Optimization in Machine Learning

Lecture 18: Hybrid, ADAM, Adagrad, RMSProp toward Proximal and Generalized Gradient Descent, Constrained Optimization

Ganesh Ramakrishnan

Department of Computer Science
Dept of CSE, IIT Bombay
https://www.cse.iitb.ac.in/~ganesh

March, 2025

Outline of next few topics

- Algorithms for Optimization: First Order and thereafter [Done]
- Accelerated Gradient Descent [Done]
- Stochastic Gradient Descent [Done]
- Accelerated Stochastic [Done]
- Accelerated Stochastic Variants: Hybrid, Adam, Adagrad, RMSProp, *etc.*
- Generalized/Proximal Gradient Descent
- Constrained Optimization and Projected Gradient Descent

SGD and Momentum [RECAP]

- Recall SGD:

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t)$$

- Stochastic Momentum: Improves upon SGD via Momentum (similar to the GD case):

$$w_{t+1} = w_t - \alpha_t \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t(w_t - w_{t-1})$$

- Heavy Ball (HB) Momentum: $\gamma_t = 0$
- Nesterov's Accelerated Gradient (NAG): $\gamma_t = \beta_t$.

Issues with SGD/Momentum [RECAP]

- These techniques are not adaptive: Require extensive tuning for learning rate schedules.
- Without the right LR schedule, convergence can be slow!
- They are also less robust to initialization
- Fix: Adapt learning rate based on gradient information until now.

A SLIGHT DETOUR: LET US UNDERSTAND THE BASIC PREMISE OF SECOND ORDER METHODS (SUCH AS THE NEWTON AND QUASI NEWTON METHODS) THAT INVOKE THE HESSIAN AND ITS APPROXIMATIONS

https://moodle.iitb.ac.in/pluginfile.php/143843/mod_resource/content/1/Basics%20of%20Convex%20Optimization%20My%20Notes.pdf

THIS WILL LEAD TO BETTER APPRECIATION OF MATRICES SUCH AS H TO FOLLOW

Adaptive Gradient Descent Framework of Algorithms

- Adaptive Methods try to automatically adapt the learning rate.
- Let H_t be a positive semi-definite Matrix[1] at the t^{th} iteration
  - The simplest definition of H_t is a diagonal matrix (recall we need SGD algorithms to be super-fast and therefore avoid matrix inversion). Eg: If $g_t = \nabla f(w_t)$ is gradient at t^{th} iteration,

$$H_T = \text{diag}\left(\left[\sum_{t=1}^{T} \eta_t g_t g_t^T\right]^{1/2}\right) \implies \left[H_T^{-1}\right]_{jj} = \frac{1}{\sqrt{\sum_{i=1}^{T} \eta_t g_{tj}^2}}$$

- The simplest Adaptive Algorithm called AdaGrad (Duchi et al 2011) is:

$$w_{t+1} = w_t - \alpha_t H_t^{-1} \nabla f(w_t)$$

[1] Reminiscent of the Hessian

Adaptive Gradient Descent Framework of Algorithms

Motivation from earlier discussion: Adaptive learning motivated by the curvature coming from the Hessian or its surrogates (approximations)

- Adaptive Methods try to automatically adapt the learning rate.
- Let H_t be a positive semi-definite Matrix[1] at the t^{th} iteration
  - The simplest definition of H_t is a diagonal matrix (recall we need SGD algorithms to be super-fast and therefore avoid matrix inversion). Eg: If $g_t = \nabla f(w_t)$ is gradient at t^{th} iteration,

$$H_T = \operatorname{diag}\left(\left[\sum_{t=1}^{T} \eta_t g_t g_t^T\right]^{1/2}\right) \implies \left[H_T^{-1}\right]_{jj} = \frac{1}{\sqrt{\sum_{t=1}^{T} \eta_t g_{tj}^2}}$$

Outer Prod

Rank 1 matrix

- The simplest Adaptive Algorithm called AdaGrad (Duchi et al 2011) is:

$$w_{t+1} = w_t - \alpha_t H_t^{-1} \nabla f(w_t)$$

Base learning rate

A different perspective today from https://www.ruder.io/optimizing-gradient-descent/#rmsprop

Perspective of accummulating gradients - (Momentum methods) and gradient squares (adaptive learning rate for which you can also recall our discussion on 4 different Armijo conditions)

[1] Reminiscent of the Hessian

One Equation to Unify them All!

- Recall $H_T = \text{diag}\left(\left[\sum_{t=1}^{T} \eta_t g_t g_t^T\right]^{1/2}\right) \implies \left[H_t^{-1}\right]_{jj} = \frac{1}{\sqrt{\sum_{t=1}^{T} \eta_t g_{tj}^2}}$.
- Now we generalize by pivoting on some G_t (see table below) and $D_t = \text{diag}(g_t \circ g_t)$ (element-wise product)
- We unify all adaptive and non-adaptive variants into a single update equation:

$$w_{t+1} = w_t - \alpha_t H_t^{-1} \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t H_t^{-1} H_{t-1}(w_t - w_{t-1})$$

where $H_t = \sqrt{G_t}$ (elementwise)

One Equation to Unify them All!

- Recall $H_T = \text{diag}\left(\left[\sum_{t=1}^{T} \eta_t g_t g_t^T\right]^{1/2}\right) \implies \left[H_t^{-1}\right]_{jj} = \frac{1}{\sqrt{\sum_{t=1}^{T} \eta_t g_{tj}^2}}$.
- Now we generalize by pivoting on some G_t (see table below) and $D_t = \text{diag}(g_t \circ g_t)$ (element-wise product)
- We unify all adaptive and non-adaptive variants into a single update equation:

$$w_{t+1} = w_t - \alpha_t H_t^{-1} \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t H_t^{-1} H_{t-1}(w_t - w_{t-1})$$

pure velocity by somewhat discounting effect of H over time

where $H_t = \sqrt{G_t}$ (elementwise)

Possibilities for G_t → I (non-adaptive SGD/momentum)

→ $\psi_t(G_{t-1}, D_t)$

$D_t = \begin{bmatrix} g_{t1}^2 & & \\ & g_{t2}^2 & \\ & & \ddots \\ & & & g_{tj}^2 \end{bmatrix}$

One Equation to Unify them All!

- Recall $H_T = \text{diag}\left(\left[\sum_{t=1}^{T} \eta_t g_t g_t^T\right]^{1/2}\right) \implies \left[H_t^{-1}\right]_{jj} = \frac{1}{\sqrt{\sum_{t=1}^{T} \eta_t g_{tj}^2}}$.
- Now we generalize by pivoting on some G_t (see table below) and $D_t = \text{diag}(g_t \circ g_t)$ (element-wise product)
- We unify all adaptive and non-adaptive variants into a single update equation:

$$w_{t+1} = w_t - \alpha_t H_t^{-1} \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t H_t^{-1} H_{t-1}(w_t - w_{t-1})$$

where $H_t = \sqrt{G_t}$ (elementwise)

	SGD	**HB**	**NAG**	**AdaGrad**	**RMSProp**	**Adam**
G_t	I	I	I	$G_{t-1} + D_t$	$\beta_2 G_{t-1} + (1-\beta_2) D_t$	$\frac{\beta_2}{1-\beta_2^t} G_{t-1} + \frac{1-\beta_2}{1-\beta_2^t} D_t$
α_t	α	α	α	α	α	$\alpha \frac{1-\beta_1}{1-\beta_1^t}$
β_t	0	β	β	0	0	$\frac{\beta_1(1-\beta_1^{t-1})}{1-\beta_1^t}$
γ	0	0	β	0	0	0

One Equation to Unify them All!

Recall that all derived matrices here are diagonal

Multiplication by matrix inverse is amounting to division

- Recall $H_T = \text{diag}\left(\left[\sum_{t=1}^{T} \eta_t g_t g_t^T\right]^{1/2}\right) \Longrightarrow \left[H_t^{-1}\right]_{jj} = \frac{1}{\sqrt{\sum_{t=1}^{T} \eta_t g_{tj}^2}}$.

These capture the relative importance of some feature (j) over some other feature

- Now we generalize by pivoting on some G_t (see table below) and $D_t = \text{diag}(g_t \circ g_t)$. (element-wise product)

D_t = squaring g_t & G_t accumulate

- We unify all adaptive and non-adaptive variants into a single update equation:

$$w_{t+1} = w_t - \alpha_t H_t^{-1} \nabla f_{i_t}(w_t + \gamma_t(w_t - w_{t-1})) + \beta_t H_t^{-1} H_{t-1}(w_t - w_{t-1})$$

where $H_t = \sqrt{G_t}$ (elementwise)

Fix significantly diminishing learning rate created by Adagrad ie $\alpha_t H_t^{-1} \to 0$

$w_{t+1} = w_t - \alpha \nabla f_{i_t}(w_t)$ $\quad w_{t+1} = w_t - \alpha \nabla f_{i_t}(w_t + \beta(w_t - w_{t-1})) + \beta(w_t - w_{t-1})$

	SGD	HB	NAG	AdaGrad	RMSProp	Adam =Adaptive(RMSProp)+Momentum
$H_t =$ G_t	I	I	I	$G_{t-1} + D_t$	$\beta_2 G_{t-1} + (1-\beta_2) D_t$	$\frac{\beta_2}{1-\beta_2^t} G_{t-1} + \frac{1-\beta_2}{1-\beta_2^t} D_t$
α_t	α	α	α	α	α	$\alpha \frac{1-\beta_1}{1-\beta_1^t}$
β_t	0	β	β	0	0	$\frac{\beta_1(1-\beta_1^{t-1})}{1-\beta_1^t}$
γ	0	0	β	0	0	0

https://ruder.io/optimizing-gradient-descent/index.html#rmsprop

Question: With the background provided
on the previous slide,
can you write down in details,
1) the steps of the ADAM algorithm
(which uses Heavy Ball Momentum)
2) the steps of a similar NADAM algorithm
(which uses Nestorov Accerated Momentum)

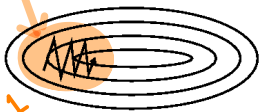
More on ADAM

- Adam is basically HB Momentum + Adaptive.
- Define $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
- Define $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \circ g_t$
- Intuition of m_t and v_t are estimates of first moment (mean) and second moment (uncentered variance) of the gradients.
- Since m_t and v_t are initialized to 0, they are biased towards zero when the decay rates are small. To counter this, they are further normalized by $1-\beta^t$.
- Define $\hat{m}_t = m_t/(1-\beta_1^t)$ and $\hat{v}_t = v_t/(1-\beta_2^t)$.
- The ADAM update is $w_{t+1} = w_t - \alpha_t \hat{m}_t \circ \hat{v}_t^{-1/2}$
- Parameters used in practice: $\beta_1 = 0.9, \beta_2 = 0.999$.

More on ADAM

NADAM = NAG + Adaptive

- Adam is basically HB Momentum + Adaptive.
- Define $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ Accumulating gradients in momentum (in a diminishing manner)
- Define $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \circ g_t$ Accummulation of squared gradients (in a diminishing manner)
- Intuition of m_t and v_t are estimates of first moment (mean) and second moment (uncentered variance) of the gradients. (that is of the trajectory of updates)
- Since m_t and v_t are initialized to 0, they are biased towards zero when the decay rates are small. To counter this, they are further normalized by $1-\beta^t$.
- Define $\hat{m}_t = m_t/(1-\beta_1^t)$ and $\hat{v}_t = v_t/(1-\beta_2^t)$.
- The ADAM update is $w_{t+1} = w_t - \alpha_t \hat{m}_t \circ \hat{v}_t^{-1/2}$
- Parameters used in practice: $\beta_1 = 0.9, \beta_2 = 0.999$.



Will make $\left(\frac{1}{1-\beta^t}\right) > 1$

Extensions

Numerous extensions of the above techniques

- AdaMax is an extension of ADAM to use the l_{infty} norm (i.e. max) instead of square.
- NADAM applies Nesterovs momentum instead of HB Momentum to Adaptive Methods.
- ADADelta is an extension of RMSProp to use the RMS operator on the weight differences as well.
- Recent Algorithm (AMSGrad) by Reddi et al (ICLR 2018) which fixes a theoretical error in ADAM (causing it to not converge even for convex functions) simply by ensuring v_t's remain positive!
- See more details to compare the different optimization algorithms (and also what they are) here: `https://ruder.io/optimizing-gradient-descent/`.

Extensions

Numerous extensions of the above techniques

- AdaMax is an extension of ADAM to use the l_{infty} norm (i.e. max) instead of square.
- NADAM applies Nesterovs momentum instead of HB Momentum to Adaptive Methods.
- ADADelta is an extension of RMSProp to use the RMS operator on the weight differences as well.
- Recent Algorithm (AMSGrad) by Reddi et al (ICLR 2018) which fixes a theoretical error in ADAM (causing it to not converge even for convex functions) simply by ensuring v_t's remain positive!
- See more details to compare the different optimization algorithms (and also what they are) here: `https://ruder.io/optimizing-gradient-descent/`.

Theoretical Results

Numerous extensions of the above techniques

- The first theoretical result was shown for AdaGrad. The convergence result there is a *Regret* bound which is common for online algorithms.
- As mentioned above, the paper introducing ADAM actually had a bug in its analysis. The same also holds for RMSProp, AdaDelta and NADAM etc. They do not have **theoretical Regret bounds** backing them.
- Paper introducing AMSGrad showed regret bounds with a modified version of ADAM (and correspondingly RMSProp, NADAM, ...)
- All this only holds for convex functions. No results known for Non-Convex Functions.

Theoretical Results

Numerous extensions of the above techniques

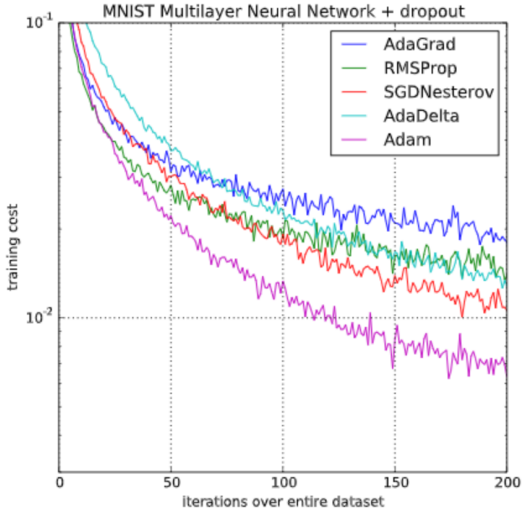
- The first theoretical result was shown for AdaGrad. The convergence result there is a *Regret* bound which is common for online algorithms.
- As mentioned above, the paper introducing ADAM actually had a bug in its analysis. The same also holds for RMSProp, AdaDelta and NADAM etc. They do not have **theoretical Regret bounds** backing them.
- Paper introducing AMSGrad showed regret bounds with a modified version of ADAM (and correspondingly RMSProp, NADAM, ...)
- All this only holds for convex functions. No results known for Non-Convex Functions.

Comparison of the different methods

Recall our stuggle through Armijo condition variants for improved convergence in the case of simple gradient descent
https://colab.research.google.com/drive/1f1y8_ZkL_WS7Kfd7q-KB9Yt2VME1wV35#scrollTo=Kdd7X5dXI5Go

- AdaGrad (Duchi et al 2011) is one of the most influential papers of the last decade!
- The starting point of numerous new techniques for adaptive methods.
- There is really no one technique that is provably better than the other. Each technique has its own pros and cons!
- In the next few slides, I'll try to put together a few takeaways from some recent papers which have studied this specifically for non-convex optimization.

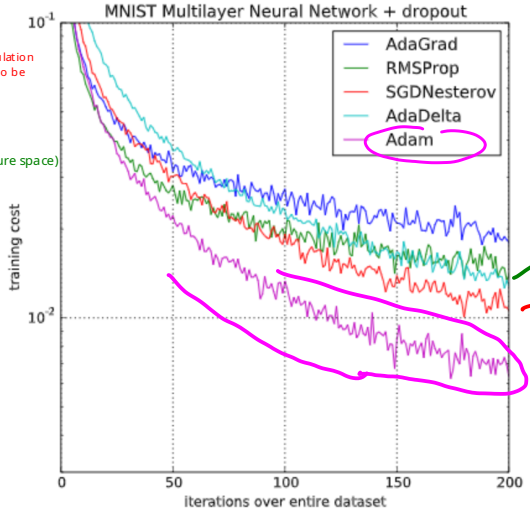
Kingma et al, ICLR 2015 – Original ADAM paper



Kingma et al, ICLR 2015 – Original ADAM paper

you need gradient accumulation over time for momentum to be effective

Adaptive starts fast (by quickly adapting to feature space) and then slows down



Adaptive + Momentum

Momentum has best of both worlds!

Adaptive vs Non Adaptive Techniques: Comparisons

- Benefits of AdaGrad: AdaGrad can significantly improve upon SGD in sparse feature sets! It automatically sets the learning rate, and secondly, automatically updates the learning rates with a decay schedule! Also, it has a per coordinate learning rate!
- In dense settings and particularly in deep models, Adagrad works very poorly because of rapid decay in learning rates
- ADAM, RMSProp, AdaDelta, ... all try to fix this issue!
- In many cases, these adaptive algorithms improve upon SGD in terms of training loss and better/faster convergence!
- Also, momentum generally improves upon the non-momentum variants!
- But....

Adaptive vs Non Adaptive Techniques: Comparisons

$\alpha H_t^{-1} = \left[\alpha/\sqrt{G_{ii}} \dots\right]$ → amounts to multiplying RHS by $\alpha/\sqrt{G_i}$

Adaptive starts fast (by quickly adaptiving to feature space)

- Benefits of AdaGrad: AdaGrad can significantly improve upon SGD in sparse feature sets! It automatically sets the learning rate, and secondly, automatically updates the learning rates with a decay schedule! Also, it has a per coordinate learning rate! (αH_t^{-1})
- In dense settings and particularly in deep models, Adagrad works very poorly because of rapid decay in learning rates
- ADAM, RMSProp, AdaDelta, ... all try to fix this issue!
- In many cases, these adaptive algorithms improve upon SGD in terms of training loss and better/faster convergence!
- Also, momentum generally improves upon the non-momentum variants!
- But....

Second order moments (gradient squares) help
when you are already availaing first order moments (gradients)

Back to SGD! Generalization....

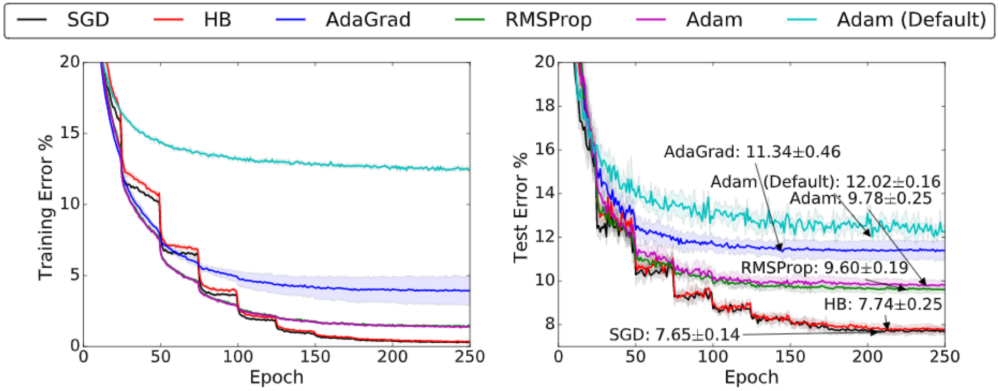
- But, in Machine Learning, Generalization is more important compared to just Training Loss!
- Recent works (for example, Wilson et al, The Marginal Value of Adaptive Gradient Methods in Machine Learning) showed a very surprising result!
- Though adaptive gradient methods tend to minimize training loss better, they do so by obtaining more complex and less generalizable solutions!
- They gave a few synthetic examples (particularly in over parameterized scenarios) where SGD and its variants obtain the less complex solutions but Adaptive variants obtain solutions which do not generalize well!

Back to SGD! Generalization....

- But, in Machine Learning, Generalization is more important compared to just Training Loss!
- Recent works (for example, Wilson et al, The Marginal Value of Adaptive Gradient Methods in Machine Learning) showed a very surprising result!
- Though adaptive gradient methods tend to minimize training loss better, they do so by obtaining more complex and less generalizable solutions!
- They gave a few synthetic examples (particularly in over parameterized scenarios) where SGD and its variants obtain the less complex solutions but Adaptive variants obtain solutions which do not generalize well!

Back to SGD! Generalization....

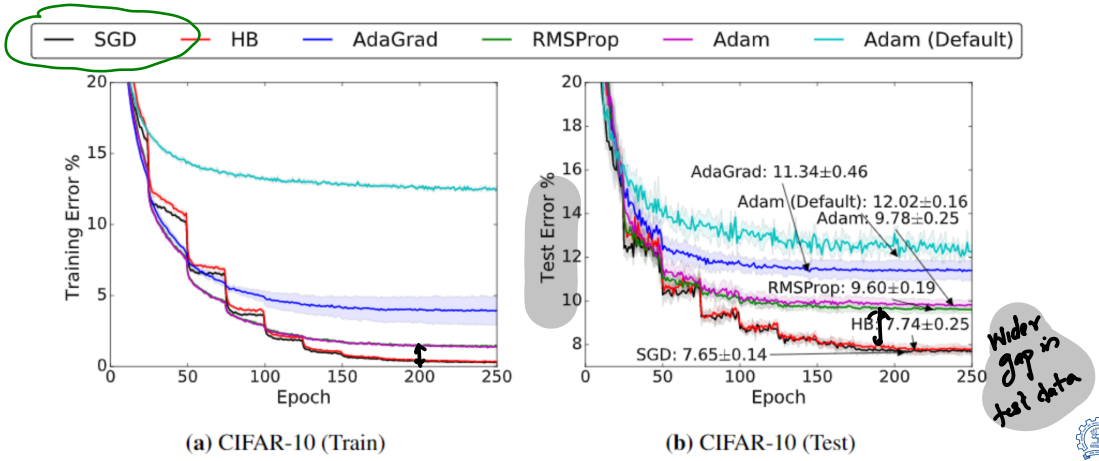
See Wilson et al, The Marginal Value of Adaptive Gradient Methods in Machine Learning, NeurIPS 2017



(a) CIFAR-10 (Train) (b) CIFAR-10 (Test)

Back to SGD! Generalization....

See Wilson et al, The Marginal Value of Adaptive Gradient Methods in Machine Learning, NeurIPS 2017



(a) CIFAR-10 (Train) (b) CIFAR-10 (Test)

Additional Reading

- Wilson et al, The Marginal Value of Adaptive Gradient Methods in Machine Learning, NeurIPS 2017
- Reddi et al, On the Convergence of ADAM and Beyond, ICLR 2018.
- Kingma and Ba, ADAM: A Method for Stochastic Optimization, ICLR 2015
- Duchi et al, Adaptive subgradient methods for online learningand stochastic optimization, Journal of Machine Learning Research 2011.
- Zeiler. ADADELTA: An Adaptive Learning Rate Method, ArXiv 2012
- `https://ruder.io/optimizing-gradient-descent/`

Empirical Risk Minimization and Gradient Descent

Start of Extra and optional reading

Some analysis that are more specific to Machine Learning

Extra and optional reading

- Learning as mathematical optimization
  - Stochastic optimization, ERM, online regret minimization
  - Offline/online/stochastic gradient descent
- Regularization
  - AdaGrad and optimal regularization
- Gradient Descent++
  - Frank-Wolfe, acceleration, variance reduction, second order methods, non-convex optimization

Recap: Machine Learning as Optimization

$$\widehat{\mathbf{w}}^* = \text{argmin}_{\mathbf{w}} \; \mathcal{L}(\mathbf{w}) + O(\mathbf{w}) \tag{1}$$

where $O(\mathbf{w})$ is the regularization term.

- **0-1 Loss:**

$$\mathcal{L}(\mathbf{w}) = \sum_{(\mathbf{x},y)} \delta\left(y \neq \mathbf{w}^T \phi(\mathbf{x})\right) \tag{2}$$

  Minimizing the 0-1 Loss is NP-hard. We therefore look for surrogates.

- **Perceptron:** A Non-convex Surrogate

$$\mathcal{L}(\mathbf{w}) = -\sum_{(\mathbf{x},y)\in\mathcal{M}} y\mathbf{w}^T \phi(\mathbf{x}) \tag{3}$$

  where $\mathcal{M} \subseteq \mathcal{D}$ is the set of misclassified examples.

Convex Surrogates for 0-1 Loss in ML

$$\widehat{\mathbf{w}}^* = \text{argmin}_{\mathbf{w}} \; \mathcal{L}(\mathbf{w}) + \Omega(\mathbf{w}) \tag{4}$$

- **Logistic Regression:**

$$\mathcal{L}(\mathbf{w}) = -\left[\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\mathbf{w}^T\phi(\mathbf{x}^{(i)}) - \log\left(1 + exp\left(\mathbf{w}^T\phi\left(\mathbf{x}^{(i)}\right)\right)\right)\right)\right] \tag{5}$$

- **Sigmoidal Neural Net:**

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\sigma_t^L\left(\mathbf{x}^{(i)}\right)\right) + \left(1 - y_k^{(i)}\right)\log\left(1 - \sigma_t^L\left(\mathbf{x}^{(i)}\right)\right)\right] \tag{6}$$

Empirical Risk Minimization and Projected Grad Descent

- Gradient depends on all data
- What about generalization?
- Simultaneous optimization and generalization
  - Faster optimization! (single example per iteration)

Statistical (PAC) learning

- $\mathcal{D}$: i.i.d distribution over $\mathcal{X} \times \mathcal{Y} = \{(\mathbf{x}^i, y^i)\}$
- Goal: To learn Hypothesis h from hypothesis class $\mathcal{H}$ that minimizes expected loss $err(h) = \mathbf{E}\left[\mathcal{L}(\mathbf{x}^i, y^i, \mathbf{w})\right]$.
- $\mathcal{H}$ is (PAC) learnable if $\forall \epsilon, \delta > 0$, there exists algorithm s.t. after seeing M examples, where $M = \mathcal{O}\left(poly(\delta, \epsilon, dimension(\mathcal{H}))\right)$, the algorithm finds h s.t. with probability $1 - \delta$,

$$err(h) \leq \min_{h^* \in \mathcal{H}} err(h^*) + \epsilon$$

Online Learning and Regret Minimization

- For $t = 1, 2 \ldots T$, $h^t \in \mathcal{H}$, and an adversarial example $(\mathbf{x}^t, y^t)$, minimize expected regret:

$$\frac{1}{T}\left[\sum_t \mathcal{L}(h^t, \mathbf{x}^t, y^t) - \min_{h^* \in \mathcal{H}} \sum_t \mathcal{L}(h^*, \mathbf{x}^t, y^t)\right]$$

- Generalization in PAC setting is achieved by vanishing of the regret:

$$\frac{1}{T}\left[\sum_t \mathcal{L}(h^t, \mathbf{x}^t, y^t) - \min_{h^* \in \mathcal{H}} \sum_t \mathcal{L}(h^*, \mathbf{x}^t, y^t)\right] \xrightarrow{T \to \infty} 0$$

Summarizing Analysis for Stochastic Gradient Descent

- One example per step, same convergence properties as gradient descent and additional analysis provides **direct generalization**!

$$\mathbf{E}\left[\frac{1}{T}\sum_{t=1}^{T}\mathcal{L}(\mathbf{x}^t, y^t, \mathbf{w}^t) - \mathcal{L}(\mathbf{x}^t, y^t, \mathbf{w}^*)\right] \leq \mathbf{E}\left[\frac{RB}{\sqrt{T}}\right]$$

- To get solution that is ϵ approximate with $\epsilon = \frac{\mathbf{R}B}{\sqrt{T}}$, you need number of gradient iterations that is $T = \left(\frac{\mathbf{R}B}{\epsilon}\right)^2 = O\left(\frac{1}{\epsilon}\right)^2$ where B is upper bound on the gradient norm and R is radius of ball around $\mathbf{w}^*$ containing the iterates.
- Recall that $\mathcal{H}$ is (PAC) learnable if $\forall \epsilon, \delta > 0$, there exists algorithm s.t. after seeing M examples, where $M = \mathcal{O}\left(poly(\delta, \epsilon, dimension(\mathcal{H}))\right)$, the algorithm finds h s.t. w.p. $1 - \delta$,

$$err(h) \leq \min_{h^* \in \mathcal{H}} err(h^*) + \epsilon$$

- Thus, the expected number of iterations for ϵ approximation is $T = M\left(\frac{\mathbf{R}B}{\epsilon}\right)^2 = O\left(M\left(\frac{1}{\epsilon}\right)^2\right)$

Empirical Risk Minimization and Gradient Descent

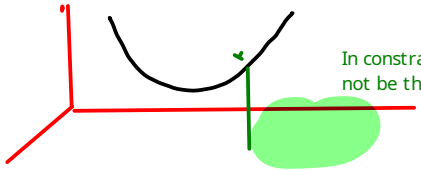
END of Extra and optional reading

So far..

- Gradient Descent Analysis for Lipschitz Smoothness/Continuity/(Strong) Convexity [Done]
  - Our running Colab Notebook:
- Nestorov's and Polyak's accelerated gradient descent [Done]
- Sub-gradient Descent and Analysis [Done]
- Stochastic Gradient Descent [Done]
- Generalized Gradient Descent [Next]
- Proximal Gradient Descent
- Projected Gradient Descent for Constrained Optimization
- Discrete/Combinatorial Optimization for Subset Selection
- Constrained Optimization, Duality & KKT Conditions
- And so on...more such algorithms

Generalized Gradient Descent - generalizing (Sub)gradient Descent for Non-Differentiable and Constrained Cases

Back to representing x as variables (w was used to represent variables only in the case of stochastic algorithms applied to Machine Learning)



In constrained minimization, the constrained minimum might not be the unconstrained minimum

Generalized Gradient Descent - generalizing (Sub)gradient Descent

- Consider the following sum of a differentiable function $f(\mathbf{x})$ and a nondifferentiable function $c(\mathbf{x})$ (an example being $\sum_i I_{C_i}(\mathbf{x})$)

$$\min_{\mathbf{x}} F(\mathbf{x}) = \min_{\mathbf{x}} f(\mathbf{x}) + c(\mathbf{x})$$

Generalized Gradient Descent - generalizing (Sub)gradient Descent

- Consider the following sum of a differentiable function $f(\mathbf{x})$ and a nondifferentiable function $c(\mathbf{x})$ (an example being $\sum_i I_{C_i}(\mathbf{x})$)

$$\min_{\mathbf{x}} F(\mathbf{x}) = \min_{\mathbf{x}} f(\mathbf{x}) + c(\mathbf{x})$$

we will later use h(x) for proximal (where h(x) won't denote constraint but just non-differentiable fn such as $\|x\|_1$)