Optimization for Training Deep Models

Deep learning reading group

Henrique Morimitsu

December 13, 2016

INRIA

Presentation based on Chapter 8 of the Deep Learning book by [Goodfellow et al., 2016]

Table of contents

What is in here?

If you want to know:

- Best batch size?
- How to avoid local minima?
- Best training algorithm?
- Optimal learning rate?
- Best weight initialization policy?

What is in here?

If you want to know:

- Best batch size?
- How to avoid local minima?
- Best training algorithm?
- Optimal learning rate?
- Best weight initialization policy?

No luck for you

What is in here?

However, if you want to **understand**:

- How batch size affects training
- The problems of local minima and saddle points
- Different training algorithm
- Importance of learning rate
- Existing initialization policies and their implications

This is for you

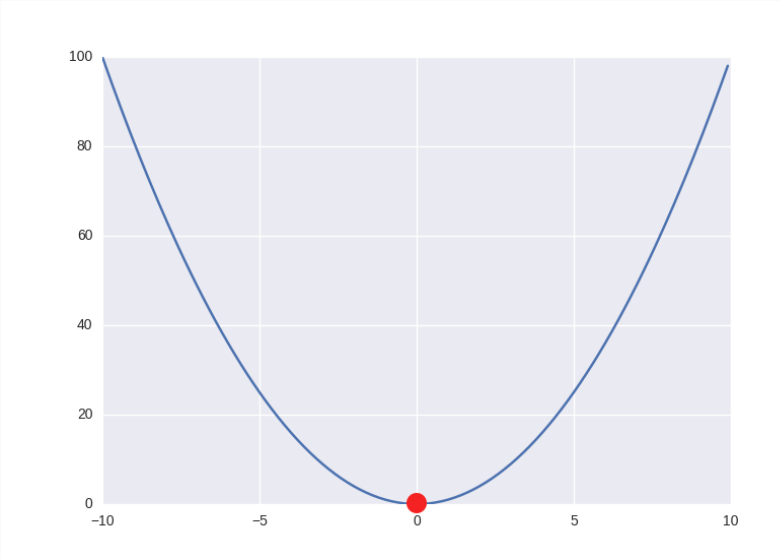
Difference between learning and pure optimization

What is optimization?

The process to find maxima or minima based on constraints

- Involved in many contexts of deep learning, but the hardest one is neural network training
    - Expend days to months to solve a single instance
    - Special techniques have been developed specifically for this case
- Presentation focus on one particular case of optimization:

Finding the parameters $\boldsymbol{\theta}$ of a network that reduce a cost function $J(\boldsymbol{\theta})$



Difference between learning and pure optimization

- Several differences
- Machine learning acts indirectly, unlike optimization
  - Usually we want to optimize a performance measure P, based on the test set
    - Problem may be intractable
  - Optimize a different cost function J instead
    - Hoping that it optimizes P as well

Example: recognizing cats

Application to recognize cats

- You don't know which images you may receive

Example: recognizing cats

Application to recognize cats

- You don't know which images you may receive

Example: recognizing cats

Application to recognize cats

- You don't know which images you may receive

Example: recognizing cats

- The performance measure P is the number of correct classifications
- But you don't have access to these images
- So instead you build a **training set**
- And measure the performance J on this set
- Minimizing J does not necessarily minimize P

Example: recognizing cats

Training

Test

Example: recognizing cats

Training

Test

Generalization error (loss function)

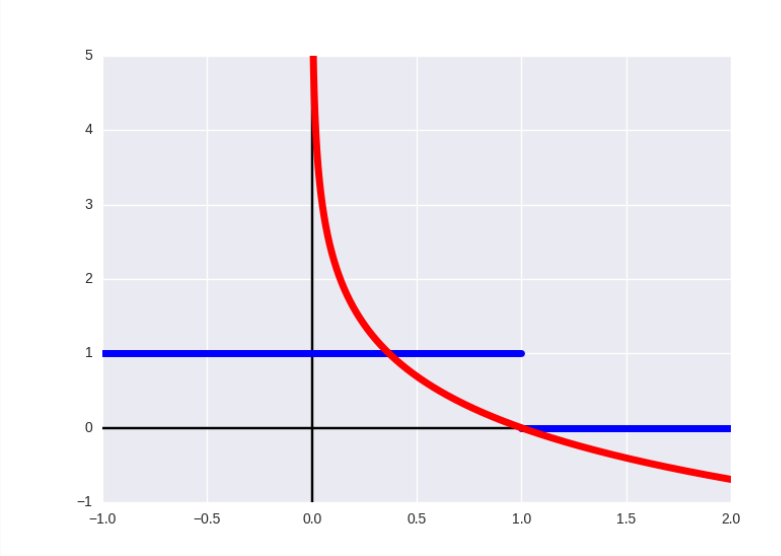
- Typically the loss function is a simple average over the training set:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(x,y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x}, \boldsymbol{\theta}), y) = \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \tag{1}$$

  where
  - $\mathbb{E}$ is the expectation operator
  - L is the per-example loss function
  - $f(\boldsymbol{x}, \boldsymbol{\theta})$ is the predicted output for input $\boldsymbol{x}$
  - $\hat{p}_{\text{data}}$ is the empirical distribution (notice the hat)
  - y is the target output
  - m is the number of training examples

Surrogate loss

- Often, minimizing the real loss is intractable (e.g. 0-1 loss)
- Minimize a surrogate loss instead
- E.g. the negative log-likelihood is a surrogate for the 0-1 loss
- Sometimes, the surrogate loss may learn more
    - Test error 0-1 loss keeps decreasing even after training 0-1 loss is zero
    - Even if 0-1 loss is zero, it can be improved by pushing the classes even farther from each other



Batch and minibatch algorithms

- In machine learning, the objective function decomposes as a sum
- Compute each update based on the cost function of a subset
- For example, the often used gradient is based on the expectation of the training set

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x};\boldsymbol{\theta}), y) \tag{2}$$

- Computing the expectation exactly at each update is very expensive

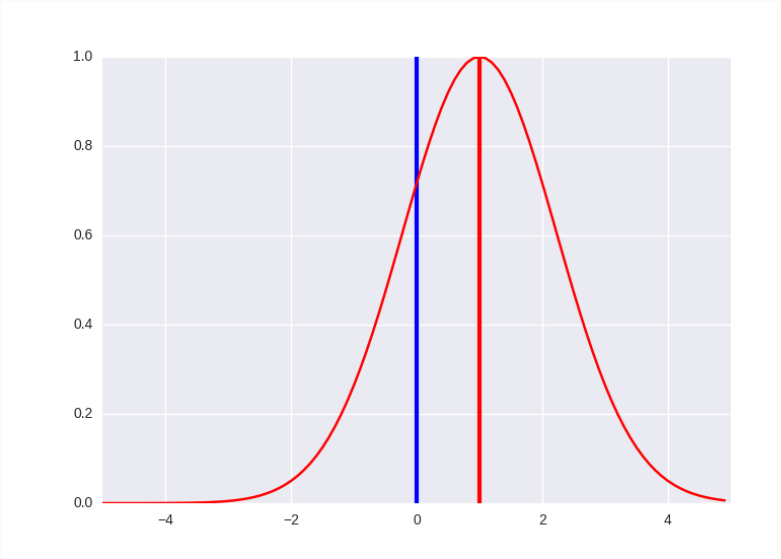
Batch and minibatch algorithms

- The gain of using more samples is less than linear
  - Estimator for the true mean μ of a distribution
  - How far the estimated mean $\hat{\mu}$ is?
  - Compute the standard deviation:

$$SE(\hat{\mu}_m) = \sqrt{\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}$$

(??)

  - Error drop proportionally only to the square root of m



Batch and minibatch algorithms

- Generate batches by random sampling
- Random sampling may also have some benefits:
  - Redundant training set $\rightarrow$ less samples for correct gradient
  - Does not happen in practice, but many samples contribute similarly

Batch and minibatch algorithms

- Algorithms that use the whole training set are called **batch** or **deterministic**
- If only subsets are used, it is called **minibatch**
  - However, many times minibatch algorithm are simply referred to as batch
- If the algorithm uses only one example, it is called **stochastic** or **online**
- Most algorithm use between 1 and a few examples
- They are traditionally called **minibatch**, **minibatch stochastic** or simply **stochastic**
  - The canonical example is **stochastic gradient descent, which will be covered later**

Batch and minibatch algorithms

- Considerations about minibatch size:
  - Larger batches are more accurate, but the return is less than linear
  - Very small batches do not make good use of multicore capabilities
  - If processed in parallel, the amount of memory scales with the batch size
  - Some hardware achieve better performance with specific sizes (typically a power of 2)
  - Small batches may offer a regularizing effect, probably due to the noise they add
    - But may require small learning rates to keep stability
    - Number of steps for convergence may increase

Batch and minibatch algorithms

- Minibatches must be selected randomly, for unbiased estimation
- Subsequent estimates must also be independent of each other
- In practice, we do not enforce true randomness, but just shuffle the data

Batch and minibatch algorithms

- Minibatch stochastic gradient descent follows the gradient of the true generalization error
  - Most algorithms shuffle the data once and then pass through it many times
  - In the first pass, each minibatch gives an unbiased estimate
  - In the next passes it becomes biased

Challenges in network optimization

Ill conditioning

- Condition number: how a change in input affects output
- High value greatly propagate errors
- May cause the gradient to become ”stuck”
    - Even small steps increase the cost

When ill conditioning is a problem

- Approximate cost function by a quadratic Taylor expansion

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^T g + \frac{1}{2}(x - x^{(0)})^T H (x - x^{(0)}) \quad (3)$$

where:

  - g is the gradient
  - H is the Hessian matrix

When ill conditioning is a problem

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^T g + \frac{1}{2}(x - x^{(0)})^T H (x - x^{(0)}) \quad (3)$$

- Updating using a learning rate of ϵ:

$$f(x - \epsilon g) \approx f(x^{(0)}) - \epsilon g^T g + \frac{1}{2}\epsilon^2 g^T H g \quad (4)$$

- The first term is the squared gradient norm (positive)
- If second term grows too much, the cost increases
- Monitor both terms to see if ill conditioning is a problem

Local minima

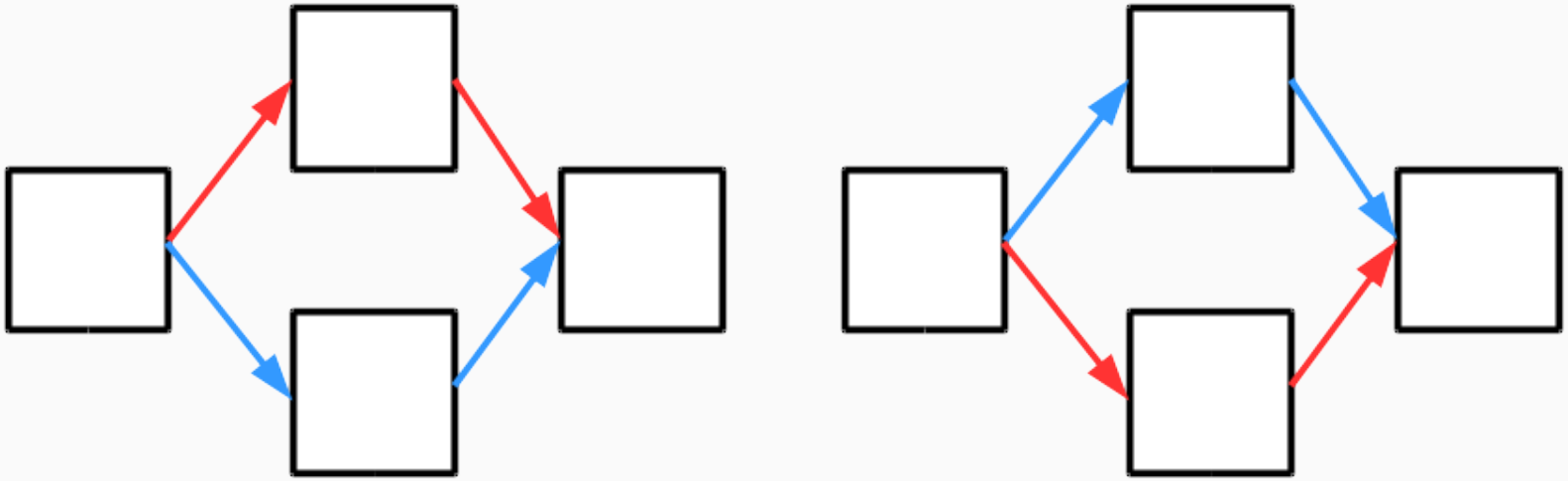
- In convex problems, any local minimum is a global minimum
- Non-convex functions may have several local minima
  - Neural networks are non-convex!
  - But actually, not a major problem

Model identifiability

- A model is **identifiable** if a large training set yields a unique set of parameters
- Models with latent variable are often not identifiable
- Neural nets are not identifiable
- The input and output weights of any ReLU or maxout units may be scaled to produce the same result

Weight space symmetry

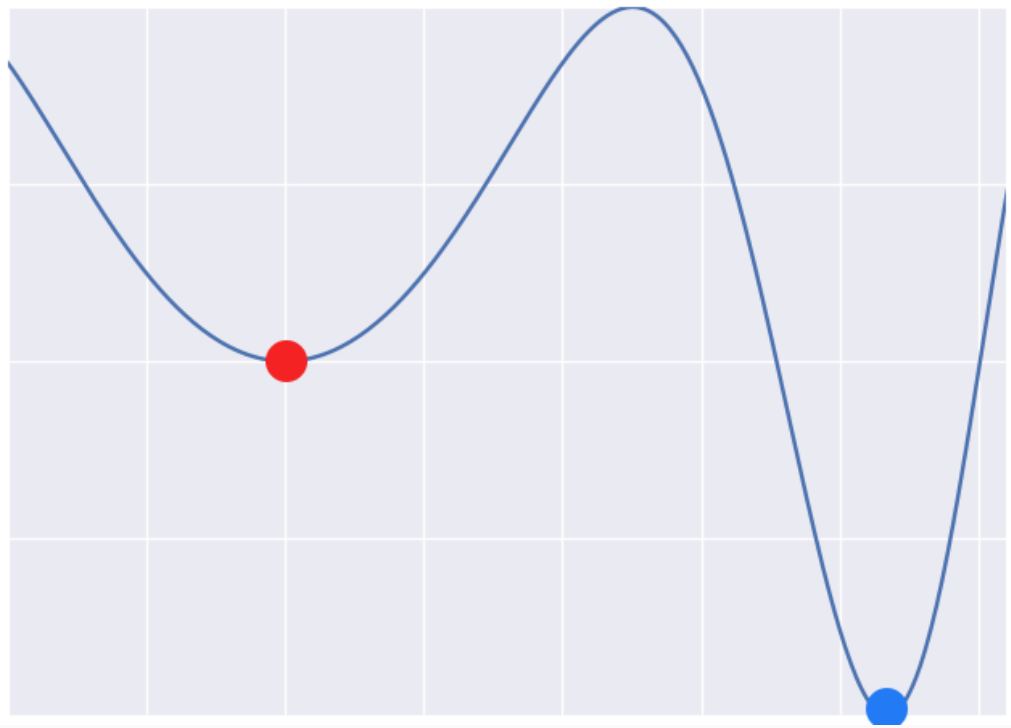
- Also, the same result can be obtained by swapping the weights between units



- This is known as **weight space symmetry**
- There are $n!^m$ permutations of the hidden units

Local minima

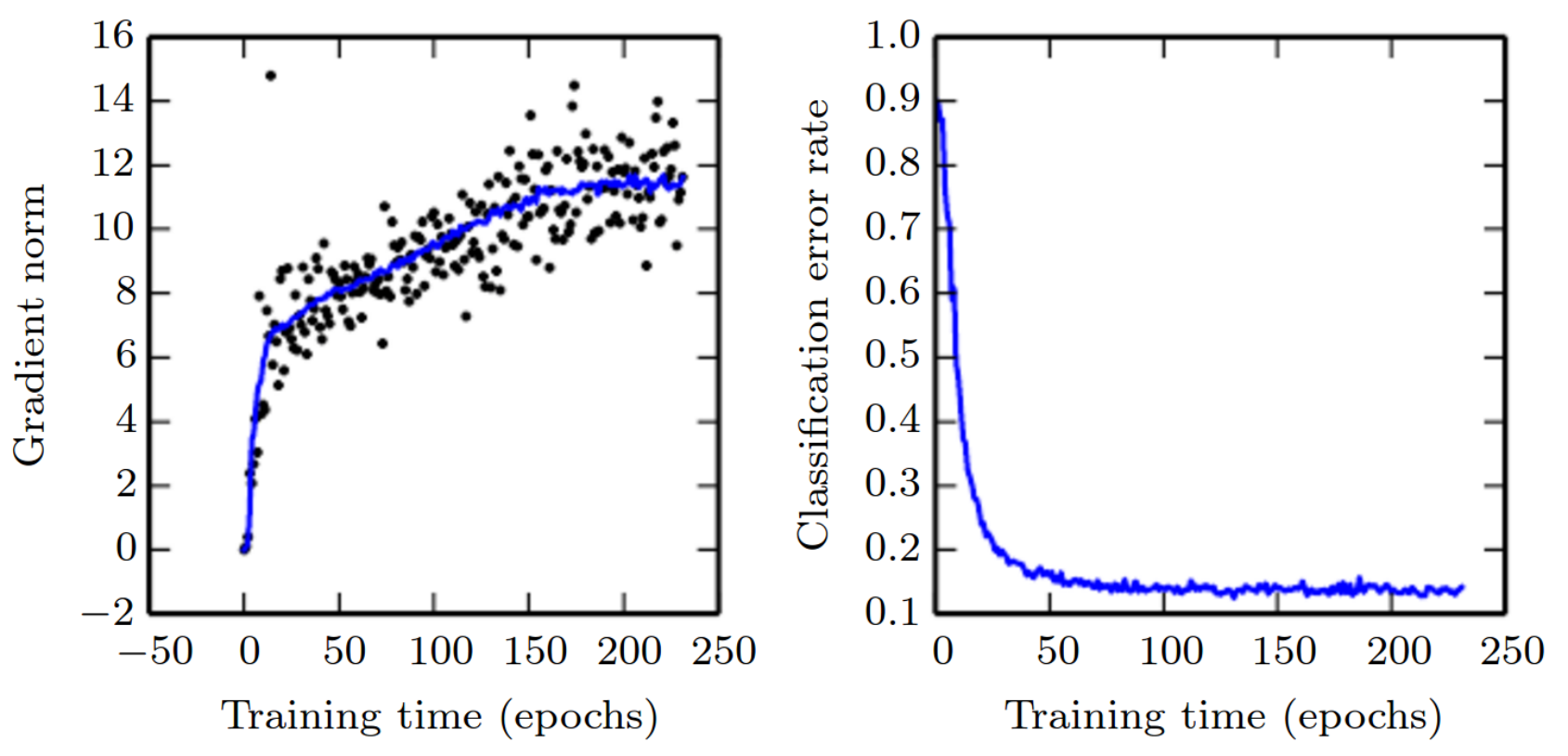
- Therefore, the cost function may have and uncountable number of local minima
  - But they are not a big problem
- Local minima are a problem when their cost is much higher than the global minimum



Local minima

- Local minima is still an open problem
- But nowadays many believe most local minima have a low cost
  [Choromanska et al., 2015, Dauphin et al., 2014, Goodfellow et al., 2015, Saxe et al., 2013]
- Test to rule out local minima problem:
  - Plot gradient norm along time

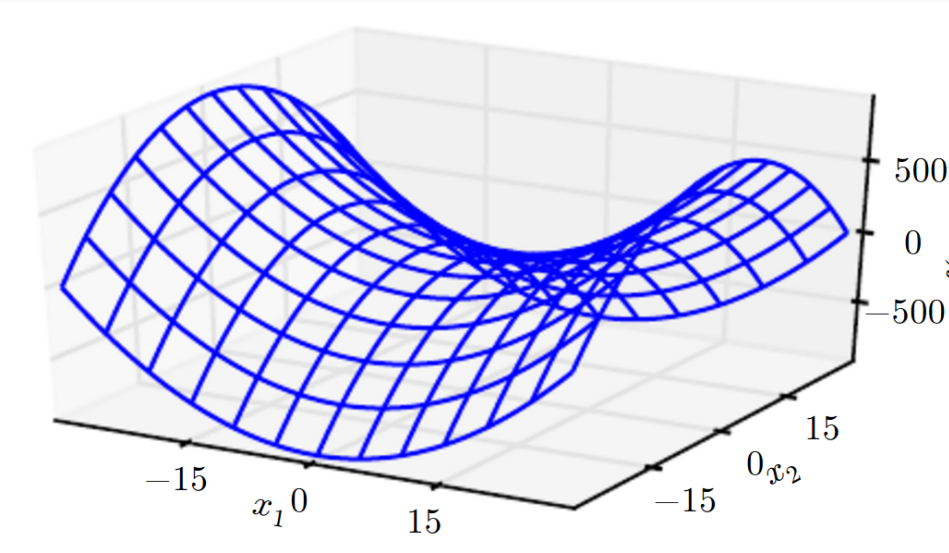
Local minima



Source: [Goodfellow et al., 2016]

Saddle points and flat regions

- Many attribute the majority of problem to local minima
- But there is another type of point that appears even more often and cause problems
  - Saddle points
  - Gradient is zero



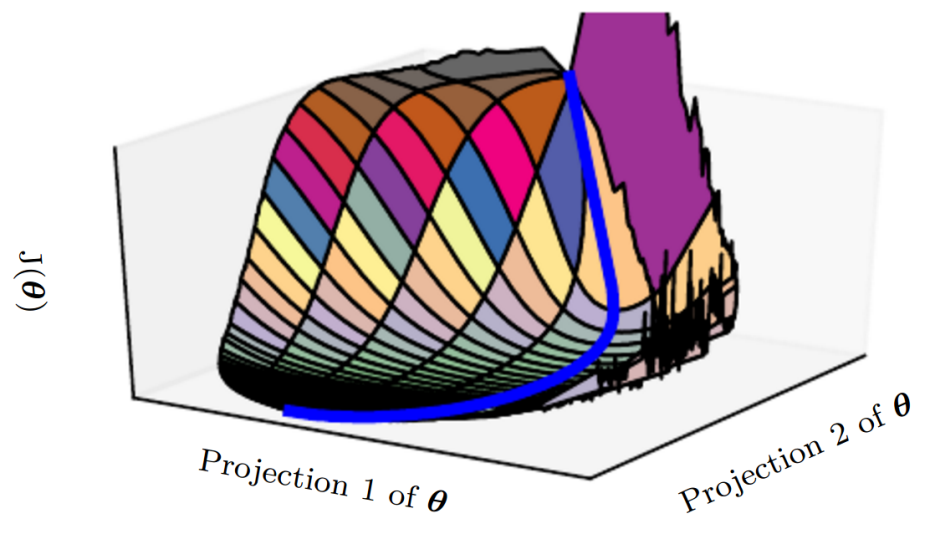
Source: [Goodfellow et al., 2016]

Are there many saddle points?

- The eigenvalues of a Hessian are all positive at a minimum
- Saddle points contain a mix of positive and negative eigenvalues
- Suppose the cost function is random
- The sign of the Hessian can be determined by a coin flip
- A minimum only happens if all coins are heads
  - At high dimension, extremely more likely to find a saddle point
- Studies showed it may happen in practice [Saxe et al., 2013]

Are saddle points a problem?

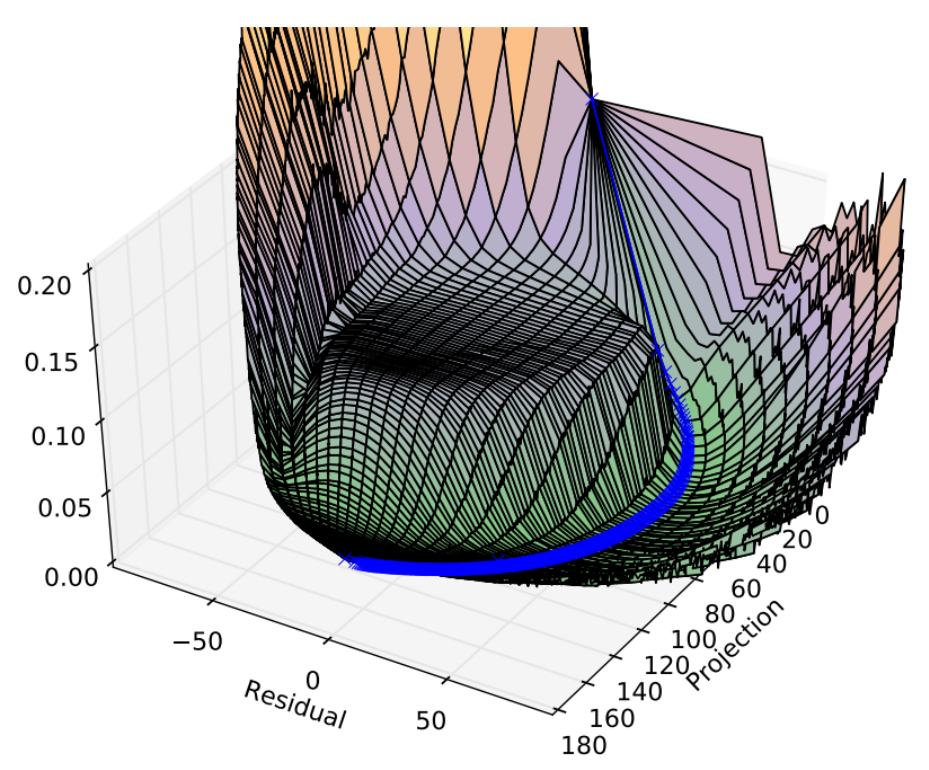
- For second-order methods (Newton's method), yes!
  - But research in this direction [Dauphin et al., 2014]
- For gradient, it is unclear
  - Intuitively, it may be
  - But studies show it is not [Goodfellow et al., 2015]



Source: [Goodfellow et al., 2016]

Are saddle points a problem?

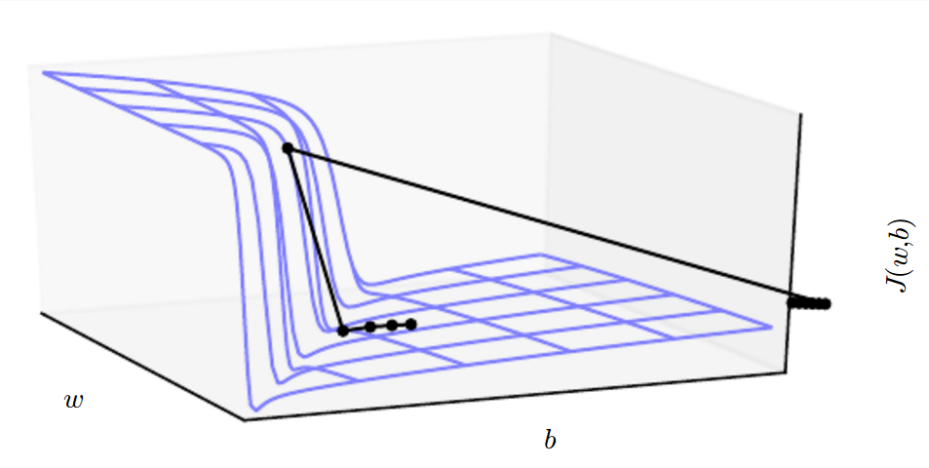
- Flat regions also have zero gradient
- Takes a long time to traverse
- Gradient wastes time circumnavigating tall mountains



Source: [Goodfellow et al., 2015]

Cliffs and exploding gradients

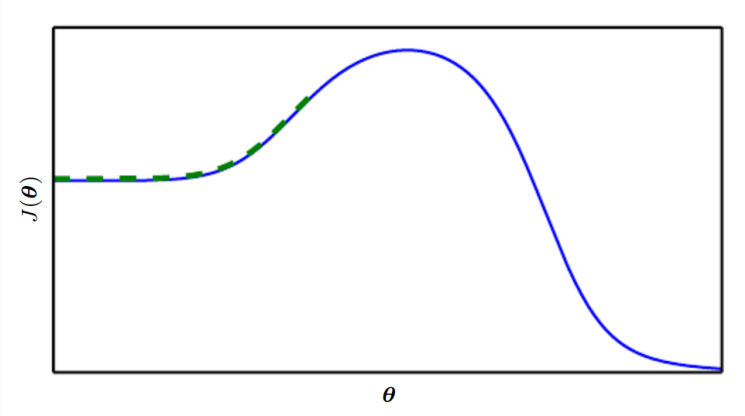
- Very steep cliffs cause gradients to explode
- More often in recurrent networks



Source: [Goodfellow et al., 2016]

Local and global structure

- Gradient only look at local structure
  - Actually, most useful approaches also have the same problem!
- If initialized at a bad position, cannot find the lowest minima
  - However, less likely in high dimension



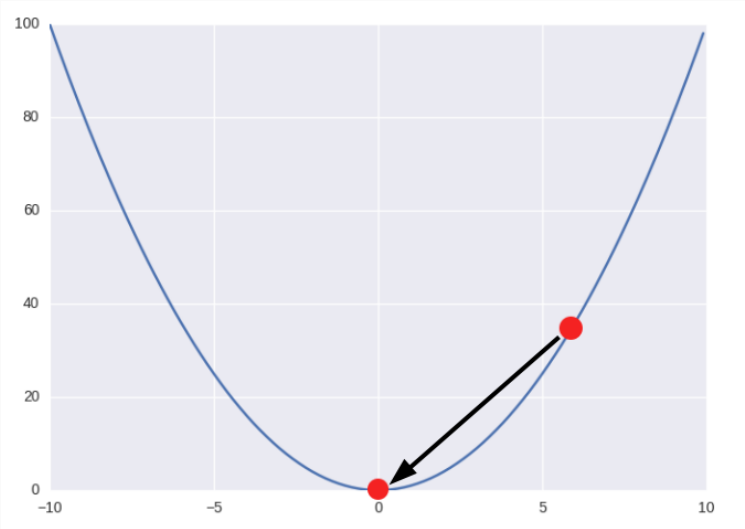
Source:

[Goodfellow et al., 2016]

Basic algorithms

Stochastic gradient descent (SGD)

- Most used algorithm for deep learning
- Do not confuse with (deterministic) gradient descent
    - Stochastic uses minibatches
- Algorithm is similar, but there are some important modifications



Gradient descent algorithm

- Full training samples $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
- Compute gradient

$$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \left(\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right) \tag{5}$$

- Apply update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g} \tag{6}$$

  where

  - ϵ is the learning rate
  - $\boldsymbol{\theta}$ are the network parameters
  - $L(\cdot)$ is the loss function

Stochastic gradient descent algorithm

- Minibatch of training samples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with targets $\boldsymbol{y}^{(i)}$
- Compute gradient

$$\hat{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \left(\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right) \tag{7}$$

- Apply update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{g} \tag{8}$$

Learning rate for SGD

- Learning rate ϵ_k must be adaptive
  - Minibatches introduce noise that do not disappear along even at the minimum
- Sufficient condition for convergence:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty \tag{9}$$

- Implying $\lim_{k\to\infty} \epsilon_k = 0$

Learning rate for SGD

- It is common to decay the learning rate linearly until iteration τ
  - Can also be decayed at intervals

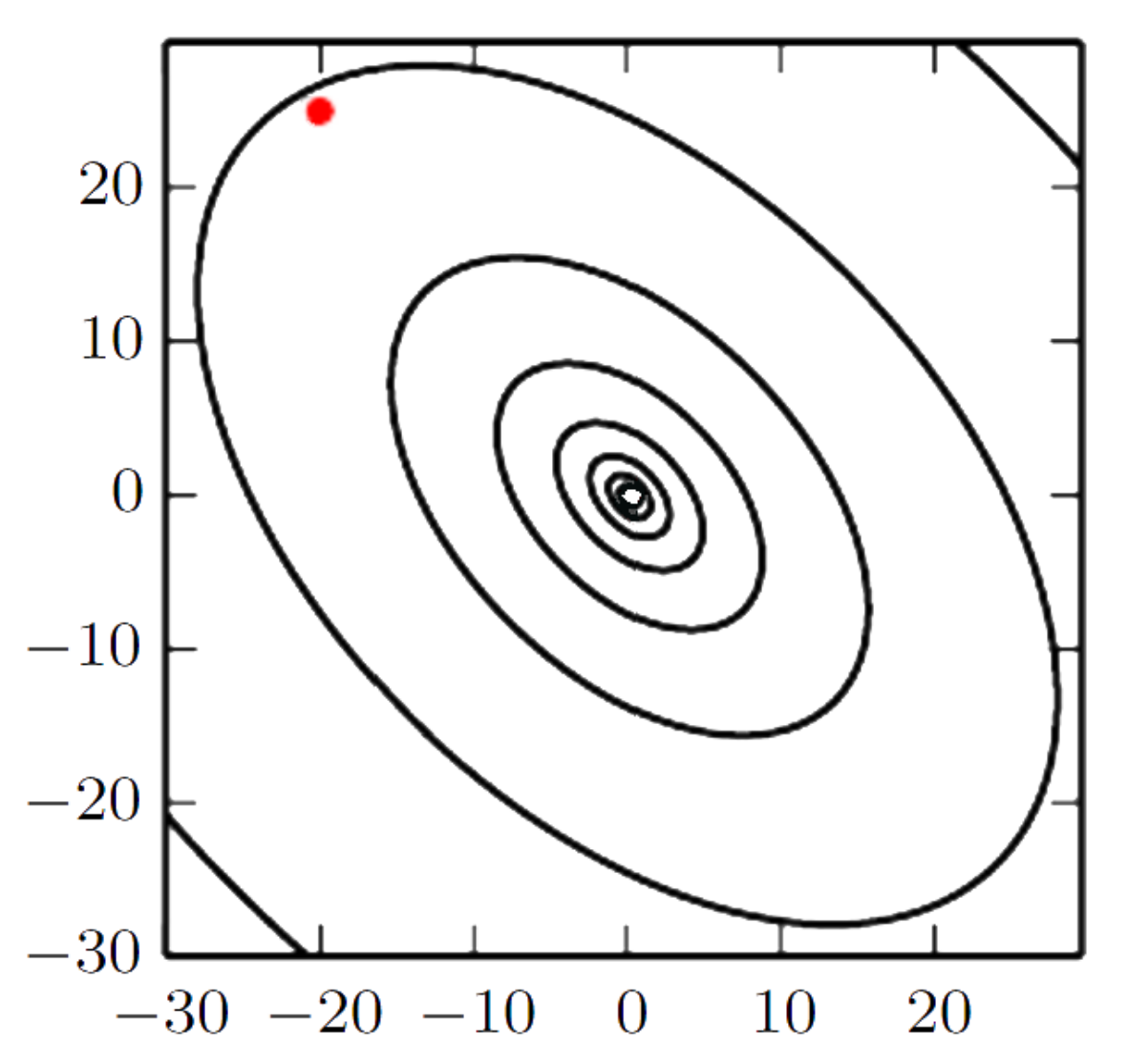
$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{10}$$

- Three parameters to choose
- Usually:
  - τ should allow a few hundred passes through the training set
  - ϵ_τ should be roughly 1% of ϵ_0

Learning rate for SGD

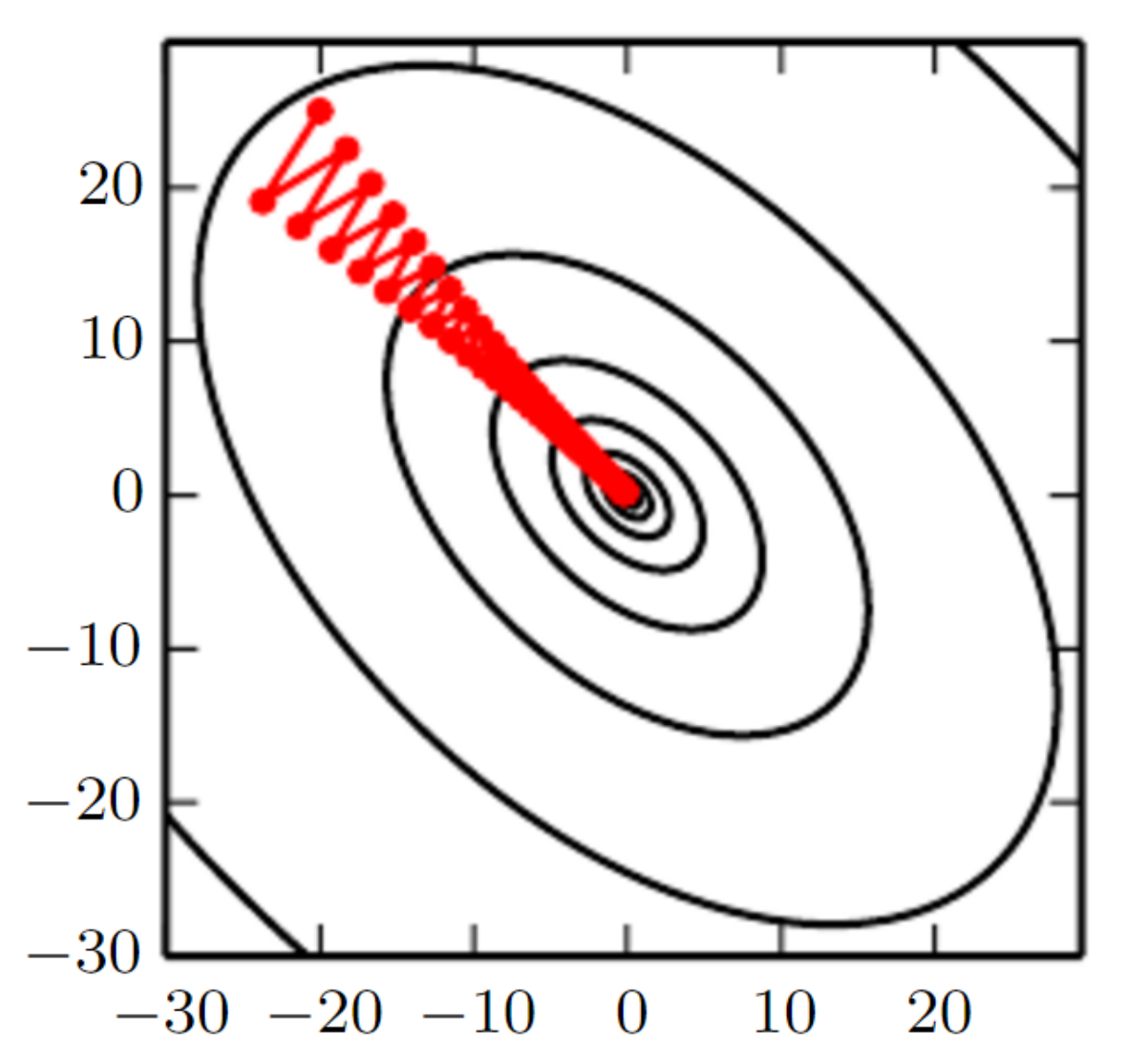
- The main problem is how to choose ϵ_0
- Tipically:
  - Higher than the best value for the first 100 iterations
  - Monitor the initial results and use a higher value
  - Too high will cause instability

Momentum



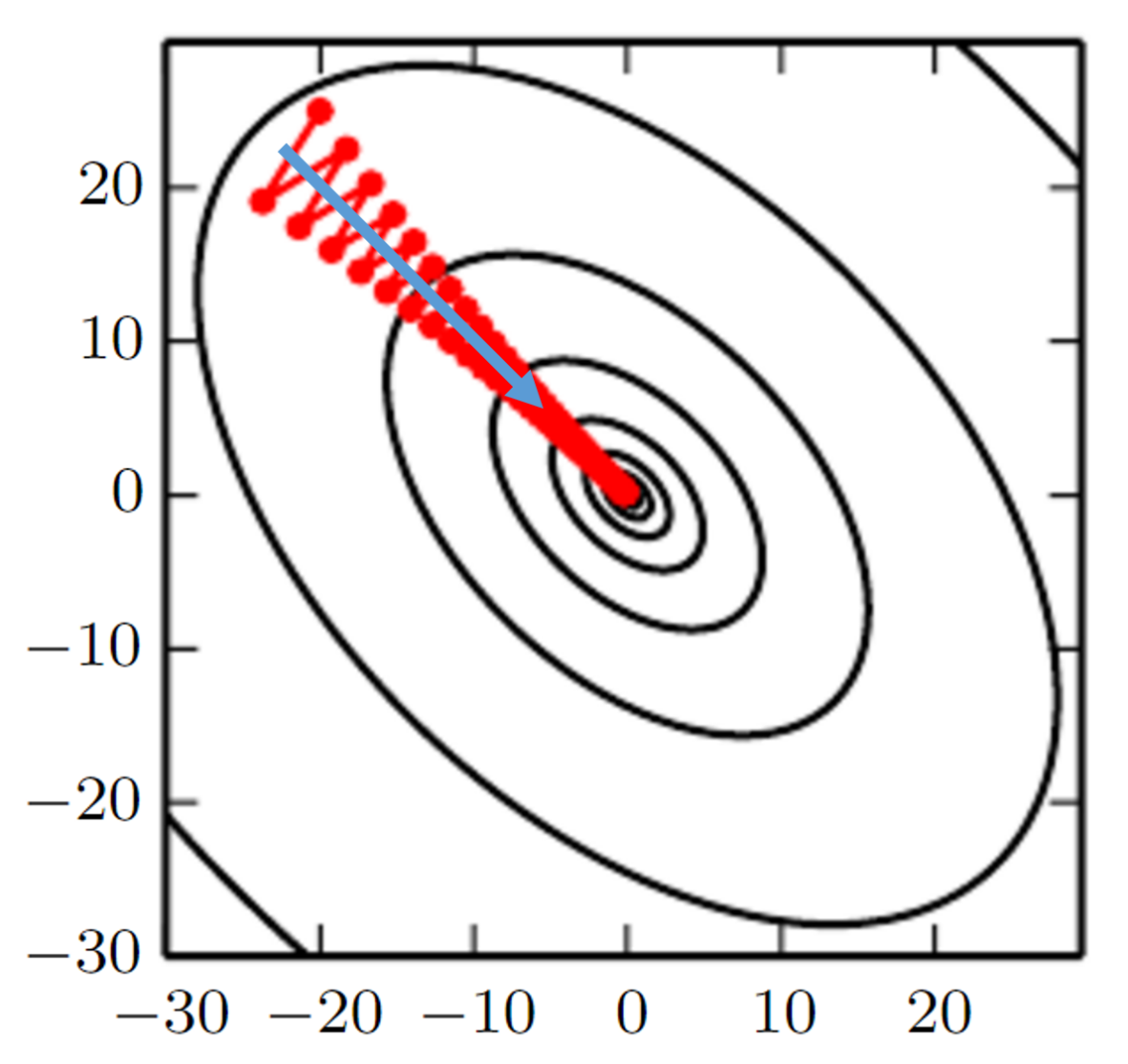
Adapted from [Goodfellow et al., 2016]

Momentum



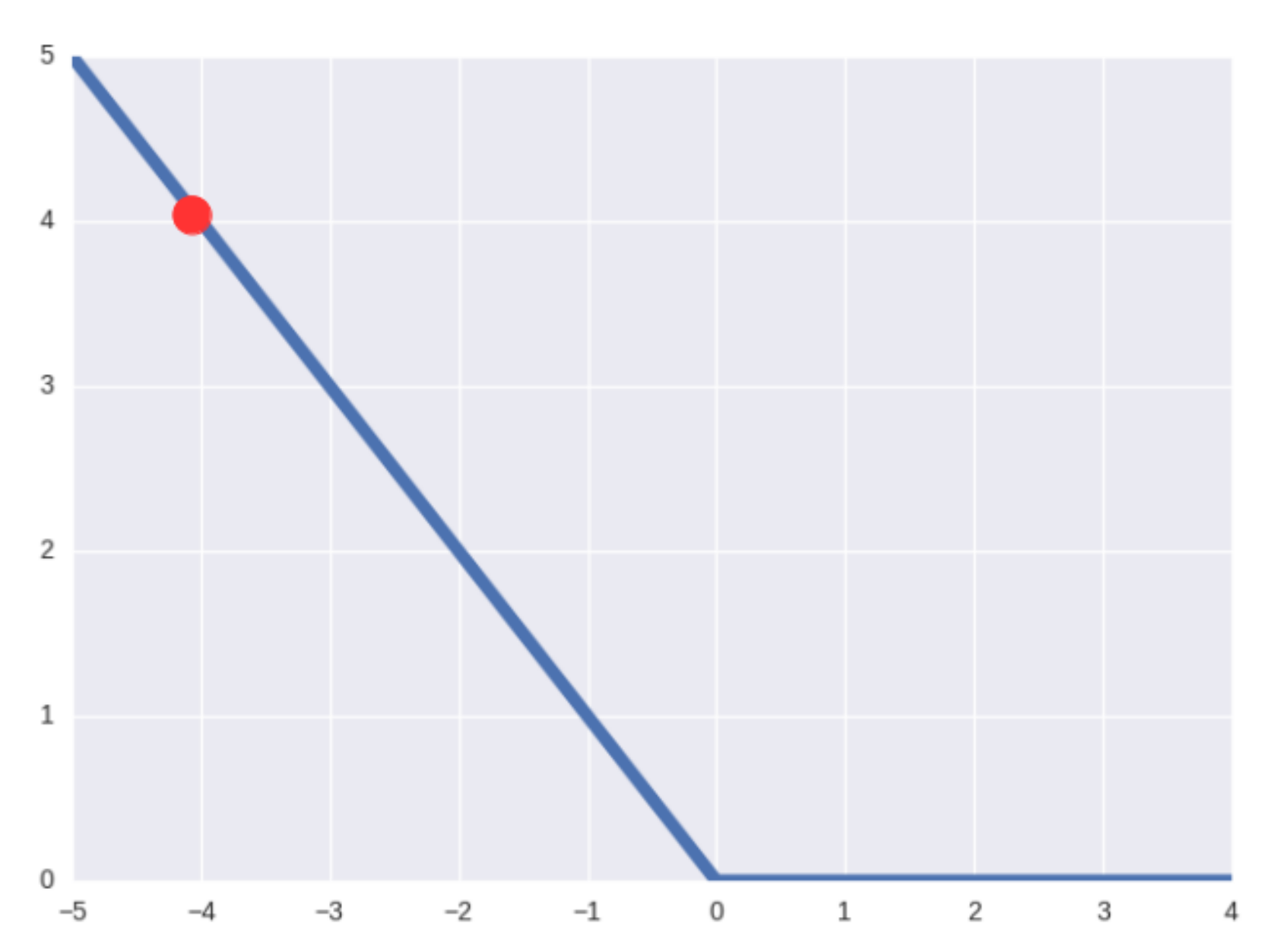
Source: [Goodfellow et al., 2016]

Momentum

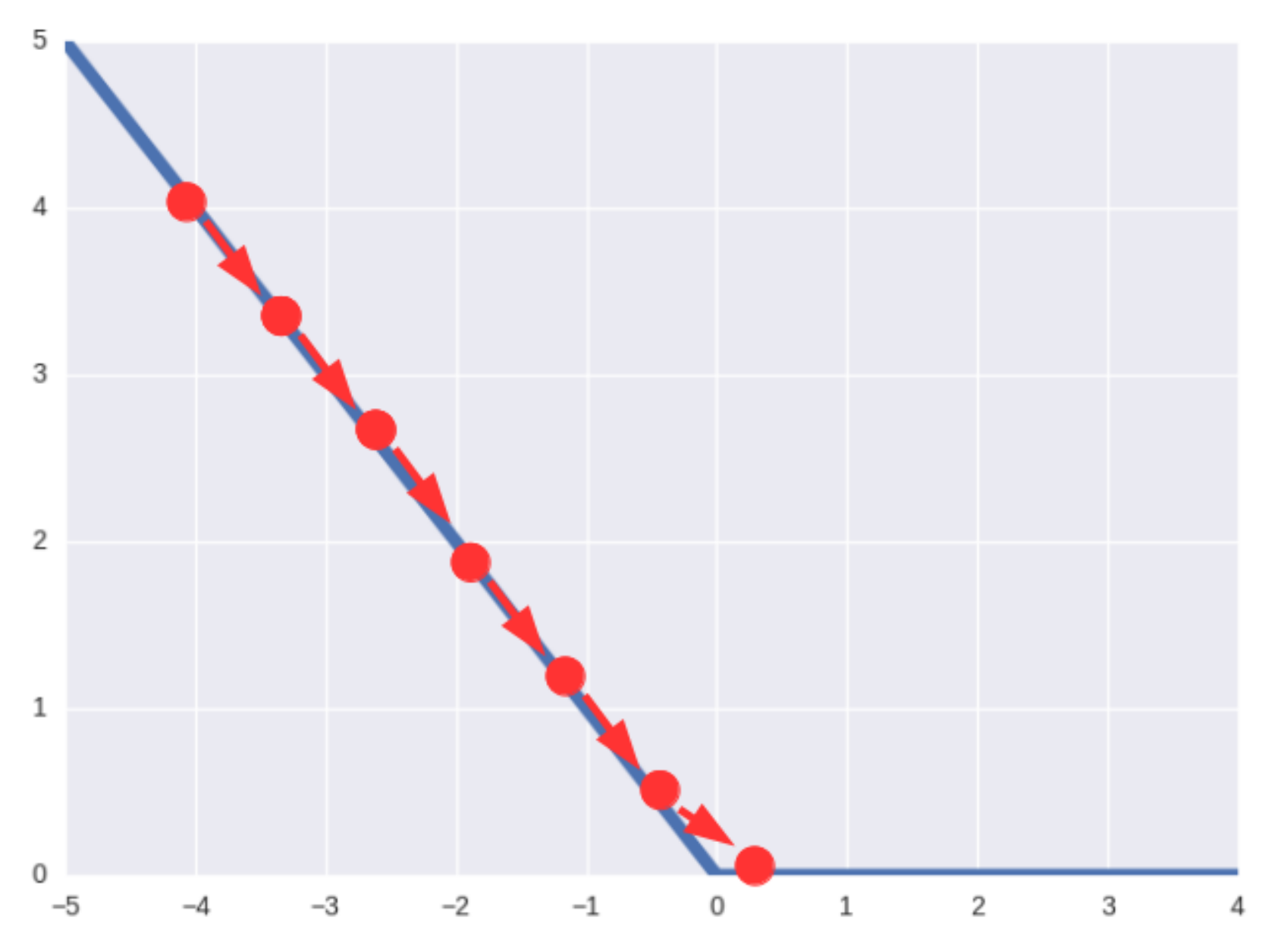


Adapted from [Goodfellow et al., 2016]

Momentum



Momentum



Momentum

- In these cases, momentum can help
- Derived from the physics term (= mass $\times$ velocity)
- Assume unit mass, so just consider velocity

Momentum

- Accumulates previous gradients

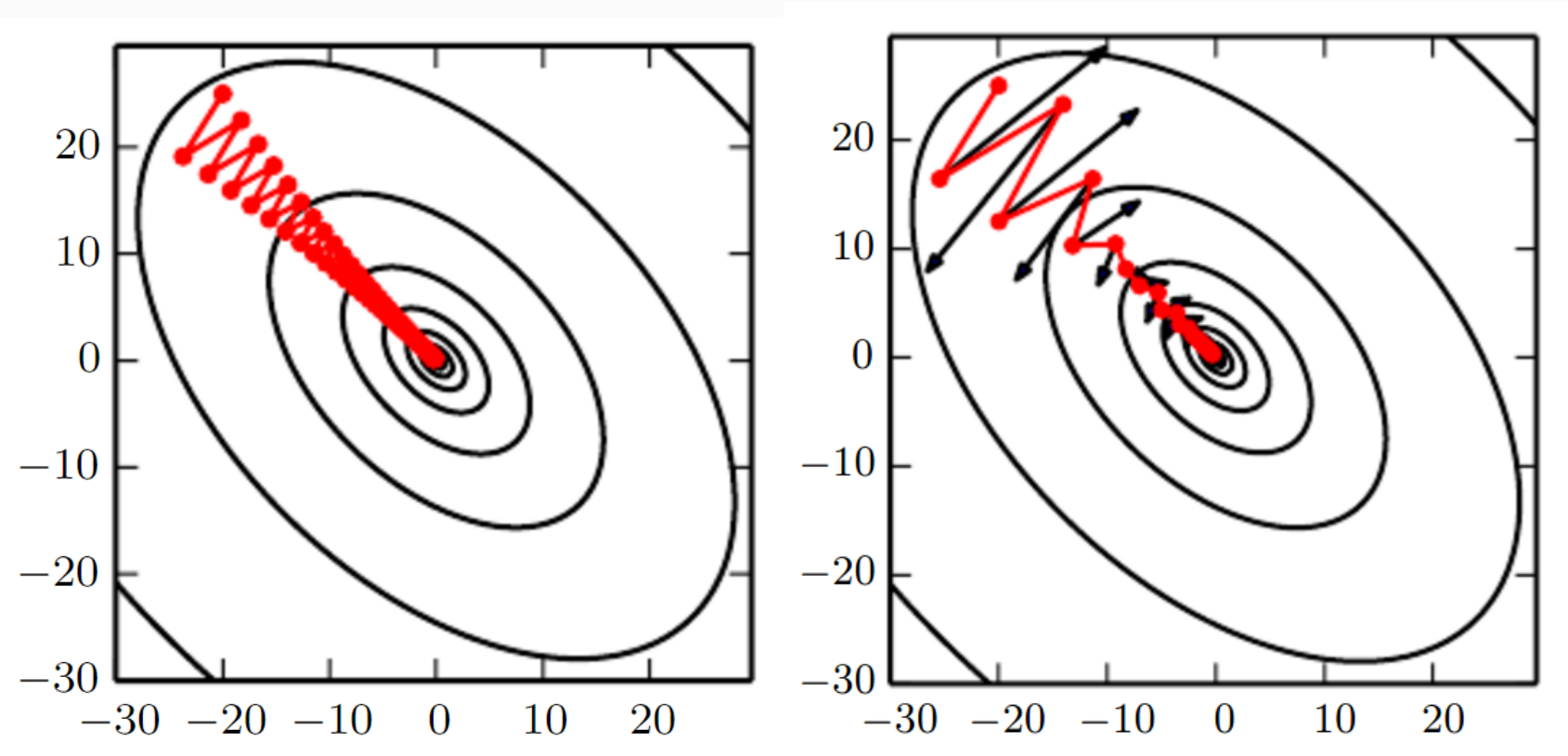
$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right) \tag{11}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \tag{12}$$

where

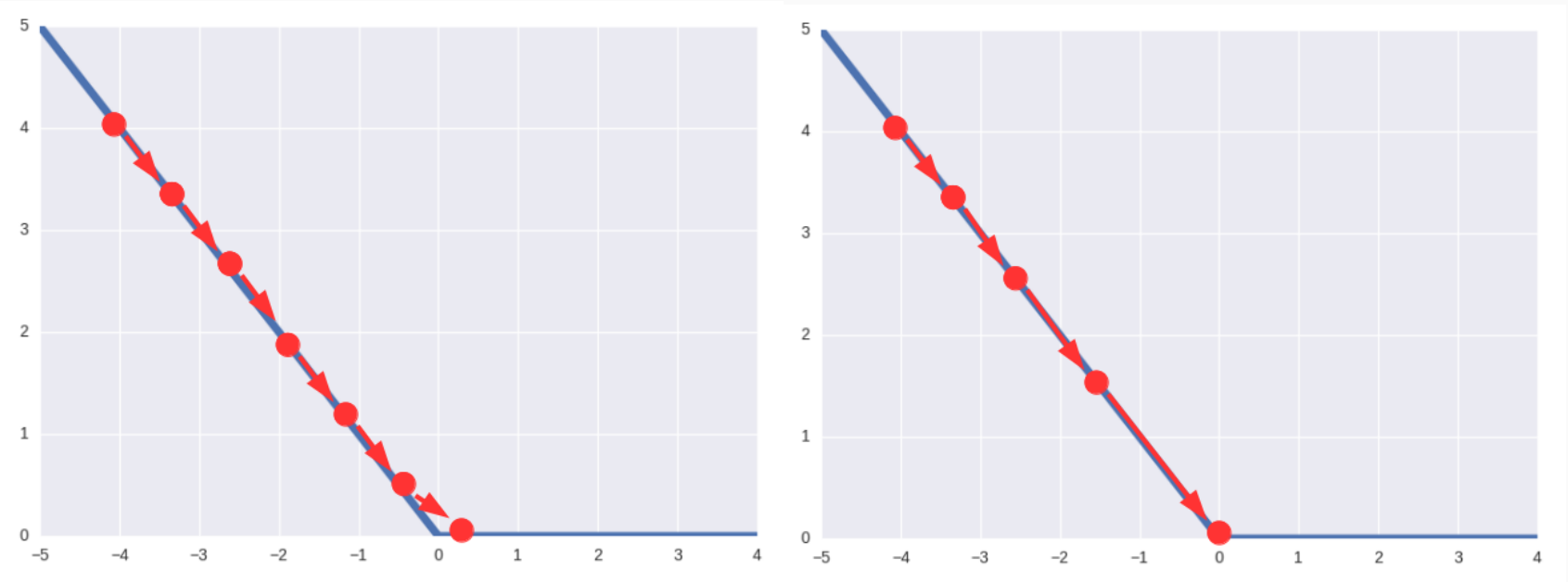
- $\alpha \in [0, 1)$ is a hyperparameter
- $\nabla_{\boldsymbol{\theta}}$ is the gradient
- $\boldsymbol{\theta}$ are the network parameters

Momentum



Source: [Goodfellow et al., 2016]

Momentum



SGD with momentum

- Compute gradient

$$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \left(\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right) \tag{13}$$

- Compute velocity update

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g} \tag{14}$$

- Apply update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v} \tag{15}$$

where

  - α is the momentum coefficient

Parameter initialization strategies

Parameter initialization

- Deep learning methods are strongly affected by initialization
- It can determine if the algorithm converges at all
- Network optimization is still not well understood
- Modern techniques are simple and heuristic

Breaking symmetry

- It is known that initialization must break symmetry between units
  - Units with same inputs and activations will be updated the same way
  - Even if using dropouts, it is better to avoid symmetry
- Common initialization choices are random and orthogonal matrices
  - The former is cheaper and works well
- Biases, however, are usually constants chosen heuristically

Gaussian initialization

- Choice of Gaussian function does not seem to affect much
- However, the scale of the distribution does matter
  - Larger weights break symmetry more, but may explode

How to choose the mean?

- The weight can be interpreted as how much units interact with each other
- If initial weight is high, we put a prior about which units should interact
- Therefore, it is a good idea to initialize the weights around zero

Normalized initialization

- Some heuristics are adapted to input/output sizes (Glorot and Bengio, 2010)

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \tag{16}$$

- Enforce same forward / backward variance between layers
- Derived from a network with no non-linearities
- But in practice works well in other networks as well
- Xavier initialization in Caffe

Sparse initialization

- If the layer is huge, normalized initialization yields very low weights
- Martens (2010) propose to have exactly k non-zero weights at each layer
- This helps to keep higher values and increase diversity
- But put a strong prior on some connections
- It may take a long time to fix wrong priors

Orthogonal matrices

- [Saxe et al., 2013] recommend using orthogonal weight matrices for initialization
- If using a non-linear function ϕ that saturates as $x \to \infty$, there exists $\gamma > \gamma_0$ in

$$x_i^{l+1} = \sum_j \gamma W_{i,j}^{(l+1,l)} \phi(x_j^l) \tag{17}$$

that avoid vanishing gradients despite the number of network layers, where:

  - x_i^l is the activity of neuron i in layer l
  - $W_{i,j}^{(l+1,l)}$ is a random orthogonal weight matrix

Bias initialization

- Bias initialization is typically easier
- It is common to initialize them as zero
- Sometimes, we might want to use other constants:
  - For output units, it may be beneficial to initialize them according to the marginal statistics of the output
  - Avoid saturation, e.g., 0.1 for ReLU
  - Initialize as 1 for gate units (LSTM)

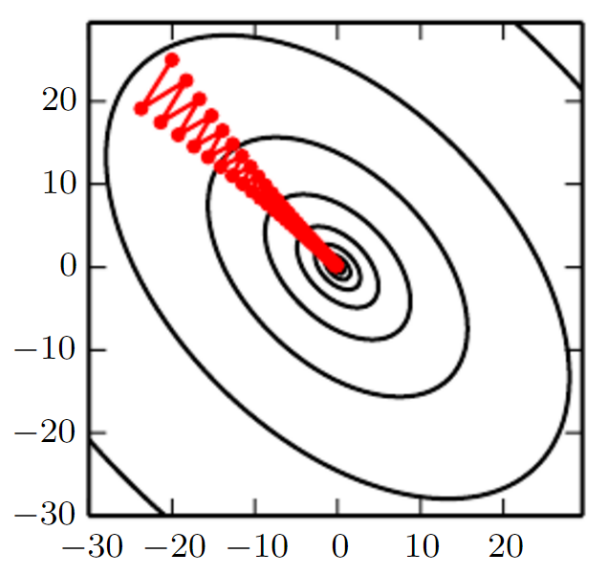
Considerations about initialization

- Often, even optimal initialization does not lead to optimal results
- This may happen because:
  - Wrong criteria. Maybe it is not good to preserve the norm through the network
  - Initial properties may be lost during training
  - Speed increases, but we may be sacrificing generalization capability
- If resources allows it, initialization should be a hyperparameter of the network

Algorithms with adaptive learning rate

Adaptive learning rate

- Learning rate is one of the hyperparameter that impacts the most
- The gradient is highly sensitive to some directions
- If we assume that the sensitivity is axis-aligned, it makes sense to use separate rates for each parameter



Source:
[Goodfellow et al., 2016]

Delta-bar-delta [Jacobs, 1988]

- Early heuristic approach
- Simple idea: if the partial derivative in respect to one parameter remains the same, increase the learning rate, otherwise, decrease
- Must be used in batch methods

AdaGrad [Duchi et al., 2011]

- Scale the gradient according to the historical norms
- Learning rates of parameters with high partial derivatives decrease fast
- Enforces progress in more gently sloped directions
- Nice properties for convex optimization
- But for deep learning decrease the rate in excess

AdaGrad algorithm

- Accumulate squared gradients

$$r \leftarrow r + g \odot g \tag{18}$$

- Element-wise update

$$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g \tag{19}$$

- Update parameters

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \tag{20}$$

where

- g is the gradient
- δ is a small constant for stabilization

RMSProp [Hinton, 2012]

- Modification of AdaGrad to perform better on non-convex problems
- AdaGrad accumulates since beginning, gradient may be too small before reaching a convex structure
- RMSProp uses an exponentially weighted moving average

RMSProp algorithm

- Accumulate squared gradients

$$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g} \tag{21}$$

- Element-wise update

$$\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g} \tag{22}$$

- Update parameters

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta} \tag{23}$$

where

  - ρ is the decay rate

Adam [Kingma and Ba, 2014]

- Adaptive Moments, variation of RMSProp + Momentum
- Momentum is incorporated directly as an estimate of the first order moment
  - In RMSProp momentum is included after rescaling the gradients
- Adam also add bias correction to the moments

Adam algorithm

- Update time step: $t \leftarrow t + 1$
- Update biased moment estimates

$$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g} \tag{24}$$

$$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g} \tag{25}$$

- Correct biases

$$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t} \tag{26}$$

$$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t} \tag{27}$$

- Update parameters

$$\Delta\boldsymbol{\theta} \leftarrow -\epsilon \frac{\hat{\boldsymbol{s}}}{\delta + \sqrt{\hat{\boldsymbol{r}}}} \tag{28}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta} \tag{29}$$

Thank you!

Questions?

References I

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015).
The loss surfaces of multilayer networks.
In AISTATS.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014).
Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.
In Advances in Neural Information Processing Systems, pages 2933–2941.

References II

- Duchi, J., Hazan, E., and Singer, Y. (2011).
  Adaptive subgradient methods for online learning and stochastic optimization.
  Journal of Machine Learning Research, 12(Jul):2121–2159.
- Goodfellow, I., Bengio, Y., and A., C. (2016).
  Deep Learning.
  MIT Press.

References III

- Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2015).
  Qualitatively characterizing neural network optimization problems.
  In International Conference on Learning Representations.
- Hinton, G. (2012).
  Neural networks for machine learning.
  Coursera, video lectures.

References IV

Jacobs, R. A. (1988).
Increased rates of convergence through learning rate adaptation.
Neural networks, 1(4):295–307.

Kingma, D. and Ba, J. (2014).
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980.

References V

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). **Exact solutions to the nonlinear dynamics of learning in deep linear neural networks.** In International Conference on Learning Representations.