Tutorial: Deep Learning

Rina Panigrahy
Google Corp.

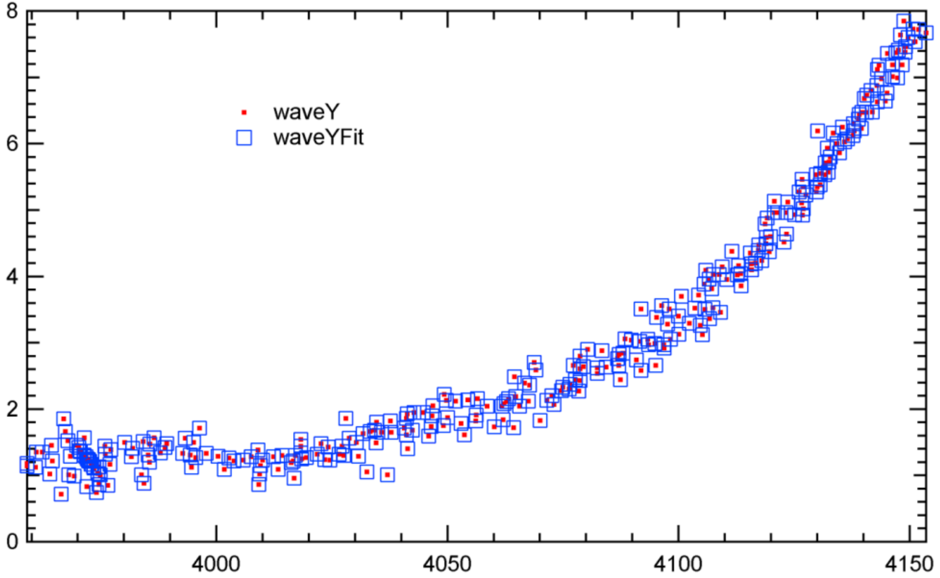
Outline

- Basics
  - Machine Learning problem specification
  - Linear and logistic regression
  - Gradient Descent Optimization
  - Deep Learning
- Applications (will use online lectures/slides from application experts)
  - MNIST
  - Image and speech recognition
  - Language Translation
- Theoretical Understanding?
  - Local vs Global Minima
  - Learning synthetic function classes.

Learning an unknown function

Learn f from training pairs (x,y) so that you can predict its output on new inputs
Like learning a manifold.

Learning an unknown function: like curve fitting



Learn f from training pairs (x,y) so that you can predict its output on new inputs
Like curve/manifold fitting.

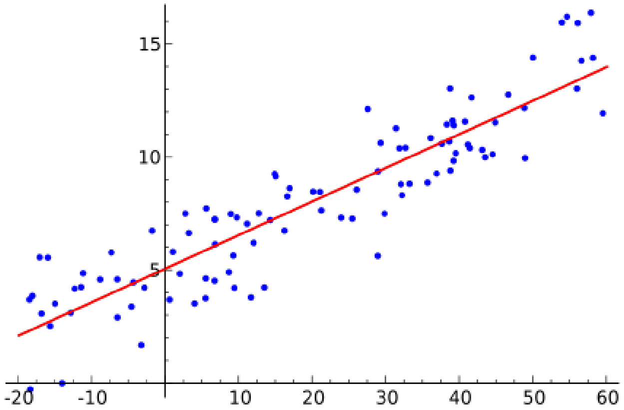
Learning a function: why?

- Want to make predictions in real life situations.
- Will a user click on an ad? (Directly affects profitability)
- User features encoded by a vector x (e.g. earlier queries)
- Predict y probability of clicking on an ad.
- Given training pairs (x, y) learn a function f so that
  - f(x)=y

Learning a function: How

- Find f from a certain function class: Modelling f
- A simple model for f: linear regression
- Logistic regression
- Deep learning
  - Useful in many engineering applications such as image/speech recognition, ad-matching

Linear Regression: Line fitting



- For input x: $f_w(x) = w_1.x + w_0$
- Find w so that predicted output $f(x_i) \approx y_i$

Minimize error(loss) in prediction

- Square Error = $L(w; x,y) = (f_w(x) - y)^2$
  - Other possibilities l_1 loss = $|f_w(x) - y|_1$
- For many examples x_i, y_i
- $L(w) = \sum_i L(w; x_i, y_i) = \sum_i (f_w(x_i) - y_i)^2$
- Find best fit w by $\min_w L(w)$

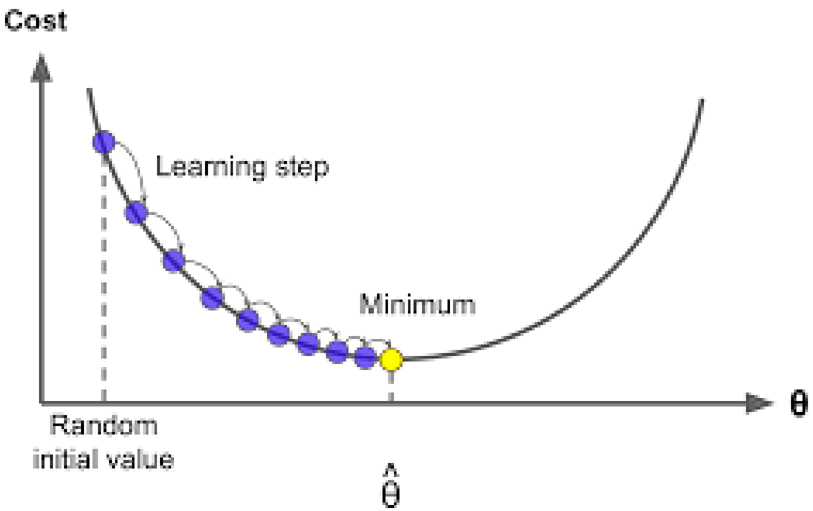
Loss measures error in prediction

- For Linear regression with l_2 squared loss
- $\min_w \sum_i (w_1.x + w_0 - y_i)^2$
- Can be solved analytically
- For other loss functions use Gradient Descent

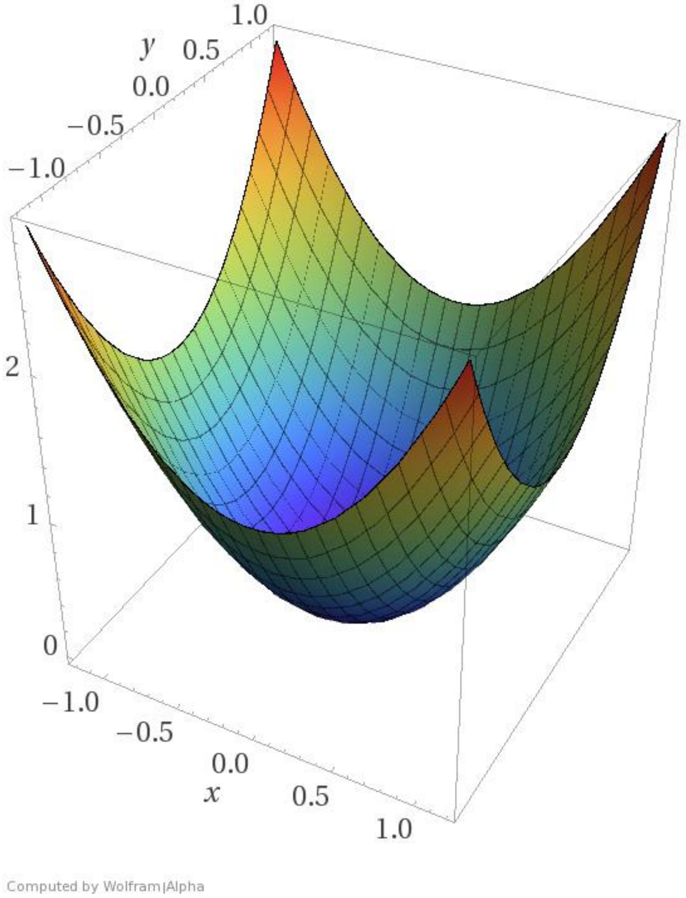
Gradient descent

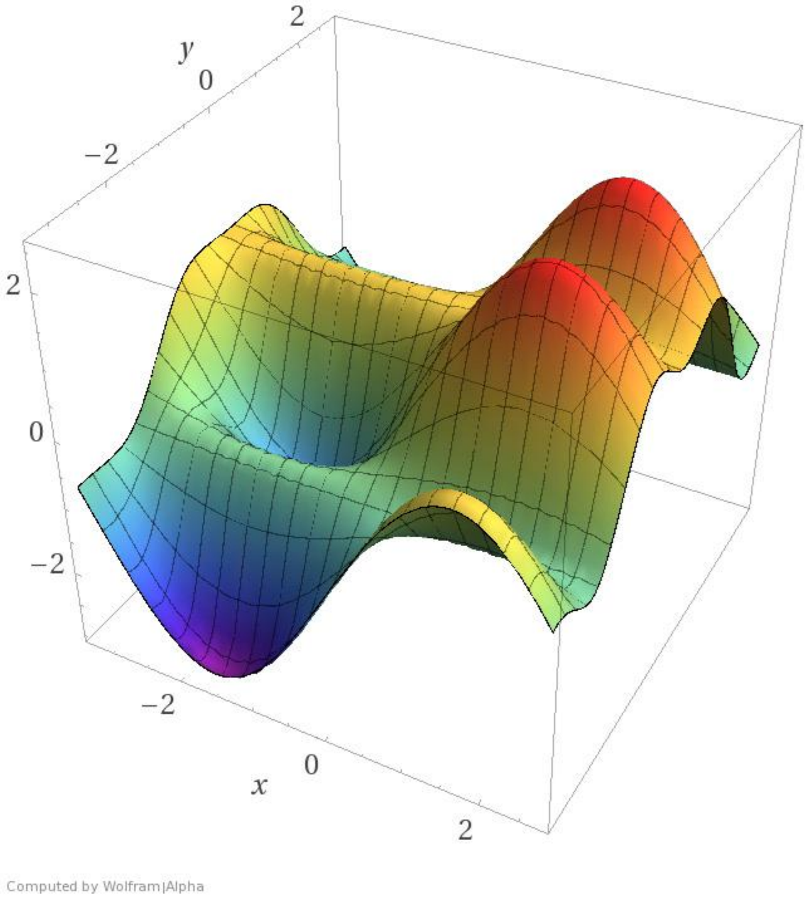
$$\Delta\theta_i \propto -\frac{\partial L}{\partial \theta_i}$$

$$\Delta\theta = -\eta \nabla L_\theta$$

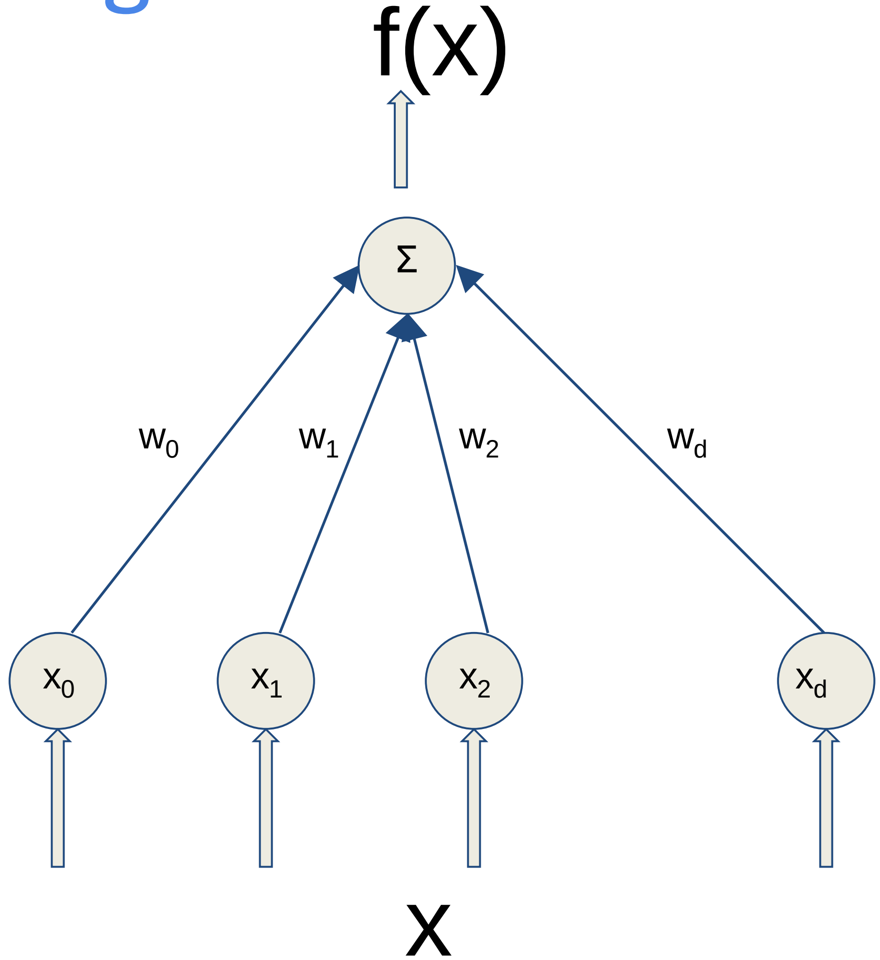


- To minimize L(θ) change each parameter θ_i in the direction that decreases L


y
1.0
0.5
0.0
−0.5
−1.0
2
1
0
x
−1.0
−0.5
0.0
0.5
1.0
Computed by Wolfram|Alpha


y
2
0
−2
2
0
−2
x
−2
0
2
Computed by Wolfram|Alpha

Learning a function: Linear Regression



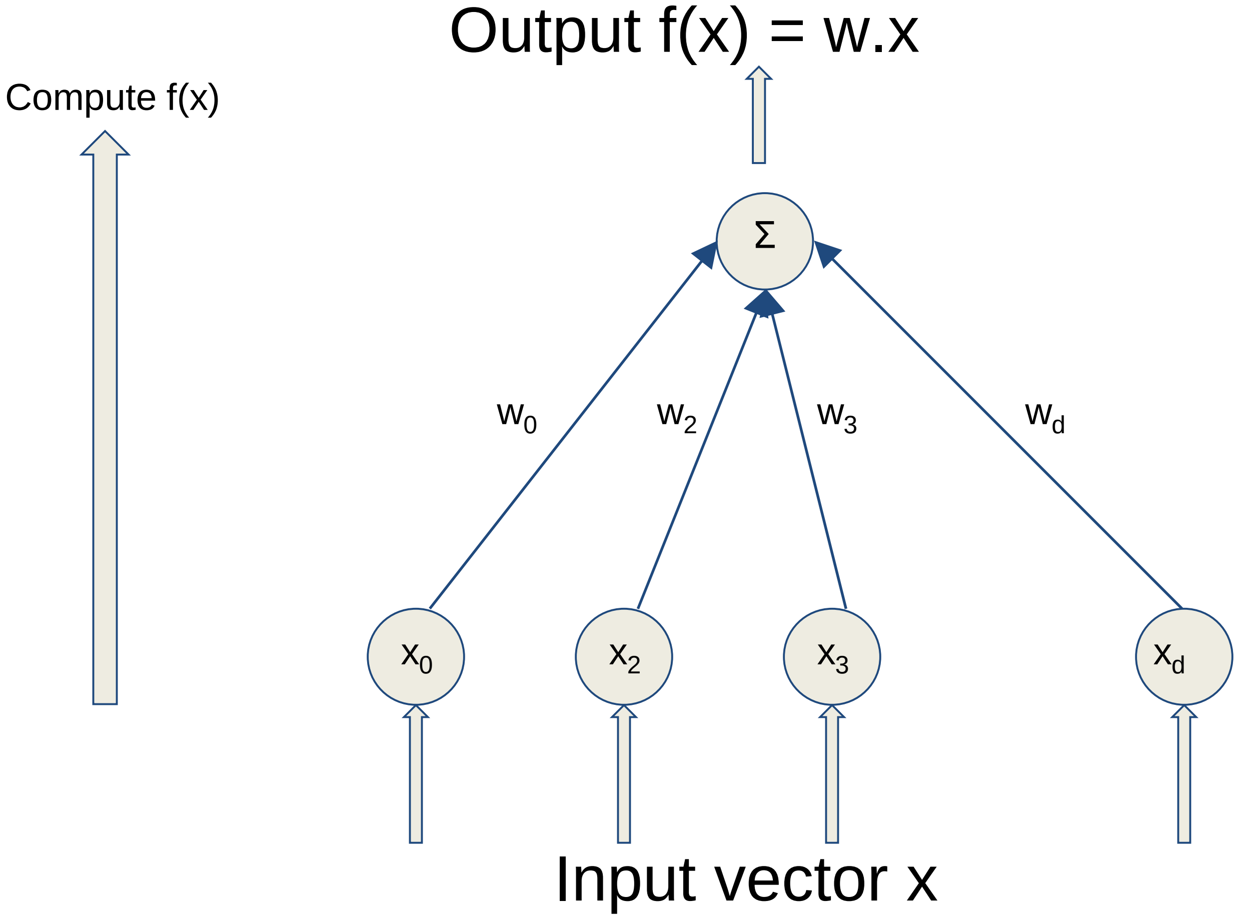
Output $f(x_i) = w.x_i$
Want this to be $\approx y_i$

Hidden weight vector w

(to be learn't from training data)

Input vector x_i

Gradient update: BackPropagation.



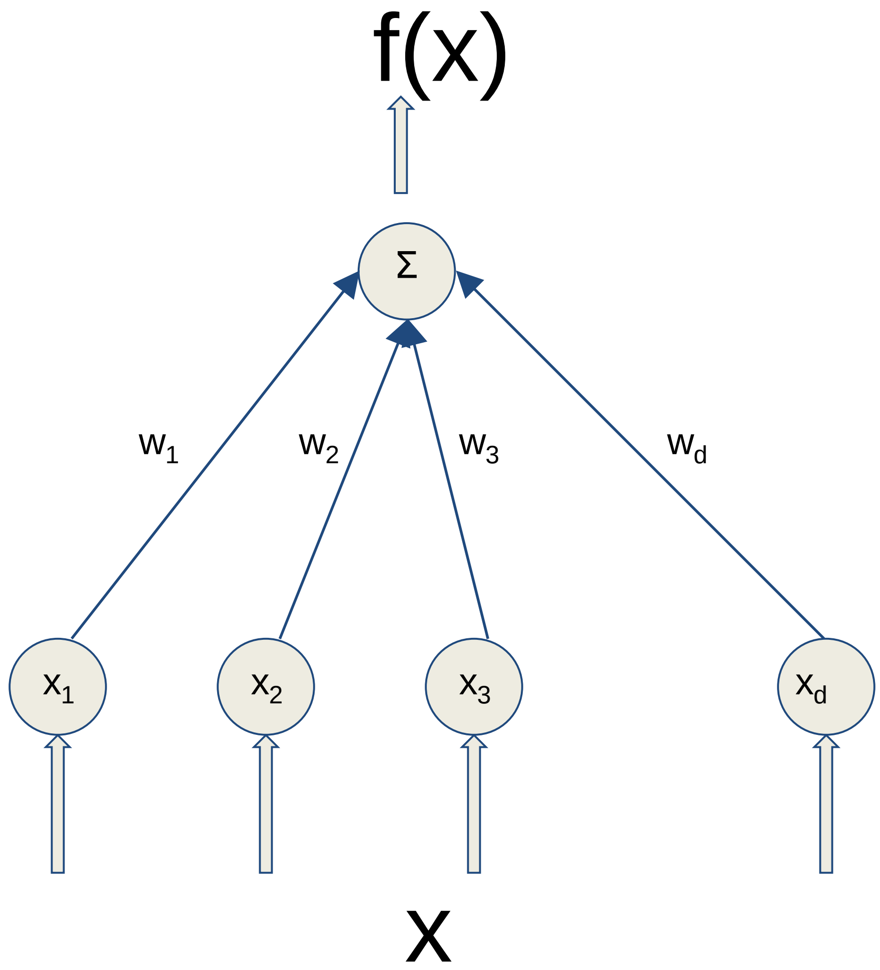
Apply gradient updates

$$L = (y - f(x))^2$$

$$\frac{\partial L}{\partial w_i} = -(y - f(x))x_i$$

$$\Delta w_i = \eta(y - f(x))x_i$$

Stochastic Gradient Descent: gradients over a few examples at a time.



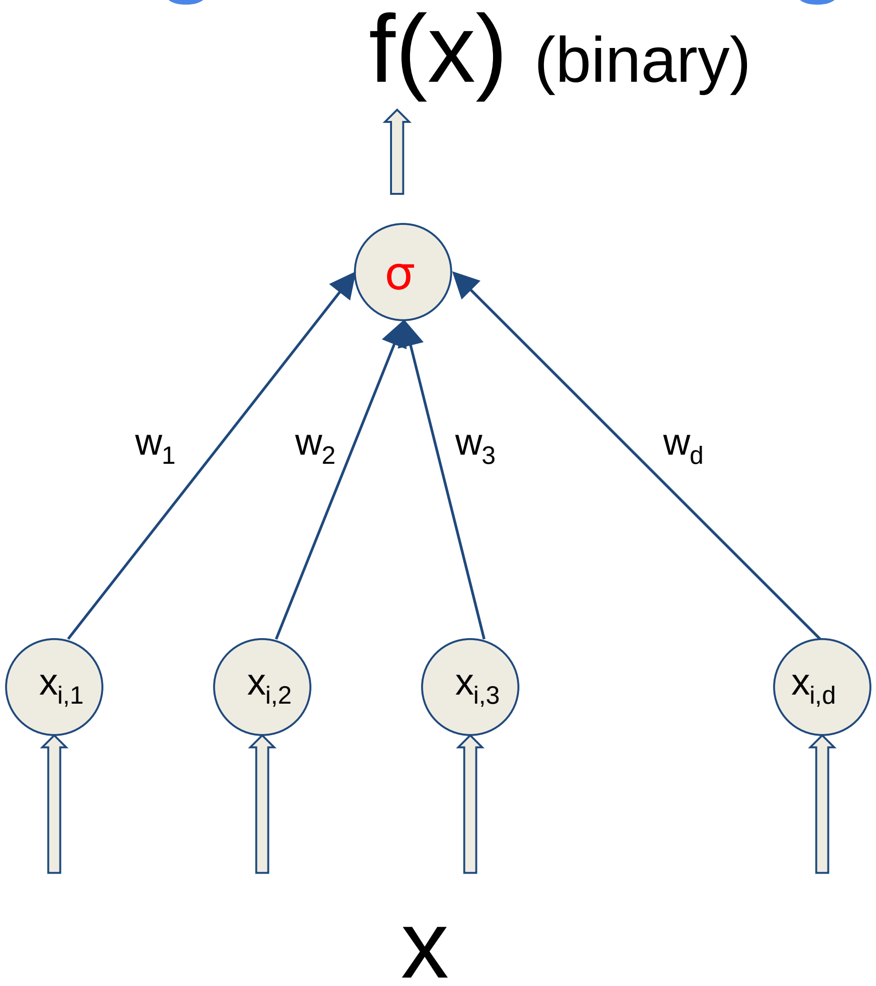
Output f(x) = w.x

$$\frac{\partial L}{\partial w_i} = -(y - f(x))x$$

$$\Delta w_i = \eta(y - f(x))x$$

Input vector x

Learning a function: Sigmoid, sign



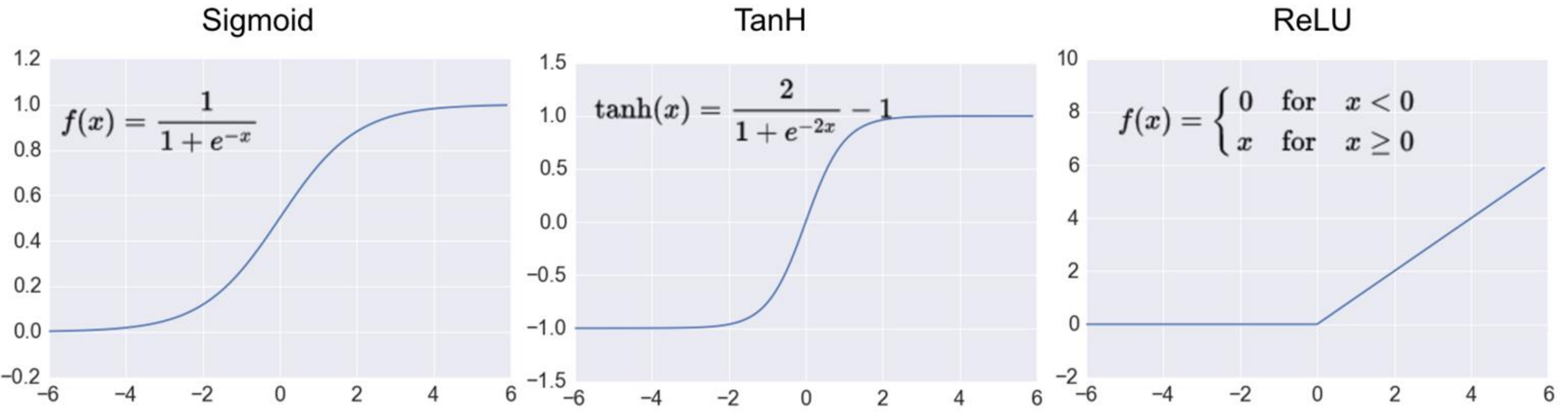
Output $f(x_i) = \sigma(w.x_i)$

Hidden weight vector w

(to be learn't from training data)

Input vector x_i

Sigmoid, RELU

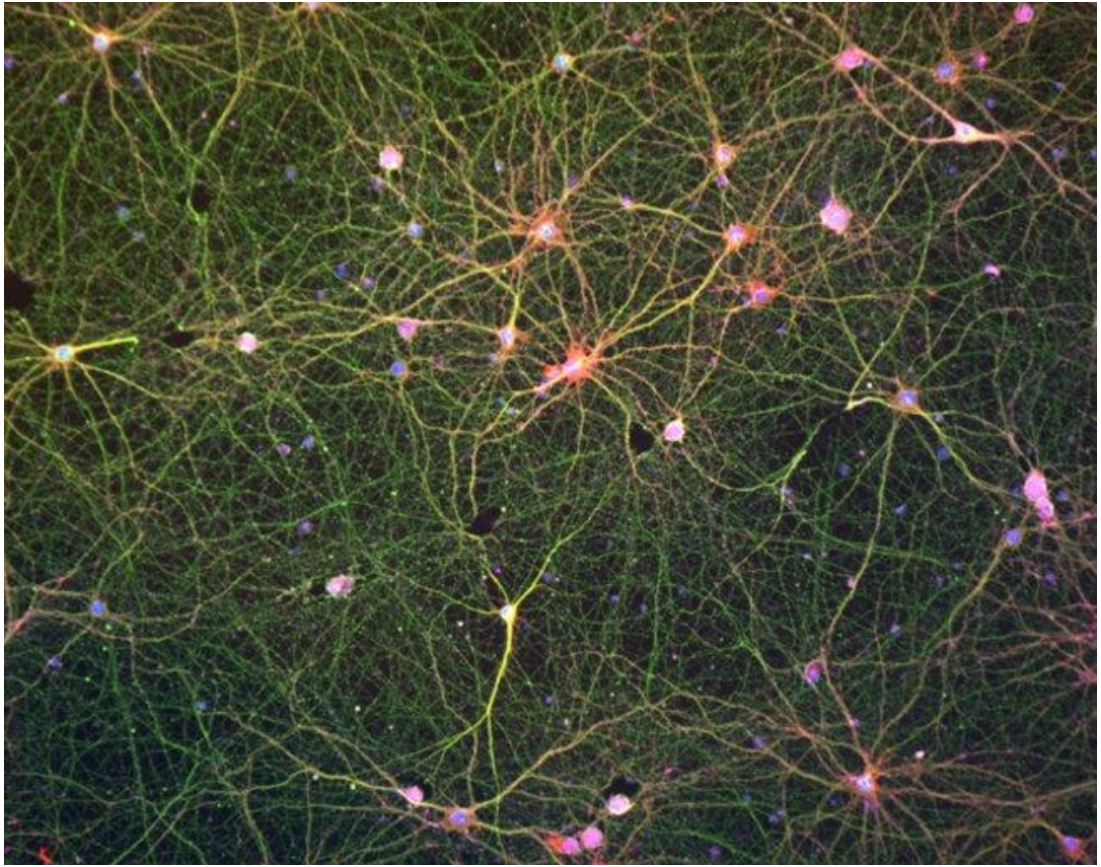


Logistic regression uses logloss

- Maximize predicted probability of observed data
- Or sum of log probabilities
- Probability = f if y=1, 1-f if y=0.
- Log loss = $L(w; x,y) = y.\log f_w(x) + (1-y) \log (1-f_w(x))$
- Cross entropy (similarity) between observed and predicted probability

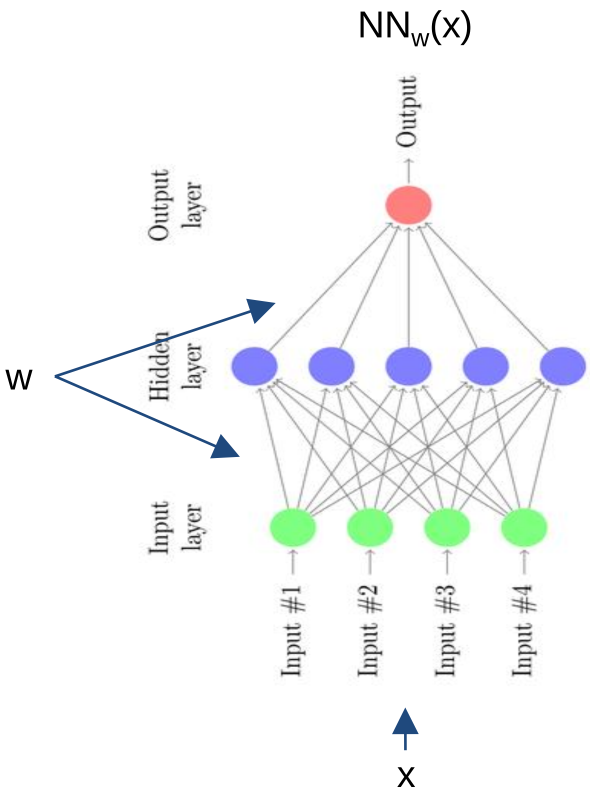
Neurons

Network of Neurons



Deep Network. Allows rich representation
Can express any function/circuit

Output $f(x) = NN_w(x)$



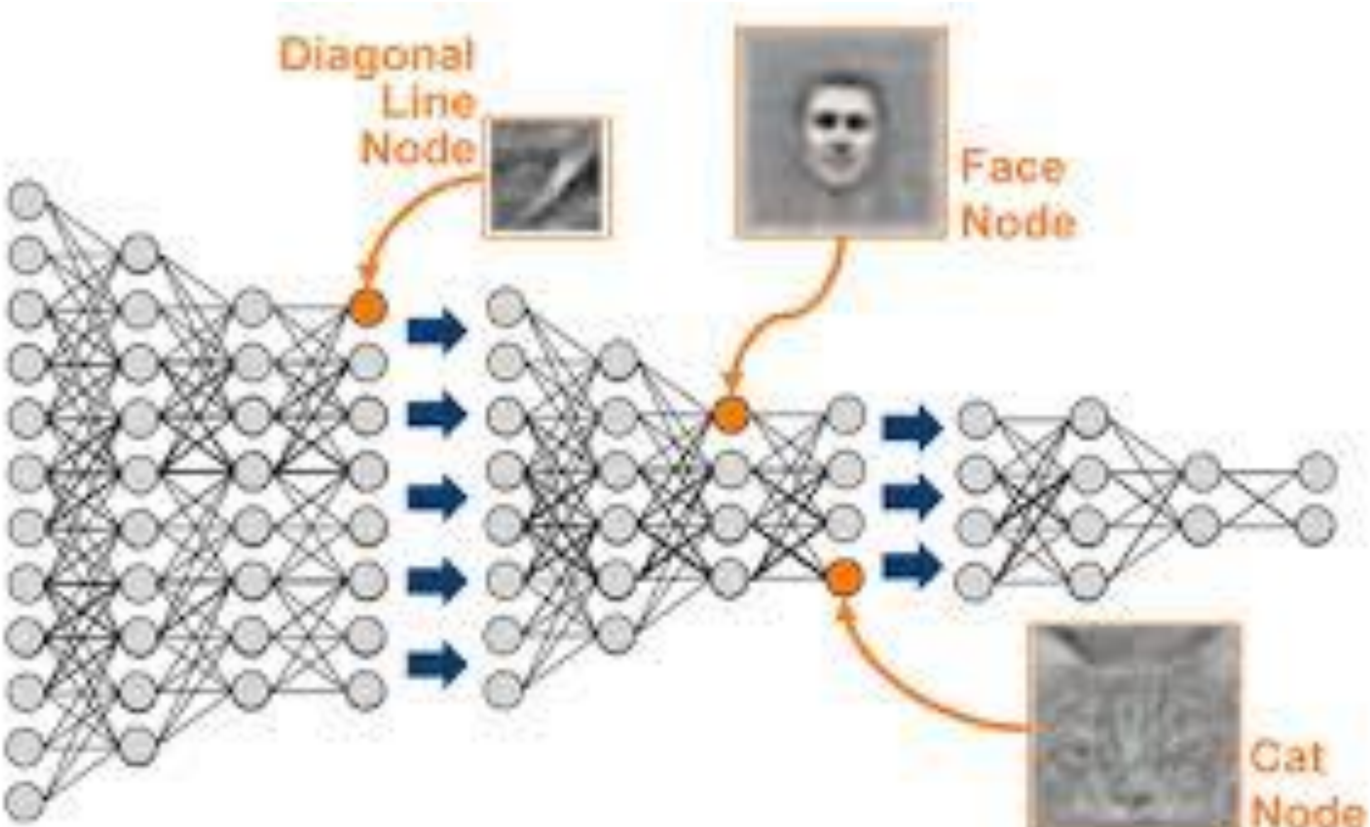
Hidden edge weight matrix w

(to be learn’t from training data)

Input vector x

Hierarchical representation of Objects

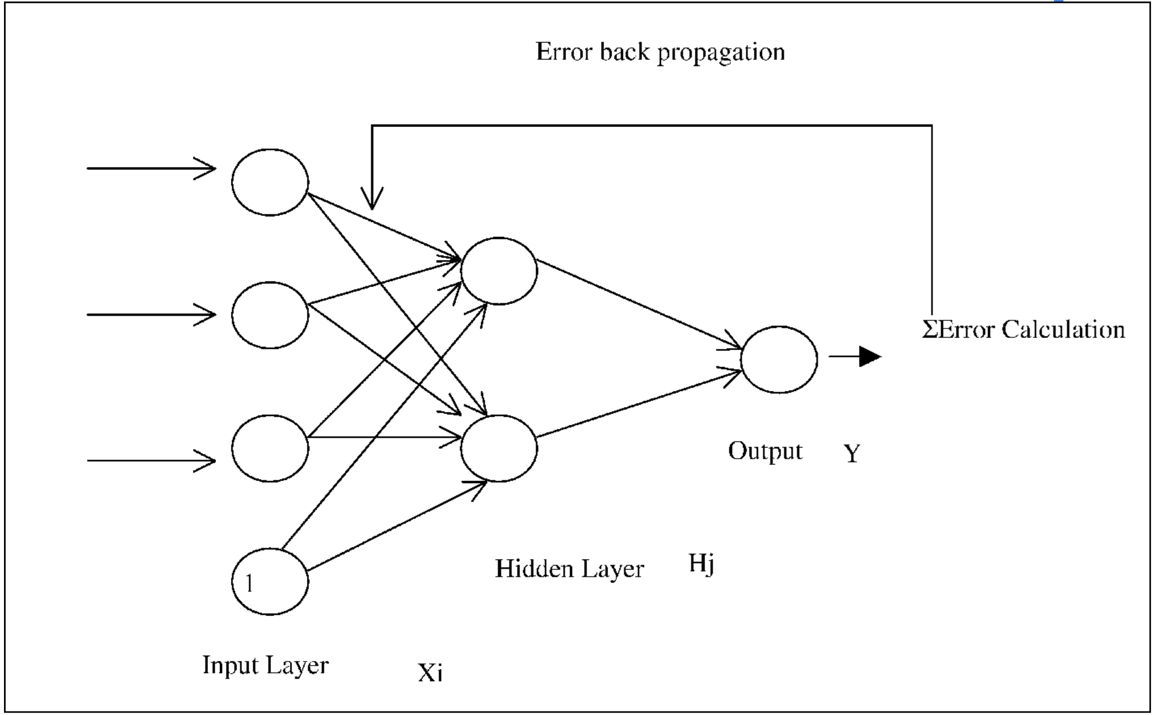
[Qvoc V. Le et al, ICML 2012]



Training w: SGD to Minimize loss

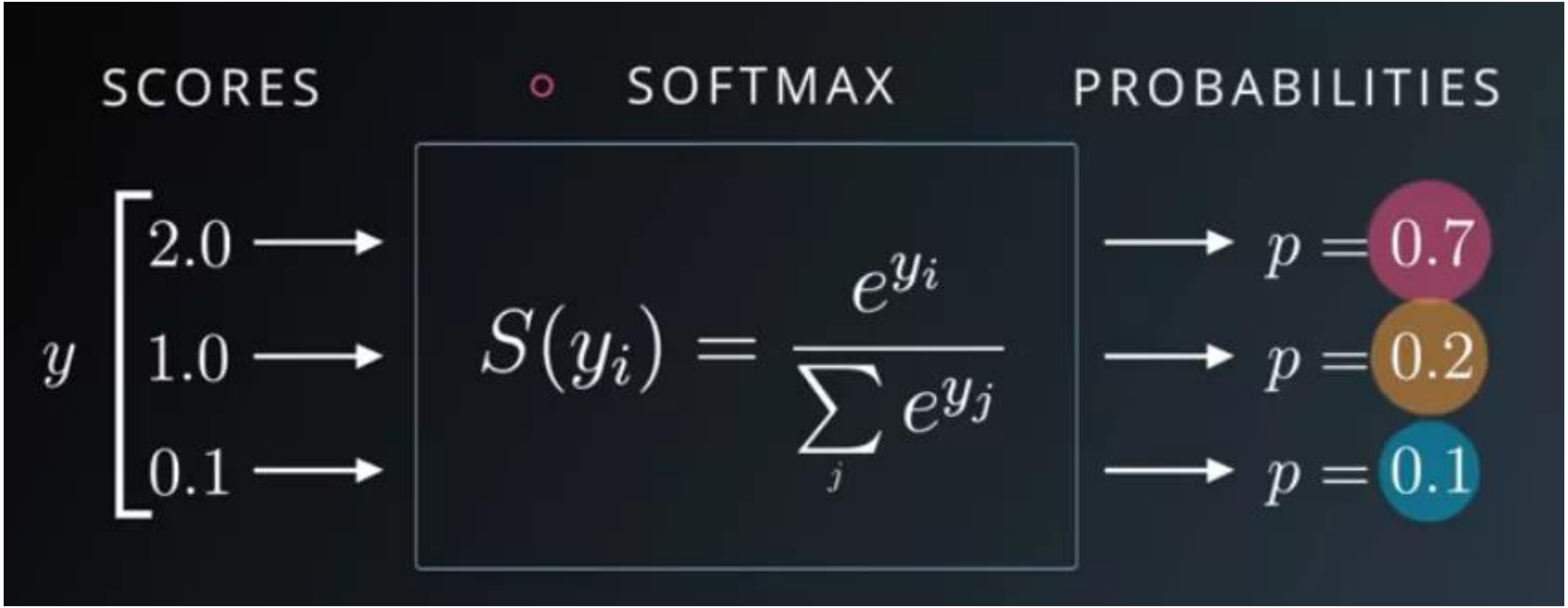
- Square Error = $L(w; x,y) = (f_w(x) - y)^2$
  - Other possibilities l_1 loss = $|f_w(x) - y|_1$
- For many examples x_i, y_i
- $L(w) = \sum_i L(w; x_i, y_i) = \sum_i (f_w(x_i) - y_i)^2$
- Find best fit w by $\min_w L(w)$
- Solve by GD
- SGD: Sample a few inputs.

Backpropagation: Gradient Descent for one example



Notes: The weight connecting node *i* in the input layer to node *j* in the hidden layer is denoted by *Wji*, and the weight connecting node *j* to the output node is represented by *Vj*

Softmax for multiclass output



Convergence of Gradient Descent for Model training

- Minimize Loss function over training data
- Loss function $L = E_x [(y - f_w(x))^2]$
- Minimize Loss function : $\min_w \; E_x [(y - f_w(x))^2]$
- Gradient over parameter space w
- Hope it converges to optimal parameters w
- This happens for linear/logistic regression
- What about deep learning?

Applications

Applications

- MNIST
- Image Recognition: Imagenet
- Speech Recognition
- Language Translation.

Many many others

- Ads matching
- Web search and ranking

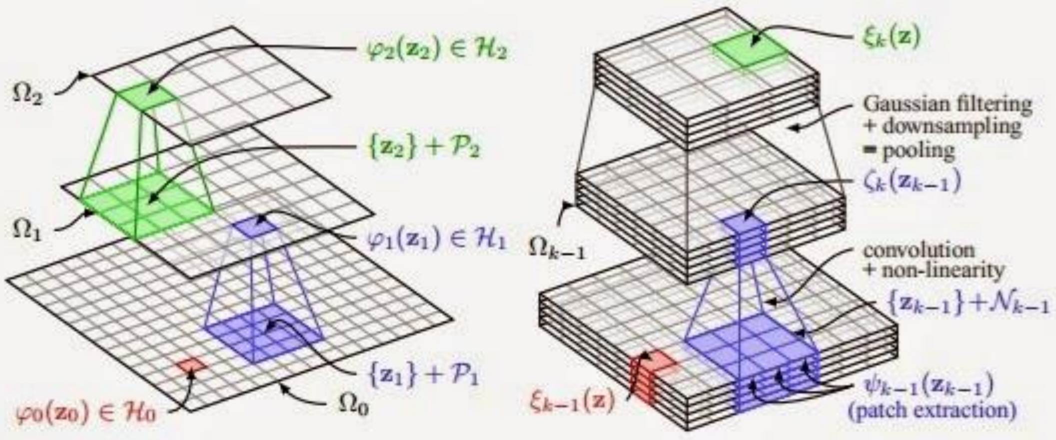
MNIST

Training data:
- 60,000 examples
- 32x32 pixels

Test data
- 10,000 examples

http://yann.lecun.com/exdb/mnist/
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/DeepLearning.pdf

Convolution and Pooling



(a) Hierarchy of Image Feature Maps

(b) Zoom between layer k and $k-1$ of our CKN

Figure 1: Left: concrete representation of the successive layers for the multilayer convolutional kernel. Right: one layer of the convolutional neural network that approximates the kernel.

Imagenet

Alexnet paper:

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Presentation:

http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf

ImageNet

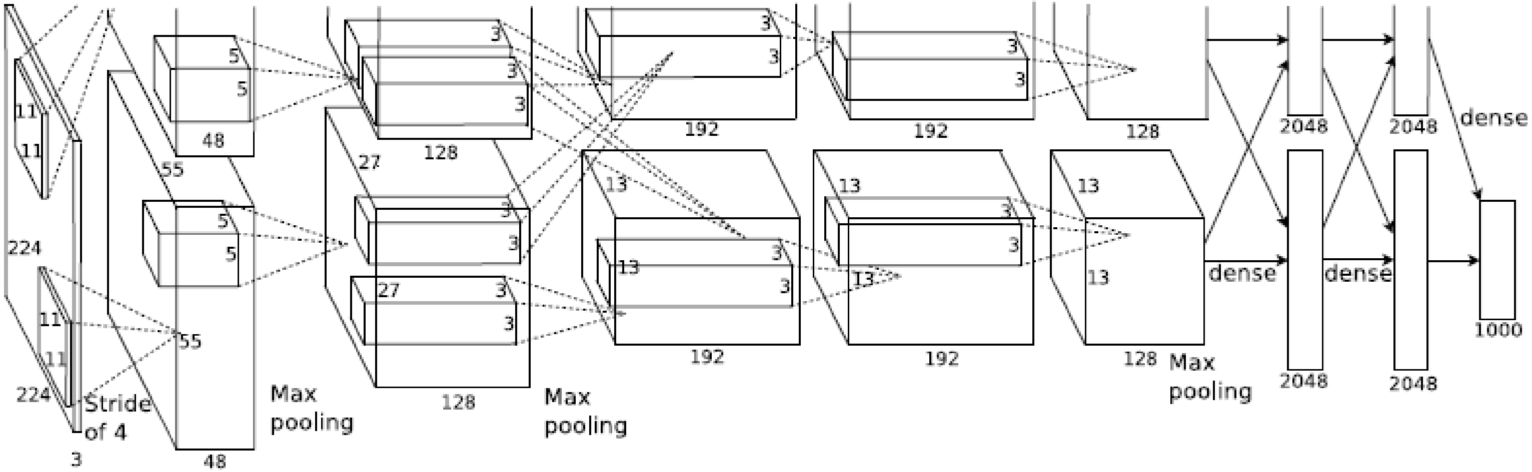


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Speech Recognition

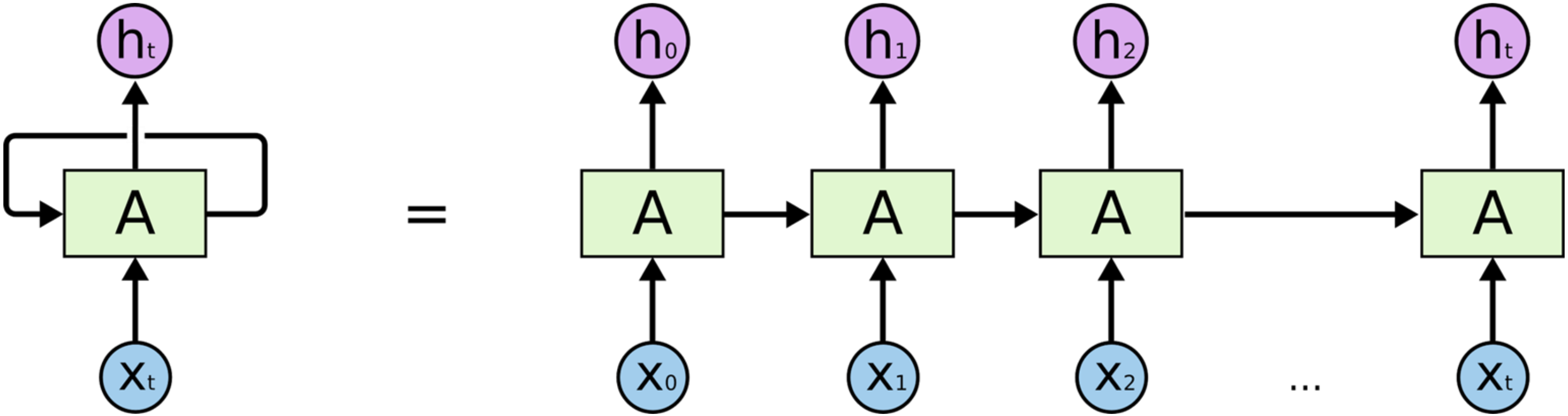
Hintons Slides:

https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec1.pdf

Machine Translation

http://www.cs.toronto.edu/~guerzhoy/321/lec/W09/rnn_translate.pdf

RNN



Videos/tutorials on Deep learning applications

Lectures by Geoff Hinton: search “hinton deep learning tutorial”

Lectures by Ruslan Salakhudinov: search “Salakhudinov deep learning tutorial simons workshop”

Lan Yeccuns slides/talk: https://cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

Language translation: http://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ilya_LSTMs_for_Translation.pdf

http://www.cs.toronto.edu/~guerzhoy/321/lec/W09/rnn_translate.pdf

Alexnet:
http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf

For imagenet results.

Here is another good source:

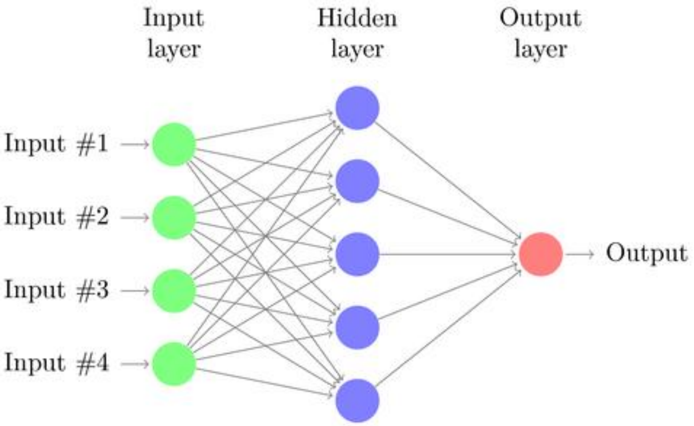
http://www.cs.cmu.edu/~aarti/Class/10701_Spring14/slides/DeepLearning.pdf

https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec7.pdf

Theoretical Understanding?

Deep Learning

- SGD works well in practice but does it reach optimum?
- Does deep learning work provably?

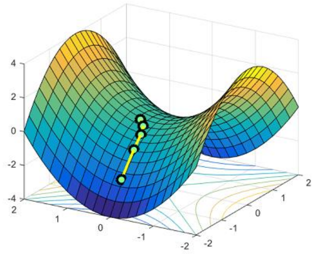


Main Question: *Why* does SGD solve $\min_{\theta} E_X[(f(X) - NN_\theta(X))^2]$

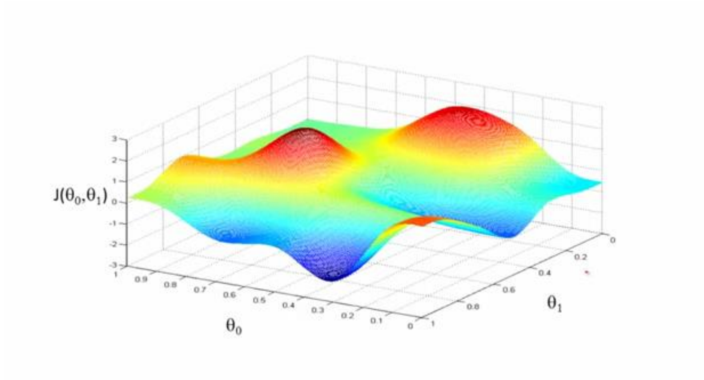
Nonconvex Optimization

- Deep learning involves minimizing non-convex loss functions, which makes analysis difficult

- Recent work shows that SGD escapes saddle points (GeHJY15)

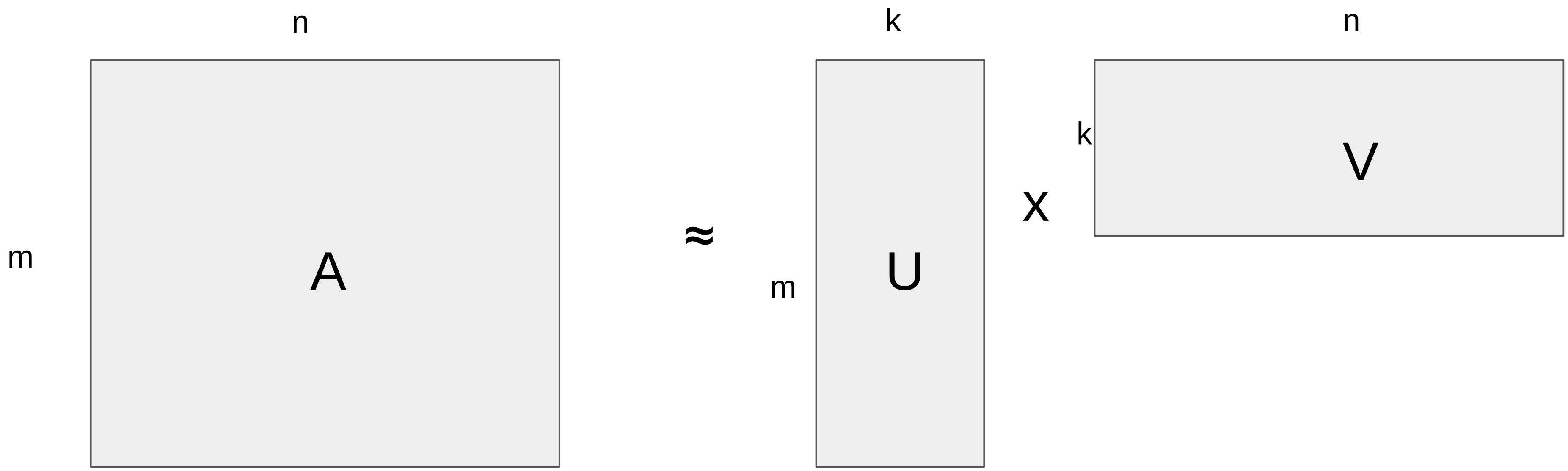


- But even a simple network admit many local minimas



- Best “explanation”: “Random” loss landscapes admit mostly saddle points when error is high.
- Statistical Physics approcahes by Ganguly et al, Choromanska et al

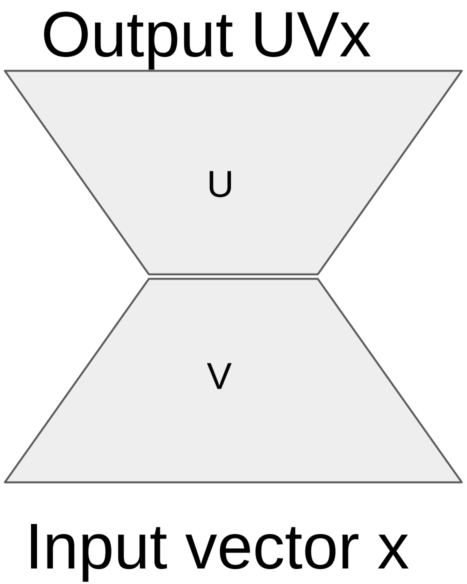
Low rank Approximation



Write matrix A as a product of two thin matrices U and V (say Netflix matrix)

Rows U_i= latent representation (embedding) of user, Columns V_j = latent representation of movie

No local minima in linear networks [Kawaguchi, NIPS 16, Ge et al, ICML 17]



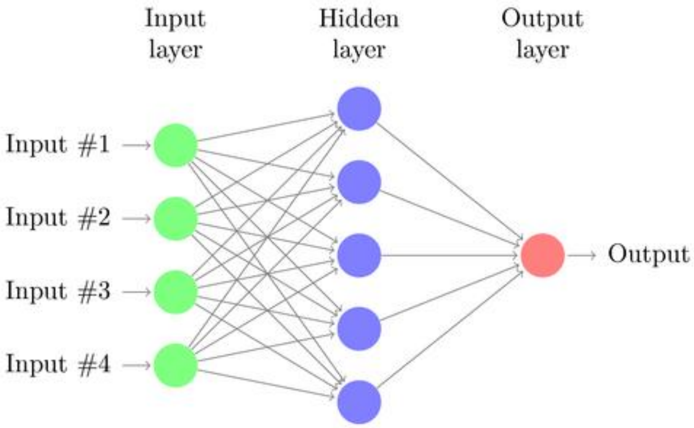
- **Low rank approximation is same as:**
- **Train 2 layer network with examples (x,Ax)**

Deep Learning

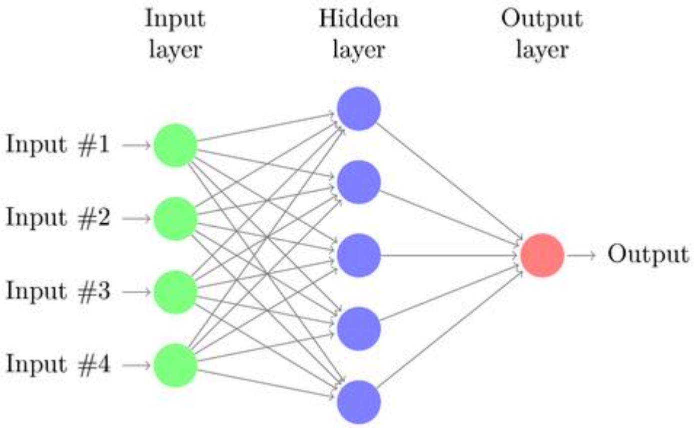
- **Theoretical Question**: What “mathematical” function classes can learned with deep learning (SGD/backprop)?
  - Using “mathematical” function classes instead of real-world functions allows for analysis
- Important “mathematical” function classes:
  - Polynomials? [A,P,V,Z ICML14]
  - Decision Trees?
  - Arithmetic Circuits?
  - Neural Circuits/Networks?

Deep Learning

- In this work, we will focus on learning f(x) = **neural networks (*using neural networks*)**.



$$f(x) = NN_w(x)$$

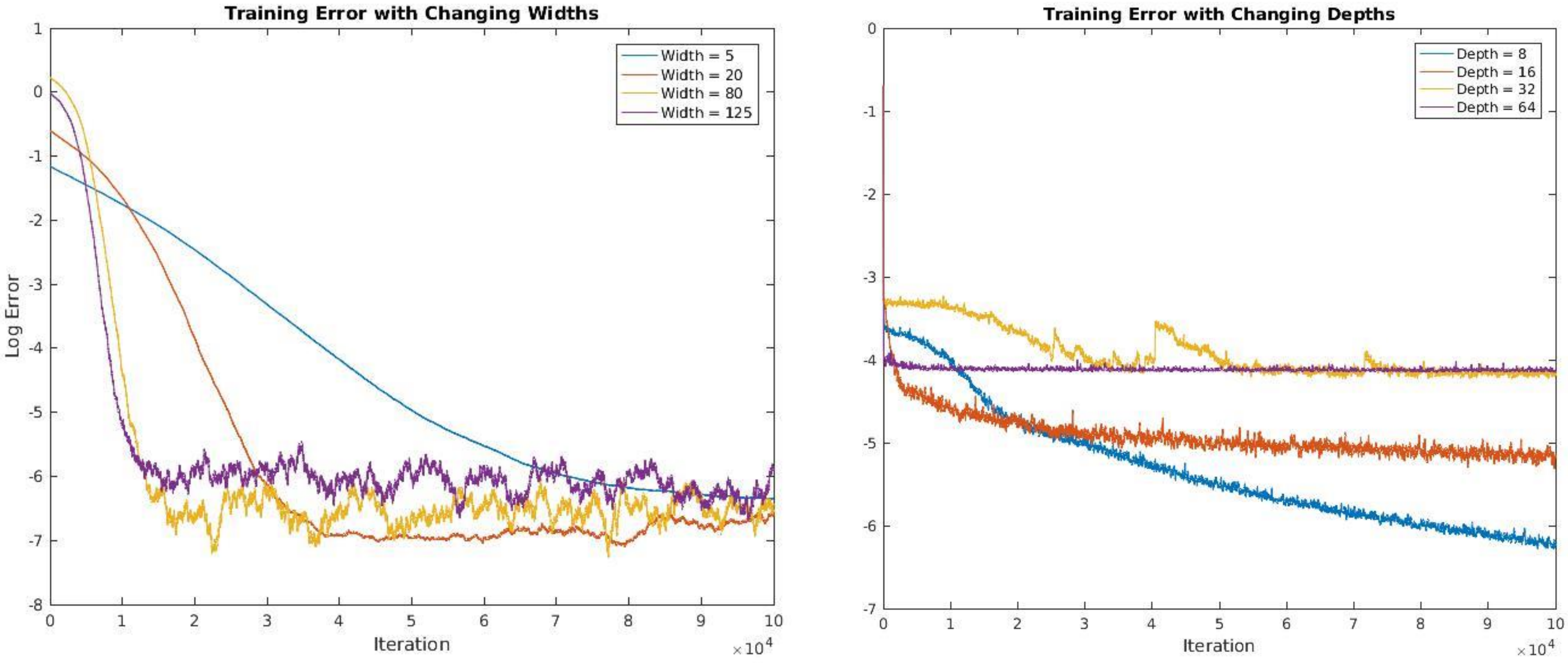


$$NN_\theta(x)$$

Main Question: Does SGD cause $\theta \to w$ (if same network structure)?

Does well experimentally

Results of training on samples from random neural networks



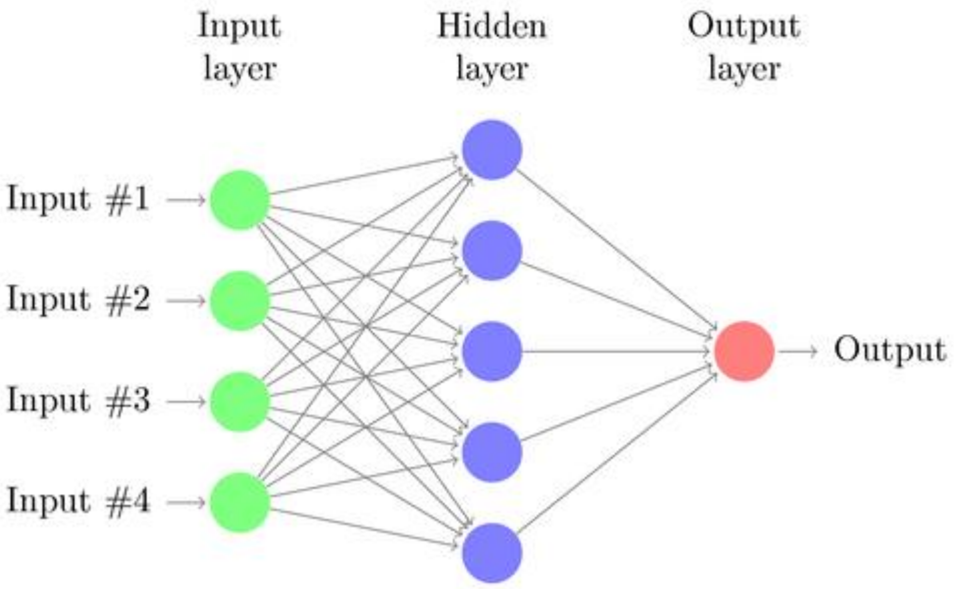
Theoretical Proof of Observed Behavior?

- Derive theoretical justifications under simplifying assumptions:
  - 1 hidden layer
  - Data is generated from a network of known shape, but random unknown weights
  - Infinite data, so it becomes GD
  - Infinitesimal step sizes

With simplifications, our target functions f are...

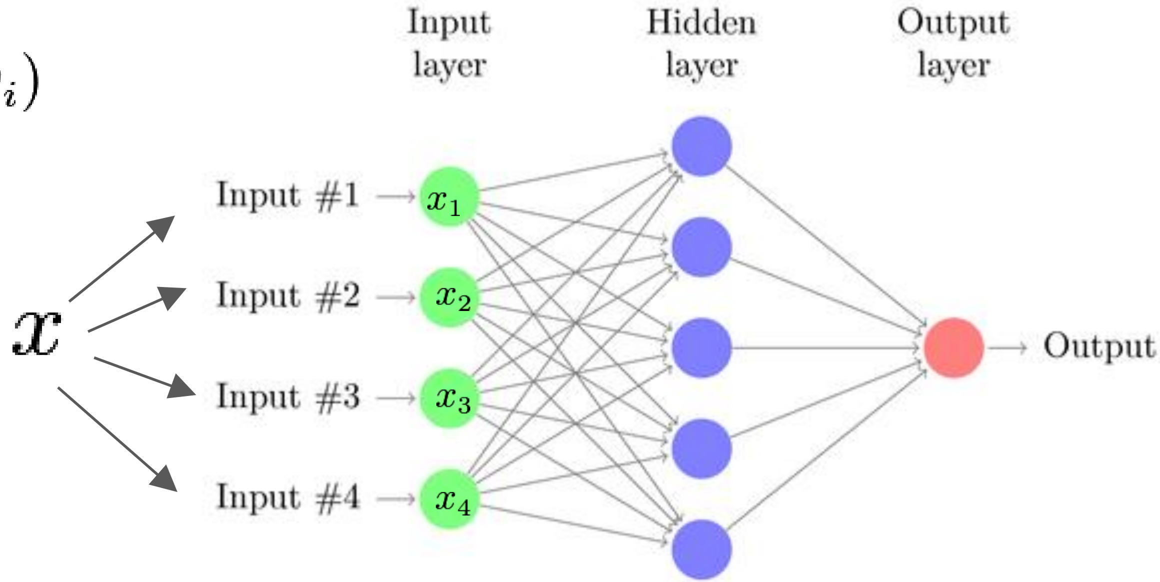
Neural networks with 1 hidden layer

$$f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$$



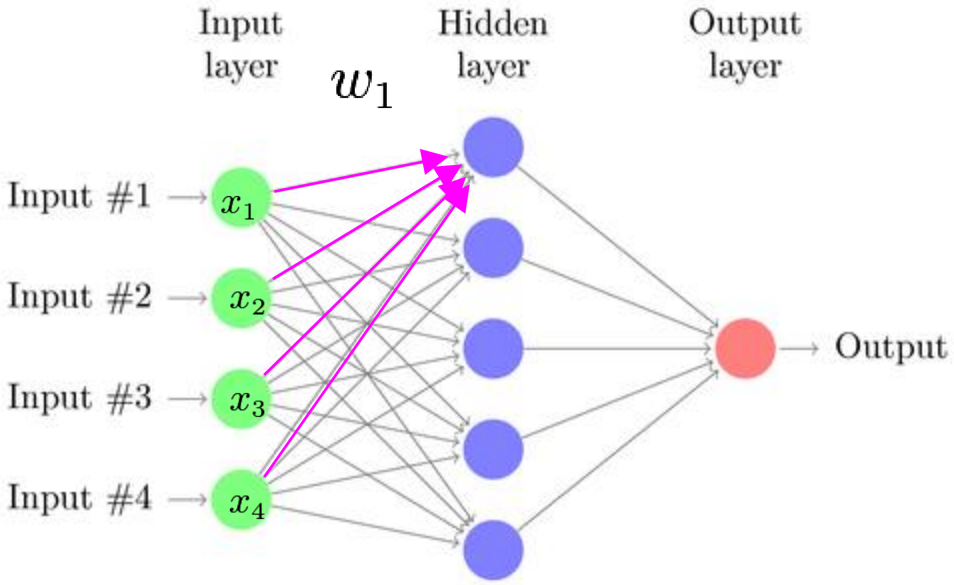
With simplifications, our target functions f are...

$$f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$$



With simplifications, our target functions f are...

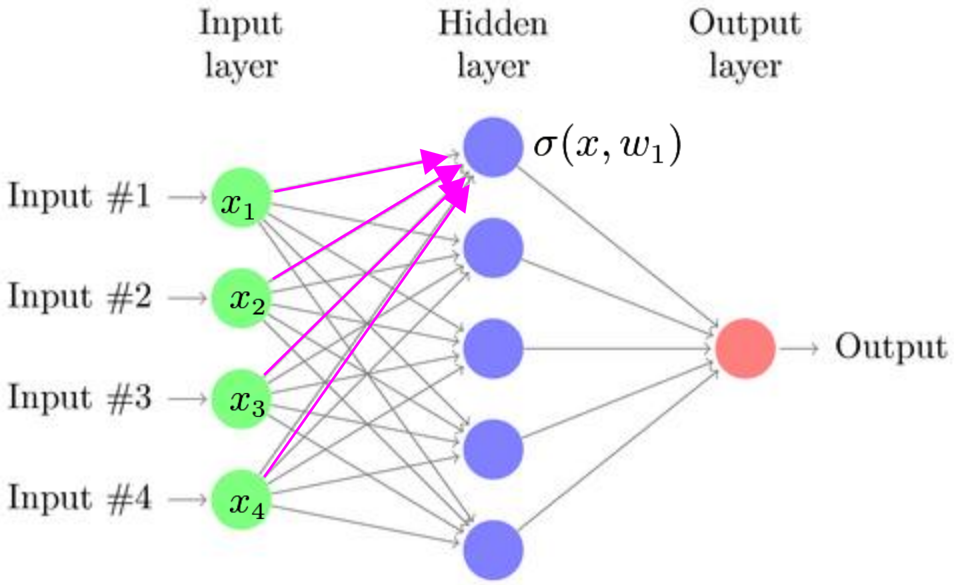
$$f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$$



With simplifications, our target functions f are...

$\sigma(x, w_1)$ is called the **transfer** function

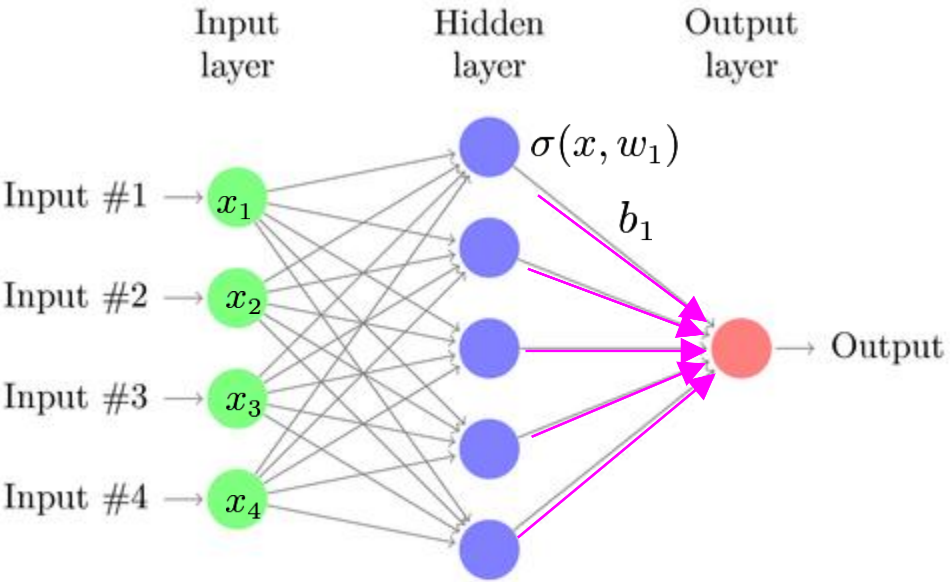
$$f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$$



With simplifications, our target functions f are...

Linear output

$$f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$$



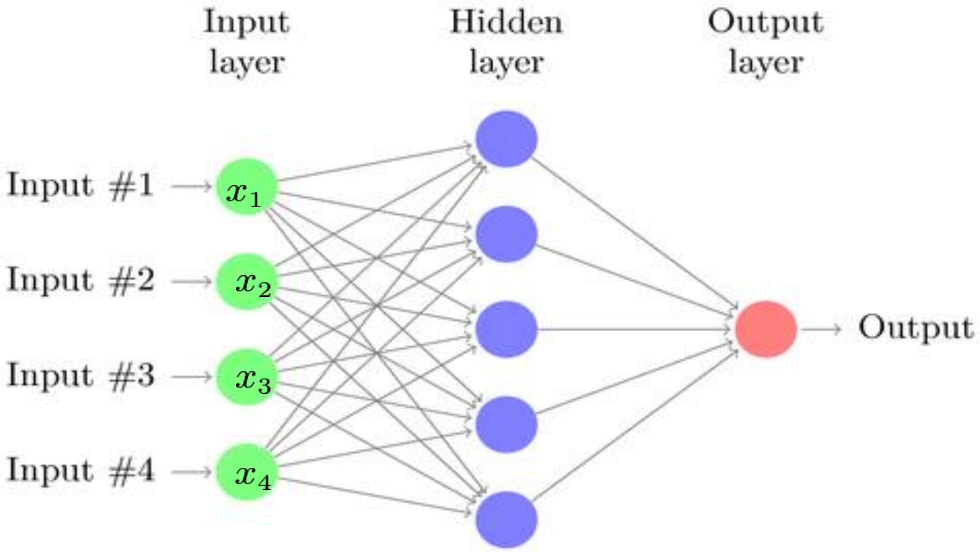
Loss function

Given our current guess of the weights a_i, θ_i and an input x, we measure loss with the squared difference

Guess: $\widehat{f}(x) = \sum_{i=1}^{n} a_i \sigma(x, \theta_i)$

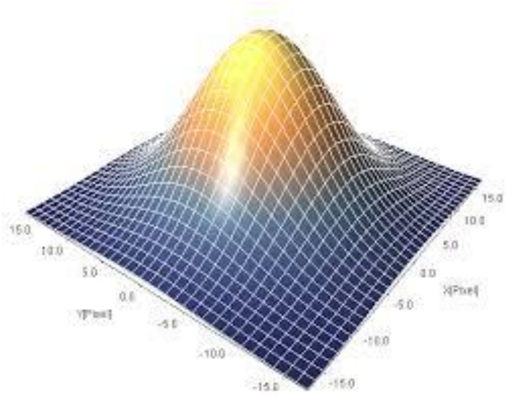
Truth: $f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$

Loss: $(\widehat{f}(x) - f(x))^2$



Training Data: Random, not Adversarial

- Adversarial training data: Makes learning **NP-hard** (also not realistic)
- Assume training data distribution **standard Gaussian**



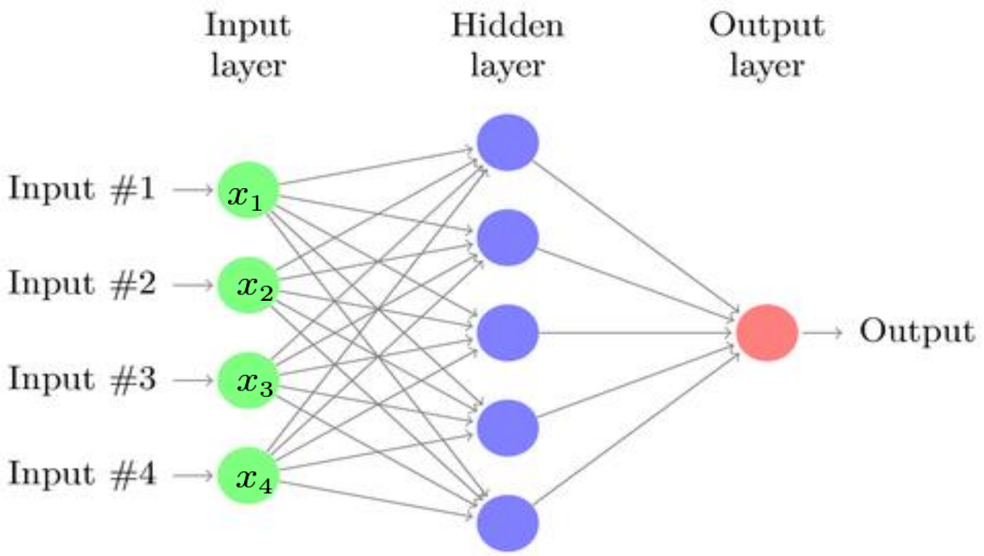
Expected Loss function

Given our current guess of the weights a_i, θ_i, we analyze SGD as GD on the expected loss under infinite data/training time,

Guess: $\widehat{f}(x) = \sum_{i=1}^{n} a_i \sigma(x, \theta_i)$

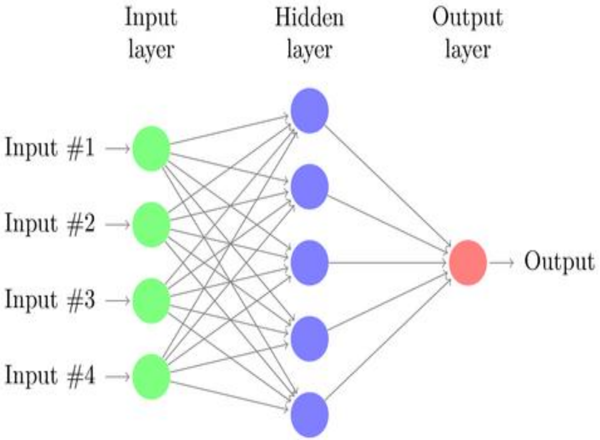
Truth: $f(x) = \sum_{i=1}^{n} b_i \sigma(x, w_i)$

Exp. Loss: $L(a, \theta) = E_X[(\hat{f}(X) - f(X))^2]$

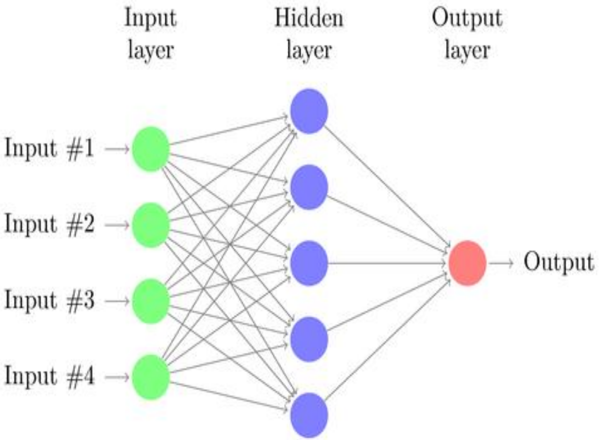


Overview of Results

1) GD dynamics equivalent to variant of electron-proton dynamics (assume a_i, b_i = 1 for simplicity)



$$f(x) = \sum_{i=1}^{n} \sigma(x, w_i)$$

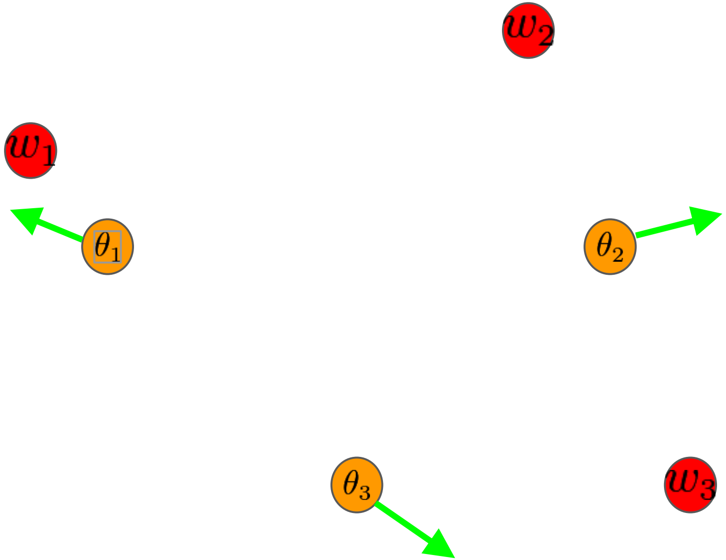


$$\hat{f}(x) = \sum_{i=1}^{n} \sigma(x, \theta_i)$$

Running GD on $L(\theta) = E_X[(f(X) - \hat{f}(X))^2]$ *is...*

Overview of Results

1) GD dynamics equivalent to variant of electron-proton dynamics (assume a_i, b_i = 1 for simplicity)



$$E_X[(f(X) - \hat{f}(X))^2] = E_X[(\sum_i \sigma(X, \theta_i) - \sum_i \sigma(X, w_i))^2]$$

$$= \sum_{ij} E_X[\sigma(X,\theta_i)\sigma(X,\theta_j)] + \sum_{ij} E_X[\sigma(X,w_i)\sigma(X,w_j)] - 2\sum_{ij} E_X[\sigma(X,\theta_i)\sigma(X,w$$

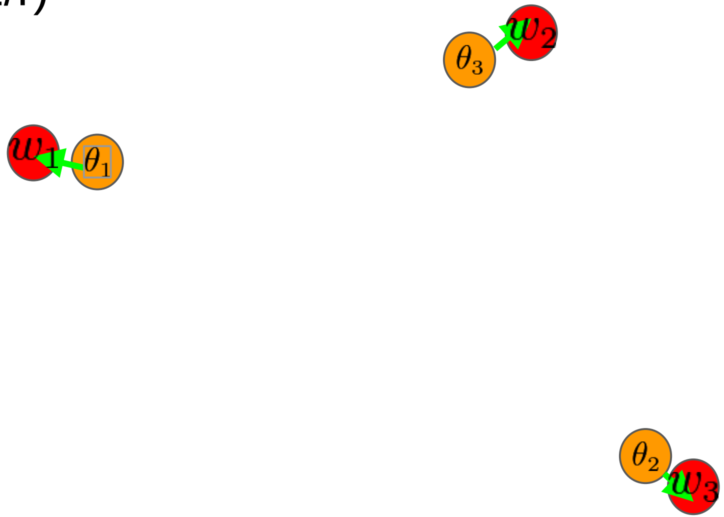
$$= \sum_{ij} \Phi(\theta_i, \theta_j) + \sum_{ij} \Phi(w_i, w_j) - 2\sum_{ij} \Phi(\theta_i, w_j)$$

Where $\Phi(\theta, w) = E_X[\sigma(X,\theta)\sigma(X,w)]$ is the **potential** function, and can be interpreted as a similarity measure

Electron-Proton dynamics under some potential! (depends on transfer function)

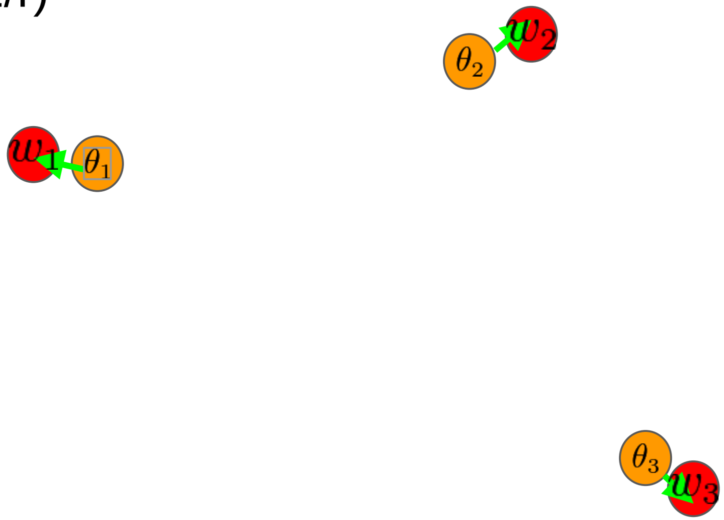
Overview of Results

2) Electron-proton dynamics matches up electrons with protons under natural electric potential (=1/r)



Overview of Results

2) Electron-proton dynamics matches up electrons with protons under natural electric potential (=1/r)



But natural electric potential has no corresponding transfer function!

Overview of Results

3) For many transfers/potentials, electron-proton interactions matches up electrons with protons under varying assumptions

Main Takeaways:

- Analyze GD by equivalently studying electron proton dynamics.
- If electrons match up with protons for some potential, then SGD learns neural networks with the corresponding transfer function.
- We study the electron proton dynamics for different potentials

Common Transfer to Potentials

Name	Transfer $(\sigma(x,\theta))$	Potential $(\Phi(\theta,w))$	Res.
Sign	$\text{sgn}(x^T\theta)$	$1-2\cos^{-1}(\theta^T w)/\pi$	Y
ReLU	$\max(x^T\theta,0)$	$\sqrt{1-(\theta^T w)^2} + \theta^T w(\pi-\cos^{-1}(\theta^T w))$	N
Hermite	$H_m(x^T\theta)$	$(\theta^T w)^m$	Y
Exponential	$\exp(x^T\theta)$	$\exp(\theta^T w)$	Y
Gaussian	$\exp((2x^T\theta-\theta^T\theta)/\sigma)$	$\exp(-\|\theta-w\|_2^2/\sigma)$	Y
Bessel	$\exp(x^T x)\prod_i \frac{\sqrt{2}}{\pi}K_0(\|x_i-\theta_i\|/\sigma)$	$\exp(-\|\theta-w\|_1/\sigma)$	Y

Overview of Results

Transfer: $\sigma(x,\theta) = sgn(x^T\theta)$, $\|\theta\| = \|w\| = 1$

Potential: $\Phi(\theta, w) = 1 - 2\cos^{-1}(\theta^T w)/\pi$

Assumptions:

- Small input or hidden layer size
- Coordinate Gradient Descent (initialize and move electrons one by one)

Overview of Results

Transfer: $\sigma(x, \theta) = e^{(2x^T\theta - \theta^T\theta)}$

Potential: $\Phi(\theta, w) = e^{-\|\theta - w\|^2}$

Assumptions:

- All output weights are 1
- Coordinate Gradient Descent

Overview of Results

Transfer: *Sum of Hermite Polynomials* $\|\theta\| = \|w\| = 1$

Potential: *Truncation of Legendre Function*

Assumptions:

1) **GD dynamics equivalent to electron-proton dynamics under some potential function**

1) Electron-proton interactions matches up electrons with protons under natural electric potential

1) For many transfers/potentials, electron-proton interactions matches up electrons with protons under varying assumptions

Formula for Expected Loss

$$L(a, \theta) = E[(\widehat{f}(X) - f(X))^2]$$

Formula for Expected Loss

$$L(a, \theta) = E[(\widehat{f}(X) + f(X))^2]$$

Formula for Expected Loss

$$L(a, \theta) = E[(\widehat{f}(X) + f(X))^2]$$

$$= E[\hat{f}(X)^2 + 2f(X)\hat{f}(X) + f(X)^2]$$

Formula for Expected Loss

$$\begin{aligned} L(a,\theta) &= E[(\widehat{f}(X) + f(X))^2] \\ &= E[\hat{f}(X)^2 + 2f(X)\hat{f}(X) + f(X)^2] \\ &= E[\hat{f}(X)^2] + 2E[f(X)\hat{f}(X)] + E[f(X)^2] \end{aligned}$$

Formula for Expected Loss

$$
\begin{aligned}
L(a, \theta) &= E[(\widehat{f}(X) + f(X))^2] \\
&= E[\hat{f}(X)^2 + 2f(X)\hat{f}(X) + f(X)^2] \\
&= E[\hat{f}(X)^2] + 2E[f(X)\hat{f}(X)] + E[f(X)^2]
\end{aligned}
$$

Formula for Expected Loss

$$E[\hat{f}(X)^2] = E[(\sum_i a_i \sigma(X, \theta_i))^2]$$

Formula for Expected Loss

$$E[\hat{f}(X)^2] = E[(\sum_i a_i \sigma(X, \theta_i))^2]$$

$$= \sum_i a_i^2 E[\sigma(X, \theta_i)^2] + 2 \sum_{i<j} a_i a_j E[\sigma(X, \theta_i)\sigma(X, \theta_j)]$$

Formula for Expected Loss

$$E[\hat{f}(X)^2] = E[(\sum_i a_i \sigma(X, \theta_i))^2]$$

$$= \sum_i a_i^2 E[\sigma(X, \theta_i)^2] + 2 \sum_{i<j} a_i a_j E[\sigma(X, \theta_i)\sigma(X, \theta_j)]$$

$$= \sum_i a_i^2 \Phi(\theta_i, \theta_i) + 2 \sum_{i<j} a_i a_j \Phi(\theta_i, \theta_j)$$

Where $\Phi(\theta, w) = E_X[\sigma(X, \theta)\sigma(X, w)]$ is the **potential** function, and can be interpreted as a similarity measure

Formula for Expected Loss

$$E[\hat{f}(X)^2] = E[(\sum_i a_i \sigma(X, \theta_i))^2]$$

$$= \sum_i a_i^2 E[\sigma(X, \theta_i)^2] + 2 \sum_{i \neq j} a_i a_j E[\sigma(X, \theta_i) \sigma(X, \theta_j)]$$

$$= \sum_i a_i^2 \Phi(\theta_i, \theta_i) + 2 \sum_{i \neq j} a_i a_j \Phi(\theta_i, \theta_j)$$

$$E[\hat{f}(X) f(X)] = E[(\sum_i a_i \sigma(X, \theta_i))(\sum_j b_j \sigma(X, w_j))]$$

$$= \sum_{ij} a_i b_j \Phi(\theta_i, w_j)$$

Formula for Expected Loss

Putting it all together:

$$L(a,\theta) = \sum_{i=1}^{n} a_i^2 \Phi(\theta_i,\theta_i) + 2\sum_{i<j} a_i a_j \Phi(\theta_i,\theta_j) + 2\sum_{i=1}^{n}\sum_{j=1}^{n} a_i b_j \Phi(\theta_i, w_j)$$

Formula for Expected Loss

Putting it all together:

$$L(a,\theta) = \sum_{i=1}^{n} a_i^2 \Phi(\theta_i,\theta_i) + 2\sum_{i<j} a_i a_j \Phi(\theta_i,\theta_j) + 2\sum_{i=1}^{n}\sum_{j=1}^{n} a_i b_j \Phi(\theta_i, w_j)$$

Further simplify by fixing $b_i = 1 \text{ and } a_i = -1$ (can be interpreted as charges)

$$L(a,\theta) = \sum_{i} \Phi(\theta_i,\theta_i) + 2\sum_{i<j} \Phi(\theta_i,\theta_j) - 2\sum_{i=1}^{n}\sum_{j=1}^{n} \Phi(\theta_i, w_j)$$

Formula for Expected Loss

Putting it all together:

$$L(a,\theta) = \sum_{i=1}^{n} a_i^2 \Phi(\theta_i,\theta_i) + 2\sum_{i<j} a_i a_j \Phi(\theta_i,\theta_j) + 2\sum_{i=1}^{n}\sum_{j=1}^{n} a_i b_j \Phi(\theta_i, w_j)$$

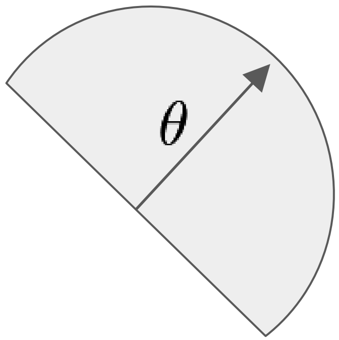
Further simplify by fixing $b_i = 1 \text{ and } a_i = -1$ (can be interpreted as charges)

$$L(a,\theta) = \sum_{i} \Phi(\theta_i,\theta_i) + 2\sum_{i<j} \Phi(\theta_i,\theta_j) - 2\sum_{i=1}^{n}\sum_{j=1}^{n} \Phi(\theta_i, w_j)$$

Minimize pairwise similarity between theta’s and maximize pairwise similarity between theta’s and w’s

Transfer function to Potential function

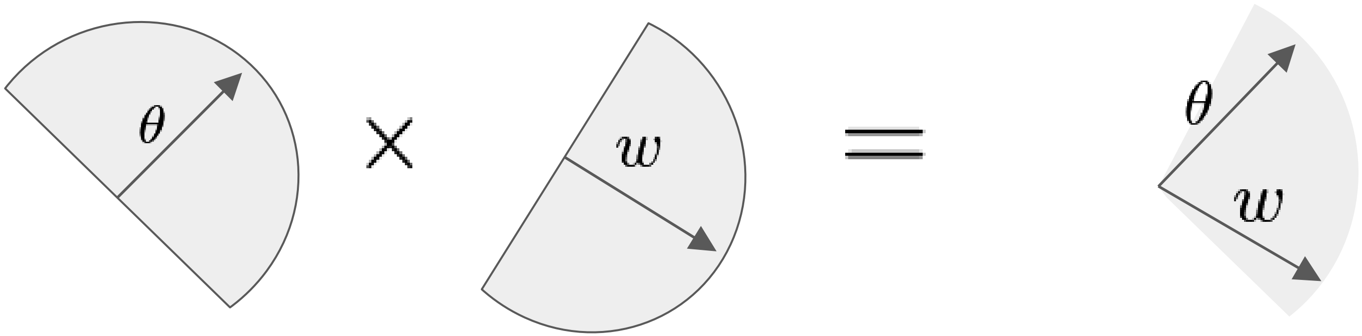
Consider the 0-1 sign transfer $\sigma(x^T\theta) = \mathbf{1}_{\mathbf{x^T}\theta\geq\mathbf{0}}$ and $||\theta|| = 1$



Transfer function to Potential function

Consider the 0-1 sign transfer $\sigma(x^T\theta) = \mathbf{1}_{\mathbf{x^T}\theta\geq\mathbf{0}}$ and $\|\theta\| = 1$

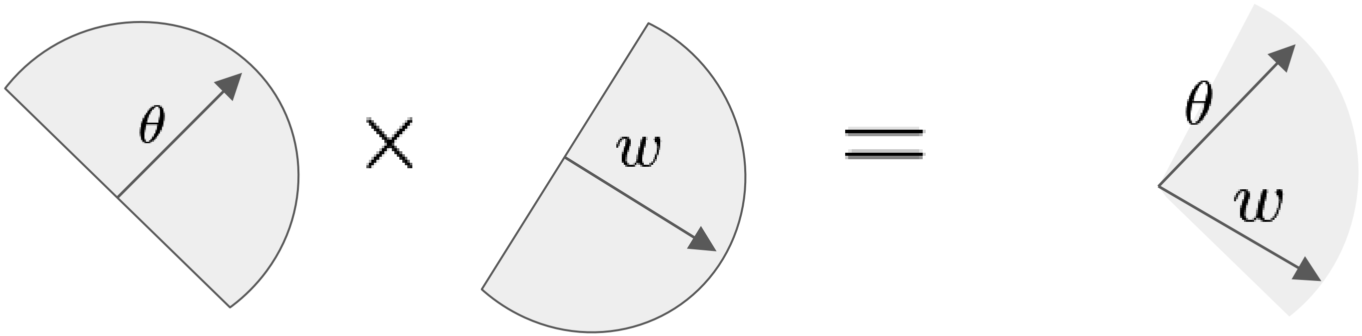
Consider the product $\sigma(w^Tx)\sigma(\theta^Tx)$



Transfer function to Potential function

Consider the 0-1 sign transfer $\sigma(x^T\theta) = \mathbf{1}_{\mathbf{x^T}\theta \geq \mathbf{0}}$ and $||\theta|| = 1$

Therefore, $\Phi(\theta, w) = E_X[\sigma(X^T\theta)\sigma(X^Tw)] = \frac{1}{2} - \frac{\cos^{-1}(\theta^T w)}{2\pi}$



GD Dynamics

Consider the pairwise potential between θ_i and w_j

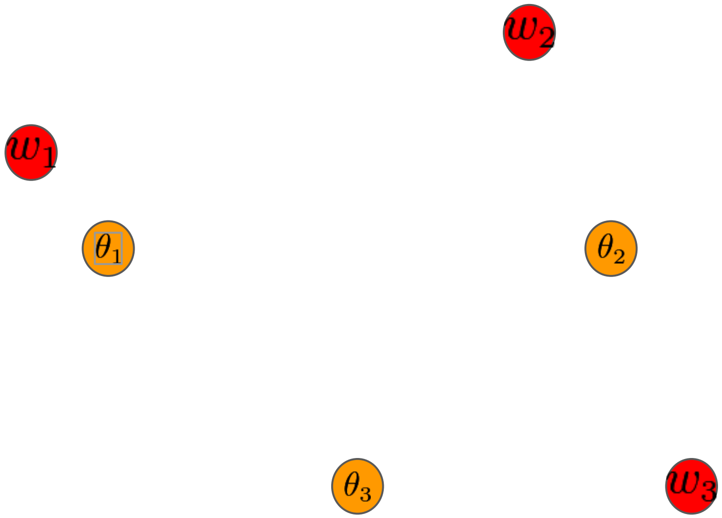
GD Dynamics

Consider the pairwise potential between θ_i and w_j

GD will induce a force that moves theta_i in the direction of maximum increase to the similarity (note w_j is fixed)

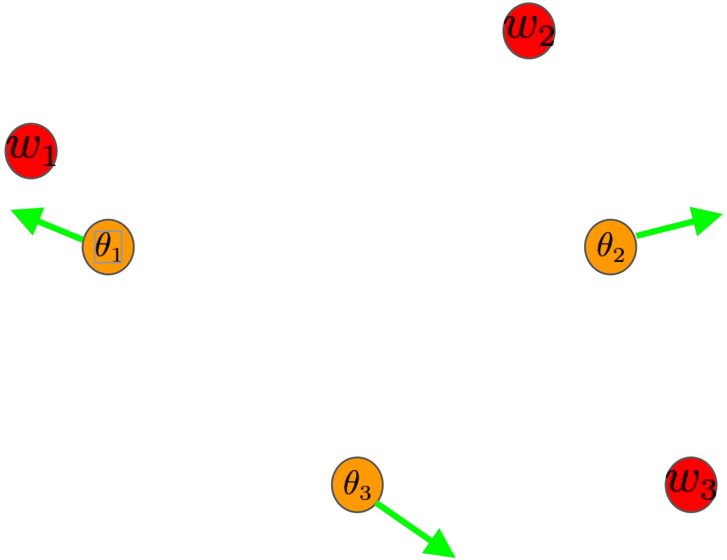
GD Dynamics

In the case of the electric potential, this exactly to corresponds to electrodynamics with fixed protons at $w_1, \ldots, w_n$ and moving electrons at $\theta_1, \ldots, \theta_n$



GD Dynamics

In the case of the electric potential, this exactly to corresponds to electrodynamics with fixed protons at $w_1, \ldots, w_n$ and moving electrons at $\theta_1, \ldots, \theta_n$



1) GD dynamics equivalent to electron-proton dynamics under some potential function

1) **Electron-proton interactions matches up electrons with protons under natural electric potential**

1) For many transfers/potentials, electron-proton interactions matches up electrons with protons under varying assumptions

Earnshaw's Theorem

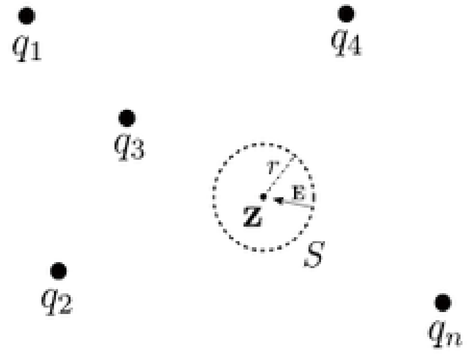
Under the electric potential in 3D, $\Phi(\theta, w) = 1/\|\theta - w\|$

Earnshaw's Theorem guarantees convergence

Theorem 1 (Earnshaw) *A collection of distinct point charges cannot be in stable equilibrium under electrostatic forces.*

Earnshaw's Theorem

Proof: Consider charges at $q_1, \ldots, q_n$ and equilibrium at point z



$$\int_S \mathbf{E} \cdot d\mathbf{A} = \int_V \nabla \cdot \mathbf{E}\, dV = \frac{Q_{\text{enc}}}{\epsilon_0} < 0$$

Earnshaw's Theorem

Proof (Alternate): By the divergenceless property of the electrical potential,

$$\nabla \cdot (-\nabla\Phi) = -\Delta\Phi = -tr(\nabla^2\Phi) = 0$$

A local minima must have a Hessian with positive eigenvalues, which implies a positive trace. Therefore, there is no local minima anywhere!

Can we get electric potential?

Are there transfer functions that give rise to electric potential? **NO, not realizable**

Why? They are **discontinuous and unbounded**.

Can we get electric potential?

Are there transfer functions that give rise to electric potential? **NO, not realizable**

Why? They are **discontinuous and unbounded**.

Main Question(s):

1) Are there other potential properties also give good convergence? **YES**
2) Are there realizable potentials with such properties? **YES**

1) GD dynamics equivalent to electron-proton dynamics under some potential function

1) Electron-proton interactions matches up electrons with protons under natural electric potential

1) **For many transfers/potentials, electron-proton interactions matches up electrons with protons under varying assumptions**

Example: Learning Sums of Gaussian Kernels

- $\Phi(\theta, w) = e^{-\|\theta - w\|_2^2}$
- **Nice property:** Laplacian is positive outside a 2-radius circle of w
- **Claim:** At local minimum, an electron is within a 2-radius circle of a proton

Example: Learning Sums of Gaussian Kernels

- $\Phi(\theta, w) = e^{-\|\theta - w\|_2^2}$
- **Nice property:** Laplacian is positive outside a 2-radius circle of w
- **Claim:** At local minimum, an electron is within a 2-radius circle of a proton

Proof Outline:

- Consider a clumped perturbation of the electrons in a single direction
- Change in the objective function is strictly electron-proton interactions
- If all electrons are far away from protons, then the perturbation creates second-order decrease, so not local minimum

Example: Learning Sums of Gaussian Kernels

- We use coordinate gradient descent and assume that each iteration will run until convergence to a local minima

Algorithm 1 (Coordinate Gradient Descent)

Iterate over thetas: For $i = 1$ through k: start with a new θ_i randomly initialized and perform gradient descent on this θ_i.

Example: Learning Sums of Gaussian Kernels

Theorem 2: *If w_i are initialized according to a Gaussian with mean 0 and variance $\Omega(\log n)$, then with high probability, coordinate gradient descent converges to the global minimum.*

<u>**Proof Outline**</u>:

- First electron must be within a 2-neighborhood of some proton
- By the gradient, the electron is within a 1/poly(n)-neighborhood
- The electron-proton pair largely cancels and it reduces to n-1 protons
- Then, the next electron will pair with one of the remaining protons and so on

More Realistic Results

Main Question: For non-fixed output layer weights, does there exist potentials that have convergence results?

(*Rephrase*) Does convergence results apply to electrodynamics with varying charges?

Positive Laplacian Eigenfunctions

Answer: Yes!

Definition: *A potential* Φ *is a positive eigenfunction of the Laplacian operator if there exists* $\lambda > 0$ *such that*

$$\Delta_\theta \Phi(\theta, w) = \lambda \Phi(\theta, w)$$

Convergence Results

Theorem 3: Let Φ be a positive Laplacian eigenfunction and $L(a, \theta)$ is differentiable with respect to θ_i at θ_i^*, then θ_i^* is not a robust local minimum.

Corollary 2: Let Φ be a positive Laplacian eigenfunction and $\Phi(\theta_i, w_j)$ is non-differentiable with respect to θ_i only at w_j, then at convergence, either $\theta_i = \theta_j$ for some $j \neq i$ or $\theta_i = w_j$ for some j.

Summary

- Analyzed correspondence between transfer and potentials
- GD can be interpreted as the physical model of electrodynamics
- Discovered classes of realizable potentials with good convergence properties under the fixed and non-fixed output weight regime
- Have partial results for the sign and polynomial transfer functions

Summary

- Analyzed correspondence between transfer and potentials
- GD can be interpreted as the physical model of electrodynamics
- Discovered classes of realizable potentials with good convergence properties under the fixed and non-fixed output weight regime
- Have partial results for the sign and polynomial transfer functions

<u>Can convergence results be extended to:</u>

- Widely used transfers? (sigmoid, ReLU, etc.)
- Higher depth neural networks?
- Less assumptions?

Learning a unknown function

- Given $(\boldsymbol{X_i}, \boldsymbol{y_i})$ input output pairs
  - learn polynomial f so that $f(\boldsymbol{X_i}) = \boldsymbol{y_i}$

Learn f so that you can predict its output on new inputs

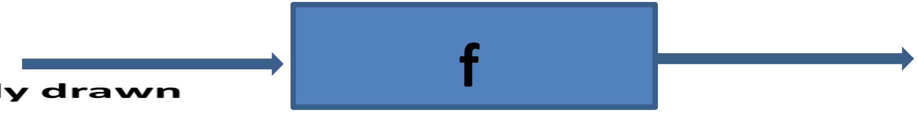
Learning a function

- Given (X_i, y_i) input output pairs
  - learn polynomial f so that $f(X_i) = y_i$

$X_i \in \Re^n$ Randomly drawn

$Y_i \in \Re$

- Known degree d polynomials
  - Can be learnt in time n^d
- Our Result: m-sparse polynomial
  - Can be learnt in time $O(d^d)\, poly(m, d, n)$



Learn f so that you can predict its output on new inputs

- Given (X_i, y_i) input output pairs
  - learn polynomial f so that $f(X_i) = y_i$

- Want to make predictions in real life situations?
- Will a user click on an ad?
- User features encoded by a vector X_i: Earlier queries.
- Predict y_i probability of clicking on an ad.
- Given (X_i, y_i) learn a function f so that
- f(X_i)=y_i

Learning a function

- Given (X_i, y_i) input output pairs:
  - learn polynomial f so that $f(X_i) = y_i$
- What function classes to choose:A simple model for f: linear regression
- Logistic regression
- Deep learning
  - Useful in many engineering applications

Learning a function

- Given (X_i, y_i) input output pairs:
  - learn polynomial f so that $f(X_i) = y_i$
- What function classes to choose:A simple model for f: linear regression
- Logistic regression
- Deep learning
  - Useful in many engineering applications