Training and Testing

Training

Data Prepoccessing

1. Subtract per-channel mean
    i. reason:
        - For gradients: gradients will not always be positive or negative.
        - For activation: ReLu will have half the neurons activated.
        - for optimization: less sensitive to small changes in weights, easier to optimize.
    ii. how should we subtract the mean?
        - unique mean for every image: not appropriate, consider pure red and pure green images, they will be exactly the same after such initialization
        - unique mean for all pixel: not appropriate, channels are handled seperately, some channels will probably be all positive or all negative, resulting in bad gradient.
2. Divide by per-channel standard deviation
    - be cautious when doing this
    - may scale the small deviation in some channel

Weight Initialization

1. target: all the activations has almost same variance
    - how will the gradients behave? Actually, one purpose of weight initialization is to make gradients has almost same magnitude, so is will be sensible to have one unique learning rate for all the weights. However, this is not precisely guaranteed. As we know, $Var(X_1X_2) = Var(X1)Var(X_2) + Var(X1) * E(X_2^2) + Var(X_2) * E(X_1^2)$, $Var(X_1 + X_2) = Var(X_1) + Var(X_2)$. The gradient is not only dependent on the magnitude of the activations, which is affected by the initialization, but also dependent on the size of the feature.
2. Xavier initialization
    - $W\ N(0, 1/Din)$, Din is the input dimension
    - for conv layers, $Din = filter\_size^2 * in\_channels$
3. He initialization
    - $W\ N(0, 2/Din)$, Din is the input dimension

Optimization

1. problem with SGD:
    i. If loss changes quickly in one direction, and slowly in another. SGD will take very slow progress along shallow dimension, while jittering along the deep dimension.
        - This means loss function has high condition number: ratio of largest to smallest singular vaue of the Hessian matrix is large.
    ii. At local minima or saddle points, SGD can get stuck.
    iii. Minibatch gradient can be noisy
2. SGD + Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

    - ρ gives 'friction', typically 0.9 or 0.99
    - α is the learning rate
3. Adam

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla f(x_t) \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2)\nabla f(x_t)^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
x_{t+1} &= x_t - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)
\end{aligned}
$$

- β_1 and β_2 are hyperparameters, typically 0.9 and 0.999
- ϵ is a small constant to avoid division by zero
- α is the learning rate, 1e-3 or 5e-4 is a good starting point.

Learning Rate Scheduling

Assume α_0 is the initial learning rate, α_t is the learning rate at step t, T is the total number of steps, and t is the current step.

1. Step decay: reduce rate at a few fixed points. E.g. multiply by 0.1 at 100000 steps, 0.01 at 200000 steps.
2. Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + cos(t\pi/T))$
3. Linear: $\alpha_t = \alpha_0(1 - t/T)$
4. Inverse sqrt: $\alpha_t = \alpha_0/sqrt(t)$
5. Liear Warmup: hign initial learning rate can make loss explode; linearli increase learning rate from 0 over the first ~5000 iterations to prevent this.

Adam is a good default choice, SGD + Momentum can outperform Adam but it needs more tuning. Cosine scheduling request very little parameters, so it's good to try out.

Batch Size and Learning Rate

1. Empirical rule: if you increase the batch size by N, you can also scale the initial learning rate by N.

Underfitting

Both training and validation loss are high, usually caused by:

1. limited model capacity
2. unsatisfactory optimization

Solution1: Batch normalization

1. architecture: add a batch normalization layer after each fully connected or convolutional layer, before non-linearity.
2. model:
   - input: $x \in R^{N \times D}$
   - parameters: $\gamma, \beta \in R^D$
   - train time:

$$
\begin{aligned}
\mu_j &= \frac{1}{N}\sum_{i=1}^{N} x_{i,j} \\
\sigma_j^2 &= \frac{1}{N}\sum_{i=1}^{N} (x_{i,j} - \mu_j)^2 \\
\hat{x}_{i,j} &= \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \\
y_{i,j} &= \gamma_j \hat{x}_{i,j} + \beta_j \\
\mu_{rms} &= \rho \mu_{rms} + (1-\rho)\mu \\
\sigma_{rms}^2 &= \rho \sigma_{rms}^2 + (1-\rho)\sigma^2
\end{aligned}
$$

   - test time:

$$
\begin{aligned}
\hat{x}_{i,j} &= \frac{x_{i,j} - \mu_{rms,j}}{\sqrt{\sigma_{rms,j}^2 + \epsilon}} \\
y_{i,j} &= \gamma_j \hat{x}_{i,j} + \beta_j
\end{aligned}
$$

3. benefits:

- makes the network much easier to train
- improve gradient flow
- allows higher learning rates, faster convergence
- network becomes more robust to initialization
- act as regularization during training

4. why does it work?
   - mitigate the "internal covariate shift"
   - smoothen the loss landscape
5. problems with BN:
   - Batch normalization will cause the network to behave differently during training and testing
   - if batch size in the training time is too small, then μ, σ in a training batch can be very random
   - may lead to huge performance drop at test time even for training data.
6. Different Normalization Techniques
   - Layer Normalization: normalize each sample independently, i.e. for every sample, find the mean and variance across all the channels.
   - Instance Normalization: for each channel in a sample, calculate the mean and variance independently.
   - Group Normalization: divide the channels(of every single sample) into groups, calculate the mean and variance for each group, and normalize the channels within the group.

   NOTE: GroupNorm outperforms BatchNorm, when
   - Batch size is small
   - Instance in a batch are highly correlated
7. Why does BN differs in train and eval mode?
   - During training, BN computes the mean and variance of the current batch, and uses these values to normalize the current batch.
   - During evaluation, BN uses the running mean and variance, which are computed from the training data.
   - If we also use the mean and variance computed from the validation data, then the validation output will depend on the validation batch. Different batch sequences (1,2,3 is different from 1,4,5) will result in different outputs.
8. what is internal covariance shift?
   internal covariance shift means the activations of a network changes in distribution after weight update. This cause the following layers need to adapt to different distribution continuously, make training hard.

Solution2: ResNet

1. intuition: deeper network have more representative power, but harder to optimize. Actually, it is found that deep CNN can not learn identity function, i.e. f(x) = x.
2. benefits:
   - learn residual mapping instead of directly learn the underlying mapping
   - provides bypaths for gradients to backpropagate
   - smoothes the loss landscape

Overfitting

training loss is low, validation loss is high, usually caused by:

1. imbalance between data and model

Solution1: Data Augmentation

1. Position augmentation:
   - Scaling
   - Cropping
   - Flipping
   - Padding
   - Rotation
   - Translation
   - Affine transformation
2. Color augmentation:
   - Brightness
   - Contrast
   - Saturation
   - Hue

Solution2: Regularization

Solution3: Dropout

In each forward pass, randomly set some neurons to zero. Probability of dropping is set by a hyperparameter, 0.5 is common.