can again relate it to the *second order derivatives* of the function

at minimum Hessian matrix is positive definite

Review: *Gradient of a Function*

When do we have zero gradient? or when we are at a *saddle point*



We can again relate it to the *second order derivatives* of the function

at saddle point Hessian matrix is neither negative nor positive definite

Review: *Gradient of a Function*

Just as a reminder: Hessian is the matrix of all *second order derivatives*

$$\nabla^2 f(\mathbf{w}_0) = \begin{bmatrix} \frac{\partial^2}{\partial w_1^2} f(\mathbf{w}_0) & \frac{\partial^2}{\partial w_1 \partial w_2} f(\mathbf{w}_0) & \cdots & \frac{\partial^2}{\partial w_1 \partial w_N} f(\mathbf{w}_0) \\ \vdots & & & \vdots \\ \frac{\partial^2}{\partial w_N \partial w_1} f(\mathbf{w}_0) & \frac{\partial^2}{\partial w_N \partial w_2} f(\mathbf{w}_0) & \cdots & \frac{\partial^2}{\partial w_N^2} f(\mathbf{w}_0) \end{bmatrix}$$

We never use the Hessian matrix in this course

Moral of Story: *Gradient Decent*

+ *What is the whole motive of this discussions?*
- Simple: *at any point $\mathbf{w}_0$, if we want to move in a direction that the function reduces, the best direction is negative of gradient at $\mathbf{w}_0$*

So, we can complete our optimization algorithm as follows:

1: Initiate at some $\mathbf{w}^{(0)} \in \mathbb{R}^D$ and deviation $\Delta = +\infty$
2: Choose some small ϵ and η, and set $t = 1$
3: **while** $\Delta > \epsilon$ **do**
4: Update weights as $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla \hat{R}(\mathbf{w}^{(t-1)})$
5: Update the deviation $\Delta = |\hat{R}(\mathbf{w}^{(t)}) - \hat{R}(\mathbf{w}^{(t-1)})|$
6: **end while**

The scalar η is the step-size we take in each iteration:

we usually call it learning rate

Behavior of Gradient Decent

+ *Can we always use gradient descent?*
– Pretty much Yes! The problem starts only when *the empirical loss is not differentiable*

How to handle this problem?

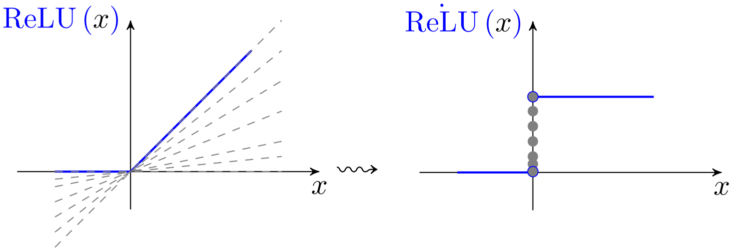
There are two sources for *being non-differentiable*

1. a function that is continuous but not differentiable
2. a discontinuous function

Let's look at each case separately

Behavior of Gradient Decent: *Non-differentiable Elements*

An example of a non-differentiable continuous function is ReLU

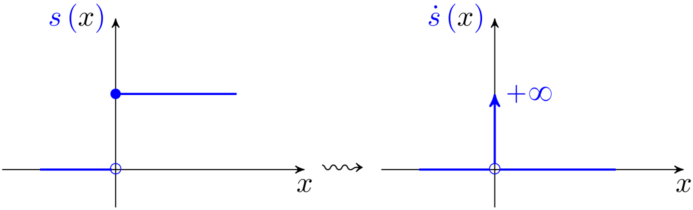


In this case, we define a *sub-gradient* and use it instead of gradient

take the slope of a line that lies below the curve at the point

Behavior of Gradient Decent: *Discontinuous Elements*

An example of a discontinuous function is the step function



The gradient is somehow infinite! *We can only rely on the sign of variation*

we always avoid such elements in our model and loss

Bingo! You may recall that *we discouraged the choice of activation and loss function in the perceptron machine*

Behavior of Gradient Decent: *At Stationary Points*

+ *Now, let's assume that we've handled differentiability. Does gradient decent always end up at the minimum point?*
– This brings up the concept of *convergence*

Let's look at the algorithm again

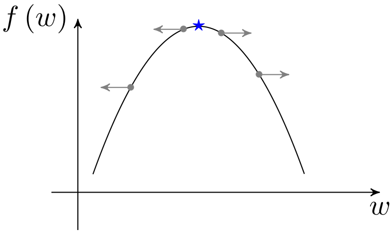
1: Initiate at some $\mathbf{w}^{(0)} \in \mathbb{R}^D$ and deviation $\Delta = +\infty$
2: Choose some small ϵ and η, and set $t = 1$
3: **while** $\Delta > \epsilon$ **do**
4: Update weights as $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla \hat{R}(\mathbf{w}^{(t-1)})$
5: Update the deviation $\Delta = |\hat{R}(\mathbf{w}^{(t)}) - \hat{R}(\mathbf{w}^{(t-1)})|$
6: **end while**

Intuitively, if we set ϵ *very small*: *the algorithm stops when the gradient is close to zero, i.e., when we are at a maximum, minimum or an inflection/saddle-point*

Let's see how the algorithm behaves when we get close to such point

Behavior of Gradient Decent: *At Stationary Points*

When we are around a maximum



If we are exactly at a maximum; then, the algorithm stops. But, in reality

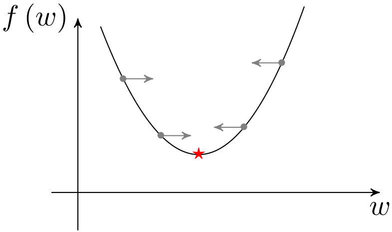
we land somewhere around it

at such points, *the algorithm always pushes us outwards*

Gradient descent practically does not get into a maximum

Behavior of Gradient Decent: *At Stationary Points*

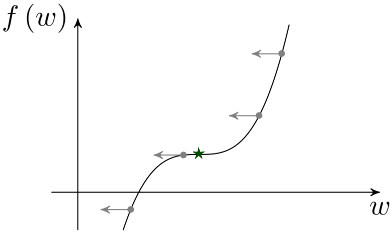
When we are around a minimum



Around minima, the algorithm always pushes us towards the minimum

Behavior of Gradient Decent: *At Stationary Points*

When we are around an inflection point



If we are exactly at an inflection; then, the algorithm stops. But, in reality

we land somewhere around it

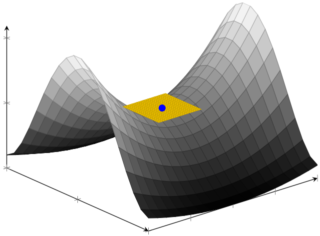
at such points, *the algorithm always pushes us somewhere else*

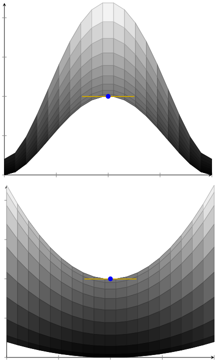
Gradient descent practically does not get into an inflection

Behavior of Gradient Decent: *At Stationary Points*

+ *Can we extend this conclusion to saddle-points?*
– Yes, but with a bit of caution!

At saddle points, function is maximized in a direction and minimized in another





Behavior of Gradient Decent: *At Stationary Points*

So, for a saddle-point we can conclude: if we are exactly at a saddle-point; then, the algorithm stops. But, in reality

we land somewhere around it

If at that point, the gradient has a component in the direction that the function is maximized; then, *the algorithm pushes us outwards*.

+ *Can it happen that we do not land at such point?*
- Thinking with an engineer's mind: *Not really!*

So we could say

Gradient descent almost never gets trapped at a saddle-point

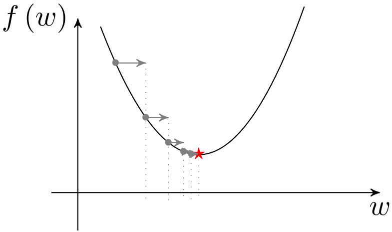
Behavior of Gradient Decent: *Convergence*

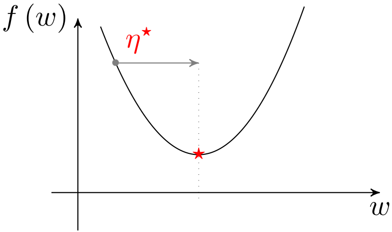
Moral of Story

Gradient descent almost never gets trapped at a point that is not minimum

+ *Nice! But, does it always converge?*
– Well! If we choose the *learning rate* properly; then, Yes!

With small *learning rates*, the algorithm *converges*; *how small?* $\boxed{\eta < \eta^\star}$





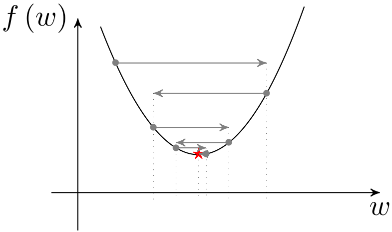
Behavior of Gradient Decent: *Convergence*

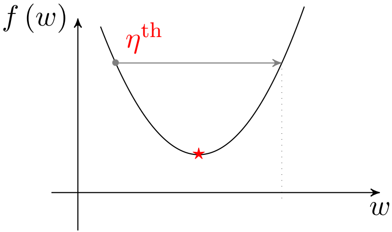
Moral of Story

Gradient descent almost never gets trapped at a point that is not minimum

- \+ *Nice! But, does it always converge?*
- – Well! If we choose the *learning rate* properly; then, Yes!

With larger *learning rates*, the algorithm starts *oscillating*: $\boxed{\eta^\star < \eta < \eta^{\rm th}}$





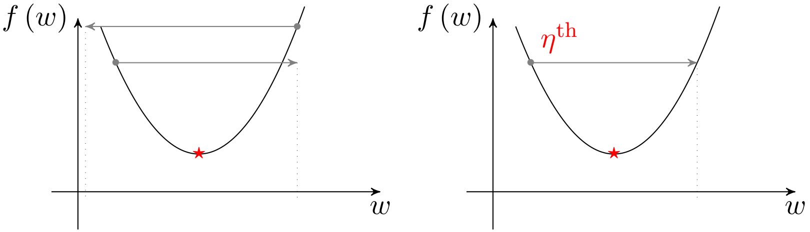
Behavior of Gradient Decent: *Convergence*

Moral of Story

Gradient descent almost never gets trapped at a point that is not minimum

+ *Nice! But, does it always converge?*
- Well! If we choose the *learning rate* properly; then, Yes!

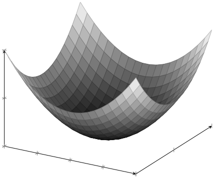
With extremely large *learning rates*, the algorithm *diverges*: $\boxed{\eta > \eta^{\mathrm{th}}}$

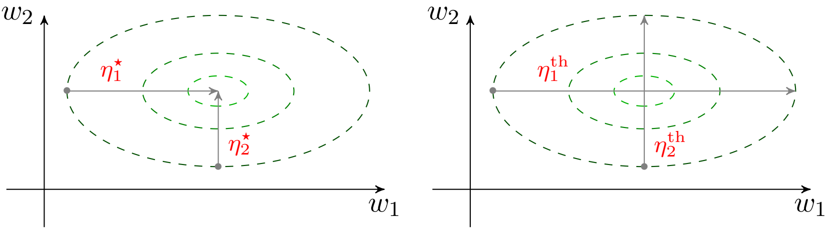


Behavior of Gradient Decent: *Convergence*

We can extend this idea to the multi-dimensional functions; however,

$\eta^\star$ and η^{th} are different on each axis





Behavior of Gradient Decent: *Convergence*

One may suggest that we use a *vector of learning rates*, i.e.,

$$w_n^{(t)} \leftarrow w_n^{(t-1)} - \eta_n \frac{\partial}{\partial w_n} \hat{R}(\mathbf{w}^{(t-1)})$$

for each $n = 1, \ldots, N$. *This is however not easy; the easier way is to focus on*

$$\eta^\star = \min_n \eta_n^\star \qquad \text{and} \qquad \eta^{\text{th}} = \min_n \eta_n^{\text{th}}$$

Behavior of Gradient Decent: *Convergence*

Clearly, there is always a trade-off

- We can choose a large learning rate
  - \+ *Gradient descent converges faster: high convergence speed*
  - – *The chance of divergence however increases: high divergence rate*
- We can choose a small learning rate
  - – *Gradient descent converges slowly: low convergence speed*
  - \+ *The chance of divergence is now very low: low divergence rate*

\+ *Well, say we are patient! Then, choosing a small learning rate is safe! Right?!*

– Well! If the empirical risk is convex; then, Yes! But, with non-convex risks *not always!*

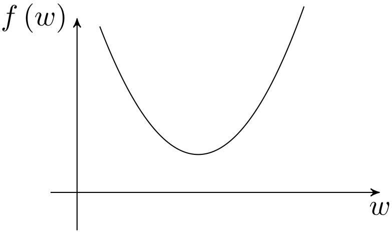
Recap: Convex Function

You really don't need to know the definition of a convex function for this course; however, just in case you're interested, here it goes:

$f(\cdot) : \mathbb{R}^N \mapsto \mathbb{R}$ is convex, if for any two points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^N$, we have

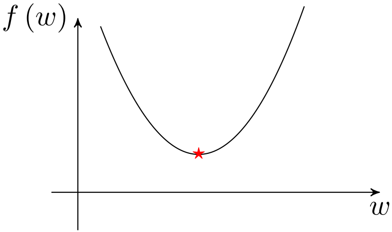
$$f(\lambda \boldsymbol{x}_1 + (1-\lambda)\, \boldsymbol{x}_2) \leqslant \lambda f(\boldsymbol{x}_1) + (1-\lambda)\, f(\boldsymbol{x}_2)$$

for all $0 \leqslant \lambda \leqslant 1$



Behavior of Gradient Decent: *Convergence*

In convex functions, we don't have disjoint local minima



So, if we choose a small learning rate

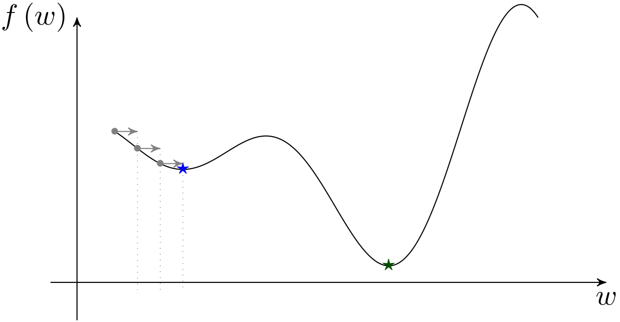
we surely converge to the global minimum

Behavior of Gradient Decent: *Convergence*

But, most empirical losses in deep learning are non-convex:

we have multiple disjoint local minima

Gradient descent converges to one of them, but not necessarily the global one



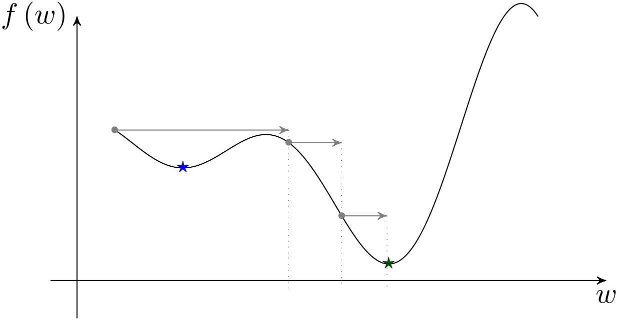
Too small learning rate can leave us in a bad local minimum!

Behavior of Gradient Decent: *Convergence*

But, most empirical losses in deep learning are non-convex:

we have multiple disjoint local minima

Gradient descent converges to one of them, but not necessarily the global one



Initial larger learning rates can take us out of bad local minima!

Behavior of Gradient Decent: *Convergence*

We can hence conclude a more general trade-off

- We can choose a large learning rate
  - + *Gradient descent converges faster: high convergence speed*
  - + *Gradient descent may fall out of a local minimum: lower risk*
  - – *The chance of divergence however increases: high divergence rate*
- We can choose a small learning rate
  - – *Gradient descent converges slowly: low convergence speed*
  - – *Gradient descent traps in a local minimum: high risk*
  - + *The chance of divergence is now very low: low divergence rate*

+ *How do we do it in practice?*

– In practice, we start with large learning rates and reduce it gradually as we get close to the minimum

We will talk about this more once we start training practical neural networks!

Gradient Decent: *Summary*

1: Initiate at some $\mathbf{w}^{(0)} \in \mathbb{R}^D$ and deviation $\Delta = +\infty$
2: Choose some small ϵ and η, and set $t = 1$
3: **while** $\Delta > \epsilon$ **do**
4: Update weights as $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta \nabla \hat{R}(\mathbf{w}^{(t-1)})$
5: Update the deviation $\Delta = |\hat{R}(\mathbf{w}^{(t)}) - \hat{R}(\mathbf{w}^{(t-1)})|$
6: **end while**

Gradient descent converges almost always to a local minimum

- *With convex empirical risks, this is global minimum*
- *With non-convex empirical risks, this is not necessarily global minimum*
- *Learning rate is a crucial parameter that tunes the convergence*

There are other optimization algorithms that work based on gradient: we will talk about them later!