CS 182/282A Designing, Visualizing and Understanding Deep Neural Networks
Spring 2021 Sergey Levine **Discussion 4**

This discussion will cover CNN architectures, batch normalization, weight initializations, ensembles and dropout.

1 Convolutional Neural Networks Architectures

We will survey the some most famous convolutional neural net architectures.

LeNet. Among the earlier CNN architectures, LeNet is the most widely known. LeNet was used mostly for handwritten digit recognition on the MNIST dataset. Importantly, LeNet used a series of convolutional layers, then pooling layers, followed by several fully connected (FC) layers.

AlexNet. The AlexNet architecture popularized CNNs in computer vision, when it won the ImageNet ILSVRC Challenge in 2012 by a large margin. AlexNet has a similar architectural design as LeNet, except that it is bigger (more neurons) and deeper (more layers). In addition, AlexNet demonstrated the benefits of using the ReLU activation and dropout for vision tasks, as well as the use of GPUs for accelerated training.

VGGNet This network was the runner-up in ILSVRC 2014 to GoogLeNet, and showed the benefit of (a) increasing the number of layers, and (b) using only convolutional operators stacked on each other. A downside is that this network has roughly 138 million parameters, so in general, consider using Residual Nets (see next item).

ResNet These networks use skip connections to allow inputs and gradients to propagate faster throughout the network (either forward or backwards). Residual networks were state of the art for image recognition results in mid-2016, and the general backbone is commonly used as of today. They have substantially fewer parameters than VGG. The exact number depends on what type of "ResNet-X" is used, where "X" represents the number of layers; PyTorch offers pretrained models for 18, 34, 50, 101, and 152. For reference, ResNet-152 should have about 60 million parameters.)

Problem: Vanishing Gradients in ResNet

How does skip connection in ResNet help solve the vanishing gradient problem?

2 Batch Normalization

The main idea behind Batch Normalization is to transform every sampled batch of data so that they have $\mu = 0$, $\sigma^2 = 1$. Using Batch Normalization typically makes networks significantly more robust to poor initialization. It based on the intuition that it is better to have unit Gaussian inputs to layers at initialization. However, the reason for why batch normalization works is not entirely understood, and there are conflicting views between whether Batch Normalization reduces covariate shift, improves smoothness over the optimization landscape, or other reasons.

In practice, when using batch normalization, we add a BatchNorm layer immediately after ecah FC or convolutional layer, either before or after the non-linearity. The key observation is that normalization is a relatively simple differentiable operation, so we do not add too much additional complexity in the network.

Noticeably, Batch Normalization proceeds by first computing the empirical mean and variance of some mini-batch B of size m from the training set.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} a_i$$
$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu_B)^2$$

Then, for a layer of network, each dimension, $a^{(i)}$ is normalized appropriately,

$$\bar{a}_i^{(k)} = \frac{a_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \epsilon}}$$

where ϵ is added for numerical stability.

In practice, after normalizing the input, we squash the result through a linear function with learnable scale γ and bias β, so, we have,

$$\bar{a}_i^{(k)} = \frac{a_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \epsilon}}\gamma + \beta$$

Intuitively, γ and β allows us to restore the original activation if we would like, and during training, they can learn other distribution that would be better initialization than standard Gaussian.

Problem: Examining the BatchNorm Layer

1. Draw out the computational graph of the BatchNorm layer
2. Given some *dout*, the derivative of the output of the BatchNorm layer, compute the derivatives with respect to input x and parameters γ, β

3 Ensembles

Definition 1 (Ensemble)**.** *Ensemble (bagging or boosting) group several models trained on the same task into a single model to aggregate predictions.*

Intuition The intuition for ensembles come from the recognition that neural networks have many parameters, often with high variance. Then, if we have multiple learners, we can average out the variance.

Ensemble Methods There are two ways we typically proceed with ensemble methods:

1. **Prediction Averaging**. Train N neural networks independently. Then, average their predictions (either probabilistically or by majority vote)
2. **Parameter Averaging**. Parameter averaging does not work in the same way as prediction averaging. Instead, we would only average over parameters from the context of snapshot ensembles, and average parameters over one trajectory, not over independent runs.

In practice, we do not need to reshuffle our dataset (and resample with replacement), since there is already a lot of randomness in neural network training form weight initialization, minibatch shuffling and SGD.

Making Ensemble Methods Faster Unfortunately, a downside to ensemble methods is that they can be very slow.

1. Only make classification layers (e.g., FC layers) ensembles.
2. Snapshot ensemble. Save out parameter snapshots over the course of SGD optimization and use each snapshot as a model.

4 Dropout

Definition 2 (Dropout)**.** *Dropout is a popular technique for regularizing neural networks by randomly removing nodes with probability* $1 - p_{keep}$ *in the forward pass. However, the model is unchanged at test time.*

Intuition Dropout can be thought of as representing an ensemble of neural networks, since each forward pass is effectively a different neural network, since random nodes are removed.

Activation Scaling A caveat about dropout is that we must divide the activation by p, since we do not change the model at test time, but we notice that none of our dimensions will then be forced to 0. Below is sample code to demonstrate how Dropout works in practice for a 3-layer network.

```
def dropout_train(X, p):
    """
    Forward pass for a 3-layer network.
    NOTE: For simplicity, we do not include backwards pass or parameter update

    X: Input
    p: Probability of keeping a unit active (e.g., higher p leads to less dropout)
    """
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p
    H1 *= U1 # Drop the activations
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p
    H2 *= U2 # Drop the activations
    out = np.dot(W3, H2) + b3
    return out

def predict(X):
    """ Forward pass at test time """
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
    return out
```

Problem: Dropout Review

Explain why Dropout could improve performance and when we should use it

5 Weight Initialization

One of the reasons for poor model performance can be attributed to poor weight initialization. In class, we discussed two types of weight initialization,

1. Basic initialization: Ensure activations are reasonable and they do not grow or shrink in later layers (for example, Gaussian random weights or Xavier initialization)
2. Advanced initialization: Work with the eigenvalues of Jacobians

Problem: Deriving Xavier Initialization

Let our activation be the `tanh` activation, which is approximately linear with small inputs (i.e., $\mathsf{Var}(a) = \mathsf{Var}(z)$, where z is the output of the activation followed by some linear layer). We furthermore assume that weights and inputs are i.i.d. and centered at zero, and biases are initialized as zero.
We would like the magnitude of the variance to remain constant with each layer. Derive the Xavier Initialization, which initializes each weight as,

$$W_{ij} = \mathcal{N}\left(0, \frac{1}{D_a}\right)$$

where D_a is the dimensionality of a

6 Aside: ReLU Activations and its Relatives

Definition 3 (ReLU Activation)**.** *ReLU Activation is defined as,* $ReLU(x) = \max(0, x)$*, and is a popular activation function.*

On top of ReLU Activation, there exists its close relatives, like:

- **Leaky ReLU**. Instead of defining the ReLU as 0 for all $x < 0$, Leaky ReLU defines it as a small linear component of x.
- **ELU**. Instead of defining the ReLU as 0 for all $x < 0$, ELU defines it as $\alpha(e^x - 1)$ for some α

Problem: (Review) Forward and Backward Pass for ReLU

Compute the output of forward pass of a ReLU layer with input x as given below:

$$y = \text{ReLU}(x)$$

$$x = \begin{bmatrix} 1.5 & 2.2 & 1.3 & 6.7 \\ 4.3 & -0.3 & -0.2 & 4.9 \\ -4.5 & 1.4 & 5.5 & 1.8 \\ 0.1 & -0.5 & -0.1 & 2.2 \end{bmatrix}$$

With the gradients with respect to the outputs $\frac{dL}{dy}$ given below, compute the gradient of the loss with respect to the input x using the backward pass for a ReLU layer:

$$\frac{dL}{dy} = \begin{bmatrix} 4.5 & 1.2 & 2.3 & 1.3 \\ -1.3 & -6.3 & 4.1 & -2.9 \\ -0.5 & 1.2 & 3.5 & 1.2 \\ -6.1 & 0.5 & -4.1 & -3.2 \end{bmatrix}$$

Problem: ReLU Potpourri

1. What advantages does using ReLU activations have over sigmoid activations?
2. ReLU layers have non-negative outputs. What is a negative consequence of this problem? What layer types were developed to address this issue?

7 Summary

- Recall the main ConvNet architectures (LeNet, AlexNet, GoogLeNet, VGGNet, ResNet). In particular, recall why bottleneck layers is ResNet are important.
- Batch Normalization proceeds by first computing empirical mean and variance, then rescaling each activation and squashing through γ and β
- Ensembles group several models into a single model. To make this quicker, we can either only make classification layers ensembles or use snapshot ensemble.
- Dropouts are methods for randomly removing nodes, and intuitively represent an ensemble of networks, since each forward pass is effectively a different network