EE488 Special Topics in EE <Deep Learning and AlphaGo>

Sae-Young Chung

Lecture 8

October 23, 2017

Chap. 8 Optimization for Training

- Stochastic gradient descent
- Minibatch
- Gradient descent with momentum
- Parameter initialization
- Adaptive learning rate
- Batch normalization
- Transfer learning
- Continuation method

Optimization for Learning

- Differences from plain optimization
  - Minimizes empirical risk (based on empirical distribution from training data) rather than the true risk (based on true distribution from test data), which may cause overfitting

$$\min_{\boldsymbol{\theta}} -\sum_{i=1}^{m} \log p_{\text{model}}(\mathbf{y}^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta})$$

  - Often minimizes a surrogate loss function (e.g., NLL) instead of the true loss function (e.g., 0-1 loss, e.g., $I(y \neq \hat{y})$) since then we get gradient information. Also, minimizing a surrogate loss function can make learning more robust, e.g., by making different classes more apart from each other.
  - Does not stop at a local minimum, e.g., early stopping stops at a point where validation error starts to increase. Early stopping criterion typically uses the true loss function (e.g., 0-1 loss)

Stochastic GD & Minibatch

- Gradient descent requires calculating the gradient using all examples in the training set (batch gradient method)

$$-\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}; \boldsymbol{\theta})$$

- Calculation of the above gradient is very computationally demanding
- Instead, we can use a random subset (minibatch) of training examples for each gradient descent step (called stochastic gradient descent (SGD))
- SGD is the most popular optimization algorithm for deep learning
- In practice, perform random shuffling first (to remove any correlation due to ordering), then run SGD by taking samples sequentially
- Lowering the number of examples by 100 (100 fold improvement in computation speed) will result in increase in standard deviation of error from the true mean by only 10

Stochastic GD & Minibatch

- Small batch size can even give a regularization effect (due to noise in the random sampling)
- Small batch size can have a randomization effect during early stage of training if the learning rate is set to a big value initially and gradually decreased (simulated annealing)

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

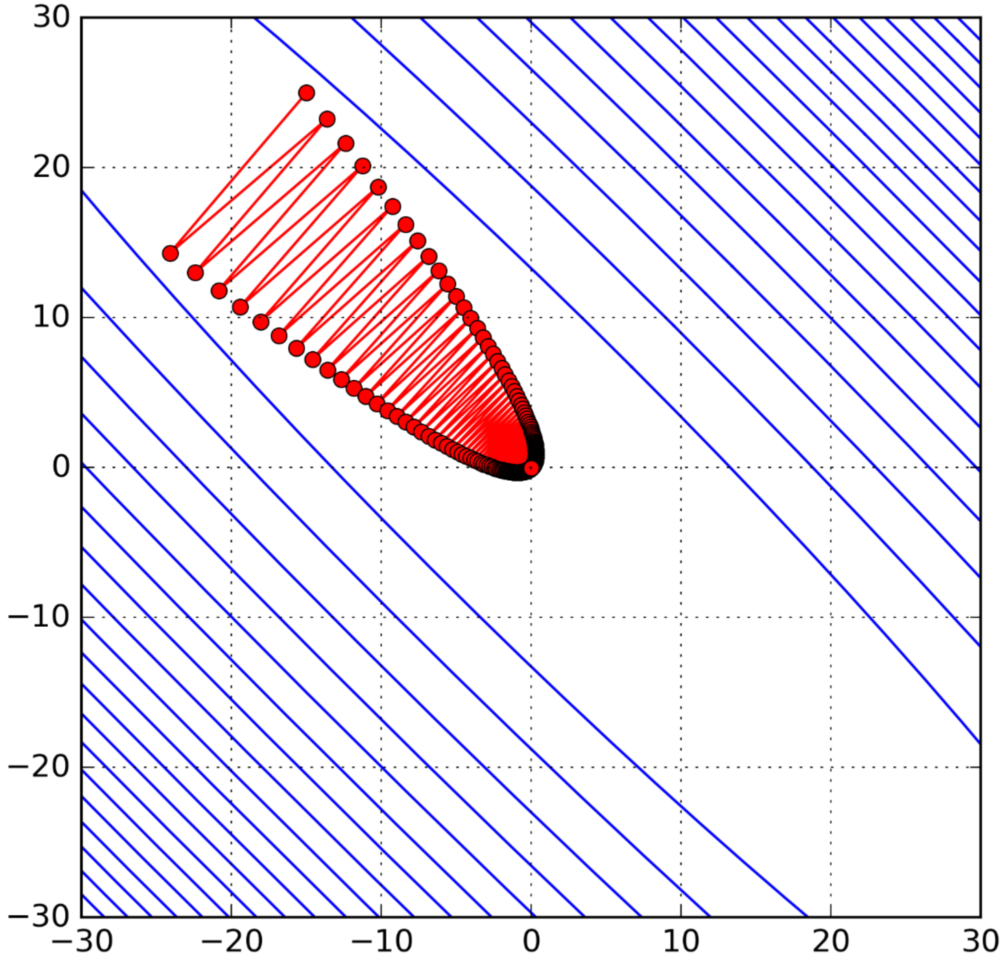
  where $\alpha = k/\tau$

  - ϵ_τ should be small enough due to randomness caused by small minibatch size
  - For $k > \tau$, use $\epsilon_k = \epsilon_\tau$
  - Sufficient condition for convergence to a local minimum: $\sum_{k=1}^{\infty} \epsilon_k = \infty$ & $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$
- Typically minibatch size of about 100 is used
- Typically multiple passes through the training set are done (one such pass is called "epoch")
  - In case of batch gradient descent that uses the entire training set, the entire training set is processed in each iteration

Ill-conditioned Hessian

$$f(x, y) = 50(x + y)^2 + (x - y)^2$$

$$\epsilon = 0.0099$$



Gradient Norm

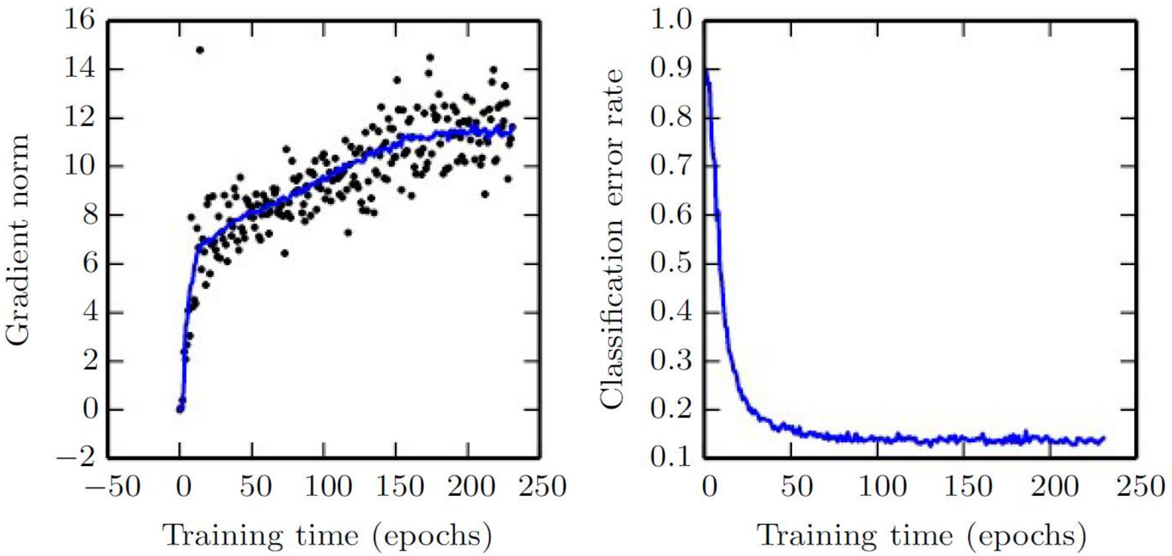
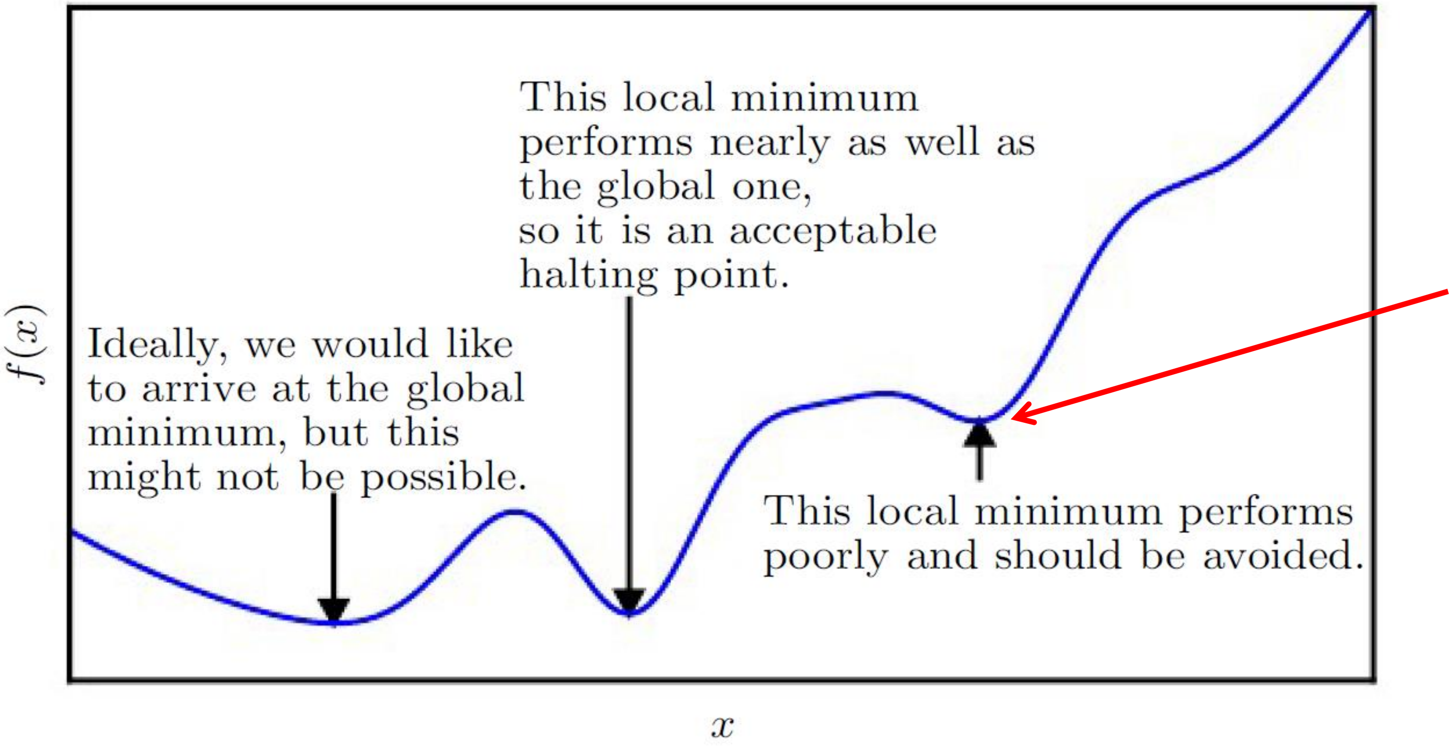


Fig. 8.1

Example showing gradient is still big while the error rate is sufficiently small, which can happen when the Hessian is ill-conditioned

Local Minima

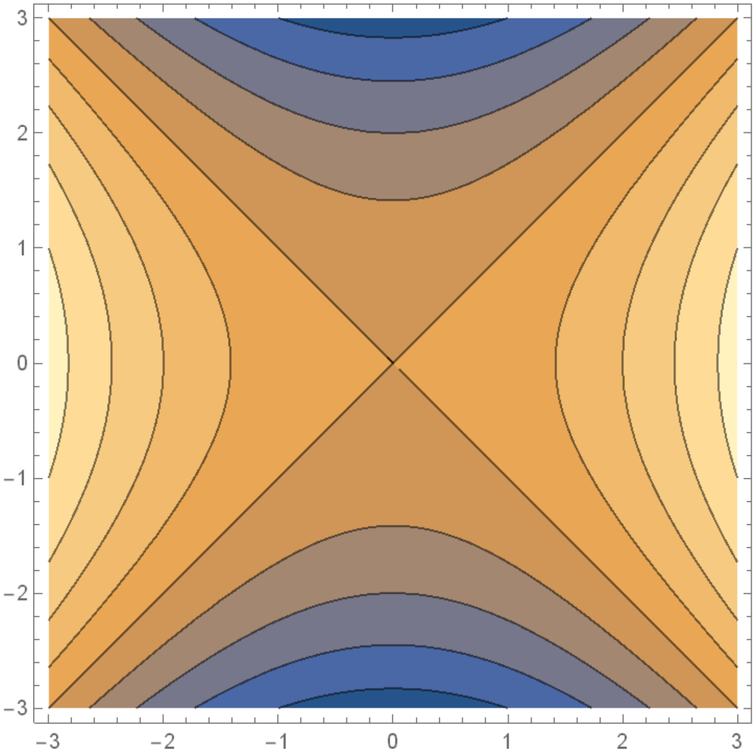


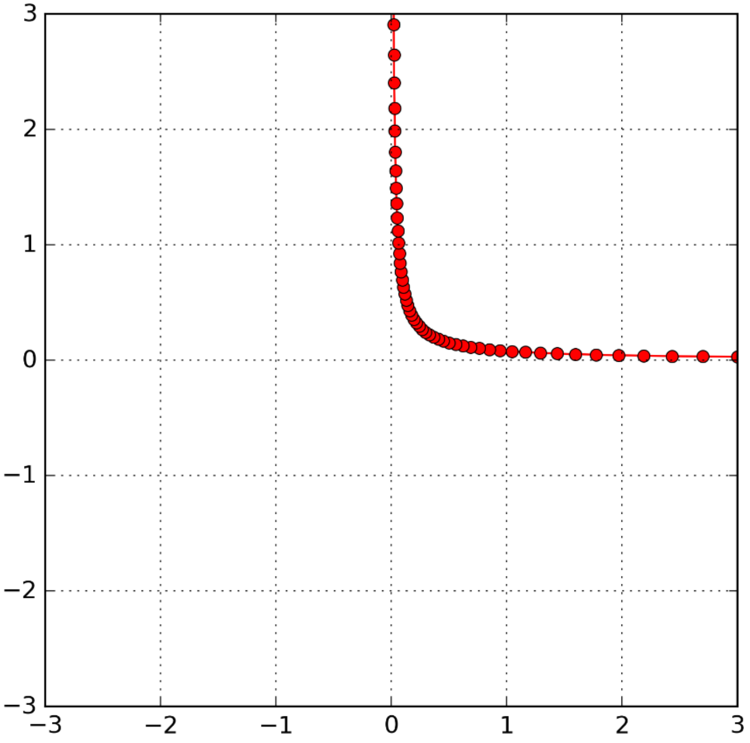
Bad local minima: Believed to be not many in deep learning

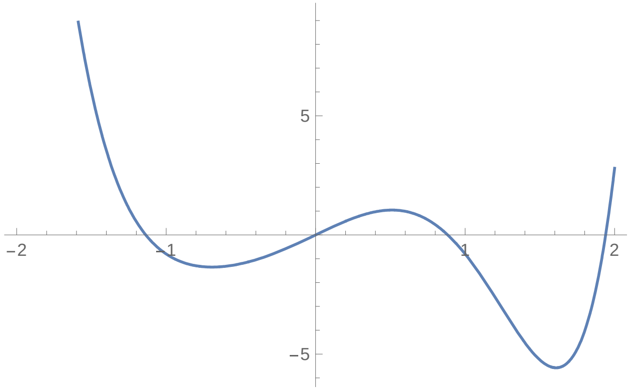
Fig. 4.3

Saddle Points

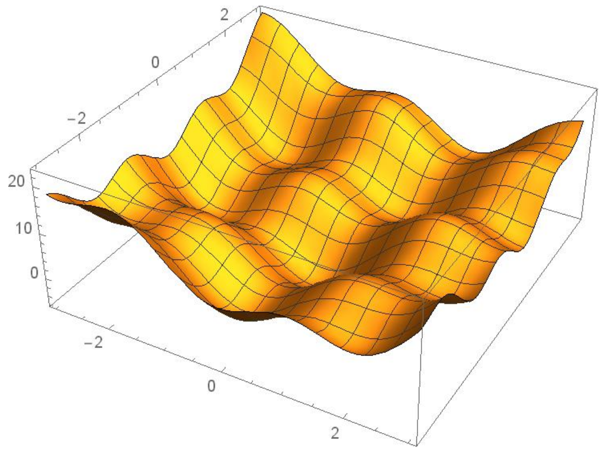
- Gradient descent can be very slow near a saddle point





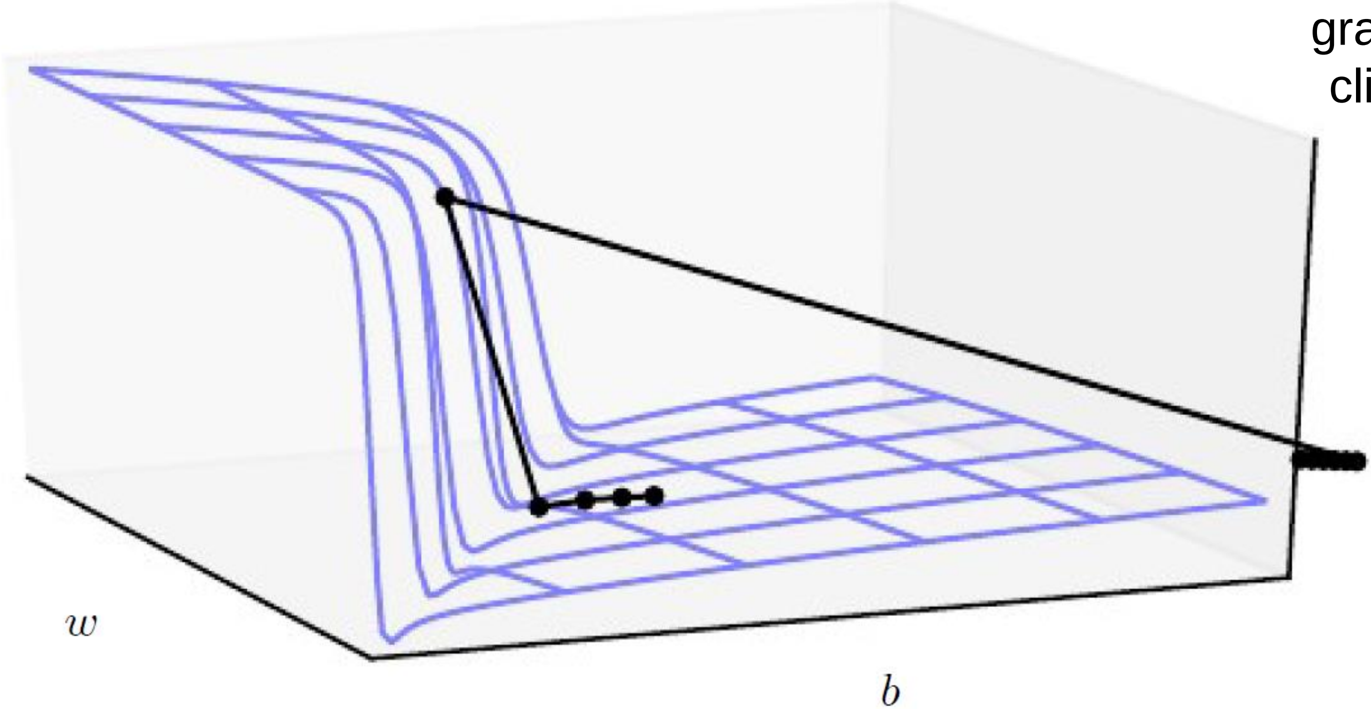


- Low-dimensional optimization
  - Main problems: bad local minima



- High-dimensional optimization
  - Main problems: saddle points
  - Conceptually speaking, there can be many more saddle points than local minima since all eigenvalues of H are positive at a local minimum while they can have mixed signs at a saddle point.

Cliffs



Limit maximum gradient by clipping it

Fig. 8.3

Averaging

- One-tap infinite impulse response (IIR) filter that acts as averaging with exponential (or geometric) profile

$$x[k] = \alpha x[k-1] + (1-\alpha)u[k]$$

- If $u[k] = u$ for all k, then $x[k] = u$ for all k

- If the input is an impulse, i.e., $u[k] = \delta[k] = \begin{cases} 1 & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$, and assume $x[k] = 0$ for $k < 0$, then

$$\begin{aligned} x[-1] &= 0 \\ x[0] &= 1-\alpha \\ x[1] &= (1-\alpha)\alpha \\ &\vdots \\ x[k] &= (1-\alpha)\alpha^k \\ &\vdots \end{aligned}$$

Averaging

- More generally,

$$\begin{aligned}
x[-1] &= (1-\alpha)u[-1] + (1-\alpha)\alpha u[-2] + (1-\alpha)\alpha^2 u[-3] + \ldots \\
x[0] &= (1-\alpha)u[0] + (1-\alpha)\alpha u[-1] + (1-\alpha)\alpha^2 u[-2] + \ldots \\
x[1] &= (1-\alpha)u[1] + (1-\alpha)\alpha u[0] + (1-\alpha)\alpha^2 u[-1] + \ldots \\
&\vdots \\
x[k] &= (1-\alpha)u[k] + (1-\alpha)\alpha u[k-1] + (1-\alpha)\alpha^2 u[k-2] + \ldots \\
&\vdots
\end{aligned}$$

- Therefore, $x[k]$ gives a time-averaged version of $u[k]$ with exponential profile.
- The effective width of the exponential decay defined as $\sum_{k=0}^{\infty} k(1-\alpha)\alpha^k$ is equal to $\frac{\alpha}{1-\alpha}$, which is called the time constant of the one-tap IIR low-pass filter
- If $\alpha = 0.9$, then you are averaging about 10 samples effectively
- If $\alpha = 0.99$, then you are averaging about 100 samples effectively

Momentum

- Progress of SGD can be slow near small gradients
- Introduction of momentum can prevent such slowing down of progress

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$$

- Equivalently,

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

i.e., $\mathbf{v}$ is a time-averaged version of $\frac{-\epsilon}{1-\alpha} \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}) \right)$

- Time-averaging gives momentum
- If gradients are the same all the time, then the learning rate will effectively become $\frac{\epsilon}{1-\alpha}$
- Therefore, it is more natural to do $\mathbf{v} \leftarrow \alpha \mathbf{v} - (1-\alpha) \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}) \right)$ instead

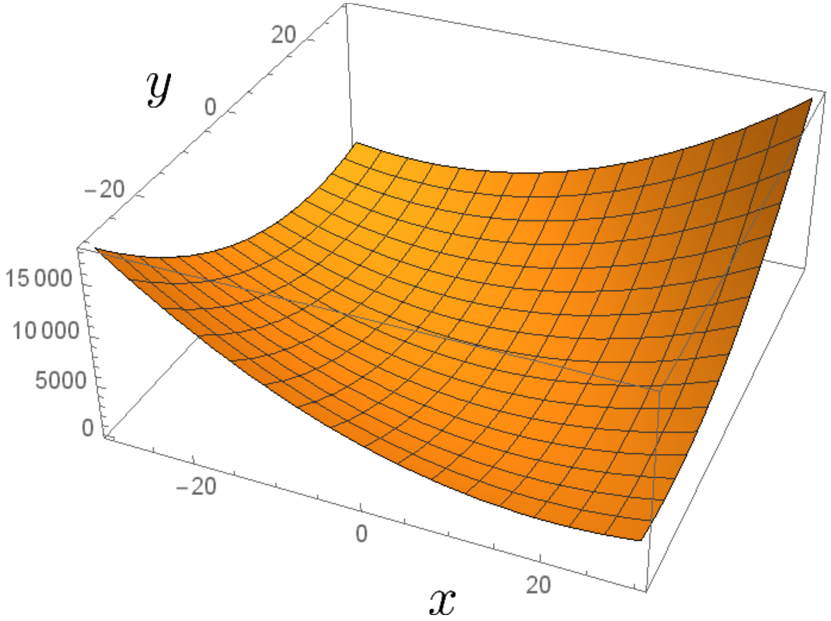
Recap – Gradient Descent

- Gradient descent (or steepest descent): a simple algorithm to find a local minimum
- Go downhill such that the gradient is the steepest, i.e.,

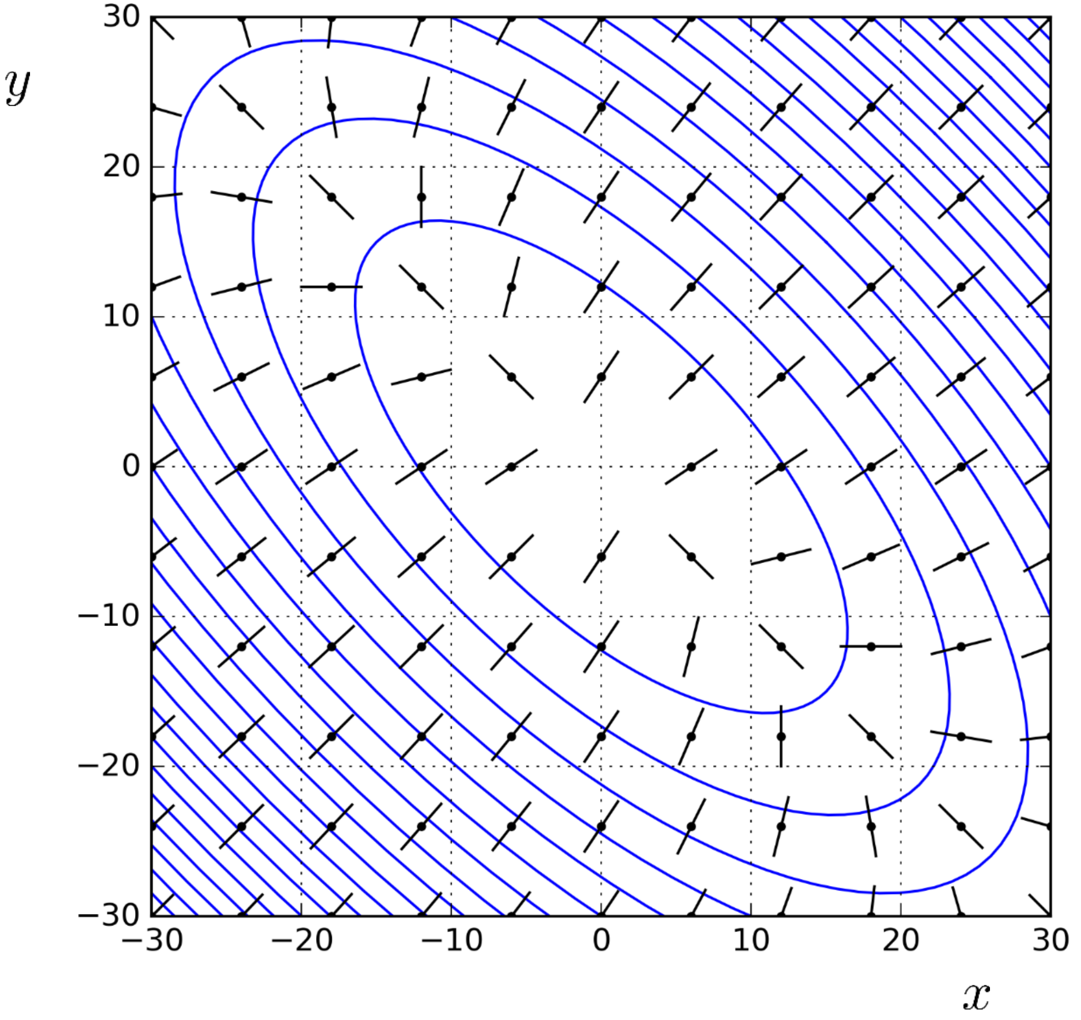
$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla f(\mathbf{x})$$

- ϵ: learning rate

$$f(x, y) = 5(x + y)^2 + (x - y)^2$$

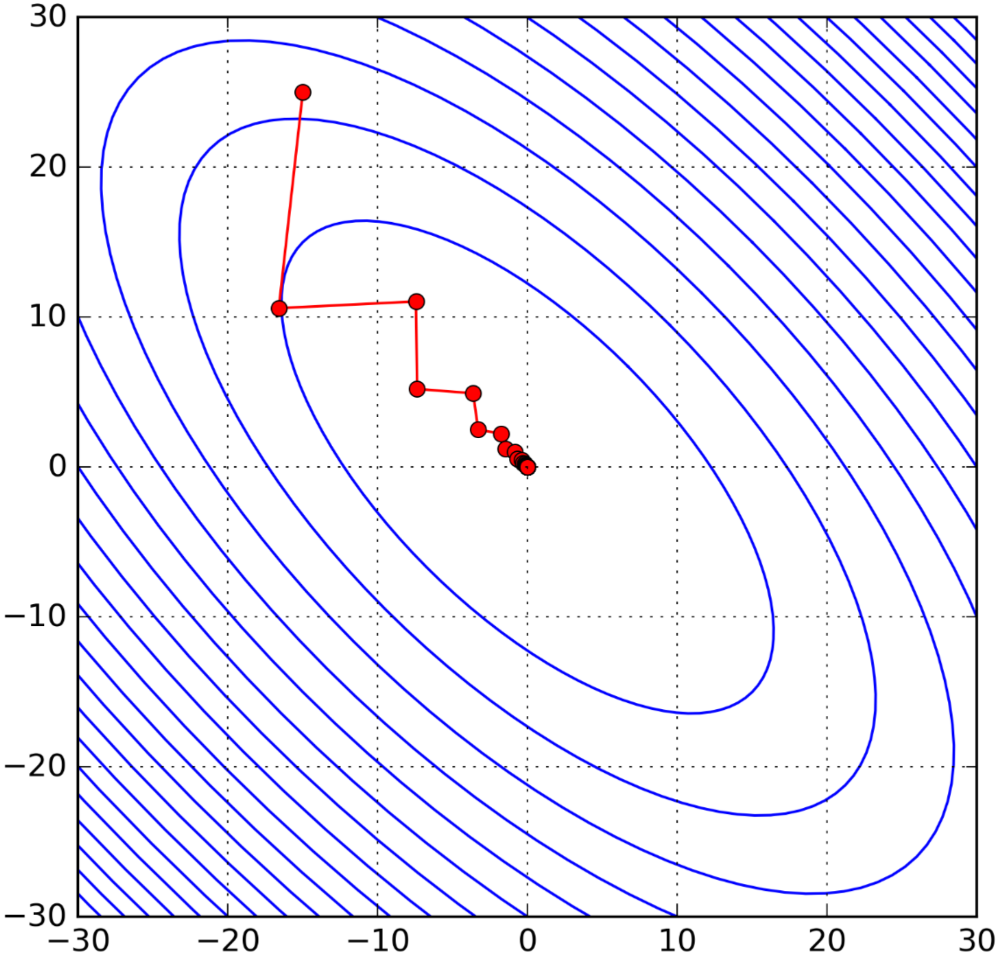


$$\nabla f(x,y) = (10(x+y)+2(x-y), 10(x+y)-2(x-y))^T$$



No Momentum

$\epsilon = 0.08$



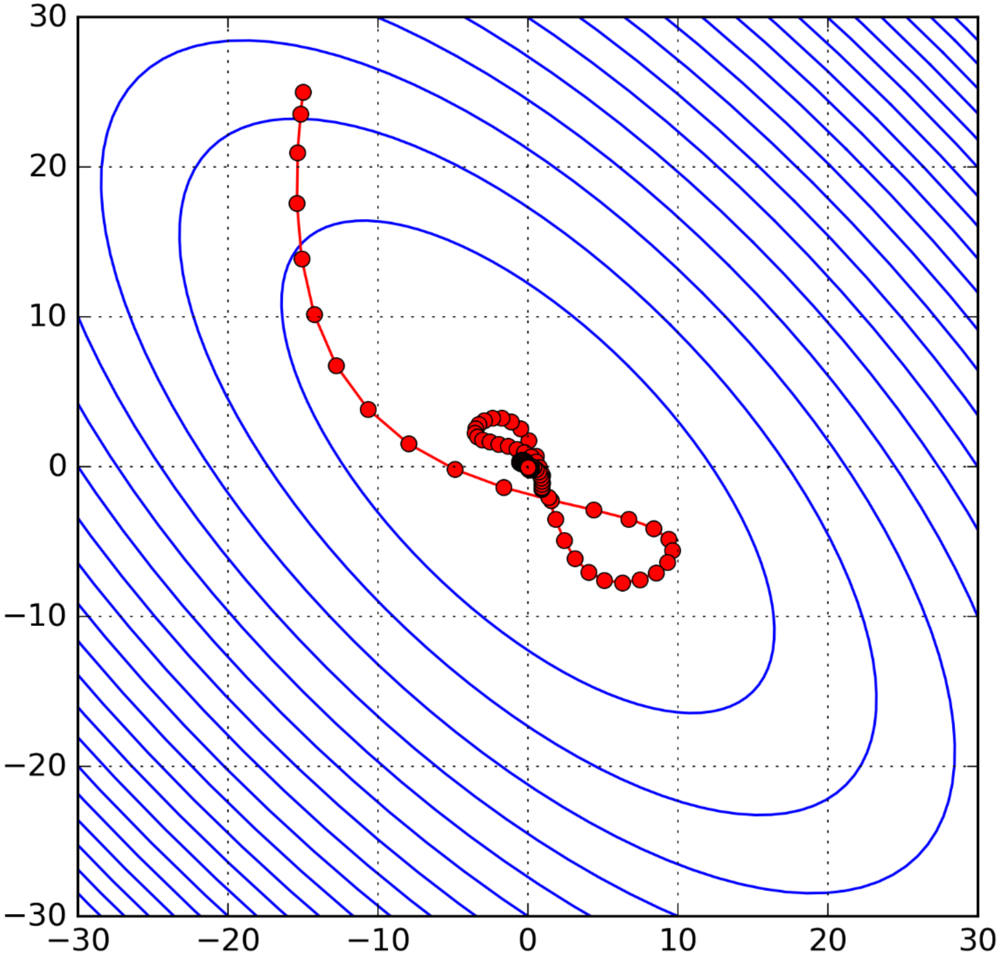
With Momentum

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$$

$\epsilon = 0.08$

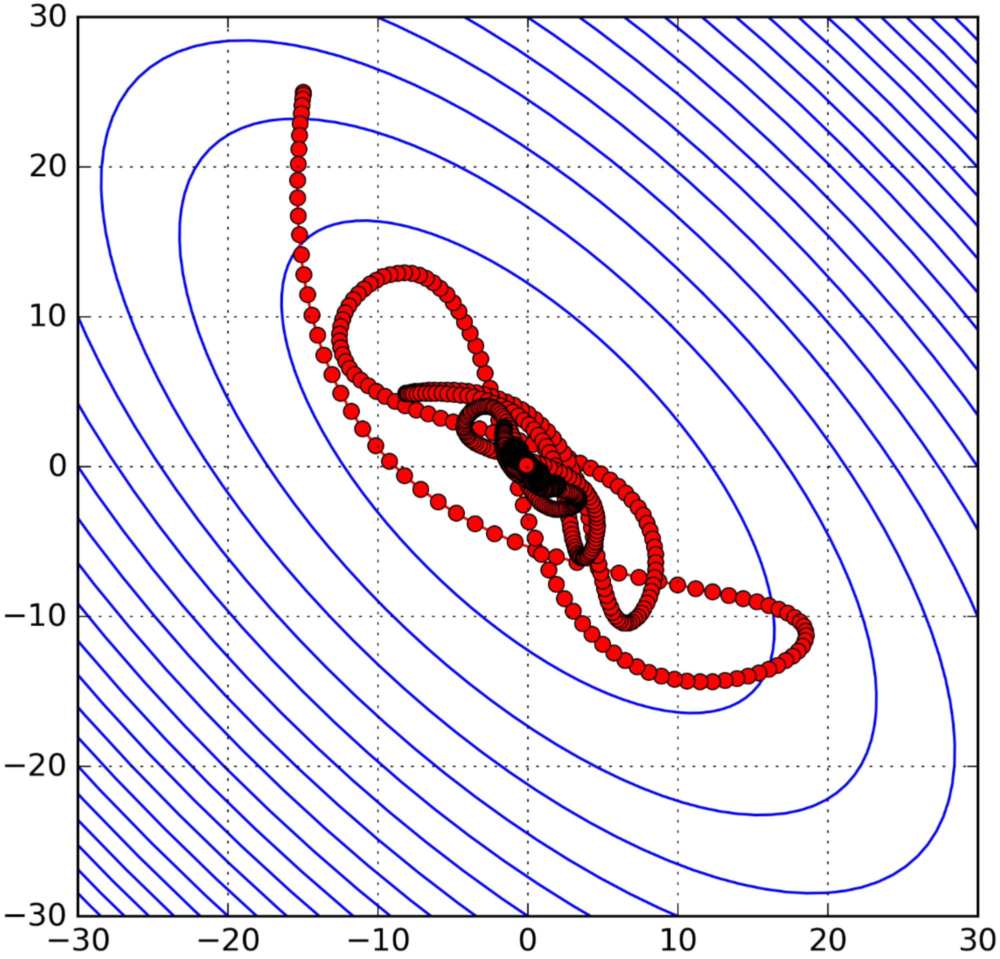
$\alpha = 0.9$



With More Momentum

$$\epsilon = 0.08$$

$$\alpha = 0.99$$

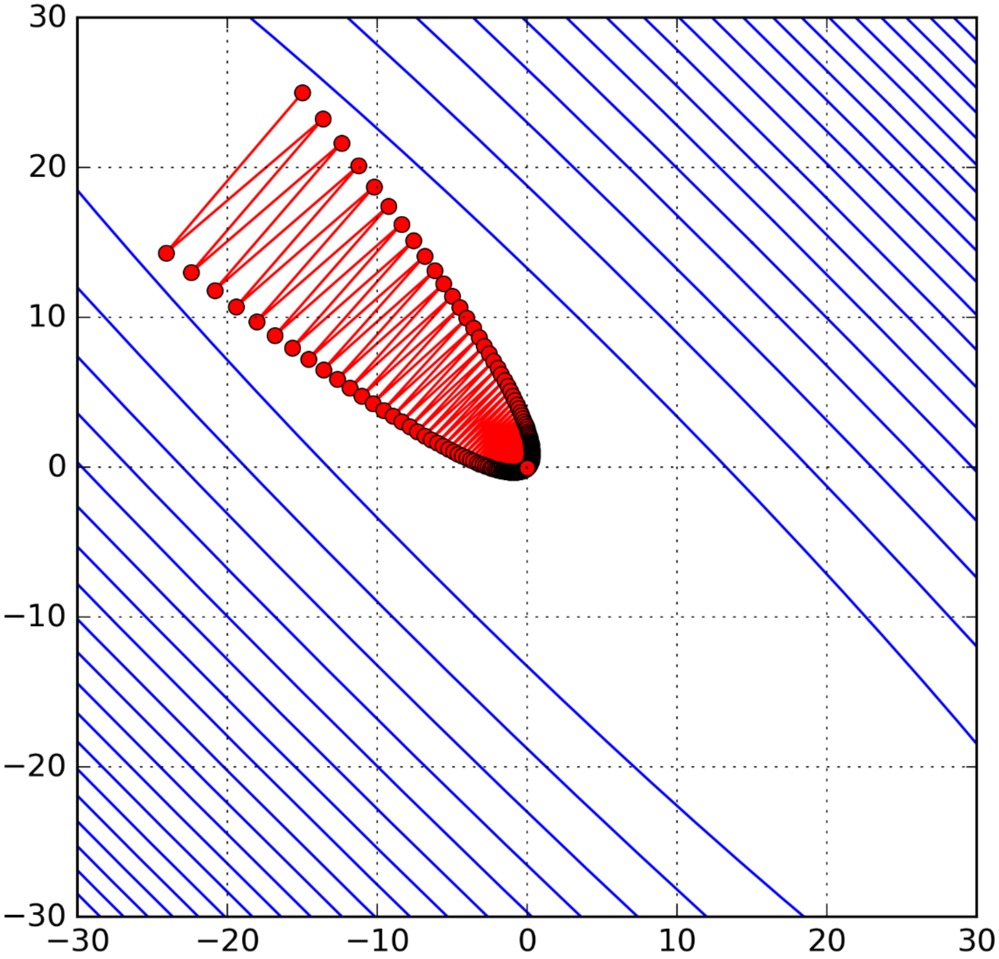


Ill-conditioned Hessian

$$f(x,y) = 50(x+y)^2 + (x-y)^2$$

$$\epsilon = 0.0099$$

$$\alpha = 0 \quad \text{no momentum}$$

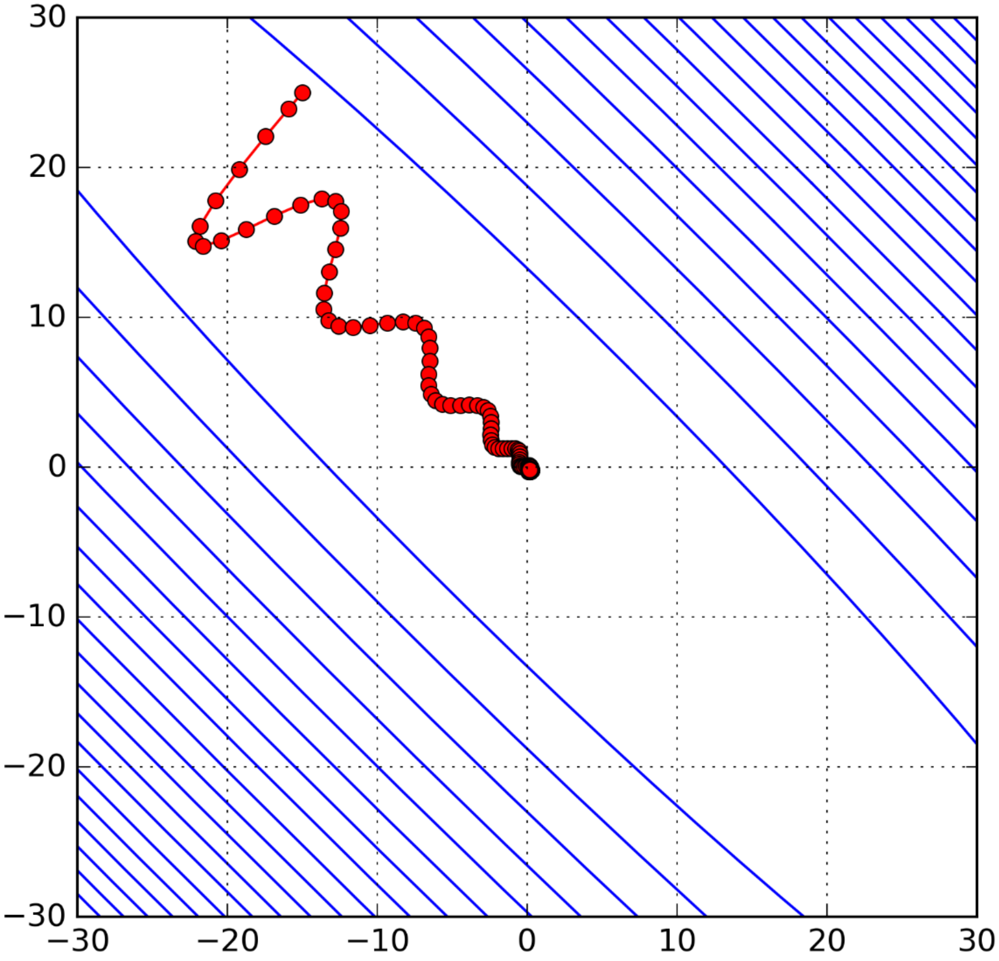


With Momentum

$$f(x,y) = 50(x+y)^2 + (x-y)^2$$

$$\epsilon = 0.0099$$

$$\alpha = 0.9$$



Nesterov Momentum

- Nesterov momentum

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

- Gradient is evaluated with a correction term $\alpha\mathbf{v}$ added
- But, convergence rate is not improved for stochastic gradient descent

Parameter Initialization

- Parameter initialization (weights and biases)
  - Initialization will affect convergence and generalization error.
  - For low-dimensional problems, initialization is more important, e.g., XOR problem, since wrong initialization can result in bad local minimum.
  - For high-dimensional problems, random initialization based on some huristics is often good enough.
  - Why? The number of possible initialization choices grows exponentially with dimension (curse of dimensionality). However, if the performance really differs a lot depending on initialization due to many bad local minima, there's no hope to solve the problem anyway. For example, it's impossible to train a deep feedforward neural network to learn XOR of 100 bits. But, luckly we do not intend to solve such problems using deep learning. Although there's curse of dimensionality, the class of problems that we want to solve using deep learning are simple in a sense, e.g., not many bad local minima. Think of classifying cats and dogs. Such jobs are not as excessively combinatoric as learning XOR of 100 bits.
  - Learning how to do XOR requires a different approach, e.g., learning the concept of XOR first and then applying the XOR rule repeatedly. Such a task can be easily trained and performed by a recurrent neural network. Think about how people learn how to multiply two big numbers.

Parameter Initialization

- Common initialization methods
  - (Truncated) Gaussian and uniform distribution for weights, e.g., for fully-connected layer with m inputs and n outputs

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

  - Small positive value for biases if ReLU is used as activation at a hidden unit so that the function is activated initially and thus giving non-zero gradient.
  - Many other methods and guidelines (read textbook)

Adaptive Learning Rates

- AdaGrad

$$\begin{aligned}
\mathbf{g} &\leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{m} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}),\mathbf{y}^{(i)}) \\
\mathbf{r} &\leftarrow \mathbf{r} + \mathbf{g}\odot\mathbf{g} \\
\Delta\boldsymbol{\theta} &\leftarrow -\frac{\epsilon}{\delta+\sqrt{\mathbf{r}}}\odot\mathbf{g} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}
\end{aligned}$$

- Adaptively adjusts learning rate componentwise by dividing by the square root of the accumulated squared gradients
- Too much accmulation of squared gradients due to accumulation from the beginning
- Not good for deep learning

Adaptive Learning Rates

- RMSProp

$$
\begin{aligned}
\mathbf{g} &\leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}),\mathbf{y}^{(i)}) \\
\mathbf{r} &\leftarrow \rho\mathbf{r} + (1-\rho)\mathbf{g}\odot\mathbf{g} \\
\Delta\boldsymbol{\theta} &\leftarrow -\frac{\epsilon}{\sqrt{\delta+\mathbf{r}}}\odot\mathbf{g} \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}
\end{aligned}
$$

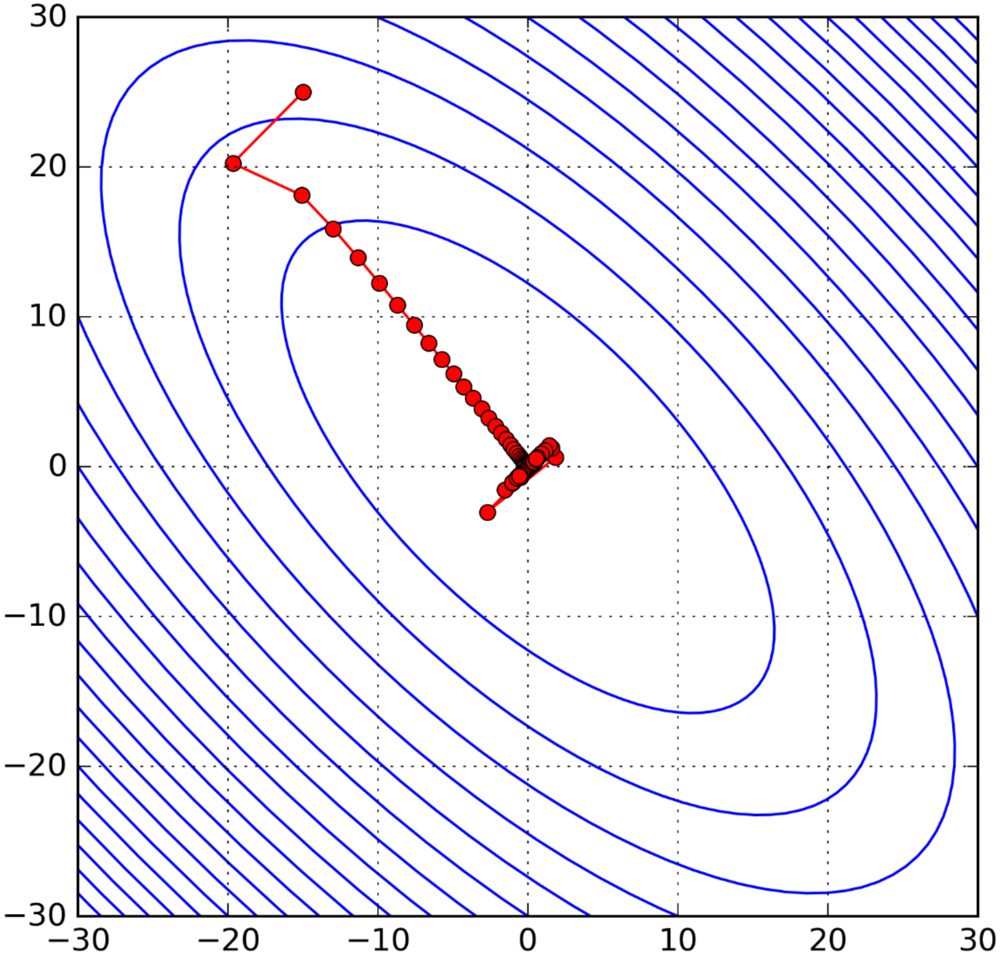
- Geometric averaging of the squared gradients
- Performs componentwise normalization of gradients
- Good choice for deep learning

RMSProp

$\epsilon = 1.5$

$\rho = 0.9$

$\delta = 1$



Adaptive Learning Rates

- Adam (adaptive momentum)

$$\begin{aligned}
\mathbf{g} &\leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{m}L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}),\mathbf{y}^{(i)})\\
\mathbf{s} &\leftarrow \rho_1\mathbf{s}+(1-\rho_1)\mathbf{g}\\
\mathbf{r} &\leftarrow \rho_2\mathbf{r}+(1-\rho_2)\mathbf{g}\odot\mathbf{g}\\
\hat{\mathbf{s}} &\leftarrow \frac{\mathbf{s}}{1-\rho_1^t}\\
\hat{\mathbf{r}} &\leftarrow \frac{\mathbf{r}}{1-\rho_2^t}\\
\Delta\boldsymbol{\theta} &\leftarrow -\epsilon\frac{\hat{\mathbf{s}}}{\sqrt{\delta+\hat{\mathbf{r}}}}\\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta}+\Delta\boldsymbol{\theta}
\end{aligned}$$

Adaptive Learning Rates

- Which algorithm should I use?
  - SGD
  - SGD with momentum
  - RMSProp
  - RMSProp with momentum
  - AdaDelta
  - Adam
- These are all used in practice. Try to experiment with them to see which one performs better for your application.

Batch Normalization

- Choosing and using a single learning rate across all layers can be problematic due to strong interactions across layers
- Vanishing gradient or exploding gradient problem can also occur with many layers
- Batch normalization: normalization at each layer (just before activation function) to coordinate learning across layers

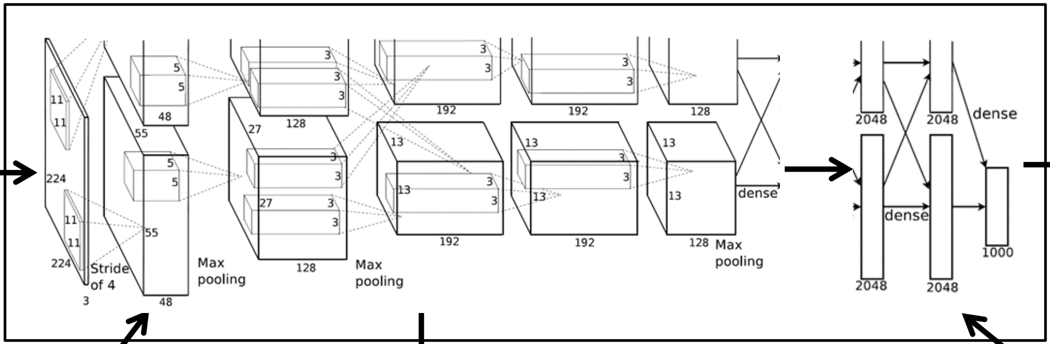
$$\mathbf{H}' = \frac{1}{\boldsymbol{\sigma}}(\mathbf{H} - \boldsymbol{\mu}), \text{ i.e., } H'_{i,j} = (H_{i,j} - \mu_j)/\sigma_j$$

$$\boldsymbol{\mu} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{H}_{i,:} \text{ (mean of each unit)}$$

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_{i=1}^{m}(\mathbf{H} - \boldsymbol{\mu})_i^2} \text{ (standard deviation of each unit)}$$

- Back propagation should account for normalization

Transfer Learning

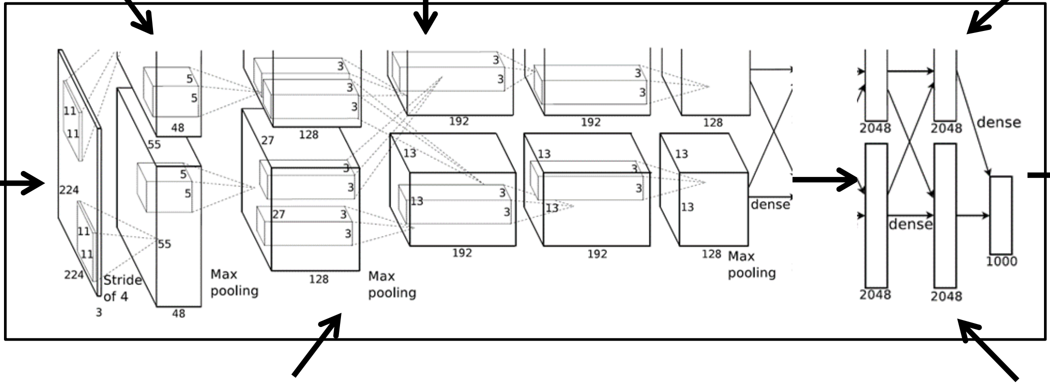


cat

Responsible for detecting features, e.g., edges, triangles, circles, and more complicated shapes

Transfer parameters in lower layers

Responsible for classification



airplane

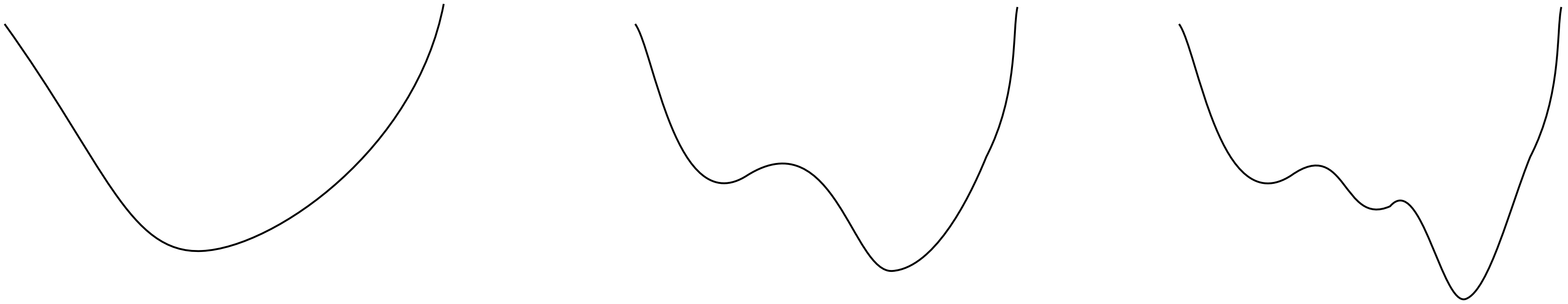
Fewer training examples are OK due to transfer learning

Either fixed or fine tuned for new training examples

Newly trained for new classes

Continuation Method

- Continuation method: Optimize assuming a simpler (more smooth) cost function and gradually move to more complex cost function
- Similar effect as simulated annealing
- Good for avoiding bad local minima. Although local minima are not believed to be a big problem for large-scale deep learning, continuation method can still help.
- Related to curriculum learning - learning simpler concepts first and then more complex concepts based on simpler ones



Assignments, etc.

- Reading assignment: Chapter 9 of DL book