EE2211 Pre-Tutorial 8

Dr Feng LIN
feng_lin@nus.edu.sg

Agenda

- Recap
- Self-learning
- Tutorial 8

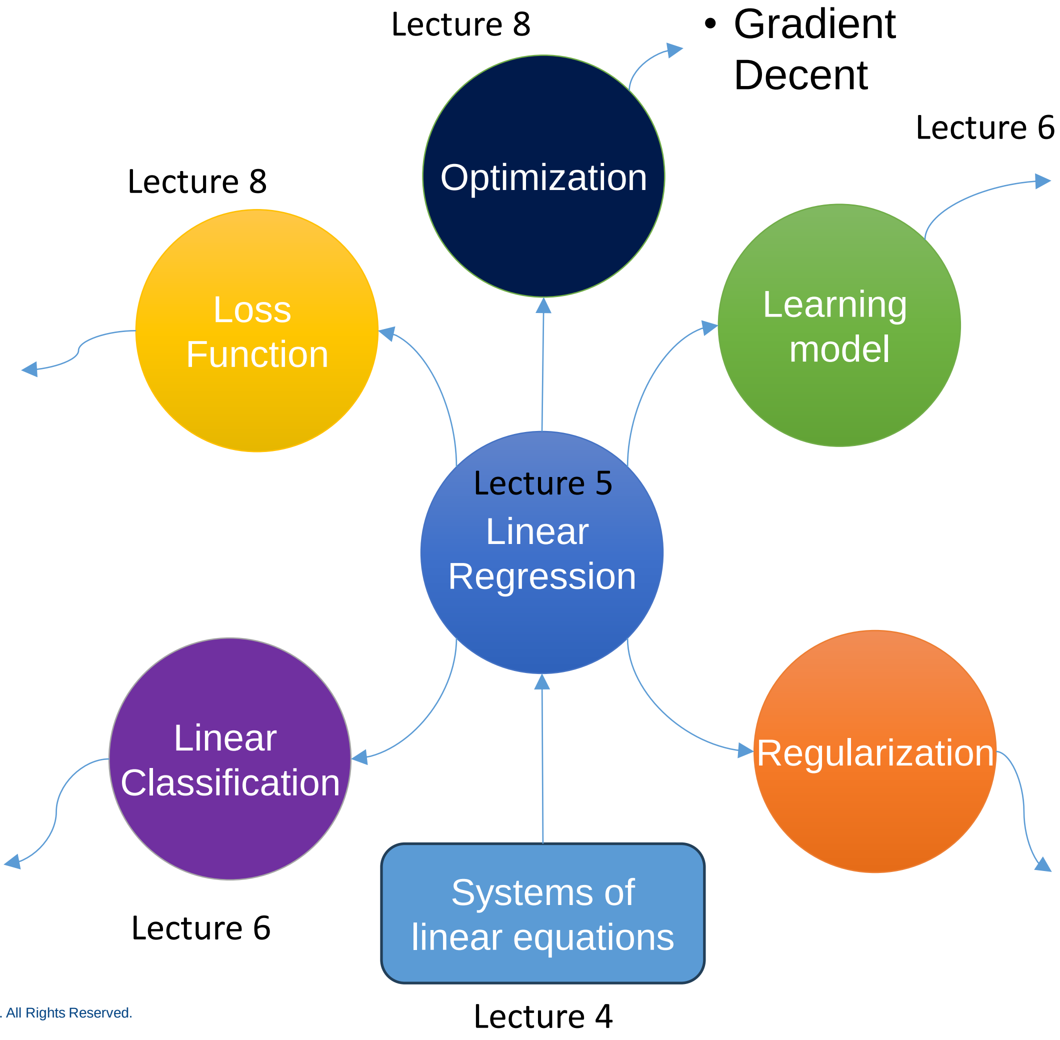
Today’s Attendance

Recap

- Fundamental Machine Learning Algorithms I (Helen)
  - Systems of linear equations
  - Least squares, Linear regression
  - Ridge regression, Polynomial regression
- Fundamental Machine Learning Algorithms II (Helen)
  - Over-fitting, bias/variance trade-off
  - Optimization, Gradient descent
  - Decision Trees, Random Forest

Recap

- Square error loss
- Binary loss
- Hinge loss
- Exponential loss



- Linear model
- Affine model
- Polynomial model
- Sigmoid function
- Rectified linear unit (ReLU)
- Exponential

Lecture 8

Lecture 6&7

- L2 regularization (Ridge)
- L1 regularization (LASSO)
- …

- Binary classification
- Multi-Category Classification

Loss Function & Learning Model

- For polynomial regression (previous lectures)

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, (\mathbf{P}\mathbf{w} - \mathbf{y})^T(\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w}$$
$$= \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (\mathbf{p}_i^T\mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $f(\mathbf{x}_i, \mathbf{w})$ be the prediction of target y_i from features $\mathbf{x}_i$ for i-th training sample. For example, suppose $f(\mathbf{x}_i, \mathbf{w}) = \mathbf{p}_i^T\mathbf{w}$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2 + \lambda \mathbf{w}^T\mathbf{w}$$

- Let $L(f(\mathbf{x}_i, \mathbf{w}), y_i)$ be the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when true value is y_i. For example, suppose $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$, then above becomes

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T\mathbf{w}$$

Loss function & Learning model

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, C(\mathbf{w}) = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

Cost Function Loss Function Learning Model Regularization

- Square error loss
- Binary loss
- Hinge loss
- Exponential loss

- Linear model
- Affine model
- Polynomial model
- Nonlinear models

- L2 regularization (Ridge)
- L1 regularization (LASSO)
- …

Building Blocks of ML Algorithms

- From previous slide

$$\operatorname*{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda \mathbf{w}^T \mathbf{w}$$

- To make it even more general, we can write

$$\operatorname*{argmin}_{\mathbf{w}} C(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{m} L(f(\mathbf{x}_i, \mathbf{w}), y_i) + \lambda R(\mathbf{w})$$

- Learning model f reflects our belief about the relationship between the features $\mathbf{x}_i$ & target y_i
- Loss function L is the penalty for predicting $f(\mathbf{x}_i, \mathbf{w})$ when the true value is y_i
- Regularization R encourages less complex models
- Cost function C is the final optimization criterion we want to minimize
- Optimization routine to find solution to cost function

Motivations for Gradient Descent

- Different learning function f, loss function L & regularization R give rise to different learning algorithms
- In polynomial regression (previous lectures), optimal $\mathbf{w}$ can be written with the following “closed-form” formula (primal solution):

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- For other learning function f, loss function L & regularization R, optimizing $C(\mathbf{w})$ might not be so easy
- Usually have to estimate $\mathbf{w}$ iteratively with some algorithm
- Optimization workhorse for modern machine learning is gradient descent

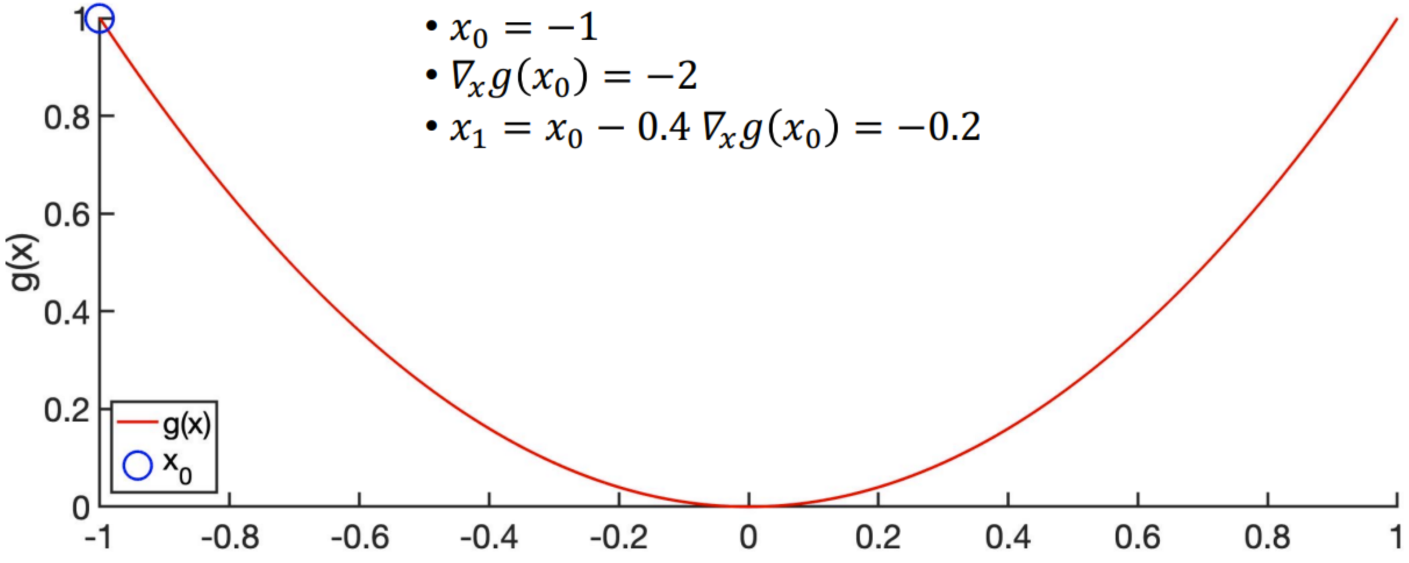
Gradient Descend Algorithm

- Suppose we want to minimize $C(\mathbf{w})$ with respect to $\mathbf{w} = [w_1, \cdots, w_d]^T$
- Gradient $\nabla_{\mathbf{w}} C(\mathbf{w}) = \begin{pmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_d} \end{pmatrix}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is vector & function of $\mathbf{w}$
  - $\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is increasing most rapidly, so $-\nabla_{\mathbf{w}} C(\mathbf{w})$ is direction at $\mathbf{w}$ where C is decreasing most rapidly
- Gradient Descent:

Initialize $\mathbf{w}_0$ and learning rate η;
while *true* **do**
  Compute $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla_{\mathbf{w}} C(\mathbf{w}_k)$
  if *converge* **then**
    return $\mathbf{w}_{k+1}$
  end
end

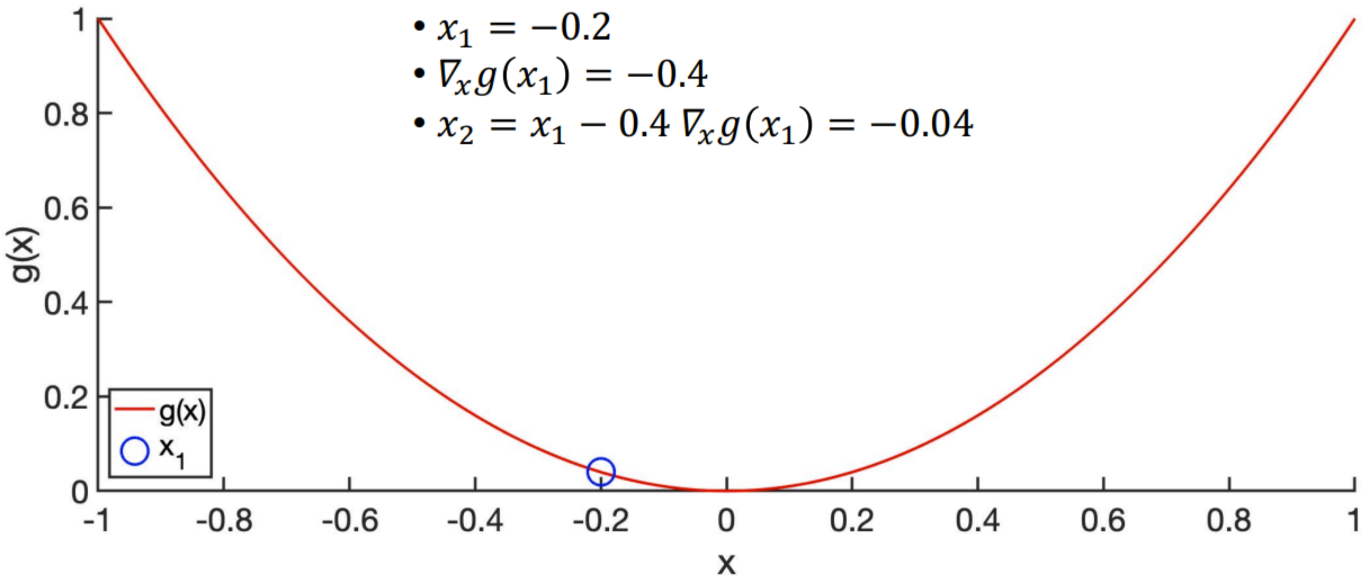
Gradient Descent

- Gradient $\nabla_x g(x) = 2x$
- Initialize $x_0 = -1$, learning rate $\eta = 0.4$
- At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



Gradient Descent

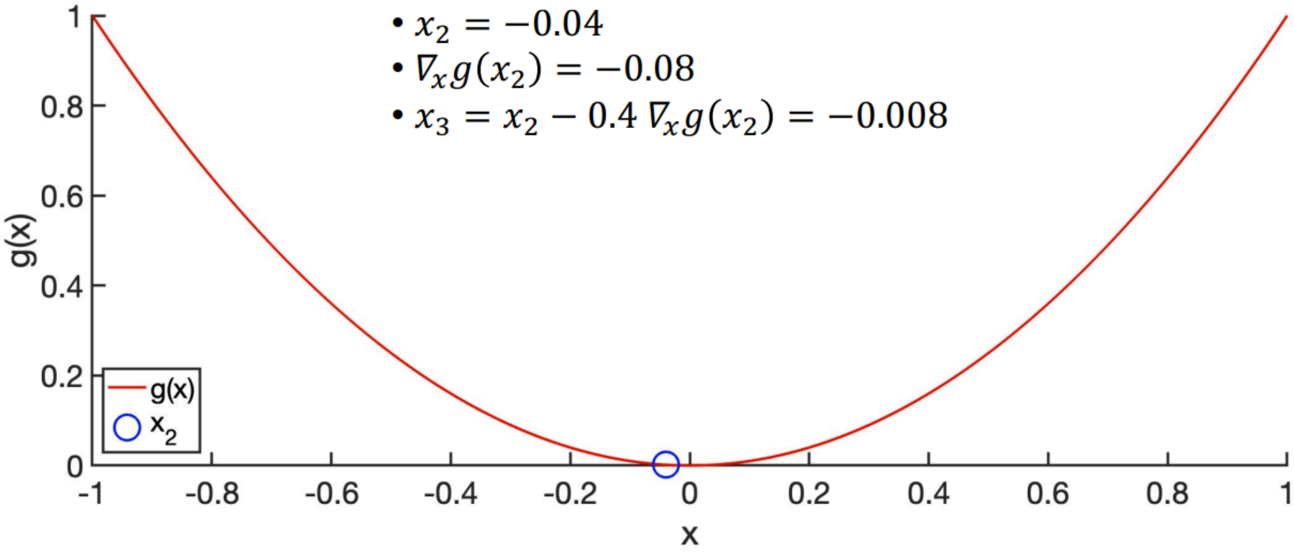
- Gradient $\nabla_x g(x) = 2x$
- Initialize $x_0 = -1$, learning rate $\eta = 0.4$
- At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



- $x_1 = -0.2$
- $\nabla_x g(x_1) = -0.4$
- $x_2 = x_1 - 0.4\, \nabla_x g(x_1) = -0.04$

Gradient Descent

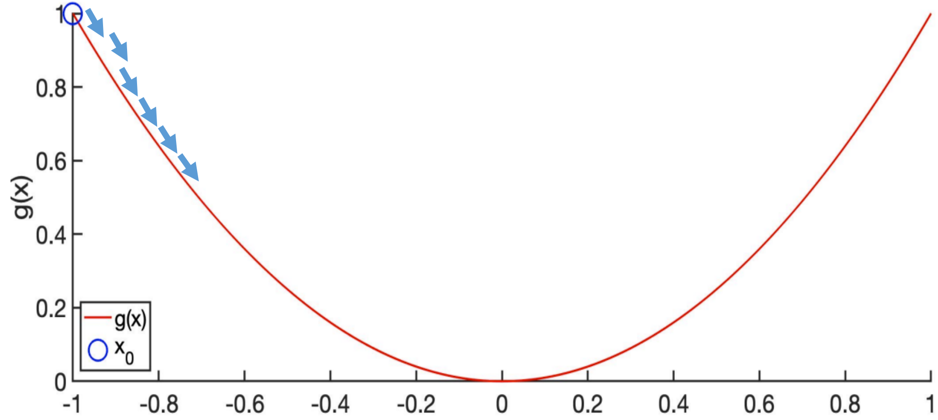
- Gradient $\nabla_x g(x) = 2x$
- Initialize $x_0 = -1$, learning rate $\eta = 0.4$
- At each iteration, $x_{k+1} = x_k - \eta \nabla_x g(x_k)$



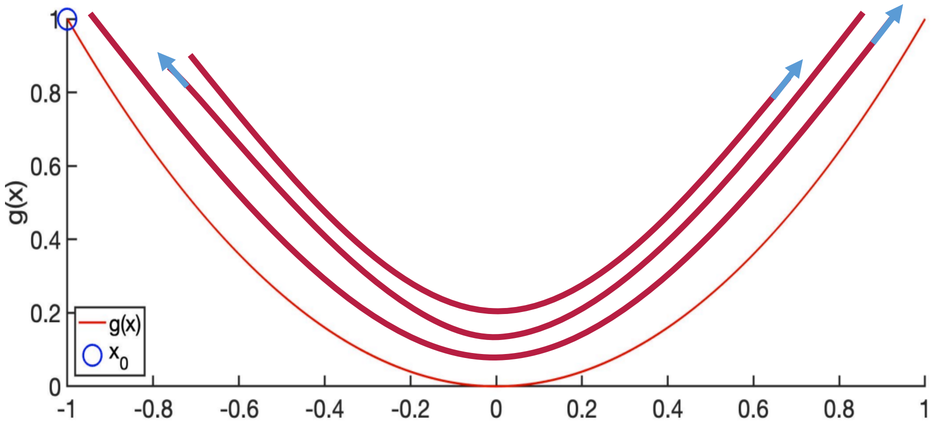
- $x_2 = -0.04$
- $\nabla_x g(x_2) = -0.08$
- $x_3 = x_2 - 0.4\, \nabla_x g(x_2) = -0.008$

Learning Rate

If learning rate is too small, may take too long to converge

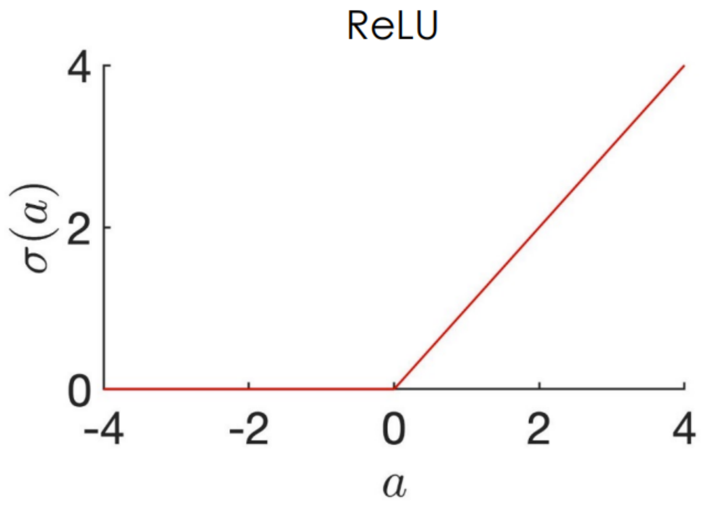


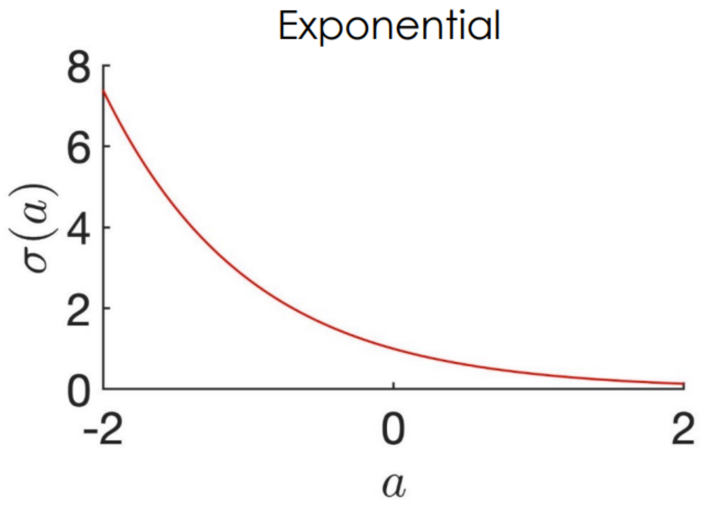
If learning rate is too big, may jump between local minimas and never converge



Different Learning Models

- $f(\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{p}_i^T \mathbf{w})$, where σ can be different functions:
- Rectified linear unit (ReLU): $\sigma(a) = \max(0, a)$
- Exponential: $\sigma(a) = \exp(-a)$

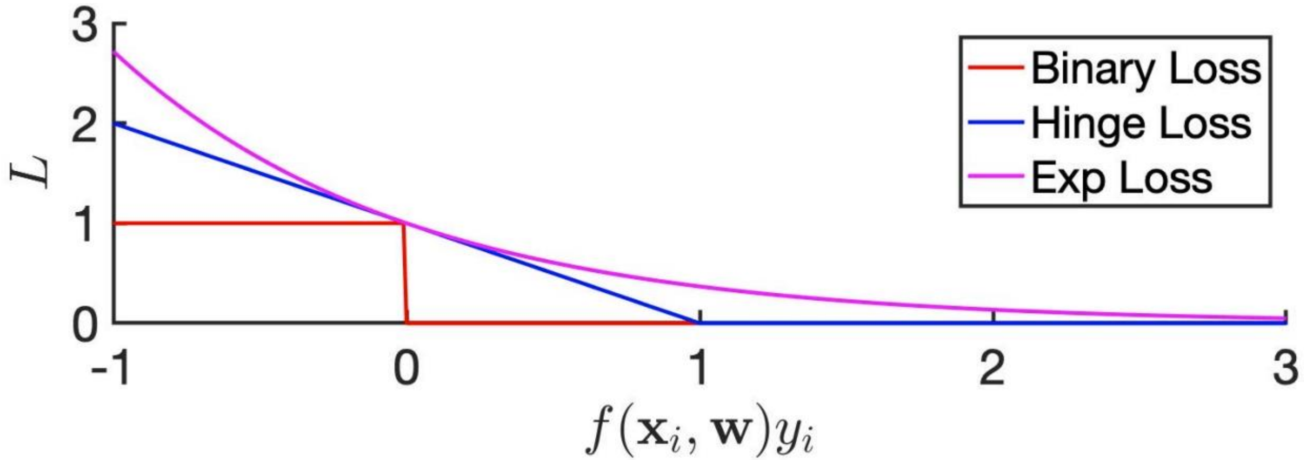




Different Loss Functions

- Binary loss, where y_i is class -1 or class 1 & $f(\mathbf{x}_i, \mathbf{w})$ is a number between $-\infty$ and ∞: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i > 0 \\ 1 & \text{if } f(\mathbf{x}_i, \mathbf{w})y_i < 0 \end{cases}$

- Binary loss not differentiable, so two other possibilities
  - Hinge loss: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \max(0, 1 - f(\mathbf{x}_i, \mathbf{w})y_i)$
  - Exponential loss: $L(f(\mathbf{x}_i, \mathbf{w}), y_i) = \exp(-f(\mathbf{x}_i, \mathbf{w})y_i)$



Recap

- Building blocks of machine learning algorithms
  – Learning model: reflects our belief about relationship between features & target we want to predict
  – Loss function: penalty for wrong prediction
  – Regularization: penalizes complex models
  – Optimization routine: find minimum of overall cost function
- Gradient descent algorithm
  – At each iteration, compute gradient & update model parameters in direction opposite to gradient
  – If learning rate η is too big => may not converge
  – If learning rate η is too small => converge very slowly
- Different learning models, e.g., linear, polynomial, sigmoid, ReLU, exponential, etc
- Different loss functions, e.g., square error, binary, logistic, etc

THANK YOU