CS229 Section: Midterm Review

Hong Jun Jeon

February 9, 2024

Outline

Supervised Learning: Recap

- **Given**: a set of data points (or attributes) $\{x^{(1)}, x^{(2)}, ..., x^{(m)}\}$ and their associated labels $\{y^{(1)}, y^{(2)}, ..., y^{(m)}\}$
- **Dimensions**: x usually d-dimensional $\in \mathbb{R}^d$, y typically scalar
- **Goal**: build a model that predicts y from x for unseen x

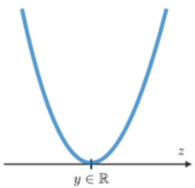
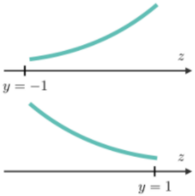
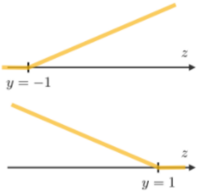
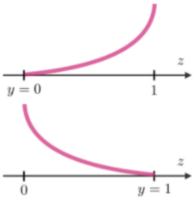
Supervised Learning: Recap

Types of predictions

- y is continuous, real-valued: Regression
  - Loss fn: Squared error, absolute error.
  - Example: Linear regression
- y is discrete classes: Classification
  - Loss fn: Cross-entropy loss
  - Example: Logistic regression

Notations and Concepts

- **Hypothesis**: Denoted by h_θ. Given an input $x^{(i)}$, predicted output is $h_\theta(x^{(i)})$
- **Loss Function**: Function $L(z, y) : \mathbb{R} \times \mathbb{Y} \mapsto \mathbb{R}$ computes how different the predicted value z and the ground truth label are

Least squared error	Logistic loss	Hinge loss	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0, 1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

Notations and Concepts

- **Cost function**: Function J taking model parameters θ as input and outputs a score to reflect how badly the model performs. The empirical loss, for example, sums the loss over all predictions on the training set:

$$J(\theta) = \sum_{i=1}^{m} L(h_\theta(x^{(i)}), y^{(i)})$$

- **Likelihood**: Maximizing likelihood $L(\theta)$ corresponds returning the parameters θ^* which maximize the likelihood of the data. We express the log likelihood $\ell(\theta) = \log L(\theta)$ and maximize it.

$$\theta^* = \text{argmax}_\theta \; \ell(\theta)$$

- **Posterior Likelihood**: Return θ^* which maximizes the posterior probability:

$$\theta^* = \underset{\theta'}{\text{argmax}} \, \mathbb{P}(\theta = \theta' | \text{data})$$

Outline

Optimization: Gradient Descent

- To find the optimal θ that minimizes the cost function $J(\theta)$, we can use gradient descent with a learning rate $\alpha \in \mathbb{R}$

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta J(\theta^{(t)})$$

Stochastic Gradient Descent

- In Stochastic gradient descent (SGD), we update the parameter based on **each** training example, whereas in batch gradient descent we update based on a batch of training examples.

Optimization: Newton's method

- Numerical method to estimate θ such that $J'(\theta)$ is 0
- We update θ as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \frac{J'(\theta^{(t)})}{J''(\theta^{(t)})}$$

- For the multi-dimensional case:

$$\theta^{(t+1)} = \theta^{(t)} - \left[\nabla_\theta^2 J(\theta^{(t)})\right]^{-1} \nabla_\theta J(\theta^{(t)})$$

Recap: Gradients and Hessians

- Gradient and Hessian (differentiable function $f : \mathbb{R}^d \mapsto \mathbb{R}$)

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_d} \end{bmatrix}^\top \in \mathbb{R}^d$$

$$\nabla_x^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix} \in \mathbb{R}^{d \times d}$$

Outline

Linear Regression

- Model: $h_\theta(x) = \theta^T x$
- Training data: $\left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$, $x^{(i)} \in \mathbb{R}^d$
- Loss: $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
- Update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

Stochastic Gradient Descent (SGD)

Pick one data point $x^{(i)}$ and then update:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}$$

Solving Least Squares: Closed Form

- Loss in matrix form: $J(\theta) = \frac{1}{2} \|X\theta - y\|_2^2$, where $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$
- Normal Equation (set gradient to 0):

$$X^T (X\theta^\star - y) = 0$$

- Closed form solution:

$$\theta^\star = \left(X^\top X\right)^{-1} X^\top y$$

Connection to Newton's Method

$$\theta^\star = \left[\nabla_\theta^2 J\right]^{-1} \nabla_\theta J, \quad \text{when the gradient is evaluated at } \theta = 0$$

Newton's method is exact with only one step iteration if we started from $\theta^{(0)} = 0$.

Outline

Logistic Regression

A binary classification model and $y^{(i)} \in \{0,1\}$

- Assumed model:

$$\mathbb{P}(Y = y \mid X, \theta) = \begin{cases} g_\theta(X) & \text{if } y = 1 \\ 1 - g_\theta(X) & \text{if } y = 0 \end{cases}, \quad \text{where } g_\theta(x) = \frac{1}{1 + e^{-\theta^\top X}}$$

- **Log-likelihood function:**

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^{n} \log \mathbb{P}(Y = y^{(i)} \mid X = x^{(i)}, \theta) \\ &= \sum_{i=1}^{n} \left[y^{(i)} \log g_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - g_\theta(x^{(i)})) \right] \end{aligned}$$

- Find parameters through **maximizing log-likelihood**, $\mathrm{argmax}_\theta\, \ell(\theta)$ (in Pset1).

Sigmoid and Softmax

- **Sigmoid**: The sigmoid function (also known as logistic function) is given by:

$$g(z) = \frac{1}{1 + e^{-z}}$$

- **Softmax regression**: Also called as multi-class logistic regression, it generalizes logistic regression to multi-class cases

$$\mathbb{P}(Y = k | X; \theta) = \frac{\exp \theta_k^\top X}{\sum_j \exp \theta_j^\top X}$$

Outline

Exponential Family

Definition

Probability distribution with **natural or canonical parameter** η, **sufficient statistic** $T(y)$ and a **log-partition** function $a(\eta)$ whose density (or mass function) can be written as

$$p(y;\eta) = b(y)\exp\left(\eta^\top T(y) - a(\eta)\right)$$

- Oftentimes, $T(y) = y$
- In many cases, $\exp(-a(\eta))$ can be considered as a normalization term that makes the probabilities sum to one

Common Exponential Distributions

Bernoulli distribution:

$$p(y;\phi) = \phi^y (1-\phi)^{1-y} = \exp\left(y \log\left(\frac{\phi}{1-\phi}\right) + \log(1-\phi) \right)$$

$$\implies b(y) = 1, \quad T(y) = y, \quad \eta = \log\left(\frac{\phi}{1-\phi}\right), \quad a(\eta) = \log(1 + e^{\eta})$$

More examples:

Categorical distribution, Poisson distribution, Multivariate normal distribution, etc

Common Exponential Distributions

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1+\exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^{η}	$\frac{1}{y!}$
Geometric	$\log(1-\phi)$	y	$\log\left(\frac{e^{\eta}}{1-e^{\eta}}\right)$	1

Properties

- $\mathbb{E}\left[T\left(Y\right);\eta\right] = \nabla_\eta a\left(\eta\right)$
- $\text{Var}\left(T\left(Y\right);\eta\right) = \nabla^2_\eta a\left(\eta\right)$

Non-exponential Family Distribution

Uniform distribution over interval $[a, b]$:

$$p(y; a, b) = \frac{1}{b-a} \cdot \mathbb{1}_{[a \leq y \leq b]}$$

Reason: $b(y)$ cannot depend on parameter η.

Outline

Generalized Linear Model (GLM)

Generalized Linear Models (GLM) aim at predicting a random variable y as a function of x and rely on the following components:

Assumed model:

$$\mathbb{P}\left(Y \mid X, \theta\right) \sim \mathrm{ExponentialFamily}\left(\eta\right)$$

- $\eta = \theta^\top x$
- Predictor: $h\left(x\right) = \mathbb{E}\left[T\left(Y\right); \eta\right] = \nabla_\eta a\left(\eta\right)$.
- Fitting through maximum likelihood:

$$\max_\theta \ell\left(\theta\right) = \max_\theta \sum_{i=1}^{n} \mathbb{P}(Y = y^{(i)} \mid X = x^{(i)}; \eta)$$

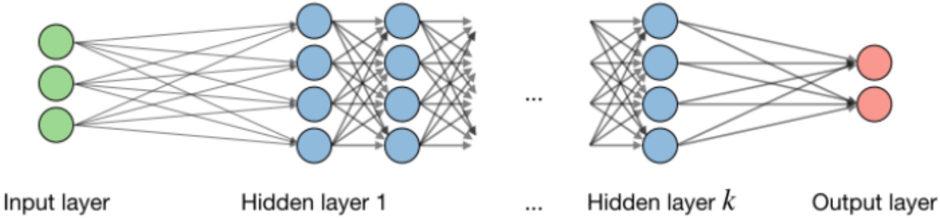
Generalized Linear Model (GLM)

Examples

- GLM under Bernoulli distribution: Logistic regression
- GLM under Poisson distribution: Poisson regression (in Pset1)
- GLM under Normal distribution: Linear regression
- GLM under Categorical distribution: Softmax regression

Outline

Neural Networks



By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$U_j^{[i]} = w_j^{[i]\top} U^{[i-1]}$$

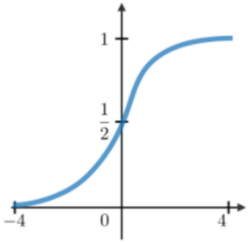
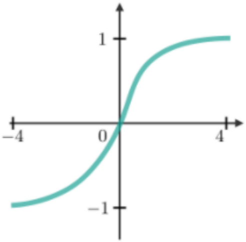
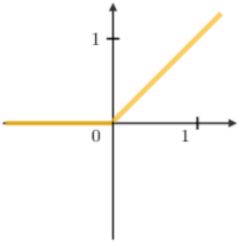
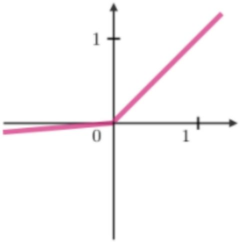
Note we omit the bias for notational brevity, but can assume $U^{[i-1]}$ has an extra dimension with value 1.

Neural Networks

Multi-layer Fully-connected Neural Networks (with Activation Function σ)

$$
\begin{aligned}
U^{[0]} &= X \\
U^{[1]} &= \sigma\left(W^{[1]} U^{[0]}\right) \\
U^{[2]} &= \sigma\left(W^{[2]} U^{[1]}\right) \\
&\cdots \\
U^{[k-1]} &= \sigma\left(W^{[k-1]} U^{[k-2]}\right) \\
U^{[k]} &= W^{[k]} U^{[k-1]}
\end{aligned}
$$

Activation Functions

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Updating Weights

- Step 1: Take a batch of training data
- Step 2: Perform forward propagation to obtain the corresponding loss
- Step 3: Backpropagate the loss to get the gradients
- Step 4: Use the gradients to update the weights of the network

Backpropagation

Let J be the loss function and $U^{[k]} = W^{[k]} U^{[k-1]}$. By chain rule, we have

Layer k Weight Gradient

$$\frac{\partial J}{\partial W_{ij}^{[k]}} = \frac{\partial J}{\partial U_i^{[k]}} \frac{\partial U_i^{[k]}}{\partial W_{ij}^{[k]}}$$

Backpropagation

Let J be the loss function and $U^{[k]} = W^{[k]}U^{[k-1]}$. By chain rule, we have

Layer k Weight Gradient

$$\begin{aligned}\frac{\partial J}{\partial W_{ij}^{[k]}} &= \frac{\partial J}{\partial U_i^{[k]}} \frac{\partial U_i^{[k]}}{\partial W_{ij}^{[k]}} \\ &= \frac{\partial J}{\partial U_i^{[k]}} U_j^{[k-1]}\end{aligned}$$

Backpropagation

Let J be the loss function and $U^{[k]} = W^{[k]} U^{[k-1]}$. By chain rule, we have

Layer k Weight Gradient

$$\begin{aligned}\frac{\partial J}{\partial W_{ij}^{[k]}} &= \frac{\partial J}{\partial U_i^{[k]}} \frac{\partial U_i^{[k]}}{\partial W_{ij}^{[k]}} \\ &= \frac{\partial J}{\partial U_i^{[k]}} U_j^{[k-1]}\end{aligned}$$

$$\frac{\partial J}{\partial W^{[k]}} = \frac{\partial J}{\partial U^{[k]}} U^{[k-1]\top}$$

Backpropagation

Let J be the loss function and $U^{[k]} = W^{[k]} U^{[k-1]}$. By chain rule, we have

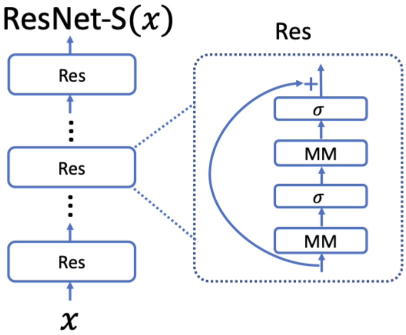
Layer $k-1$ Output Gradient

$$\frac{\partial J}{\partial U_j^{[k-1]}} = \sum_{i=1}^{d_k} \frac{\partial J}{\partial U_i^{[k]}} \frac{\partial U_i^{[k]}}{\partial U_j^{[k-1]}} = \sum_{i=1}^{d_k} \frac{\partial J}{\partial U_i^{[k]}} W_{ij}^{[k]}$$

Modules in Modern NNs

Residual Connections:

$$\mathrm{Res}(x) = z + \sigma(A_2\sigma(A_1 x)).$$



Modules in Modern NNs

Convolutional Layers:

1-D Convolution

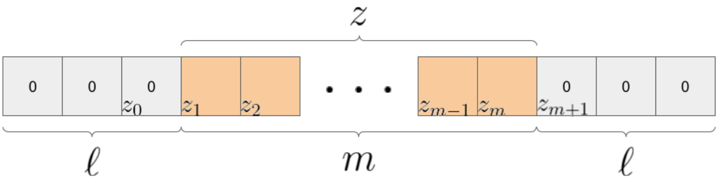
A 1-D convolutional *filter* is identified by a vector $w \in \Re^k$ (WLOG assume $k = 2\ell + 1$ is odd). The filter can be *applied* on any vector $z \in \Re^m$ by first zero padding:

$$\mathrm{Conv}_w(z)_i = \sum_{j=1}^{2\ell+1} w_j z_{i-\ell+(j-1)}.$$

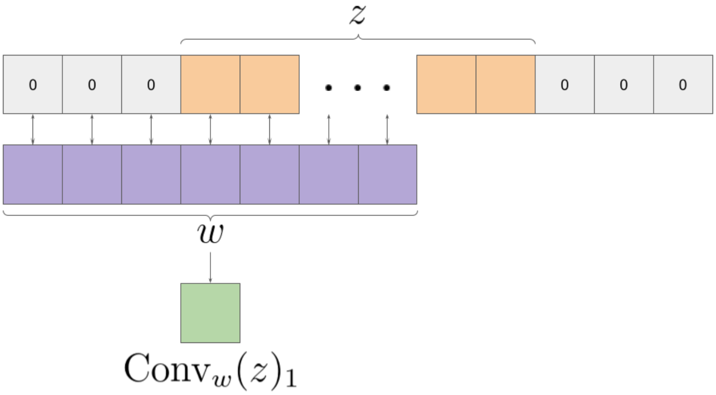
Modules in Modern NNs

1-D Convolution

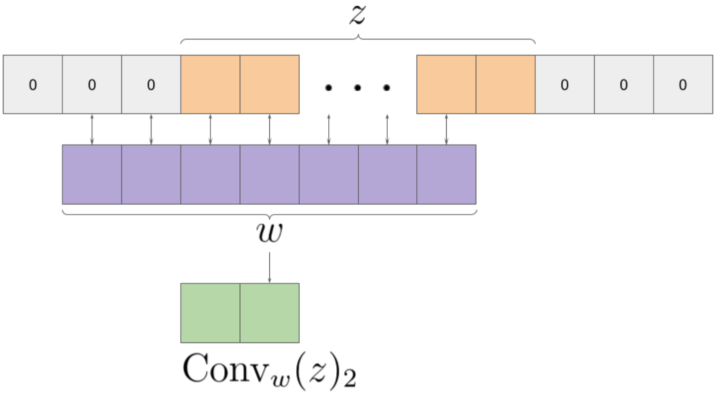
$$\mathrm{Conv}_w(z)_i = \sum_{j=1}^{2\ell+1} w_j z_{i-\ell+(j-1)}.$$



Modules in Modern NNs



Modules in Modern NNs



Modules in Modern NNs

Convolution Properties

1. Output dimension matches input dimension: $\text{Conv}_w(z) \in \Re^m$
2. Exist variants which do not zero-pad: output dimension $\leq$ input dimension.
3. Multiple convolutional layers can be applied sequentially.
4. Usually several fully-connected layers are applied before final output of the network.
5. Architecture breaks permutation invariance of standard fully connected layers.
6. Useful for data for which spacial structure is important i.e. images, audio.

Tips

- Practice, practice, practice
- For proofs, give reasoning and show how you go from one step to the next
- Prepare a cheat sheet – easy to run out of time in open book exams
- Pay attention to notation and indices. "Silly mistakes" can completely change the meaning of your reasoning

All the best :)