Optimization: A Multi-armed Bandit In Deep Neural Nets

Snehanshu Saha

CSIS & Anuradha and Prashanth Palakurthi Centre for Artificial Intelligence Research (APPCAIR), BITS PILANI K K Birla Goa Campus
Visiting Scientist, ISI Bangalore

Data/Papers: http://astrirg.org/projects.html
ML Blog: https://beginningwithml.wordpress.com/
Github: https://github.com/sahamath?tab=repositories

Highlights

A Fragmented Timeline

- Pattern Recognition (SDPR): Lesser training issues
- The Elegance and the Masters
- Porting SDPR-What did we miss? SVM Kernels in CKN? Shift-invariance?
- The Elegance can't be ignored
- Re-emergence of (Methodological) elegance in AI?
- No Free Lunch, Mate! Bandit Optimization
- Cost of Training?784 hidden layers, really?
- Bring in Dynamical systems? Additional Baggage?
- Cost of losing interpretability? Which Loss to choose?

NN: Gentle Introduction

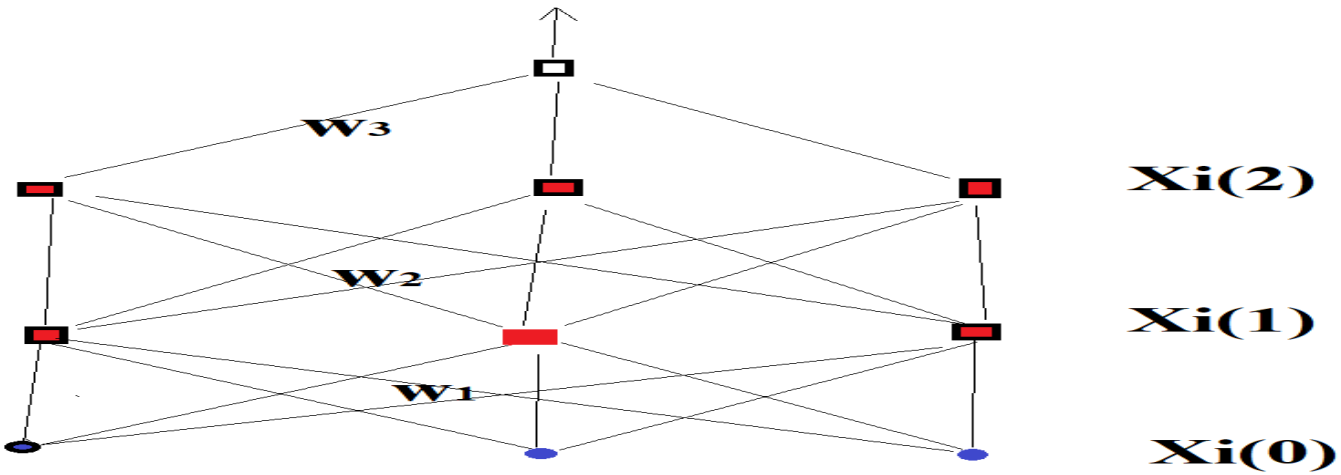


Figure: A Non-artiste's view of a NN

A neural network contains $m+1$ layers. The first layer describes the input vector $x(0)$: $x_i(0) = x_i^1(0), x_i^2(0), ..., x_i^n(0); i = 1; ....; I$

The image of the input vector $x_i(0)$ on the $k-$th layer is denoted: $x_i(k) = x_i^1(k), x_i^2(k), ..., x_i^n{}_k(k); i = 1; ....; l$ where n_k is the dimensionality of vectors $x_i(k); i = 1; ....; l$. n_k is the number of neurons in $k-$th layer and $n_m = 1$. The layer $k-1$ is connected with $k-$th layer through the $n_k * n_k - 1$ matrix $w(k)$:
$x_i = Sw(k)x_i(k-1); k = 1; ...; m;$ and $i = 1; ..; l$:
Where $S(.)$ is a sigmoid function such that $S(-\infty) = 0, S(+\infty) = 1$. For example:
$S_1(u) = \frac{1}{1+e^-u}; S_2(u) = \frac{2arctan(u)+\pi}{2\pi}$
Universal approximation theorems (for example G. Cybenko 1989): one-hidden layer is adequate but with no control over number of neurons. Deep neural networks thus found practical value. How?
Deep Vs Shallow: (computationally) infeasible to have a very large number of neurons in the single hidden layer network. Parameters: the number of inputs, hidden layer size and output layer size: When the hidden layer size $--> $ beyond control, this is a huge set of parameters to deal with while learning. Deep: use multi-hidden layer structures to reduce individual layer size...GD/SGD. A composition of the activations at the hidden layers discovered from the past

The Problem

Constrained (non-convex) Minimization Problem for (Deep/Shallow) Neural Networks

The goal is to minimize the functional:
$l(w(1), ..., l(w(m)) = \sum_{i=1}^{l}(y_i - x_i(m))^2$
subject to constraints: $x_i = Sw(k)x_i(k-1); k = 1; ..; m;$ and $i = 1; ...; l$
Lagrange Multiplier:
$L(W, X, B) =$
$(1/l)\sum_{i=1}^{l}(y_i - x_i(m))^2 - \sum_{i=1}^{l}\sum_{i=k}^{m} b_i(k).[x_i(k) - Sw(k)x_i(k-1)$

Necessary Conditions

Local Minimum

The necessary conditions are: $\nabla W_{B;X;L}(W;X;B) = 0$; which can be solved to obtain $b_i(k); x_i(k); w_i(k)$; for $i = 1; ...; l; k = 1; ...; m$:

3 conditions

- Condition 1: $\frac{\partial L(W,X,B)}{\partial b_i(k)} = 0 \forall i, k$: Forward Pass Equations
- Condition 2: $\frac{\partial L(W,X,B)}{\partial x_i(k)} = 0 \forall i, k$: Backward Pass (Last Layer and Hidden layers)
- Condition 3: $\frac{\partial L(W,X,B)}{\partial w(k)} = 0 \forall w(k)$: Backward Pass (evaluated by the method of steepest gradients):
  $w(k+1) = w(k) - \eta(k)\frac{\partial L(W,X,B)}{\partial w(k)}; k = 1, ..., m$ (Update the weight matrix)

The Control Agents

- Activation functions: FP
  UAT: polynomial/non-polynomial?
  Adapt Activations?
- Kind of functions (fitting) to learn
- Deep/Shallow: Complexity
  Certain Activations (ReLU/AReLU ensures convergence in over-parameterized settings with 1-hidden layer NN
- Optimization: Back-propagation
  Gradient Descent: has error bounds for convex problems:
  $n \geq \frac{2L(\boldsymbol{f}(\boldsymbol{w}^0)-\boldsymbol{f}(\boldsymbol{w}^*))}{\epsilon}$ at least, $n = O(\frac{1}{\epsilon})$ iterations to achieve the error bound: $\left\|\nabla f(w^k)\right\| \leq \epsilon; \|f(.)\| \leq L$ (Non-Convex case)
  $n \geq \frac{(\boldsymbol{f}(\boldsymbol{w}^0)-\boldsymbol{f}(\boldsymbol{w}^*))}{\epsilon}$ (Convex case) [1]
- Order of approximation: Hard Problems: RL type

[1] Saha et. al: LALR, Applied Intelligence; https://doi.org/10.1007/s10489-020-01892-0

Chinks in the Armor: Bandit Optimization

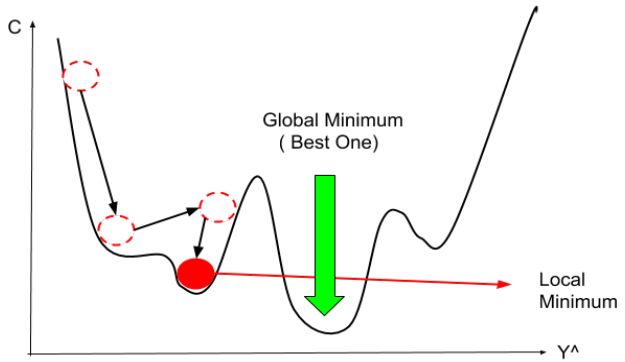
Figure: Jagged Landscape: learning is not easy!

Figure: Taming of the Shrew!

- Usual Suspects: Learning Rates (Lipschitz): (Convergence Speed?)
- Weights and biases: Overparameterization (ReLU/ARelU: 1 hidden layer?): [2]
- Robust Loss Functions: No Free Lunch (label noise): Approximate Loss? Dynamical systems?....Apprx Accuracy)-First/Second Order?
- (Convergence Type?) Pointwise Approximation or Landscape Optimization?
- Adapt Activations? Learn or parameterize (Again?)

[2] Saha, Nagaraj, Mathur, Yedida, Sneha; Evolution of Novel Activation Functions in Neural Network Training for Astronomy Data: Habitability Classification of Exoplanets; EPJ Special Topics (Springer), 229, 2629–2738 (2020)

Rescue Op



- Physics: PSO-DL; $\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y_j)\frac{\partial y_j}{\partial net_j}x_{ji}$; $\frac{\partial y_j}{\partial net_j}$: role of activation function? Error Gradient-Loss Agnostic?
- Global Minima not the problem (Potato chips are!!)...Network size explodes, time to global minima is exponential!

The Really Deep Neural Nets

Society is about to experience an epidemic of false positives coming out of big-data projects-Prof. Michael Jordan, UC Berkeley

Figure: A certain fool's gold element to our current obsession

Deep NNs leading to false, meaningless inferences...

The cost

Computing Setup

The Really Deep Neural Nets

Master Node 2 x Intel(R) Xeon(R) CPU E5-2620 six-core 48 GB Memory (4GB per core) • 900GB SAS 10K x 4 Compute Node : 10 Qty • 2 x Intel(R) Xeon(R) CPU E5-2650 v2 eight-core • 64 GB Memory (4GB per core) • 600GB SAS 10K x 1 GPU Node : 2 Qty • Nvidia Tesla K20Xm cards • 2 x Intel(R) Xeon(R) CPU E5-2650 v2 eight-core • 128 GB Memory (8GB per core) • 900GB SAS 10K x 4

Neural networks (ANN), is a system of interconnected units organized in layers, which processes information signals by responding dynamically to inputs. Layers of the network are oriented in such a way that inputs are fed at input layer and output layer receives output after being processed at neurons of one or more hidden layers.

The Really Deep Neural Nets

Approximate Budget: 75 Lacs!

A hard Optimization Problem

small data AI

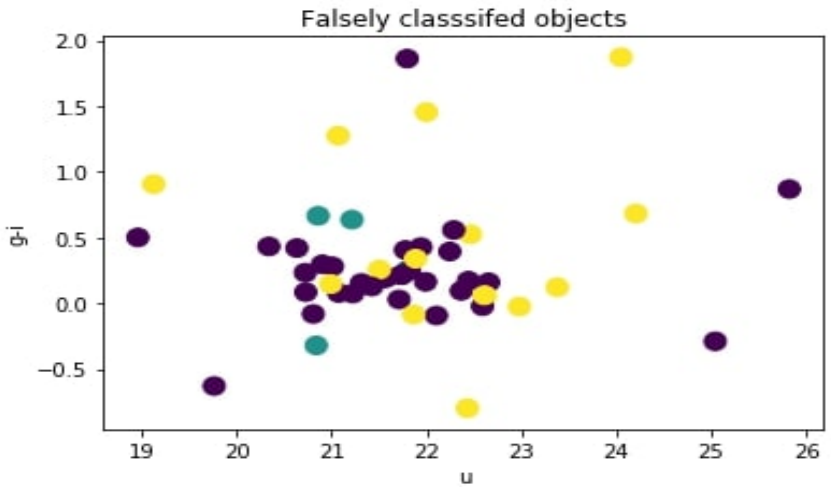


Figure: The 2-class classifier: Redshift can't be used as feature–Violet: stars, yellow: quasars and blue: galaxies

False positives; detected by redshift[a]

[a]Makhija, Saha, Basak, Das; Separating Stars from Quasars: Machine Learning Investigation Using Photometric Data, Astronomy and Computing (Elsevier), 29, Nov 2019

Figure: My collaborator training a DL model in NC State

- Optimize, optimize, optimize.....
- New methods in optimization for prediction error to hit rock bottom!
- Get rid of expensive architectures (too many hidden layers, too many neurons, too many weight updates)
- Single-shot learning

Tame the bandits i.e. hyperparameters

What we need

Advanced Calculus, Advanced Statistics, Linear Algebra
Need some geniuses and their theories: Hilbert, Banach, Cauchy, Lipschitz...
Functional Analysis, Convex optimization, Differential Equations, Chaos theory

Benchmarking & other nonsense

What are the performance benchmarks we should be looking at

- Accuracy, loss etc.. (obvious)–HOW?
- Epochs to converge to optima–If Loss is credible
- CPU/memory utilization–Ditto
- Carbon footprint–Ditto

What we should avoid?

- Inferences that make no sense
- Avoid ridiculous mistakes (redshift, Surf Temp estimation)
- ethical/social/racial misdemeanor

The Learning problem

Key Players: Activation, UAT (Shallow/Deep NN), Gradient Learning, overparameterization, Learning Rate, Gradient Approximation, Non-convex loss landscape, trouble with pointwise approximation

UAT Shallow–functions approximate–No Control over number of neurons (viz. Cybenko)
UAT Deep– Neuron **cardinality** can be controlled (Non-Cybenko; Sonoda-type: Ridgelet Transforms)
Monotonicity–Not required (SWISH/MISH/SBAF)
Parametrize Activations?

Loss Functions: MSE, BCE/CE, MAE $\rightarrow$ (Extensions): Check Loss, Quantile Loss (New: NOT TODAY!)
Some of these are robust to label noise i.e. *interpretable*–Another Day

Loss Minimization

Local Minima/Global Minima/Saddle Point?
Cost to Global Minima: Exponential Time[a], Early G-min: Overfits Local Minima is fine! Pointwise Approximation works in convex/appx convex/non-convex landscape $\rightarrow$ **AdaSwarm** $\rightarrow$ order of approximation? Do we need to approximate gradients everywhere? Borrow from PSO Dynamics $\rightarrow$ **AdaSwarm–Again?**

[a] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," in Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, [Online]. Available: http://proceedings.mlr.press/v38/choromanska15.html

So, what is AdaSwarm?

$$\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y_j)\frac{\partial y_j}{\partial net_j}x_{ji}$$

Uncharacteristic Loss:AdaSwarm?

Heat Equation (Easy); Schrodinger EQ (Harder)–WHY DNN?

What's odd?

- Irregular Loss: Non-Differentiable and Non-standard (NOT MSE/CE)
- Domain Knowledge? Probability Regularization?
- Don't rub it in! Study Physics now??....I'm a CS guy
- Use a Grad-approximation technique? AdaSwarm
- Test AdaSwarm on Non-diff loss: MAE?
- What's good with MAE? robust to label noise
- Challenges: Grad apprx and Auto-Learning rate–HOW?
- AdaSwarm is an approxation method: test it on smooth loss anyway

AdaSwarm?

Merging PSO with Gradient Learning? Control Dynamics?

$$\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y_j)\frac{\partial y_j}{\partial net_j}x_{ji}$$

Loss Agnostic

- Speed - Auto LR, faster
- Approximation -SOTA near optima
- Hyper-parameter Trap - Minimal BUT.....NO Free Lunch, OK?
- Order of Accuracy: First Order–Not higher: Approximation is not as good globally (away from the minima)!
- Approximation: Still point-wise, not landscape (RL?)

The Forward Pass (FP): Activation Functions

back propagation (BP): Loss functions:: **Handle trajectories, jagged landscapes, local minima, smoothness, convexity**

- Hyper-parameters: Adaptive learning: η, Chaos Theory: k, α
- More Hyper-parameters: $c_1, c_2, r_1, r_2, \beta$ from EMPSO: Error Gradient Approximation

- Devil's in the details: lr-η; k, α :: tune the bandits??
- $\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y_j)\frac{\partial y_j}{\partial net_j}x_{ji}$; EMPSO Appx: TODAY!

An Unusual Loss Function

Schrödinger Solver: The first EM-PSO guinea pig!

$$-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}\psi(x) + V(x)\psi(x) = E\psi(x)$$

- argument in favor of classical analytic methods to solve differential equations?
- approximation by discretization is tedious, computationally expensive, and that there is no guarantee of convergence to the analytical solution
- Motivation: construction of the trial function could be avoided by training the NN to satisfy initial/boundary conditions along with the differential equation optimization function.

Schrödinger Loss

gradient-free optimizers. Ex: Non-differentiable functions, discrete feasible space, large dimensionality, multiple local minima

ODE converted to a minimization problem to resemble the cost function. Consider a 2^{nd} order Boundary Value Problem (BVP) [3]:

$$D[f] \equiv \frac{d^2}{dx^2} f(x) + a\frac{df}{dx} + bf - c = 0 \tag{1}$$

$$f(x_0) = f_0$$

$$f(x_1) = f_1$$

Let the output of the NN be u. If we set —

$$\hat{u} = u_1 \left(\frac{x - x_0}{x_1 - x_0} \right) + u_0 \left(\frac{x - x_1}{x_0 - x_1} \right) + (x - x_0)(x - x_1)u \tag{2}$$

[3] Jivani, Vaidya, Bhattacharya and S.Saha; A Swarm Variant for the Schrodinger Solver; (IJCNN 2021)

$\hat{u}$ automatically satisfies the boundary conditions $u(x_0) = u_0$ and $u(x_1) = u_1$ by construction. NN optimizes the loss function $L = (D[\hat{u}])^2$ identically to 0, the differential equation solved;the boundary conditions obeyed.

Enter PSO: navigates the search space by embracing a *swarm* which is nothing but a population of particles. The swarm, guided by characteristic equations, attempt to converge to an optima [Ebenhart and Shi, 1995].

Velocity and the position update equations:

$$v_i^{(t+1)} = \phi v_i^{(t)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \quad (3)$$

$$x_i^{(t+1)} = x_i^t + v_i^{(t+1)} \quad (4)$$

where $\phi, c_1, c_2 \geq 0$. x_i^t suggests the position of particle i at time t, v_i^t: velocity of particle i at time t, p_i^{best}: particle-i best position; g_i^{best}: global best position of the swarm.

PSO Representation

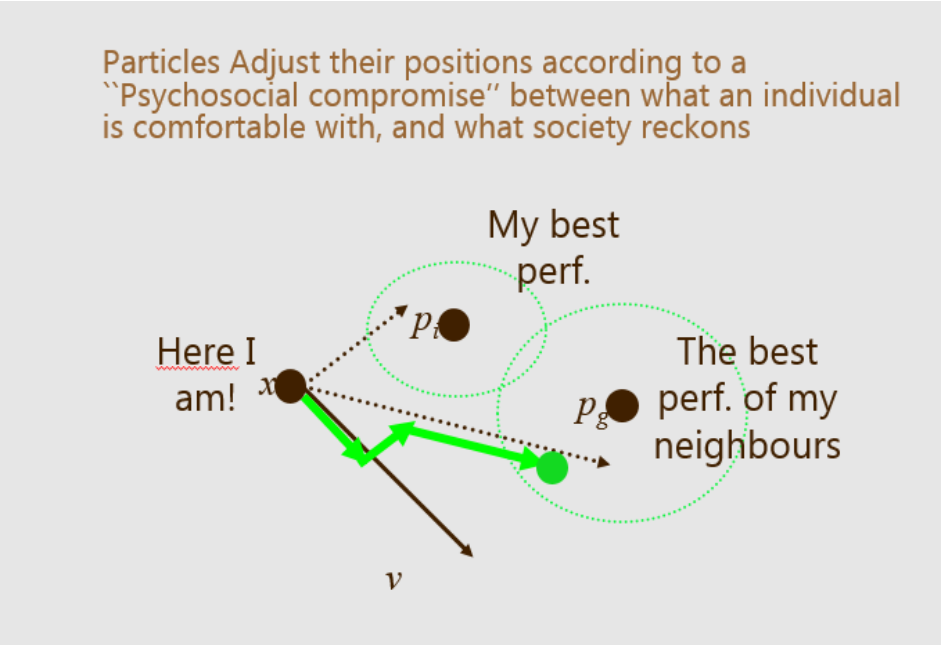


Figure: current velocity PLUS a weighted random portion in the direction of its personal best PLUS a weighted random portion in the direction of the neighbourhood best[5].

[4]Aishwarya, Raychoudhury, S. Saha, Kar, Anusha, CARE-Share: A Cooperative and Adaptive Ride Strategy for Distributed Taxi Ride Sharing, IEEE Transactions on Intelligent Transportation Systems (I.F: 6.45-ScimagoJR Q1); March 2021; DOI: 10.1109/TITS.2021.3066439

PSO with Momentum

Borrow from NN: momentum, we'll return the favors later

standard PSO equations embellished with the momentum term[6].

$$v_i^{(t+1)} = \left[v_i^{(t)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)})\right] \times (1-\lambda) + \lambda v_i^{t-1}$$

Here, λ denotes the momentum factor.

$$v_i^{(t+1)} = M_i^{(t+1)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \tag{5}$$

where,

$$M_i^{(t+1)} = \beta M_i^t + (1-\beta) v_i^{(t)} \tag{6}$$

[6]Saha et. al: EMPSO; https://arxiv.org/abs/2006.09875

EMPSO-baby steps

β: the momentum factor, M_i^t: momentum of a particle. (5)- (6) yields a new representation of the mathematical formulation of EMPSO.

$$v_i^{(t+1)} = \beta M_i^t + (1-\beta) v_i^{(t)} + c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)}) \quad (7)$$

$$\underbrace{v_i^{(t)}}_{\text{Exploration}} + \underbrace{c_1 r_1 (p_i^{best} - x_i^{(t)}) + c_2 r_2 (g^{best} - x_i^{(t)})}_{\text{Exploitation}}$$

The exploration phase, due to the new approach, is now determined by the **exponential weighted average of the historical velocities** only. The negligible weights in MPSO do not contribute to the required acceleration. The momentum in EMPSO is the exponential collection of velocities that the particles have seen so far, over time. We accumulate the velocities by multiplying by an exponential factor β, as we move ahead in time. Thus, β factor assigns more weight to the recent velocity than to older velocities.

The Intuition

recursively expand (6)...
$M_i^{(t)} = \beta M_i^{(t-1)} + (1-\beta) v_i^{(t-1)}$ which is expanded easily to
$M_i^{(t)} = \beta^2 M_i^{t-2} + \beta(1-\beta) v_i^{t-2} + (1-\beta) v_i^{t-1}$. Generalizing[7] —

$$M_i^{(t)} = \beta^n M_i^{(t-n)} + \beta^{(n-1)}(1-\beta) v_i^{(t-n)} + \\ \ldots + \beta(1-\beta) v_i^{(t-2)} + (1-\beta) v_i^{(t-1)} \quad (8)$$

Motivation: Any type of loss, ease of gradient computation, Use of Swarm velocities (near the minima) and Gradient dynamics together, control over the swarm population (independent of the number of neurons) i.e. computationally feasible.

[7] older velocities get a much smaller weight and, therefore, contribute less to the overall value of the Momentum.

What's Next?

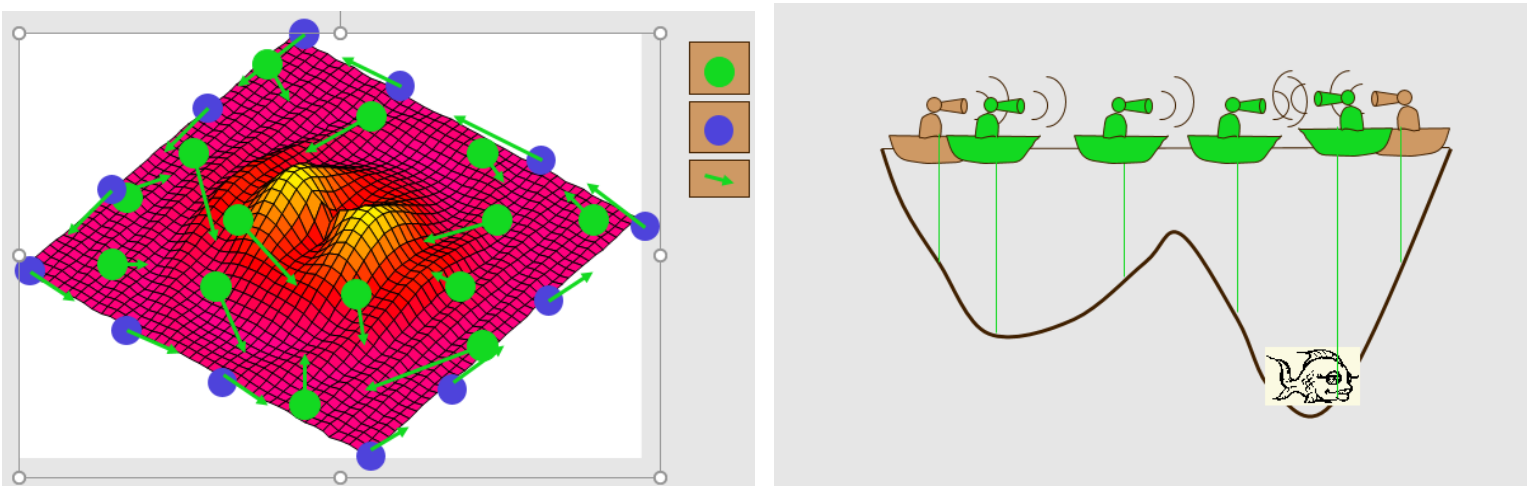


Figure: Co-operation Principle in PSO

- Particle-in-a-box problem solved: EMPSO is an optimizer
- Further inroads?
- Test Optimization problems: Some sample functions?
- Hyper-parameter tuning not encouraged. NOTE!

Some Visual Inspection

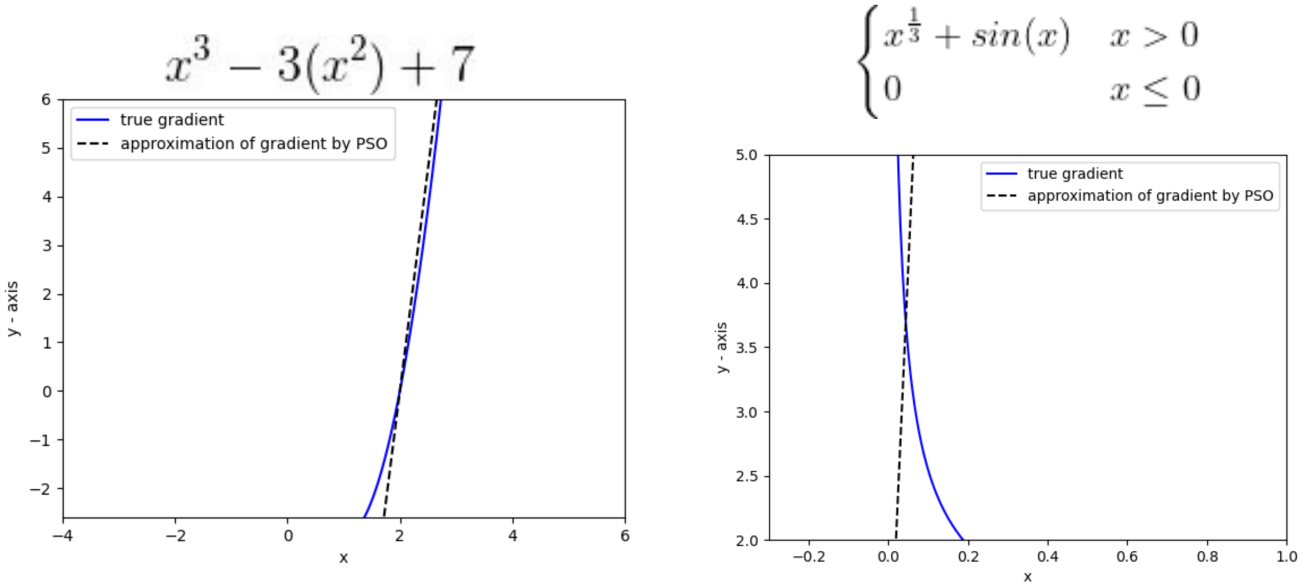


Figure: True Gradient *vs.* Approximate Gradient calculated using PSO
Parameters: The approximation accomplished: Diff and Non-Diff functions

Humble Beginnings: Gradient Approximated: How?

The Mind Map

EMPSO $\Longrightarrow$ Optimization on some sample functions $\Longrightarrow$ extend to 1-D optimization $\Longrightarrow$ empirical validation on some ODE/PDE solvers $\Longrightarrow$ Question: How is the gradient approximation? $\Longrightarrow$ Attempt derivative approximation via EMPSO; success $\Longrightarrow$: Theorem 1 on derivative approximation $\Longrightarrow$ Theorem 2 on Extend to Error Gradient approximation in NN: Deep and Shallow $\Longrightarrow$ Loss Agnostic? YES! $\Longrightarrow$: Applies to several loss functions: MAE, MSE, BCE/CE, Check Loss, Irregular Loss....NN Optimization: Borrowed momentum from NN, gave back to NN

From Sample test functions $\rightarrow$ to Loss functions: Gradient and Swarm Dynamics: Hunt in Pairs

The First Theorem

Theorem - Under reasonable assumptions, for an arbitrary objective function $f(w)$ (*differentiable or not*), if PSO converges at $w = g^{best}$, the gradient approximation theorem states that $\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(w - g^{best})$

Note: $w^{(t)}$: weight matrix of the NN, η: learning rate; f: loss function for the given network. weight update rule of GD —

$$w^{(t+1)} = w^{(t)} + \eta \frac{\partial f}{\partial w}|_{w^{(t)}} \tag{9}$$

Swarm and GD pair: At the timestep with weight $w^{(t)}$, substitute the Taylor-expansion of the derivative in the GD update

multidimensional objective: $f(w)$, centering its Taylor-expansion about $w =$ **w'** —

$$\frac{\partial f}{\partial w} = f'(w') + f''(w')(w - w') + ... + E_{n-1}(w; w') \tag{10}$$

Key Points

- $f' = \nabla_w f$ and $f'' = \frac{\partial^2 f}{\partial w_i \partial w_j}$ with w_i ranging over the dimensions of the weights (slight abuse of notations)
- $f'(w') = 0$ (local minima)
- The purpose of GD is to find the minimum of an objective. However, the same minima could be found by PSO if we let $x \equiv w$ i.e., the particle dimensions are that of the weights of the NN.

The PSO position update equation —

$$\begin{aligned} & x^{(t+1)} - x^{(t)} \\ & = v^{(t+1)} \\ & = \omega v^{(t)} + c_1 r_1 (p^{best} - x^{(t)}) + c_2 r_2 (g^{best} - x^{(t)}) \end{aligned} \tag{11}$$

- $v^{(t)} \approx 0$; $p^{best} \approx g^{best}$ near convergence;$g^{best} = w'$ after convergence

Key Assumptions

1. $w^{(t)}$ is inside a small, $\delta-$neighborhood centred about w'
2. $\eta = \omega$ for mathematical convenience (*not required for the minima condition, near or far from the minima in a NN setting,* $f'() = 0$ *irrespective of this setting*)
3. $x \equiv w$ i.e., x, w are used interchangeably; x in PSO is used to update particle position while w in a neural-network setting, is used to update weights.

$$\eta f''(w')(w^{(t)} - w') = c_1 r_1 (p^{best} - x^{(t)}) + c_2 r_2 (g^{best} - x^{(t)})$$

$$\implies f''(w')(w^{(t)} - w') = -\frac{(c_1 r_1 + c_2 r_2)(w^{(t)} - g^{best})}{\eta}$$

$p^{best} \approx g^{best}$, $x \equiv w$ and $g^{best} \approx w' \implies : f''(w') = -\frac{(c_1 r_1 + c_2 r_2)}{\eta}$; Some Algebra and....gradient equivalence theorem —
$\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(w - g^{best})$

AdaSwarm[8]: Equivalence Theorems

Theorem 1: (The GD version): $\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(w - g^{best})$

SGD with Momentum and EMPSO for Differentiable (loss) functions

Theorem 2: (The SGD version): Under identical assumptions, the following equivalence holds: $\eta = (1 - \beta)$, $f''(\mathbf{w'}) = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}$ and $\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(w - g^{best})$

Non-Diff Loss: a continuous, bounded function $f(x)$ BUT non-differentiable. $f(x + \alpha)$, where α is a shift parameter with an initial value equal to zero. Define $z \equiv x + \alpha$, so that $f(z) \equiv f(x + \alpha)$.

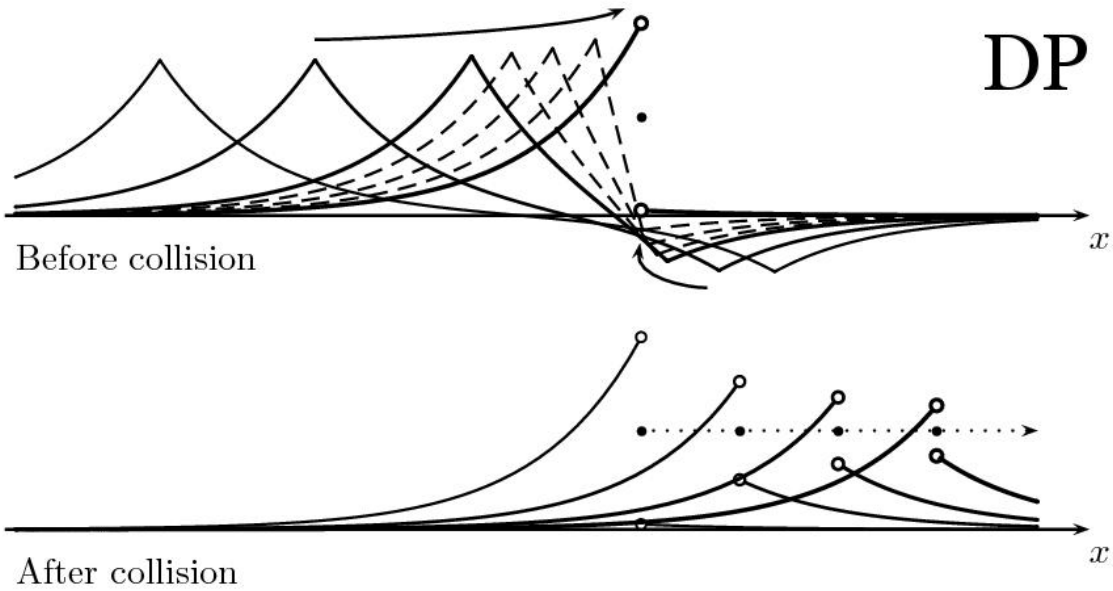
Theorem 3: $\frac{\partial f}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(z - g^{best})$

[8] Mohapatra, Saha, Coello, Bhattacharya, Dhavala and Saha; AdaSwarm: Augmenting gradient-based optimizers in Deep Learning with Swarm Intelligence, the IEEE Transactions on Emerging Topics in Computational Intelligence (Accepted, 2021)

Where does this take us?

Let's back-pedal to another life: Shallow water waves

$$\underbrace{m_t}_{\text{Evolution}} + \underbrace{c\left(m_x + \frac{1}{6}h^2 u_{xxx}\right)}_{\text{Dispersion}} + \underbrace{(um_x + bu_x m)}_{\text{Nonlinearity}} = 0.$$



Degasperis-Procesi peakon-antipeakon collision.

- Some used in classification: MAE, Check Loss, Quantile Loss
- Laminar Flows (2)
- Singular solutions to PDEs (different from Particle in a box type) (3)
- Periodic Solutions to ODEs/PDEs (4)

Standard activations don't learn (2)-(4) well! Loss is not well-defined, difficult to handle Loss functions

Some representation which can handle this: diff losses: AdaSwarm! Apprx $\frac{\partial E}{\partial y}$

neural update rule (AdaSwarm):

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \times D(activation) \times x \tag{12}$$

replace $\frac{\partial E}{\partial y}$ [Eq. 12] by an approximate value of the gradient;
weight update rule in NNs:

$$\frac{\partial E}{\partial w} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y) \times D(activation) \times x$$

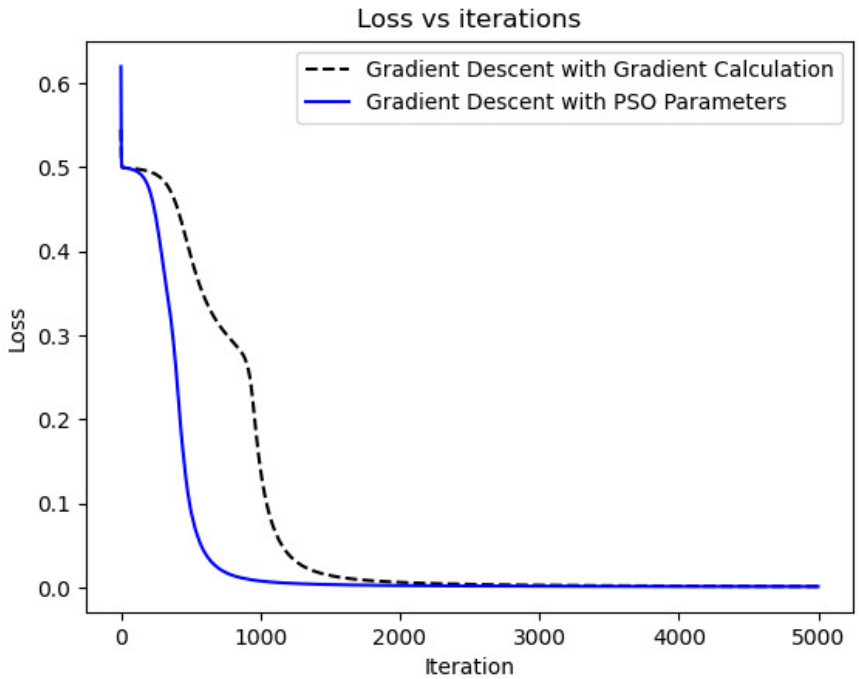


Figure: Error Gradient approximation in backpropagation by EMPSO on simulated

Code Snippets[9]

$staticmethoddefbackward(ctx, grad_{output}) :; yy, yy\_pred = ctx.saved_{tensors}; sum_{cr} = ctx.sum_{cr}; eta = ctx.eta; grad_{input} = torch.neg((sum_{cr}/eta) * (ctx.gbest - yy)); returngrad_{input}, grad_{output}, None, None, None$

https://github.com/rohanmohapatra/pytorch-cifar/blob/master/cifar10/main_adaswarm.py
this where all things are coming together

SGD and variants: find the gradients of the loss and use it in the GD equation, AdaSwarm: actually approximating the gradients, NOT computing df/dw.

[9] https://github.com/rohanmohapatra/pytorch-cifar/blob/master/nn_utils.py

More Intuition and Motivation

Several places.....

- S. M. Mikki and A. A. Kishk, Particle Swarm Optimizaton: A Physics- Based Approach. Morgan Claypool, 2008.
- M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," In Proceedings of the 28th International Conference on Machine Learning, p. 681–688, 2011.
- Y.-A. Ma, Y. Chen, C. Jin, N. Flammarion, and M. I. Jordan, "Sampling can be faster than optimization," Proceedings of the National Academy of Sciences, vol. 116, no. 42, pp. 20 881–20 885, 2019. [Online]. Available: https://www.pnas.org/content/116/42/20881
- S. Yokoi and I. Sato, "Bayesian interpretation of sgd as ito process," ArXiv, vol. abs/1911.09011, 2019.

Recap

So, what's AdaSwarm

* a theoretical equivalence to Error gradient computation in Neural Nets (Deep and Shallow)
* PSO integrated with the backpropagation dynamics of Deep NNs.
*The outcome: beating all available SOTA Optimizers: GD/SGD, Adam, Adagrad, AMSGrad, AdaDelta, DiffGrad.
* Started with A novel Exponentially weighted Momentum Particle Swarm Optimizer (EMPSO)
* The ability of AdaSwarm to tackle optimization problems
* We show that the gradient of any function, differentiable or not, can be approximated by using the parameters of EMPSO.

Performance Benchmarking: test functions, 1-d test optimization problems, classification on simulated data, classification on benchmark data (15 non-CV, 5 CV)

Limitations

- Still line-search
- Approximation not good (away from the minima)
- first order

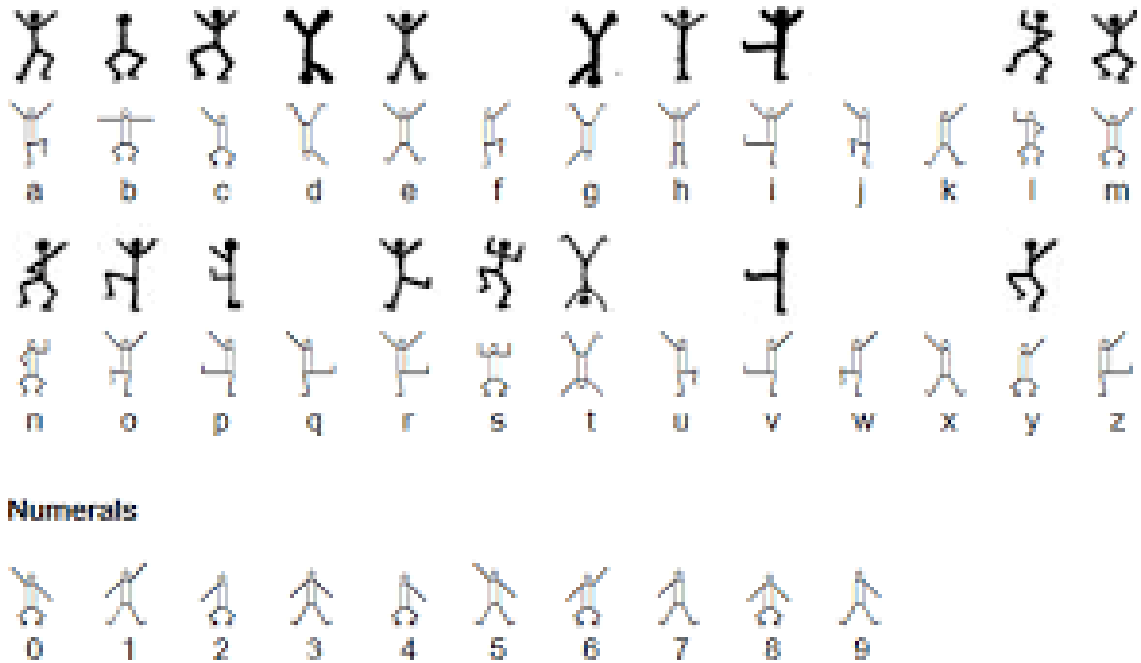
Trust-Region Policy based Optimization?
What's the way forward for AdaSwarm?..higher order embedding.....Some important references in this context

1) https://www.sciencedirect.com/science/article/pii/S0377042709003598
2) https://link.springer.com/chapter/10.1007%2F978-0-387-40065-5_4
3) http://diposit.ub.edu/dspace/bitstream/2445/52216/1/635635.pdf
4) https://optimization.mccormick.northwestern.edu/index.php/Trust-region_methods

Subtlety?.....

Subtlety?.....

What we want to do.....



topology/trajectory.

SBAF: Forward Pass (FP)

- We explore the efficacy of using a novel activation function in Artificial Neural Networks (ANN) in characterizing exoplanets
- We call this Saha-Bora Activation Function (SBAF) as the motivation - The function is demonstrated to possess nice analytical properties doesn't seem to suffer from local oscillation problems.
- Neural networks, commonly known as Artificial Neural network(ANN), is a system of interconnected units organized in layers, which processes information signals by responding dynamically to inputs. Layers of the network are oriented in such a way that inputs are fed at input layer and output layer receives output after being processed at neurons of one or more hidden layers.

$$y = \frac{1}{1 + kx^{\alpha}(1-x)^{1-\alpha}}; \tag{13}$$

$$\frac{dy}{dx} = \frac{y(1-y)}{x(1-x)} \cdot (\alpha - x) \tag{14}$$

Existence of Optima: Second order Differentiation of SBAF for Neural Network

$$\Rightarrow \frac{d^2y}{dx^2} = \frac{y(1-y)}{x(1-x)}$$

The first derivative vanishes when $\alpha = x$, the second derivative is positive when $\alpha > x$ and is negative when $\alpha < x$

Visualizing SBAF

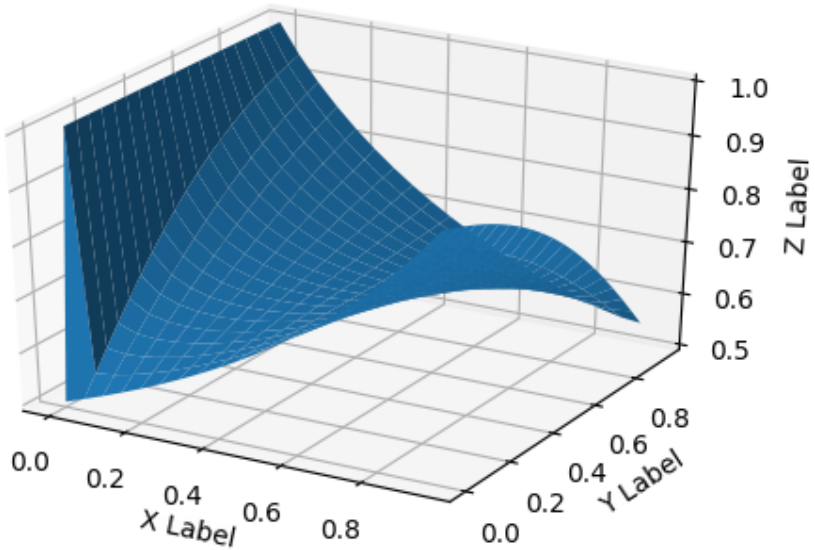


Figure: Surface Plot of SBAF

Key Takeaways from SBAF

- Bingo! Remarkable performance
- Other activation functions are special cases of SBAF-namely **Sigmoid** and **ReLU**
- Unique Optima
- Production Function perspective- K can't be negative! This is useful for training and tuning the neural network.

NN as approximator; some unknown f approximated by weighted combination of inputs and biases, operated by the Activation function $\sigma()$; Cybenko (1989): $\sigma()$ == SBAF satisfies this UAT!

The Points of interest

Evolution of Novel Activation Functions in Neural Network Training: A Production function inspired approach: Auto tuning neural networks!

Outcome:

- Define "SBAF", an activation function and relate to production functions; draw similarities with other activations
- Discuss the evolution and impact of SBAF
- Parameter tuning or the lack of it- an efficient training pipeline?
- Qualitative aspects and offspring of SBAF- A-ReLU!

What's up with an ODE?

Let's consider the first order differential equation

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{y(1-y)}{x(1-x)} \cdot (x - \alpha)$$

which is a representation of the standard form:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y)$$

The solution ?? $\rightarrow$ *SBAF*
defined with parameters k, α (these will be estimated subsequently and no effort is spent on tuning these parameters while training) and input x (Data in the input layer), produces output y as follows: $y = \frac{1}{1+kx^{\alpha}(1-x)^{1-\alpha}}$; *Rings a bell?*

- **COBB-Douglas!**
- $k > 0$ but how to restrict?
- $0 \leq \alpha \leq 1; \alpha + 1 - \alpha = 1 \rightarrow$ CRS!
- $0 < x < 1$ Normalized input–*Standard practice*

Taming the bandits: k, α for FP

- Contraction Mapping: Let (X, d) be a Banach space equipped with a distance function d. There exists a transformation, T such that $T : X \to X$ is a contraction, if there is a guaranteed $q < 1, q$ may be zero which squashes the distance between successive transformations (maps). In other words, $d(T(x), T(y)) < qd(x, y) \quad \forall x, y \in X$. q is called Lipschitz constant and determines speed of convergence of iterative methods.
- Fixed point: A fixed point x_* of a function $f : \mathbb{R} \to \mathbb{R}$ is a value that is unchanged by repeated applications of the function, i.e., $f(x_*) = x_*$. Banach space endowed with a contraction mapping admits of a unique fixed point. Note, for any transformation on Turing Machines, there will always exist Turing Machines unchanged by the transformation.
- Stability: Consider a continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$ with a fixed point x_*, $f(x_*) = x_*$. The fixed point x_* is defined to be *stable* if $f'(x_*) < 1$. If $f'(x_*) > 1$, then x_* is defined as *unstable*.
- SBAF optima: exists, unlike sigmoid!

The bandits: k, α for SBAF

Definition

Contours: 2-D representation of a 3-D surface: provides valuable insights to changes in y as input variables $x\&1-x$ change. **First Return Map:** Iterative map on the set S, $f : S \to S$; initial value $x_0 \in S$, iterate the map f to yield a trajectory: $x_i = f(x_{i-1})$. Plot of x_{n+1} *vs.* x_n—FRM $n > 0$, $n \in Z$. **Fixed point theory/contraction map:** ODE has a solution, y; unique for fixed choices of k, α. Optima: $x = \alpha$; global optima (Hessians). Optima:$x = 0.5$...matches the FP Map!

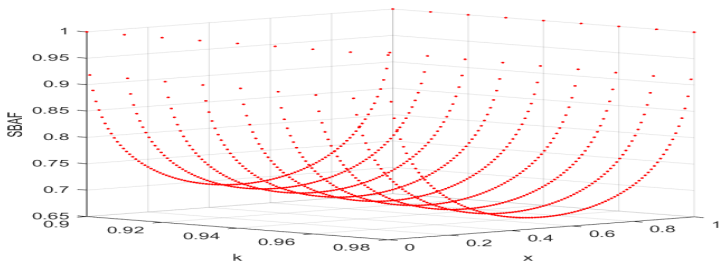


Figure: Plot of SBAF function: a minima for all values of $k(0.90-0.98)$ at $x = 0.5$; confirmed stable fixed point at $k = 0.98, \alpha = 0.5$(empirical) (mimima of SBAF: $x = \alpha \implies \alpha = 0.5$). k, α values used in SBAF to train the classifier.

Note: Optimization trick

$\frac{dy}{dx} = \frac{y(1-y)}{x(1-x)} \cdot (x - \alpha)$.

when $\alpha = x$,

$$\Rightarrow \frac{d^2y}{dx^2} = \frac{y(1-y)}{x(1-x)} \tag{15}$$

Clearly, the first derivative vanishes when $\alpha = x$, and the derivative is positive when $\alpha < x$ and is negative when $\alpha > x$ (implying range of values for α so that the function becomes increasing or decreasing). Hence, $\frac{d^2y}{dx^2} > 0$ ensuring optima of y– **WHY?**

RECALL: **Sigmoid:saddle point/squashing activation–FLAT MSE; SBAF has minima— diminishing MSE with iterations!**

k to conquer next!

$y = \frac{1}{1+kx^{\alpha}(1-x)^{1-\alpha}}; 0 < x < 1, 0 \leq \alpha \leq 1, k > 0$; Why restrict k? Let's think about $kx^{\alpha}(1-x)^{1-\alpha}$ and the non-linear X-OR problem. $k < 0 \implies$ YOU borrow from market and produce nothing... $\implies$
the separation width will shrink so much the classification is incorrect!

Production function gives us nice activation unit to work with a neural net and justifies some lower bound on k used to define SBAF, solution to the ODE with k, the constant of integration.

- Paradox Solved? Not yet!
- Is negative k invalidated from some other theory?

Enter Chaos theory: SBAF

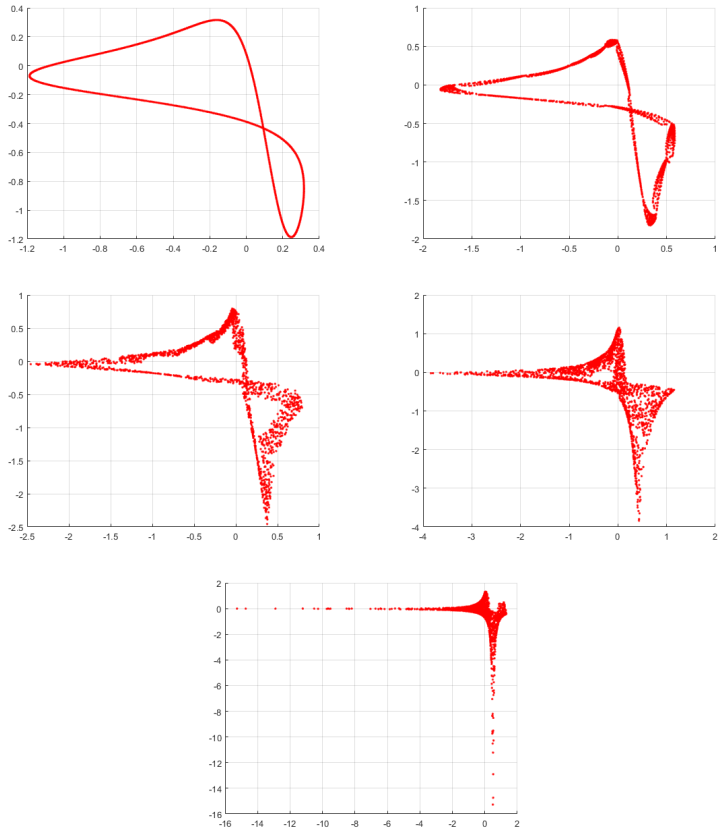


Figure: First return maps of SBAF for $\alpha = 0.5$ and various values of $k < 0$ (top to bottom, left to right): $\{-2.0, -1.97, -1.90, -1.75, -1.5, -1.0\}$ indicating absence of a fixed point. The plot confirms absence of fixed points for all values of $k \leq 0$.

Chaos is beautiful! SBAF has fixed point

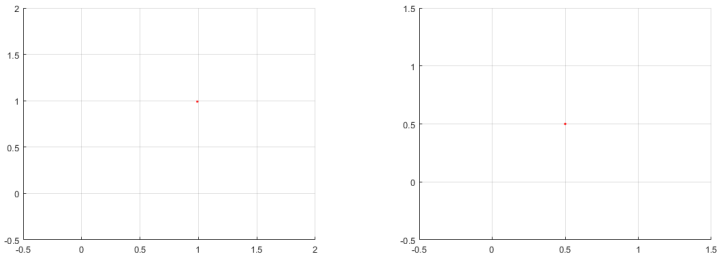


Figure: First return maps of SBAF for $\alpha = 0.5$ and various values of $k > 0$: $\{0.1, 2.0\}$ indicating presence of a fixed point.

Explicitly compute the values of k for which there exists a stable fixed point when $\alpha = 0.5$: $kx^{*(0.5+1)}(1 - x^*)^{-0.5} = 1$ implies $k = \frac{\sqrt{(1-x^*)}}{x^*\sqrt{x^*}}$.
Seek fixed point x^*; $0 < x^* < 1$, $x^* = SBAF(x^*)$, and require for stability: $|SBAF'(x^*)| < 1$. Ex: if the stable fixed point is $x^* = \frac{1}{\sqrt{2}}$ then $k = 0.91018$. When $0 < x^* < 1$, the value of $SBAF'(x^*)$ varies from -0.5 to 0.5 (thus $|SBAF'(x^*)| < 1$, making it a stable fixed point).

Stability, FP and the bandit, k: Optimization tricks

- Economic theory is consistent with our assertion that the choices of parameters for optimal classification performance are not accidental!
- the first return maps of SBAF reveal that no stable fixed point exists when $k < 0$.
- We found the *suitable boy* (borrowing from Vikram Seth) $k = 0.9$ and a small neighborhood around it..... implies not a **bandit** anymore!.

Progression in parameter handling: from tuning to selection

What's the big deal? Immensely cheaper computation! remarkable precision!!

Optimization: Lip-Acceleration in DNN

Devil's in the details: lr :: tune the bandits??

Optimise weights and biases by using back propagation on SBAF::
Initialize all weights w_{ij}**, biases** b_i**,** n_epochs**, lr, k and** α**;**

The forward Pass: Use appropriate function approximation

- ODE theory to the rescue.....for k, α

Adaptive learning to the rescue.....for **lr** specific loss

Non-Specific Loss: Loss Agnostic Minimization?
$\frac{\partial E}{\partial w_{ji}} = \frac{-(c_1 r_1 + c_2 r_2)}{\eta}(g^{best} - y_j)\frac{\partial y_j}{\partial net_j}x_{ji}$ EMPSO Appx: TODAY!

Loss Function in BP

"lr", the learning rate in weight update, recall?

back propagation & gradient descent: the squared loss function; Gradient Descent update Rule: REVISIT $\mathbf{w} := \mathbf{w} - \alpha \cdot \nabla_{\mathbf{w}} f$

- Use Lipschitz Constant on the squared loss function
- Use MVT
- **Minimal assumption:** functions are Lipschitz continuous and differentiable up to first order only [a]
- $\alpha = \frac{1}{L}$, we force $\Delta \mathbf{w} \leq 1$, constraining the change in the weights.

[a] Note this is a weaker condition than assuming the gradient of the function being Lipschitz continuous. We exploit merely the boundedness of the gradient.

What is it?

For a function, the Lipschitz constant is the least positive constant L such that $\|f(\mathbf{w}_1) - f(\mathbf{w}_2)\| \leq L \|\mathbf{w}_1 - \mathbf{w}_2\|$

$$\begin{aligned} f(\mathbf{w}_1) - f(\mathbf{w}_2) &= \nabla_{\mathbf{w}} f(\mathbf{v}) \mathbf{w}_1 - \mathbf{w}_2 \\ &\leq \sup_{\mathbf{v}} \nabla_{\mathbf{w}} f(\mathbf{v}) \mathbf{w}_1 - \mathbf{w}_2 \end{aligned}$$

Thus, $\sup_{\mathbf{v}} \nabla_{\mathbf{w}} f(\mathbf{v})$ is such an L. Since L is the least such constant, $L \leq \sup_{\mathbf{v}} \nabla_{\mathbf{w}} f(\mathbf{v})$. We use $\max \nabla_{\mathbf{w}} f$ to derive the Lipschitz constants.

LALR: knowing the "lr"

Minimal assumption: functions are Lipschitz continuous and differentiable up to first order only [a]

[a] Note this is a weaker condition than assuming the gradient of the function being Lipschitz continuous. We exploit merely the boundedness of the gradient.

$\alpha = \frac{1}{L}$, we force $\Delta \mathbf{w} \leq 1$, constraining the change in the weights. Stress: Not computing the Lipschitz constants of the *gradients* of the loss functions, but of the losses themselves. Assume: the loss is L-Lipschitz. Argument: set the learning rate to the reciprocal of the Lipschitz constant. Claim: supported by our experimental results.

Let the loss be given by $E(\mathrm{a}^{[L]}) = \frac{1}{2m}\left(\mathrm{a}^{[L]} - \mathrm{y}\right)^2$ *where the vectors contain the values for each training example.*

Note: Chain Rule in GD

$$\frac{\partial E}{\partial w_{ij}^{[L]}} = \frac{\partial E}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot \frac{\partial z_j^{[L]}}{\partial w_{ij}^{[L]}}$$
$$= \frac{\partial E}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot a_j^{[L-1]}$$

This gives us $\max_{i,j} \frac{\partial E}{\partial w_{ij}^{[L]}} = \max_j \frac{\partial E}{\partial a_j^{[L]}} \cdot \max_j \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} \cdot \max_j a_j^{[L-1]}$. The third part cannot be analytically computed; we denote it as K_z.

The formula!

Using the results above and other "tricks", we obtain:

$\max_{i,j} \frac{\partial E}{\partial w_{ij}^{[L]}} = \frac{K}{m}\mathbf{X}^T\mathbf{X} - \frac{1}{m}\mathbf{y}^T\mathbf{X}$; K is the upper bound on the weight vectors

Binary Classification-binary cross entropy loss: $\boxed{L = \frac{1}{2m}\mathbf{X}}$

Multiclass Classification-softmax regression loss: $\boxed{L = \frac{k-1}{km}\mathbf{X}}$

- L is lipschitz constant; **lr**, learning rate $= \frac{1}{L}$

Such choice of Learning Rate?

Assumption: Gradients cannot change arbitrarily fast

A typical Gradient Descent algorithm is in the form of

$$\boldsymbol{w^{k+1}} = \boldsymbol{w^k} - \eta_k \nabla_w f \tag{16}$$

for $k \in \mathbb{Z}$ and stop if $\|\nabla_w f\| \leq \epsilon$
$\forall v, w, \exists L$ such that $\|f(\mathbf{v}) - f(\mathbf{w})\| \leq L \|\mathbf{v} - \mathbf{w}\|$. Also, $\nabla^2 f(w) \leq LI$

$$\boldsymbol{f(v)} = \boldsymbol{f(w)} + \nabla \boldsymbol{f(w)^T(v-w)} + \frac{1}{2}(\boldsymbol{v-w})^T \nabla^2 \boldsymbol{f(w)(v-w)}$$

$$\boldsymbol{f(v)} \leq \boldsymbol{f(w)} + \nabla \boldsymbol{f(w)^T(v-w)} + \frac{L}{2}\|\boldsymbol{v-w}\|^2 \tag{17}$$

- a convex quadratic upper bound which can be minimized using gradient descent with the learning rate $\eta_k = \frac{1}{L}$

Convergence

It follows, $\boldsymbol{f}(\boldsymbol{w}^{\boldsymbol{k}+1}) \leq \boldsymbol{f}(\boldsymbol{w}^{\boldsymbol{k}}) - \frac{1}{2L}\left\|\nabla \boldsymbol{f}(\boldsymbol{w}^{\boldsymbol{k}})\right\|^2$

Gradient Descent decreases $\boldsymbol{f}(\boldsymbol{w})$ if $\eta_k = \frac{1}{L}$ and $\eta_k < \frac{2}{L}$. This proof also enables one to derive the rate of convergence with Lipschitz adaptive learning rate.

Let n represent the number of iterations. We know that,

$$\boldsymbol{f}(\boldsymbol{w}^{\boldsymbol{k}+1}) \leq \boldsymbol{f}(\boldsymbol{w}^{\boldsymbol{k}}) - \frac{1}{2L}\left\|\nabla \boldsymbol{f}(\boldsymbol{w}^{\boldsymbol{k}})\right\|^2$$

Several steps after.... $n \geq \frac{2L(\boldsymbol{f}(\boldsymbol{w}^0) - \boldsymbol{f}(\boldsymbol{w}^*))}{\epsilon}$

Gradient Descent requires, at least, $n = O(\frac{1}{\epsilon})$ iterations to achieve the error bound: $\left\|\nabla f(w^k)\right\| \leq \epsilon$

Loss Function, NOT Activation

- For a neural network that uses the sigmoid, ReLU, or softmax activations, it is easily shown that the gradients get smaller towards the earlier layers in backpropagation. Because of this, the gradients at the last layer are the maximum among all the gradients computed during backpropagation.
- Our framework is independent of activation functions
- the framework is extensible to all loss function satisfying the *lipschitz condition*
- Thus, we set up a pipeline for classification problems-SYMNet.

- Economic theory is consistent with Our models: A new SVM Kernel and NN training (Activation functions derived from Lip Conditions on 1st Order DEs)
- We found the *suitable boy* (borrowing from Vikram Seth) LR $= 1/L$ and ..... implies not a **bandit** anymore!.

Progression in parameter handling: from tuning to selection

What's the big deal? Immensely cheaper computation! remarkable prediction performance!!

Recap: The beast conquered!

- Theoretical framework for computing an adaptive learning rate; this is also "adaptive" with respect to the data.
- "large" learning rates may not be harmful as once thought; remedy: guarded value of L_2 weight decay.

What did we gain?

- Novel Approximation functions used during forward pass with "little effort in parameter tuning, k, α; OUTCOME: Accuracy beating state-of-the-art
- Loss Function manipulation to devise learning rate formula; OUTCOME: NO Need to "handcraft" **lr**; performance as good as state of the art
- Metric gains: Time to converge, # of iterations to converge, CPU/RAM utilization

A pipeline with SBAF in forward pass + Adaptive **lr** in GD back-prop: SYMNET: Released v.1.0 ..https://github.com/sahamath/sym-netv1

Takeaway

So what have these new kids on the block done?

- saves number of iterations to converge
- accuracy, precision, recall and other performance metrics are better, at least on the data sets applied so far (13 different public data sets)
- Improvement in CPU and RAM utilization
- non-negligible improvement in time complexity

Parsimonious computing; *something I always wanted to do*

The classics

- Vladimir Vapnik, Estimation of Dependences Based on Empirical Data
- Vapnik et. al, Support Vector Regression Machines
- Ying Tan, and Jun Wang, Support Vector Machine with a Hybrid Kernel and Minimal Vapnik-Chervonenkis Dimension
- Mikhail Belkin et. al, Reconciling modern machine learning practice and the bias-variance trade-off
- VAPNIK et. al, Support-Vector Networks
- J. Mercer, Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations
- Vapnik et. al, On a class of perceptrons
- Sebastian Bock et. al, An improvement of the convergence proof of the ADAM-Optimizer
- ZUOWEI SHEN et. al, NONLINEAR APPROXIMATION VIA COMPOSITIONS

Publications to read

- Hrushikesh Mhaskar et. al, When and Why Are Deep Networks Better than Shallow Ones?
- Preetum Nakkiran et. al, Compressing Deep Neural Networks using a Rank-Constrained Topology

Classics in DL: My favorites

Langevian Dynamics-PSO
Newton Flows-
Du, Singh-Overpa ARELU
Belkin, Double-descent
Stochastic DE-Arora
Chaotic Continual learning

Robert Gallagher's take on Shannon's style of research:
http://www.ifp.illinois.edu/ tieliu/Shannon.html

India's first Data Scientists

Contribution:

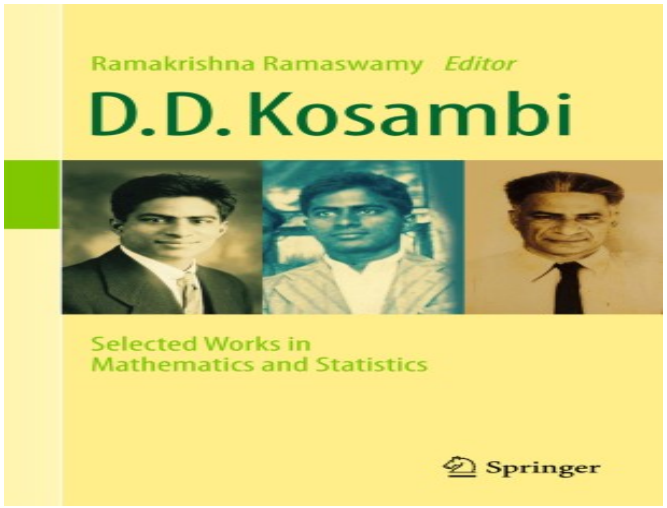


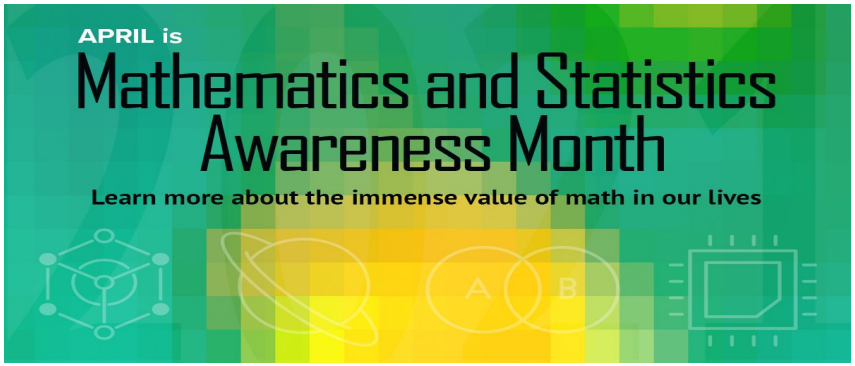
Figure: Prof. P.C Mahalanobis and Prof. D.D. Kosambi

Application

Removal of Statistical bias while comparing different dimensions, clustering: Customer segmentation in Retail business; Kosambi Map function, KL expansion-proper orthogonal decomposition (POD)

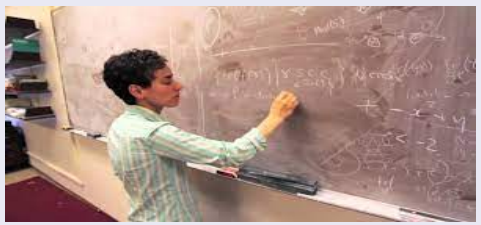
April 2021

Mathematics and Statistics Awareness Month:



From Kolmogorov (1957)-Cybenko (1989) via Tikhonov-Leshno (1993) to Sonoda (2015), Pure Mathematics (Russians and Jews) has been driving modern Machine Learning.

A Turing-like Mind: Maryam Mirzakhani



My Comrades in this struggle

Soma.D (Wadhani AI)

Nithin (NIAS)

Carlos Coello (UNSW)

Archana (NMIT)

Sriparna (IIT Patna)

C.Bajaj (UT Austin)

THANK YOU!

Non-convex Example

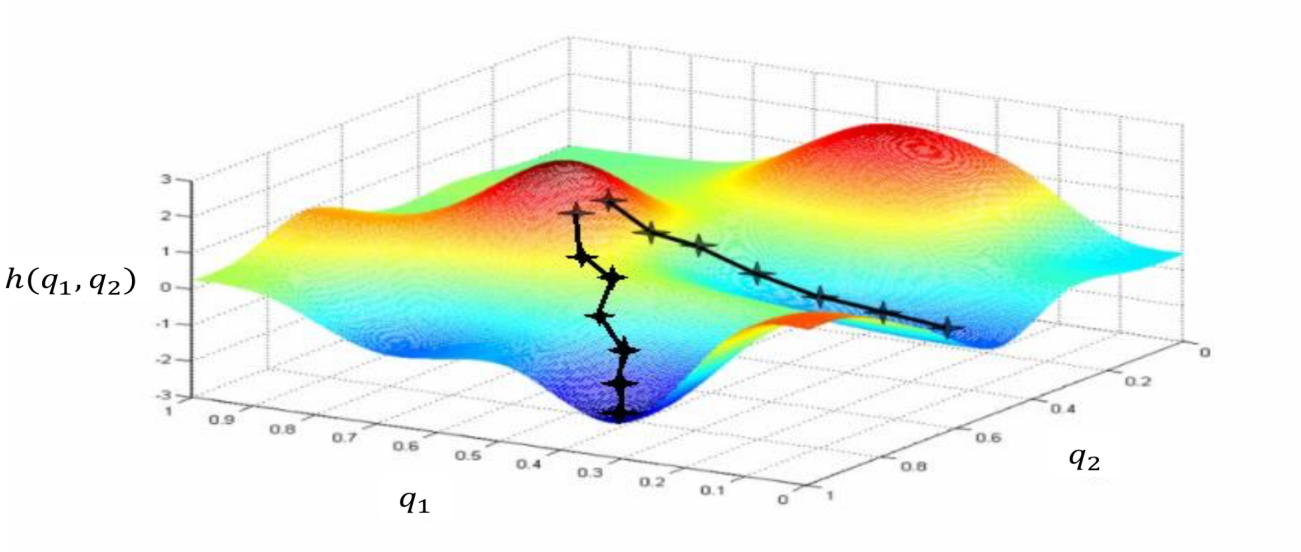


Figure: Do better than heuristics!

I gratefully acknowledge multiple online sources for images used in the presentation.