CMSC 491/691: Computer Vision **Fall 2023**

Lecture 1: 2023-09-06 Title

Lecturer: Tejas Gokhale *Scribe: Emmanuel Ugwuabonyi*

1.1 Recap of Last Lecture

In the last lecture, we learned that as the neural network gets larger, incremental learning strategy or gradient descent enables us to learn the weight parameters. This can be stated as follows: Given several examples $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ and a perceptron $\hat{y} = wx$, Modify weight w such that $\hat{y}$ gets '**closer**' to y. Then the Loss function is used to compute the value of the difference between y and $\hat{y}$. They include the Absolute or L1 Loss $l(\hat{y}, y) = |\hat{y} - y|$
the Euclidean or L2 Loss $l(\hat{y}, y) = |\hat{y} - y|^2$
Zero-one Loss $l(\hat{y}, y) = 1|\hat{y} - y|$ and
Hinge Loss $l(\hat{y}, y) = max(0, 1 - y.\hat{y})$
but we will not use the 0-1 loss because it is not differentiable.

1.2 Gradient descent

Gradient descent allows us to calculate the loss at each step. Once we know our gradient, we can update the weight using the update rule $w = w - \nabla w$. In order words the weight update rule allows us to update the weight with each new example.

To calculate the loss in Table 1.1

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

Table 1.1: data

Using the data in the first row of the table $w = 10.1/10 = 1.01$, $\hat{y} = 10.1$, $y = 10.1$, therefore the loss $l = |\hat{y} - y| = 0$ But if we apply it to the second example, the loss increases, we then need to update the weight so that the prediction $\hat{y}$ gets closer to the true output y

The Loss $L = 1/2(y - \hat{y})^2$
The rate of change of loss with respect to weight $\frac{dL}{dw} = 1/2(w^2x^2 - 2wxy + 2y^2) = x(wx - y) = -(y - \hat{y})x = \nabla w$.

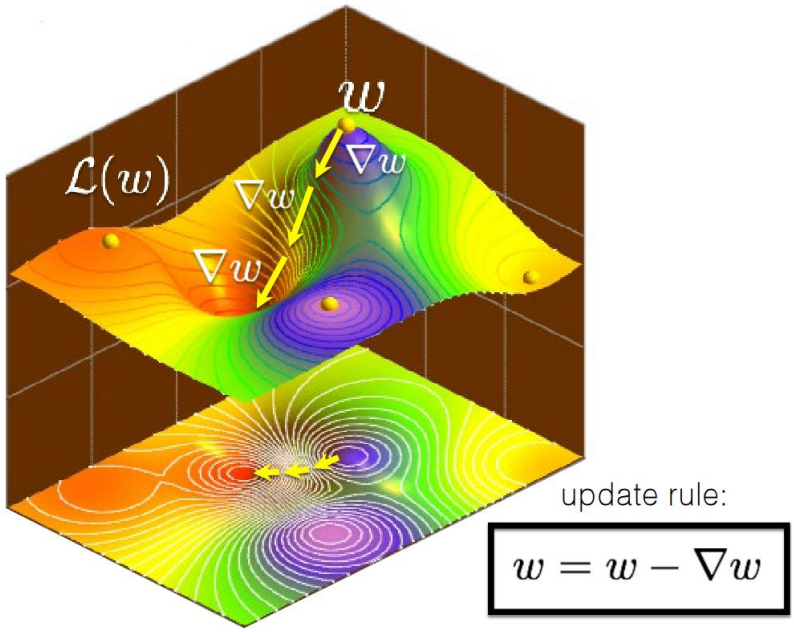

Figure 1.1: Gradient Descent

The idea behind the **gradient descent** is to
For each sample x_i, y_i
1. Compute the forward pass $\hat{y} = wx_i$
2. Calculate the loss $L = 1/2(y - \hat{y})^2$
3. Find the gradient which is the rate of change of loss ∇w
4. Update the weight $w = w - \nabla w$

Multi-Layered perceptron

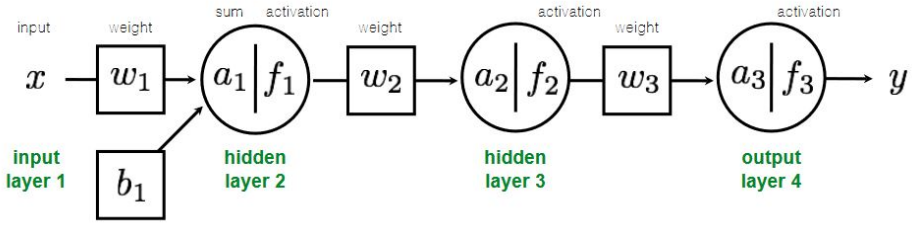

Figure 1.2: multi-layer perceptron

The above neural network can be computed as follows: 1: compute the linear function $a_1 = w.x_1 + b_1$
2: $a_2 = w_2.f_1(w.x_1 + b_1)$
3: $a_3 = w_3.f - 2(w_2.f_1(w.x_1 + b_1))$
$y = f_3(w_3.f - 2(w_2.f_1(w.x_1 + b_1)))$
where f_1, f_2, f_3 are activation functions.

To update the weight for each of the parameters, we compute the rate of change of loss with respect to each weight. That is we compute $\frac{dL}{dw_3}, \frac{dL}{dw_2}, \frac{dL}{dw_1}, \frac{dL}{db}$.
This can be done using the chain rule. For example, to compute $\frac{dL}{dw_1}$ we use

$$\frac{dL}{dw_3} = \frac{dL}{df_3}.\frac{df_3}{da_3}.\frac{da_3}{dw_3} = -\eta(y-\hat{y})\frac{df_3}{da_3}.\frac{da_3}{dw_3}$$

1.2.1 Learning Rate

Learning rate η defines the adjustment in the weights of our network with respect to the loss gradient descent.

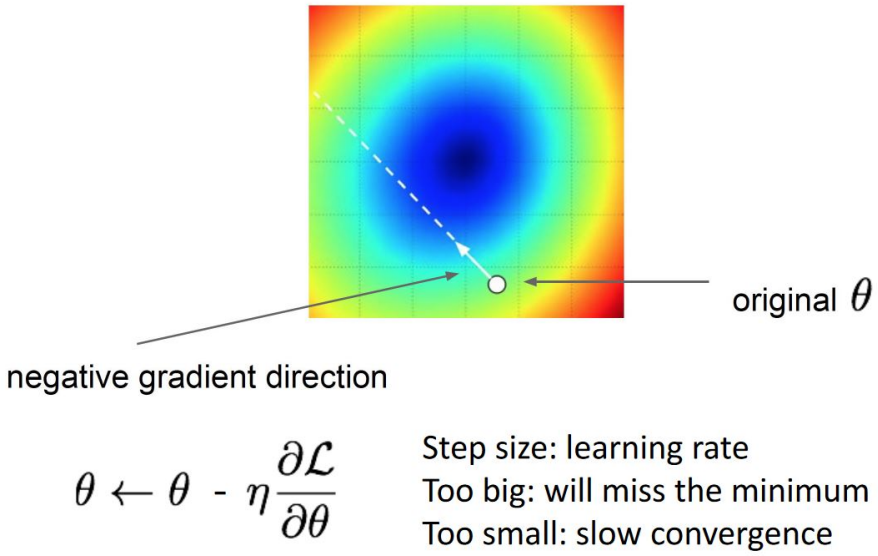

Figure 1.3: Learning rate

.

- If the η is too small say 0.001, it will take time to get to the minimum because it will move slowly.
- If the η is too large, you will never get to the minimum.

Hence it is important to choose the right learning rate for better convergence at . We can also control the learning rate using Adaptive learning rate.
SGD + Momentum is give a better update especially in training CNNs

1.3 Convolutional Neural Networks (CNNs)

CNNs changed a lot of things for Computer Vision. Convolutions are useful for vision. So we can learn features like edges or textures using convolutions hence it is a nice idea to use CNNs while training the neural networks especially for image data. It helps neural networks to become better and efficient.CNNs in 2012 was a major breakthrough in image classification, it used the model known as AlexNet.
Before deep learning the process usually follows: given an input image, extract the features, concatenate into a vector and apply a linear classifier such as SVM. The key challenge here is that when the image is flattened, we lose some structures. However, CNNs allow us to retain the image that is non-linear giving the 3D shape of the height, width and depth which is usually a 3 color channel(RGB). For instance a cifar10 dataset consists of $32x32x3$, that is height, width and depth respectively.

1.3.1 ConvNets

ConvNets are neural networks with 3D ativations and weight sharing. At every layer, you apply convolutions followed by an activation function.

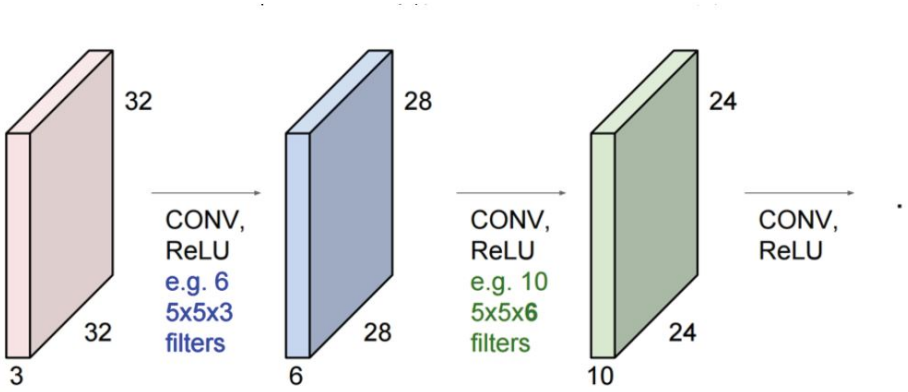

Figure 1.4: Enter Caption

1.3.2 3D Activations

All neural networks are arranged in 3 dimensions; height, width and depth. From the above diagram,

- The input in 3D Activations is a $3x32x32$ input image
- Assuming that we have a $3x5x5$ filters, then the neuron depends on a $3x5x5$ chunk of the input.
- Here, we **convolve** the filter with the image meaning slide over the image spatially computing the dot products.
- the neuron also has a $3x5x5$ set of weights and bias. i.e $w^T x + b$

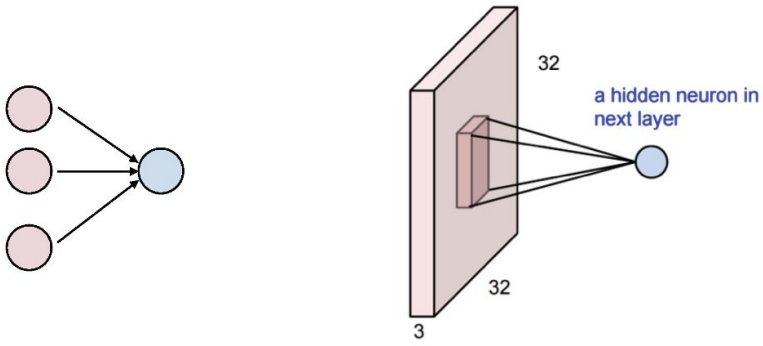

Figure 1.5: (A) 1D Activation (B) 3D Activation

1.3.3 Padding

Padding is used to add extra pixels around the input image or feature map to maintain spatial dimensions during the convolution operation.

1.3.4 Stride

Strides is used to reduce the size of the output. When the stride is 1, the filter moves across the input matrix 1 pixel at a time. When the stride is 2, the filter jumps 2 pixels at a time and so on. This will give lesser number of rows and columns in the output. In general if you want to control the intermediate layer, you can change the value of strides or padding.

1.3.5 Pooling

Pooling creates a smaller representation while retaining the most important information.
The **'Max'** operation is the most common. It calculates the maximum value for patches of a feature map, and uses it to create a downsampled (pooled) feature map. In general, max pool means divide your output into a grid of 2x2 windows and in each window select the maximum value.
There is also an **average** pooling but it is not necessarily a good choice because it smooths out the image and hence the sharp features may not be identified. **Back propagation rule for max pooling**

- In the forward pass, store the index that took the max.
- The backprop gradient rule is the input gradient at that index.

ConvNet Example[hbt!]

From the image (Figure 1.7), the dark area shows that ReLU zero out the negative gradient.

Generally, training a neural network involves:

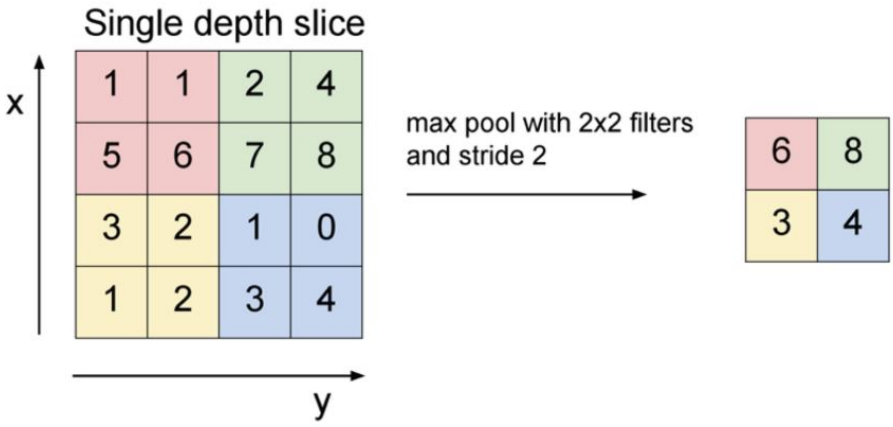

Figure 1.6: Enter Caption

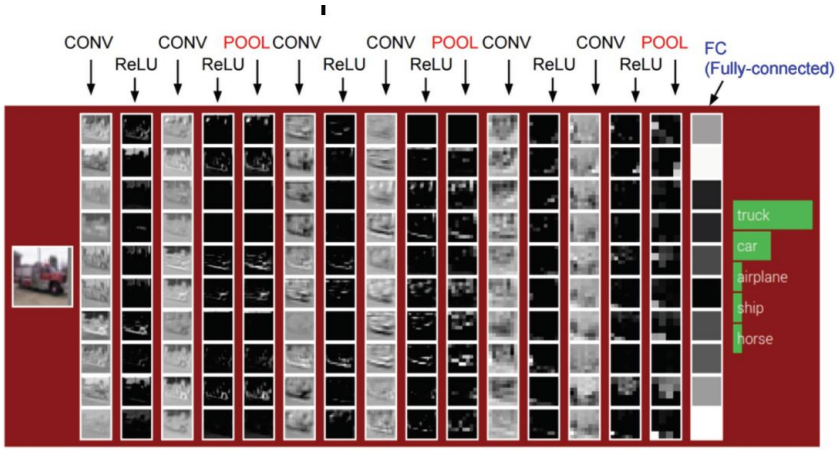

Figure 1.7: Convnet visualization

1. Split and process your data.
2. Choose your network architecture.
3. Initialize the weights.
4. Find a learning rate and regularization strength.
5. Minimize the loss and monitor progress.
6. Fiddle with knobs

Mini-Batch Gradient Descent

- Sample of a batch of training data(100 images)
- Forward Pass: compute loss
- Backward Pass: compute gradient.

- update all parameters

1.3.6 Regularization

This is used to reduce overfitting. Regularization can be L1, L2, Elastic Net, MAx Norm and Dropout.

1.3.7 Overfitting

This involves modeling noise in the training set instead of the **'true'** underlying relationships. Here, the model is memorizing the data it has seen and is unable to generalize to unseen examples

1.4 References

Lecture Slides