GRADIENT DESCENT FOR DEEP LEARNING

KAIM `17

Anand Krish

Agenda

- Deep Learning & Optimization
  - *The Old that is New*
  - *What is being Optimized?*
- Gradient Descent Algorithms
  - *Why Gradient Descent?*
  - *Overview of Stochastic Variants*
- Discussion

EPISODE 1

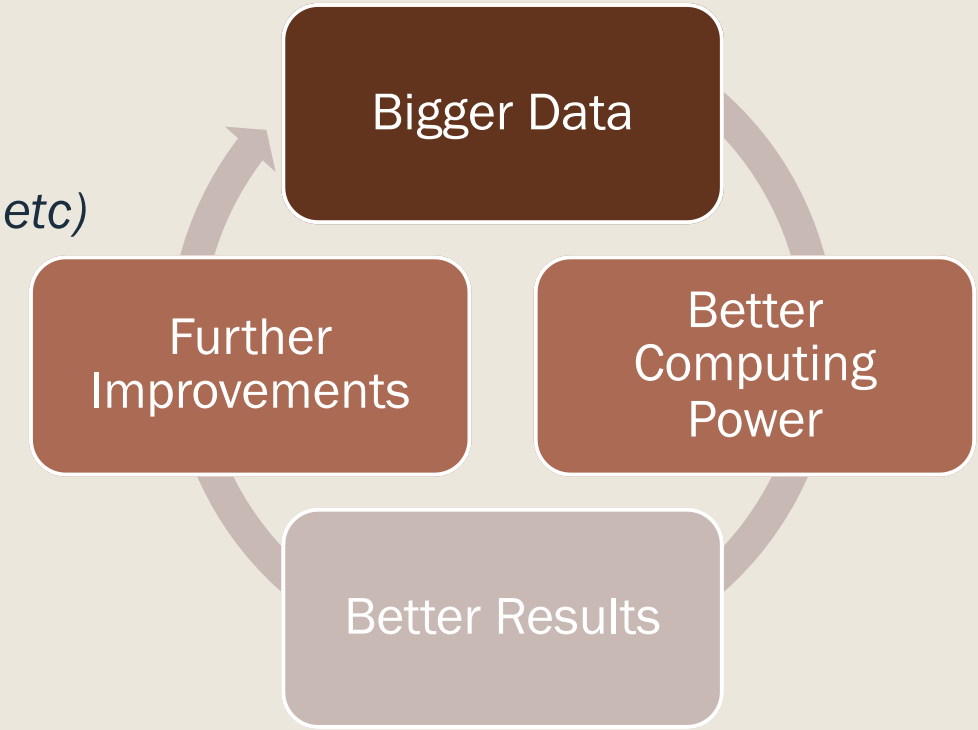
DEEP LEARNING & OPTIMIZATION

Introduction to Deep Learning

- Automatic feature extraction and selection
- Hierarchical data representation – based on levels of abstraction
- Perform broad categories of tasks over **generic datasets**
- “Kinda-sorta” biologically inspired
- Automatic learning
- Multi-dimensional
- **Proof of learning by generation**
- Learn, Unlearn and Relearn

The Old that is New

- Deep Learning is a modern (fancy) name for the idea that has existed since 1980s
  - *Convolutional Neural Networks was proposed in 1998 [LeCun et. al]*
  - *LSTMs [example of a Recurrent Neural Network] in 1997*
- Successfully resurrected due to –
  - *Availability of large amounts of data*
  - *Higher Computing Power (GPUs, TPUs etc)*
  - *Bigger Models*
  - *Better Learning Algorithms - Backprop*

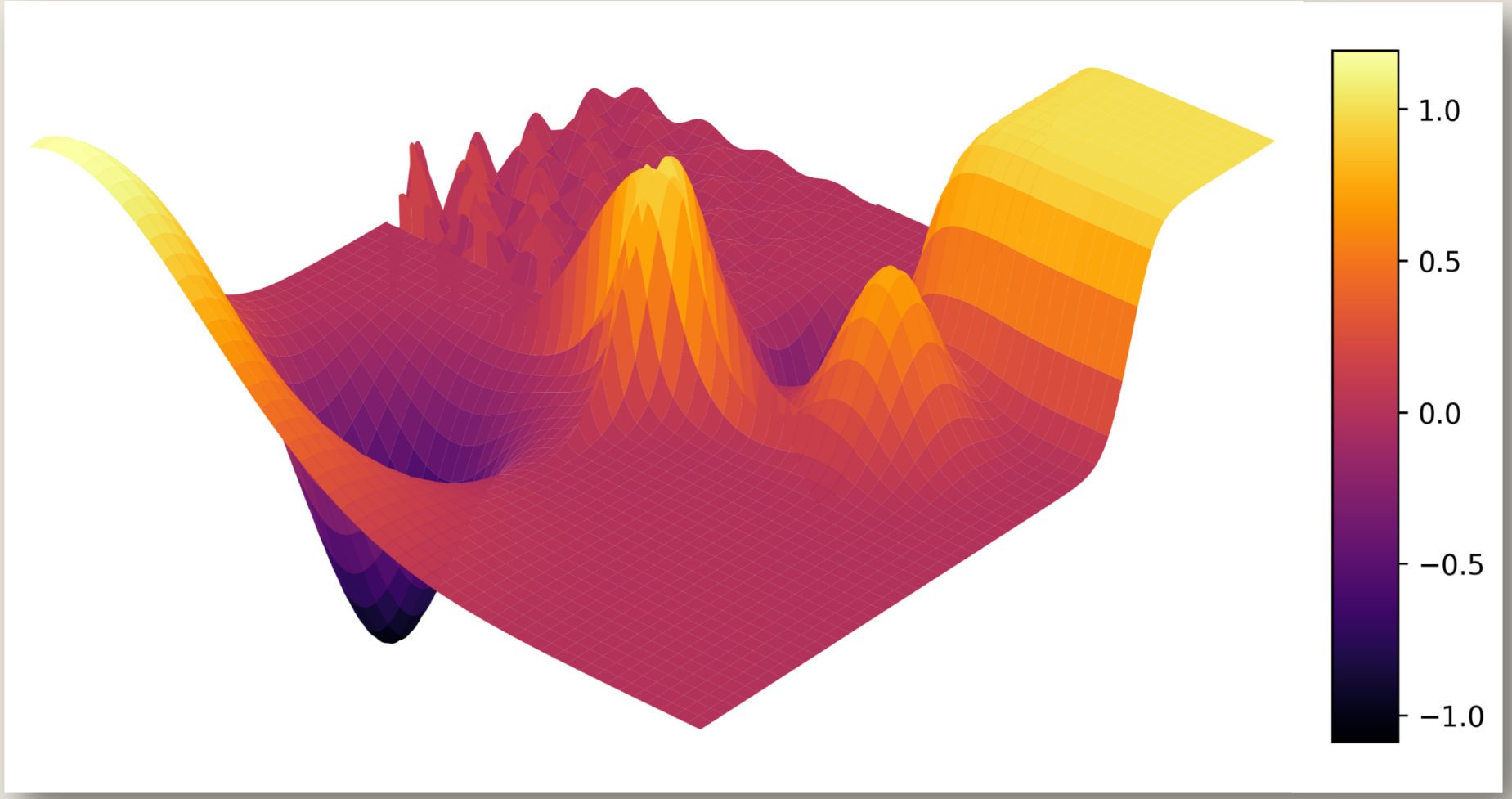


Optimise!!! … But What?

- The error between our model prediction and the target of course!
- **Beginners' Wisdom**: Deep Learning = Optimization
- But do what we want? – Reduction in the generalization Error
- What do we do? – Reduce the Training Error
- Training Error and Generalization error are **NOT** the same
  - *Has dire consequences (meh! We will still do it)*
- Ergo, Deep Learning ≠ Optimization
- **Consensual Reality :** Optimize *anything* that reduces the generalization error
  - *Finding it is tough!*

Optimization Landscape

- Local Extrema, Saddle Points, Cliffs, Plateaus, Narrow gorges and sharp peaks



EPISODE 2

GRADIENT DESCENT

The Gradient Descent

- Minimize a function without any specific knowledge about it –
- Use Taylor Expansion to approximate around a neighbourhood

$$J(x+h) = J(x) + \nabla J(x)^T h + \frac{1}{2} h^T H(x) h + \ \ldots$$

  - *Let* $h = -\alpha \nabla J(x)$
  - $J(x - \alpha \nabla J(x)\ ) = J(x) - \alpha \nabla J(x)^T \nabla J(x) < J(x)$
- $\alpha =$ Learning Rate - controls how big the step is going to be
- Decaying Learning rates
  - $\alpha_k = (1 - \gamma)\alpha_0 + \gamma \alpha_n$
  - *Decay gradient for proper convergence*
  - *Can be interpreted as a stabilizing parameter to prevent oscillations*

Idea – Minibatch

- Batch processing – Computationally hard, not so good for non-smooth or non-convex error surface
- Randomly select few data points (say m) from the training set (of size N) to train
- The subset is called as *minibatch*
- Advantages –
  - *Weight updates depend on the minibatch* size $m \ll N$
  - *Computing the gradient update becomes computationally cheaper*
  - *True Expectation of the gradient rather than the empirical mean*
    [Reason: Destroys correlation between data points]
- But accuracy of the gradient is compromised – Yet guaranteed to converge!!
- Keep in mind: The subset must be randomly chosen i.e. unbiased.
- So what's the catch?
  - *Slower convergence*

Stochastic Gradient Descent

Algorithm 1 Stochastic Gradient Descent

Require: $\alpha_0, \ldots, \alpha_T$: The learning rates for each timestep (presumably annealed)
Require: $f_i(\theta)$: Stochastic objective function parameterized by θ and indexed by timestep i
Require: θ_0: The initial parameters
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f_t(\theta_{t-1})$
    $\theta_t \leftarrow \theta_{t-1} - \alpha_t \mathbf{g}_t$
  end while
  return θ_t

Idea – Momentum

- A ball falling along a slope gathers momentum (due to gravity) ⟶ Falls faster
- Incorporate *Momentum* in our gradient descent algorithm (But without gravity of course!)
- Simple – Gradient direction is weighted based on previous directions
- Problem – Overshoot Solution – Average the past gradients
- Advantages –
  - *Much faster convergence*
  - *Omits narrow gorges and performs well along cliffs*
  - *Avoids sudden changes in gradient directions*
- Insight – Basically acts as a correcting mechanism

SGD with Momentum

Algorithm 1 Stochastic Gradient Descent with Nestorov Momentum

Require: Initial parameter θ, initial velocity $\mathbf{v}$ Stochastic Objective function L.

procedure SGD(α, β)
  while stopping criterion is not met **do**
    Sample minibatch of m samples from the training set
    $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \beta \mathbf{v}$ ▷ Interim Update (Look-ahead step)
    $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \tilde{\boldsymbol{\theta}}), y_i)$ ▷ Compute gradient
    $\mathbf{v} \leftarrow \beta \mathbf{v} - \alpha \mathbf{g}$ ▷ Velocity Update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{v}$ ▷ Parameter Update
  end while
end procedure

Hmm… How can we improve it further?

- What if the algorithm can automatically adapt the learning rates irrespective of the terrain?
  - *Convex or Non-convex terrains, shallow valleys etc*
- Adaptive learning rate for each of the parameters to be optimised
- Account for initialization bias i.e. irrespective of where we start, the algorithm must perform optimally (**Extremely Important**)
- Less sensitive to sudden asymmetric changes in the landscape i.e in some gradient directions but not the other

Idea – Adaptive Learning Rates

- Introduce adaptive learning rate mechanism for every parameter
- Introduce 2 different learning rates
  - m_1: *Same as original momentum with decaying weights*
  - m_2: *Momentum based on squared gradient [Avoids cliffs and jumps over plateaus]*
- Two hyper-parameters β_1, β_2 for decaying the above momenta
- Account for initialization bias by correcting moments to zero
- Decay the learning rate based on the above mechanism

ADAM Optimizer

Algorithm 2 Adam Optimizer with learning rate decay

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\mathbf{v}$, initial learning rate α_0. Stochastic Objective function L, $\delta = 10^{-8}$ for numerical stabilization

procedure ADAM(α, β_1, β_2)
  $\mathbf{m}_1 \leftarrow \mathbf{0}, \mathbf{m}_2 \leftarrow \mathbf{0}$ ▷ Initialise the moments
  $t \leftarrow 1$ ▷ Initialise time step
  while stopping criterion is not met **do**
    Sample minibatch of m samples from the training set
    $\mathbf{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \boldsymbol{\theta}), y_i)$ ▷ Compute gradient
    $\mathbf{m}_1 \leftarrow \beta_1 \mathbf{m}_1 + (1 - \beta_1)\mathbf{g}$ ▷ 1^{st} Moment Update
    $\mathbf{m}_2 \leftarrow \beta_2 \mathbf{m}_2 + (1 - \beta_2)\mathbf{g} \odot \mathbf{g}$ ▷ 2^{nd} Moment Update
    $\hat{\mathbf{m}}_1 \leftarrow \mathbf{m}_1 / (1 - \beta_1^t)$ ▷ Correcting 1^{st} Moment bias
    $\hat{\mathbf{m}}_2 \leftarrow \mathbf{m}_2 / (1 - \beta_2^t)$ ▷ Correcting 2^{nd} Moment bias
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha_t \, \hat{\mathbf{m}}_1 / (\sqrt{\hat{\mathbf{m}}_2} + \delta)$ ▷ Parameter Update
    $\alpha_t \leftarrow e^{-\gamma} \alpha_t$ ▷ Learning rate decay
    $t \leftarrow t + 1$ ▷ Update time step
  end while
end procedure

DISCUSSION TIME