EECS 182/282 - Designing, Visualizing, and Understanding Deep Neural Networks

Lecturer: Anant Sahai

September 15, 2022

Scribe: Hasitha Sithadara Wijesuriya

1 Momentum and Adaptive Gradient Descent Methods

Vanilla gradient descent has many benefits, but speed is not one of them. The reason behind that is the constraint on the learning rate. This effect can be illustrated by solving the simple scalar equation given in equation 1 which considers the squared loss as the loss function ($L(w)$). We want to minimize $L(w)$ over w until we get as close as possible to the ground truth (y).

$$\begin{aligned} &\sigma w = \hat{y} \\ &\min_w L(w) = (y - \sigma w)^2 \end{aligned} \tag{1}$$

Gradient descent step at k^{th} step is given in 2 with a learning rate (η).

$$\begin{aligned} w_{k+1} &= w_k - \eta \nabla_w L(w^k) \\ w_{k+1} &= w_k + 2\eta\sigma(y - \sigma w_k) \\ w_{k+1} &= (1 - 2\eta\sigma^2) w_k + 2\eta\sigma y \\ \left(w_{k+1} - \frac{y}{\sigma}\right) &= (1 - 2\eta\sigma^2)\left(w_k - \frac{y}{\sigma}\right) \end{aligned} \tag{2}$$

Then the w^{k+1} can be written in terms of the initial guess of w_0 by considering the pattern in equation 2, as below (3)

$$w_{k+1} = (1 - 2\eta\sigma^2)^{k+1}\left(w_0 - \frac{y}{\sigma}\right) + \frac{y}{\sigma} \tag{3}$$

In order to make sure the (3) is recurrence stable, $-1 < (1 - 2\eta\sigma^2) < 1$ this condition should be satisfied. From that, we get a constraint $\eta < \frac{1}{\sigma^2}$. Figure 1 shows the behavior of gradient descent update with the choice of η. As the η is small enough, the solution converges to the optimal solution but takes a lot of iterations. But when η passes some value, it shows an oscillatory behavior, and after increasing it further, it starts to diverge from the solution.

A slight modification called momentum for the gradient descent is applied to solve these problems.

1.1 Momentum

The idea of momentum is finding a safe way to make the η bigger. We know that the dimensions with larger singular values start to oscillate quickly. So the thought is to somehow low pass filter (LPF) those directions that would otherwise oscillate if we take smaller steps. By doing that, instead of moving in the direction of the gradient, the update is moved towards the direction of the average gradient. The functioning of the simplest LPF is where this idea of averaging originally came from, Figure 2.

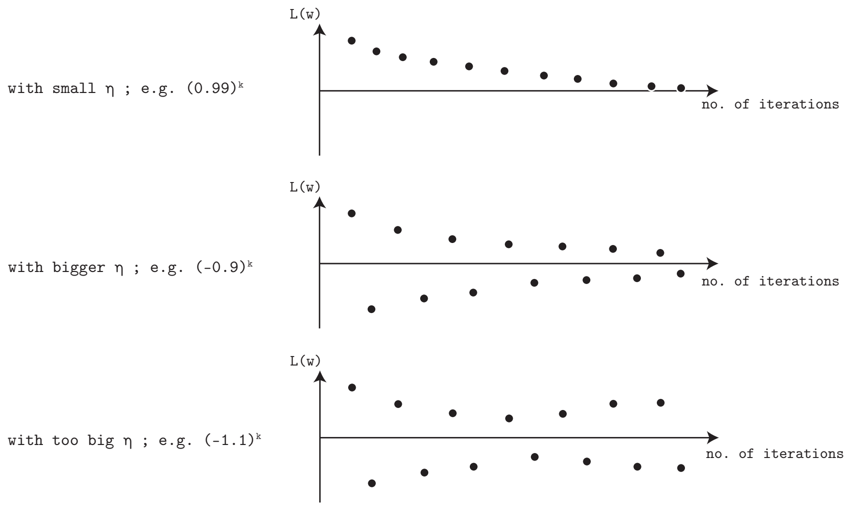

Figure 1: Gradient descent update with the choice of η

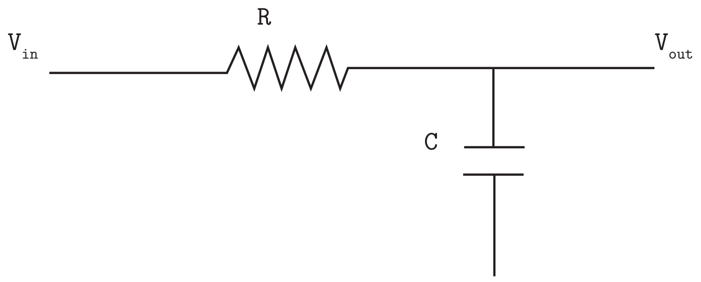

Figure 2: Simplest Low Pass Filter (LPF)

We can solve the first order differential equation that governs the LPF on 2 by equation 4 with a dummy variable τ. The exponential weighted average part is taken as $h(t-\tau)$, which is the impulse response that defines the filter. Since $h(t-\tau)$ is an exponentially weighted average of the V_{in}, the dying exponential has lesser weight as the τ gets further into the past.

$$\begin{aligned} V_{out}(t) &= \int_{-\infty}^{t} V_{in}(\tau) \frac{e^{-\frac{1}{RC}(t-\tau)}}{RC} d\tau \\ V_{out}(t) &= \int_{-\infty}^{t} V_{in}(\tau) h(t-\tau) d\tau \end{aligned} \tag{4}$$

This is also known as convolution integral as what it does is sliding along the input and taking averages by re-centering the input to t. And it integrates to 1 as it expresses a normalized average. In the discrete-time point of view, we can write 4 as a sum, equation 5. The geometrically dying exponential ($h(t-\tau)$ can be written with normalization parameter β as shown in 5. $\beta(1-\beta)^{t-\tau}$ makes sure that sums to 1.

$$V_{out}(t) = \sum_{-\infty}^{t} V_{in}(\tau)\beta(1-\beta)^{t-\tau} \tag{5}$$

This discrete time solution is also the solution to a first-order difference equation. Instead of input V_{in} we can plug in the gradients of $L(w)$ at current step, equation 6. β controls the averaging, as closer it gets to zero, the more averaging we get through the recurrence relationship of past gradients given by a_k.

$$a_{k+1} = (1-\beta)a_k + \beta \nabla_w L(w^k) \tag{6}$$

This momentum term a_{k+1} is used to update the $(k+1)^{st}$ gradient update rather than directly using the gradient as in vanilla gradient descent, equation 7.

$$w_{k+1} = w_k - \eta a_{k+1} \tag{7}$$

In this case, we have two hyper-parameters that can be adjusted, β & η. With the appropriate choice of β, we can increase the η than the limit imposed in vanilla gradient descent. The slower directions converge faster by allowing that increment, increasing the overall speedup.

The current gradient used in 6 can have two interpretations depending on which variant we want to use. The two variants are "Vanilla momentum" and "Nesterov Momentum", equation 8.

$$\begin{aligned} &\text{"Vanilla" Momentum } \nabla L(w_t) \\ &\text{"Nesterov" Momentum } \nabla L(w_t - \eta(1-\beta)a_t) \end{aligned} \tag{8}$$

The vanilla momentum uses the gradient where the current weights are. In the Nesterov momentum, we take advantage of the fact where we are going by peeking into the future, as given from the first part of the momentum term (6). This is because we already know where we are going to end up, and by taking that new information to the calculation of the gradient at this step, we can take a bit of an advantage on learning.

1.2 Adaptive (e.g. Adam) Methods

The simple perspective of the origin of the adaptive methods comes from the fact that even with momentum, the largest singular value and the smallest singular value are the ones that govern the process. Values in between are not relevant for setting up the parameters. So the simple idea is that instead of having a single step size, use different step sizes (η_i) along different dimensions of the parameter vector. In vanilla gradient descent, we go down in the steepest gradient direction. But in the adaptive methods, we no longer go in the steepest direction but still in a downward direction.

The formulation of Adam's method where different step sizes in a different direction are shown in equation 9. In the large gradient directions, it takes smaller steps and vice versa. That way, it takes evenly sized steps along different directions. To track the size of the gradient, a vector $\vec{V}$ is introduced, which depends on the element-wise product of the gradient vector. Then the gradient update for the i^{th} parameter is computed with the square root of elements of $\vec{V}$ to make the units right. A constant (ϵ) is added to the denominator to make sure it is stable.

$$\begin{aligned} \vec{a}_{k+1} &= (1-\beta)\vec{a}_k + \beta \nabla_w L(w^k) \\ \vec{V}_{k+1} &= (1-\beta')\vec{V}_k + \beta' \nabla_w \begin{pmatrix} \cdots \\ \left(\frac{\partial L(w)}{\partial w_i}\right)^2 \\ \cdots \end{pmatrix} \\ w_{k+1}[i] &= w_k[i] - \eta \frac{a_{k+1}[i]}{\sqrt{V_{k+i}[i]} + \epsilon} \end{aligned} \tag{9}$$

Typically $\beta' << \beta$, to make sure the average size of the gradient is over a longer period than the average gradient. Otherwise, it would create oscillatory movement when it is closer to the convergence. This can be observed by the numerator and the denominator in the weight update step. Also, this creates another challenge when the β' is very small (which is typically the case). In the start, this creates problems as the $\vec{V}$ is going to be small for a while until it builds up. This is partially protected by the (ϵ). The solution for this problem is normalizing, as shown in equation 10. This normalization dies away as the iterations keep going but solve the problem at the start.

$$
\begin{aligned}
\hat{\vec{a}}_{k+1} &= \frac{\vec{a}_{k+1}}{1-\beta^k} \\
\hat{\vec{V}}_{k+1} &= \frac{\vec{V}_{k+1}}{1-(\beta')^k} \\
w_{k+1}[i] &= w_k[i] - \eta \frac{\hat{\vec{a}}_{k+1}[i]}{\sqrt{\hat{\vec{V}}_{k+1}[i]}+\epsilon}
\end{aligned} \tag{10}
$$

2 Convolution neural networks

Convolutional Neural Networks (CNN) is a class of neural networks that is most commonly used to analyze images. Why do we need CNN's in the first place? Figure 3 shows the naive approach to analyzing images. As the input is high dimensional and it itself is very big. For an image with 128×128 resolution with three color channels, it has around 50,000 pixels. If we approach this problem with a fully connected network, we get roughly around 3 million weights for the first layer. And it is very large for just one layer. That is why people adopt the idea of convolution for analyzing images with fewer weights.

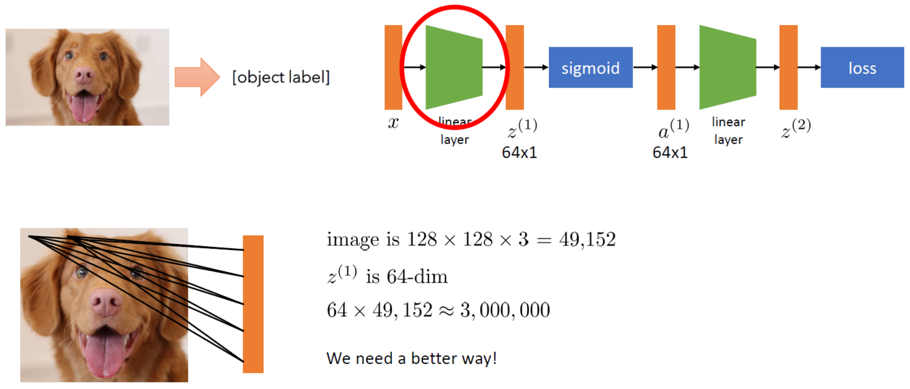

Figure 3: Naive approach with images [Lev21]

As the goal of any neural network, the objective is to learn the pattern in our data. And our objective is to build an architecture toward the kind of patterns we expect to see. There are several ways that CNN's achieves this objective, as shown below.

- Respecting ”locality”

In the learned functions, the pixels that are near each other are important to figure out the relationship of the learned function. This idea is manifest as the convolution structure itself. Figure 4 shows the local features in small neighborhoods inside of an image. If we learn a very narrow field of view of an image, we sometimes see edges as the edges are the smallest local signature that we can observe. when we make the network deeper, it sees the larger and larger parts of the image, and we can maybe start observing parts of the objects.

- Respecting ”invariances”

This idea can be simply explained as the learned function should not be affected by the translation of objects within the dataset. For example, in the case of a classifier, it should give the same prediction even if the object has moved within the input image. This idea is manifested by weight sharing and data augmentation during the training process. Figure 5 shows the idea of weight sharing. If we have different blocks with different weights for every patch of the image, that doesn't capture the idea of invariance. If we use the same patch to cover the whole image by shifting the patch by a pixel at a time, it is much more tractable to lift it to the 64 channels. And it is a much more reasonable number of parameters.

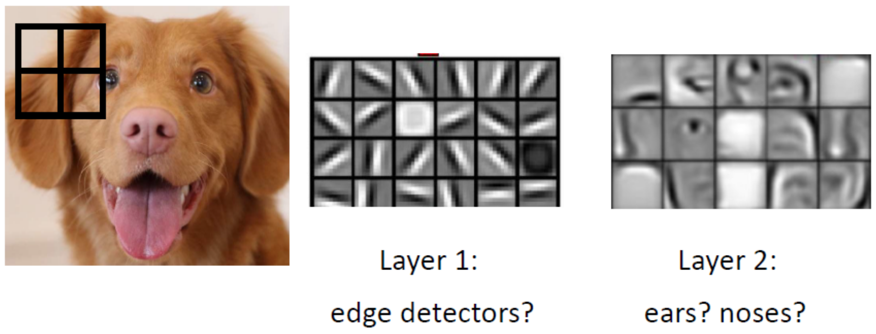

Figure 4: Locality in CNN's[Lev21]

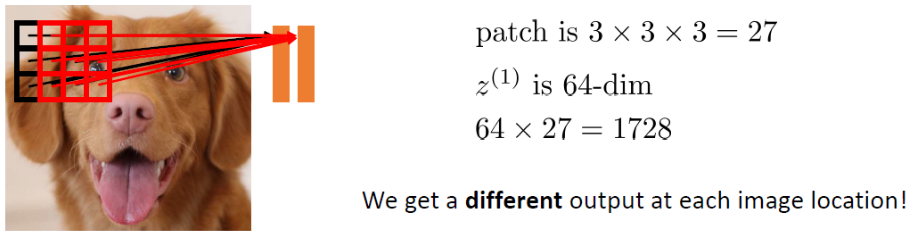
patch is $3 \times 3 \times 3 = 27$
$z^{(1)}$ is 64-dim
$64 \times 27 = 1728$
We get a **different** output at each image location!

Figure 5: Idea of weight sharing[Lev21]

- Support hierarchical structure and multi-resolution understanding

This idea is explained as the patterns that we are trying to learn are visible not at the level of local level but only when we see the whole image. For example, the edges can be identified at the local level, but the parts of the objects we are considering are only visible at another level, and the object themselves is only visible at the global level. This manifests as the combination of depth of the network, downsampling with depth (stride and pooling), and lifting from pixel space to abstract level by increasing the number of channels with depth.

- ”Room to play” & ”redundancy”

This idea is during the learning process, if things are too tightly constrained, we are usually stuck in local minima, and it is hard to move on from there. As the learning process is going, we should be able to have room to build new features to get it solved. In other words, we should have redundancy to work on different features as we progress with our learning. This manifests as the combination of adding more channels and a particular way of learning called dropouts.

3 What we wish this lecture also had to make things clearer?

- It would have been great if we could see the real-time demonstration of the effect of momentum on gradient descent as shown on this website. https://distill.pub/2017/momentum/
- Also, using the tools in https://alexlenail.me/NN-SVG/LeNet.html, we could have seen a clear demonstration of CNNs that covers the weight sharing, number of channels per layer, etc.

References

[Lev21] Sergey Levine. Lecture notes in designing, visualizing and understanding deep neural networks. https://cs182sp21.github.io/static/slides/lec-6.pdf, January 2021.