Review for Final

Ying WANG

CDS, NYU

May 3, 2023

Contents

Note: For materials in Week 1 - 5, please refer to Review for Midterm slide

Brief Recap: Bayesian Methods

- Prior represents belief about θ before observing data $\mathcal{D}$.
- Posterior represents the **rationally "updated" beliefs** after seeing $\mathcal{D}$.
- All inferences and action-taking are based on the posterior distribution.
- In the Bayesian approach,
    - No issue of "choosing a procedure" or justifying an estimator.
    - Only choices are
        - **family of distributions**, indexed by Θ, and the
        - **prior distribution** on Θ
    - For decision making, need a **loss function**.
    - Everything after that is **computation**.

Brief Recap: Bayesian Methods

1. **Define the model**:
    - Choose a parametric family of densities:
    $$\{p(\mathcal{D} \mid \theta) \mid \theta \in \Theta\}.$$
    - Choose a distribution $p(\theta)$ on Θ, called the **prior distribution**.
2. After observing $\mathcal{D}$, compute the **posterior distribution** $p(\theta \mid \mathcal{D})$.
$$\begin{aligned} p(\theta \mid \mathcal{D}) &\propto p(\mathcal{D} \mid \theta) p(\theta) \\ &= \underbrace{L_{\mathcal{D}}(\theta)}_{\text{likelihood}} \underbrace{p(\theta)}_{\text{prior}} \end{aligned}$$
3. Choose **action** based on $p(\theta \mid \mathcal{D})$.

Brief Recap: Multi-class classification

- Problem: Multiclass classification $\mathcal{Y} = \{1, \ldots, k\}$
- Solution 1: One-vs-All
  - Train k models: $h_1(x), \ldots, h_k(x) : \mathcal{X} \to \mathrm{R}$.
  - Predict with $\arg\max_{y \in \mathcal{Y}} h_y(x)$.
  - Gave simple example where this fails for linear classifiers
- Solution 2: Multiclass loss
  - Train one model: $h(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathrm{R}$.
    - $h(x, y)$ gives compatibility score between input x and output y
  - Prediction involves solving $\arg\max_{y \in \mathcal{Y}} h(x, y)$.
    - $$\mathcal{F} = \{x \mapsto \underset{y \in \mathcal{Y}}{\arg\max}\, h(x, y) \mid h \in \mathcal{H}\}$$
    - Final prediction function is a $f \in \mathcal{F}$

Brief Recap: Multi-class classification

- A structured prediction problem is a multiclass problem in which Y is very large, but has (or we assume it has) a certain structure.
- For POS tagging, Y grows exponentially in the length of the sentence.
- Typical structure assumption: The POS labels form a Markov chain.
    - i.e. $y_{n+1} \mid y_n, y_{n-1,\ldots,}, y_0$ is the same as $y_{n+1} \mid y_n$

Brief Recap: Decision Tree, Random Forest and Adaboost

Decision Trees:

- Decision Trees Setup

  Goal: Find a tree that minimize the task loss (squared loss) within a given complexity.

  Problem: Finding the optimal binary tree is computationally intractable.

  Solution: *Greedy* algorithm.

  - Find the best split (according to some criteria) for a non-terminal node (initially the root)
  - Add two children nodes
  - Repeat until a stopping criterion is reached (max depth)
- Properties of Decision Trees
  - Non-linear classifier that recursively partitions the input space
  - Non-metric: make no use of geometry, i.e. no inner-product or distances
  - Non-parametric: make no assumption of the data distribution

Brief Recap: Decision Tree, Random Forest and Adaboost

Ensemble methods:

Ensemble methods:

- Combine outputs from multiple models.
  - Same learner on different datasets: ensemble + bootstrap = bagging.
  - Different learners on one dataset: they may make similar errors.
- Parallel ensemble: models are built independently, bagging
  - Reduce variance of a low bias, high variance estimator by ensembling many estimators trained in parallel.
- Sequential ensemble: models are built sequentially, boosting
  - Reduce the error rate of a high bias estimator by ensembling many estimators trained in sequential.
  - Try to add new learners that do well where previous learners lack

Brief Recap: Decision Tree, Random Forest and Adaboost

Random Forest:

Key idea of Random Forest: Use bagged decision trees, but modify the tree-growing procedure to reduce the dependence between trees.

- Build a collection of trees independently (in parallel).
- When constructing each tree node, restrict choice of splitting variable to a randomly chosen subset of features of size m.
    - Avoid dominance by strong features.
- Typically choose $m \approx \sqrt{p}$, where p is the number of features.
- Can choose m using cross validation.

Brief Recap: Decision Tree, Random Forest and Adaboost

Adaboost Algorithm:

- Training set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
- Start with equal weight on all training points $w_1 = \cdots = w_n = 1$.
- Repeat for $m = 1, \ldots, M$:
    - Base learner fits weighted training data and returns $G_m(x)$
    - Increase weight on the points $G_m(x)$ misclassifies
- Final prediction $G(x) = \operatorname{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$. (recall $G_m(x) \in \{-1, 1\}$)
- What are desirable α_m's?
    - nonnegative
    - larger when G_m fits its weighted $\mathcal{D}$ well
    - smaller when G_m fits weighted $\mathcal{D}$ less well

Brief Recap: Forward stagewise additive modeling, Gradient Boosting

- **FSAM**: a method used in boosting, greedily fit one function at a time without adjusting previous functions.
- **Learning with FSAM**: Optimizing one basis function each step and add it to the target function.
- **Optimization**: find the best basis function each step, uses gradient-based method. (details next slide.)
- Practice GBM with loss functions we discussed.
- **Note:** using exponential loss, GBM is the same as Adaboost.

Brief Recap: Forward stagewise additive modeling, Gradient Boosting

GBM in computing basis function: for each step

- compute the unconstrained gradient considering all training samples, i.e.

$$g = \nabla_{\boldsymbol{f}} J(\mathrm{f}) = \left(\partial_{\mathrm{f}_1} \ell\left(y_1, \mathrm{f}_1\right), \ldots, \partial_{\mathrm{f}_n} \ell\left(y_n, \mathrm{f}_n\right)\right)$$

- then, compute the basis function parameter within hypothesis space that has smallest Euclidean distance to the gradient, i.e.

$$h = \underset{h \in \mathcal{H}}{\arg\min} \sum_{i=1}^{n} \left(-g_i - h\left(x_i\right)\right)^2$$

- The step size can be predefined or learnt using line search. Finally, we have $f_m \leftarrow f_{m-1} + v_m h_m$

Brief Recap: Neural Networks

- **Intuition**: Learning intermediate features.
- **Optimization**: backpropagation, based on chain rule.
    - for final: look at partial derivative of affine transformation and activation/transfer functions
    - sigmoid, ReLU (subgradient), tanh, softmax
- **Note**: Revising the XOR example could be helpful!
- (optional) problem on NN optimization : risk of gradient exploding/vanishing.

Brief Recap: k-Means, GMM, Expectation Maximization

- **Differences K-Means v.s. GMM**:
  - Hard v.s. soft clustering (utilizes the density in Gaussian).
  - "circular" v.s. "oval-shaped" clusters
- **Optimization in GMM**: Expectation Maximization
- **Idea from Latent Variable Model**:
  - we want to compute $p(x)$
  - we start from $p(z)p(x|z)$, where $p(x|z)$ is modeled with parameters θ
  - we do not know $p(z)$, so we use another distribution $q(z)$ to approximate $p(z)$
  - try to get $\mathcal{L}(q,\theta) - \mathrm{KL}(q(z) \| p(z \mid x;\theta)) + \log p(x;\theta)$ by yourself!
  - we will test LVM in the final!
- **Expectation Maximization**:
  - E-step: we update $q(z)$ (GMM: the γ, you can think that π is defined by the γ)
  - M-step: we update parameters $p(x|z)$ of, i.e. θ. (GMM: μ, Σ)

Question 1: Bayesian

Bayesian Bernoulli Model
Suppose we have a coin with unknown probability of heads $\theta \in (0,1)$. We flip the coin n times and get a sequence of coin flips with n_h heads and n_t tails.
Recall the following: A Beta (α,β) distribution, for shape parameters $\alpha, \beta > 0$, is a distribution supported on the interval (0, 1) with PDF given by

$$f(x;\alpha,\beta) \propto x^{\alpha-1}(1-x)^{\beta-1}$$

The mean of a Beta (α,β) is $\frac{\alpha}{\alpha+\beta}$. The mode is $\frac{\alpha-1}{\alpha+\beta-2}$ assuming $\alpha, \beta \geqslant 1$ and $\alpha+\beta > 2$. If $\alpha = \beta = 1$, then every value in (0, 1) is a mode.

Question 1 Continued

1. Give an expression for the likelihood function $L_D(\theta)$ for this sequence of flips.
2. Suppose we have a Beta (α, β) prior on θ, for some $\alpha, \beta > 0$. Derive the posterior distribution on θ and, if it is a Beta distribution, give its parameters.
3. If your posterior distribution on θ is Beta(3, 6), what is your MAP estimate of θ?

Question 1 Solution

1. $L_D(\theta) = \theta^{n_h}(1-\theta)^{n_t}$
2. $$\begin{aligned} p(\theta \mid \mathcal{D}) &\propto p(\theta)L(\theta) \\ &\propto \theta^{\alpha-1}(1-\theta)^{\beta-1}\theta^{n_h}(1-\theta)^{n_t} \\ &\propto \theta^{n_h+\alpha-1}(1-\theta)^{n_t+\beta-1} \end{aligned}$$
3. Based on information box above, the mode of the beta distribution is $\frac{\alpha-1}{\alpha+\beta-2}$ for $\alpha, \beta > 1$. So the MAP estimate is $\frac{2}{7}$.

Question 2: Boostrap

1. What is the probability of not picking one datapoint while creating a bootstrap sample?
2. Suppose the dataset is fairly large. In an expected sense, what fraction of our bootstrap sample will be unique?

Question 2 Solution

1. $\left(1-\frac{1}{n}\right)^n$
2. As $n \to \infty$, $\left(1-\frac{1}{n}\right)^n \to \frac{1}{e}$. So $1-\frac{1}{e}$ unique samples.

Question 3: Random Forest and Boosting

Indicate whether each of the statements (about random forests and gradient boosting) is true or false.

1. True or False: If your gradient boosting model is overfitting, taking additional steps is likely to help
2. True or False: In gradient boosting, if you reduce your step size, you should expect to need fewer rounds of boosting (i.e. fewer steps) to achieve the same training set loss.
3. True or False: Fitting a random forest model is extremely easy to parallelize.
4. True or False: Fitting a gradient boosting model is extremely easy to parallelize, for any base regression algorithm.
5. True or False: Suppose we apply gradient boosting with absolute loss to a regression problem. If we use linear ridge regression as our base regression algorithm, the final prediction function from gradient boosting always will be an affine function of the input.

Question 3 Solution

False, False, True, False, True

Question 4: Neural Networks

1. **True or False**: Consider a hypothesis space $\mathcal{H}$ of prediction functions $f : \mathbb{R}^d \to \mathbb{R}$ given by a multilayer perceptron (MLP) with 3 hidden layers, each consisting of m nodes, for which the activation function is $\sigma(x) = cx$, for some fixed $c \in \mathbb{R}$. Then this hypothesis space is strictly larger than the set of all affine functions mapping $\mathbb{R}^d$ to $\mathbb{R}$.
2. **True or False**: Let $g : [0,1]^d \to \mathbb{R}$ be any continuous function on the compact set $[0,1]^d$. Then for any $\varepsilon > 0$, there exists $m \in \{1,2,3,\ldots\}$,

   $a = (a_1, \ldots, a_m) \in \mathrm{R}^m, b = (b_1, \ldots, b_m) \in \mathrm{R}^m$, and $W = \begin{pmatrix} - & w_1^T & - \\ \vdots & \vdots & \vdots \\ - & w_m^T & - \end{pmatrix} \in \mathbb{R}^{m \times d}$ for which

   the function $f : [0,1]^d \to \mathbb{R}$ given by

   $$f(x) = \sum_{i=1}^{m} a_i \max(0, w_i^T x + b_i)$$

   satisfies $|f(x) - g(x)| < \epsilon$ for all $x \in [0,1]^d$.

Question 4 Solutions

False, True

Question 5: Mixture Models

Suppose we have a latent variable $z \in \{1,2,3\}$ and an observed variable $x \in (0,\infty)$ generated as follows:

$$z \sim \text{Categorical}(\pi_1, \pi_2, \pi_3)$$

$$x \mid z \sim \text{Gamma}(2, \beta_z),$$

where $(\beta_1, \beta_2, \beta_3) \in (0,\infty)^3$, and $\text{Gamma}(2,\beta)$ is supported on $(0,\infty)$ and has density $p(x) = \beta^2 x e^{-\beta x}$. Suppose we know that $\beta_1 = 1, \beta_2 = 2, \beta_3 = 4$. Give an explicit expression for $p(z=1|x=1)$ in terms of the unknown parameters π_1, π_2, π_3.

Question 5 Solutions

$$p(z=1|x=1) \propto p(x=1|z=1)p(z=1) = \pi_1 e^{-1}$$
$$p(z=2|x=1) \propto p(x=1|z=2)p(z=2) = \pi_2 4e^{-2}$$
$$p(z=3|x=1) \propto p(x=1|x=3)p(z=3) = \pi_3 16e^{-4}$$

$$p(z=1|x=1) = \frac{\pi_1 e^{-1}}{\pi_1 e^{-1} + \pi_2 4e^{-2} + \pi_3 16e^{-4}}$$

Regularization strategies

universality of neural networks also means they can overfit

strategies for variance reduction:

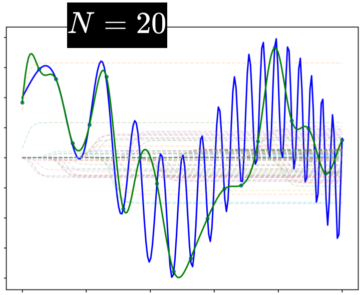
- L1 and L2 regularization *(weight decay)*
- data augmentation
- noise robustness
- early stopping
- bagging and dropout
- sparse representations *(e.g., L1 penalty on hidden unit activations)*
- semi-supervised and multi-task learning
- adversarial training
- parameter-tying

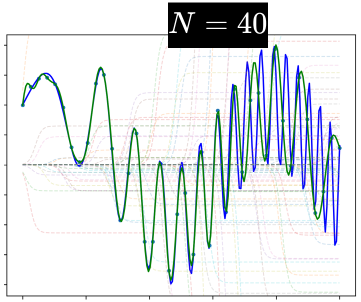
Data augmentation

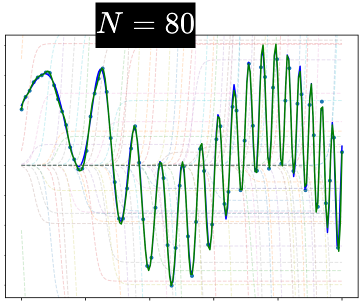
a larger dataset results in a better generalization

example: in all 3 examples below training error is close to zero

however, a larger training dataset leads to better generalization

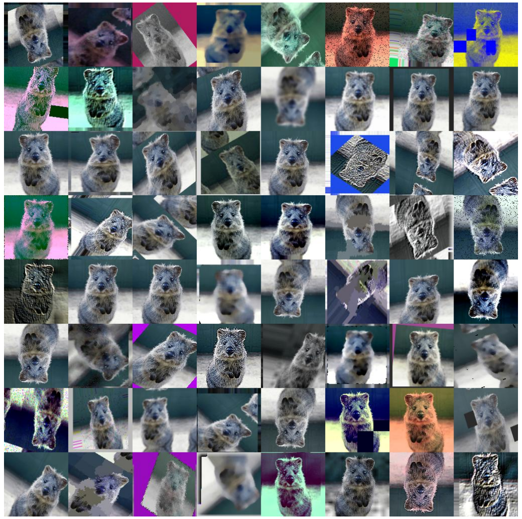






Data augmentation

a larger dataset results in a better generalization



idea

increase the size of dataset by adding reasonable transformations $\tau(x)$
that change the label in predictable ways; e.g., $f(\tau(x)) = f(x)$

special approaches to data-augmentation

- adding noise to the input
- adding noise to hidden units
  - noise in higher level of abstraction

- learn a **generative model** $\hat{p}(x, y)$ of the data
  - use $x^{(n')}, y^{(n')} \sim \hat{p}$ for training

sometimes we an achieve the same goal by designing the models
that are **invariant** to a given set of transformations

image: https://github.com/aleju/imgaug/blob/master/README.md

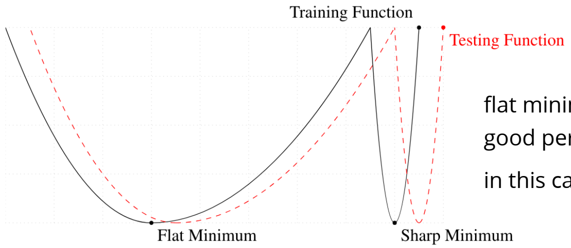
Noise robustness

make the model robust to noise in

input (data augmentation)

hidden units (*e.g.*, in dropout)

weights the loss is not sensitive to small changes in the weight (flat minima)



flat minima generalize better

good performance of SGD using small minibatch is attributed to flat minima

in this case, SGD regularizes the model due to **gradient noise**

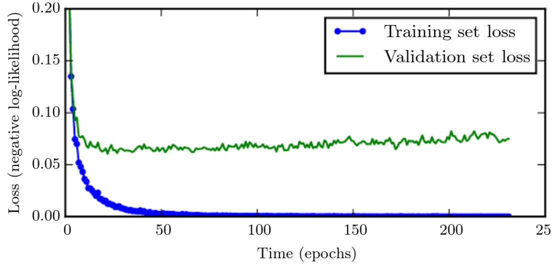
output (avoid overfitting, specially to wrong labels)

a heuristic is to replace hard labels with "soft-labels" label smoothing

$$\text{e.g., } [0,0,1,0] \rightarrow [\frac{\epsilon}{3}, \frac{\epsilon}{3}, 1-\epsilon, \frac{\epsilon}{3}]$$

image credit: Keshkar et al'17

Early stopping



the **test loss**-vs-**time step** is "often" U-shaped

use validation for early stopping

also saves computation!

early stopping bounds the region of the parameter-space that is reachable in T time-steps

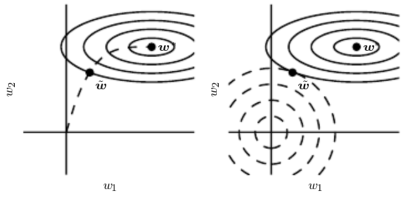
assuming

> *bounded gradient*
> *starting with a small w*

it has an effect similar to L2 regularization

we get the regularization path (various λ)

we saw a similar phenomena in boosting



Bagging

several sources of variance in neural networks, such as

- optimization
  - initialization
  - randomness of SGD
  - learning rate and other hyper-parameters
- choice of architecture
  - number of layers, hidden units, etc.

use bagging or even averaging without bootstrap to reduce variance

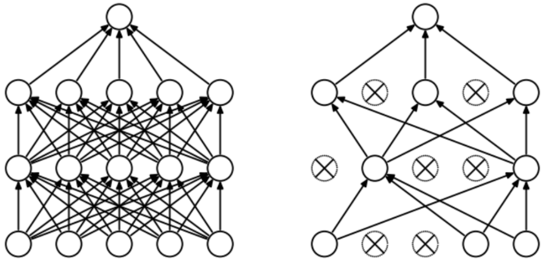
issue: computationally expensive

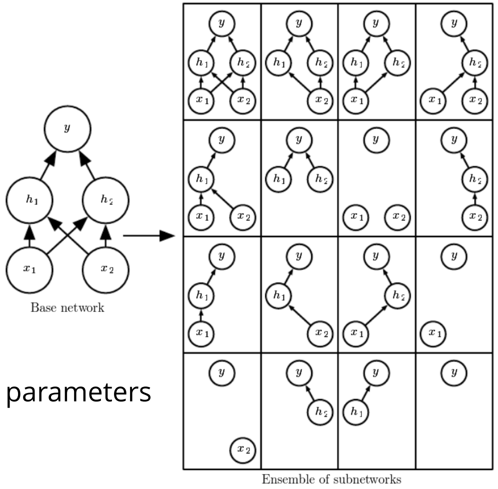
Dropout

idea

randomly remove a subset of units during training

as opposed to bagging a single model is trained





can be viewed as exponentially many subnetworks that share parameters

is one of the most effective regularization schemes for MLPs