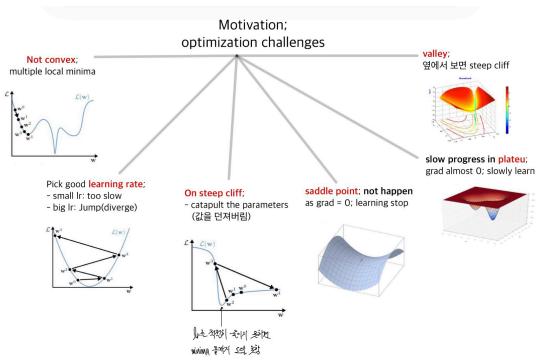


Lec5. optimization

Debugging Tips (Review when you make model.)



Motivation

Emergency First Response

- Start with a simple model that is known to work for this type of data (e.g. linear regression, use a standard loss if possible)
- Turn off all bells and whistles, e.g. regularizations, data augmentation
- If fine-tuning a model, double-check the preprocessing: for it to be the same as the one the model's training.
- Verify your input data.
- Start with a really small dataset (10 samples). Overfit on it and gradually add more data.
- Start gradually adding back all the pieces that were omitted: augmentation/regulation, custom loss functions, try more complex models.

Visualizing training process

- Visualize the training progress and evolution of your layers. Help to detect regularization issues. For example, if the magnitude of the weights is decreasing, it is a regularization theory. If the model is getting worse, it is probably due to overfitting.
- Check for NaN values. If you see NaN values in your gradients or weights, it means that the gradients are not appropriate. Gradient descent sometimes overflows or underflows. In these cases, the gradients will generally start at zero and increase rapidly. This can lead to NaN values.
- Monitoring loss
- Monitoring metrics
- Monitoring gradients

Dataset Issues

- Check your input data
- Try random input
- Check the data loader
- Make sure input is connected to output
- Verify size of the dataset
- Shuffle the data
- Reduce class imbalance
- Verify number of training examples
- Make sure your batches don't contain a single label
- Use a standard dataset

Data Normalization/Augmentation Issues

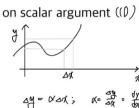
- standardize the features
- Check for too much data augmentation
- Check the preprocessing of your pretrained model
- Check the preprocessing for train/validation/test set

Implementation Issues

- Try solving a simpler version of the problem
- Make sure your training procedure is correct
- Check your loss function
- Verify loss input
- Adjust loss weights
- Monitor other metrics
- Test any custom layers
- Check for "frozen" layers or variables
- Increase network size
- Check for hidden dimension errors
- Exploding/Vanishing gradients
- Increase/Decrease learning rate
- Try solving a simpler version of the problem
- Make sure your training procedure is correct
- Check your loss function

Calculating Gradients

Derivatives



Matrix

Hessian

$$\nabla^2 f(\mathbf{x}) = \nabla(\nabla f(\mathbf{x})) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Minimization

Algorithm

```

Initialization:
  -  $\mathbf{x}^* = \mathbf{x}_0$ 
  -  $\mathbf{g}^* = \mathbf{0}$ 
  -  $\mathbf{f}^* = f(\mathbf{x}_0)$ 
  -  $\epsilon = 10^{-6}$ 
  -  $\text{max\_iter} = 1000$ 
  -  $\text{grad\_norm} = \|\mathbf{g}^*\|$ 
  -  $\text{loss\_norm} = |f(\mathbf{x}^*) - \mathbf{f}^*|$ 
  -  $\text{grad\_tol} = \epsilon$ 
  -  $\text{loss\_tol} = \epsilon$ 

for i = 1 to max_iter do
  if grad_norm <= grad_tol then
    break
  else
     $\mathbf{x}^* = \mathbf{x}_i$ 
     $\mathbf{g}^* = \nabla f(\mathbf{x}^*)$ 
     $\mathbf{f}^* = f(\mathbf{x}^*)$ 
     $\text{grad\_norm} = \|\mathbf{g}^*\|$ 
     $\text{loss\_norm} = |f(\mathbf{x}^*) - \mathbf{f}^*|$ 
    if loss_norm <= loss_tol then
      break
    end if
  end if
end for

```

Example

$$f(x_1, x_2, x_3) = x_1^2 + x_1(1-x_2) + (x_2-1)^2 - x_2x_3 + x_3^2 + x_1$$

$$\text{gradient: } \nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{pmatrix} = \begin{pmatrix} 2x_1 + 1 - x_2 \\ x_1 - 2x_2 + 1 \\ -x_2 + 2x_3 \end{pmatrix} = \begin{pmatrix} 2x_1 + 1 - x_2 \\ x_1 - 2x_2 + 1 \\ -x_2 + 2x_3 \end{pmatrix}$$

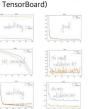
$$\text{Hessian matrix: } \nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix} = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Optimization algorithms

Optimization strategies

Optimization strategies

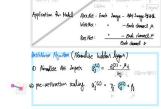
(1) Monitoring the training process (on TensorBoard)



(2) Data Preprocessing

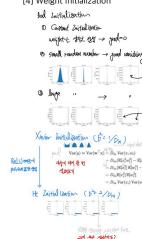


(3) Data Augmentation

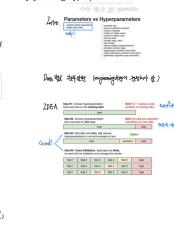


Optimization strategies

(4) Weight initialization



(5) Learning rate scheduling

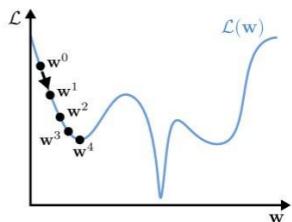


(6) Setting hyperparameters

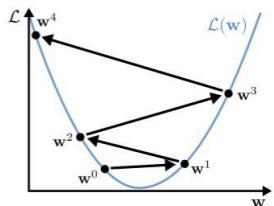


Motivation; optimization challenges

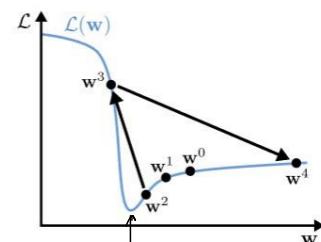
Not convex;
multiple local minima



Pick good **learning rate**;
 - small lr: too slow
 - big lr: Jump(diverge)

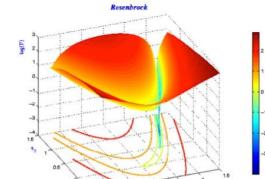


On steep cliff;
 - catapult the parameters
(값을 던져버림)

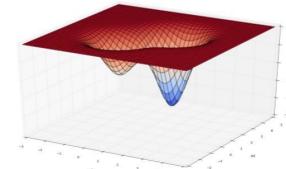


값을
던져버리자
Local
minima
를
넘어가자

valley;
옆에서 보면 steep cliff



slow progress in plateau;
grad almost 0; slowly learn



Optimization algorithms

Gradient Descent

↓ pick mini-batch ($B = 2^n$ data) 매번 다 계산할까?

SGD

Algorithm

- ① initialize w^0, η, B
- ② Draw random mini batch $\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_B, \mathbf{y}_B)\}^B$
- ③ for i^{th} element $(\mathbf{x}_i, \mathbf{y}_i)$
 - forward: $\mathbf{w}_i \rightarrow \hat{\mathbf{y}}_i$
 - backward: $\nabla_{\mathbf{w}} L_b(\mathbf{w}^t) = \nabla_{\mathbf{w}} L_b(\hat{\mathbf{y}}_i, \mathbf{y}_i, \mathbf{w})$
- ④ Update gradient $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{1}{B} \sum \nabla_{\mathbf{w}} L_b(\mathbf{w}^t)$

perf: SGD works



problem of SGD : how to find good η ?

- ① Jittering in shallow dimension
 - ② Stuck in local minima & saddle point
 - ③ noisy gradient
- :: gradient이 끊임없이 바뀌면, minibatch의 gradient

TMI:

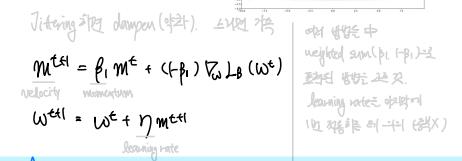
대각선 흥미로
Momentum의 문제점
(唯獨!)

$$\begin{aligned} \mathbf{v}_{\text{new}} &= \rho \mathbf{v}_{\text{old}} - \alpha \nabla f(\mathbf{w}_{\text{old}}) & \mathbf{v}_{\text{new}} &= \rho \mathbf{v}_{\text{old}} + \alpha \nabla f(\mathbf{w}_{\text{old}}) \\ \mathbf{w}_{\text{new}} &= \mathbf{w}_{\text{old}} + \mathbf{v}_{\text{new}} & \mathbf{w}_{\text{new}} &= \mathbf{w}_{\text{old}} - \alpha \mathbf{v}_{\text{old}} \\ \mathbf{v}_{\text{old}} &= \mathbf{v}_{\text{old}} + \alpha \nabla f(\mathbf{w}_{\text{old}}) & \mathbf{v}_{\text{old}} &= \mathbf{v}_{\text{old}} - \alpha \nabla f(\mathbf{w}_{\text{old}}) \end{aligned}$$

Solution of SGD

Momentum

SGD + Momentum



Scaling

AdaGrad

Adaptive Gradient Estimation

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

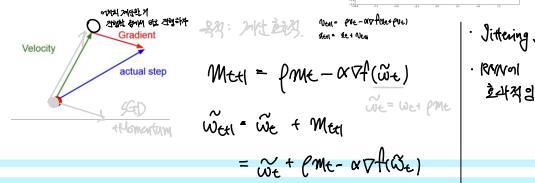
scaling: dimension \mathbb{R}^d 의 각 gradient \mathbb{R}^d 에 대한 정규화 (parameter)
 \mathbb{R}^d : adaptive lr는 정규화

scaling: dimension \mathbb{R}^d 의 각 gradient \mathbb{R}^d 에 대한 정규화 (parameter)
 \mathbb{R}^d : adaptive lr는 정규화

Adam(good) Momentum + RMSprop

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
    dx = compute_gradients(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

SGD + Nesterov Momentum



RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradients(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

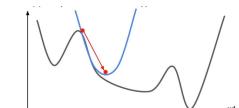
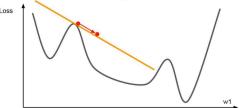
decay-rate $\neq 1$, weighted sum 사용
 \mathbb{R}^d 초기 설정으로, learning OK (잘 adaptive 됨)

Understanding with Math

Momentum

Bias correction

AdaGrad / RMSProp



TMI: Second-order optimization

RNN은 \mathbb{R}^d X, \mathbb{R}^d Y에 맞는다.

성능이 좋지 않아, 초기값에 따라 초기화가 됨.

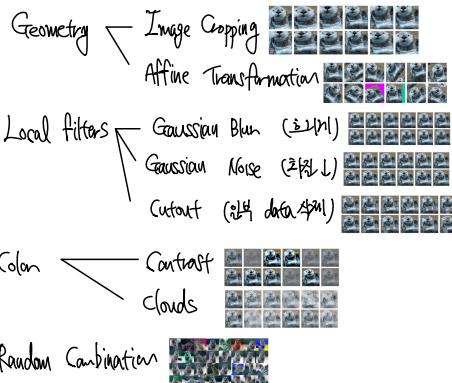
Optimization strategies

[1] Monitoring the training process (on TensorBoard)



[2] Data Augmentation

By data augmentation → data augmentation



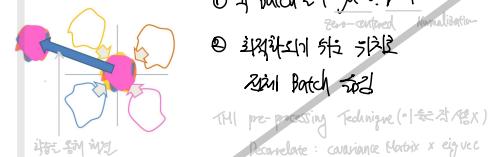
[3] Data Preprocessing

Motivation : positive grads (cf. sigmoid) \rightarrow 편향된 최적화
data distribution of each layer is different
cf. SGD (Red dot)

Batch Normalization

SGD 편향된 batch \rightarrow , but Batch의 distribution \neq
Normalize 1st layer ; gradient \rightarrow $\frac{\partial L}{\partial \mu} = \frac{1}{N} \sum_{i=1}^N \frac{\partial L}{\partial \mu_i}$ \rightarrow $\frac{\partial L}{\partial \mu} = \frac{1}{N} \sum_{i=1}^N \frac{\partial L}{\partial \mu_i} = \frac{1}{N} \cdot N = 1$

Solution (Not finish!)



THI pre-processing Technique (+ 편향 방지)

Decorrelate : covariance matrix \times eigenvector

Written : " \rightarrow $\tilde{x}_j = \frac{1}{\sqrt{N}} \sum_{i=1}^N x_i v_i$ "

Application for Model

AlexNet : Each 2image -> 2image

VGGNet : " -> Each channel μ

ResNet : " -> Each channel μ
Each channel σ

BatchNorm Algorithm (Normalize hidden layers)

$$\text{① Normalize Net Inputs } \tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$\text{② pre-activation scaling } a_j^{(i)} = \gamma_j \cdot \tilde{x}_j^{(i)} + \beta_j$$

Note

① Batch Norm은 training algorithm \neq test "

1. 훈련할 때 사용

? (2종)

(= inference)

sample \hat{y}_t on μ_t to σ_t

Batch data = (μ_t , σ_t)

where $t \in \{1, 2, \dots, T\}$

② Batch Norm은 covariance shift를 해결하는 방법

작간 편향을 줄이기 ; 높은 쪽에 선정하고 낮은 쪽에 선정

③ Batch Norm 적용 방식

Pre-Activation : $\text{layer} \rightarrow \text{BatchNorm} \rightarrow \text{activation f} \rightarrow \text{2nd layer}$

Post-Activation : " \rightarrow activation f \rightarrow BatchNorm \rightarrow "

④ Batch Size 개선으로 성능 안정화.

⑤ Batch Size N \rightarrow 증가. 성능의 x 연관성이 높아짐

⑥ BatchNorm은 layer를 처리할 때

각각 다른 backprop 구조 (computation graph 각각 다른 구조)

⑦ Batch Normalization for ConvNet



⑧ 각각 Normalization 구조

Layer Normalization

Instance "

Group "



Optimization strategies

[4] Weight Initialization

Bad initialization

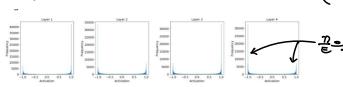
① Constant Initialization

weight = 상수 설정 $\rightarrow \text{grad} = 0$

② Small random number $\rightarrow \text{grad vanishing}$ ($b=0.01$)



③ Large " " \rightarrow ($b=0.2$)



Xavier Initialization ($b^2 = 1/Din$)

$$\text{Var}(y) \approx \text{Var}(w^T x) = D_{in} \text{Var}(x_i w_i)$$

수행에 대해 한 번
설명하기

$$= D_{in} (\mathbb{E}[x_i^2 w_i^2] - \mathbb{E}[x_i w_i]^2)$$

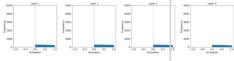
$$= D_{in} (\mathbb{E}[x_i^2] \mathbb{E}[w_i^2] - \mathbb{E}[x_i]^2 \mathbb{E}[w_i]^2)$$

$$= D_{in} \mathbb{E}[x_i^2] \mathbb{E}[w_i^2]$$

$$= D_{in} \text{Var}(x_i) \text{Var}(w_i)$$

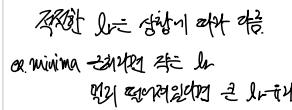
ReLU에서
positive gradient

He Initialization ($b^2 = 2/Din$)

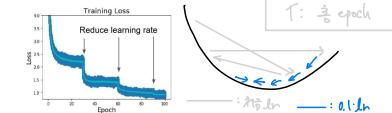


그에 따른 성과가?

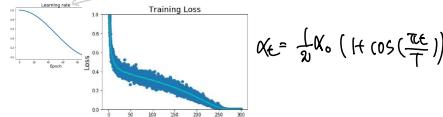
[5] Learning rate scheduling



① 90 epochs of lr = 0.1 막상 (cf. ResNet)



② Cosine- $\frac{\pi}{T}$ lr-조정법 사용



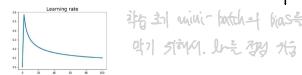
③ Linear-sim lr-조정법 사용 (cf. Beat)

$$\alpha_t = \alpha_0 (1 - \frac{t}{T})$$

④ Inverse sqrt lr (cf. transformer; attention is all you need)

$$\alpha_t = \frac{\alpha_0}{\sqrt{t}}$$

⑤ lr 증가하기 (a.k.a. linear warm up)



[6] Setting hyperparameters

하나의 학습률은 parameter

Intro.

Parameters vs Hyperparameters

- minibatch size
- data normalization schemes
- number of epochs
- number of hidden layers
- number of hidden units
- learning rates
- random seed, why?
- loss function
- various weights (embedding dims)
- various model types
- regularization schemes (more later)
- weight initialization schemes (more later)
- optimization algorithm (one more later)
- ...

Data 품질 조건(학습률 설정하기)

ZDEA

Idea #1: Choose hyperparameters that work best on the training data

train

BAD: K = 1 always works perfectly on training data

Idea #2: choose hyperparameters that work best on test data

train test

BAD: No idea how algorithm will perform on new data

averaging

테스트 사용X

Idea #3: Split data into train, val, choose hyperparameters on val and evaluate on test

train validation test

Better!

Good!

Idea #4: Cross-Validation: Split data into folds, try each fold as validation and average the results

fold 1 fold 2 fold 3 fold 4 fold 5 test

fold 1 fold 2 fold 3 fold 4 fold 5 test

fold 1 fold 2 fold 3 fold 4 fold 5 test

Debugging Tips *(Review when you make model)*

Emergency First Response

- Start with a simple model that is known to work for this type of data cf. VGG for images, use a standard loss if possible
- Turn off all bells and whistles, e.g. regularization and data augmentation
- If finetuning a model, double check the preprocessing, for it should be the same as the original model's training
- Verify that the input data is correct
- Start with a really small dataset(1-10 samples). Overfit on it and gradually add more data
- Start gradually adding back all the pieces that were omitted: augmentation/regularization, custom loss functions, try more complex models.

Visualize the training process

- ▶ Monitor the activations, weights, and updates of each layer. Make sure their magnitudes match. For example, the magnitude of the updates to the parameters (weights and biases) should be 1×10^{-3} .
- ▶ Consider a visualization library like Tensorboard and Crayon. In a pinch, you can also print weights/biases/activations.
- ▶ Be on the lookout for layer activations with a mean much larger than 0. Try Batch Norm or ELUs.
- ▶ Weight histograms should have an approximately Gaussian (normal) distribution, after some time. For biases, these histograms will generally start at 0, and will usually end up being approximately Gaussian. Keep an eye out for parameters that are diverging or biases that become very large.

Overcoming NaNs

- ▶ Decrease the learning rate, especially if you are getting NaNs in the first 100 iterations.
- ▶ NaNs can arise from division by zero or natural log of zero or negative number.
- ▶ Try evaluating your network layer by layer and see where the NaNs appear.

Dataset Issues

- Check your input data
- Try random input
- Check the data loader
- Make sure input is connected to output
- Verify noise in the dataset
- Shuffle the dataset
- Reduce class imbalance
- Verify number of training examples
- Make sure your batches don't contain a single label
- Use a standard dataset

Training Issues

- Solve for really small dataset
- Check weights initialization
- Change your hyperparameters
- Reduce regularization
- Give it time
- Switch from Train to Test mode
- **Visualize the training process**
- Try a different optimizer
- Exploding/Vanishing gradients
- Increase/Decrease learning rate
- **Overcoming NaNs**

Data Normalization/ Augmentation Issues

- standardize the features
- Check for too much data augmentation
- Check the preprocessing of your pretrained model
- Check the preprocessing for train/validation/test set

Implementation Issues

- Try solving a simpler version of the problem
- Make sure your training procedure is correct
- Check your loss function
- Verify loss input
- Adjust loss weights
- Monitor other metrics
- Test any custom layers
- Check for "frozen" layers or variables
- Increase network size
- Check for hidden dimension errors
- Explore gradient checking
- Try solving a simpler version of the problem
- Make sure your training procedure is correct
- Check your loss function

부록