Mathematics of AI

Optimisation in deep learning

Goodfellow et al

Deep Mysteries of Deep Learning

- Why does it converge at all?
- Why does it work well?
  - non-convex optimisation
  - many local minima
  - generalises to unseen data
- How does regularisation happen?
- Why do high dimensions seem to help?
  - segue into project…

Solving XOR

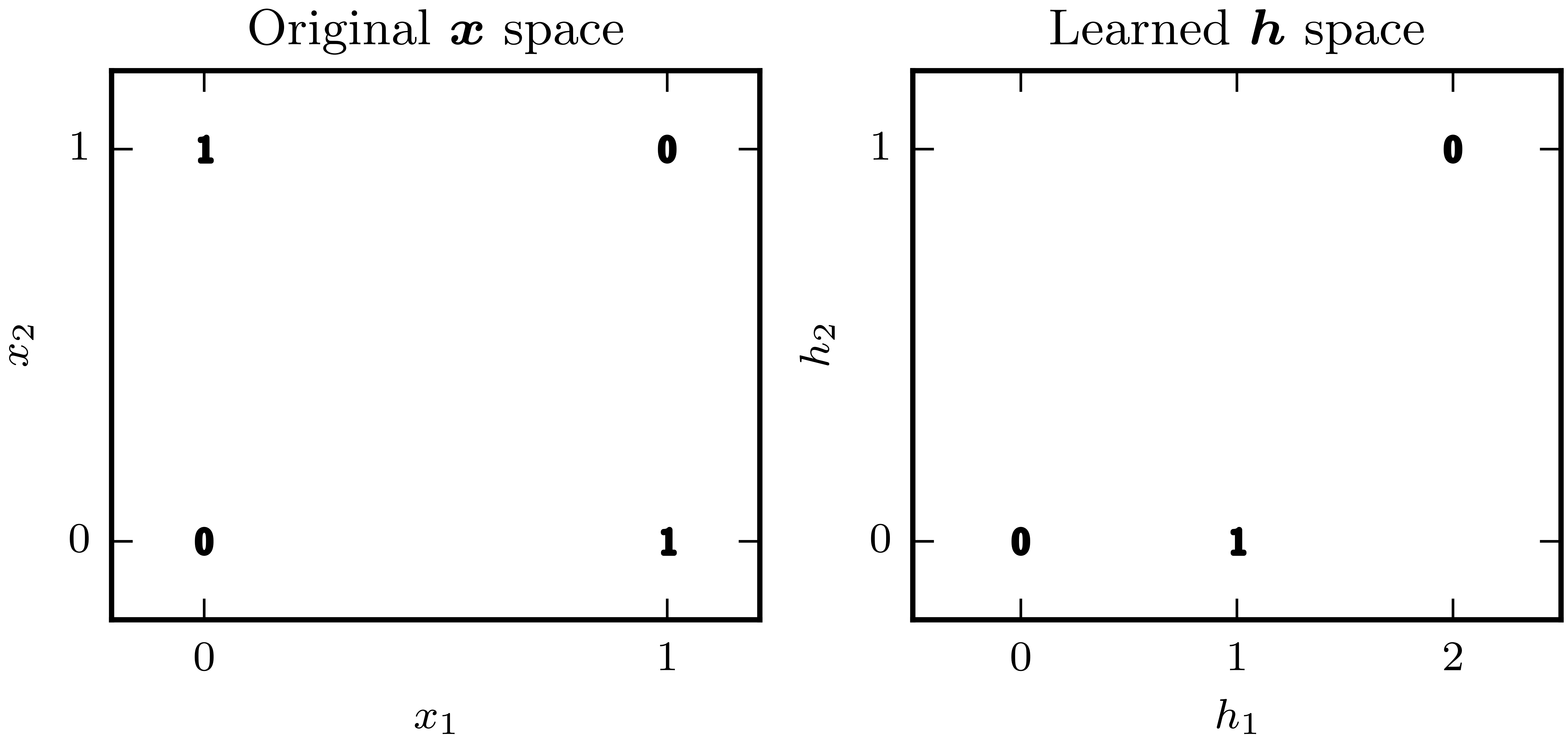


Figure 6.1

(Goodfellow 2017)

High-dimension non-convex cost functions

What type of critical point is most common?

- Global minima?
- Local minima?
- Saddle points?
- Local maxima?
- Global maxima?

(What does the distribution of eigenvalues of a large, random Hessian look like?)

Gradient Descent and the Structure of Neural Network Cost Functions

presentation by Ian Goodfellow

adapted for www.deeplearningbook.org
from a presentation to the
CIFAR Deep Learning summer school on August 9, 2015

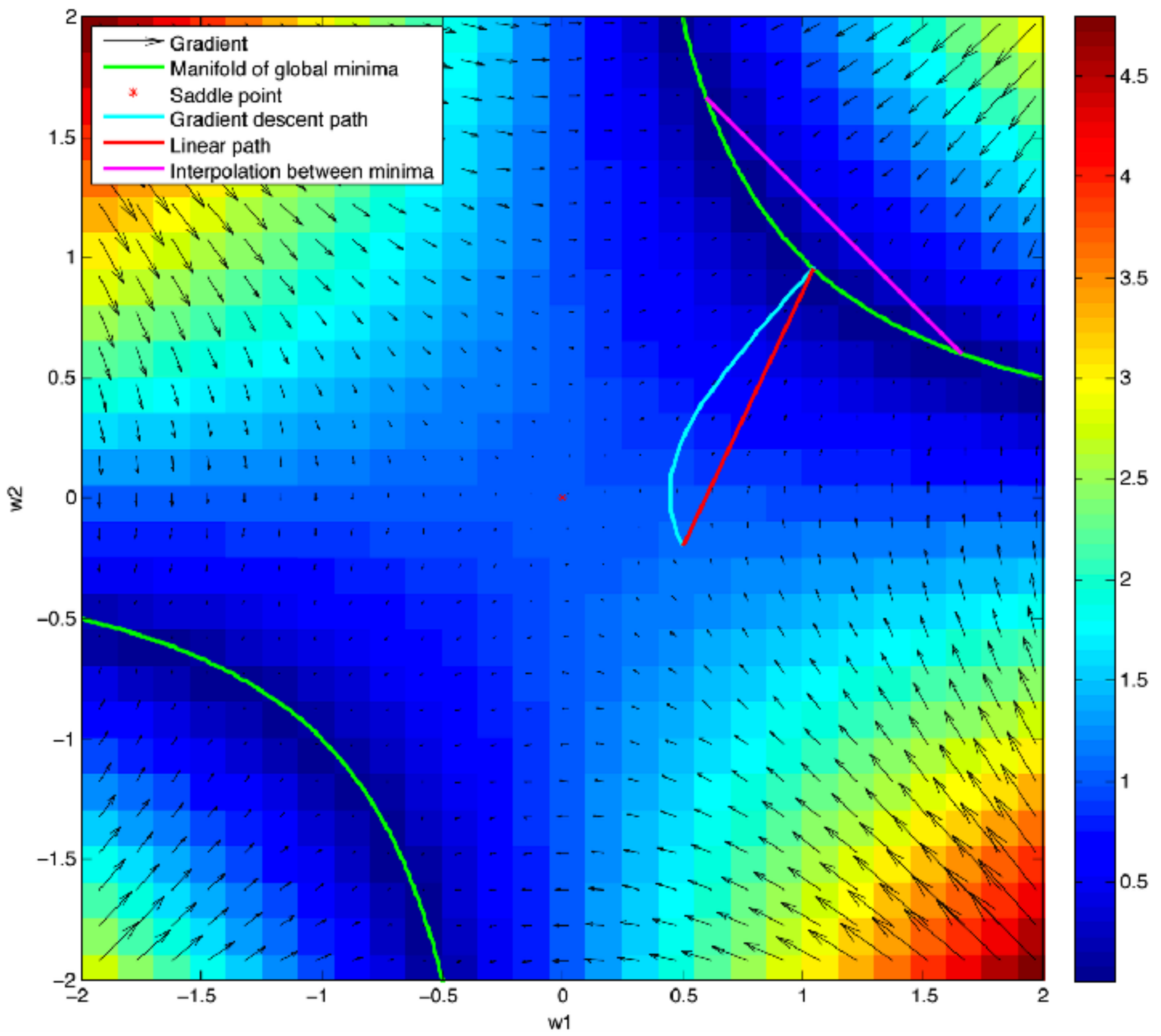
Are saddle points or local minima more common?

- Imagine for each eigenvalue, you flip a coin
- If heads, the eigenvalue is positive, if tails, negative
- Need to get all heads to have a minimum
- Higher dimensions -> exponentially less likely to get all heads
- Random matrix theory:
- The coin is weighted; the lower J is, the more likely to be heads
- So most local minima have low J!
- Most critical points with high J are saddle points!

(Goodfellow 2015)

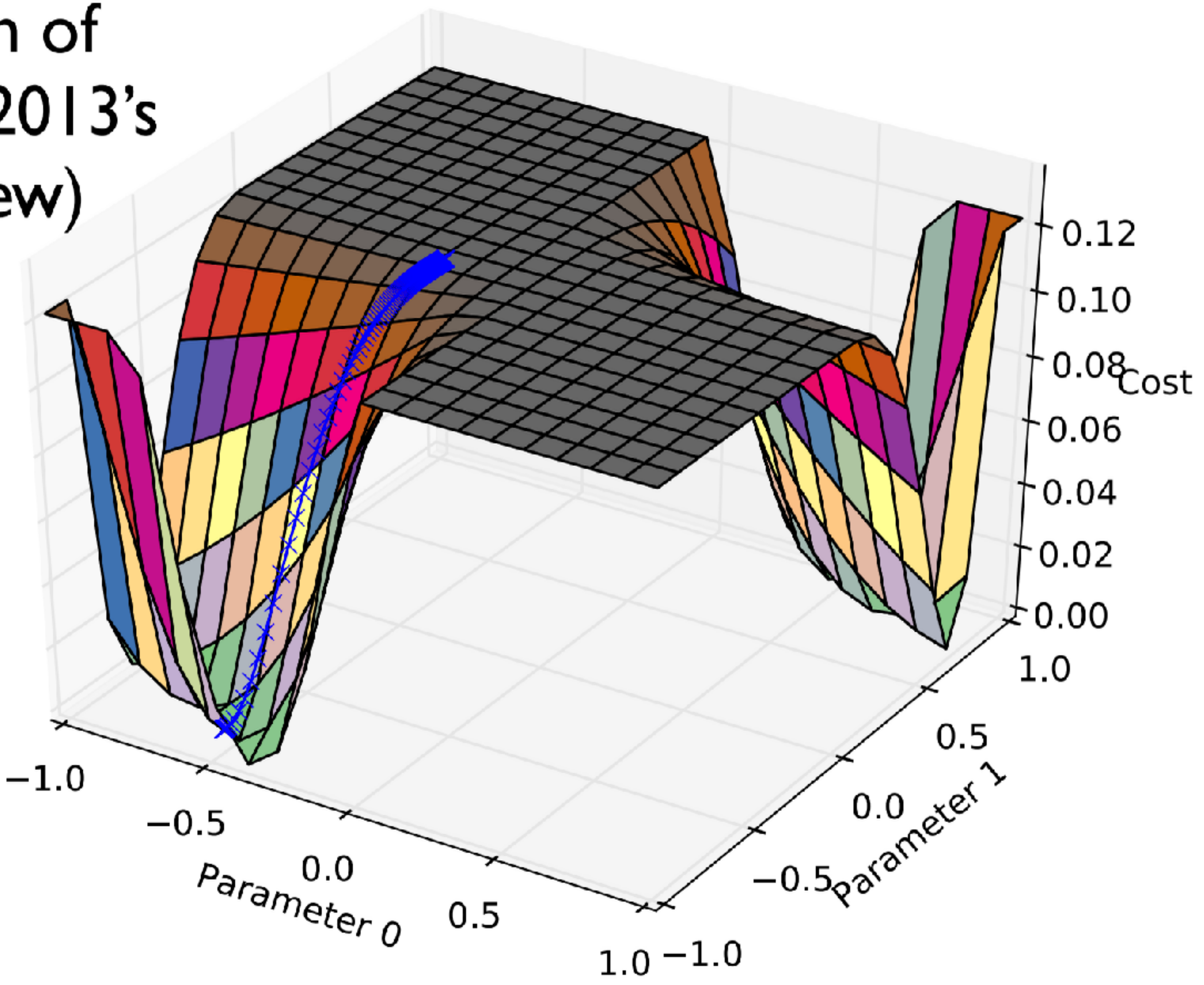
Do neural nets have saddle points?

- Saxe et al, 2013:
- neural nets without non-linearities have many saddle points
- all the minima are global
- all the minima form a connected manifold



(Goodfellow 2015)

(Cartoon of Saxe et al 2013's worldview)



Do neural nets have saddle points?

- Dauphin et al 2014: Experiments show neural nets do have as many saddle points as random matrix theory predicts
- Choromanska et al 2015: Theoretical argument for why this should happen
- Major implication: **most minima are good, and this is more true for big models**.
- Minor implication: the reason that *Newton's method* works poorly for neural nets is its attraction to the ubiquitous saddle points.

(Goodfellow 2015)

Mathematics > History and Overview

[Submitted on 17 Jan 2018]

Deep Learning: An Introduction for Applied Mathematicians

Catherine F. Higham, Desmond J. Higham

Stochastic Gradient Descent: 2 approaches

$$p \to p - \eta \nabla \mathrm{Cost}(p).$$ where $$\nabla \mathrm{Cost}(p) = \frac{1}{N}\sum_{i=1}^{N} \nabla C_{x^{\{i\}}}(p).$$ Too expensive!

Approach 1: (online)

1. Shuffle the integers $\{1, 2, 3, \ldots, N\}$ into a new order, $\{k_1, k_2, k_3, \ldots, k_N\}$.
2. For $i = 1$ upto N, update

$$p \to p - \eta \nabla C_{x^{\{k_i\}}}(p). \tag{4.7}$$

Approach 2: (minibatch)

1. Choose m integers, $k_1, k_2, \ldots, k_m$, uniformly at random from $\{1, 2, 3, \ldots, N\}$.
2. Update

$$p \to p - \eta \frac{1}{m}\sum_{i=1}^{m} \nabla C_{x^{\{k_i\}}}(p). \tag{4.8}$$

https://arxiv.org/abs/1801.05894

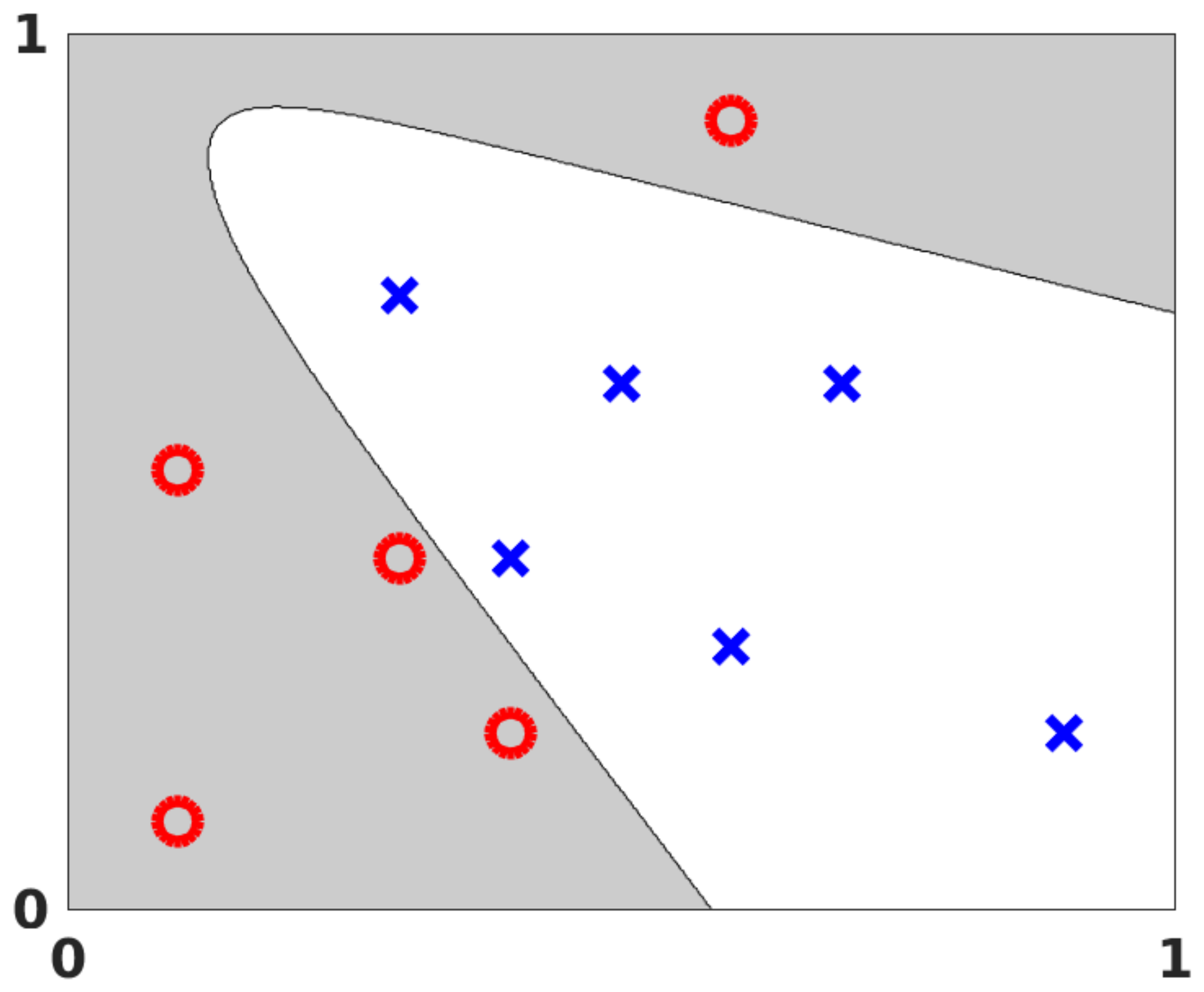
Mathematics > History and Overview

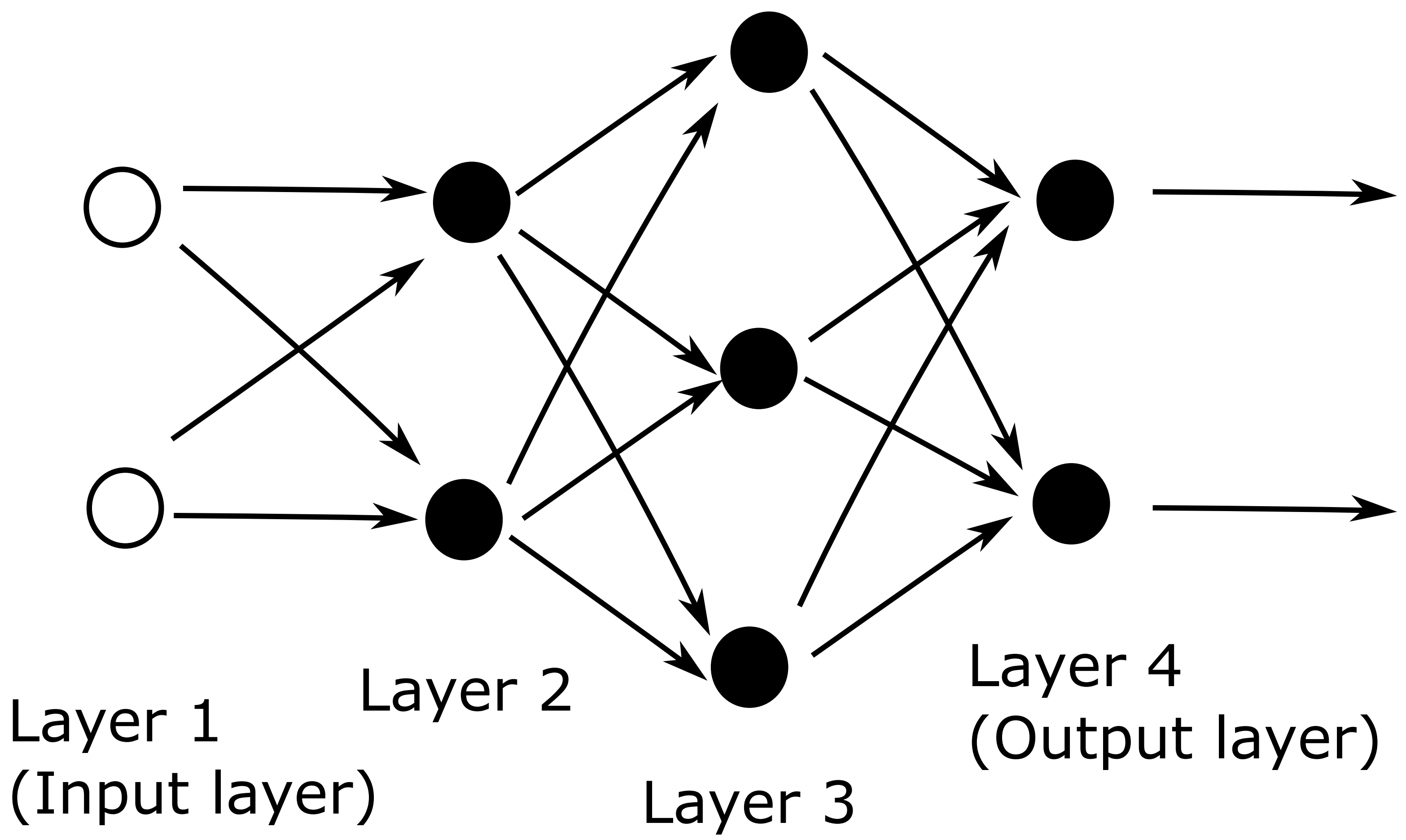
[Submitted on 17 Jan 2018]

Deep Learning: An Introduction for Applied Mathematicians

Catherine F. Higham, Desmond J. Higham

Multilayered artificial neural networks are becoming a pervasive tool in a host
revolution are familiar concepts from applied and computational mathematics;
linear algebra. This article provides a very brief introduction to the basic ideas
perspective. Our target audience includes postgraduate and final year undergr
the area. The article may also be useful for instructors in mathematics who wis
deep learning techniques. We focus on three fundamental questions: what is a
stochastic gradient method? We illustrate the ideas with a short MATLAB code
state-of-the art software on a large scale image classification problem. We fin





$$\text{Cost}\left(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\right) = \frac{1}{10}\sum_{i=1}^{10} \tfrac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2.$$

https://arxiv.org/abs/1801.05894

Stochastic gradient descent

Algorithm 3.4: pseudo-code of a Stochastic Gradient Algorithm to minimise $f(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\boldsymbol{x})$.

input: Gradient functions $\nabla f_i(\boldsymbol{x})$, and starting point $\boldsymbol{x}_0$.
output: An estimate of the minimiser $\boldsymbol{x}_k$.

```
initialise k = 0;
while not reached compute time deadline do
    Choose index i_k ∈ {1, ..., m} uniformly at random;
    Choose λ_k > 0 somehow;
    Set x_{k+1} = x_k − λ_k ∇f_{i_k}(x_k);
    k = k + 1;
end
```

SGD with momentum

Algorithm 3.5: pseudo-code of a Stochastic Gradient Descent Algorithm with momentum to minimise $f(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\boldsymbol{x})$.

---

input: Gradient functions $\nabla f_i(\boldsymbol{x})$, velocity estimate $\boldsymbol{v}_0$ and starting point $\boldsymbol{x}_0$.
output: An estimate of the minimiser $\boldsymbol{x}_k$.

initialise $k = 0$;
while *not reached compute time deadline* **do**
    Choose index $i_k \in \{1\,, \ldots\,, m\}$ uniformly at random;
    Choose $\lambda_k > 0\,, \alpha_k > 0$ somehow;
    Let $\boldsymbol{v}_{k+1} = \alpha_k \boldsymbol{v}_k - \lambda_k \nabla f_{i_k}(\boldsymbol{x}_k)$;
    Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_{k+1}$;
    $k = k + 1$;
end

SGD with momentum

Algorithm 3.5: pseudo-code of a Stochastic Gradient Descent Algorithm with momentum to minimise $f(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\boldsymbol{x})$.

input: Gradient functions $\nabla f_i(\boldsymbol{x})$, velocity estimate $\boldsymbol{v}_0$ and starting point $\boldsymbol{x}_0$.
output: An estimate of the minimiser $\boldsymbol{x}_k$.

initialise $k = 0$;
while *not reached compute time deadline* **do**
    Choose index $i_k \in \{1, \ldots, m\}$ uniformly at random;
    Choose $\lambda_k > 0$, $\alpha_k > 0$ somehow;
    Let $\boldsymbol{v}_{k+1} = \boxed{\alpha_k \boldsymbol{v}_k} - \lambda_k \nabla f_{i_k}(\boldsymbol{x}_k)$; "velocity", accumulates gradients
    Set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{v}_{k+1}$;
    $k = k + 1$;
end

SGD with Nesterov momentum

Algorithm 3.6: pseudo-code of a Stochastic Gradient Descent Algorithm with Nesterov momentum to minimise $f(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\boldsymbol{x})$.

input: Gradient functions $\nabla f_i(\boldsymbol{x})$, velocity estimate $\boldsymbol{v}_0$ and starting point $\boldsymbol{x}_0$.
output: An estimate of the minimiser $\boldsymbol{x}_k$.

```
initialise k = 0;
while not reached compute time deadline do
    Choose index i_k ∈ {1, ..., m} uniformly at random;
    Choose λ_k > 0 , α_k > 0 somehow;
    Let v_{k+1} = α_k v_k − λ_k ∇f_{i_k}(x_k + α_k v);
    Set x_{k+1} = x_k + v_{k+1};
    k = k + 1;
end
```

$$\boldsymbol{v}_{k+1} = \alpha_k \boldsymbol{v}_k - \lambda_k \nabla f_{i_k}(\boldsymbol{x}_k \boxed{+ \alpha_k \boldsymbol{v}});$$

A birds-eye view of optimization algorithms

FABIAN PEDREGOSA
Google AI

November
2018

http://fa.bianp.net/teaching/2018/eecs227at/

also https://distill.pub/2017/momentum/

Do first, then understand

1. Run the compareOptimisationAlgos.ipynb notebook
   - From Liquet, Moka, & Nazarathy: deeplearningmath.org
   - This does “deterministic” GD, but same principle

2. Heavy ball analogy!

3. Adapt the code to do SGD, on the world’s most simple ML problem
   - Start with “online” SGD
   - Then do minibatch SGD (probably with more datapoints)

Adaptive gradient methods

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 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 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
  end while

Adaptive gradient methods

Algorithm 8.4 The AdaGrad algorithm

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

Adaptive gradient methods

Algorithm 8.7 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 defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    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: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Batch Normalization

OpenAI

Ian Goodfellow

Deep Learning Study Group
San Francisco
September 12, 2016

Batch Normalization

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$$

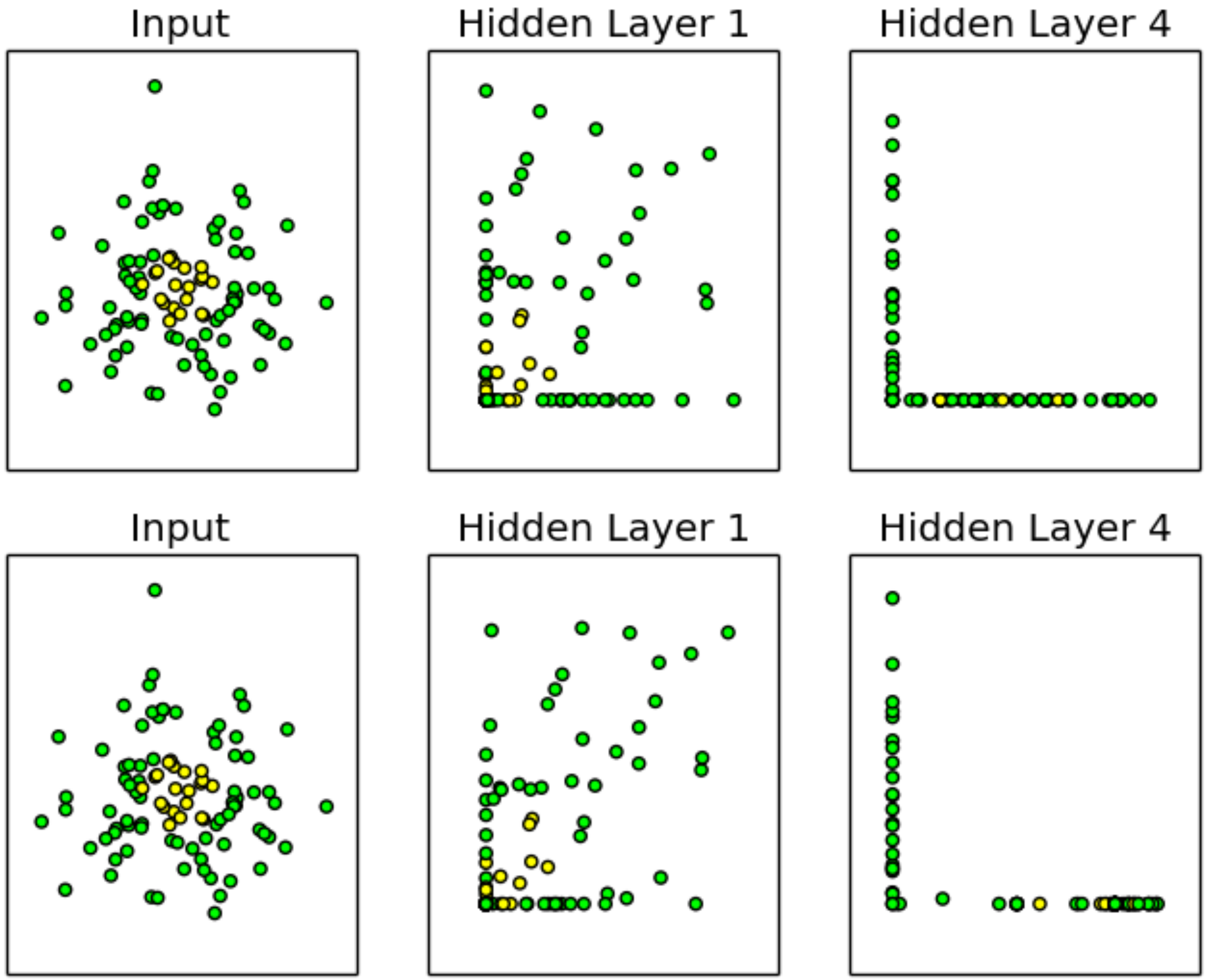
$$\tilde{\boldsymbol{Z}} = \boldsymbol{Z} - \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{Z}_{i,:}$$

$$\hat{\boldsymbol{Z}} = \frac{\tilde{\boldsymbol{Z}}}{\sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m} \tilde{\boldsymbol{Z}}_{i,:}^{2}}}$$

$$\boldsymbol{H} = \max\{0, \boldsymbol{\gamma}\hat{Z} + \boldsymbol{\beta}\}$$

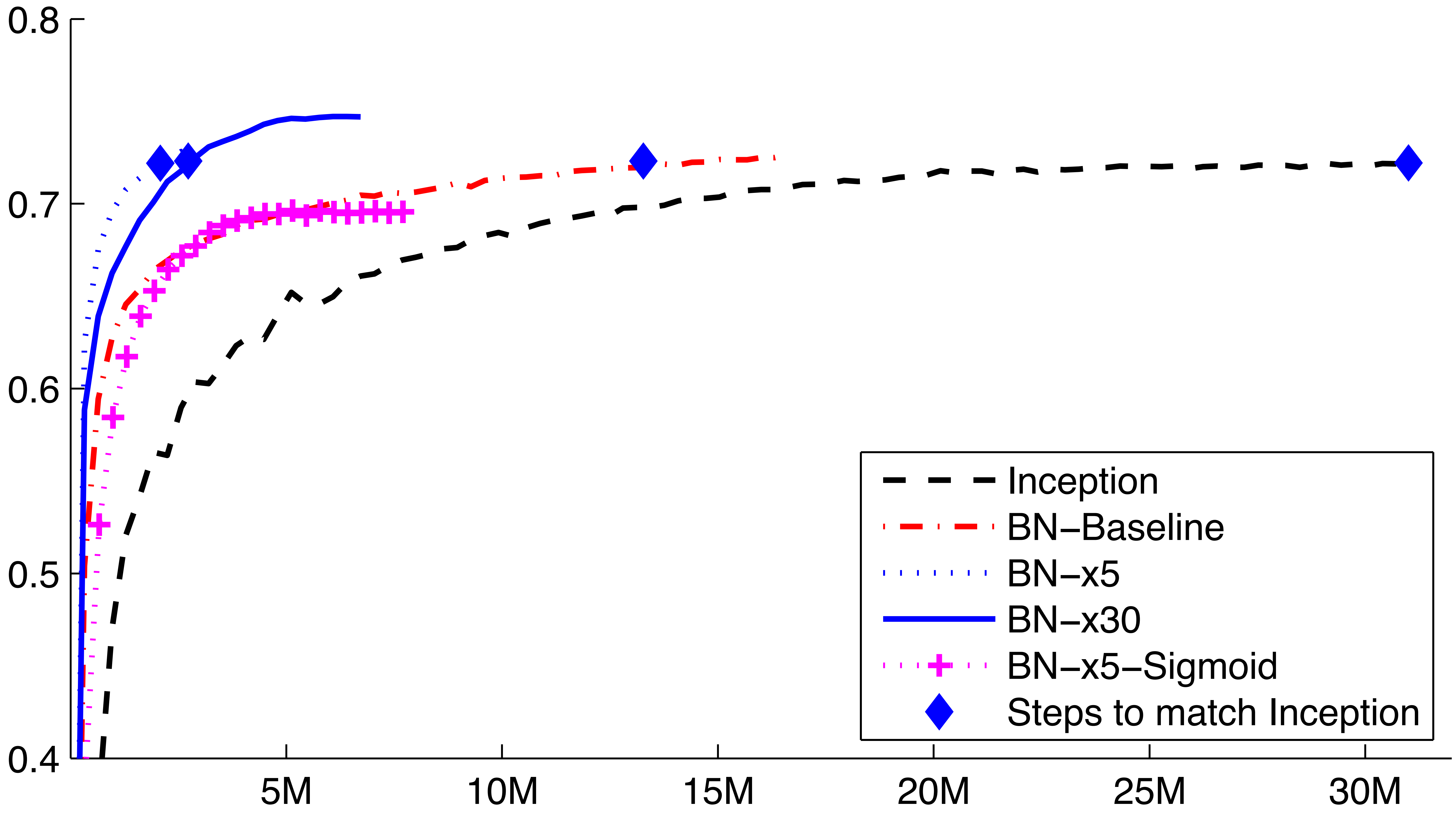
"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Ioffe and Szegedy 2015

Before SGD step



After SGD step

“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Ioffe and Szegedy 2015



“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Ioffe and Szegedy 2015