EECS E6690: SL for Bio & Info
Lecture 9: On Unified Supervised Learning Theory, Neural Networks and Deep Learning

Prof. Predrag R. Jelenković

Time: Tuesday 4:10-6:40pm

Dept. of Electrical Engineering
Columbia University , NY 10027, USA
Office: 812 Schapiro Research Bldg.
Phone: (212) 854-8174
Email: predrag@ee.columbia.edu
URL: http://www.ee.columbia.edu/∼predrag

On Unified Supervised Learning Theory

Supervised learning: Given training data $(\boldsymbol{x}, \boldsymbol{y})$, i.e.,

$$(x_1, x_2, \ldots, x_n) \rightarrow \boxed{\boldsymbol{f}(x)} \rightarrow (y_1, y_2, \ldots, y_n)$$

Problem:

- Don't know $\boldsymbol{f}$
- Find the "best" approximation $\hat{\boldsymbol{f}}$

What have we seen?

- Linear Ridge: $\hat{f}(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$

$$\min_{\beta_0, \boldsymbol{\beta}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- Lasso: same as above, just change the penalty to ℓ_1 norm:

$$\lambda \sum_{j=1}^{p} \beta_j^2 \rightarrow \lambda \sum_{j=1}^{p} |\beta_j|$$

On Unified Supervised Learning Theory

Basis expansion:

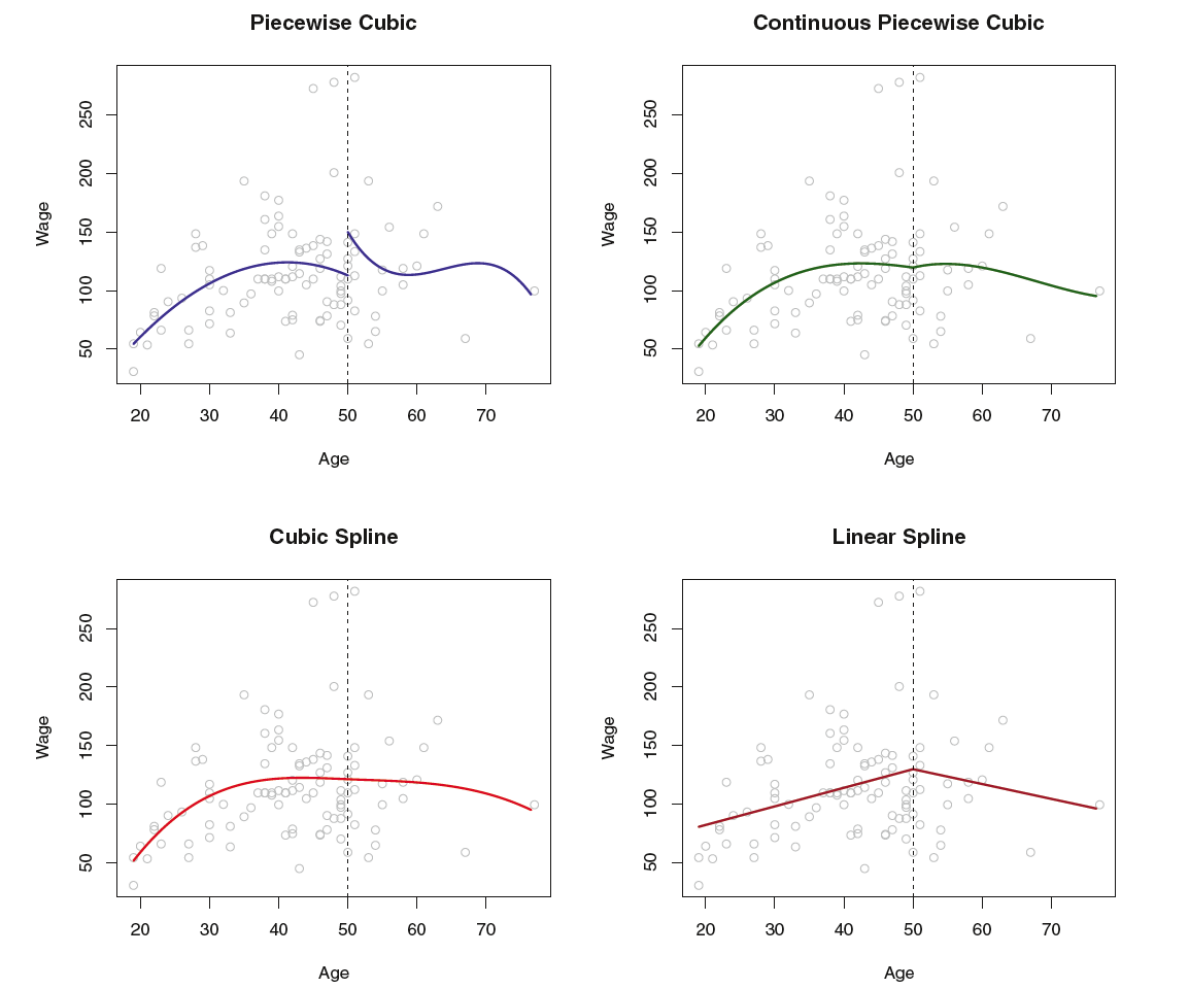
- Polynomial Ridge or Lasso: Same as before, but more general, polynomial, approximation function

$$\hat{f}(X) = \beta_0 + \sum_{j=1}^{p} \sum_{l=1}^{k} \beta_{jl} (X_j)^l$$

- Splines: Pick $\hat{f}(X)$ to be piecewise polynomial, e.g., piecewise linear. Problem: discontinuities.
- Smoothing Splines: Impose continuity and derivative constrains.

Example: Splines

Top left: cubic - no constraint; Top right: cubic and continuous
Bottom left: cubic - continuous, with continuous $\hat{f}'$ and $\hat{f}''$
Bottom right: linear continuous



Hinge Loss Formulation of SVC

Recall

$$\min_{\beta_j, \epsilon_j} \frac{\|\boldsymbol{\beta}\|^2}{2}$$

$$\text{subject to} \quad y_i(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + \beta_0) \geq (1 - \epsilon_i), \qquad \forall i$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C$$

Or, equivalently

$$\min_{\beta_i, \epsilon_i} \frac{\|\boldsymbol{\beta}\|^2}{2} + c \sum_{i=1}^{n} \epsilon_i$$

$$\text{subject to} \quad \epsilon_i = \max(0, 1 - y_i(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + \beta_0))$$

Implying

$$\min_{\boldsymbol{\beta}, \beta_0} \frac{\|\boldsymbol{\beta}\|^2}{2} + c \sum_{i=1}^{n} \max(0, 1 - y_i(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + \beta_0))$$

$$\Leftrightarrow \min_{\boldsymbol{\beta}, \beta_0} \sum_{i=1}^{n} \max(0, 1 - y_i(\langle \boldsymbol{\beta}, \boldsymbol{x} \rangle + \beta_0)) + \lambda \|\boldsymbol{\beta}\|^2$$

On Unified Supervised Learning Theory

Classification: let $y_i \in \{-1, 1\}$

- Support Vector Classifier: can be obtained by solving

$$\min_{\beta_0, \boldsymbol{\beta}} \sum_{i=1}^{n} \max\left[0, 1 - y_i(\beta_0 + x_{i1}\beta_1 + \cdots + x_{ip}\beta_p)\right] + \lambda \sum_{j=1}^{p} \beta_j^2$$

  $\max\left[0, 1 - y_i(\beta_0 + x_{i1}\beta_1 + \cdots + x_{ip}\beta_p)\right]$ is called **hinge loss**
  For general SVM use a kernel generalization of the hyperplane

- Logistic regression with ℓ_2 penalty: replace the hinge loss in the preceding expression with

$$\left\{y_i(\beta_0 + x_{i1}\beta_1 + \cdots + x_{ip}\beta_p) - \ln\left(1 + e^{\beta_0 + x_{i1}\beta_1 + \cdots + x_{ip}\beta_p}\right)\right\}$$

On Unified Supervised Learning Theory

In general, all the preceding leaning problems can be written as

$$\hat{f}^* = \arg\min_{\hat{f}\in\mathcal{H}} \sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) + \lambda\Omega(\hat{f}) \tag{1}$$

where L is a general loss function, and $\lambda\Omega(\hat{f})$ is the penalty, and $\mathcal{H}$ is the space of approximation functions that we are considering.

- Example: linear functions $\hat{f}(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$
  - $\mathcal{H}$: set off all p-dimensional linear functions
  - Penalty - square of the ℓ_2 norm
    $$\Omega(\hat{f}) = < \hat{f}, \hat{f} > = \|f\|^2 = \sum_{j=1}^{p} \beta_j^2$$
  - Loss: $L(y_i, \hat{f}(x_i)) = (y_i - \hat{f}(x_i))^2$

Q: What is the most general that $\hat{f}, \mathcal{H}, L, \Omega$, can be such that the problem has nice analytical and computational properties?

RKHS: Reproducing Kernel Hilbert Spaces

We have already seen it with SVM: here some more details[1]

- Hilbert space $\{\mathcal{H}\}$: natural generalization of Euclidian spaces.
- It has an inner product: $< f, g >$, for any $f, g \in \mathcal{H}$, which allows computing angles between the elements of $\{\mathcal{H}\}$.
  In particular, $< f, g >= 0$, then f, g are orthogonal.
- Norm/distance: Norm $\|f\| = \sqrt{< f, f >}$, and
  distance $d(f, g) = \|f - g\| = \sqrt{< f - g, f - g >}, \qquad f, g \in \mathcal{H}$
- Euclidian example: if $x, y \in \mathbb{R}^d$, then $< x, y >= x_1 y_1 + \cdots + x_p y_p$

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_p^2}, \quad d(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_p - y_p)^2}$$

- Example - matrices: $< A, B >= \text{trace}(A^\top B)$
- Example - random variable: $< X, Y >= \text{cov}(X, Y)$

RKHS: Reproducing Kernel Hilbert Spaces

RKHS - $\mathcal{H}_k$: Subset of Hilbert spaces with really nice properties

- Kernel is a positive definite function $k(x, y)$
  $(\sum \alpha_i \alpha_j k(x_i, x_j) \geq 0)$
- **Reproducing property**: if $f \in \mathcal{H}_k$, then

$$< f, k(\cdot, x) >= f(x)$$

- Each kernel, $k(x, y)$, defines uniquely the Hilbert space, $\mathcal{H}_k$ Hence, in this space, $\mathcal{H}_k$, all we need to know if the kernel!
- Functions in this space $f(x) \in \mathcal{H}_k$ can be written as

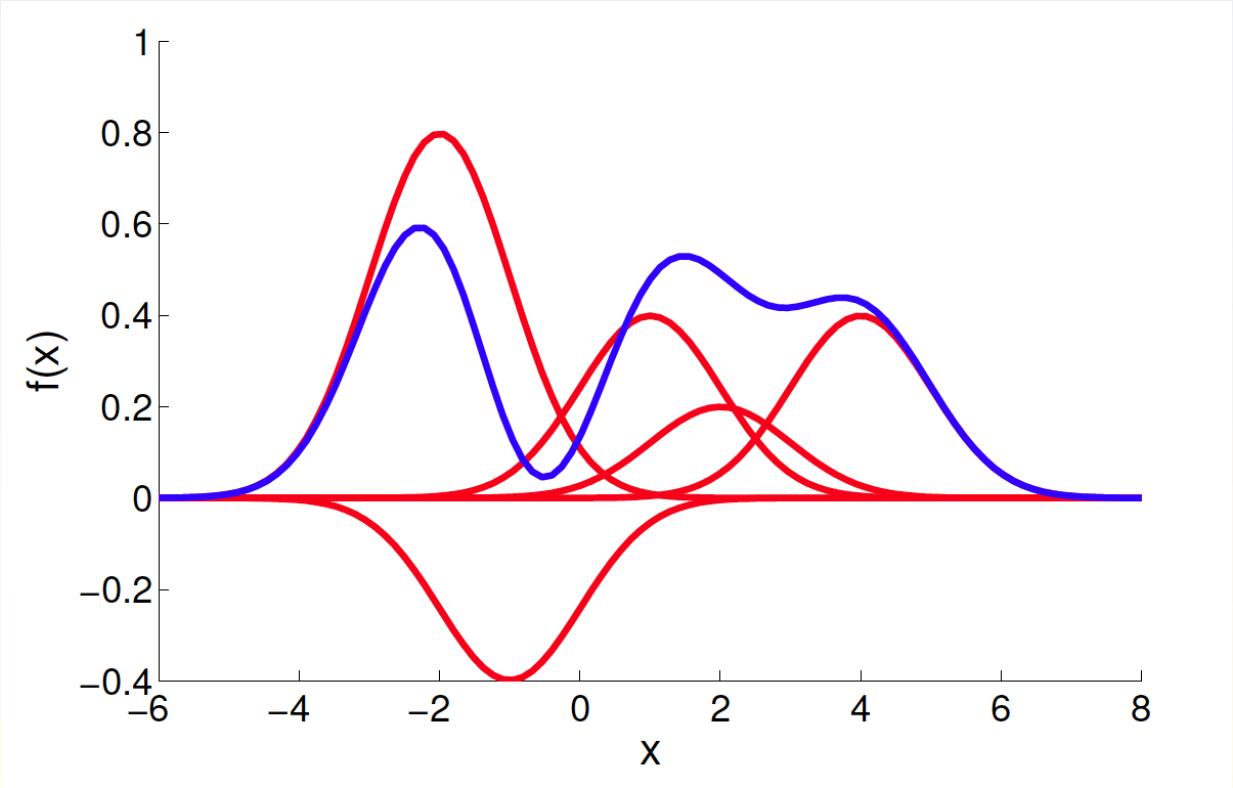
$$f(x) = \sum_i \beta_i k(x, x_i)$$

Example: Gaussian (Radial) Kernel

Gaussian kernel in 1D: $k(x, y) = e^{-\frac{(x-y)^2}{2\sigma}}$

Typical function: sum of weighted Gaussian "blobs" - blue line below

$$f(x) = \sum_i \beta_i k(x, x_i) = \sum_i \beta_i e^{\frac{-(x-x_i)^2}{2\sigma}}$$



$\mathcal{H}_k$ is a linear space, but $f(x)$ is very much nonlinear.

Generalized Learning in RKHS

Here we optimize over all approximation function in $\mathcal{H}_k$

$$\hat{f}^* = \arg\min_{\hat{f} \in \mathcal{H}_k} \sum_{i=1}^{n} L(y_i, \hat{f}(x_i)) + \lambda \Omega(\|\hat{f}\|^2_{\mathcal{H}_k}) \tag{2}$$

Representer Theorem For any loss function L and any strictly increasing function $\Omega : \mathbb{R} \to \mathbb{R}$, an optimal solution, $\hat{f}^*$, of Equation (2) has a form

$$\hat{f}^*(x) = \sum_{i=1}^{n} \beta_i k(x, x_i)$$

$\hat{f}^*$ has a finite representation (!) even though $\mathcal{H}_k$ can be (is) infinite. We can put much of the supervised learning under the same umbrella. **Drawback**: Theoretical framework works only with ℓ_2 norm, $\|\hat{f}\|_{\mathcal{H}_k}$ - doesn't work for Lasso.

Representer Theorem: Proof

Let's drop "hat" from $\hat{f}$ and let f_s be functions spanned by $k(x, x_i)$

$$f_s(x) = \sum_{i=1}^{n} \beta_i k(x, x_i)$$

Then, any function $f \in \mathcal{H}_k$ can be decomposed as

$$f(x) = f_s(x) + f_\perp(x) \tag{3}$$

where $f_\perp(x)$ is perpendicular to f_s. Hence, (Pythagoras theorem for $\mathcal{H}_k$)

$$\|f\|^2_{\mathcal{H}_k} = \|f_s\|^2_{\mathcal{H}_k} + \|f_\perp\|^2_{\mathcal{H}_k} \geq \|f_s\|^2_{\mathcal{H}_k} \tag{4}$$

Next, by kernel reproducing property and orthogonality

$$f(x_i) =< f, k(\cdot, x_i) >=< f_s, k(\cdot, x_i) >= f_s(x) \tag{5}$$

Finally, Equation (4) and Equation (5) imply that an optimizer of Equation (2) must be in the form of $f_s(x)$ from Equation (3) since Ω is increasing and

$$L(y_i, f(x_i)) = L(y_i, f_{\boldsymbol{s}}(x_i)), \qquad \Omega(\|\hat{f}\|^2_{\mathcal{H}_k}) \geq \Omega(\|f_{\boldsymbol{s}}\|^2_{\mathcal{H}_k})$$

Generalized Ridge

For quadratic loss function

$$\hat{f}^* = \arg\min_{\hat{f}\in\mathcal{H}_k} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2 + \lambda \|\hat{f}\|^2_{\mathcal{H}_k} \tag{6}$$

The coefficients, β_i^*, of the optimizer

$$\hat{f}^*(x) = \sum_{i=1}^{n} \beta_i^* k(x, x_i)$$

are explicitly given by

$$\boldsymbol{\beta}^* = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$$

where $\boldsymbol{K}$ is a square matrix with elements $K_{ij} = k(x_i, x_j)$, $\boldsymbol{I}$ is an identity matrix, and $\boldsymbol{y}$ is the column vector $(y_1, \ldots, y_n)^\top$.

Generalized Ridge: Proof

From the Representer Theorem we know that the minimizer has to be of the form

$$f(x) = \sum_{i=1}^{n} \beta_i k(x, x_i)$$

and their quadratic norm is given by

$$\|f\|^2_{\mathcal{H}_k} =< f, f >= f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j k(x_j, x_i) = \boldsymbol{\beta}^\top \boldsymbol{K}\boldsymbol{\beta}$$

since, by reproducing property, $< k(\cdot, x_j), k(\cdot, x_i) >= k(x_j, x_i)$. Also, by reproducing property,

$$f(x_i) =< f(\cdot), k(\cdot, x_i) >= \sum_{j=1}^{n} \beta_j k(x_j, x_i)$$

By replacing the preceding 2 equations in the optimization function in Equation (6)

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \beta_j k(x_j, x_i))^2 + \lambda \boldsymbol{\beta}^\top \boldsymbol{K}\boldsymbol{\beta}$$

which by taking the derivative w.r.t. $\boldsymbol{\beta}$ and setting it to 0 yields

$$\boldsymbol{\beta} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$$

Neural Networks Verus Kernel Methods

- Similarity: both parametric methods
- Difference: fixed basis for kernels, while NN is discovering basis/features

Neural networks:

- Rich $\mathcal{H}$: Provides a versatile parametric class of functions
  - Universal function approximation
  - Difficult to train: non-convex optimization
  - Often accurate predictions, but difficult to interpret
- Automatic feature/basis extraction
  - Traditional Feature Engineering approach: expert constructs feature mapping $\phi : \mathcal{X} \to \Phi$. Then, apply machine learning to find a linear predictor on $\phi(\mathbf{x})$.
  - "Deep learning" approach: neurons in hidden layers can be thought of as features that are being learned automatically from data
  - Shallow neurons corresponds to low level features, while deep neurons correspond to high level features

General Parametric Supervised Learning

- $\mathcal{H}$ is a parametric class of functions $f(w, x), w \in \mathcal{W}, w = (w_1, \ldots, w_k)$
  - Examples: Generalized Ridge, or neural networks
- Finding $f \in \mathcal{H}$ is equivalent to finding $w \in \mathcal{W}$

Hence, our general supervised learning problem can be formulated in terms of w as (say, $x_i \in \mathbb{R}^p, y_i \in \mathbb{R}^m$,$\ell : \mathbb{R}^{p+m} \to \mathbb{R}$ - loss function)

$$\hat{w} = \arg\min_{w \in \mathcal{W}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(w, x_i)) + \lambda R(w) \tag{7}$$

How do we solve the preceding problem?

- When the problem is convex and we are lucky, we can find an explicit $\hat{w}$ by solving (e.g., generalized (Kernel) Ridge regression)

$$\frac{\partial}{\partial w_i} \left(\frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(w, x_i)) + \lambda R(w) \right) = 0, \qquad i = 1, \ldots, k.$$

Parametric Supervised Learning: Numerical Optimization

Let us denote the objective function

$$F(w) := \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(w, x_i)) + \lambda R(w)$$

In general, we use the following numerical algorithm:

1. Initialization: Pick an initial value w_0, possibly random
2. Iteration: Keep updating w in small steps Δw

$$w_{n+1} = w_n + \Delta w,$$

such that $F(w_{n+1}) < F(w_n)$.
Stoping criteria: $F(w_n) - F(w_{n+1}) < \epsilon$.

For the preceding procedure to find a local minimum

- We need to find a direction where F has a maximum decrease/steepest descent
- Avoid getting stuck on a flat surfaces, say flat saddle point

If F is convex, we can find a global minimum.

Recall Multi Calc: Gradient and Directional Derivatives

- Let $f(x) : \mathbb{R}^n \to \mathbb{R}, x = (x_1, \ldots, x_n)$
- **Gradient** is a vector of partial derivatives (assuming they exist)

$$\nabla f(x) := \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

- Let $u = (u_1, \ldots, u_n)$ be a unit vector ($\|u\|_2^2 = \sum u_i^2 = 1$)
- **Directional derivative** of $f(x)$ in direction u is

$$D_u f(x) := \frac{d}{dt} f(x+ut) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} u_i = \nabla f(x) \cdot u = \|\nabla f(x)\|_2 \cos\theta,$$

  where $y \cdot z = \langle y, z \rangle$ is the dot product and θ is the angle between $\nabla f(x)$ and u.
- Hence, $-\|\nabla f(x)\|_2 \leq D_u f(x) \leq \|\nabla f(x)\|_2$, i.e.,
  $\nabla f(x)$: direction of steepest ascent/max increase of $f(x)$
  $-\nabla f(x)$: direction of steepest descent/max decrease of $f(x)$
  $\nabla f(x)$ is perpendicular to level curves $f(x) = c$ (prove this)

Path of Gradient Descent

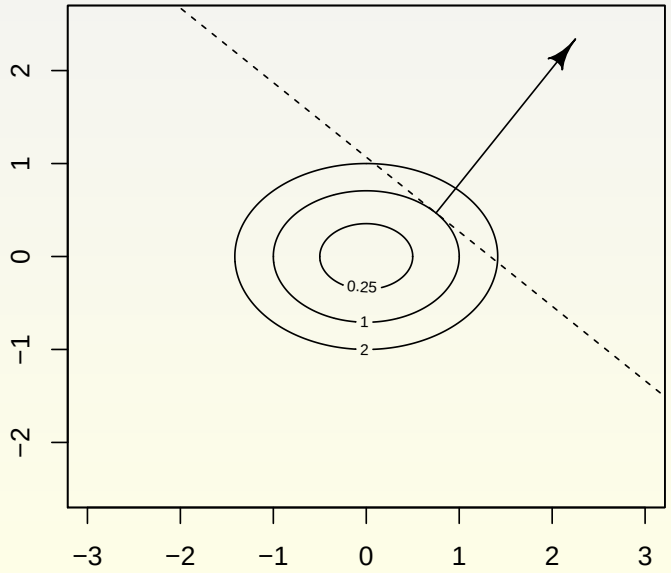
- $\mathbf{x}(t)$ - path of steepest gradient descent given initial condition $\mathbf{x}(0)$ is given by an ODE

$$\frac{d\mathbf{x}(t)}{dt} = -\eta \nabla f(\mathbf{x}(t)), \tag{8}$$

where η is the rate/speed of descent, a.k.a. learning rate. Note that $x'(t)$ is tangent to the curve $x(t)$, and thus parallel to $\nabla f(\mathbf{x}(t))$.

- Example: $f(\mathbf{x}) = ax_1^2 + bx_2^2, a, b > 0, \Rightarrow$
$\nabla f(\mathbf{x}) = (2ax_1, 2bx_2) \Rightarrow x_1'(t) = -2\eta a x_1(t), x_2'(t) = -2\eta b x_2(t)$

$$x_1(t) = x_1(0)e^{-2\eta a t}, \qquad x_2(t) = x_2(0)e^{-2\eta b t}$$



Gradient Descent Algorithm

GD Algorithm is a discrete linear approximation to Equation (8)

$$\frac{\mathbf{x}(t+\Delta t)-\mathbf{x}(t)}{\Delta t} \approx \frac{d\mathbf{x}(t)}{dt} = -\eta \nabla f(\mathbf{x}(t))$$

or equivalently (with a small abuse of notation)

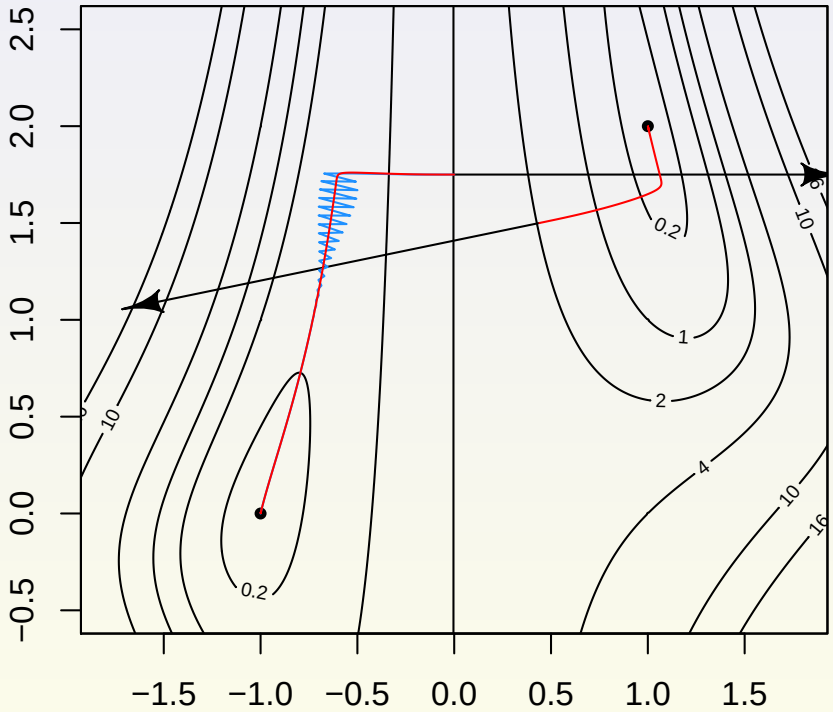
$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla f(\mathbf{x}^{(t)})$$

- Hence, after initialization at $\mathbf{x}^{(0)}$, the GD Algorithm follows the preceding iteration to a local minimum
- Stopping criterion (could be): $|f(\mathbf{x}^{(t+1)}) - f(\mathbf{x}^{(t)})| < \epsilon$

Adaptive GD (AdaGrad): modifies the learning rate η_t in each iteration t

More Interesting Landscape

- $f(x) = (x_1^2 - 1)^2 + (x_1^2 x_2 - x_1 - 1)^2$
- Gradient
$$\nabla f(x) = \begin{bmatrix} 4x_1(x_1^2 - 1) + 2(2x_1x_2 - 1)(x_1^2x_2 - x_1 - 1) \\ 2x_1^2(x_1^2x_2 - x_1 - 1) \end{bmatrix}$$
- Oscillations in "narrow valleys"

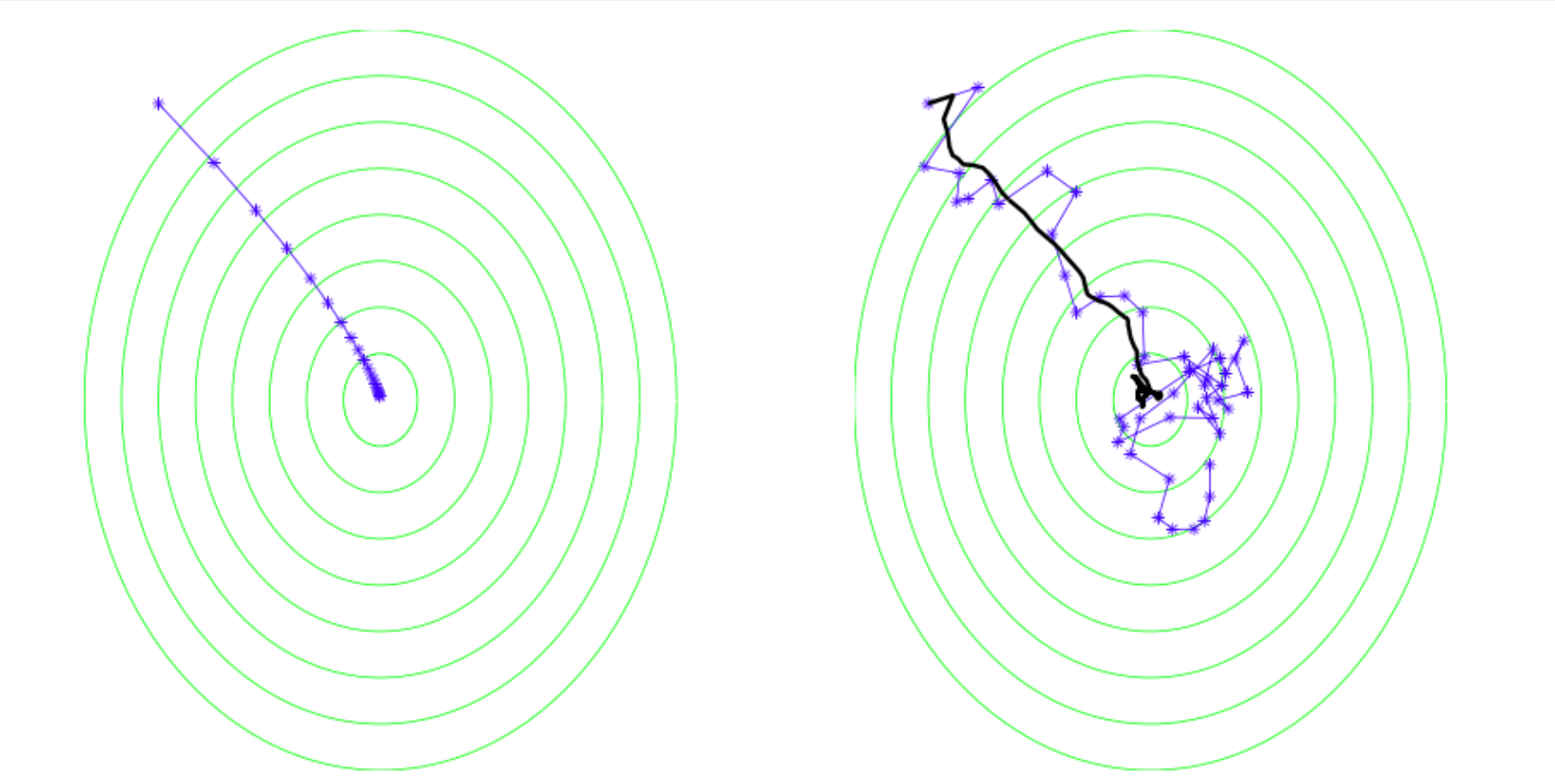


Motivation for momentum: remembers/averages previous Δx

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta_t \nabla f(\mathbf{x}^{(t)}) + \mu_t(\mathbf{x}^{(t)} - \mathbf{x}^{(t-1)})$$

Stochastic Gradient Descent

- Dates back to Robbins and Monroe (1951).
- Stochastic gradient is an unbiased estimator of the gradient
- Stochastic versus regular gradient descent



Stochastic Gradient Descent

- Stochastic approximation of gradient descent
- Function (typically encountered in learning)

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

- Computationally expensive gradient for large n
- Approximation: pick a random subset $\mathcal{S} \in [1, n]$

$$\frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \nabla f_i(x)$$

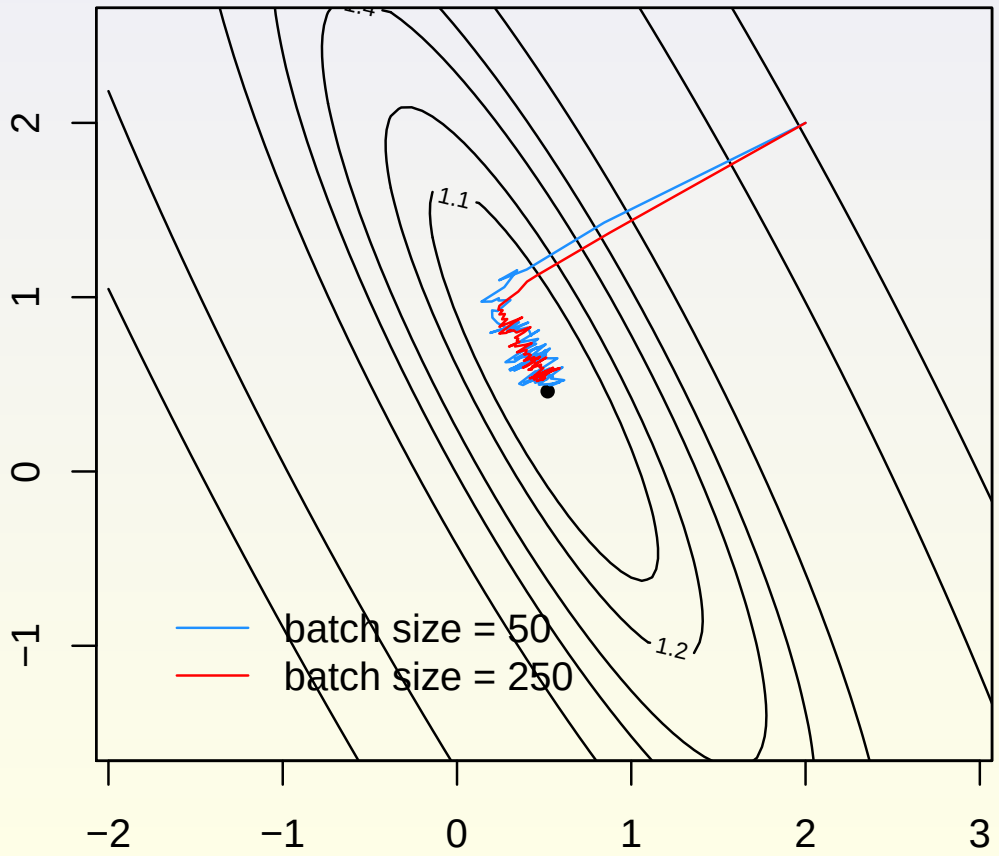
- $\mathcal{S}$ - called batch/mini-batch
- Example
  - n scalar data points $x_1, x_2, \cdots, x_n$
  - objective

$$\min_c \frac{1}{n} \sum_{i=1}^{n} (x_i - c)^2$$

SGD Example: Linear Regression

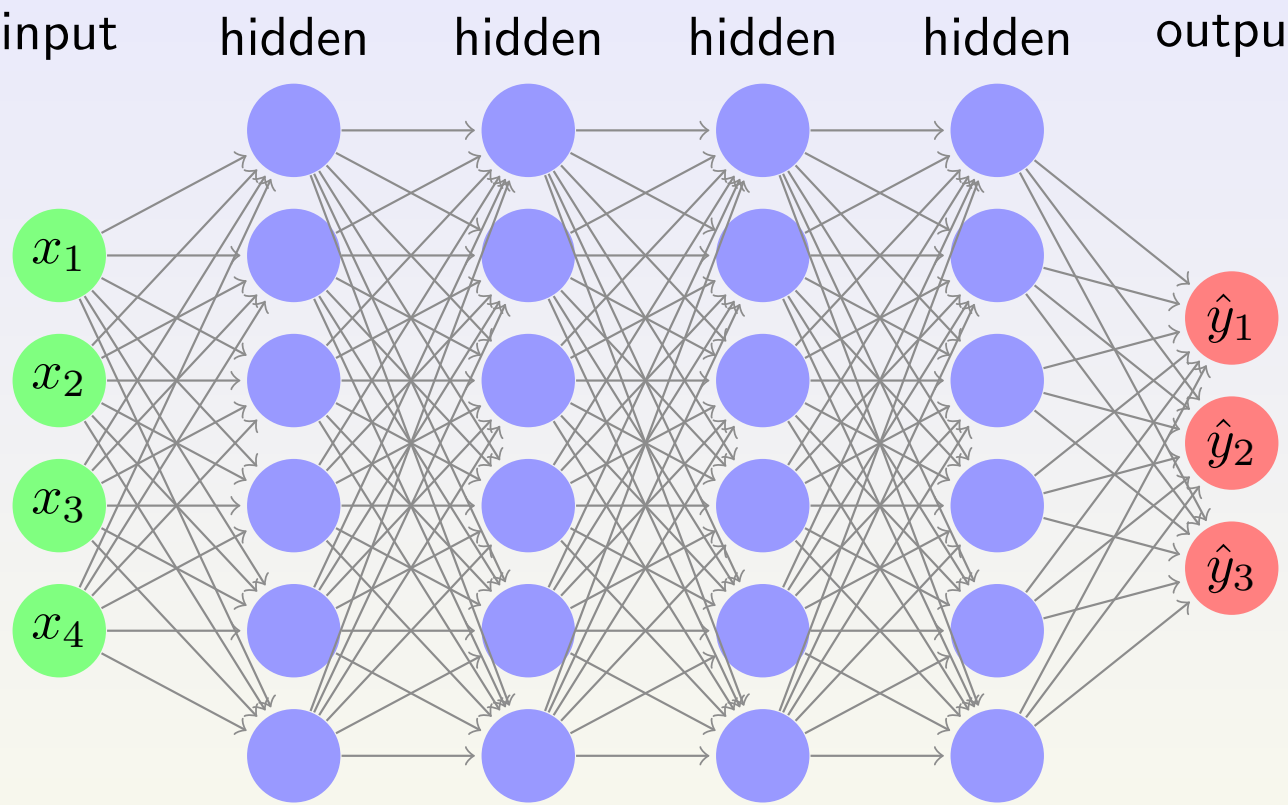
- Data: $(x_i, y_i)_{i=1}^n$, $n = 10^5$
- Loss function: $L(\beta, x, y) = \frac{1}{n}\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2$
- Stochastic gradient

$$\nabla_\beta \hat{L}(\beta, x, y) = \frac{2}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \begin{bmatrix} (\beta_0 + \beta_1 x_i - y_i) \\ x_i(\beta_0 + \beta_1 x_i - y_i) \end{bmatrix}$$



Deep Neural Networks, a.k.a. Multilayer Perceptrons

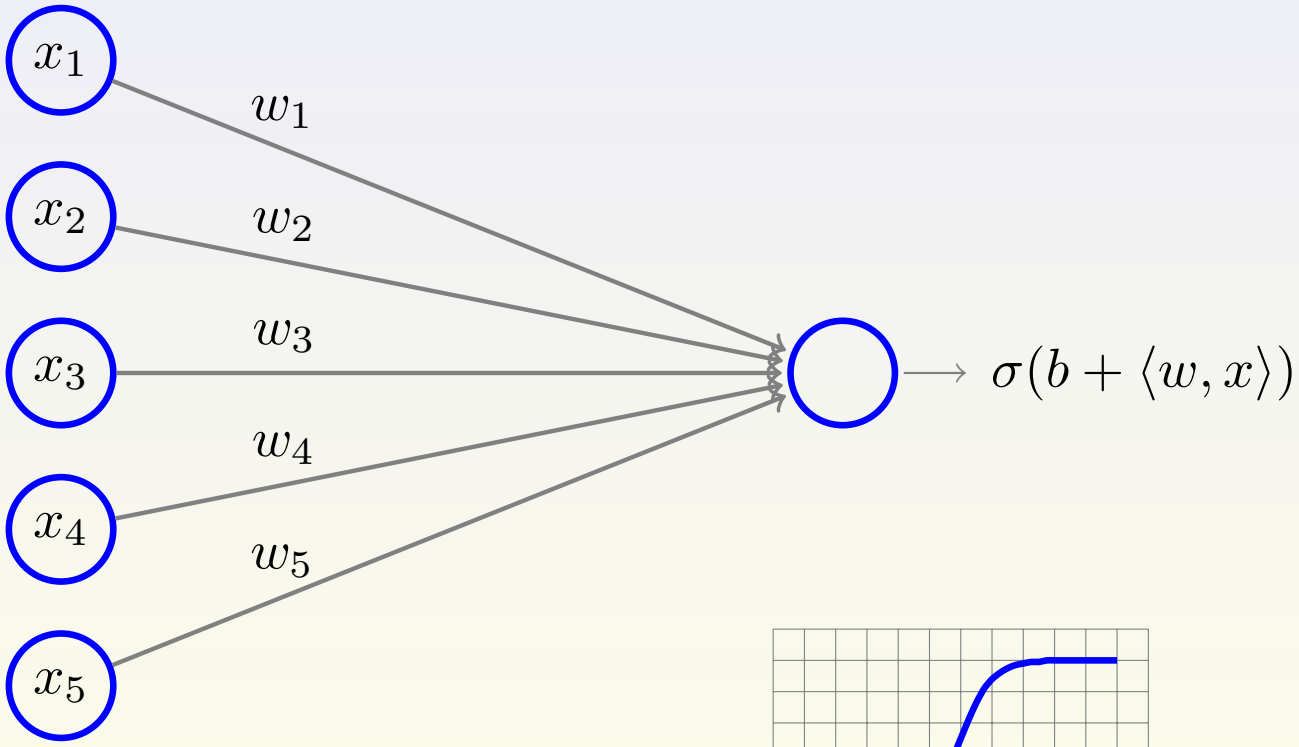
- Feed-forward network



- Designed to mimic the function of neurons
- Blue nodes: activation functions/neurons
- Depth $=$ lengths of a longest path
- Deep network: depth ≥ 3
- Very successful in solving practical problems

A Single Artificial Neuron

- A single neuron function: $\mathbf{x} \mapsto \sigma(b + \langle \mathbf{w}, \mathbf{x} \rangle)$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is called the activation function of the neuron. Inner/dot product: $\langle \mathbf{w}, \mathbf{x} \rangle = \sum x_i w_i$.
- More compact notation $\langle \tilde{\mathbf{w}}, \tilde{\mathbf{x}} \rangle$, where $\tilde{\mathbf{x}} = (1, \mathbf{x}), \tilde{\mathbf{w}} = (b, \mathbf{w})$



- E.g., σ is a sigmoidal function

Common Activation Functions/Perceptrons

- step function

$$\sigma(x) = 1_{\{x>0\}} \qquad \sigma'(x) = 0,\ x \neq 0$$

- logistic

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$

- rectified linear unit (ReLU)

$$\sigma(x) = \max\{x, 0\} \qquad \sigma'(x) = 1_{\{x>0\}},\ x \neq 0$$

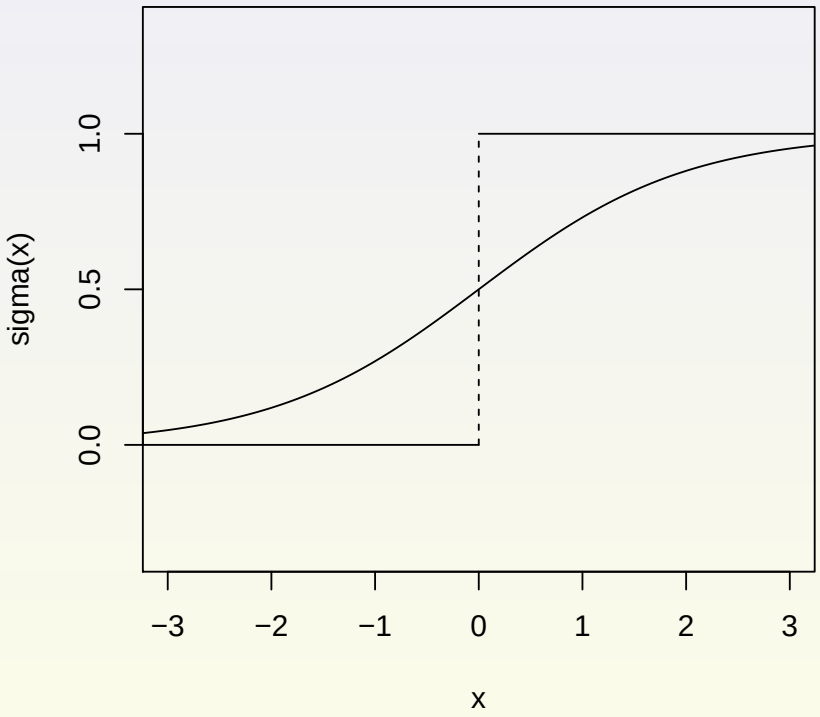
- soft-plus

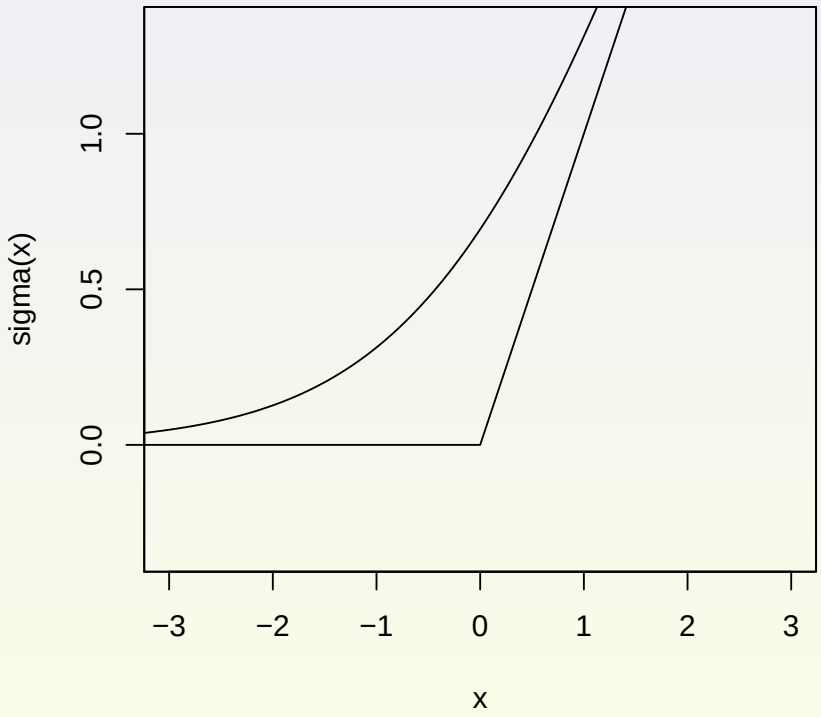
$$\sigma(x) = \log(1+e^{x}) \qquad \sigma'(x) = \frac{1}{1+e^{-x}}$$

Comparing Activation Functions

Logistic and soft-plus have continuous derivatives in comparison to step function and ReLU

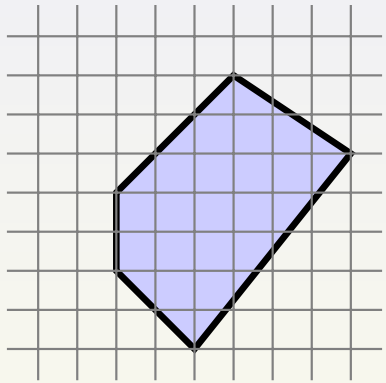
- Positive derivative avoids vanishing gradient





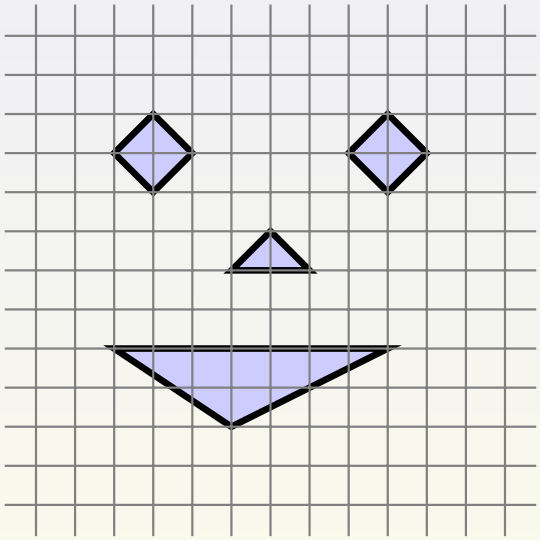
Example

- Single neuron is a binary half-space classifier: $\text{sign}(w \cdot x + b)$
- 2 layer networks can express intersection of halfspaces



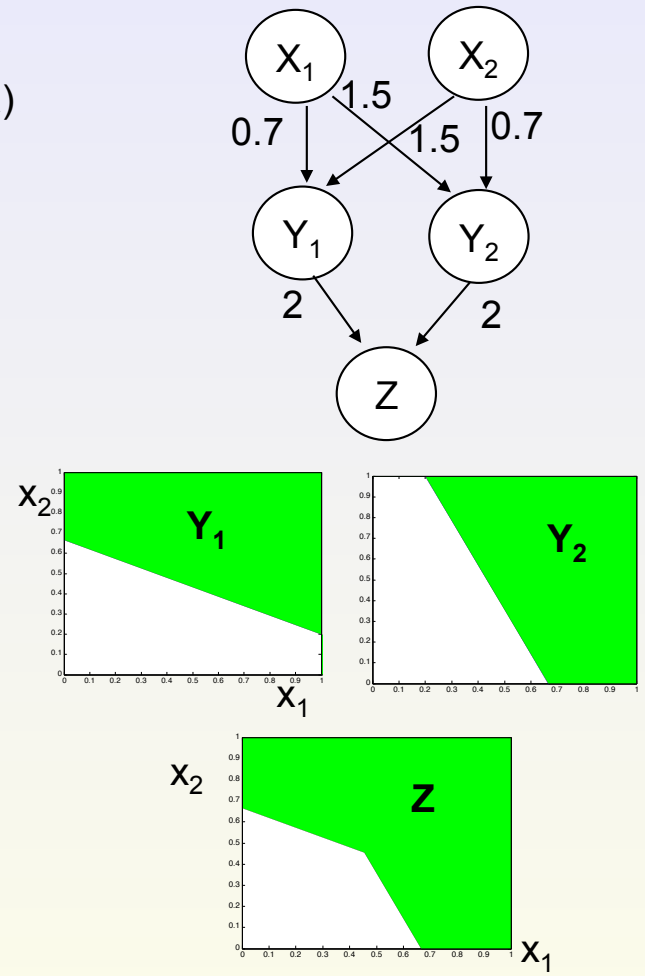
Example

- 3 layer networks can express unions of intersection of halfspaces

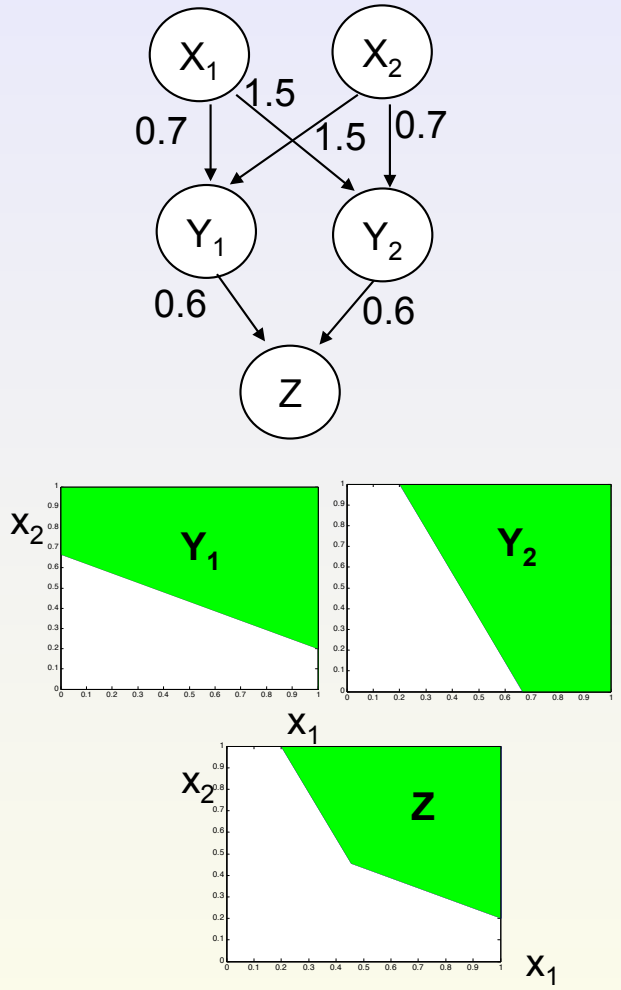


Examples: Step function activation - $f(x) = \mathbf{1}(x)$

a)

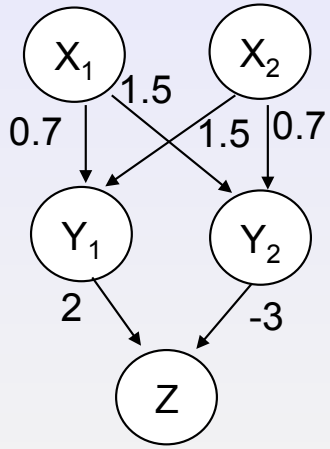


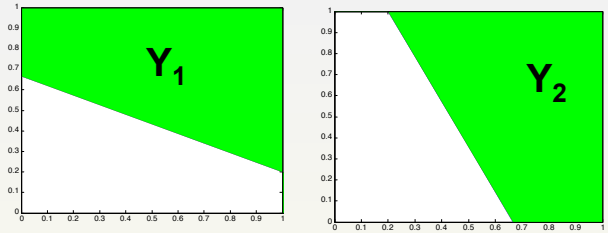
b)

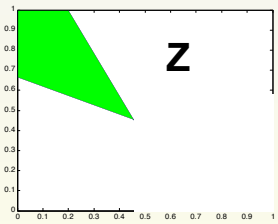


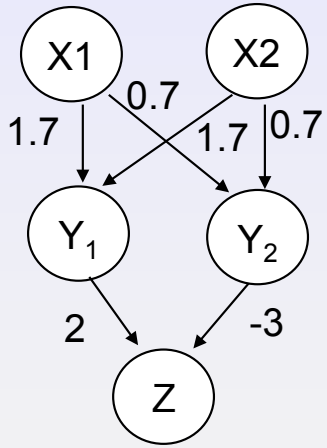
Green - $Y = 1, Z = 1$

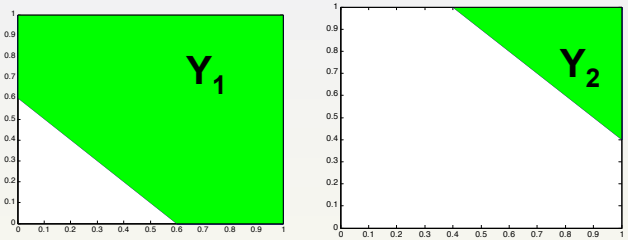
Examples: Step function activation - $f(x) = \mathbf{1}(x)$

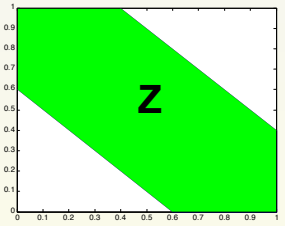












Green - $Y = 1, Z = 1$

How to train neural network?

- Neural nets: excellent hypothesis class, but difficult to train
- Main technique: Stochastic Gradient Descent (SGD)
- Not convex, no guarantees, can take a long time, but:
  - Often still works fine, finds a good solution
  - With some luck:)

Stochastic Gradient Descent (SGD) for Neural Networks

Common Training Ideas:

- Random initialization: rule of thumb, $w[u \to v] \sim U[-c, c]$ where $c = \sqrt{3/|\{(u', v) \in E\}|}$ (or small Gaussian instead of $U[-c, c]$)
- Update step with Nesterov's momentum: Initialize $\theta = 0$ and:

$$\theta_{t+1} = \mu_t \theta_t - \eta_t \tilde{\nabla} L(w_t + \mu_t \theta_t)$$
$$w_{t+1} = w_t + \theta_{t+1}$$

  where:
  μ_t is momentum parameter (e.g. $\mu_t = 0.9$ for all t)
  η_t is learning rate (e.g. $\eta_t = 0.01$ for all t)
  $\tilde{\nabla} L$ is an estimate of the gradient of L based on a small set of random examples (often called a "minibatch")

- Efficient gradient calculation: Backpropagation

Backpropagation: Efficient Implementation of Chain Rule

Chain rule

$$(f(g(x)))' = f'(g(x))g'(x)$$
$$(f(f(x)))' = f'(f(x))f'(x)$$
$$(f(f(f(x))))' = f'(f(f(x)))f'(f(x))f'(x)$$

Chain rule in vector form

$$\nabla_{\boldsymbol{x}} \boldsymbol{z} = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} \boldsymbol{z}$$

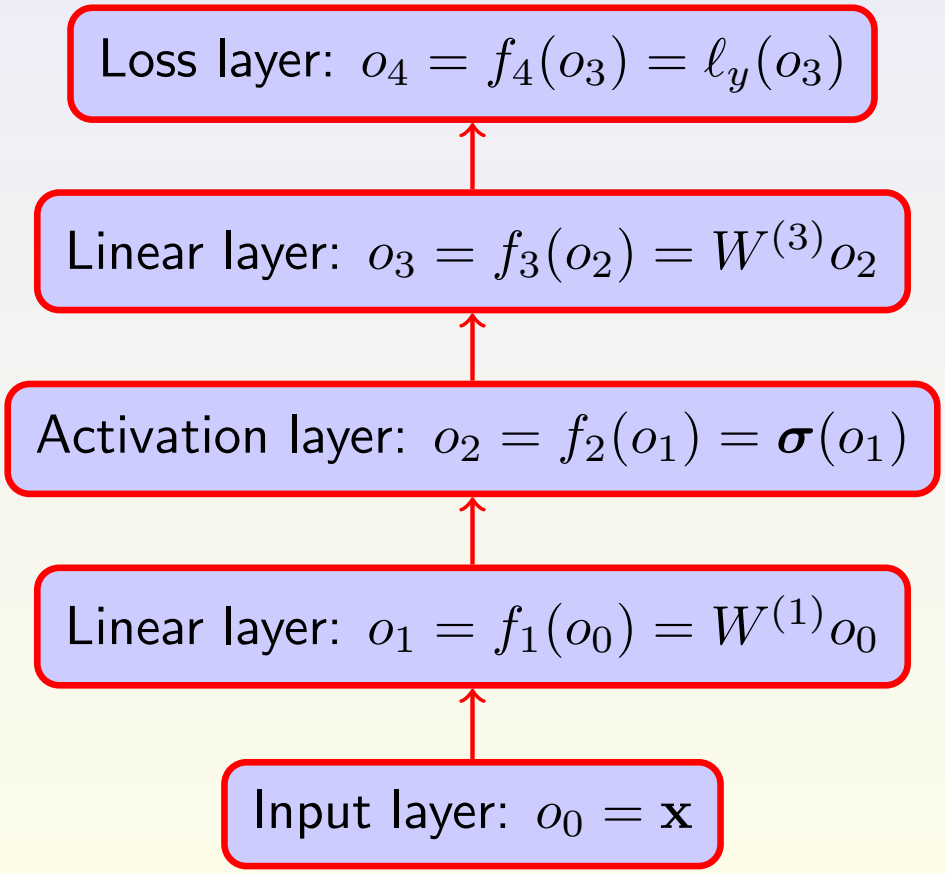
Back-Propagation

- The back-propagation algorithm is an efficient way to calculate $\nabla \ell(h_w(x), y)$ using the chain rule
- Let $\mathbf{f}(\mathbf{w}) : \mathbb{R}^n \to \mathbb{R}^m$, $\mathbf{f} = (f_1, \ldots f_m)$, $f_i(\mathbf{w}) : \mathbb{R}^n \to \mathbb{R}$
- $J_{\mathbf{w}}(\mathbf{f})$ - Jacobian of $\mathbf{f}(\mathbf{w})$ is the $m \times n$ matrix whose i, j element is the partial derivative of $\partial f_i(\mathbf{w}) / \partial w_j$
  i.e., ith row of $J_{\mathbf{w}}(\mathbf{f})$ is equal to $\nabla f_i(\mathbf{w})$
  Examples
  - If $\mathbf{f}(\mathbf{w}) = A\mathbf{w}$ then $J_{\mathbf{w}}(\mathbf{f}) = A$.
  - If $\boldsymbol{\sigma} : \mathbb{R}^n \to \mathbb{R}^n$ is element-wise application of $\sigma : \mathbb{R} \to \mathbb{R}$ then $J_{\boldsymbol{\theta}}(\boldsymbol{\sigma}) = \mathrm{diag}((\sigma'(\theta_1), \ldots, \sigma'(\theta_n)))$.
- Chain rule:

$$J_{\mathbf{w}}(\mathbf{f} \circ \mathbf{g}) = J_{g(\mathbf{w})}(\mathbf{f}) J_{\mathbf{w}}(\mathbf{g})$$

Back-Propagation

Let $\ell_y : \mathbb{R}^k \to \mathbb{R}$ be the loss function at the output layer.
It's convenient to describe the network as a sequence of simple layer functions:



Back-Propagation

- Can write $\ell(h_{\mathbf{w}}, (\mathbf{x}, y)) = (f_{T+1} \circ \ldots \circ f_3 \circ f_2 \circ f_1)(\mathbf{x})$
- Denote $F_t = f_{T+1} \circ \ldots \circ f_{t+1}$ and $\delta_t = J_{o_t}(F_t)$, then

$$\begin{aligned}\delta_t &= J_{o_t}(F_t) = J_{o_t}(F_{t-1} \circ f_{t+1}) \\ &= J_{f_{t+1}(o_t)}(F_{t-1}) J_{o_t}(f_{t+1}) = J_{o_{t+1}}(F_{t-1}) J_{o_t}(f_{t+1}) \\ &= \delta_{t+1} J_{o_t}(f_{t+1})\end{aligned}$$

- Note that

$$J_{o_t}(f_{t+1}) = \begin{cases} W^{(t+1)} & \text{for linear layer} \\ \operatorname{diag}(\boldsymbol{\sigma}'(o_t)) & \text{for activation layer} \end{cases}$$

- Using the chain rule again we obtain

$$J_{W^{(t)}}(\ell(h_{\mathbf{w}}, (\mathbf{x}, y))) = \delta_t o_{t-1}^\top$$

Back-Propagation: Pseudo-code

Forward:

- set $o_0 = \mathbf{x}$ and for $t = 1, 2, \ldots, T$ set

$$o_t = f_t(o_{t-1}) = \begin{cases} W^{(t)} o_{t-1} & \text{for linear layer} \\ \boldsymbol{\sigma}(o_{t-1}) & \text{for activation layer} \end{cases}$$

Backward:

- set $\delta_{T+1} = \nabla \ell_y(o_T)$ and for $t = T, T-1, \ldots, 1$ set

$$\delta_t = \delta_{t+1} J_{o_t}(f_{t+1}) = \delta_{t+1} \cdot \begin{cases} W^{(t+1)} & \text{for linear layer} \\ \operatorname{diag}(\boldsymbol{\sigma}'(o_t)) & \text{for activation layer} \end{cases}$$

- For linear layers, set the gradient w.r.t. the weights in $W^{(t)}$ to be the elements of the matrix $\delta_t o_{t-1}^\top$

Interesting Questions in Deep Learning

Why does it work well?

- Mathematically, it is not well understood.

This motivated the development of a new class, called

- E6699: Mathematics of Deep Learning

I thought it in Spring'19, Spring'21, and will teach again in Spring'22. Here, some interesting topics/questions that the course addressed: (Maybe I'll discuss further some of the topics next week.)

- Expressiveness of neural nets
- Expressive power of deep learning: why is depth good, depth separation results
- Global versus local optimality
- Generalization error: Why DL models generalize well?
- Random initialization
- Wide nets and connection to Kernels
- Etc.

Recall the Final Project Outline

- Done in groups of 4 students - assemble the groups
- Deliverables: 15+ page **paper** & **presentation** with slides
- **Due:** during the finals week: Dec 16 - 23, very likely **Dec 17**. One slot for presentations on **Tue, Dec 14, 4:10-6:40pm**.
- **Data Repositories**: First, select a paper(s) from either:
  - UC Irvine Machine Learning Repository `https://archive.ics.uci.edu/ml/datasets.php`
  - GEO Data Repository `https://www.ncbi.nlm.nih.gov/geo/`, or Bioconductor Datasets: `http://www.bioconductor.org`
- **Final Paper Outline:** 5 sections
  1. Introduction: e.g., describe the application area, problems considered, etc
  2. Data set(s) and paper(s): e.g., describe data in detail, what was done in the paper(s), common stat/machine learning tools, etc
  3. Reproduce the results from the paper(s)
  4. Try different techniques learned in class, or propose new ones
  5. Discussion and conclusion: e.g., compare different techniques, pros and cons, future work, etc

Bioconductor and Additional Datasets

- Bioconductor provides tools in R for the analysis genomic data: `https://https://www.bioconductor.org/`
- Installing Bioconductor: `https://https://www.bioconductor.org/install/`
  Run the following code:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(version = "3.12")
```

- Then, install Bioconductor packages: `https://www.bioconductor.org/install/#install-bioconductor-packages`
- Datasets supported by Bioconductor: `http://www.bioconductor.org/packages/release/data/experiment/`

Reading

ESL: Chapter 11 on Neural Networks

Homework: Work the final project.

Reading on Deep Learning:

- Chapters 14& 20 in: Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014.
- Chapter 6 in: Deep Learning, I. Goodfellow and Y. Bengio and A. Courville, MIT Press, 2016. `http://www.deeplearningbook.org`
- Software - Tensor Flow in R: `https://tensorflow.rstudio.com`

Optional reading on Kernels (RKHS):
(These books are available online through CU Library)

1. Chapters 1, 2, 13-16 in
   B. Schölkopf and A. J. Smola. *Learning with Kernels.* MIT Press, Cambridge, MA, 2002.
2. Chapters 1, 5.3, 6, 7 in (this book is mathematically advanced)
   A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics.* Kluwer, 2004.