More on Optimization and Adaptive Learning Rate for Neural Networks

Soufiane Belharbi

soufiane.belharbi@insa-rouen.fr

LITIS lab., Apprentissage team - INSA de Rouen, France
ASI 4

March, 29th, 2017

Plan

Results

When you finish this course you will:

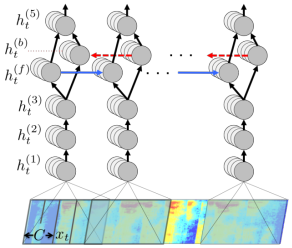
1. Know more about the state of the art of the optimization algorithms for training neural networks.
2. Know more about scheduled and adaptive learning rate.

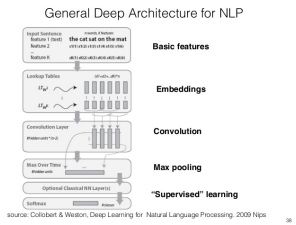
By doing the **Hands on Lab 3**, you will be able to:

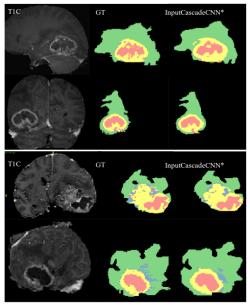
1. Benchmark different training strategies and adaptive learning rate for training neural network using Keras.
2. Implement one of the these strategies to train a neural network using Theano.

Deep learning Today

Deep learning state of the art







What is new today?

- Large data
- Calculation power (GPUs, clouds)

$\Rightarrow$ optimization

- Dropout
- Momentum, AdaDelta, AdaGrad, RMSProp, Adam, Adamax
- Maxout, Local response normalization, local contrast normalization, batch normalization
- RELU
- CNN, RBM, RNN

Empirical Risk & Neural Networks

Define a cost function J over **the training set**:

$$J(\theta) = \mathbb{E}_{(x,y)\sim\hat{p}_{data}} L(f(x;\theta), y)$$

where L is the per-example loss function, $f(x;\theta)$ is the predicted output when the input is x, $\hat{p}_{data}$ is the empirical data.
What we usually prefer: minimize J over **the data generating distribution** p_{data}:

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y)$$

J^* is known by the **risk**.
But, we do not know p_{data}, therefore, minimize **the empirical risk**:

$$\mathbb{E}_{(x,y)\sim\hat{p}_{data}}[L(f(x;\theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)})$$

where m is the number of the training samples. This is known as **the empirical risk minimization**. Does it work for neural networks?

Empirical Risk & Neural Networks

Empirical Risk Minimization (ERM):

$$\mathbb{E}_{(x,y)\sim\hat{p}_{data}}[L(f(x;\theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta),y^{(i)})$$

Does ERM work for neural networks? No.

- Over-fitting.
- Uses generally, 0-1 loss: $L(f(x;\theta),y) = I(f(x;\theta) \neq y)$
- 0-1 has no useful derivative (0 or undefined everywhere) (gradient based-methods will not work)

Solution: use a **surrogate loss function** for example: the negative log-likelihood (able to do ERM and more).
$\rightarrow$ Optimization in general is a little bit different than optimization for neural networks: we seek generalization (early stopping, ...). The optimization can stop while J is still large.

Batch and Minibatch

Decompose J over the samples.
Example: Mixium Likelihood Estimation

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{\text{model}}(x^{(i)}, y^{(i)}; \theta)$$

Maximizing this sum is equivalent to maximizing the expectation over the empirical distribution of the training set:

$$J(\theta) = \mathbb{E}_{x,y \sim p_{data}} \log p_{\text{model}}(x, y; \theta)$$

Most of the optimization methods for neural networks are based on the expectation over the training samples. For example:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,y \sim \hat{p}_{data}} \nabla_\theta \log p_{\textbf{model}}(x, y; \theta)$$

exact computation is very expensive (need to evaluate all the samples in the training set) $\rightarrow$ Compute the expectations by randomly sampling a small number of examples from the dataset by taking the average over only these samples.
Batch, mini-batch, online?

Batch and Minibatch

Batch size factors:

- Large batches provide a more accurate estimate of the gradient.
- Exploit the Multicore architecture using suitable size.
- Limiting factor: batch size and the memory size. (scales when processing is done in parallel)
- Suit the hardware. GPUs better runtime with size of power of 2.
- Small batches can offer regularization effect (add noise). Best with size of 1. But, requires small learning rate (high variation in the gradient). Takes a long time (samll learning rate + size of the data).
- Methods that use only the gradient information $\boldsymbol{g}$ require small batch size whereas second order method that use $\mathbb{H}$ require large batch size.
- The examples within the minibtach must be selected randomly (unbiased estimation).
- Nowadays: datasets size is growing faster than computation power. Issues: under-fitting and computation efficiency. (no enough time to see the whole set, test larger models, ...)

Estimation of the gradient over a minibatch:

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

Challenges in Neural Networks Optimization

- Optimization is generally a difficult task.
- Design carefully the objective function.
- For neural networks training, the cost is non-convex.

Many issues are raised.

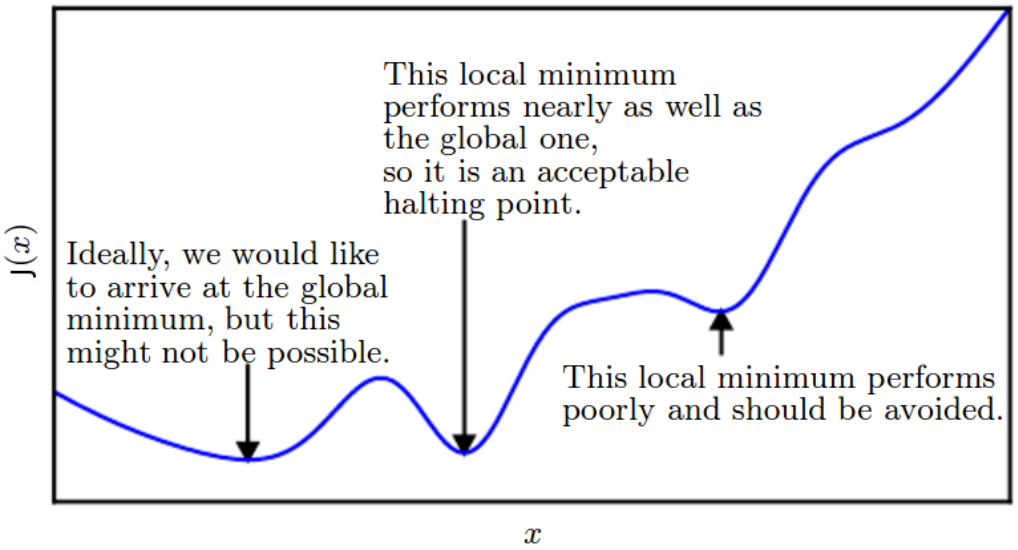
Ill-Conditioning

- A general issue in optimization.
- Ill-conditioning of the Hessian matrix $\mathbb{H}$.
- The SGD get stuck: even very small steps increase the cost function.
- Happens when facing extreme curvature: $\frac{1}{2}\epsilon^2 \boldsymbol{g}^T \mathbb{H} \boldsymbol{g} > \epsilon \boldsymbol{g}^T \boldsymbol{g}$: (gradient shrinking)

$$J(\boldsymbol{\theta} - \epsilon \boldsymbol{g}) \sim J(\boldsymbol{\theta}^{(0)}) - \epsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^T \mathbb{H} \boldsymbol{g}$$

If ϵ is large, the gradient descent step can move uphill. $\Rightarrow$ need very small learning rate $\Rightarrow$ slow learning.

Local Minima



- Convex-function: any local minimum is a a global minimum.
- Non-convex function: many local minima. (up to now, it is not a serious problem)
- Weight space symmetry. (model identifiability, local minima)
- A local minima can be a serious problem when it has a higher cost that the global minima.
- Nowadays' observations: most local minima have low cost values. it is not important to find the true global minimum.

Plateaus, Saddle Points and Flat Regions

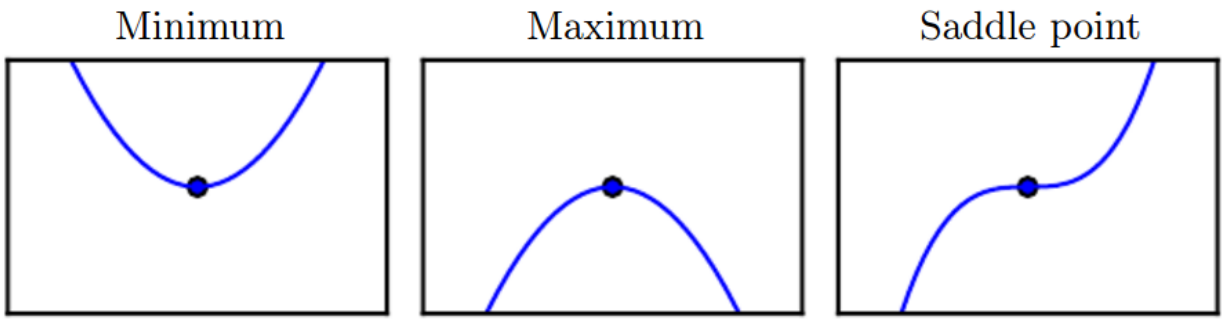


Figure 2: Critical point is a point with zero slope.

- In higher dimensional spaces, local minima are rare and saddle points are common. (eigenvalues of the Hessian $\mathbb{H}$ at saddle points are $+/-$. At a local minima, they are +)
- 2nd order methods struggle with saddle points. (e.g. Newton's method requires adaptation: Saddle-free Newton method)

Plateaus, Saddle Points and Flat Regions

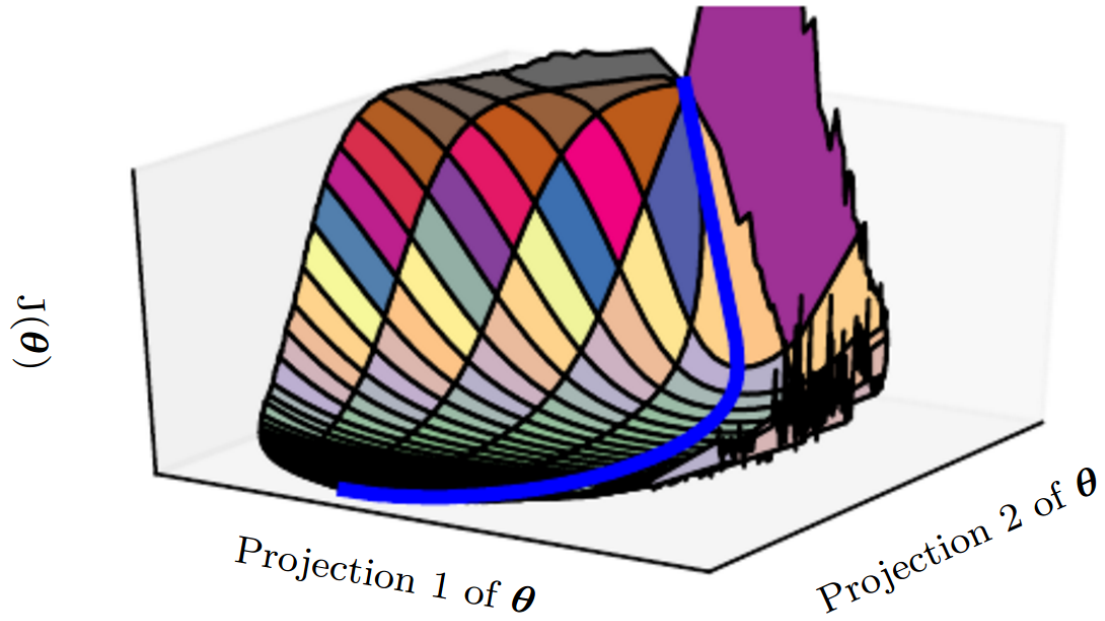


Figure 3: SGD trajectory escaping a saddle point.

- Stochastic Gradient Descent (1st order information) seems to escape saddle points in many cases.
- Flat regions are also an issue. They may be global minima (convex optimization) or high value of the cost (general optimization).

Cliffs and Exploding Gradient

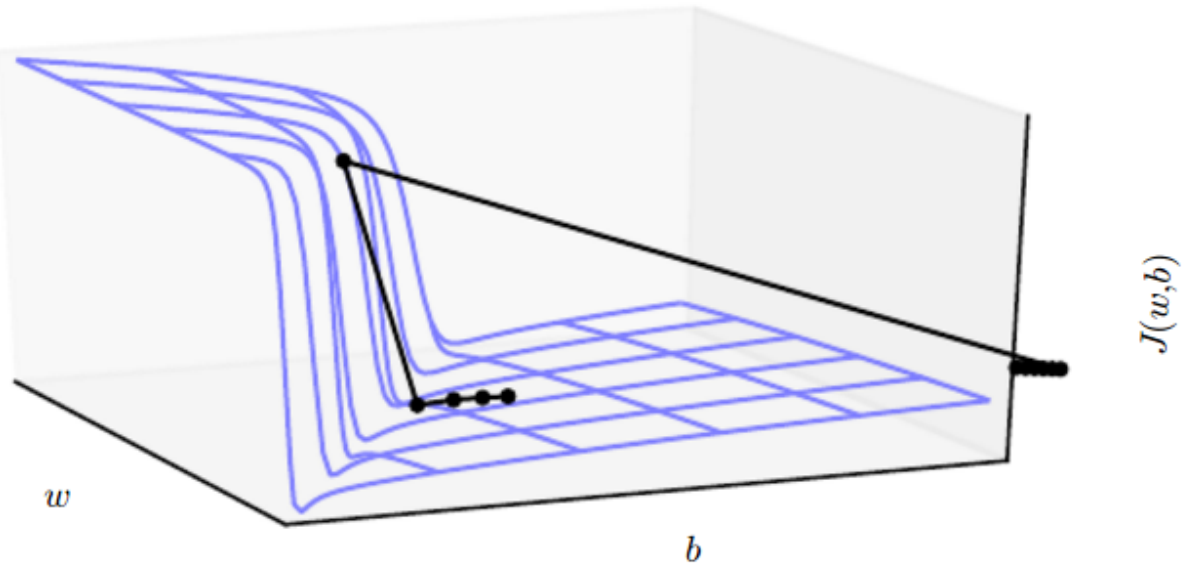


Figure 4: Gradient update step jumping off of the cliff far away.

- Happens often for networks with high non-linearity (deeper networks) and recurrent neural networks.
- Reason: high (magnitude) derivatives as a result from the multiplication of several parameters$\Rightarrow$ solution: **gradient clipping** (heuristic)

Cliffs and Exploding Gradient

Gradient clipping: the **gradient** does not indicate the optimal step size, but only the optimal **direction**.

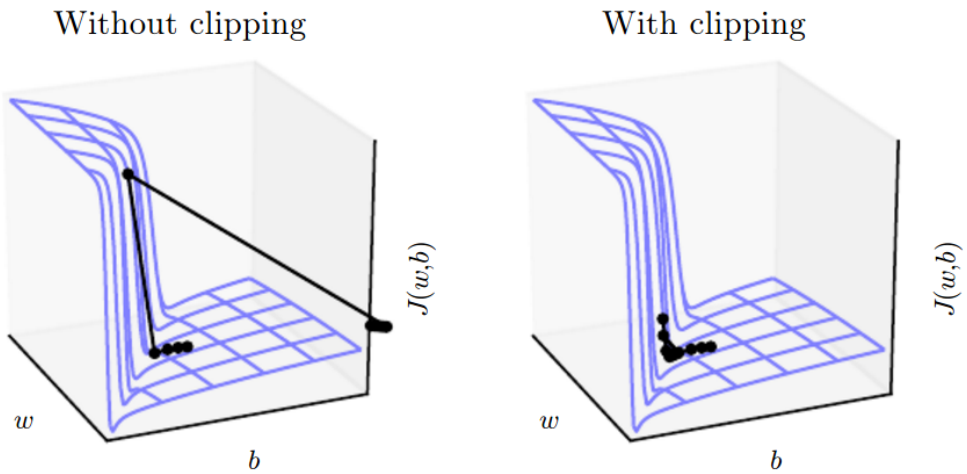


Figure 5: Effect of clipping the gradient.

Heuristics:

- Clip the gradient values: Element-wise clipping.
- Clip the norm $||\boldsymbol{g}||$ of the gradient $\boldsymbol{g}$:

$$\text{if } ||\boldsymbol{g}|| > v:$$
$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}v}{||\boldsymbol{g}||} \quad v \text{ is the norm threshold.}$$

Long-Term Dependencies

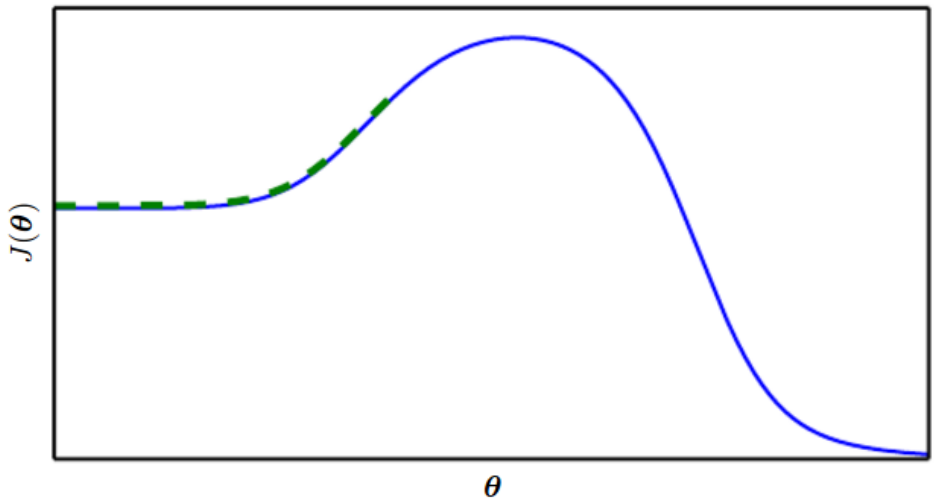
- Extremely deep computational graphs: deeper networks, recurrent networks. (high non-linearity)
- In a graph with parameter **w** multiplied t times $\Rightarrow$ the gradient will scale according to $diag(\lambda)^t$, for λ is the eigenvalues of **w**.
- Leads to: vanishing gradient when $|\lambda| < 1$ and exploding gradient when $|\lambda| > 1$

Inexat Gradient

Most optimization algorithms are based on the assumption that we have access to the exact gradient. But:

1. Objective function is intractable
2. $\Rightarrow$ Intractable gradient
3. $\Rightarrow$ Approximation of the gradient
4. $\Rightarrow$ Noisy gradient.

Poor Correspondence: Local and Global Structure



In order to minimize $J(\theta)$:

- Most methods use J properties at a single point θ
- $\Rightarrow$ It is difficult to make a decision step if $J(\theta)$ is poorly conditioned at θ or if θ lies on a cliff, or it is a saddle point in order to progress downhill.
- Even if we overcome these issues, we still perform poorly if the direction that results locally does not point toward distant regions of mush lower cost.
- Researchers argue that much of the runtime of training is due to the length of the trajectory needed to arrive at the solution.
- Find a better initial points?

Conclusion

- Training neural networks is difficult.
- Optimizing a neural network does not necessarily seek to find the minimum of the loss function but to obtain a good **generalization**.

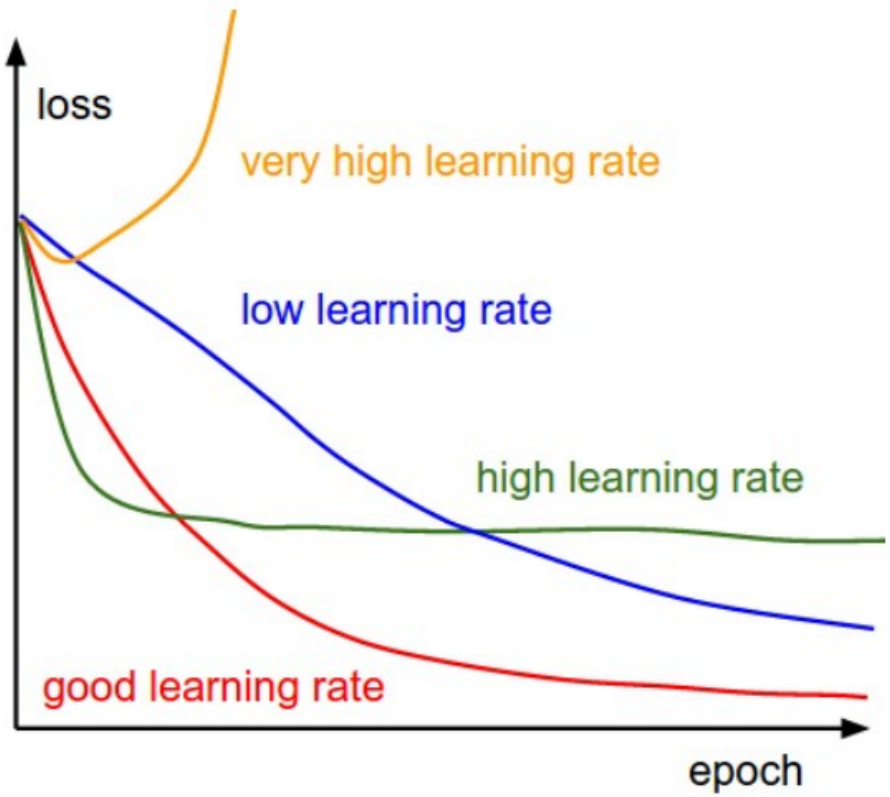
Stochastic Gradient Descent

Algorithm 1 Stochastic Gradient Descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter θ
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
    Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
    Apply update: $\theta \leftarrow \theta - \epsilon_k \hat{\boldsymbol{g}}$
  end while

- ϵ_k is very important. It is necessary to gradually decease it over time.
- How to update ϵ_k? heuristics.
  - Step decay: reduce ϵ by some factor every few epochs.
  - Annealed: $\epsilon_k = \frac{\epsilon_0}{(1+kt)}$. t: decay factor.
  - Exponential decay: $\epsilon_k = \epsilon_0 e^{-kt}$. t: decay factor.
  - Linear decay until iteration τ: $\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$. $\alpha = \frac{k}{\tau}$. After τ, it is common to leave ϵ constant.
- How to set ϵ_0? trial-error. (check the first iterations)

Stochastic Gradient Descent



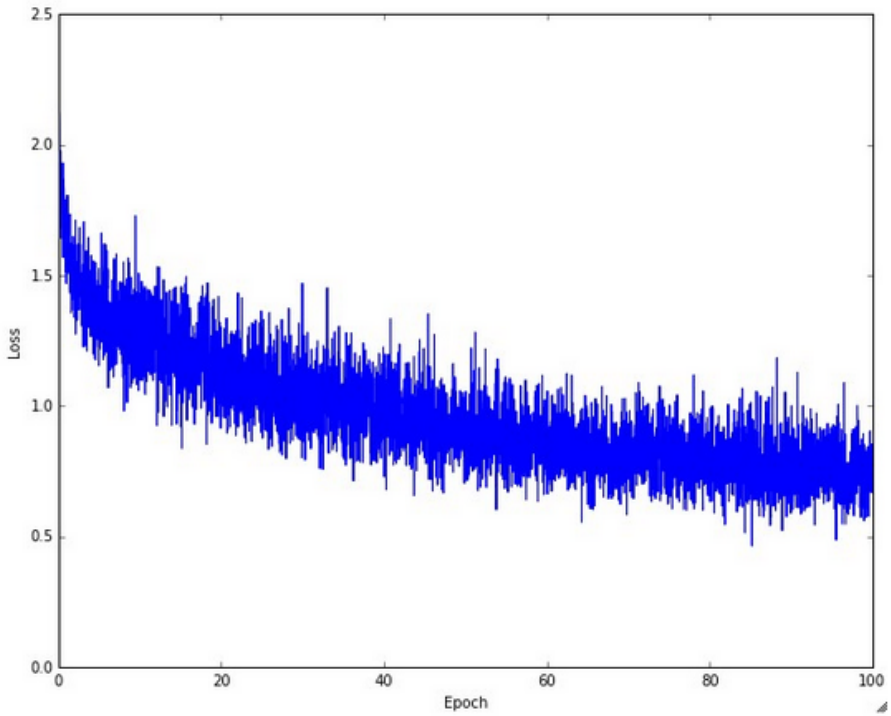


Figure 6: **Left**: Effect of the learning rate over the loss. **Right**: Typical loss curve (noisy).

Momentum

- SGD is slow.
- Momentum designed to accelerate learning. Confront: high curvature, noisy gradient, small but consistent gradient.
- It accumulates an exponentially decaying moving average of the past gradients and continues to move in their direction.

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

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

v is the velocity (direction and speed) which accumulates the gradient. $\alpha \in [0, 1]$ hyper-parameter to indicate how much to consider the past (accumulated gradients). The larger α is relative to ϵ, the more the previous gradients affects the current direction.
$\rightarrow$ Imagine a blind ball rolling downhill. It gains speed as long as it is going down.

Momentum

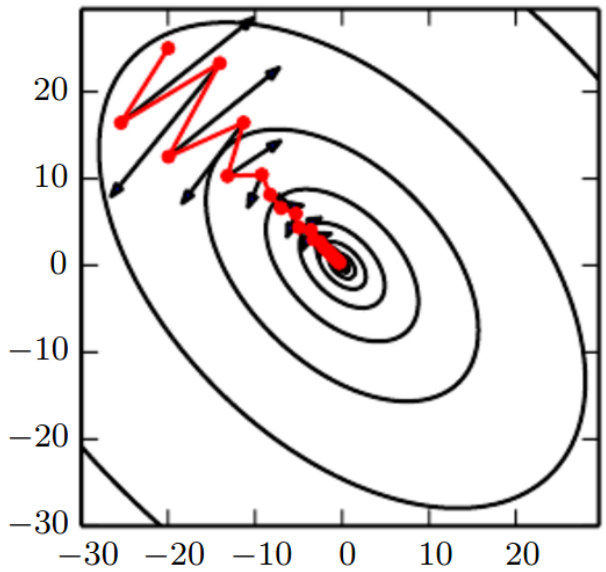


Figure 7: Black: SGD (waisting time. oscillating). Red: momentum (reducing the noisy gradient).

→ **The accumulation effect**: the step size is increase when many successive gradients points into similar directions (i.e. go faster). It reduces (i.e. slow down) when they change directions.
→ Common values of α: 0.5, 0.9, 0.99. α may be adapted over time (it begins with small values, and it is later raised).
→ It is more important to shrink ϵ over time that adapt α.

Momentum

Algorithm 2 Stochastic Gradient Descent (SGD) with momentum

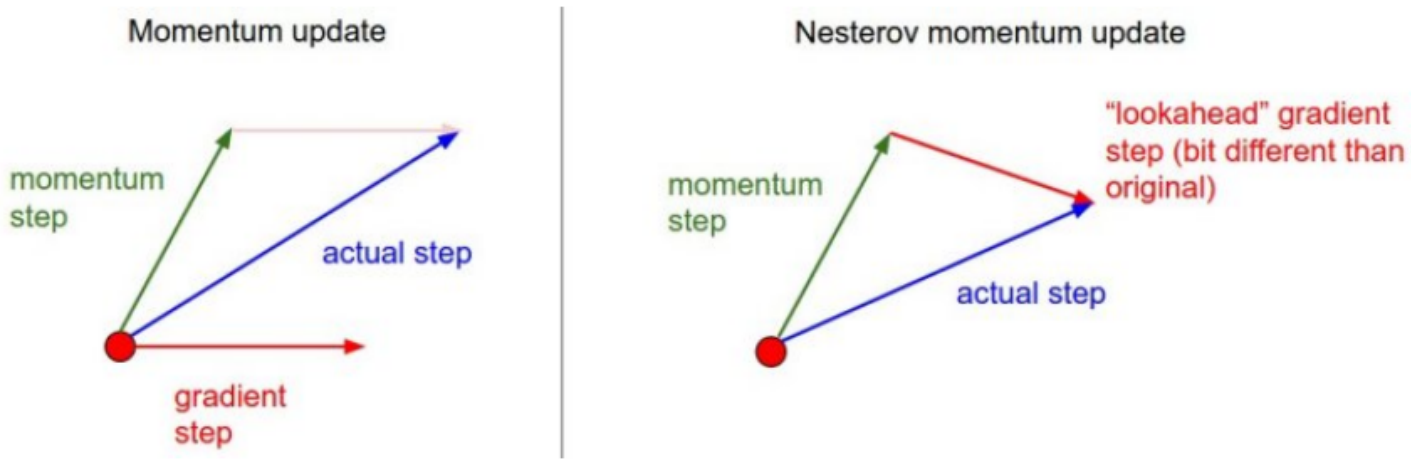
Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter θ, initial velocity v.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
    Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
    Compute velocity update: $v \leftarrow \alpha v - \epsilon \hat{\boldsymbol{g}}$
    Apply update: $\theta \leftarrow \theta + v$
  end while

Nesterov Momentum

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

$$\theta \leftarrow \theta + v$$

The difference between Nesterov momentum and standard momentum is the position where the gradient is evaluated.
$\rightarrow$ Imagine a ball, that has notion where it is going, rolling downhill. It can slow down **before** the hill slopes up because it knows what is going to happen (look a head).
$\rightarrow$ Make the error then correct it. (correction factor)



Nesterov Momentum

Algorithm 3 Stochastic Gradient Descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter θ, initial velocity v.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$
Compute gradient (at interim point): $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}; \tilde{\theta}), y^{(i)})$
Compute velocity update: $v \leftarrow \alpha v - \epsilon \hat{\boldsymbol{g}}$
Apply update: $\theta \leftarrow \theta + v$
end while

Adaptive Learning Rate

Instead of using a single learning rate for all the parameters, we adapt a learning rate for each local parameter.

- AdaGrad.
- RMSProp.
- AdaDelta.
- Adam.
- Adamx.

AdaGrad

- Adapts individually the learning rates of all the model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values.
- Parameters with the largest partial derivative have a rapid decrease in their learning rate.
- Parameters with small partial derivatives have a small decrease in their learning rate.
- $\rightarrow$ The model has a greater progress in gently sloped directions.

Issues:
$\rightarrow$ Empirically, in training neural networks, the accumulation of squared gradient from the beginning of training can result in premature and excessive decrease in the effective learning rate. why?
$\rightarrow$ AdaGrad preforms well for some but not all deep learning models.

AdaGrad

Algorithm 4 The AdaGrad algorithm

Require: Global learning rate ϵ.
Require: Initial parameter $\boldsymbol{\theta}$.
Require: Small constant δ, e.g. 10^{-7}, for numerical stability. Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(x^{(i)}; \boldsymbol{\theta}), y^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$ (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

RMSProp

- RMSProp modifies AdaGrad.
- AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving close to a convex structure.
- RMSProp uses an exponentially decaying (weighted) average to discard history from the extreme past so that it can converge rapidly after finding a convex structure.
- The use of the moving average introduces a new hyperparameter, ρ that controls the length scale of the moving average.

RMSProp

Algorithm 5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$.
Require: Small constant δ, e.g. 10^{-6}, to stabilize division by small numbers.
  Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
    Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(x^{(i)}; \boldsymbol{\theta}), y^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$ ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

RMSProp

Algorithm 6 The RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity v.
Require: Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha v$
    Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(x^{(i)}; \tilde{\boldsymbol{\theta}}), y^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
    Compute velocity update: $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$ ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + v$
  end while

Empirically, RMSProp has shown to be effective and practical for optimization algorithms for neural networks.

AdaDelta

- An extension to AdaGrad (close to RMSProp).
- Uses a moving weighted average over the squared gradient.
- Eliminate the need to a global learning rate (ϵ) but introduces a second element: another moving average over the squared parameter updates.

AdaDelta

Algorithm 7 The AdaDelta algorithm

Require: Decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$.
Require: Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$, $\boldsymbol{u} = \mathbf{0}$.
Require: Small constant δ use for numerical stabilization.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(x^{(i)}; \boldsymbol{\theta}), y^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\sqrt{\boldsymbol{u}}}{\sqrt{\boldsymbol{r}}} \hat{g}$
Accumulate update: $\boldsymbol{u} \leftarrow \rho \boldsymbol{u} + (1 - \rho) \Delta\boldsymbol{\theta} \odot \Delta\boldsymbol{\theta}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Adam

- Adaptive moments. (1st and 2nd)
- Variation of a combination of RMSProp and Momentum.
- Incorporates an estimate of the 1st order of the gradient with exponential weighting (similar to momentum). [mean]
- Incorporates an estimate of the 2nd order of the gradient with exponential weighting (similar to RMSProp) [uncentered variance]
- Both estimation are biased. Need correction.

Adam

Algorithm 8 The Adam algorithm

Require: Step size ϵ (suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1[$ (suggested values: 0.9 and 0.999, respectively)
Require: Small constant δ use for numerical stabilization. (Suggested default: 10^{-8}
Require: Initial parameter θ. Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$. Initialize time step $t = 1$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
  Compute gradient: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(x^{(i)}; \boldsymbol{\theta}), y^{(i)})$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\hat{\boldsymbol{g}}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
  Compute velocity update: $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \hat{\boldsymbol{g}}$ ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
  Compute update: $\Delta\boldsymbol{\theta} = -\epsilon\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operattion applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Empirically, Adam is seen to be robust to the choice of the hyperparameters. The learning rate needs sometimes to be changed.

Which One to Choose?

Unfortunately for you, there is no consensus.
$\rightarrow$ Actively used algorithms: SGD, SGD with momentum, RMSProp, RMSProp with momentum, AdaDelta and Adam.
$\rightarrow$ You are on your own. You need to be more familiar with the algorithms (the art of the hyperparameters tunning)

Second Order Methods

But, what about second order methods?

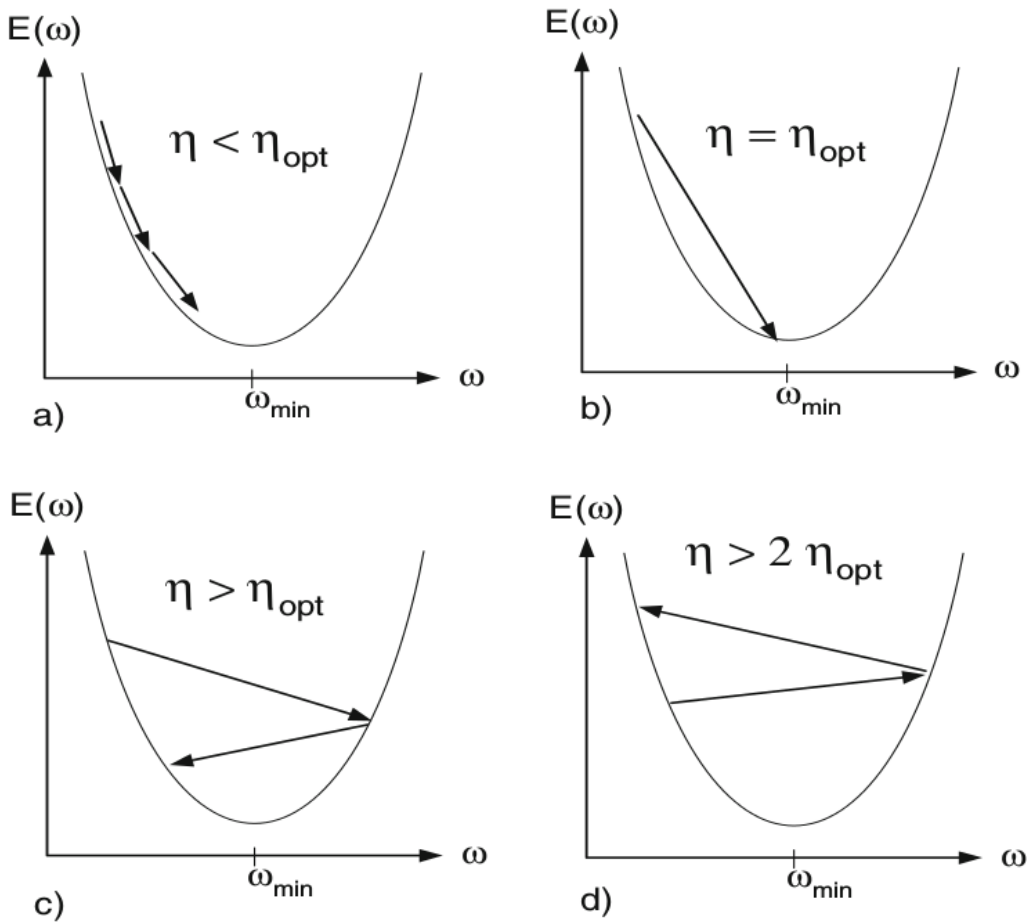
$$J(\theta) = \mathbb{E}_{(x,y)\sim\hat{p}_{data}(x,y)}[L(f(x;\theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta),y^{(i)})$$

$$J(\theta) \simeq J(\theta_0) + (\theta-\theta_0)^T\nabla_\theta J(\theta_0) + \frac{1}{2}(\theta-\theta_0)^T\mathbb{H}(\theta-\theta_0)$$

$$\theta^* = \theta_0 - \mathbb{H}^{-1}\nabla_\theta J(\theta_0)$$

The short story: $\rightarrow$ Impractical.

Conclusion



Keep in Mind: The learning rate is a very important factor in optimization.

Questions

Thank you for your attention,

Questions?

Is that all?
Yes.
What now?
Hands on Lab 3.

Hands on Lab 3

Break for 15 minutes.

. . . .

1. Benchmark different training strategies and adaptive learning rate for training neural network using Keras.
2. Implement one of the these strategies to train a neural network using Theano.