# Neural Tips & Tricks

cs224d Deep NLP

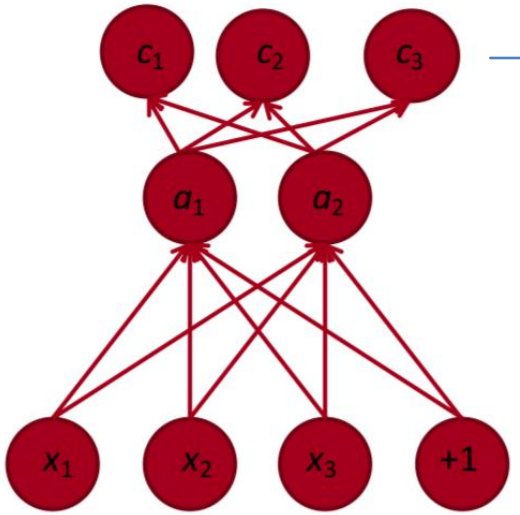
lec6-Neural Tips and Tricks & Recurrent Neural Network

장예훈

2017.11.9 Thu

# The model

- Base Model – Neural Network

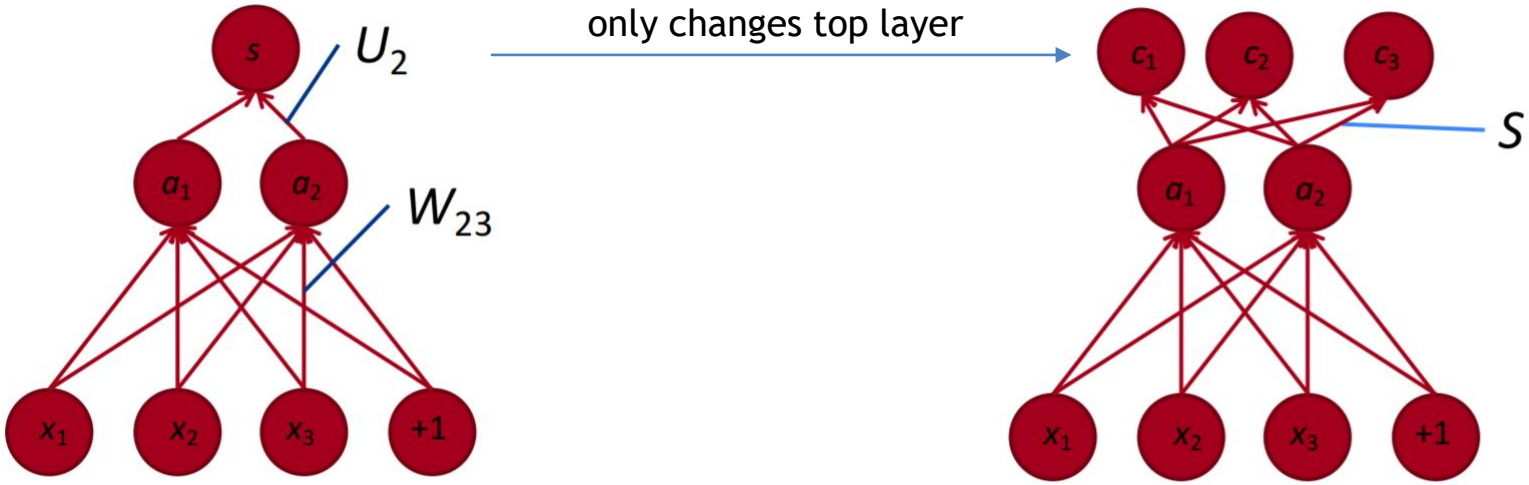



single scalar score with
a *Softmax* classifier

e.g., name entity tags, sentimental…
Any kind of label (specific word)

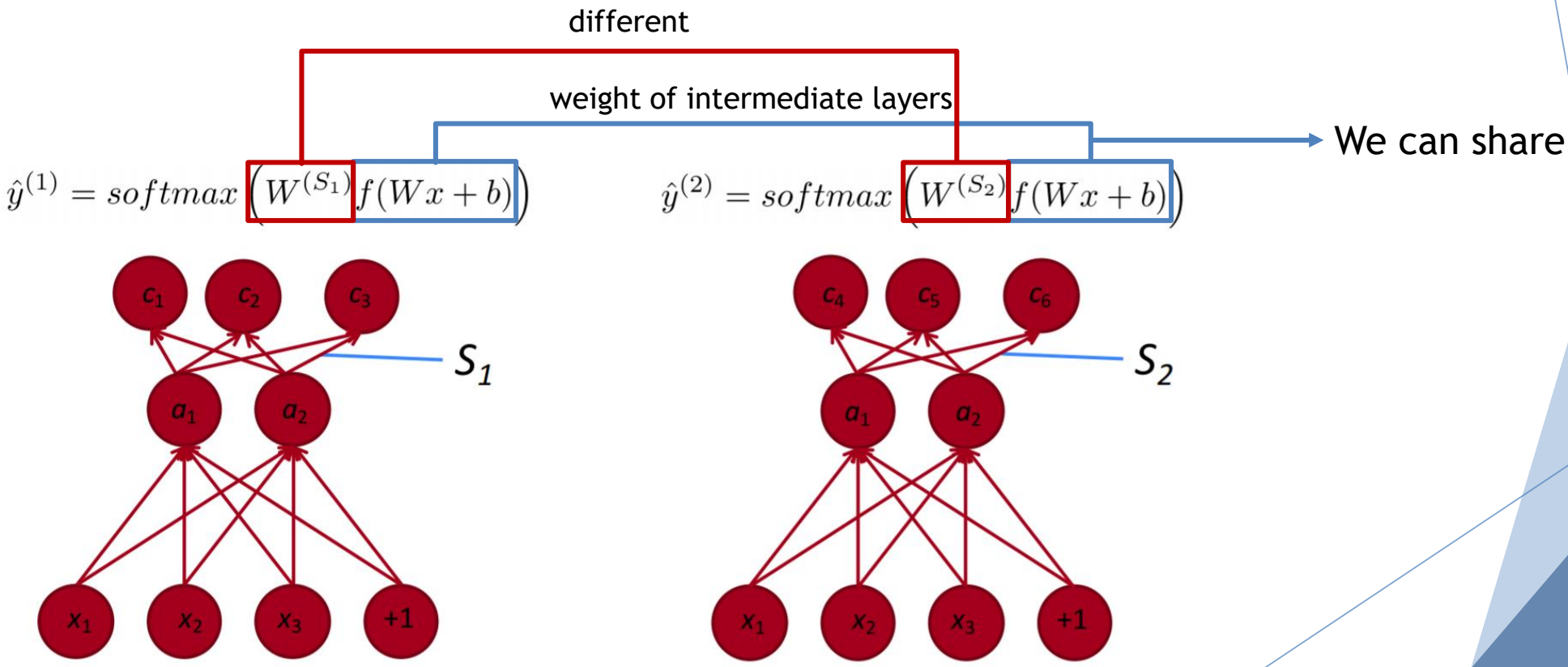
# The model

- Deep Learning 에서 흥미로운 점
  - input x와 hidden layer의 feature들도 학습이 되는 것
- 두 개의 final layer가 나올 수 있음



# Multi-task Learning / Weight Sharing

- Main Idea
  - word vector와 hidden layer weight는 공유 가능, softmax weight만 다름
- Cost Function은 두 개의 cross entropy error의 합



# The multitask model -Training

- window에서 center의 NER태그와 POS태그를 예측
- Efficient Implements
  - 같은 forward prop사용
  - Hidden vector의 error 계산:

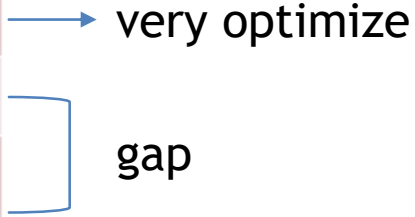
$$\delta^{total} = \delta^{NER} + \delta^{POS}$$

cf. backprop할 때는 $\delta^{total}$사용

# The secret sauce

- The unsupervised word vector pre-training on a large text collection
- 130,000 word embedding, 11 word window, 100 unit hidden layer

| | POS WSJ (acc.) | NER CoNLL (F1) |
|---|---|---|
| State-of-the-art* | 97.24 | 89.31 |
| Supervised NN | 96.37 | 81.47 |
| Word vector pre-training followed by supervised NN** | 97.20 | 88.87 |
| + hand-crafted features*** | 97.29 | 89.59 |



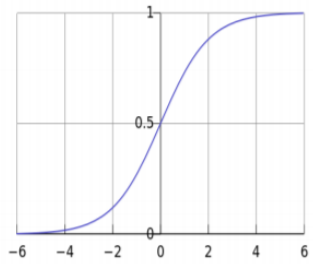
# General Strategy for Successful NNets

- 문제에 적합한 네트워크 구조 고르기
  - 구조: single word, fixed window, 문서 레벨; bag of word, RNN, CNN
  - 비선형
- Gradient check를 통함 debug
- 파라미터 초기화
- 최적화 tricks
- 모델 overfitting check
  - Overfitting 이라면 → 모델의 구조를 바꾸거나, 모델의 사이즈를 더 크게
  - Overfitting 이 아니라면 → Regularize

# Non-linearities: What's used

sigmoid

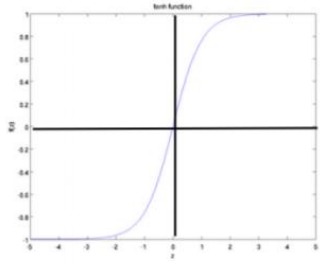
$$f(z) = \frac{1}{1 + \exp(-z)}.$$




$$f'(z) = f(z)(1 - f(z))$$

tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



$$f'(z) = 1 - f(z)^2$$

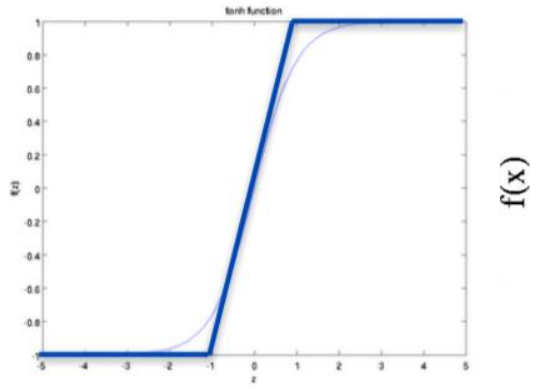
Tanh는 sigmoid를 그냥 shift 한 것!

- For many models, tanh is the best!
- Sigmoid와 비교해서
  - 초기화시 0 에 가까운 값
  - 더 빠른 convergence

# Non-linearities: There are various other choices

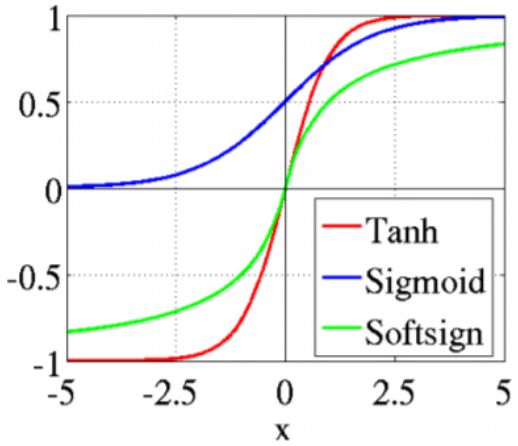
hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$$



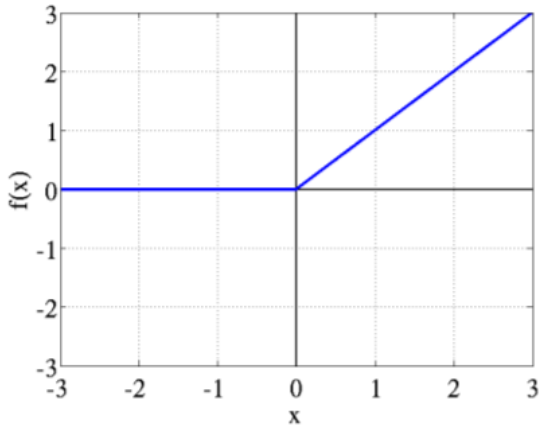

softsign

$$\text{softsign}(z) = \frac{a}{1+|a|}$$




ReLu

$$\text{rect}(z) = \max(z, 0)$$




- Backprop가 깊어질수록 error가 0에 가까워지고 lower layer에서는 error signal이 없어지는 현상 발생 → ReLu 사용하면 방지할 수 있음

# Gradient Checks are Awesome!

- 모델 구조에는 이상이 없다고 전제
- Step
  - Gradient 실행
  - 네트워크 파라미터를 반복하면서 미분을 추정하여 유한한 difference를 구현

$$f'(\theta) \approx \frac{J\left(\theta^{(i+)}\right) - J\left(\theta^{(i-)}\right)}{2\epsilon}$$

$$\theta^{(i+)} = \theta + \epsilon \times e_i$$

  - 위 둘을 비교하며 같다는 것을 증명

# Using gradient checks and model simplification

- 복잡한 NN model, visualization carefully , implement model

  & Gradient check → 실패, 이유는 알 수 없을 때

- **Create a very tiny synthetic model and dataset**
  - 아주 간단히  model debug 해 줌

- Choose what you want
  - 고정된 input에만 softmax사용
  - word vector를 backprop하고 softmax
  - single unit, single hidden layer 추가
  - multi unit, single layer 추가
  - bias 추가
  - Etc.

# General Strategy for Successful NNets

- ~~문제에 적합한 네트워크 구조 고르기~~
  - ~~구조: single word, fixed window, 문서 레벨; bag of word, RNN, CNN~~
  - ~~비선형~~
- ~~Gradient check를 통함 debug~~
- 파라미터 초기화
- 최적화 tricks
- 모델 overfitting check
  - Overfitting 이라면 → 모델의 구조를 바꾸거나, 모델의 사이즈를 더 크게
  - Overfitting 이 아니라면 → Regularize

# Parameter Initialization

- 만약 weight가 0 이면 hidden layer의 bias를 0으로 초기화하고,
  최적의 bias를 output으로 출력
- weight를  초기화
  - fan-in(이전 layer size)과 fan-out(다음 layer size)에 반비례하는 상수 (-r,r)

$$\sqrt{6/(\text{fan-in} + \text{fan-out})}$$

# Stochastic Gradient Descent (SGD)

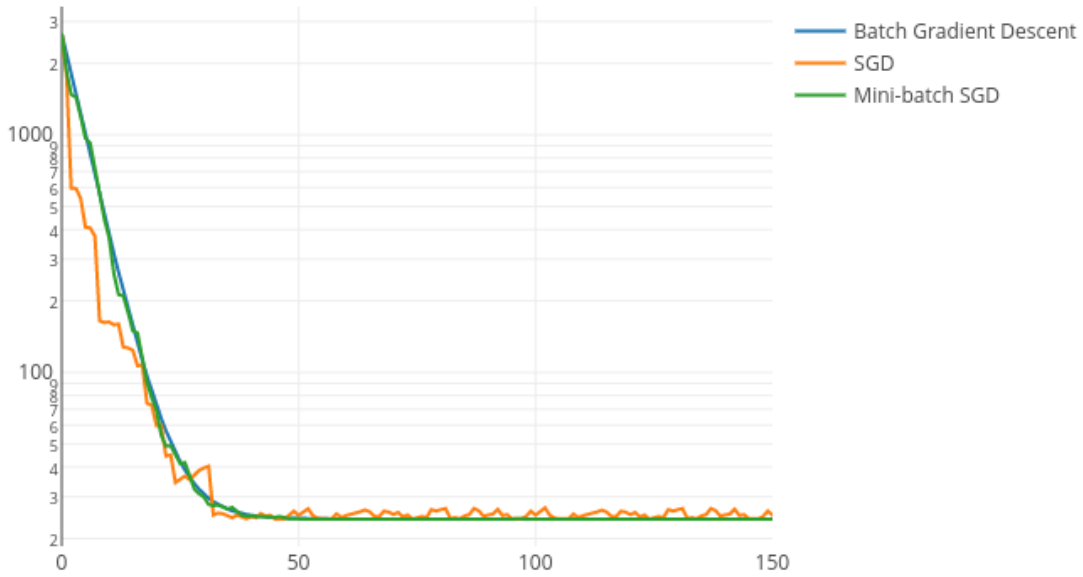
- Gradient descent는 example을 update할 때 마다 전반에 걸친 total gradient를 사용
- SGD는 1개 또는 몇 개의 example 후에 update함

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J_t(\theta)$$

- $J^t =$ 현재의 loss function, $\mu = parameter\ vector$
- **Batch method 로 평범한 gradient descent는 매우 느림 절대 쓰지 마!!**
- L-BFGS(Limited memory BFGS) 사용 권장
- Large dataset = SGD / small dataset = L-BFGS , Conjugate Gradient

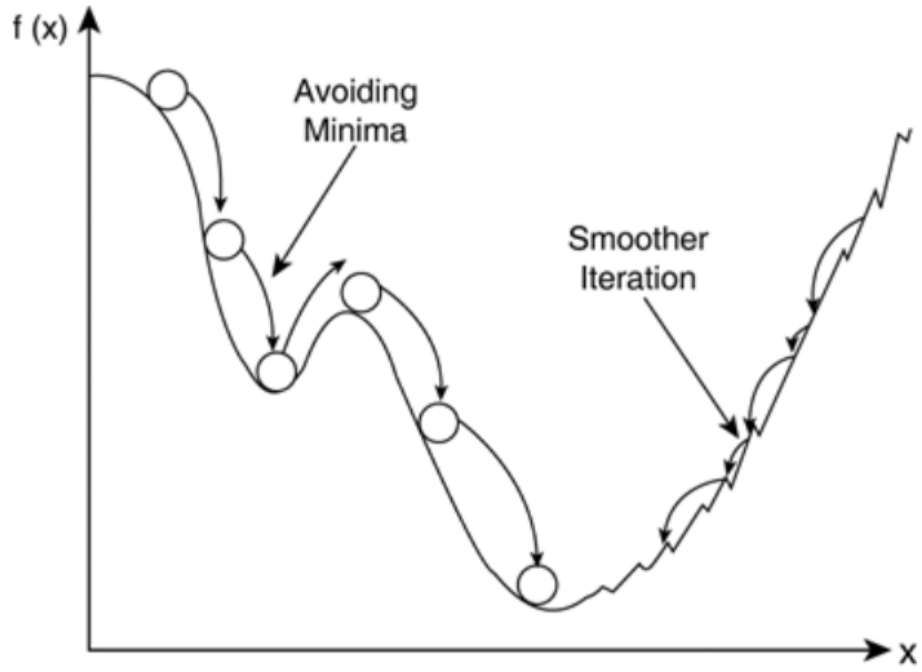
# Mini-batch Stochastic Gradient Descent

- Mini batch 많이 사용 (mini-batch size: 20~1000)
- 병렬 배치의 여러 elements의 gradient 를 계산하여 모든 모델을 병렬 처리 하는데 도움을 줌



# Improvement over SGD: Momentum

- Idea: 현재 v를 update 하기 위해 이전의 fraction v를 더함
- Gradient가 계속 같은 방향을 가리킬 때, 최소한의 변화를 줌
- Momentum을 많이 사용 할 때는 전역 learning rate를 줄여야 함
- Momentum 이 몇 epoch를 돌고 나면 증가함 0.5 → 0.99



# Intuition Momentum

- Friction을 더하는 것
- 파라미터들은 일정한 기울기 방향으로 속도를 형성함

- 간단한 convex 함수의 최적화 예시

without momentum

with momentum:

Average the direction of previous time steps

# Learning Rates

- 간단한 recipe: 그냥 $\alpha$ 고정하고, 파라미터 계속 같은 거 사용

- 좋은 결과를 얻기 위한 learning rate줄이기 options:
  - validation error 증가(?)가 멈추면 $\alpha$를 0.5배 줄임

# AdaGrad

- **Super Helpful!**
  - 다른 것들보다 훨씬 덜 자주 사용하는 확실한 파라미터 있을 때
- Learning rate를 모든 parameter마다 바꿔주는 것

- 예시
  - in large optimization problem
  - rare words 는 large update, common words는 small update

a, an, the,
he, she,
…

in every training samples

zebra, aardvark,
…

really rare words
that you see only once

# AdaGrad

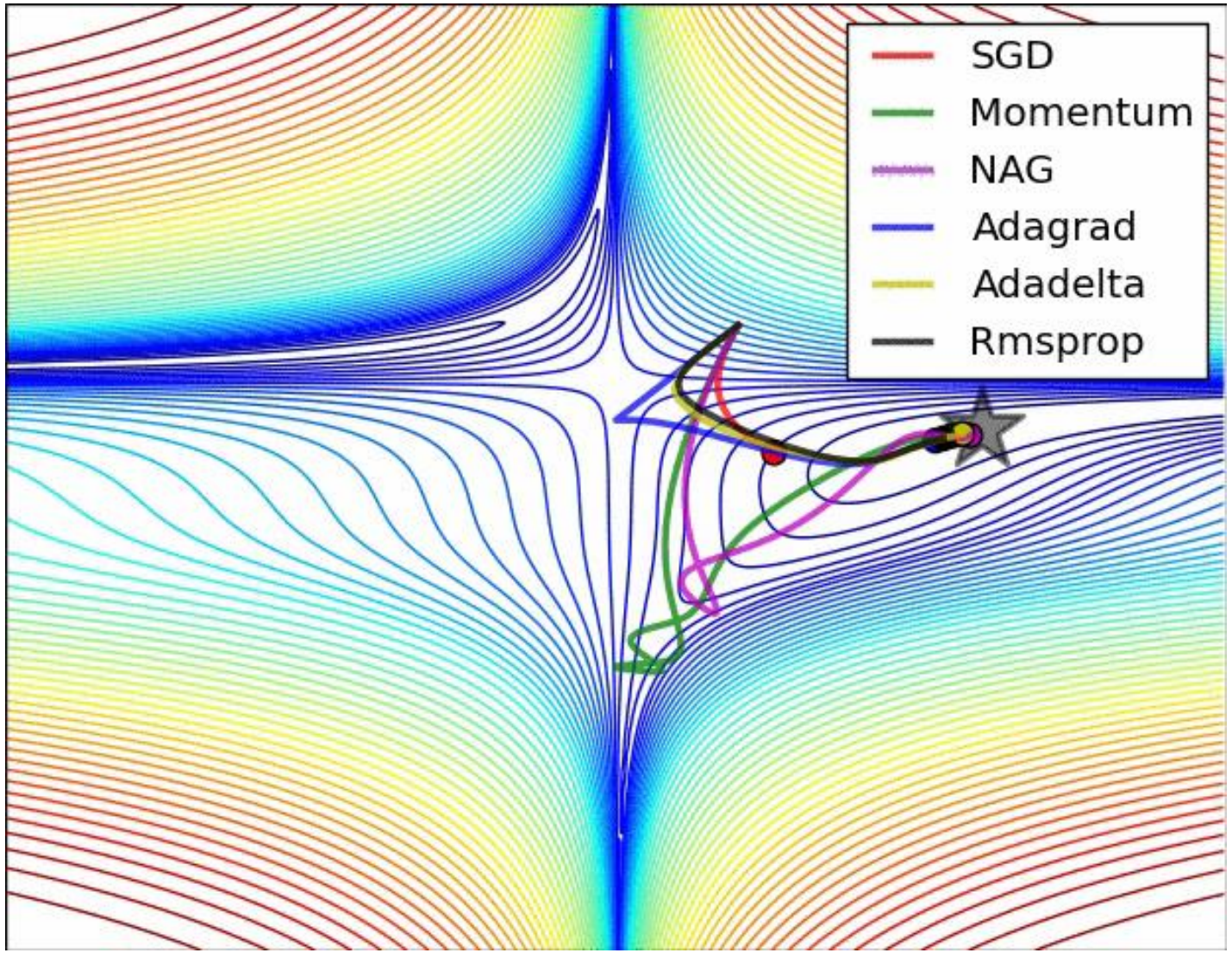
data vector

$$g_{t,i} = \frac{\partial}{\partial \theta_i^t} J_t(\theta) \quad , \theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} \cdot g_{t,i}$$

time step of iteration

sum of past time steps

- 별로 필요 없으면 → update하지 않고 값 유지
- stuck in a local optimal → 해당 parameter update 를 전혀 하지 않을 수도 있음
- 특정 epoch마다 reset도 가능
  - e.g., 4 epoch 마다 reset
  - AdaGrad 자체가 default 값도 성능이 좋음


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

# General Strategy for Successful NNets

- ~~문제에 적합한 네트워크 구조 고르기~~
  - ~~구조: single word, fixed window, 문서 레벨; bag of word, RNN, CNN~~
  - ~~비선형~~
- ~~Gradient check를 통함 debug~~
- ~~파라미터 초기화~~
- ~~최적화 tricks~~
- 모델 overfitting check
  - Overfitting 이라면 → 모델의 구조를 바꾸거나, 모델의 사이즈를 더 크게
  - Overfitting 이 아니라면 → Regularize

# Prevent Overfitting: Model Size & Regularization

- **Reduce Model Size**
- Weight에 표준 L1 or L2 정규화
- Early Stopping: 가장 좋은 validation error 주는 parameter 에서 stop
  - 알맞은 수의 parameter 를 가지고 있는데 너무 오랫동안 기울기가 flat하고, overfit 하면 그냥 stop
- ~~Hidden activation 에 sparsity constraints 주기~~ (잘 안 씀)

**Tips!**

가장 보편적인 방법은 마지막 50개의 iteration의 weight를 저장하고,
하나의 epoch마다의 weight를 저장한 후 validation error를 확인
그 중에서 best validation error를 찾아
(보통 이틀정도, 데이터가 크면 3~4일 정도 돌려보면 알 수 있음)

Good indicator 가 될 거임!

# Prevent Feature Co-adaptation

- Dropout
  - Popular technique
- Main idea: training time 에 각 layer마다 랜덤하게 input의 50%를 delete
- Hurting the model
  - Neural net은 memorizing 기능이 상당히 좋아서 model 이 input의 pixel 값까지 기억하고, user에게 그에 정확히 상응하는 output 을 출력해 줄 것임

    → Overfitting!
  - 이를 dropout으로 방지

# 감사합니다