1. What is the concept of cyclical momentum?

I’ve been enjoying several good weeks. Some of that is to be expected at this time of year. I’m a big fan of summer, complete with long hours of daylight, soaring temps, and abundant flora and fauna everywhere I look. By contrast, I can get a little grim and enervated for a few weeks on either side of the winter solstice.

This is far from unique to me. Most people are familiar with the notion of circadian rhythms, cycles based on a day, but circannual rhythms are also a known phenomenon. Circa-monthly, too (I don’t know a tidier term for that one).

Such cycling goes well beyond the biological. Put humans together in a society, and they start playing off of one another to create other cycles. Bull-and-bear stock markets. Larger-scale economic upturns and downturns. Political sentiment. An individual immersed in all of these cycles might be hard-pressed to identify which of them is having what effect on his subjective reality. One might even subscribe to – or invent – unverifiable cyclic events in a search for explanation, if not predictability (astrology, anyone?).

So, as I imagine most people do, I perceive cycles in my life. Mood, energy-level, motivation. Spates of good/bad news, even “luck,” as unscientific as such a thing seems to be for a physician to talk about. As a human, being a pattern-recognition machine, I probably selectively notice and overestimate things that mesh with whatever cyclic phase I’m in. If I’m in a good mood and a dozen random events come my way, I might perceive more of them to be positive than I otherwise would…and/or I might be less inclined to notice unhappier stuff.

(Lest it need saying: I’m not talking about pathological, unstable stuff like cyclothymia here. Any readers experiencing ego-dystonic cycling should probably be seeking individualized treatment from a mental health professional rather than hoping for direction from online blogs.)

I’ve come to find it worth my while to recognize my cycles, however subjective they may be, and when possible to adjust my behavior to take advantage of them. Or, to minimize whatever negative impact they might otherwise have.

An analogy: You’re pushing a kid on a swing-set. The swing is cycling between forward and backward arcs. If you want to make it go higher, you give it a push when the momentum of the cycle is in your favor—that is, the kid is moving forward, away from you. Pushing against the momentum, when he’s coming back towards you, will be disruptive, taking away from his overall arc (and being pretty jarring to the kid).

Similarly, if I can tell I’m on an upswing—good mood, high energy, things seeming to be going my way—I’ve got a better chance of good results when taking on tasks: getting chores done, solving problems, even writing columns like this one. If I’m “not feeling it,” I’m much better off leaving things for later on than forcing myself to plow ahead.

Sometimes that’s not an option, for instance if I’m on a schedule—a meeting or a deadline later today that can’t be pushed back without consequence. Having to “power through” when I’m in a downturn, even if absolutely necessary, can be unpleasant, feeling an awful lot more effort-intensive. Further, when looking back on things later on, it can be painfully obvious to me that things didn’t go as well as they might have. My performance in the meeting, or the quality of whatever I turned in for the deadline, was a middling B- instead of an A+.

There are some ways to prevent this from happening. Regarding deadlines, for instance, I like to get things done comfortably before they are due. In addition to keeping the pressure off, it gives me more chances to work on projects when I am cycling favorably. If my due-date is Friday and that turns out to be a lousy day, but I only got working on the task that morning, I’ve got no options. But, if I start looking to get it done at the outset of the week, and recognize that Tuesday or Thursday are turning out to be good days for me, I can declare either of them to be “soft” deadlines on the fly.

Of course, we don’t always have such a luxury of prep-time to “choose our window” for performance. That meeting scheduled for this afternoon, for instance, or the dozens of cases on your worklist, are stationary, pressing concerns. Sometimes there’s nothing for it but to grab an extra cup of coffee in the hope of goosing yourself into a brief, artificial upswing (and, truth be told, sometimes such “fake it till you make it” maneuvers actually work, reversing a downward cycle-phase).

A little over two years ago in this column, I described how I participated in a business-venture that almost succeeded. One of the ways I knew it had a good chance was that, initially, it felt like everything about it was flowing along easily, practically running itself without a hint of friction or resistance. My partners and I, our allies, even the market-sector we were pursuing, were all in a good place. We had momentum, and it was carrying us forward almost with a sense of guaranteed success, like it was meant to happen.

As months went by, however, that momentum faded. For whatever reason, the cycle had turned. Our every action now felt like it was facing resistance. Or without any effect whatsoever, like trying to push on a rope. We still gave it our best, but as it became increasingly clear that things no longer “wanted” to move forward, we saw that it made no sense to keep on throwing good resources (including time and effort) after bad.

In contrast, I’ll reiterate that these past few weeks have been good ones for me. I’ve taken multiple opportunities to let my cyclic momentum propel me forward—and it feels like this has resulted in something of a positive-feedback loop. That is, optimistically taking on more stuff (and at least perceiving that I’m doing well with it) might just have prolonged my usual summer-solstice upswing, which usually doesn’t last this long.

1. What callback keeps track of hyperparameter values (along with other data) during training?

**Defining a callback in Keras**

Keras callbacks help you fix bugs more quickly and build better models.

*“A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.”*

This is exactly what we need as now we can get\_weights() after each mini batch(i.e. after each iteration). The weights are stored in a *weight\_history* list to be accessed later. A separate list is also maintained for the bias terms.

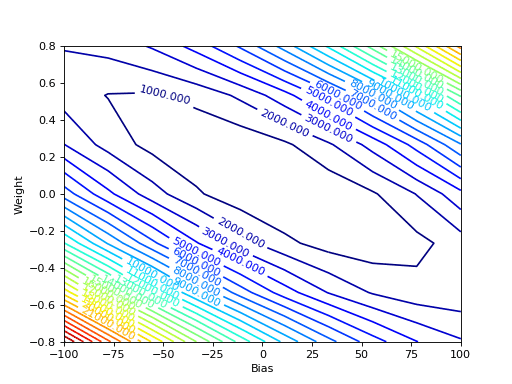
weight\_history = []  
bias\_history = []  
class MyCallback(keras.callbacks.Callback):  
 def on\_batch\_end(self, batch, logs):  
 weight, bias = model.get\_weights() B = bias[0]  
 W = weight[0][0]  
 weight\_history.append(W)  
 bias\_history.append(B)callback = MyCallback()

The created callback is passed along with the inputs and outputs for training the model.

model.fit(X\_train, Y\_train, epochs = 10, batch\_size = 10,  
 verbose = True, **callbacks=[callback]**)

Now, the stored weights could be used to plot the cost function(J) with respect to weight(W) and bias(B).

The contour plot is plotted solely on the basis of the *weight\_history* and *bias\_history.*There is no need to calculate the cost function here.

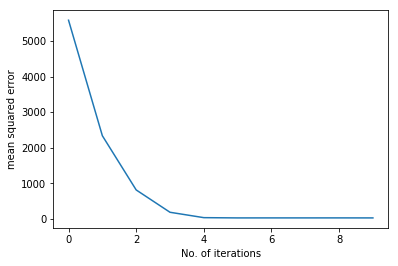


**Interpreting a contour plot**

The basic intuition of a contour plot is that the continuous lines represent constant magnitude (called contour lines) and the magnitude increases as we go from the middle to the outward parts of the plot.

The magnitude of the contour lines has been given and here, it represents the possible values of cost function(J). You can roughly observe that the cost(the red line) starts from close to 5000 and goes on decreasing until it stops at particular point.

This is in correspondence to the loss function values which was also taken as mean-squared-error.



***Note****: Error between the two plots is due to the fact that the mean squared error (above) is calculated in terms of the validation split, and the contour plot is plotted using the entire training data.*

**What works too ?**

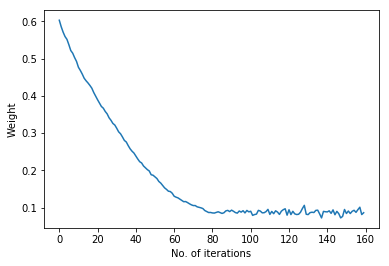
Plotting the loss function over iterations, as above, can also be used for Hyperparameter Tuning. In fact, that is the most commonly used technique by data scientists.

**Why use contour plots ?**

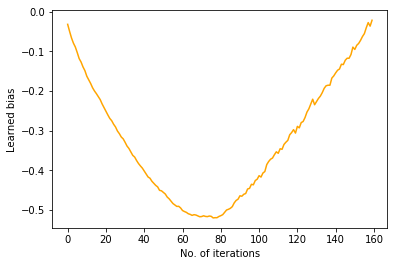
The advantage provided by the contour plots is that, they give a better intuition about the track followed by the gradient descent algorithm w.r.t updates in the model/network parameters over iterations.

**Finally…**

As we already have access to the model parameters it may be worthwhile to observe the trends followed by them over time.



Weight vs. Time



Bias vs. Time

Therefore, the trends followed by weight and bias of our model to reach the local minimum of Cost Function can be observed.

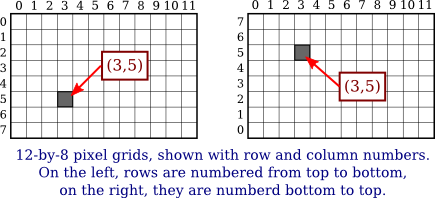
Now that you have access to all the plots you can efficiently check whether your model learns slowly or overshoots (learning rate), whether mini-batching yields a observable benefit, ideal no. of iterations(or even epochs),etc.

1. In the color dim plot, what does one column of pixels represent?

To create a two-dimensional image, each point in the image is assigned a color. A point in 2D can be identified by a pair of numerical coordinates. Colors can also be specified numerically. However, the assignment of numbers to points or colors is somewhat arbitrary. So we need to spend some time studying **coordinate systems**, which associate numbers to points, and **color models**, which associate numbers to colors.

### 2.1.1  Pixel Coordinates

A digital image is made up of rows and columns of pixels. A pixel in such an image can be specified by saying which column and which row contains it. In terms of coordinates, a pixel can be identified by a pair of integers giving the column number and the row number. For example, the pixel with coordinates (3,5) would lie in column number 3 and row number 5. Conventionally, columns are numbered from left to right, starting with zero. Most graphics systems, including the ones we will study in this chapter, number rows from top to bottom, starting from zero. Some, including OpenGL, number the rows from bottom to top instead.

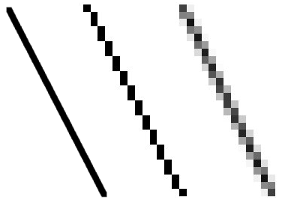


Note in particular that the pixel that is identified by a pair of coordinates (*x*,*y*) depends on the choice of coordinate system. You always need to know what coordinate system is in use before you know what point you are talking about.

Row and column numbers identify a pixel, not a point. A pixel contains many points; mathematically, it contains an infinite number of points. The goal of computer graphics is not really to color pixels—it is to create and manipulate images. In some ideal sense, an image should be defined by specifying a color for each point, not just for each pixel. Pixels are an approximation. If we imagine that there is a true, ideal image that we want to display, then any image that we display by coloring pixels is an approximation. This has many implications.

Suppose, for example, that we want to draw a line segment. A mathematical line has no thickness and would be invisible. So we really want to draw a thick line segment, with some specified width. Let's say that the line should be one pixel wide. The problem is that, unless the line is horizontal or vertical, we can't actually draw the line by coloring pixels. A diagonal geometric line will cover some pixels only partially. It is not possible to make part of a pixel black and part of it white. When you try to draw a line with black and white pixels only, the result is a jagged staircase effect. This effect is an example of something called "aliasing." Aliasing can also be seen in the outlines of characters drawn on the screen and in diagonal or curved boundaries between any two regions of different color. (The term aliasing likely comes from the fact that ideal images are naturally described in real-number coordinates. When you try to represent the image using pixels, many real-number coordinates will map to the same integer pixel coordinates; they can all be considered as different names or "aliases" for the same pixel.)

**Antialiasing** is a term for techniques that are designed to mitigate the effects of aliasing. The idea is that when a pixel is only partially covered by a shape, the color of the pixel should be a mixture of the color of the shape and the color of the background. When drawing a black line on a white background, the color of a partially covered pixel would be gray, with the shade of gray depending on the fraction of the pixel that is covered by the line. (In practice, calculating this area exactly for each pixel would be too difficult, so some approximate method is used.) Here, for example, is a geometric line, shown on the left, along with two approximations of that line made by coloring pixels. The lines are greatly magnified so that you can see the individual pixels. The line on the right is drawn using antialiasing, while the one in the middle is not:



Note that antialiasing does not give a perfect image, but it can reduce the "jaggies" that are caused by aliasing (at least when it is viewed on a normal scale).

There are other issues involved in mapping real-number coordinates to pixels. For example, which point in a pixel should correspond to integer-valued coordinates such as (3,5)? The center of the pixel? One of the corners of the pixel? In general, we think of the numbers as referring to the top-left corner of the pixel. Another way of thinking about this is to say that integer coordinates refer to the lines between pixels, rather than to the pixels themselves. But that still doesn't determine exactly which pixels are affected when a geometric shape is drawn. For example, here are two lines drawn using HTML canvas graphics, shown greatly magnified. The lines were specified to be colored black with a one-pixel line width:



The top line was drawn from the point (100,100) to the point (120,100). In canvas graphics, integer coordinates correspond to the lines between pixels, but when a one-pixel line is drawn, it extends one-half pixel on either side of the infinitely thin geometric line. So for the top line, the line as it is drawn lies half in one row of pixels and half in another row. The graphics system, which uses antialiasing, rendered the line by coloring both rows of pixels gray. The bottom line was drawn from the point (100.5,100.5) to (120.5,100.5). In this case, the line lies exactly along one line of pixels, which gets colored black. The gray pixels at the ends of the bottom line have to do with the fact that the line only extends halfway into the pixels at its endpoints. Other graphics systems might render the same lines differently.

The following interactive demo lets you experiment with pixels and antialiasing. (Note that in any of the interactive demos that accompany this book, you can click the question mark icon in the upper left for more information about how to use it.)

All this is complicated further by the fact that pixels aren't what they used to be. Pixels today are smaller! The resolution of a display device can be measured in terms of the number of pixels per inch on the display, a quantity referred to as PPI (pixels per inch) or sometimes DPI (dots per inch). Early screens tended to have resolutions of somewhere close to 72 PPI. At that resolution, and at a typical viewing distance, individual pixels are clearly visible. For a while, it seemed like most displays had about 100 pixels per inch, but high resolution displays today can have 200, 300 or even 400 pixels per inch. At the highest resolutions, individual pixels can no longer be distinguished.

The fact that pixels come in such a range of sizes is a problem if we use coordinate systems based on pixels. An image created assuming that there are 100 pixels per inch will look tiny on a 400 PPI display. A one-pixel-wide line looks good at 100 PPI, but at 400 PPI, a one-pixel-wide line is probably too thin.

In fact, in many graphics systems, "pixel" doesn't really refer to the size of a physical pixel. Instead, it is just another unit of measure, which is set by the system to be something appropriate. (On a desktop system, a pixel is usually about one one-hundredth of an inch. On a smart phone, which is usually viewed from a closer distance, the value might be closer to 1/160 inch. Furthermore, the meaning of a pixel as a unit of measure can change when, for example, the user applies a magnification to a web page.)

Pixels cause problems that have not been completely solved. Fortunately, they are less of a problem for vector graphics, which is mostly what we will use in this book. For vector graphics, pixels only become an issue during rasterization, the step in which a vector image is converted into pixels for display. The vector image itself can be created using any convenient coordinate system. It represents an idealized, resolution-independent image. A rasterized image is an approximation of that ideal image, but how to do the approximation can be left to the display hardware.

### 2.1.2  Real-number Coordinate Systems

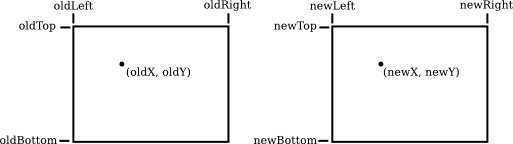
When doing 2D graphics, you are given a rectangle in which you want to draw some graphics primitives. Primitives are specified using some coordinate system on the rectangle. It should be possible to select a coordinate system that is appropriate for the application. For example, if the rectangle represents a floor plan for a 15 foot by 12 foot room, then you might want to use a coordinate system in which the unit of measure is one foot and the coordinates range from 0 to 15 in the horizontal direction and 0 to 12 in the vertical direction. The unit of measure in this case is feet rather than pixels, and one foot can correspond to many pixels in the image. The coordinates for a pixel will, in general, be real numbers rather than integers. In fact, it's better to forget about pixels and just think about points in the image. A point will have a pair of coordinates given by real numbers.

To specify the coordinate system on a rectangle, you just have to specify the horizontal coordinates for the left and right edges of the rectangle and the vertical coordinates for the top and bottom. Let's call these values *left*, *right*, *top*, and *bottom*. Often, they are thought of as *xmin*, *xmax*, *ymin*, and *ymax*, but there is no reason to assume that, for example, *top* is less than *bottom*. We might want a coordinate system in which the vertical coordinate increases from bottom to top instead of from top to bottom. In that case, *top* will correspond to the maximum *y*-value instead of the minimum value.

To allow programmers to specify the coordinate system that they would like to use, it would be good to have a subroutine such as

setCoordinateSystem(left,right,bottom,top)

The graphics system would then be responsible for automatically transforming the coordinates from the specified coordinate system into pixel coordinates. Such a subroutine might not be available, so it's useful to see how the transformation is done by hand. Let's consider the general case. Given coordinates for a point in one coordinate system, we want to find the coordinates for the same point in a second coordinate system. (Remember that a coordinate system is just a way of assigning numbers to points. It's the points that are real!) Suppose that the horizontal and vertical limits are *oldLeft*, *oldRight*, *oldTop*, and *oldBottom* for the first coordinate system, and are *newLeft*, *newRight*, *newTop*, and *newBottom* for the second. Suppose that a point has coordinates (*oldX,oldY*) in the first coordinate system. We want to find the coordinates (*newX,newY*) of the point in the second coordinate system



Formulas for *newX* and *newY* are then given by

newX = newLeft +

((oldX - oldLeft) / (oldRight - oldLeft)) \* (newRight - newLeft))

newY = newTop +

((oldY - oldTop) / (oldBottom - oldTop)) \* (newBottom - newTop)

The logic here is that *oldX* is located at a certain fraction of the distance from *oldLeft* to *oldRight*. That fraction is given by

((oldX - oldLeft) / (oldRight - oldLeft))

The formula for *newX* just says that *newX* should lie at the same fraction of the distance from *newLeft* to *newRight*. You can also check the formulas by testing that they work when *oldX* is equal to *oldLeft* or to *oldRight*, and when *oldY* is equal to *oldBottom* or to *oldTop*.

As an example, suppose that we want to transform some real-number coordinate system with limits *left*, *right*, *top*, and *bottom* into pixel coordinates that range from 0 at left to 800 at the right and from 0 at the top 600 at the bottom. In that case, *newLeft* and *newTop* are zero, and the formulas become simply

newX = ((oldX - left) / (right - left)) \* 800

newY = ((oldY - top) / (bottom - top)) \* 600

Of course, this gives *newX* and *newY* as real numbers, and they will have to be rounded or truncated to integer values if we need integer coordinates for pixels. The reverse transformation—going from pixel coordinates to real number coordinates—is also useful. For example, if the image is displayed on a computer screen, and you want to react to mouse clicks on the image, you will probably get the mouse coordinates in terms of integer pixel coordinates, but you will want to transform those pixel coordinates into your own chosen coordinate system.

In practice, though, you won't usually have to do the transformations yourself, since most graphics APIs provide some higher level way to specify transforms. We will talk more about this in [Section 2.3](https://math.hws.edu/graphicsbook/c2/s3.html).

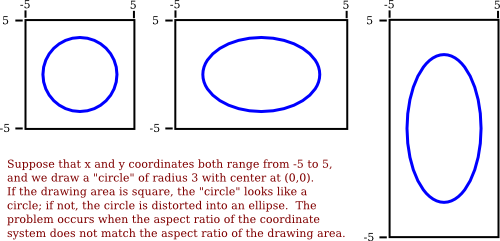
### 2.1.3  Aspect Ratio

The **aspect ratio** of a rectangle is the ratio of its width to its height. For example an aspect ratio of 2:1 means that a rectangle is twice as wide as it is tall, and an aspect ratio of 4:3 means that the width is 4/3 times the height. Although aspect ratios are often written in the form *width*:*height*, I will use the term to refer to the fraction *width/height*. A square has aspect ratio equal to 1. A rectangle with aspect ratio 5/4 and height 600 has a width equal to 600\*(5/4), or 750.

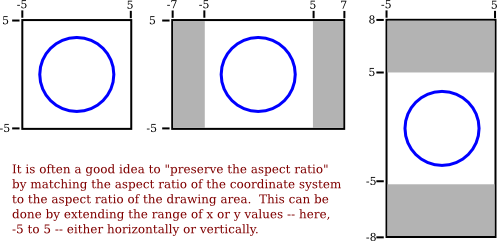
A coordinate system also has an aspect ratio. If the horizontal and vertical limits for the coordinate system are *left*, *right*, *bottom*, and *top*, as above, then the aspect ratio is the absolute value of

(right - left) / (top - bottom)

If the coordinate system is used on a rectangle with the same aspect ratio, then when viewed in that rectangle, one unit in the horizontal direction will have the same apparent length as a unit in the vertical direction. If the aspect ratios don't match, then there will be some distortion. For example, the shape defined by the equation *x2* +*y2* = 9 should be a circle, but that will only be true if the aspect ratio of the (*x*,*y*) coordinate system matches the aspect ratio of the drawing area.



It is not always a bad thing to use different units of length in the vertical and horizontal directions. However, suppose that you want to use coordinates with limits *left*, *right*, *bottom*, and *top*, and that you do want to preserve the aspect ratio. In that case, depending on the shape of the display rectangle, you might have to adjust the values either of *left* and *right* or of *bottom* and *top* to make the aspect ratios match:



We will look more deeply into geometric transforms later in the chapter, and at that time, we'll see some program code for setting up coordinate systems.

4. In color dim, what does "poor teaching" look like? What is the reason for this?

I will share with you a little incident that happened at a bus stop.

A kid, aged around 6 years comes to me begging for money. I say no. He goes to other people. They don't give him too. I observed him for a while and called him back. He came running. This is the conversation that happened between me and the kid.

Me: "What do you want?"  
He: "I haven't eaten since morning"  
Me: "Ok if talk to me for 2 minutes, I will give you the money. Now tell me where are your parents? What do they do?"  
He: "They are at X place. Construction work." (X is a place 15kms away from where we were there)  
Me: "Ok. What do you do?"  
He: -silence-  
Me: "Don't you go to school?"  
He: "No"  
Me: "Do you want to?"  
He: -bored now-  
Me: "I will put you into school if you want to. Do you know school is fun? New friends, new books and you get food too! And you don't have to ask people for money. You can eat whatever you want."  
He: "Akka, I am very hungry"  
*(The conversation was in a local language and not English)*  
I gave the kid a 10 rupee note and left as I was running late.

I agree the kid was hungry and it was not the right time to ask, but I am sure this would be the same reaction even otherwise. The kid was totally disinterested when I spoke to him about school. There was no twinkle in his eyes. Why is this?

**Lack of awareness.**  
Let us look at the kid's point of view. He is a 6 year old with illiterate parents. How can we expect him to know the value of education? He doesn't know it is a tool that helps people to stand on their own legs. All he thinks of it is that you are punished in school, you are beaten up if you don't do homeworks etc. These are not my assumptiions but I have heard kids say that. He finds an easy way(begging) to earn his bread, why school? He doesn't know what begging means. He doesn't know why begging is a blow to the self-respect. He is a 6 year old for god sake! He doesn't know and there is no one to tell him! Lack of awareness.

**Poverty.**  
Now let us look from a parents' point of view. They have a 6 year old kid who can run, walk and beg! Why waste time by sending the kid to school? Instead let the kid beg from 6-9 years and then, put him to get the petty jobs on construction site to be done. Voila! source of income! Every parent loves his kid as much as anyone else. But they do not know how education can enable the kid to grow to be a independent individual.

**Drop-outs.**  
I happened to speak to a Government school teacher. He says that the kids are provided with all the school accessories and free food but still the drop-out rate is more. Kids don't like working hard. They just think it is not their cup of tea and drop out.

**What can we do?**

Bring in an awareness about how important education is. Kids, parents everyone has to be present. If you convince the parents, your work is half done. Most of them do not even know that there are free schools and few of them provide the kids with free food too.

Moral values. Kids need to know that theft and begging should not even be the last option to earn money on a regular basis. They need to know it is not-so-good thing to do.

With the parents' consent enroll them to a school. Make sure they are comfortable there. Have a talk with the parents, kids and teachers about the facilities that will be provided in school. The discussion should be transparent. Give them your(or any concerned person) contact number so that they can contact you when they find it hard to deal with the politics in the school. That way they will be confident about the whole process.

I know this one will take a lot of time and is completely out of our hands but I will still mention it. The education system in Government schools. I wish it was more than classworks and homeworks. I wish the little ones were taught using creative story-telling sessions. I wish they were taught with paints and colors. I wish they did not have to carry bags that weighs more than them, I wish!

5. Does a batch normalization layer have any trainable parameters?

## What is Batch Normalization?

Batch Normalization is a technique used to improve the training of deep [neural networks](https://deepai.org/machine-learning-glossary-and-terms/neural-network). Introduced by Sergey Ioffe and Christian Szegedy in 2015, batch normalization is used to normalize the inputs of each layer in such a way that they have a mean output activation of zero and a [standard deviation](https://deepai.org/machine-learning-glossary-and-terms/standard-deviation) of one. This normalization process helps to combat issues that deep neural networks face, such as internal covariate shift, which can slow down training and affect the network's ability to generalize from the training data.

## Understanding Internal Covariate Shift

Internal covariate shift refers to the change in the distribution of network activations due to the update of weights during training. As deeper layers depend on the outputs of earlier layers, even small changes in the initial layers can amplify and lead to significant shifts in the distribution of inputs to deeper layers. This can result in the need for lower learning rates and careful parameter initialization, making the training process slow and less efficient.

## How Batch Normalization Works

Batch normalization works by normalizing the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. After this step, the result is then scaled and shifted by two learnable parameters, gamma and beta, which are unique to each layer. This process allows the model to maintain the mean activation close to 0 and the activation standard deviation close to 1.

The normalization step is as follows:

1. Calculate the mean and [variance](https://deepai.org/machine-learning-glossary-and-terms/variance) of the activations for each feature in a mini-batch.
2. Normalize the activations of each feature by subtracting the mini-batch mean and dividing by the mini-batch standard deviation.
3. Scale and shift the normalized values using the learnable parameters gamma and beta, which allow the network to undo the normalization if that is what the learned behavior requires.

Batch normalization is typically applied before the [activation function](https://deepai.org/machine-learning-glossary-and-terms/activation-function) in a network layer, although some variations may apply it after the activation function.

## Benefits of Batch Normalization

Batch normalization offers several benefits to the training process of deep neural networks:

* **Improved Optimization:** It allows the use of higher learning rates, speeding up the training process by reducing the careful tuning of parameters.
* **Regularization:** It adds a slight noise to the activations, similar to dropout. This can help to regularize the model and reduce overfitting.
* **Reduced Sensitivity to Initialization:** It makes the network less sensitive to the initial starting weights.
* **Allows Deeper Networks:** By reducing internal covariate shift, batch normalization allows for the training of deeper networks.

## Batch Normalization During Inference

While batch normalization is straightforward to apply during training, it requires special consideration during inference. Since the mini-batch mean and variance are not available during inference, the network uses the moving averages of these statistics that were computed during training. This ensures that the normalization is consistent and the network's learned behavior is maintained.

6. In batch normalization during preparation, what statistics are used to normalize? What about during the validation process?

#### Key Takeaways:

* Batch normalization helps prevent overfitting and speeds up training of deep neural networks
* It normalizes activations of each layer by subtracting mean and dividing by standard deviation
* Rescaling and offsetting is done using learnable parameters gamma and beta
* Batch normalization handles internal covariate shift and smoothens the loss landscape

Training deep neural networks presents difficulties such as vanishing gradients and slow convergence. In 2015, Sergey Ioffe and Christian Szegedy introduced ***Batch Normalization***as a powerful technique to tackle these challenges. This article will explore Batch Normalization and how it can be utilized in Keras, a well-known deep-learning framework.

## What is meant by Batch Normalization in Deep Learning?

**Batch Normalization**is a technique used in deep learning to standardize the inputs of each layer, ensuring stable training by reducing internal covariate shifts and accelerating convergence. It involves normalizing the activations with mean and variance calculated over mini-batches, along with learnable parameters for scaling and shifting.

## Applying Batch Normalization in Keras using BatchNormalization Class

The ***keras.layers.BatchNormalization***class in Keras implements Batch Normalization, a technique used to normalize the activations of a layer in a neural network.

### ****Syntax of BatchNormalization Class in Keras****

keras.layers.BatchNormalization(

axis=-1,

momentum=0.99,

epsilon=0.001,

center=True,

scale=True,

beta\_initializer="zeros",

gamma\_initializer="ones",

moving\_mean\_initializer="zeros",

moving\_variance\_initializer="ones",

beta\_regularizer=None,

gamma\_regularizer=None,

beta\_constraint=None,

gamma\_constraint=None,

synchronized=False,

\*\*kwargs)

### ****BatchNormalization Class**** Parameters

Here’s a breakdown of its parameters:

* **axis**: Specifies the axis along which normalization is applied. By default, it normalizes along the last axis (usually the features axis).
* **momentum**: A float value between 0 and 1 that represents the exponential decay rate for the moving mean and moving variance estimates. A higher momentum value means the statistics from previous batches have more influence.
* **epsilon**: A small float value added to the variance to prevent division by zero.
* **center**: If True, the layer will learn an offset parameter (beta). If False, this parameter is disabled.
* **scale**: If True, the layer will learn a scale parameter (gamma). If False, this parameter is disabled.
* **beta\_initializer**: Initializer for the beta (offset) parameter.
* **gamma\_initializer**: Initializer for the gamma (scale) parameter.
* **moving\_mean\_initializer**: Initializer for the moving mean parameter.
* **moving\_variance\_initializer**: Initializer for the moving variance parameter.
* **beta\_regularizer**: Regularizer function applied to the beta parameter.
* **gamma\_regularizer**: Regularizer function applied to the gamma parameter.
* **beta\_constraint**: Constraint function applied to the beta parameter.
* **gamma\_constraint**: Constraint function applied to the gamma parameter.
* **synchronized**: A boolean indicating whether Batch Normalization should be synchronized across replicas during distributed training. This is useful for distributed training setups.
* **kwargs**: Additional keyword arguments accepted by the base Layer class.

These parameters allow for fine-tuning and customization of the Batch Normalization layer according to specific requirements and architectural considerations. For example, you can control whether to include learnable parameters (beta and gamma), specify the initialization and regularization methods, and adjust the axis of normalization.

## Implementing ****Batch Normalization Class**** in Keras

In this section, we are going to cover all the steps required to implement Batch Normalization in Keras with help of **BatchNormalization Class**. Let’s discuss the steps:

### Step 1: Importing Libraries

import numpy as np

from sklearn.model\_selection import train\_test\_split

from keras.models import Sequential

from keras.layers import Dense, BatchNormalization

### Step 2: Create a dummy dataset

# Generate toy dataset

np.random.seed(0)

X = np.random.randn(1000, 10) # 1000 samples, 10 features

y = np.random.randint(2, size=(1000,)) # Binary labels

### Step 3: Define the Model

A sequential model is defined using [Sequential()](https://www.geeksforgeeks.org/tensorflow-js-tf-sequential-function/). It consists of three dense layers. The first two layers have[ReLU activation functions](https://www.geeksforgeeks.org/activation-functions-neural-networks/) and Batch Normalization layers after them, and the final layer has a [sigmoid activation](https://www.geeksforgeeks.org/activation-functions-neural-networks/) function for binary classification.

# Define the model

model = Sequential()

model.add(Dense(64, input\_shape=(10,), activation='relu'))

**model.add(BatchNormalization())**

model.add(Dense(32, activation='relu'))

**model.add(BatchNormalization())**

model.add(Dense(1, activation='sigmoid'))

### Step 4: Compiling the Model

# Train the model

model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.1)

7. Why do batch normalization layers help models generalize better?

### What is Batch Normalization?

Batch normalization is a method that can enhance the efficiency and reliability of deep neural network models. It is very effective in training [convolutional neural networks (CNN)](https://viso.ai/deep-learning/convolutional-neural-networks/" \t "_blank), providing faster [neural network](https://viso.ai/deep-learning/deep-neural-network-three-popular-types/) convergence. As a supervised learning method, BN normalizes the activation of the internal layers during training. The next layer can analyze the data more effectively by resetting the output distribution from the previous layer.

 Neural network layers and activations from the [Artificial Neural Networks](https://viso.ai/deep-learning/artificial-neural-network/) blog by viso.ai

 Internal covariate shift denotes the effect that parameters change in the previous layers have on the inputs of the current layer. This makes the optimization process more complex and slows down the model convergence.

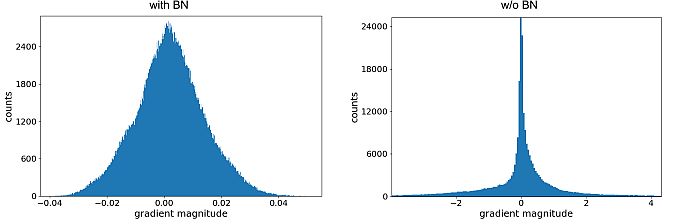
In batch normalization – the activation value doesn’t matter, making each layer learn separately. This approach leads to faster learning rates. Also, the amount of information lost between processing stages may decrease. That will provide a significant increase in the precision of the network.

**How Does Batch Normalization Work?**

The batch normalization method enhances the efficiency of a [deep neural network](https://viso.ai/deep-learning/deep-neural-network-three-popular-types/) by discarding the batch mean and dividing it into the batch standard deviation. The [gradient descent](https://viso.ai/computer-vision/gradient-descent/) method scales the outputs by a parameter if the loss function is large. Subsequently – it updates the weights in the next layer.

Batch normalization aims to improve the training process and increase the model’s generalization capability. It reduces the need for precise initialization of the model’s weights and enables higher learning rates. That will accelerate the training process.

Batch normalization multiplies its output by a standard deviation parameter (γ). Also, it adds a mean parameter (beta) when applied to a layer. Due to the coaction between batch normalization and gradient descent, data may be disarranged when adjusting these two weights for each output. As a result, a reduction of data loss and improved network stability will be achieved by setting up the other relevant weights.

Gradient magnitudes at initialization for layer 55 of a network with and without BN – [Source](https://arxiv.org/pdf/1806.02375.pdf)

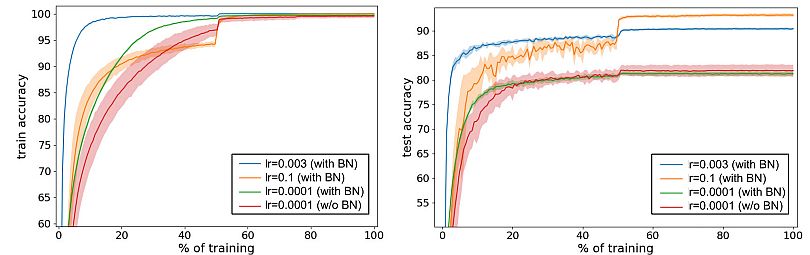
Commonly, CV experts apply batch normalization before the layer’s activation. It is often used in conjunction with other regularization functions. Also, deep learning methods, including [image classification](https://viso.ai/computer-vision/image-classification/), [natural language processing](https://viso.ai/deep-learning/natural-language-processing/), and machine translation utilize batch normalization.

**Batch Normalization in CNN Training**

##### **Internal Covariate Shift Reduction**

Google researchers Sergey Ioffe and Christian Szegedy [defined the internal covariate shift](https://arxiv.org/pdf/1502.03167.pdf) as a change in the order of network activations due to the change in network parameters during training. To improve the training, they aimed to reduce the internal covariate shift. Their goal was to increase the training speed by optimizing the distribution of layer inputs as training progressed.

Previous researchers (Lyu, Simoncelli, 2008) applied statistics over a single training example, or, in the case of image networks, over different feature maps at a given location. They wanted to preserve the information in the network. Therefore, they normalized the activations in a training sample relative to the statistics of the entire training dataset.

Training and testing phases using a ResNet-50 with BN on [ImageNet](https://viso.ai/deep-learning/imagenet/" \t "_blank) – [Source](https://arxiv.org/pdf/2209.14778.pdf)

The[gradient descent](https://viso.ai/computer-vision/gradient-descent/) optimization doesn’t take into account the fact that the normalization will happen. Ioffe and Szegedy wanted to ensure that the [network always produces activations](https://viso.ai/deep-learning/artificial-neural-network/) for parameter values. Due to the gradient loss, they applied the normalization and calculated its dependence on the model parameters.

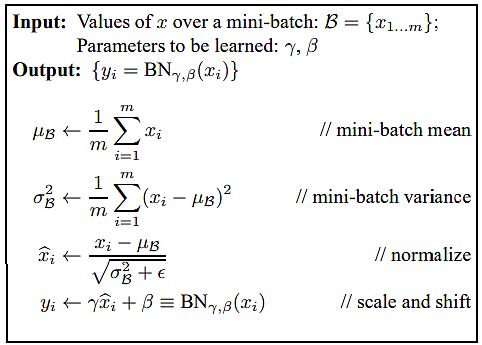
##### **Training and Inference with Batch-Normalized CNNs**

Training can be more efficient by normalizing activations that depend on a mini-batch, but it is not necessary during inference. (Mini-batch is a portion of the training dataset). The researchers needed the output to depend only on the input. By applying moving averages, they tracked the accuracy of a model, while it trained.

Normalization can be done by applying a linear transformation to each activation, as means and variances are fixed during inference. To batch-normalize a CNN, researchers specified a subset of activations and inserted the BN transform for each (Algorithm below).

Authors considered a mini-batch B of size m. They performed normalization to each activation independently, by focusing on a particular activation x(k) and omitting k for clarity. They got m values for each activation in the mini-batch:  
B = {x1…m}  
They denoted normalized values as x1…m, and their linear transformations were y1…m. Researchers have defined the transform  
BN γ,β : x1…m → y1…m

to be the Batch Normalizing transform. They conducted the BN Transform algorithm given below. In the algorithm, σ is a constant added to the mini-batch variance for numerical stability.

Batch Normalizing Transform, applied to activation x over a mini-batch – [Source](https://arxiv.org/pdf/1502.03167.pdf)

The BN transform has been added to a network to manipulate all activations. By y = BN γ,β(x), researchers indicated that the parameters γ and β should be learned. However, they noted that the BN transform does not independently process the activation in each training example.

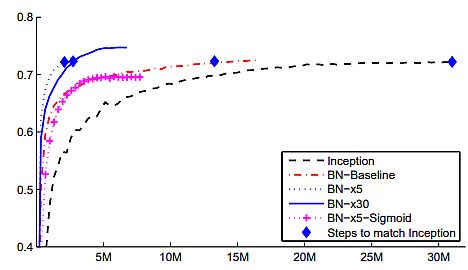
Consequently, BN γ,β(x) depends both on the training example and the other samples in the mini-batch. They passed the scaled and shifted values y to other network layers. The normalized activations xb were internal to the transformation, but their presence was crucial. The distributions of values of all xb had the expected value of 0 and the variance of 1.

All layers that previously received x as the input – now receive BN(x). Batch normalization allows for the training of a model using batch gradient descent, or stochastic gradient descent with a mini-batch size m > 1.

##### **Batch-Normalized Convolutional Networks**

Szegedy et al., (2014) used batch normalization to create a new Inception network, trained on the [ImageNet](https://viso.ai/deep-learning/imagenet/) [classification](https://viso.ai/computer-vision/image-classification/) task. The network had a large number of convolutional and pooling layers.

* They included a SoftMax layer to predict the image class, out of 1000 possibilities. Convolutional layers include ReLU as the nonlinearity.
* The main difference between their CNN was that the 5 × 5 [convolutional layers were replaced](https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/" \t "_blank) by two consecutive layers of 3 × 3 convolutions.
* By using batch normalization researchers matched the accuracy of Inception in less than half the number of training steps.
* With slight modifications, they significantly increased the training speed of the network. BN-x5 needed 14 times fewer steps than Inception to reach 72.2% accuracy.

Validation accuracy of Inception and its batch-normalized variants – [Source](https://arxiv.org/pdf/1502.03167.pdf)

By increasing the learning rate further (BN-x30) they caused the model to train slower at the start. Still, it was able to reach a higher final accuracy. It reached 74.8% after 6·106 steps, i.e. 5 times fewer steps than required by Inception.

### Benefits of Batch Normalization

Batch normalization brings multiple benefits to the learning process:

* **Higher learning rates.** The training process is faster since batch normalization enables higher learning rates.
* **Improved generalization.** BN reduces [overfitting](https://viso.ai/computer-vision/what-is-overfitting/) and improves the model’s generalization ability. Also, it normalizes the activations of a layer.
* **Stabilized training process.** Batch normalization reduces the internal covariate shift that occurs during training, improving the [stability of the training process](https://viso.ai/deep-learning/model-training-errors/). Thus, it makes it easier to optimize the model.
* **Model Regularization.** Batch normalization treats the training example together with other examples in the mini-batch. Therefore, the training network no longer produces deterministic values for a given training example.
* **Reduced need for careful initialization.** Batch normalization decreases the model’s dependence on the initial weights, making it easier to train.

### What’s Next?

Batch normalization offers a solution to address challenges with training deep neural networks for computer systems. By normalizing the activations of each layer, batch normalization allows for smoother and more stable optimization, resulting in faster convergence and improved generalization performance. Because it can mitigate issues like internal covariate shifts it enables the development of more robust and efficient neural network Architecture

8.Explain between MAX POOLING and AVERAGE POOLING is number eight.

#### ****Max Pooling****

1. Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.
2. This can be achieved using MaxPooling2D layer in keras as follows:  
   **Code #1 : Performing Max Pooling using keras**

* Python3

|  |
| --- |
| import numpy as np  from keras.models import Sequential  from keras.layers import MaxPooling2D    # define input image  image = np.array([[2, 2, 7, 3],                    [9, 4, 6, 1],                    [8, 5, 2, 4],                    [3, 1, 2, 6]])  image = image.reshape(1, 4, 4, 1)    # define model containing just a single max pooling layer  model = Sequential(      [MaxPooling2D(pool\_size = 2, strides = 2)])    # generate pooled output  output = model.predict(image)    # print output image  output = np.squeeze(output)  print(output) |

1. **Output:**

[[9. 7.]

[8. 6.]]

### ****Average Pooling****

1. Average pooling computes the average of the elements present in the region of feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.
2. **Code #2 : Performing Average Pooling using keras**

* Python3

|  |
| --- |
| import numpy as np  from keras.models import Sequential  from keras.layers import AveragePooling2D    # define input image  image = np.array([[2, 2, 7, 3],                    [9, 4, 6, 1],                    [8, 5, 2, 4],                    [3, 1, 2, 6]])  image = image.reshape(1, 4, 4, 1)  # define model containing just a single average pooling layer  model = Sequential(      [AveragePooling2D(pool\_size = 2, strides = 2)])    # generate pooled output  output = model.predict(image)    # print output image  output = np.squeeze(output)  print(output) |

1. **Output:**

[[4.25 4.25]

[4.25 3.5 ]]

9. What is the purpose of the POOLING LAYER?

* Dimensionality reduction: The main advantage of pooling layers is that they help in reducing the spatial dimensions of the feature maps. This reduces the computational cost and also helps in avoiding overfitting by reducing the number of parameters in the model.
* Translation invariance: Pooling layers are also useful in achieving translation invariance in the feature maps. This means that the position of an object in the image does not affect the classification result, as the same features are detected regardless of the position of the object.
* Feature selection: Pooling layers can also help in selecting the most important features from the input, as max pooling selects the most salient features and average pooling preserves more information.

10. Why do we end up with Completely CONNECTED LAYERS?

The fully connected layer, also known as the dense layer, plays a crucial role in convolutional neural networks (CNNs) and is an essential component of the network architecture. Its purpose is to capture global patterns and relationships in the input data by connecting every neuron from the previous layer to every neuron in the fully connected layer. This layer is typically placed at the end of the CNN, following the convolutional and pooling layers.

The primary function of the fully connected layer is to perform high-level reasoning and decision-making based on the features extracted by the preceding layers. It accomplishes this by learning complex non-linear mappings between the input and output data. Each neuron in the fully connected layer receives inputs from all the neurons in the previous layer and produces an output by applying a set of weights and biases, followed by an activation function.

By connecting every neuron to every other neuron in the fully connected layer, the network is able to learn intricate relationships and dependencies in the data. This allows the model to make predictions based on a combination of different features rather than relying solely on individual features. The fully connected layer acts as a powerful feature extractor, transforming the learned features into a format that can be used for classification or regression tasks.

To illustrate the role of the fully connected layer, consider a CNN trained to classify images of handwritten digits. The convolutional layers extract low-level features such as edges, corners, and textures, while the pooling layers reduce the spatial dimensions of the feature maps. The fully connected layer then takes these abstracted features and combines them to make predictions about the digit shown in the image. For example, it might learn that a combination of curved lines, loops, and closed shapes indicates the presence of a particular digit.

In addition to its feature extraction capabilities, the fully connected layer also contributes to regularization and model capacity control. The large number of parameters in the fully connected layer enables the network to learn complex representations, but it also increases the risk of overfitting. To mitigate this, regularization techniques such as dropout or L2 regularization can be applied to the fully connected layer, preventing the network from relying too heavily on any single connection.

The fully connected layer in a CNN is responsible for capturing global patterns and relationships in the input data by connecting every neuron from the previous layer to every neuron in the fully connected layer. It performs high-level reasoning and decision-making based on the learned features and contributes to regularization and model capacity control.

11. What do you mean by PARAMETERS?

* Relevance to the position: The algorithm will look for keywords and skills that match the job description.
* Education and experience: The algorithm will consider the candidate's education level and the number of years of work experience they have.
* Job titles: The algorithm will look at the candidate's previous job titles to determine if they have the necessary experience for the position.
* Certifications and licenses: The algorithm will take into account any relevant certifications or licenses the candidate holds.
* Accomplishments: The algorithm will look for specific accomplishments or achievements the candidate has accomplished in their career.
* Formatting: The algorithm will also take into account the formatting and structure of the resume, including the use of bullet points and white space.

12. What formulas are used to measure these PARAMETERS?

Growth is the most fundamental characteristic of living beings. It is characterised by an irreversible, permanent increase in the size of

* an individual cell
* an organ or its parts
* an organism

over a period of time.  
  
**Parameters for growth:**  
  
Growth is measured by a number of parameters like increase in cell number, dry weight, fresh weight, length, area and volume etc.

* Number: Measuring the increase in the number of cells is a parameter to measure the growth. This is easily observed in algae, plant cells growing in a culture.
* Weight: The weight of the plant helps us in determining the growth in plants. There are two types of weight taken into consideration:
  + Fresh weight: Weight of tissues along with water content.
  + Dry weight: Weight of tissues without the water content.
* Length: Increase in the length due to cell division and elongation represents growth, for example, pollen tube of the germinating pollen.