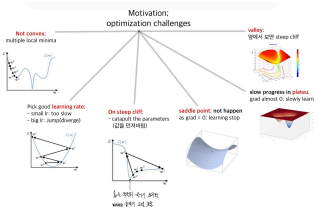
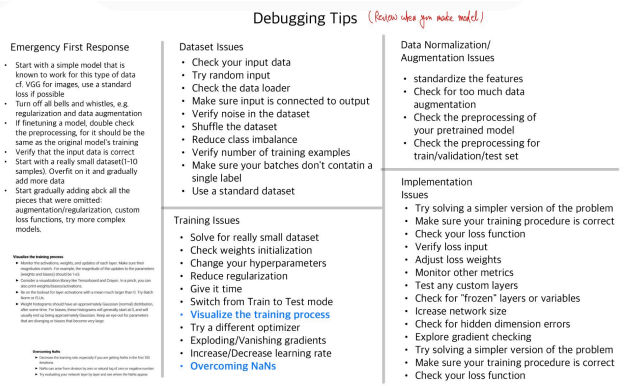
Lec6. CNNs

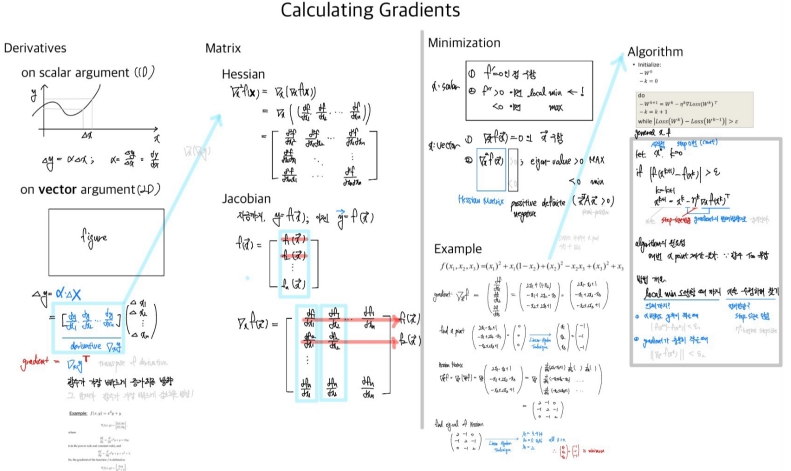
Introduction



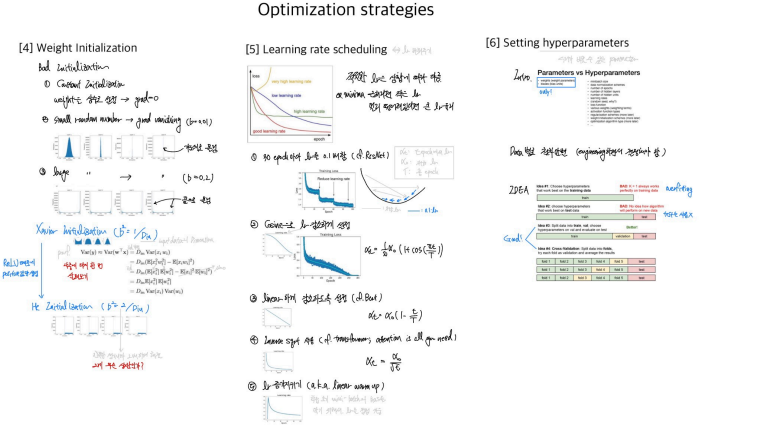
Transfer learning



Convolutional Neural Networks

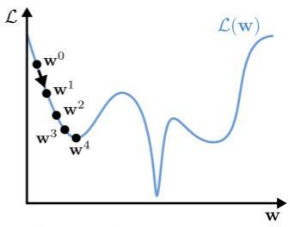


CNN architectures

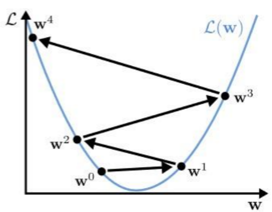


Motivation;
optimization challenges

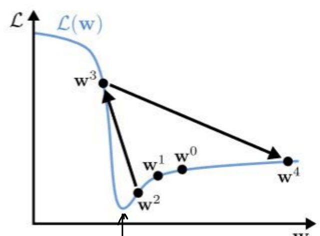
Not convex; multiple local minima



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

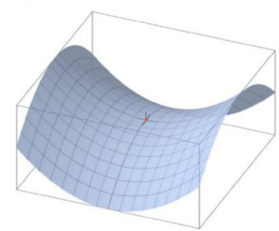


On steep cliff;
- catapult the parameters (값을 던져버림)

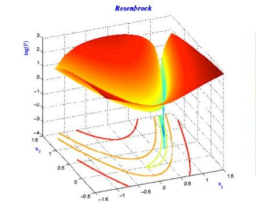


lr을 적절히 줄이지 못하면
minima 끝까지 도달 못함

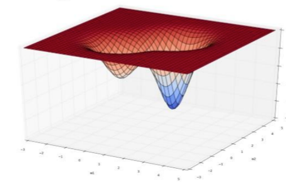
saddle point; **not happen**
as grad = 0; learning stop



valley;
옆에서 보면 steep cliff



slow progress in plateu;
grad almost 0; slowly learn



Optimization algorithms

TMI:

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

$x_{t+1} = x_t + v_{t+1}$, $x_{t+1} = x_t - \alpha v_{t+1}$

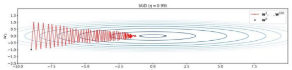
$= x_t + (\rho v_t - \alpha \nabla f(x_t))$, $= x_t - (\alpha \rho v_t + \alpha \nabla f(x_t))$

다양한 형태로 Momentum식 구현 가능 (다른 뜻!)

Gradient Descent

↓ pick mini-batch ($B = 2^n$ 개 data) 매번 다 계산 부담

SGD



Algorithm

① initialize w^0 η B

② Draw random mini batch $\{(x_1, y_1), \cdots (x_B, y_B)\}$

③ for 각 element (x_i, y_i)

forward: $x_b \longrightarrow \hat{y}_b$

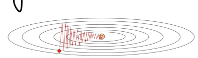
backward: $\nabla_w L_b(w^t) = \nabla_w L_b(\hat{y}_b, y_b, w)$

④ Update gradient $w^{t+1} = w^t - \eta \frac{1}{B} \sum \nabla_w L_b(w^t)$ Loss

proof: SGD works

problem of SGD : how to find good lr?

① Jittering in shallow dimension



② Stuck in local minima & saddle point

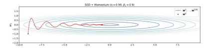
③ noisy gradient

∵ 전체가 아닌, minibatch의 gradient

Solution of SGD

Momentum

SGD + Momentum



Jittering하면 dampen(약화). 느리면 가속

$m^{t+1} = \beta_1 m^t + (1-\beta_1) \nabla_w L_B(w^t)$

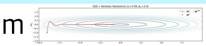
velocity momentum

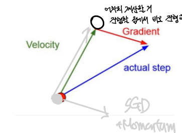
$w^{t+1} = w^t + \eta m^{t+1}$

learning rate

여러 방법론 中 weighted sum(β_1, $1-\beta_1$)으로 표현된 방법론 고른 것. learning rate는 여기에 1번 적용하는 게 나아 (중복X)

SGD + Nestrov Momentum





목적: 계산 효율성

$M_{t+1} = \rho M_t - \alpha \nabla f(\tilde{w}_t)$

$\tilde{w}_t = w_t + \rho M_t$

$\tilde{w}_{t+1} = \tilde{w}_t + M_{t+1}$

$= \tilde{w}_t + \rho M_t - \alpha \nabla f(\tilde{w}_t)$

· Jittering ↓

· RNN에 효과적임

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)
```

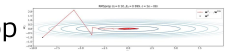
계속 누적 → ∞

scaling: dimension 별로 다른 gradient 적용되도록 (parameter)

lr: adaptive lr로 표현

⟶ step-size → 0 방지하기 위해

RMSProp



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

decay-rate 추가, weighted sum 형태

lr을 크게 설정해도, learning OK (잘 adaptive하게 조정됨)

Adam(good)

Momentum + RMSprop

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(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))
```

Momentum

Bias correction

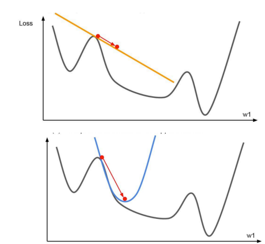
AdaGrad / RMSProp

Understanding with Math

TMI: Second-order optimization

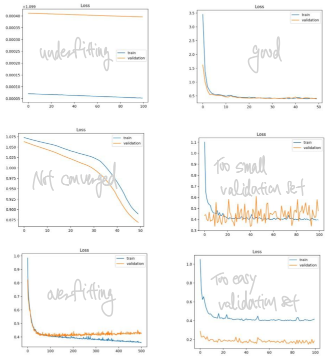
RNN은 사용 X. 계산량 너무 많음.

사용한다면, 훨씬 빠르게 최적화 가능.



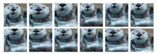
Optimization strategies

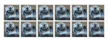
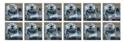
[1] Monitoring the training process (on TensorBoard)

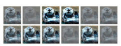
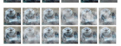


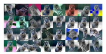
[2] Data Augmentation

1개 data를 조작해서 다양한 data 생성

Geometry
- Image Cropping 
- Affine Transformation 

Local filters
- Gaussian Blur (흐리게) 
- Gaussian Noise (화질↓) 
- Cutout (일부 data 삭제) 

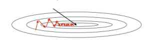
Color
- Contrast 
- Clouds 

Random Combination 

[3] Data Preprocessing

Motivation : positive grads (cf. sigmoid) → 편향된 최적화

data distribution에 따라 학습속도 다름

cf. SGD 

Batch Normalization

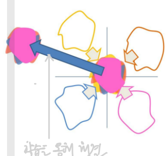
SGD 진행할 때 Batch 뽑음, but Batch별 distribution ≠

a.k.a. covariate shift

Normalize 1st layer ; gradient → 0, → ∞

$\frac{dx}{dx_i} < 0$, > 0 $\frac{dL}{dx_i} = \frac{dL}{dx_N} \cdot \frac{dx_N}{dx_{N-1}} \cdots \frac{dx_{i+1}}{dx_i}$

Solution (Not finish!)



① 각 Batch들의 $\mu = 0$, $\sigma = 1$ (zero-centered, Normalization)

② 최적화되기 쉬운 위치로 전체 Batch 옮김

THI pre-processing Technique (이들은 잘 사용X)

Decorrelate : covariance Matrix × eig.vec

Whiten : 〃 ÷ $\sqrt{(\text{eig.vec})^2}$

Application for Model

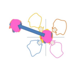
AlexNet : Each Image − 전체 Image 평균값

VGGNet : 〃 − Each channel μ

ResNet : $\frac{\text{〃} - \text{Each channel } \mu}{\text{Each channel } \sigma}$

BatchNorm Algorithm (Normalize hidden layers)

① Normalize Net Inputs $\hat{z}_j^{(i)} = \frac{z_j^{[i]} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$ (분모=0 방지)



② pre-activation scaling $a_j^{(i)} = \gamma_j \cdot \hat{z}_j^{(i)} + \beta_j$ ($\gamma_j \to \sigma$, $\beta_j \to \mu$)

Note

① BatchNorm은 train할 때 algorithm ≠ test (= Inference) 〃

μ, σ 구해서 사용 ? (2종)

- sample 입력받을 때 μ, σ 구하기
- 전체 data의 μ, σ 쓰기 (비슷할 거라고 가정)

② BatchNorm이 covariate shift를 없앤다는 증명X

약간 도움이 되는 수준 ; 높은 크기 선정해도 괜찮다 학습속도↑

③ BatchNorm 적용 방식

- Pre-Activation : 이번 layer → BatchNorm → activation f → 다음 layer (주로 사용 ; 순서 참고)
- Post-Activation : 〃 → activation f → BatchNorm → 〃

④ Batch Size 커질수록 성능 안정적.

⑤ Batch Size는 2^n으로 이루어짐. (컴퓨터 2^n 연산 처리 선호)

⑥ BatchNorm은 하나의 layer로 이해할 수 있다. 

미분 가능하므로 backprop 가능 (computation graph로 구할 수 있다)

⑦ Batch Normalization for ConvNet

⑧ 다른 Normalization 유형

Layer Normalization

Instance 〃

Group 〃

Optimization strategies

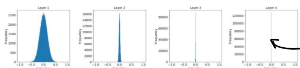
[4] Weight Initialization

Bad Initialization

① Constant Initialization

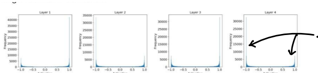
weight는 상수로 설정 → grad=0

② small random number → grad vanishing (b=0.01)



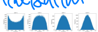
가운데로 몰림

③ large " → " (b=0.2)



끝으로 몰림

Xavier Initialization ($b^2 = 1/D_{in}$) input data의 Dimension



proof.

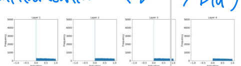
$$\mathrm{Var}(y) \approx \mathrm{Var}(\mathbf{w}^\top \mathbf{x}) = D_{in}\,\mathrm{Var}(x_i w_i)$$
$$= D_{in}(\mathbb{E}[x_i^2 w_i^2] - \mathbb{E}[x_i w_i]^2)$$
$$= D_{in}(\mathbb{E}[x_i^2]\,\mathbb{E}[w_i^2] - \mathbb{E}[x_i]^2\,\mathbb{E}[w_i]^2)$$
$$= D_{in}\mathbb{E}[x_i^2]\,\mathbb{E}[w_i^2]$$
$$= D_{in}\,\mathrm{Var}(x_i)\,\mathrm{Var}(w_i)$$

iid 가정, iid, μ=0

나중에 다시 한 번 살펴보기

ReLU 때문에 positive 값만 생성

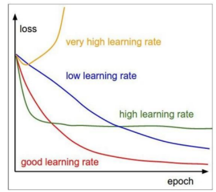
He Initialization ($b^2 = 2/D_{in}$)



한쪽만 생기니까 2배하여 해결

2배 무슨 상관인가?

[5] Learning rate scheduling ⇔ lr 바꿔주기



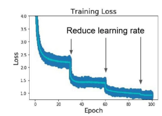
적절한 lr는 상황에 따라 다름.

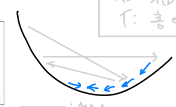
ex. minima 근처라면 작은 lr

멀리 떨어져있다면 큰 lr 유리

① 30 epoch 마다 lr을 0.1배함 (cf. ResNet)

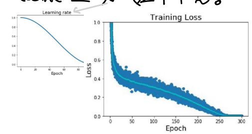
α_t: t epoch에서 lr
α_0: 처음 lr
T: 총 epoch





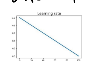
—— : 처음 lr —— : 0.1·lr

② Cosine으로 lr 감소하게 설정



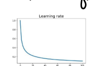
$$\alpha_t = \frac{1}{2}\alpha_0\left(1+\cos\left(\frac{\pi t}{T}\right)\right)$$

③ linear 하게 감소하도록 설정 (cf. Bert)



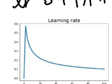
$$\alpha_t = \alpha_0\left(1 - \frac{t}{T}\right)$$

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



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

⑤ lr 증가시키기 (a.k.a. linear warm up)



학습 초기 mini-batch의 bias을 막기 위해서. lr은 점점 감소

[6] Setting hyperparameters

스스로 바꿀 수 없는 parameter

Intro.

Parameters vs Hyperparameters

- weights (weight parameters)
- biases (bias units)

only!

- 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 (weighting terms)
- activation function types
- regularization schemes (more later)
- weight initialization schemes (more later)
- optimization algorithm type (more later)
- ...

Data 별로 천차만별 (engineering하면서 결정해야 함.)

IDEA

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

BAD: K = 1 always works perfectly on training data — overfitting

train

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

BAD: No idea how algorithm will perform on new data — test는 사용X

train	test

Good!

Idea #3: Split data into **train**, **val**; choose hyperparameters on val and evaluate on test — **Better!**

train	validation	test

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 abck 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-e3.
- 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 contatin 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
- Icrease 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