Midterm Review

CMPUT 467: Intermediate Machine Learning

Comments

- Midterm on Chapters 1 - 9 (up to and including neural networks)
- The goal of the exam is to test (a) did you understand the basic ideas and (b) can you apply that understanding
- Brief Review
- Then Practice Midterm
- Then Q&A session. Consider also looking again at Review Slides for Ch 1-7

Chapters 1-4

- Covered in Previous Review
- Should be comfortable with
  - Basic probability info (won't be tested, just needed to understand the course)
  - Basic Optimization concepts, including first and second order gradient descent, SGD, vector stepsizes and momentum
  - Basic matrix operators, including weights that are matrices, matrix multiplication and SVD
  - The role of l2 regularization (in any GLM) and the bias-variance trade-off in linear regression
  - Understand why (and when) we might use SGD and GD, as well as first-order versus scond-order GD

Ch. 5: Generalized Linear Models

- Understand the purpose of the generalization from linear regression to GLMs
- Understand that the exponential family distribution underlies GLMs
- Know that linear regression, Poisson regression, logistic regression and multinomial logistic regression are examples of GLMs
- Know the distributions and transfers that correspond to each of these four GLMs
    - e.g., Poisson regression has a Poisson distribution p(y | x) with transfer exp
- Don't expect you to know most formulas, but expect at this point you know these four distributions and the transfers for each distribution

How do we use GLMs?

- In a GLM we learn E[Y | x], which fully characterizes our p(y | x)
  - Bernoulli, Poisson and Multinomial all only have one key parameter, which is E[Y]
  - Gaussian has mean and variance, but we assume the variance is fixed and that we not learning it; so its key param is also only E[Y | x]
- How do we use GLMs for prediction?
  - Mode of p(y|x) is a reasonable answer
  - Mean E[Y | x] is also a reasonable answer

Why mode or mean?

- We will suffer a cost for our prediction
  - recall: we want to minimize expected cost
- If we picked a squared cost, then the best choice was E[Y | x]
- If we picked a 0-1 cost, then the best choice was argmax p(y | x) (mode)

Chapter 6: Constrained Optimization

- Optimization of the form $\min_{\mathbf{w} \in \mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$ for smooth c, nonsmooth r
- Example: c is squared errors and r is box constraints
- Smooth means differentiable everywhere

Questions for optimization $\min_{\mathbf{w} \in \mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$

- What is c and what is r for linear regression + l1 regularization?
- What is c and what is r for **logistic regression** + l1 regularization?
- What is c and what is r for linear regression + l2 regularization + l1 regularization?

Questions for optimization $\min_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$

- What is c and what is r for linear regression + l1 regularization?
- $c(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i\mathbf{w} - y_i)^2$ and $r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$
- What is c and what is r for **logistic regression** + l1 regularization?
- What is c and what is r for linear regression + l2 regularization + l1 regularization?

Questions for optimization $\min_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$

- What is c and what is r for linear regression + l1 regularization?
- $c(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} (\mathbf{x}_i\mathbf{w} - y_i)^2$ and $r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$
- What is c and what is r for **logistic regression** + l1 regularization?
- $c(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} \left[-y_i \ln \sigma(\mathbf{x}_i\mathbf{w}) - (1 - y_i)\ln y_i \ln(1 - \sigma(\mathbf{x}_i\mathbf{w}))\right]$ and $r(\mathbf{w}) = r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$
- What is c and what is r for linear regression + l2 regularization + l1 regularization?

Questions for optimization $\min_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$

- What is c and what is r for linear regression + l1 regularization?
- $c(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i\mathbf{w} - y_i)^2$ and $r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$
- What is c and what is r for **logistic regression** + l1 regularization?
- $c(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left[-y_i \ln\sigma(\mathbf{x}_i\mathbf{w}) - (1-y_i)\ln y_i \ln(1-\sigma(\mathbf{x}_i\mathbf{w}))\right]$ and $r(\mathbf{w}) = r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$
- What is c and what is r for linear regression + l2 regularization + l1 regularization?
- $c(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i\mathbf{w} - y_i)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$ and $r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$

Chapter 6: Constrained Optimization

- Optimization of the form $\min_{\mathbf{w} \in \mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$ for smooth c, nonsmooth r
- Proximal update: $\mathbf{w}_{t+1} = \text{prox}_{\eta_t r}(\mathbf{w}_t - \eta_t \nabla c(\mathbf{w}_t))$
- **Do not need to know**
  - Specific proximal operators; just need to know where to use the given proximal operator
  - How to use vector stepsizes or momentum; we only did scalar stepsizes
  - I will not get you to derive solutions with Lagrangians

Chapter 6: Constrained Optimization

- Optimization of the form $\min_{\mathbf{w} \in \mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$ for smooth c, nonsmooth r
- Proximal update: $\mathbf{w}_{t+1} = \text{prox}_{\eta_t r}(\mathbf{w}_t - \eta_t \nabla c(\mathbf{w}_t))$
- **You should know**
  - That we used proximal gradient descent for l1 regularization
  - That we do not always have closed-form solutions for the proximal operator, and sometimes have to solve a simple optimization to get the projection step (proximal operator), as in Section 6.3

Exercise: l1 regularization and independent features

- Imagine we have a feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^\top$
- Imagine y is independent of x_2 and dependent on x_6
- Imagine we have 1000 samples and d = 30
- If we use l1-regularization, what might happen?
- If we don't use any regularization, what might happen?

Chapter 7: Estimating GE and Cross Validation

- Goal is to estimate generalization error (GE) for a learned function f

Question about GE

- What is the generalization error for a linear regression model?
- What is the generalization error for a logistic regression model?
- What is the generalization error for a multinomial logistic regression model?
- [Extra Q] What is the generalization error for a Poisson regression model?

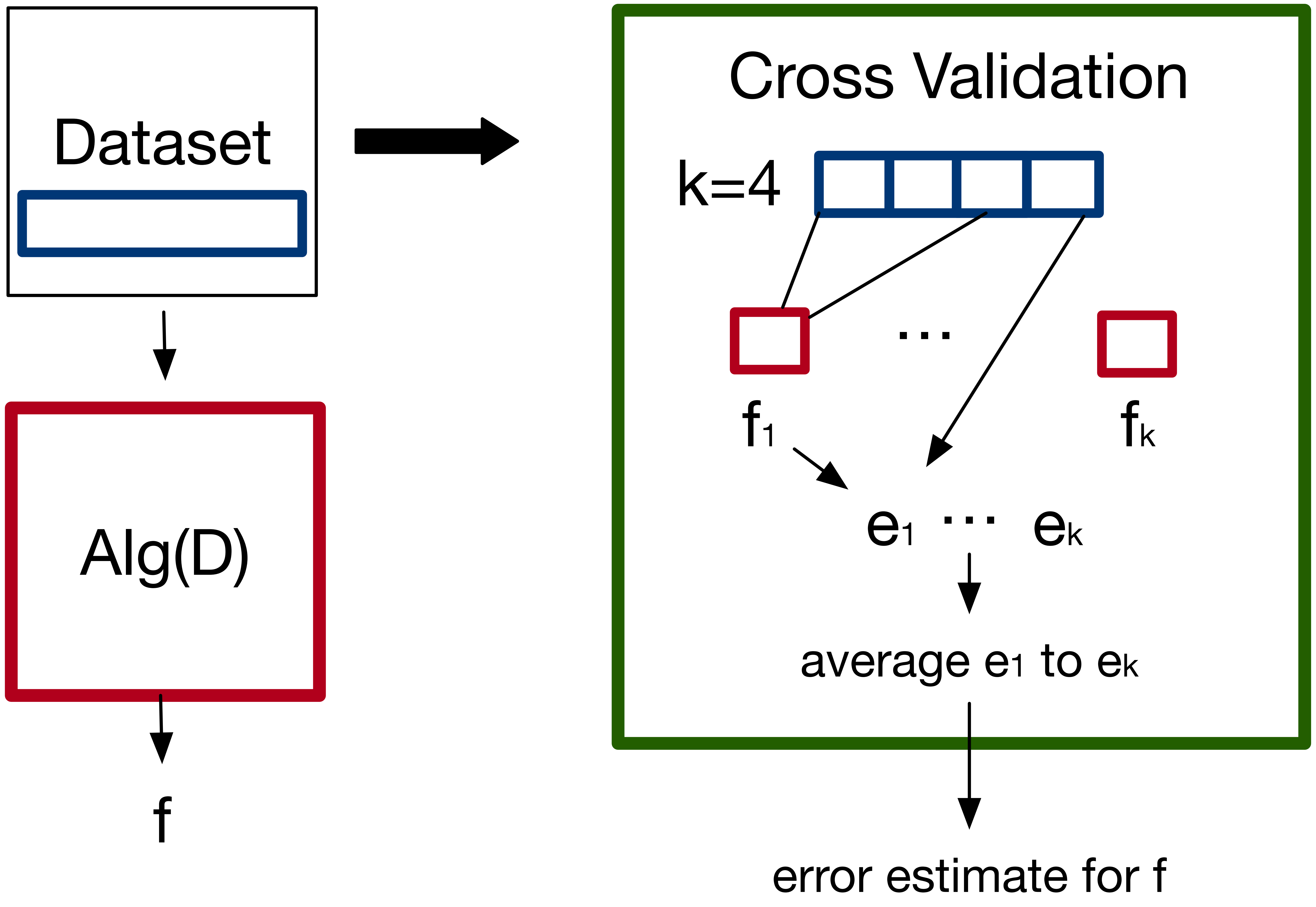
Question about GE

- What is the generalization error for a linear regression model?
    - $GE(f) = \mathbb{E}[(f(X) - Y)^2]$
- What is the generalization error for a logistic regression model?
    - $GE(f) = \mathbb{E}[\mathbf{1}(f(X) == Y)]$
- What is the generalization error for a multinomial logistic regression model?
    - $GE(f) = \mathbb{E}[\mathbf{1}(f(X) == Y)]$
- [Extra Q] What is the generalization error for a Poisson regression model?
    - Typically use $GE(f) = \mathbb{E}[(f(X) - Y)^2]$

Chapter 7: Estimating GE and Cross Validation

- Goal is to estimate generalization error (GE) for a learned function f
- Having a training and testing split can be data inefficient
- Cross-validation lets us use the training data for training and evaluation

Cross validation



Step 1: Learn f on the entire dataset

Step 2: Do CV to estimate the GE for f

Step 2 consists of
1. Get k partitions of the dataset, to get k training and test splits

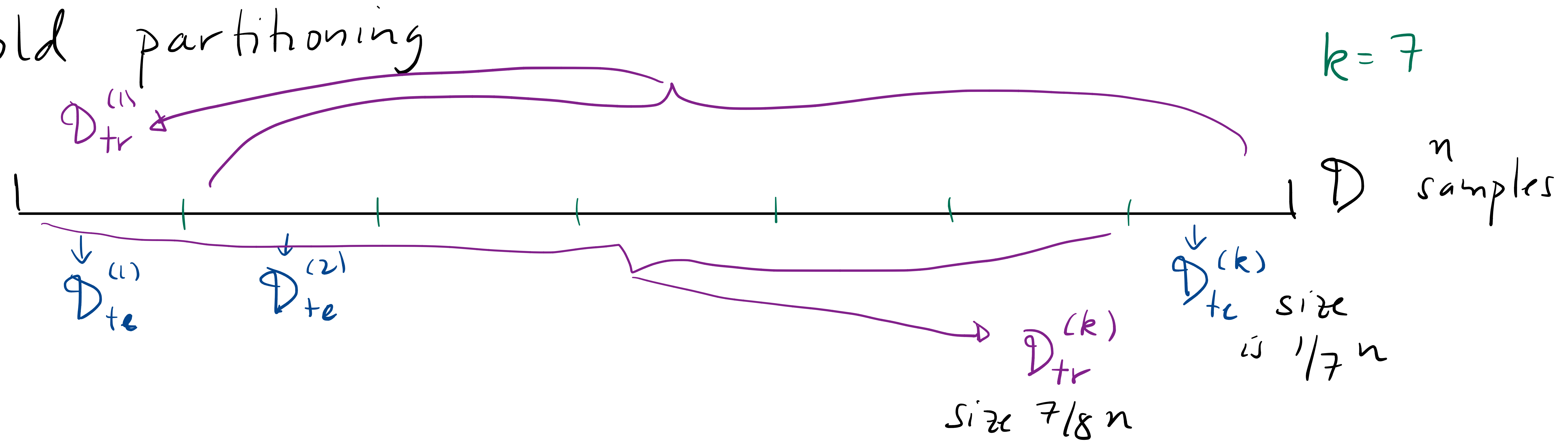
2. For every i = 1 to k,
train $f_i = \text{Alg}(\mathcal{D}_{tr}^{(i)})$ and
compute error e_i on $\mathcal{D}_{test}^{(i)}$

3. Get average error $\frac{1}{k}\sum_i e_i$

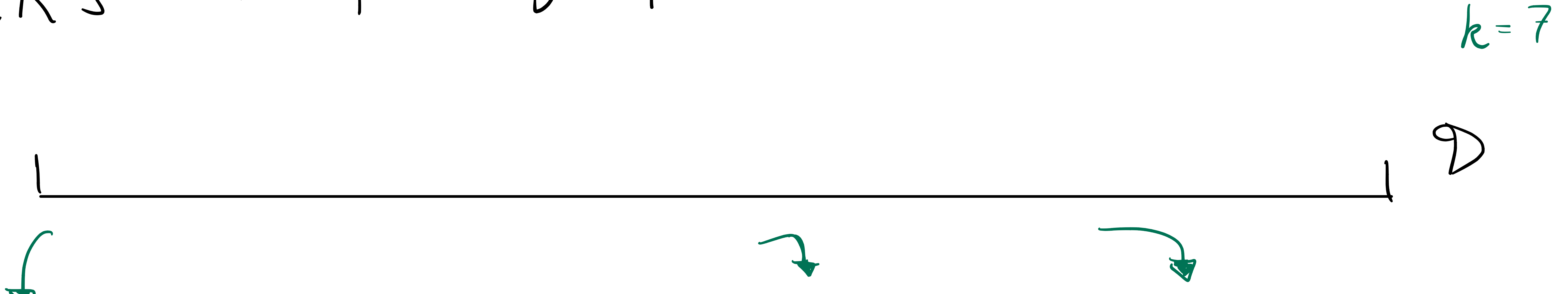
k-fold vs RSS

- Partition means disjoint subsets that cover the data
- k-fold is one way to get partitioning
  - Partition data into k folds/chunks
  - Each fold is set to a test dataset, the training is union of the remaining folds
- Repeated random subsampling (RSS) is another way to get a partitioning
  - Randomly sample points for test dataset (without replacement), and set the rest to the training set
  - Have to specify percentage for test p and number repeats k

k-fold partitioning



RRS with percentage p for test



Randomly shuffle $\mathcal{D}$
Set first $(1-p)n$ as $\mathcal{D}_{tr}^{(1)}$
Set last pn as $\mathcal{D}_{te}^{(1)}$

. . .

Randomly shuffle $\mathcal{D}$
Set first $(1-p)n$ as $\mathcal{D}_{tr}^{(k)}$
Set last pn as $\mathcal{D}_{te}^{(k)}$

How do we pick k and p?

- For **lower bias** pick **k large** for k-fold and **p smaller** for RRS
    - Bigger k means training set size (k-1)/k n closer to full dataset size n
    - Smaller p means training set size (1-p) n closer to full dataset size n
    - Each f_i more similar to f learned on all the data

How do we pick k and p?

- For **lower bias** pick **k large** for k-fold and **p smaller** for RRS
- **But** variance can increase with large k for k-fold or smaller p for RRS, as variance of errors larger (error is computed with smaller # of testing samples)
  - And for large k/smaller p likely more covariance between errors
- Finally, large k is computationally expensive, so rarely set very big
- No clear answers, just some rules of thumb, usually pick interim k (e.g., k=10)

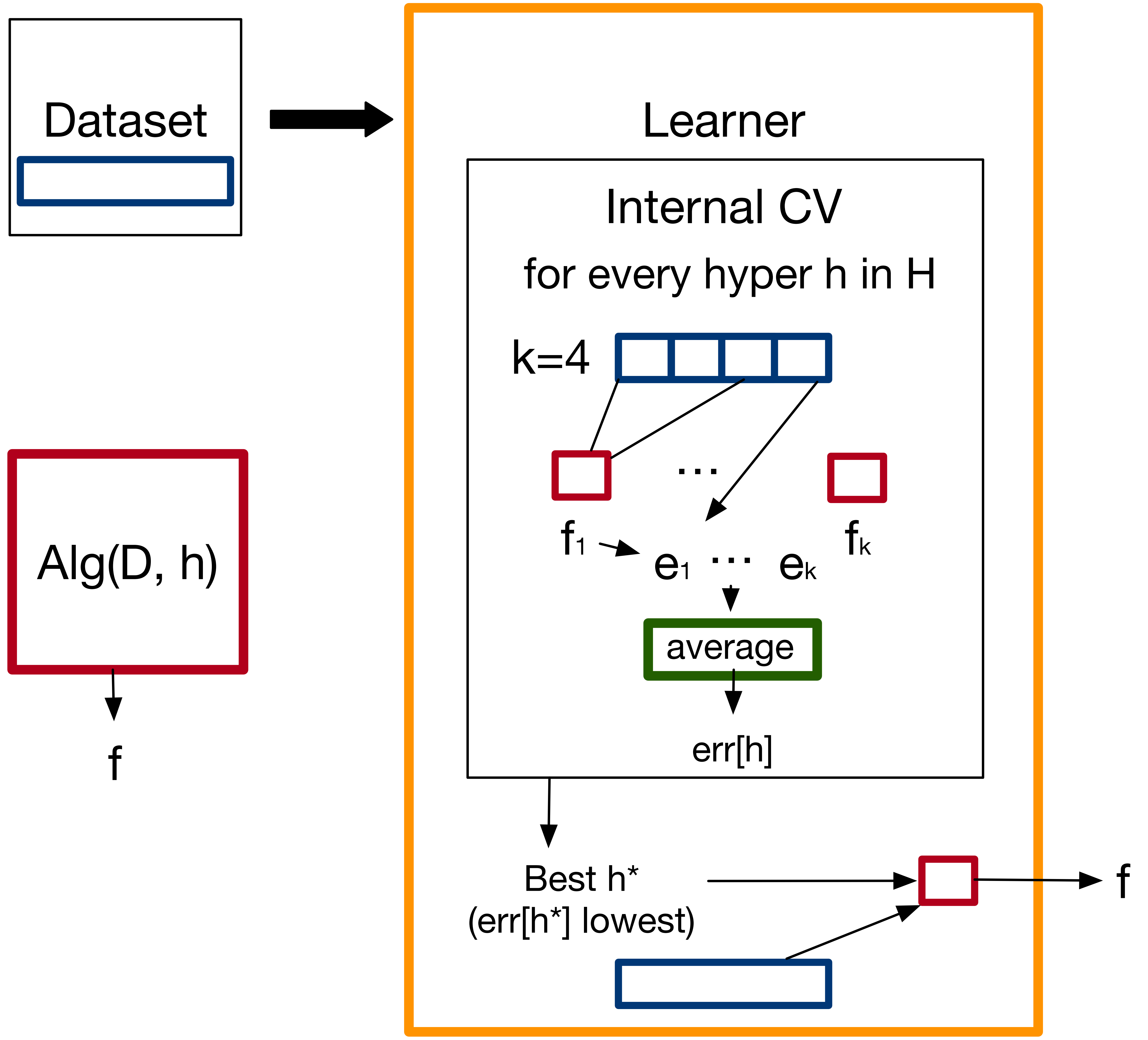
Chapter 7: Estimating GE and Cross Validation

- Goal is to estimate generalization error (GE) for a learned function f
- Having a training and testing split can be data inefficient
- Cross-validation let's us use the training data for training and evaluation
- k-fold and RSS as two partitioning approaches
- **You do not need to know**
    - All the sources of bias and variance in CV, just know that our estimator is biased and that the choice of k (and p) can impact bias and variance

Chapter 7: CV for hyperparameter selection

- Our estimate of (GE) is a good criteria to pick hyperparameters

CV for hyper selection



Chapter 7: CV for hyperparameter selection

- Our estimate of (GE) is a good criteria to pick hyperparameters
- We still need to evaluate the model produce by Learner
- Can use training / validation set to evaluate it
    - Step 0: Split data into training $\mathscr{D}_{\text{tr}}$ and validation set $\mathscr{D}_{\text{test}}$
    - Step 1: Call Learner on dataset $\mathscr{D}_{\text{tr}}$, to get function f
    - Step 2: Evaluate f on $\mathscr{D}_{\text{test}}$

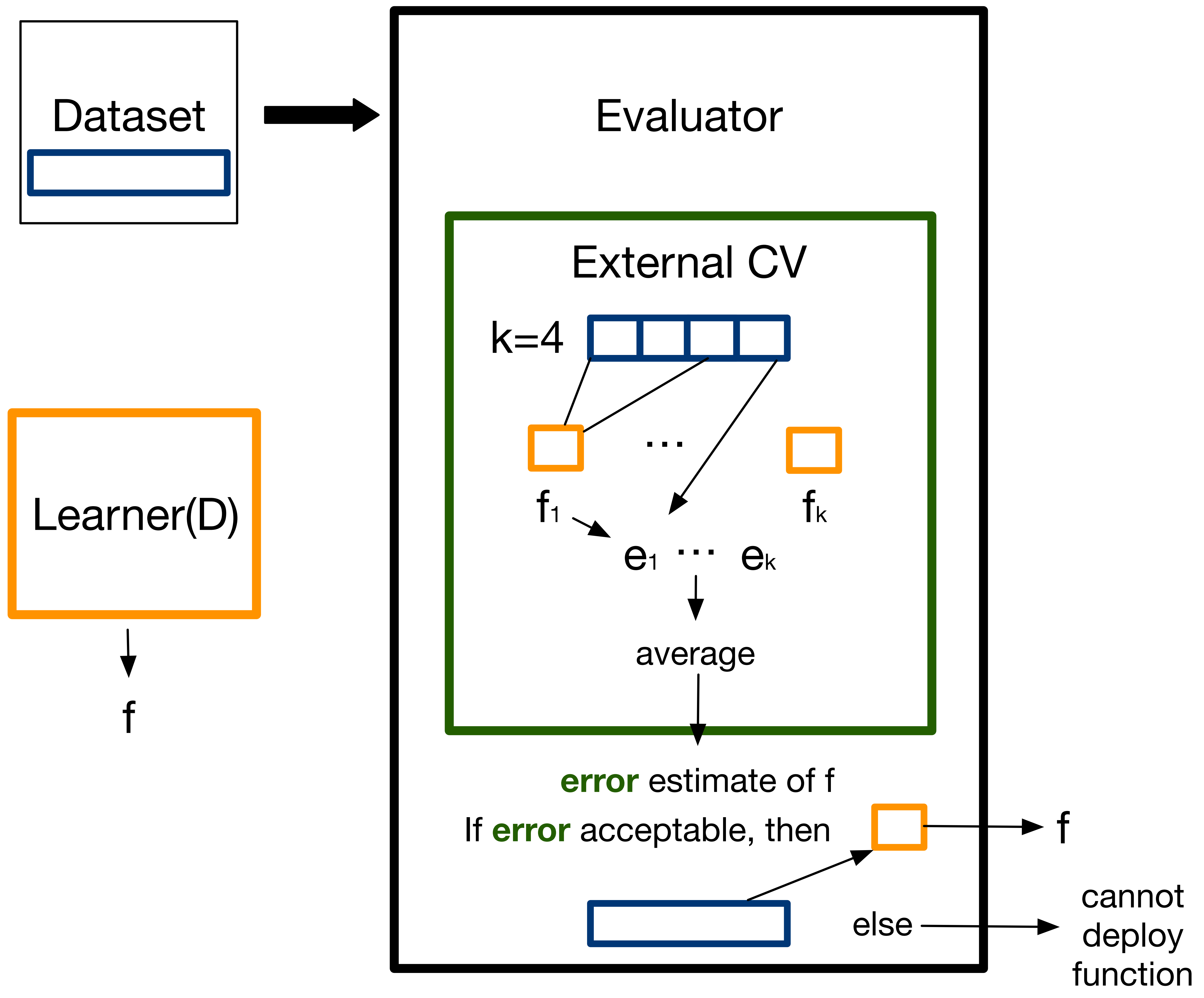
Chapter 7: CV for hyperparameter selection

- Our estimate of (GE) is a good criteria to pick hyperparameters
- We still need to evaluate the model produce by Learner
- Can use training / validation set to evaluate it
  - Step 0: Split data into training $\mathscr{D}_{\text{tr}}$ and validation set $\mathscr{D}_{\text{test}}$
  - Step 1: Call Learner on dataset $\mathscr{D}_{\text{tr}}$, to get function f
  - Step 2: Evaluate f on $\mathscr{D}_{\text{test}}$
- What is the issue with this approach?

Chapter 7: CV for hyperparameter selection

- Our estimate of (GE) is a good criteria to pick hyperparameters
- We still need to evaluate the model produce by Learner
- Can use training / validation set to evaluate it
  - Step 0: Split data into training $\mathscr{D}_{\text{tr}}$ and validation set $\mathscr{D}_{\text{test}}$
  - Step 1: Call Learner on dataset $\mathscr{D}_{\text{tr}}$, to get function f
  - Step 2: Evaluate f on $\mathscr{D}_{\text{test}}$
- What is the issue with this approach? Data inefficient, let's use CV!

Nested Cross-Validation



Step 1: Learn f on the entire dataset

Step 2: Do CV to estimate the GE for f

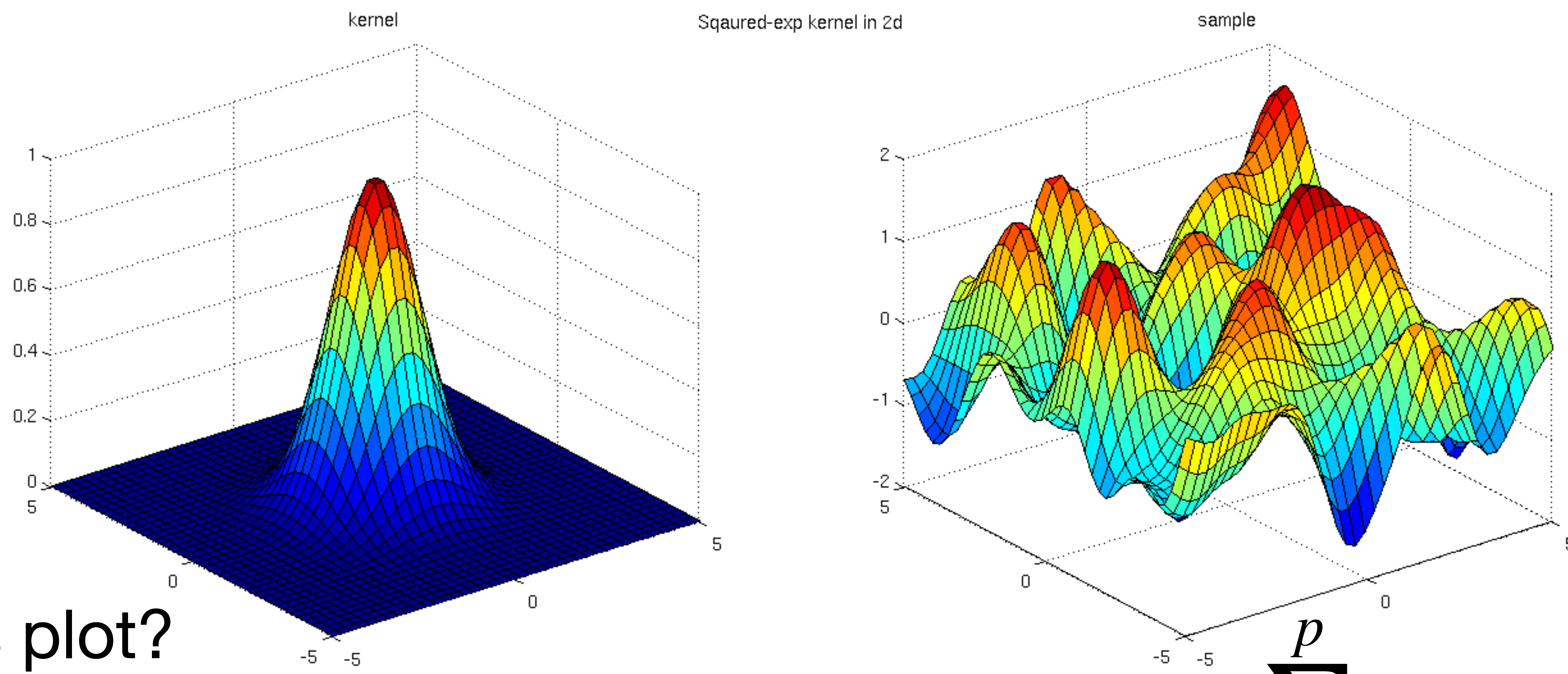
Step 2 consists of
1. Get k partitions of the dataset, to get k training and test splits

2. For every i = 1 to k,
train $f_i = \text{Alg}(\mathcal{D}_{tr}^{(i)})$ and
compute error e_i on $\mathcal{D}_{test}^{(i)}$

3. Get average error $\frac{1}{k}\sum_i e_i$

Chapter 8: Fixed Representations

- We discussed polynomials, RBF Networks and Prototype representations
- $$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} k(\mathbf{x}, \mathbf{c}_1) \\ k(\mathbf{x}, \mathbf{c}_2) \\ \vdots \\ k(\mathbf{x}, \mathbf{c}_p) \end{bmatrix}$$



What is $\mathbf{x}$ and $\mathbf{c}$ in this plot?

RBF kernel $k(\mathbf{x}, \mathbf{c})$

Function $f(\mathbf{x}) = \sum_{j=1}^{p} w_i k(\mathbf{x}, \mathbf{c}_j)$

Chapter 8: Fixed Representations

- We discussed polynomials, RBF Networks and Prototype representations
- **Question**: what is the difference between RBF Networks and Prototype representations that use an RBF kernel?

Chapter 8: Fixed Representations

- We discussed polynomials, RBF Networks and Prototype representations
- **Question**: what is the difference between RBF Networks and Prototype representations that use an RBF kernel?
- **Answer**: Prototype Rep + RBF kernel is an instance of an RBF network where the centers are prototypes (samples from the training dataset)
- Why do we use the data as centers?

Chapter 8: Fixed Representations

- We discussed polynomials, RBF Networks and Prototype representations
- We discussed how l1 regularization pushes weights to zero and also allows us to subselect prototypes
- **You do not need to know**
  - Any representability results for these functions
  - You just need to know that they let us learn nonlinear functions

Chapter 9: Learning Latent Factors

- Understand that PCA extracts a lower-dimensional representation $\mathbf{h}$ for $\mathbf{x}$
- Understand that sparse coding extracts a higher-dimensional, sparse representation $\mathbf{h}$
- Understand that for both we are trying to solve $\mathbf{x} \approx \mathbf{hD}$
- For both we try to minimize $||\mathbf{x} - \mathbf{hD}||_2^2$ for all $\mathbf{x}$, but for sparse coding we additionally add a sparsity regularizer to $\mathbf{h}$, namely $||\mathbf{h}||_1$

Chapter 9: Learning Latent Factors

- Understand that PCA extracts a lower-dimensional representation $\mathbf{h}$ for $\mathbf{x}$
- Understand that sparse coding extracts a higher-dimensional, sparse representation $\mathbf{h}$
- **You do not need to know**
    - The exact formulas for the optimizations; I will give them to you. But you should know how to reason about minimizing them
    - You do not need to know the probabilistic PCA solution, nor the closed-form PCA solution

PCA representation

- To learn the PCA weights $\mathbf{D}$ with $p < d$, we optimize the objective
$$\min_{\mathbf{h}_1,\ldots,\mathbf{h}_n \in \mathbb{R}^p, \mathbf{D} \in \mathbb{R}^{p \times d}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{h}_i \mathbf{D}\|_2^2$$
- For a new datapoint $\mathbf{x}_{\text{new}}$, we get the representation $\mathbf{h}_{\text{new}} = \arg\min_{\mathbf{h} \in \mathbb{R}^p} \|\mathbf{x}_{\text{new}} - \mathbf{h}\mathbf{D}\|_2^2$. where $\mathbf{h}_{\text{new}} = \mathbf{x}_{\text{new}} \mathbf{V}_p$ for projection$\mathbf{V}_p$ the top right singular vectors of training data matrix $\mathbf{X}$

Exercise Question

- Imagine we have 5000 datapoints for a problem with d = 10
- Imagine we first expand the dimension using a kernel representation, going from 10 features to 5000.
    - Subquestion: why are there 5000 features?
- Then we apply PCA to extract 100 features. How do we interpret what those features are?

Increasing p

- If $p > d$, then the objective $\min_{\mathbf{h}_1,\ldots,\mathbf{h}_n \in \mathbb{R}^p, \mathbf{D} \in \mathbb{R}^{p\times d}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{h}_i \mathbf{D}\|_2^2$ produces trivial solutions $\mathbf{D} = \mathbf{I}$ and $\mathbf{h}_i = \mathbf{x}_i$
- Add a regularizer on $\mathbf{h}_i$ and $\mathbf{D}$ to avoid trivial solutions
- Sparse coding we put an ℓ_1 regularizer on $\mathbf{h}_i$ to encourage sparse representations

Chapter 9: Learning Neural Networks

- Understand types of transformation on the input given by a neural network
  - series of linear functions composed with simple activations
- Understand that backpropagation is gradient descent
- Understand that linear autoencoders also extract a low-dimensional representation like PCA
- **Will not be directly tested:**
  - You will not need to derive the gradients for an NN
  - You will not be tested on supervised autoencoders

Exercise: NN choices

- An NN with three layers transforms the inputs as
  - $f_{\mathbf{w}}(\mathbf{x}) = f_1(f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})\mathbf{W}^{(1)})$ for weights $\mathbf{w}$ composed of $\mathbf{W}^{(3)}, \mathbf{W}^{(2)}, \mathbf{W}^{(1)}$
- Can think of this NN as learning p(y | x) with key parameter $\theta(\mathbf{x}) = \mathbf{h}^{(1)}\mathbf{W}^{(1)}$ for $\mathbf{h}^{(1)} = f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})$
- We pick a GLM loss and transfer f_1 for the output that matches the targets
  - e.g., what if the output is a binary 0,1 variable? What is f1?
  - e.g., what if the output is ordinal 0, 1, 2, 3, 4, 5, .., 100? What is f1?

Exercise: NN vs PCA

- An NN with three layers transforms the inputs as
  - $f_{\mathbf{w}}(\mathbf{x}) = f_1(f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})\mathbf{W}^{(1)})$ for weights $\mathbf{w}$ composed of $\mathbf{W}^{(3)}, \mathbf{W}^{(2)}, \mathbf{W}^{(1)}$
- Can think of this NN as learning p(y | x) with key parameter $\theta(\mathbf{x}) = \mathbf{h}^{(1)}\mathbf{W}^{(1)}$ for $\mathbf{h}^{(1)} = f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})$
- Can think of $\mathbf{h}^{(1)}$ as the new representation of $\mathbf{x}$. How do we extract the new representation for a new $\mathbf{x}_{\text{new}}$?

Exercise: NN vs PCA

- An NN with three layers transforms the inputs as
  - $f_{\mathbf{w}}(\mathbf{x}) = f_1(f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})\mathbf{W}^{(1)})$ for weights $\mathbf{w}$ composed of $\mathbf{W}^{(3)}, \mathbf{W}^{(2)}, \mathbf{W}^{(1)}$
- Can think of this NN as learning p(y | x) with key parameter $\theta(\mathbf{x}) = \mathbf{h}^{(1)}\mathbf{W}^{(1)}$ for $\mathbf{h}^{(1)} = f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})$
- Can think of $\mathbf{h}^{(1)}$ as the new representation of $\mathbf{x}$. How do we extract the new representation for a new $\mathbf{x}_{\text{new}}$? Ans $\mathbf{h}_{\text{new}} = f_2(f_3(\mathbf{x}_{\text{new}}\mathbf{W}^{(3)})\mathbf{W}^{(2)})$

PCA equivalent to a linear autoencoder

- Can also learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$, called a (linear) autoencoder that is a neural network with
  - identity activations
  - smaller hidden dimension $p < d$
  - loss function equal to $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{x}_i\|_2^2$
- For a new datapoint $\mathbf{x}_{\text{new}}$, we get the representation. $\mathbf{h} = \mathbf{x}\mathbf{W}^{(2)}$
- Produces same representation as PCAs $\mathbf{h}_{\text{new}} = \arg\min_{\mathbf{h} \in \mathbb{R}^p} \|\mathbf{x} - \mathbf{h}\mathbf{D}\|_2^2$

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if pick hidden dimension $p > d$?

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if pick hidden dimension $p > d$?
  - Learn linear function because $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ is the same as learning $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}$ for $\mathbf{W} \in \mathbb{R}^{d \times d}$, and get $\mathbf{W} = \mathbf{I}$
- But wait, we can always write $\mathbf{W} = \mathbf{W}^{(2)}\mathbf{W}^{(1)} \in \mathbb{R}^{d \times d}$, so whats different?

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if pick hidden dimension $p > d$?
  - Learn linear function because $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ is the same as learning $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}$ for $\mathbf{W} \in \mathbb{R}^{d \times d}$, and get $\mathbf{W} = \mathbf{I}$
- But wait, we can always write $\mathbf{W} = \mathbf{W}^{(2)}\mathbf{W}^{(1)} \in \mathbb{R}^{d \times d}$, so whats different?
  - For $p < d$, we are restricting $\mathbf{W} = \mathbf{W}^{(2)}\mathbf{W}^{(1)} \in \mathbb{R}^{d \times d}$ to be low-rank
  - More constrained linear function to reconstruct $\mathbf{x}$, don't get $\mathbf{W} = \mathbf{I}$

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if keep $p < d$ but change the output target of NN to scalar y and loss function from $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{x}_i\|_2^2$ to $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - y_i\|_2^2$?

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if keep $p < d$ but change the output target of NN to scalar y and loss function from $\sum_{i=1}^{n} ||f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{x}_i||_2^2$ to $\sum_{i=1}^{n} ||f_{\mathbf{w}}(\mathbf{x}_i) - y_i||_2^2$?
  - Learn linear function because $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times 1}$ is same as learning $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}$ for $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times 1}$

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if keep $p < d$ but change the output target of NN to scalar y and loss function from $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{x}_i\|_2^2$ to $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - y_i\|_2^2$?
- What about multinomial logistic regression $f_{\mathbf{w}}(\mathbf{x}) = \text{softmax}(\mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)})$ and $p > m$?

Questions about linear autoencoders

- Can learn $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$
- What happens if keep $p < d$ but change the output target of NN to scalar y and loss function from $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - \mathbf{x}_i\|_2^2$ to $\sum_{i=1}^{n} \|f_{\mathbf{w}}(\mathbf{x}_i) - y_i\|_2^2$?
- What about multinomial logistic regression $f_{\mathbf{w}}(\mathbf{x}) =$ softmax$(\mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)})$ and $p > m$?
  - Learn linear function because $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times p}$ and $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times 1}$ is same as learning $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}$ for $\mathbf{W}^{(2)} \in \mathbb{R}^{d \times 1}$