**Weight Decay and Batch Normalization**

**Author: Haotian Jiang**

**1. Weight decay**

When calculating loss for gradient descent, we want to add a regularization to avoid overfitting. Suppose the original loss is $L\_{0}$, the loss after adding the regularization is $L$, and the number of parameters is n, we can get:

$$L=L\_{0}+\frac{λ}{2n}\sum\_{i=1}^{n}w\_{i}^{2}$$

When we calculate the gradient of loss to a weight $w\_{i}$, we use:

$$\frac{∂L}{∂w\_{i}}= \frac{∂L\_{0}}{∂w\_{i}}+\frac{λ}{n}w\_{i}$$

We call $λ$ weight decay, it indicates the strength of the regularization.[1]

When using datasets where the images have high similarities (such as ImageNet which only has 1000 classes), we can see that the variance between two training samples can be relatively small. So, if we use only one sample for each training iteration, it is reasonable that we calculate the current gradient mostly based on the gradient in the former iteration because of the similarity between training samples. So, we can calculate the current gradient as:

$$d\_{t}=ρ∙d\_{t-1}+\frac{∂L}{∂w\_{i}}$$

Here, $d\_{t}$ is the gradient of $w\_{i}$ in the current iteration, $d\_{t-1}$ is the gradient of $w\_{i}$ in the last iteration, and $ρ$ is called momentum. Since the similarity between samples is usually big, we can set $ρ$ to a big value like 0.9. Finally, we can use the computed gradient to update our weight using this formula:

$$w\_{i,t}= w\_{i,t-1}-lr∙d\_{t}$$

Here, $w\_{t}$ is the updated $w\_{i}$, $w\_{t-1}$ is $w\_{i}$ in last iteration, and $lr$ is the learning rate. Training using one sample for each iteration with weight decay is very similar to training using a mini-batch containing several samples for each iteration with stochastic gradient descent (SGD). The former method updates the gradient every time a sample is processed while the latter method updates the gradient every time a mini-batch containing several samples is processed. The gradient changes smoothly in the former method while it changes abruptly in the latter method. By using weight decay, the process of updating gradient is very similar to a moving average in signal processing theories.

**2. Batch normalization**

In order to eliminate the repetition information in the dataset and preserve only the dimensions that are independent from each other, we need to perform whitening on the dataset. Batch normalization is an easy way to do whitening when training using mini-batches with stochastic gradient descent (SGD).

We can add a batch normalization layer before each layer in the network. A batch normalization layer is responsible of transforming the original input of the following layer to a new input. We can perform the following steps in each batch normalization layer.

Before training, we firstly calculate the mean and variance of each dimension of the inputs. For one layer, we can calculate the mean and variance using this formula:



Here, B indicates the mini-batch, so $B=\{x\_{1….m}\}$ where m is the number of samples in the mini-batch. $x\_{i}$ is a scaler which indicates a sample’s value in dimension k. In the formulas, k is omitted. We can see that the mean and variance is calculated across the samples in the mini-batch, so a mean and variance is calculated every time a new mini-batch comes. Then, we can get a new $\hat{x}\_{i}$ using:



Suppose $y\_{i}$ is the new input of the following layer, we can get it using:



Here, $γ$ and $β$ are scaling and shifting parameters respectively, and they are learned during the training process. We perform the steps above for all dimensions of the mini-batch in all the batch normalization layers of the network. After that we train the network, updating all the network weights and the $γ$ and $β$ in all batch normalization layers. After the training, we freeze all the network weights including $γ$ and $β$ for testing.

During testing, the mean and variance of each batch normalization layer is computed using:



Here, E[x] is the mean of all the samples in the training set. So, we can compute it by averaging the means of mini-batches. Var[x] is variance of all the samples in the training set. We can compute it by averaging the variances of mini-batches and transforming the biased estimation to unbiased estimation. To make computation easier, it is a good idea to store the means and variances of the mini-batches during the training process. Next, we can compute the new input for the following layer using:

[2]

Using batch normalization has many benefits. Firstly, if the mean of the inputs of layer is very big, the dynamic range of the network will be large, which makes training less efficient. For example, if we add a distribution with a small mean to a distribution will a large mean, the latter distribution changes little and the computed gradient will be very small, which adversely affects training. Second, if the variance of the inputs of a layer is big, the distribution of the inputs will be like a uniform distribution from which we cannot obtain useful information.

So, we want to transform large means and variances of the inputs to lower values. However, we don’t know the proper mean and variance values of the inputs for each layer. So, we need to train $γ$ and $β$ which indicate the proper mean and variance.

**Reference:**

[1] I. Loshchilov and F. Hutter, “Fixing weight decay regularization in Adam,” arXiv preprint arXiv:1711.05101, 2017.

[2] Ioffe, S. and Szegedy, C., “Batch normalization: accelerating deep network training by reducing internal covariate shift”, arXiv preprint arXiv:1502.03167, 2015.