CS229 Section: Midterm Review

May 12, 2023

Outline

Supervised Learning: Recap

- **Given**: a set of data points (or attributes) $\{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$ and their associated labels $\{y^{(1)}, y^{(2)}, ..., y^{(n)}\}$
- **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
- Example: Linear regression
- y is discrete classes: Classification
- Example: Logistic regression, SVM, Naive Bayes

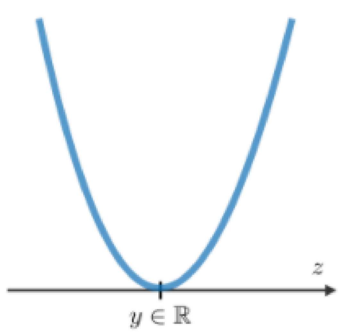
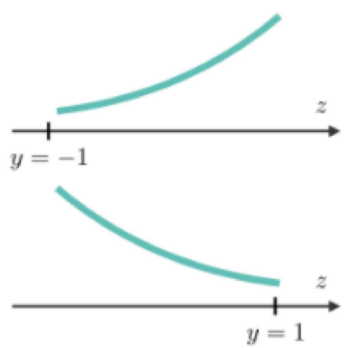
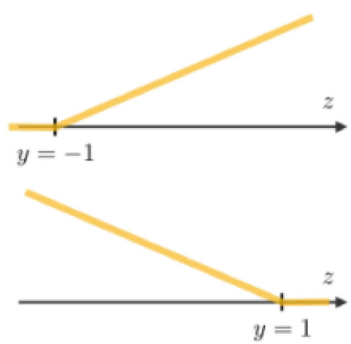
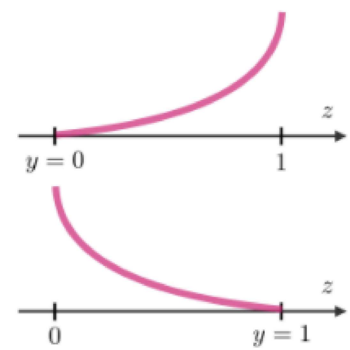
Supervised Learning: Recap

Types of models

- **Discriminative**
- Directly estimate $p(y \mid x)$ by learning decision boundary
- Example: Logistic regression, SVM
- **Generative**
- Estimate $p(x \mid y)$ and infer $p(y \mid x)$ from it
- Can generate new samples
- Example: GDA, Naive Bayes

Notations and Concepts

- **Hypothesis**: Denoted by h_θ. Given an input $x^{(i)}$, predicted output is $h_\theta(x^{(i)})$
- **Loss Function**: Function $\ell(z, y) : \mathbb{R} \times \mathbb{Y} \mapsto \mathbb{R}$ 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 giving a score to reflect how badly the model performs. Average of loss over all predictions

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(x^{(i)}, y^{(i)}; \theta) \quad \text{where} \quad L(x^{(i)}, y^{(i)}; \theta) = \ell(h_\theta(x^{(i)}), y^{(i)})$$

- **Maximum Likelihood**: Often we assume a probabilistic model $p(y \mid x; \theta)$, in which case the loss is the negative log likelihood (NLL):

$$L(x^{(i)}, y^{(i)}; \theta) = -\log p(y^{(i)} \mid x^{(i)}; \theta)$$

Minimizing NLL is equivalent to maximizing likelihood.

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 $\alpha \in \mathbb{R}$
$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta J(\theta^{(t)})$$
- By linearity of the ∇ operator, $\nabla_\theta J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta L(x^{(i)}, y^{(i)}; \theta)$

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 examples.
- Stochastic gradient is correct in expectation:

$$\mathbb{E}_{i\sim \text{Unif}[n]}[\nabla_\theta L(x^{(i)}, y^{(i)}; \theta)] = \nabla_\theta J(\theta)$$

Optimization: Newton's method

- Numerical method to estimate θ such that $\nabla J(\theta)$ is 0
- Idea: approximate $J(\theta)$ by a quadratic *locally* around current parameters $\theta^{(t)}$

$$J(\theta^{(t)} + \Delta\theta) \approx J(\theta^{(t)}) + \nabla_\theta J(\theta^{(t)})^\top \Delta\theta + \frac{1}{2}\Delta\theta^\top \nabla_\theta^2 J(\theta^{(t)})\Delta\theta$$

- To minimize, we set the derivative of this quadratic, with respect to $\Delta\theta$, equal to zero:

$$0 = \nabla_\theta J(\theta^{(t)}) + \nabla_\theta^2 J(\theta^{(t)})\Delta\theta$$

- Then jump to the minimum of the quadratic:

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

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: $h_\theta(x) = \theta^\top x$
- Loss: $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
- Update rule:

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

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):

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

- 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:

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

- Likelihood and log-likelihood:

$$p(y \mid x; \theta) = g_\theta(x)^y (1 - g_\theta(x))^{1-y}$$
$$\log p(y \mid x; \theta) = y \log g_\theta(x) + (1 - y) \log(1 - g_\theta(x))$$

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

$$p(y = k \mid x; \theta) = \frac{\exp \theta_k^\top x}{\sum_j \exp \theta_j^\top x}$$

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:

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

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

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

- $\mathbb{E}[T(Y);\eta] = \nabla_\eta a(\eta)$
- $\mathrm{Var}(T(Y);\eta) = \nabla_\eta^2 a(\eta)$

Non-exponential Family Distribution

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

$$p(y; a, b) = \frac{1}{b-a} \cdot 1_{\{a \le y \le b\}}$$

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:

$$p(y \mid x; \theta) \sim \mathrm{ExponentialFamily}(\theta^\top x)$$

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

$$\arg\max_\theta J(\theta) = \arg\max_\theta \sum_{i=1}^{n} \log p(y^{(i)} \mid \theta^\top x^{(i)})$$

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

Gaussian Discriminant Analysis (GDA)

Generative Algorithm for Classification

- Learn $p(x \mid y)$ and $p(y)$
- Classify through Bayes rule: $\text{argmax}_y\, p(y \mid x) = \text{argmax}_y\, p(x \mid y) p(y)$

GDA Formulation

- Assume $p(x \mid y) \sim \mathcal{N}(\mu_y, \Sigma)$ for some $\mu_y \in \mathbb{R}^d$ and $\Sigma \in \mathbb{R}^{d \times d}$
- Estimate μ_y, Σ and $p(y)$ through maximum likelihood, which is

$$\text{argmax} \sum_{i=1}^{n} \left[\log p(x^{(i)} \mid y^{(i)}) + \log p(y^{(i)})\right]$$

$$p(y) = \frac{\sum_{i=1}^{n} 1_{\{y^{(i)}=y\}}}{n}, \mu_y = \frac{\sum_{i=1}^{n} 1_{\{y^{(i)}=y\}} x^{(i)}}{\sum_{i=1}^{n} 1_{\{y^{(i)}=y\}}}, \Sigma = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^\top$$

Naive Bayes

Formulation

- Assume $p(x \mid y) = \prod_{j=1}^{d} p(x_j \mid y)$
- Estimate $p(x_j \mid y)$ and $p(y)$ through maximum likelihood, which gives

$$p(x_j \mid y) = \frac{\sum_{i=1}^{n} 1_{\left\{x_j^{(i)}=x_j, y^{(i)}=y\right\}}}{\sum_{i=1}^{n} 1_{\left\{y^{(i)}=y\right\}}}, \quad p\left(y\right) = \frac{\sum_{i=1}^{n} 1_{\left\{y^{(i)}=y\right\}}}{n}$$

Laplace Smoothing

Assume x_j takes value in $\{1, 2, \ldots, k\}$, the corresponding modified estimator is

$$p(x_j \mid y) = \frac{1 + \sum_{i=1}^{n} 1_{\left\{x_j^{(i)}=x_j, y^{(i)}=y\right\}}}{k + \sum_{i=1}^{n} 1_{\left\{y^{(i)}=y\right\}}}$$

Outline

Kernel

- Core idea: reparametrize parameter θ as a linear combination of featurized vectors
- Feature map: $\phi : \mathbb{R}^d \mapsto \mathbb{R}^p$
- Fitting linear model with gradient descent (assuming $\theta^{(0)} = 0$) gives us

$$\theta = \sum_{i=1}^{n} \beta_i \phi(x^{(i)})$$

- Predict a new example z:

$$h_\theta(z) = \sum_{i=1}^{n} \beta_i \phi(x^{(i)})^\top \phi(z) = \sum_{i=1}^{n} \beta_i K(x^{(i)}, z)$$

- It brings nonlinearity without much sacrifice in efficiency as long as $K(\cdot, \cdot)$ can be computed efficiently

Kernel

- Given a feature mapping ϕ, we define the kernel K as follows:

$$K(x, z) = \phi(x)^\top \phi(z)$$

- "Kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ, which is often very complicated
- Instead, only the values $K(x, z)$ are needed
- Suppose $K(x^{(i)}, x^{(j)}) = \mathbf{K}_{ij}$
- If $\mathbf{K} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ then is K a valid kernel function?
- If $\mathbf{K} = \begin{bmatrix} 3 & 5 \\ 5 & 3 \end{bmatrix}$ then is K a valid kernel function?

Kernel

Theorem

$K(x,z)$ is a valid kernel if and only if for any set of $\{x^{(1)},\ldots,x^{(n)}\}$, its Gram matrix, defined as

$$G = \begin{bmatrix} K(x^{(1)},x^{(1)}) & \ldots & K(x^{(1)},x^{(n)}) \\ \vdots & \ddots & \vdots \\ K(x^{(n)},x^{(1)}) & \ldots & K(x^{(n)},x^{(n)}) \end{bmatrix} \in \mathbb{R}^{n\times n}$$

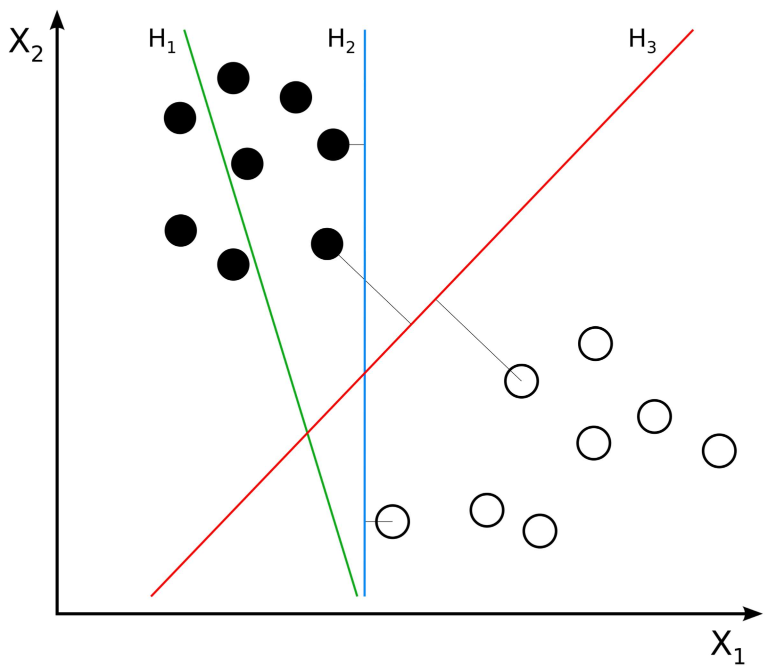
is positive semi-definite.

Examples

- Polynomial kernels: $K(x,z) = (x^\top z + c)^d$, $\forall\ c \geq 0$ and $d \in \mathbb{N}$
- Gaussian kernels: $K(x,z) = \exp(-\frac{\|x-z\|_2^2}{2\sigma^2})$

Support Vector Machine (SVM)

Support Vector Machines are *maximum margin* classifiers.



Support Vector Machine (SVM)

Goal: find the line that maximizes the minimum distance to the line
The optimal margin classifier h with ($y \in \{-1, 1\}$) is such that:

$$h(x) = \mathrm{sign}(w^\top x - b)$$

$$\begin{array}{rl} \text{min}_{w,b} & \frac{1}{2}\|w\|_2^2 \\ \text{subject to} & y^{(i)}(w^\top x^{(i)} - b) \geq 1, \quad \forall\, i \in \{1, \ldots, n\} \end{array}$$

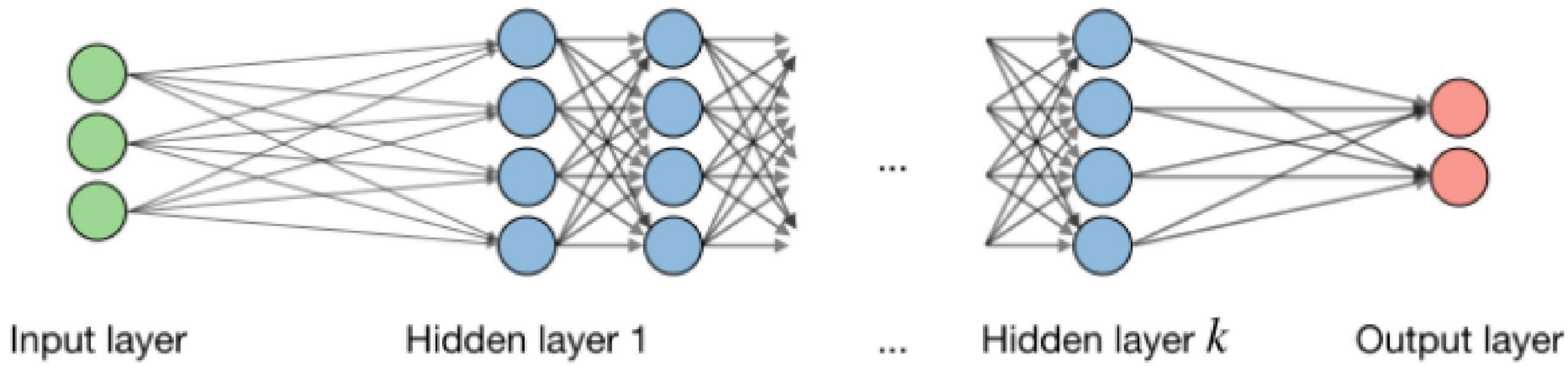
Properties

- The optimal solution has the form $w^\star = \sum_{i=1}^n \alpha_i y^{(i)} x^{(i)}$ and thus can be kernelized.
- The soft-SVM can be treated as a minimization over hinge loss plus ℓ_2 regularization:

$$\min_{w,b} \sum_{i=1}^{n} \max\left\{0, 1 - y^{(i)}(w^\top x^{(i)} - b)\right\} + \lambda \|w\|_2^2$$

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:

$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

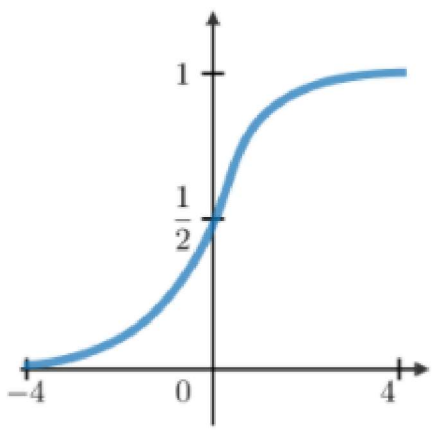
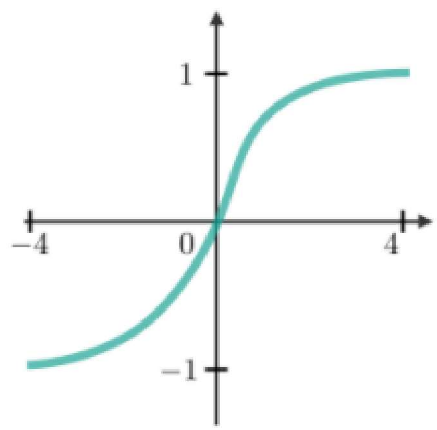
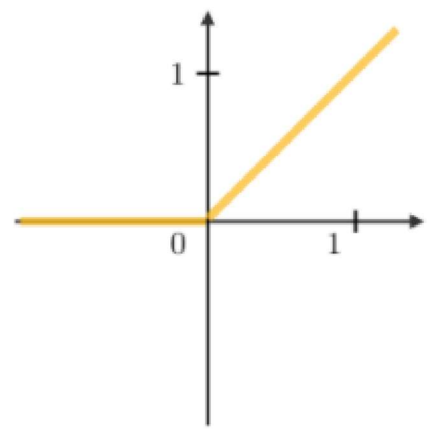
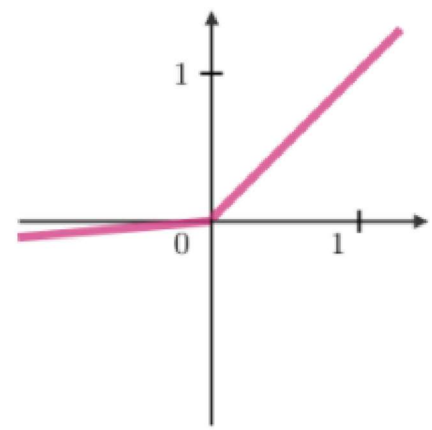
where we note w, b, z the weight, bias and output respectively.

Neural Networks

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

$$
\begin{aligned}
a^{[1]} &= \sigma(W^{[1]}x + b^{[1]}) \\
a^{[2]} &= \sigma(W^{[2]}a^{[1]} + b^{[2]}) \\
&\cdots \\
a^{[r-1]} &= \sigma(W^{[r-1]}a^{[r-2]} + b^{[r-1]}) \\
h_\theta(x) = a^{[r]} &= W^{[r]}a^{[r-1]} + b^{[r]}
\end{aligned}
$$

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 $z^{[k]} = W^{[k]}a^{[k-1]} + b^{[k]}$. By chain rule, we have

$$\frac{\partial J}{\partial W_{ij}^{[r]}} = \frac{\partial J}{\partial z_i^{[r]}} \frac{\partial z_i^{[r]}}{\partial W_{ij}^{[r]}} = \frac{\partial J}{\partial z_i^{[r]}} a_j^{[r-1]} \implies \frac{\partial J}{\partial W^{[r]}} = \frac{\partial J}{\partial z^{[r]}} a^{[r-1]\top}, \quad \frac{\partial J}{\partial b^{[r]}} = \frac{\partial J}{\partial z^{[r]}}$$

$$\frac{\partial J}{\partial a_i^{[r-1]}} = \sum_{j=1}^{d_r} \frac{\partial J}{\partial z_j^{[r]}} \frac{\partial z_j^{[r]}}{\partial a_i^{[r-1]}} = \sum_{j=1}^{d_r} \frac{\partial J}{\partial z_j^{[r]}} W_{ji}^{[r]} \implies \frac{\partial J}{\partial a^{[r-1]}} = W^{[r]\top} \frac{\partial J}{\partial z^{[r]}}$$

$$\frac{\partial J}{\partial z^{[r]}} := \delta^{[r]} \implies \frac{\partial J}{\partial z^{[r-1]}} = \left(W^{[r]\top}\delta^{[r]}\right) \odot \sigma'\left(z^{[r-1]}\right) := \delta^{[r-1]}$$

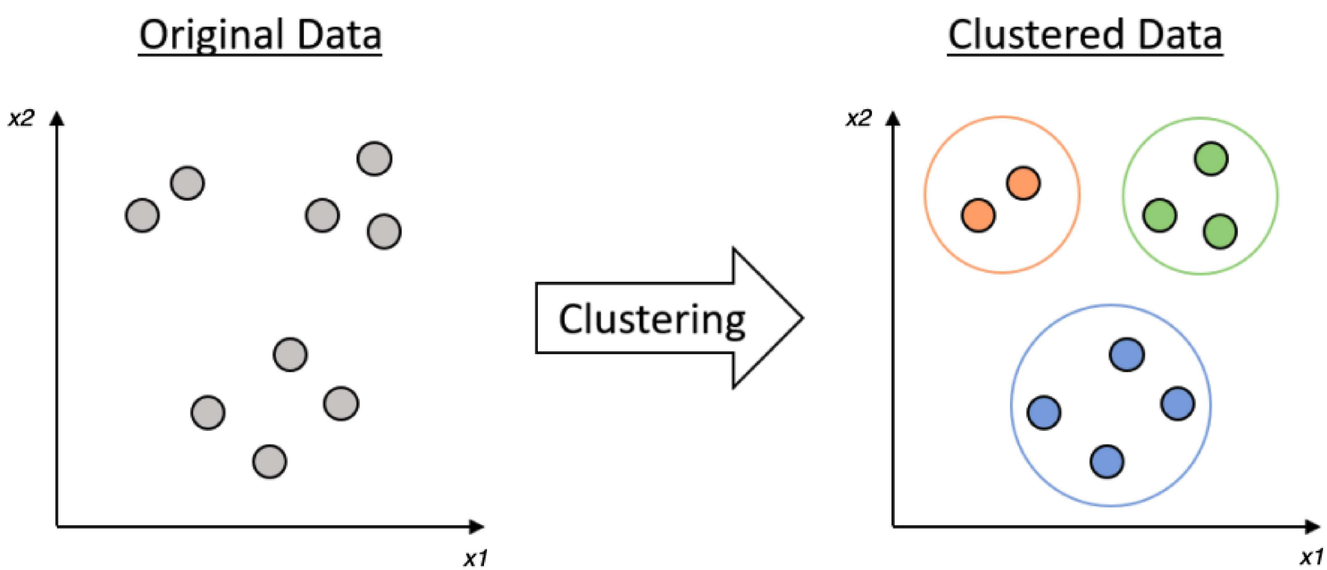
$$\implies \frac{\partial J}{\partial W^{[r-1]}} = \delta^{[r-1]} a^{[r-2]\top}, \quad \frac{\partial J}{\partial b^{[r-1]}} = \delta^{[r-1]}$$

Continue for layers $r-2, \ldots, 1$.

Outline

Clustering

In unsupervised learning, we don't have labels $y^{(i)}$. Instead, our goal is to find "interesting" patterns in the features $x^{(i)}$. One such task is to identify *clusters* of data points that are nearer to each other than they are to other points:



k-means algorithm

We randomly initialize k cluster centers $\mu^{(1)}, \ldots, \mu^{(k)}$ and then alternate between two steps:

1. Assign each point to closest $\mu^{(j)}$: $C^{(i)} = \arg\min_{j \in [k]} \|x^{(i)} - \mu^{(j)}\|$
2. Re-compute cluster centers: $\mu^{(j)} = \frac{1}{|\Omega_j|} \sum_{i \in \Omega_j} x^{(i)}$ where $\Omega_j = \{i : C^{(i)} = j\}$

Comments:

- The number of clusters k is left as a hyperparameter.
- The algorithm is guaranteed to converge to a local optimum of the cost function

$$\min_{C, \mu} \sum_{i=1}^{n} \|x^{(i)} - \mu^{C^{(i)}}\|^2$$

but may not find a global optimum.

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
- Think in vector terms!

All the best :)