DS-GA 1008
Deep Learning, Spring 2019
Homework Assignment 3
Yves Greatti - yg390
Read-Only Link to tex file

1. Fundamentals

1.1. Dropout

(a) The 2D dropout technique is implemented by torch.nn.Dropout2d(p=0.5, inplace=False)

(b) Deep neural networks, with non-linear "hidden" layers, will model almost perfectly complex relationships between the input and the correct output and there will be many different settings of the weight vectors. Each of these instance of trained neural network, will do worse on the test data than on the training data, and in essence they will overfit. With unlimited resources, the best way to "regularize" a network is to average the settings of all these weight vectors. Dropout is a cheap albeit efficient method of regularizing. Dropout provides an approximation to model combination in evaluating a bagged ensemble of exponentially many neural networks. It helps the network to not give too much importance to a particular feature. It helps to learn robust features which are more useful with many different random subsets. It also helps to reduce interdependence amongst the neurons, and limits the network ability to memorize very specific conditions during training.
Dropping out could apply to input and hidden units ,which mean they are temporarily removed from the network. In the simplest case, each unit is retained with a fixed probability p independent of other units, p is an hyperparameter which can be chosen using a validation set or is fixed. Typically, 0.5 for a hidden unit seems the optimal value for a wide range of networks and tasks and for the input units, the value is closer to 1 like 0.8.
In practice, for each minibatch a random binary mask is applied to all the input and hidden units of a layer, the mask is generated independently for each dropout layer. If a unit in a layer, is retained with a probability p during training, the outgoing outgoing weights of that unit are multiplied by p at test time: the output at test time is same as expected output at training time.
PyTorch torch.nn.Dropout implementation to keep the inference as fast possible scales the units using the reciprocal of the keep probability $\frac{1}{1-p}$ during training which yields the same result. 2D dropout in PyTorch performs the

same function as the previous one, however it drops the entire 2D feature map instead of individual unit. In early convolution layers adjacent pixels within each feature maps are strongly correlated, the regular dropout will not regularize the activations and, instead spatial dropout 2D helps in promoting independence between feature maps and should be preferred instead. Because dropout is a regularization technique, it reduces the capacity of the network, which leads to an increase of its size and the number of iterations (epochs) during training. However training time for each epoch is less. In very large datasets, regularization does not have a direct impact on the generalization error, in these cases, dropout could be less relevant. On very small training examples, dropout is less effective compared to **bayesian networks**. Dropout has inspired other approaches like **fast dropout**, or **drop out boosting** but none of them have outperformed its performances.

1.2. Batch Norm

(a) What does mini-batch refer to in the context of deep learning?
Mini-batch in the context of deep learning is related to the batch size of the data used by the gradient descent algorithm that splits the training data into small batches that are used to compute model error and update its parameters. It seeks to combine the effects of batch gradient descent and stochastic gradient descent. In batch gradient descent, the gradient is computed over the entire dataset. In stochastic gradient descent, the gradient is computed on a single instance of the dataset. On expectation, the gradient on a random sample will point to the same direction of the full dataset samples. SGD is more efficient and its stochastic property can allow the model to avoid local minima. SGD actually, helps generalization by finding "flat" minima on the training set, which are more likely to also be minima on the test set (see Theory of Deep Learning III: Generalization Properties of SGD). Minibatch will have a size ranging from 2 to 32 (see Revisiting Small Batch Training for Deep Neural Networks). Minibatch, compared to SGD, can still be parallelized.

(b) Batch normalization reduces the *"Internal Covariate Shift"* which is defined in Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, as the change in the distribution of network activations due to the change in network parameters during training: inputs to each layer are affected by the parameters of all preceding layers. Before batch normalization, saturated regime of non-linear activation resulting in vanishing gradient, were usually addressed by using Relu, small learning rates or care-

ful initialization of the weights. Batch normalization helps to avoid these issues by subtracting the mean and dividing by the batch standard distribution, normalizing the scalar feature to have a Gaussian distribution $\mathcal{N}(0, 1)$. In implementation, this technique usually amounts to insert the BatchNorm layer immediately after the fully connected layer or convolutional layer, and before non-linearities. Batch normalization, by preventing the model from getting stuck in the saturated regime of nonlinearities, enables higher learning rates and allows to achieve faster training. By whitening the inputs of each layer, BatchNorm regularizes the model removing or reducing the need for dropout. After the shift and scaling, two learnable parameters, γ and β, are used to avoid the network to undo the normalization and recover the original activations of the network. The output of a BatchNorm layer is given by:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} * \gamma + \beta$$

In PyTorch, by default, the elements of γ are sampled from a uniform distribution $\mathcal{U}(0, 1)$ and the elements of β are set to 0. The mean and standard deviation are computed over the mini-batches and each layer can keep running estimates using a momentum. The running averages for mean and variance are updated using an exponential decay based on the momentum parameter:

- running_mean = momentum * running_mean + (1 - momentum) * sample_mean (sample_mean is the new observed mean)
- running_var = momentum * running_var + (1 - momentum) * sample_var (sample_var is the new observed variance)

momentum=0 means that old information is discarded completely at every time step, while momentum=1 means that new information is never incorporated. For fully connected activation layers, BN is applied separately to each dimension (H, W) with a pair of learned parameters γ and β per dimension. For convolutional layers, BN is applied so that different elements of the same feature map, at different spatial locations, are normalized across the mini-batch. The parameters γ and β are learned per feature map. BN is a differentiable transformation and using the chain rules the gradient of loss can be explicitly formalized and it can be shown that backpropagation for BN is unaffected by the scale of its parameters and BN will stabilize the parameter growth.