Final Review

CMPUT 467/567: Machine Learning II

Winter 2026
Martha White

Goal of these Slides

- Go over each section of the notes and highlight key concepts
- Additionally highlight what I will and will not test
  - It is in the notes for your knowledge, but hard to directly test
- Practice Final will be covered in a later session, closer to the actual final
- Note: the final largely focuses on Chapter 10 onwards. But as usual it builds on your knowledge from earlier chapters

Chapter 1: Intro to ML

- Know the difference between a generative model and predictor (1.1)
- **Will not be directly tested:**
  - Relationship to Statistics and Probability (1.2)
  - The Blessing and Curse of Dimensionality (1.3)
  - SVDs and Eigenvalue decompositions (1.4)

Chapter 2: Probability Concepts

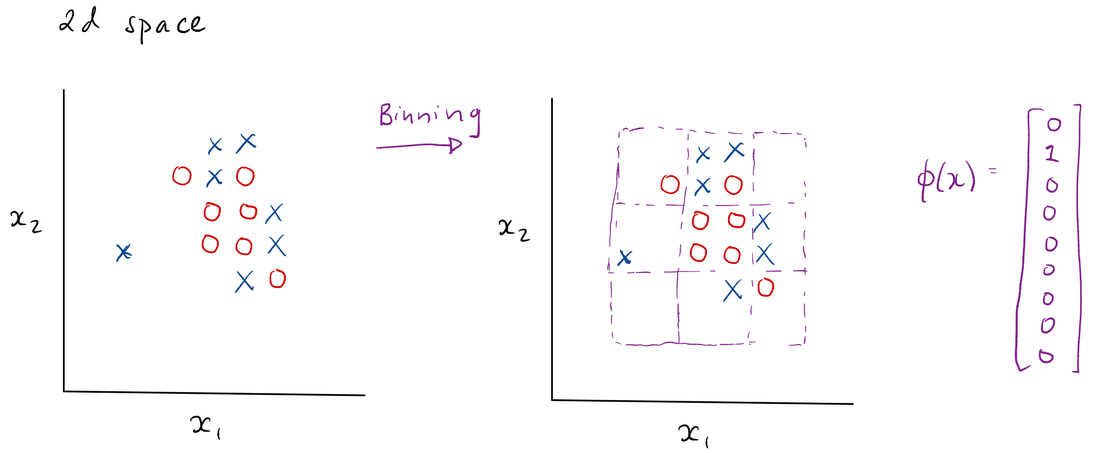
- Understand the definition of a multi-dimensional probability (2.1)
- Understand the definition of a mixture of distributions (2.2)
- Know the purpose of the KL divergence (2.3)
- **Will not be directly tested:**
  - Knowing the PMFs or PDFs of specific distributions
  - Specific expectation and variance formulas
  - Remembering the KL divergence formula

Chapter 3: Fixed Representations

- We discussed polynomials, RBF Networks and Prototype representations
- We discussed how these fixed representations allow us to learn nonlinear functions
- **Will not test**
  - Any representability results for these functions
  - Specific formulas for kernels

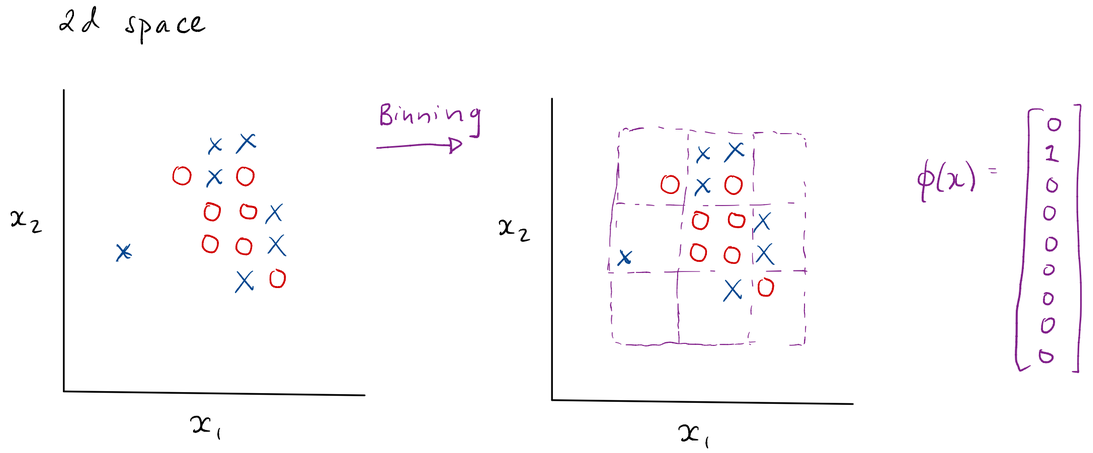
Exercise

- I didn’t give you an example of how projecting to higher dimensions also facilitates regression with simple (linear functions)
- Can you think of a similar example to this one, but for regression?



Exercise

- I didn't give you an example of how projecting to higher dimensions also facilitates regression with simple (linear functions)
- Can you think of a similar example to this one, but for regression?
  - Same ϕ lets us learn an average y per bin
  - Can also use a linear function per bin, by using (x1,x2) per bin, giving piecewise linear functions
    - e.g., bin 1: [x1, x2, 0, 0, 0, ….]
      bin 2: [0, 0, x1, x2, 0, 0, ….]



Ch 4: Revisiting Generalization Error

- The definition of generalization error (expected cost)
- Overfitting and underfitting, and relationship to function class capacity
- Understand that Linear Regression and l2-regularized linear regression have closed-form solutions (unlike most GLMs)
- Understand that this let's us characterize the bias and variance of these solutions
- Understand the LR solution is unbiased, if the true function is linear
- Understand LR+l2 is biased, but that asymptotically (as n grows) they reach the same solution
- Understand difference between realizable and nonrealizable settings
- **Will not be directly tested:**
  - Any specific closed-form solutions; I will give them to you if you need them

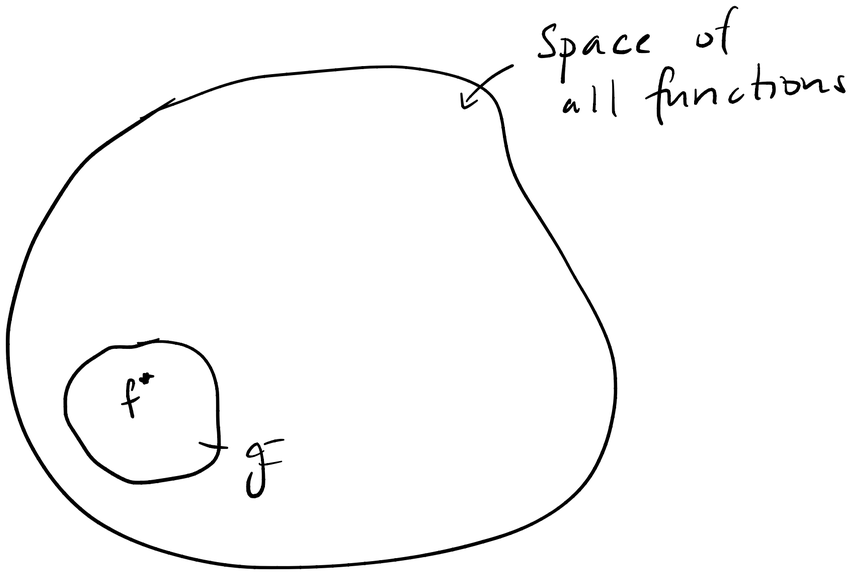
Ch4: Revisiting Generalization Error

- Understand that the generalization error of a function f is the error in expectation across all possible datapoints (expected cost)

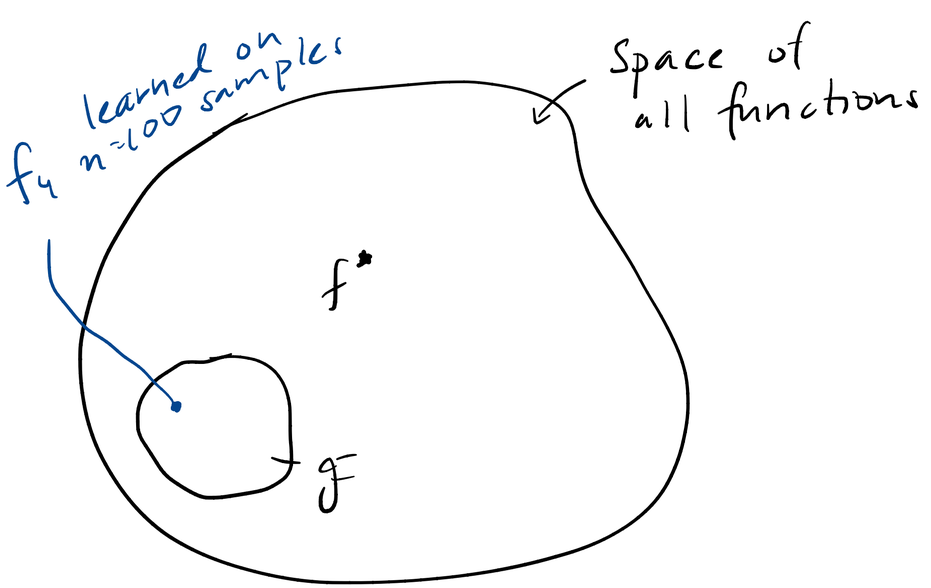
$$\mathrm{GE}(f) = \mathbb{E}[(f(X) - Y)^2] = \underbrace{\mathbb{E}[(f(X) - f^*(X))^2]}_{\text{reducible error}} + \underbrace{\mathbb{E}[(f^*(X) - Y)^2]}_{\text{irreducible error}}$$

- GE is about a specific function f, rather than about a function class where we have $f_{\mathscr{D}}$ that varies with data

Realizable vs Nonrealizable



(a) Realizable setting: $f^* \in \mathcal{F}$



(b) Non-realizable setting: $f^* \notin \mathcal{F}$

- For linear regression, assuming true f* uses the same features, we were able to directly compare learned w to w*. Is this the realizable or nonrealizable setting?

  $\mathbb{E}[\mathbf{w}(\mathscr{D})]$ compared to $\mathbf{w}^*$

- What is the definition for an estimator being unbiased more generally, if we do not compare weights?

Recall bias-variance for a function class

- Understand that we can reason about function $f_{\mathscr{D}}$ as a random variable, where randomness comes from the underlying dataset
- Understand that we can reason about expected generalization error (across datasets) of a function class, by considering the bias and variance of this $f_{\mathscr{D}}$
- Understand that reducible error of $f_{\mathscr{D}}$ decomposes into bias and variance
- For a specific x, we have

$$\mathbb{E}\left[\left(f_{\mathcal{D}}(\mathbf{x}) - f^*(\mathbf{x})\right)^2\right] = \left(\mathbb{E}\left[f_{\mathcal{D}}(\mathbf{x})\right] - f^*(\mathbf{x})\right)^2 + \mathrm{Var}\left[f_{\mathcal{D}}(\mathbf{x})\right]$$

Exercise: bias-variance across x

- For a specific x, we have

$$\mathbb{E}\left[\left(f_{\mathcal{D}}(\mathbf{x}) - f^*(\mathbf{x})\right)^2\right] = \left(\mathbb{E}\left[f_{\mathcal{D}}(\mathbf{x})\right] - f^*(\mathbf{x})\right)^2 + \mathrm{Var}\left[f_{\mathcal{D}}(\mathbf{x})\right]$$

- What is the bias-variance definition across all x?

Exercise: bias-variance across x

- For a specific x, we have

$$\mathbb{E}\left[(f_{\mathcal{D}}(\mathbf{x}) - f^*(\mathbf{x}))^2\right] = (\mathbb{E}\left[f_{\mathcal{D}}(\mathbf{x})\right] - f^*(\mathbf{x}))^2 + \mathrm{Var}\left[f_{\mathcal{D}}(\mathbf{x})\right]$$

- What is the bias-variance definition across all x?
  - Take expectation across the possible, from the data distribution
  - $p(\mathbf{x}, y) = p(y \,|\, \mathbf{x})p(\mathbf{x})$ so sample $\mathbf{x}$ from $p(\mathbf{x})$ for this expectation

$$\mathbb{E}[(f_{\mathcal{D}}(\boldsymbol{X}) - f^*(\boldsymbol{X}))^2] = \mathbb{E}_{\boldsymbol{X}}\left[(\mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}}(\boldsymbol{X})] - f^*(\boldsymbol{X}))^2 + \mathrm{Var}_{\mathcal{D}}[f_{\mathcal{D}}(\boldsymbol{X})]\right]$$

Exercise

- Imagine we are in the realizable setting and the true function is linear. We know our MLE estimator (regular old linear regresion) is an unbiased estimator
- This means $\mathbb{E}[\mathbf{w}(\mathscr{D})] = \mathbf{w}^*$. Does this mean that $\mathbb{E}[f_{\mathbf{w}(\mathscr{D})}(\mathbf{x})] = f_{\mathbf{w}^*}(\mathbf{x})$?
  Recall that $f_{\mathbf{w}^*}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{w}^*$

Exercise

- Imagine we are in the realizable setting and the true function is linear. We know our MLE estimator (regular old linear regresion) is an unbiased estimator
- This means $\mathbb{E}[\mathbf{w}(\mathscr{D})] = \mathbf{w}^*$. Does this mean that $\mathbb{E}[f_{\mathbf{w}(\mathscr{D})}(\mathbf{x})] = f_{\mathbf{w}^*}(\mathbf{x})$? Recall that $f_{\mathbf{w}^*}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{w}^*$
- Ans: Yes. Notice $\mathbb{E}[f_{\mathbf{w}(\mathscr{D})}(\mathbf{x})] = \mathbb{E}[\boldsymbol{\phi}(\mathbf{x})\mathbf{w}(\mathscr{D})] = \boldsymbol{\phi}(\mathbf{x})\mathbb{E}[\mathbf{w}(\mathscr{D})] = \boldsymbol{\phi}(\mathbf{x})\mathbf{w}^*$

Discussion

- Why did we talk about bias-variance by comparing the weights, when we could have just always talked about the bias-variance for the functions?
- Why do we talk about bias-variance at all? Ultimately, we just care about how good our model is?
- Any other questions about GE and bias-variance?

Ch. 5: Optimization

- Understanding why the Hessian in the second-order update accounts for differences in curvature in different dimensions
- The mini-batch stochastic gradient descent (SGD) update rule
- Understanding why SGD is a more computationally efficient update than GD
- Understanding the importance of an adaptive vector stepsize, and that RMSProp and Adam use vector stepsizes
- Understanding the momentum update, used in Adam
- **Will not directly test**
  - Computing any weights, Hessians, etc., convergence theory and directional derivatives, knowing any of specific formulas (e.g, Adam, RMSProp, etc)

Some optimization questions

- We use $c(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} c_i(\mathbf{w})$. But when we did LR we used $c(\mathbf{w}) = \frac{1}{2}\|\mathbf{\Phi w} - \mathbf{y}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$. Is this mismatch a problem?
- How do we write the Ridge Regression loss as $c(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} c_i(\mathbf{w})$?

A normalized RR objective

- $\frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{\phi}(\mathbf{x})_i\mathbf{w}-\mathbf{y}_i)^2+\frac{\lambda}{2n}\|\mathbf{w}\|_2^2$. What is ci?
- $c_i(\mathbf{w})=\frac{1}{2}(\boldsymbol{\phi}_i\mathbf{w}-y_i)^2+\frac{\lambda}{2n}\|\mathbf{w}\|_2^2$ because
- $\frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}((\boldsymbol{\phi}(\mathbf{x})_i\mathbf{w}-\mathbf{y}_i)^2+\frac{\lambda}{2n}\|\mathbf{w}\|_2^2)=\frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{\phi}(\mathbf{x})_i\mathbf{w}-\mathbf{y}_i)^2+\frac{1}{n}\sum_{i=1}^{n}\frac{\lambda}{2n}\|\mathbf{w}\|_2^2$ and $\frac{1}{n}\sum_{i=1}^{n}\frac{\lambda}{2n}\|\mathbf{w}\|_2^2=\frac{1}{n}n\frac{\lambda}{2n}\|\mathbf{w}\|_2^2=\frac{\lambda}{2n}\|\mathbf{w}\|_2^2$
- Makes very clear that regularizer has a diminishing role with increasing n

A normalized RR objective

- $\frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{\phi}(\mathbf{x})_i\mathbf{w}-\mathbf{y}_i)^2+\frac{\lambda}{2n}\|\mathbf{w}\|_2^2$. What is ci?
- Therefore must have $c_i(\mathbf{w})=\frac{1}{2}(\boldsymbol{\phi}_i\mathbf{w}-y_i)^2+\frac{\lambda}{2n}\|\mathbf{w}\|_2^2$
- Makes very clear that regularizer has a diminishing role with increasing n
- **Question**: What is the mini-batch SGD update for RR?

Mini-batch SGD for RR

- Therefore must have $c_i(\mathbf{w}) = \frac{1}{2}(\boldsymbol{\phi}_i \mathbf{w} - y_i)^2 + \frac{\lambda}{2n}\|\mathbf{w}\|_2^2$
- **Question**: What is the mini-batch SGD update for RR?
- $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \boldsymbol{\eta}_t \cdot \frac{1}{b} \sum_{i \in \mathcal{B}} \nabla c_i(\mathbf{w}_t)$
- where $\nabla c_i(\mathbf{w}) = (\boldsymbol{\phi}^\top \mathbf{w} - y_i)\mathbf{x}_i + \frac{\lambda}{n}\mathbf{w}$

Why do we normalize the mini-batch by 1/b? We could absorb into stepsize? Any advantages to using 1/b to get the averaged stochastic gradient?

Exercise Question

- How might the size of the dataset n interact with the number of epochs that we need to converge?

Exercise Question

- How might the size of the dataset n interact with the number of epochs that we need to converge?
- **Answer**: With a very large dataset, we are doing more updates in each epoch and likely need fewer epochs to converge.

Chapter 6: GLMs

- Understand that Generalized Linear Models (GLMs) allow us to model
  - $p(y \mid \mathbf{x})$ = any natural exponential family distribution with natural parameter $\boldsymbol{\phi}(\mathbf{x})\mathbf{w}$
  - with associated transfer function g such that $g(\boldsymbol{\phi}(\mathbf{x})\mathbf{w})$ approximates $\mathbb{E}[Y \mid \mathbf{x}]$
- Know that linear regression, Poisson regression, logistic regression and multinomial logistic regression are examples of GLMs and their corresponding transfers
- Know that the GLM objectives are convex
- **Will not be directly tested:**
  - Knowing specific GLM updates; if I need you to reason about one I will give it to you
  - The details of exponential family distributions (5.2)

Exercise Question

- Imagine you have multinomial logistic regression implemented. How would you use this code to do binary classification?

Exercise Question

- Imagine you have multinomial logistic regression implemented. How would you use this code to do binary classification?
- Transform dataset of (x,y) with y in {0,1} or y in {-1,1} to dataset with y in {[1,0], [0,1]}, then call multinonimal logistic regression on this

Exercise Question

- Imagine we want to predict targets $y \in \{0,1,2,3,\ldots,20\}$
- What GLM could we use?

Exercise Question

- Imagine we want to predict targets $y \in \{0,1,2,3,\ldots,20\}$
- What GLM could we use?
  - The targets are ordered, so using Poisson regression would be sensible. When actually deploying, if the model predicted > 20, then would return 20
  - Could also use multinomial logistic regression, but this will just discard the potentially useful ordering to the data. (Why does this matter?)

Exercise Question

- Imagine we use a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ using RBF features
- We run SGD with large minibatches for millions of epochs on the Poisson regression loss, until convergence (very small gradients)
- Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?

Exercise Question

- Imagine we use a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ using RBF features
- We run SGD with large minibatches for millions of epochs on the Poisson regression loss, until convergence (very small gradients)
- Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?
- What if instead we learned a nonlinear transformation on $\mathbf{x}$ using a neural network, again with the Poisson regression loss for millions of epochs?

Exercise Question

- Imagine we use a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ using RBF features
- Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?
  - **Ans**: Our Poisson regression objective is convex, for a fixed representation, even though the prediction itself is nonlinear in $\mathbf{x}$ due to using $\boldsymbol{\phi}(\mathbf{x})$
- What if instead we learned a nonlinear transformation on $\mathbf{x}$ using a neural network, again with the Poisson regression loss for millions of epochs?
  - **Ans**: The objective is no longer convex, cannot guarantee global min

Exercise Question

- What is instead we 1) initialize weights randomly in the NN and 2) freeze all the weights in the layers except the output layer, $\mathbf{W}^{(1)}$
- For a given input $\mathbf{x}$, we pass through multiple fixed layers to get to the final layer $\mathbf{h}^{(1)}$ and make prediction $\exp(\mathbf{h}^{(1)}\mathbf{W}^{(1)})$
- Imagine we train $\mathbf{W}^{(1)}$ for millions of epochs with the Poisson regression loss. Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?

Exercise Question

- What is instead we 1) initialize weights randomly in the NN and 2) freeze all the weights in the layers except the output layer, $\mathbf{W}^{(1)}$
- For a given input $\mathbf{x}$, we pass through multiple fixed layers to get to the final layer $\mathbf{h}^{(1)}$ and make prediction $\exp(\mathbf{h}^{(1)}\mathbf{W}^{(1)})$
- Imagine we train $\mathbf{W}^{(1)}$ for millions of epochs with the Poisson regression loss. Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?
  - **Ans**: Yes. The network just provides another fixed representation $\boldsymbol{\phi}(\mathbf{x}) \doteq \mathbf{h}^{(1)}$, (loss if convex for that last layer)

Exercise Question

- Imagine we use a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ using RBF features
- We run SGD with large minibatches for millions of epochs on the Poisson regression loss, **but now we also add l2 regularization.**
- Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?

Exercise Question

- Imagine we use a nonlinear transformation $\boldsymbol{\phi}(\mathbf{x})$ using RBF features
- We run SGD with large minibatches for millions of epochs on the Poisson regression loss, but now we also add l2 regularization.
- Can we guarantee we are at a global minima, or could we be stuck in a local minima or saddlepoint?
  - **Ans**: Yes. The l2 regularizer is convex in the weights, and the sum of two convex functions is again convex

Chapter 7: Constrained Optimization

- Optimization of the form $\min_{\mathbf{w} \in \mathbb{R}^p} c(\mathbf{w}) + r(\mathbf{w})$ for smooth c, nonsmooth r
- Understand need to use a different approach due to nonsmooth r, including for constrained optimization
- Proximal update: $\mathbf{w}_{t+1} = \text{prox}_{\eta_t r}(\mathbf{w}_t - \eta_t \nabla c(\mathbf{w}_t))$
- That we use proximal GD with l1 regularization, and that l1 regularization on the weights encourages weights to be zero
- **Will not test**
  - Specific proximal operators
  - Vector stepsizes or momentum with proximal operators; we only did scalar stepsizes
  - I will not get you to derive solutions with Lagrangians (Section 7.3)

Proximal operators

- Optimize with smooth $c(\mathbf{w})$ and non-smooth $r(\mathbf{w})$
- To solve $\min_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) + r(\mathbf{w})$ we break it into two steps
- GD Step: $\tilde{\mathbf{w}}_{t+1} = \mathbf{w}_t - \eta \nabla c(\mathbf{w}_t)$
- Projection step: $\mathbf{w}_{t+1} = \text{prox}_{\eta r}(\tilde{\mathbf{w}}_{t+1})$ where the proximal operator is
- $\text{prox}_{\eta r}(\mathbf{u}) = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w} - \mathbf{u}\|_2^2 + \eta r(\mathbf{w})$

Simple example of a proximal operator

- You solved for $\text{prox}_{\eta r}(\mathbf{u}) = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}-\mathbf{u}\|_2^2 + \eta r(\mathbf{w})$ for $r(\mathbf{w}) = \lambda\|\mathbf{w}\|_1$
- Let's do another simple example with $r(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2$
- Not necessary to use proximal gradient descent for this regularizer, because it is smooth, so it can just be included in c. But, we can if we want
- **Note**: we cannot do the other way. We cannot take nonsmooth parts and just include them in c, because we must be able to take the gradient of c for the first part of the proximal update

Proximal operator for l2-regularization

- $\text{prox}_{\eta_t \frac{\lambda}{2} \ell_2}(\tilde{\mathbf{w}}_{t+1}) = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w} - \tilde{\mathbf{w}}_{t+1}\|_2^2 + \eta_t \lambda \frac{1}{2}\|\mathbf{w}\|_2^2$
- Closed-form solution is given by
- $\nabla \left(\frac{1}{2}\|\mathbf{w} - \tilde{\mathbf{w}}_{t+1}\|_2^2 + \eta_t \lambda \frac{1}{2}\|\mathbf{w}\|_2^2 \right) = \mathbf{w} - \tilde{\mathbf{w}}_{t+1} + \eta_t \lambda \mathbf{w} = 0$
- Implies $\mathbf{w} = (1 + \eta_t \lambda)^{-1} \tilde{\mathbf{w}}_{t+1} = (1 + \eta_t \lambda)^{-1} (\mathbf{w}_t - \eta_t \nabla c(\mathbf{w}_t))$
- Self-test: What exactly is $\text{prox}_{\eta_t \frac{\lambda}{2} \ell_2}(\tilde{\mathbf{w}}_{t+1})$ here?

Chapter 8: Cross Validation

- Understand that cross validation allows us to estimate GE of a model trained on the entire dataset, without having to have a hold-out test set
- Understand the k-fold CV algorithm
- Understand the repeated random subsampling (RRS) CV algorithm
- General role k and p play in the bias of our GE estimator (outlined in these slides)
- **Will not test**
  - The more detailed nuances about the bias-variance distinctions for different CV choices

Chapter 8: Cross Validation

- Know what it means to select hyperparameters
- Understand the utility of CV for hyperparameter selection
- Understand the difference between internal CV and external CV in nested CV
  - internal CV is for hyperparameter selection and external is to evaluate the algorithm that might use internal CV
- **Will not be directly tested:**
  - Knowing how to pick the set of hyperparameters to be tested with CV

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
  - We have a massive dataset, so can make train, validation and test big
  - And sometimes compromise if our model is very expensive to train, making CV impractical to use

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren’t there also three datasets (train, validation and test)?

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren’t there also three datasets (train, validation and test)?
  - Validation is used for hyperparameter selection (role of internal CV)
  - Test is used for evaluation of the final function (role of external CV)
  - CV uses the whole dataset for both evaluation and training, so it actually only makes splits for evaluation, and trains on the entire dataset

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren’t there also three datasets (train, validation and test)?
- Do we have to use nested cross validation? Can we use a single train-test split externally, and k-fold CV internally?

.

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren't there also three datasets (train, validation and test)?
- Do we have to use nested cross validation? Can we use a single train-test split externally, and k-fold CV internally?
  - Yes! The learner uses CV to algorithmically set its hyperparameters. We can evaluate this learner in any way we want (external CV or one train-test split)

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren’t there also three datasets (train, validation and test)?
- Do we have to use nested cross validation? Can we use a single train-test split externally, and k-fold CV internally?
- Can we use a single train-validation split internally, and CV externally?

Exercise Question

Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren't there also three datasets (train, validation and test)?
- Do we have to use nested cross validation? Can we use a single train-test split externally, and k-fold CV internally?
- Can we use a single train-validation split internally, and CV externally?
  - Yes! The learner uses a validation set to algorithmically set its hyperparameters. We can evaluate this learner in any way we want.

Exercise Question

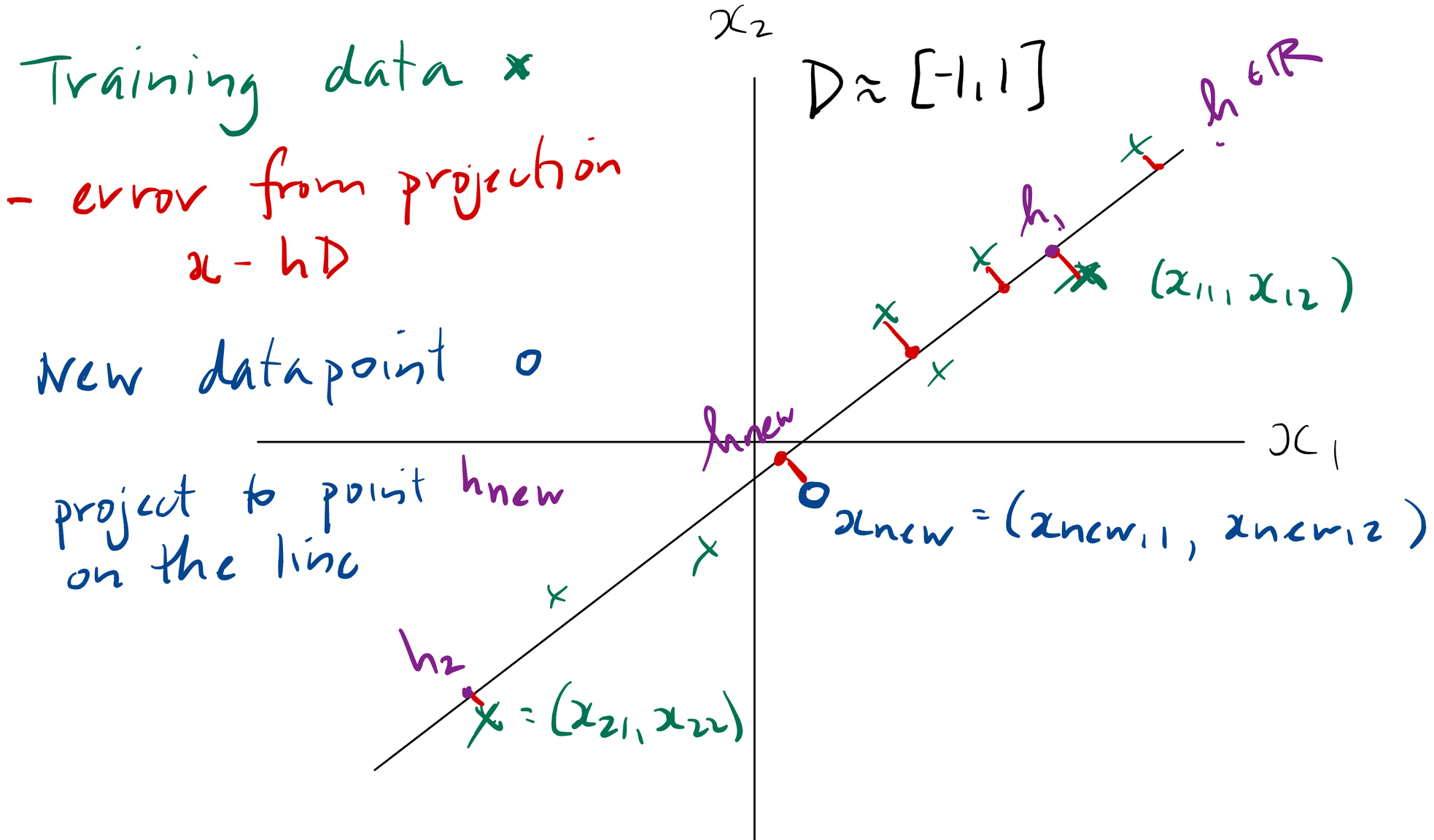
Do we have to use nested cross-validation?

- When should we use a single train-validation-test split?
- Why does CV only have train and test partitions? Why aren’t there also three datasets (train, validation and test)?
- Do we have to use nested cross validation? Can we use a single train-test split externally, and k-fold CV internally?
- Can we use a single train-validation split internally, and CV externally?
- Should we do either of these?

Chapter 9: Latent Factors

- Understand that PCA extracts a lower-dimensional representation
- Understand the objective underlying PCA (minimize $\|\mathbf{x} - \mathbf{hD}\|_2^2$ for every x)
- Understand that sparse coding similarly minimizes $\|\mathbf{x} - \mathbf{hD}\|_2^2$, but additionally has an l1 regularizer on h to find a high-dimensional sparse representation
- **Will not be directly tested:**
  - PPCA
  - Algorithms for PCA and sparse coding
  - Interpretations of latent factors (was only for intuition about what might be learned)

PCA on training data and new points



Exercise Question

- Imagine we first expand the dimension using a kernel representation, going from 10 features to 5000.
- Imagine we compute the SVD of Phi, and the top 100 singular values are reasonably big, and the rest are relatively small.
- We decide to use PCA to get 100 features (i.e., use the computed SVD and take the top k singular vectors as the new representation).
- How do we interpret these 100 features?

Chapter 10: Neural Networks

- Understand transformation on the input given by a simple feedforward neural network
  - series of linear functions composed with simple activations
- Understand that backpropagation is gradient descent
- Understand can use different architectures: skip connections, convolutions, sparse connectivity
- **Will not test**
  - You will not need to derive gradients for an NN, nor know the backprop update formulas

Exercise Question

- We discussed that many transformations consist of (1) linear weighting followed by (2) nonlinear activation (differentiable almost everywhere)
- What are some other activations we could consider using in a network, beyond the three we discussed (ReLU, sigmoid, tanh)? Why would these be reasonable?

Exercise Question

Assume we are predicting a scalar target

- Write down the set of functions $\mathscr{F}_1$ obtained using a kernel representation with kernel k, and a random subset of 100 points from the training data as centers, giving features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^{101}$ (assume $\mathscr{X}$ space of possible inputs $\mathbf{x}$)

- Write down the set of functions $\mathscr{F}_2$ obtained using an NN with one hidden layers of size 256, with ReLu activations, for regression

Exercise Question

Assume we are predicting a scalar target

- Write down the set of functions $\mathcal{F}_1$ obtained using a kernel representation with kernel k, and a random subset of 100 points from the training data as centers, giving features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^{101}$ (assume $\mathcal{X}$ space of possible inputs $\mathbf{x}$)
  - Ans: $\mathcal{F}_1 = \{f : \mathcal{X} \to \mathbb{R} : f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})\mathbf{w}$ for any $\mathbf{w} \in \mathbb{R}^{101}\}$
- Write down the set of functions $\mathcal{F}_2$ obtained using an NN with one hidden layers of size 256, with ReLu activations, for regression
  - Ans:
    $\mathcal{F}_2 = \{f : \mathcal{X} \to \mathbb{R} : f(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}^{(2)})\mathbf{W}^{(1)}$ for any $\mathbf{W}^{(1)} \in \mathbb{R}^{256\times 1}, \mathbf{W}^{(2)} \in \mathbb{R}^{d\times 256}\}$

Exercise Question

- Write down the set of functions $\mathscr{F}_1$ obtained using a kernel representation with kernel k, and a random subset of 100 points from the training data as centers, giving features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^{101}$ (assume $\mathscr{X}$ space of possible inputs $\mathbf{x}$)
- Write down the set of functions $\mathscr{F}_2$ obtained using an NN with one hidden layers of size 256, with ReLu activations, for regression
- If I told you that an $\mathscr{F}_1$ is a subset of $\mathscr{F}_2$, what does that mean? Which class has higher complexity (or capacity)?
- How do you know one is a subset of the other? Is $\mathscr{F}_1$ a subset of $\mathscr{F}_2$ here?

Exercise Question

- Recall bias-variance decomposition

$$\mathbb{E}\left[(f_{\mathcal{D}}(\mathbf{x}) - f^*(\mathbf{x}))^2\right] = (\mathbb{E}\left[f_{\mathcal{D}}(\mathbf{x})\right] - f^*(\mathbf{x}))^2 + \mathrm{Var}\left[f_{\mathcal{D}}(\mathbf{x})\right]$$

- Imagine $\mathscr{F}_1$ is a subset of $\mathscr{F}_2$
- Do you think $\mathscr{F}_1$ or $\mathscr{F}_2$ has higher bias?
- Do you think $\mathscr{F}_1$ or $\mathscr{F}_2$ has higher variance?

Exercise Question

- Recall bias-variance decomposition

$$\mathbb{E}\left[\left(f_{\mathcal{D}}(\mathbf{x}) - f^*(\mathbf{x})\right)^2\right] = \left(\mathbb{E}\left[f_{\mathcal{D}}(\mathbf{x})\right] - f^*(\mathbf{x})\right)^2 + \mathrm{Var}\left[f_{\mathcal{D}}(\mathbf{x})\right]$$

- Imagine $\mathscr{F}_1$ is a subset of $\mathscr{F}_2$
- Do you think $\mathscr{F}_1$ or $\mathscr{F}_2$ has higher bias? **Ans**: Yes, equal or higher bias
- Do you think $\mathscr{F}_1$ or $\mathscr{F}_2$ has higher variance? **Ans**: No, equal or lower variance

Chapter 11: Regularization for NNs

- Understand that linear autoencoders also extract a low-dimensional representation like PCA
- Understand we can use the reconstruction loss as an auxiliary loss
- **Will not be directly tested:**
  - Good to know about the ability to use auxiliary losses, but will not be directly tested on supervised autoencoders
  - Will not be tested on Section 11.3 (double descent)

Exercise

- We saw a linear autoencoder is equivalent to PCA
  - $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{(2)}\mathbf{W}^{(1)}$ with bottleneck ($\mathbf{W}^{(2)} \in \mathbb{R}^{d \times k}, \mathbf{W}^{(1)} \in \mathbb{R}^{k \times d}, k < d$
- What if we use a multilayer autoencoder with nonlinear activations?
  - $f_{\mathbf{w}}(\mathbf{x}) = f_1(f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})\mathbf{W}^{(1)})$
- What is the size of the output, $f_{\mathbf{w}}(\mathbf{x})$?
- What loss do we use to train this multilayer autoencoder?

Exercise

- We saw a linear autoencoder is equivalent to PCA
- What if we use a multilayer autoencoder with nonlinear activations
  - $f_{\mathbf{w}}(\mathbf{x}) = f_1(f_2(f_3(\mathbf{x}\mathbf{W}^{(3)})\mathbf{W}^{(2)})\mathbf{W}^{(1)})$
- What is the size of the output, $f_{\mathbf{w}}(\mathbf{x})$?
  - **Ans**: Same size as $\mathbf{x}$, because we are trying to reconstruct $\mathbf{x}$
- What loss do we use to train this multilayer autoencoder?
  - **Ans**: Reconstruction error to $\mathbf{x}$ (e.g., squared-error loss $\|\mathbf{x} - \hat{\mathbf{x}}\|_2^2$)

Chapter 12: Mixture Models

- Understand how we sample from mixture models
- Understand that the EM algorithm consists of (a) the introduction of auxiliary variables z (which component of the mixture a sample belongs to) and (b) alternating between updating $p(z_i \mid \mathbf{x}_i)$ and parameters $\mathbf{w}$
- Understand that the E-step updates $p(z_i \mid \mathbf{x}_i)$ for fixed $\mathbf{w}$, and the M-step updates $\mathbf{w}$ for fixed $p(z_i \mid \mathbf{x}_i)$ with each component updated independently using a (weighted) log-likelihood
- **Will not be directly tested:**
  - You do not need to memorize the EM algorithm, but you should be able to recognize key components of it

EM Algorithm for any component distribution

$\mathbf{w} = (\boldsymbol{\theta}, \boldsymbol{\alpha})$

Algorithm 10: EM for any component distribution

Input: number of components m, with components distributions $p(\cdot|\boldsymbol{\theta}_1), \ldots, p(\cdot|\boldsymbol{\theta}_m)$

Initialize $\theta_k^{(0)}$ and $\alpha_k^{(0)}$ for all $k \in 1$ to m, $t = 0$

while not converged **do**

  $p_t[i,k] = \frac{\alpha_k^{(t)} p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t)})}{\sum_{j=1}^m \alpha_j^{(t)} p(\mathbf{x}_i|\boldsymbol{\theta}_j^{(t)})}$ for all $i \in \{1,2,\ldots,n\}$, $k \in \{1,2,\ldots,m\}$

  Compute $p_t[k] \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n p_t[i,k]$

  for $k \in \{1,2,\ldots,m\}$ **do**

    $\alpha_k^{(t+1)} = p_t[k]$

    $\boldsymbol{\theta}_k^{(t+1)} = \text{argmin}_{\boldsymbol{\theta}_k} - \sum_{i=1}^n p_t[i,k] \ln p(\mathbf{x}_i|\boldsymbol{\theta}_k)$

  $t \leftarrow t+1$

return $\alpha_k^t, \boldsymbol{\theta}_k^{(t)}$ for all $k \in \{1,2,\ldots,m\}$

Which part is the E step (update $p(z_i \mid \mathbf{x}_i)$) and which is the M step (update $\mathbf{w}$)?

EM Algorithm for any component distribution

$\mathbf{w} = (\boldsymbol{\theta}, \boldsymbol{\alpha})$

Algorithm 10: EM for any component distribution

Input: number of components m, with components distributions $p(\cdot|\boldsymbol{\theta}_1), \ldots, p(\cdot|\boldsymbol{\theta}_m)$

Initialize $\theta_k^{(0)}$ and $\alpha_k^{(0)}$ for all $k \in 1$ to m, $t = 0$

while not converged **do**

E-step

$$p_t[i,k] = \frac{\alpha_k^{(t)} p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t)})}{\sum_{j=1}^{m} \alpha_j^{(t)} p(\mathbf{x}_i|\boldsymbol{\theta}_j^{(t)})} \text{ for all } i \in \{1,2,\ldots,n\},\ k \in \{1,2,\ldots,m\}$$

Compute $p_t[k] \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} p_t[i,k]$

for $k \in \{1,2,\ldots,m\}$ **do**

M-step

$$\alpha_k^{(t+1)} = p_t[k]$$

$$\boldsymbol{\theta}_k^{(t+1)} = \text{argmin}_{\boldsymbol{\theta}_k} - \sum_{i=1}^{n} p_t[i,k] \ln p(\mathbf{x}_i|\boldsymbol{\theta}_k)$$

$t \leftarrow t + 1$

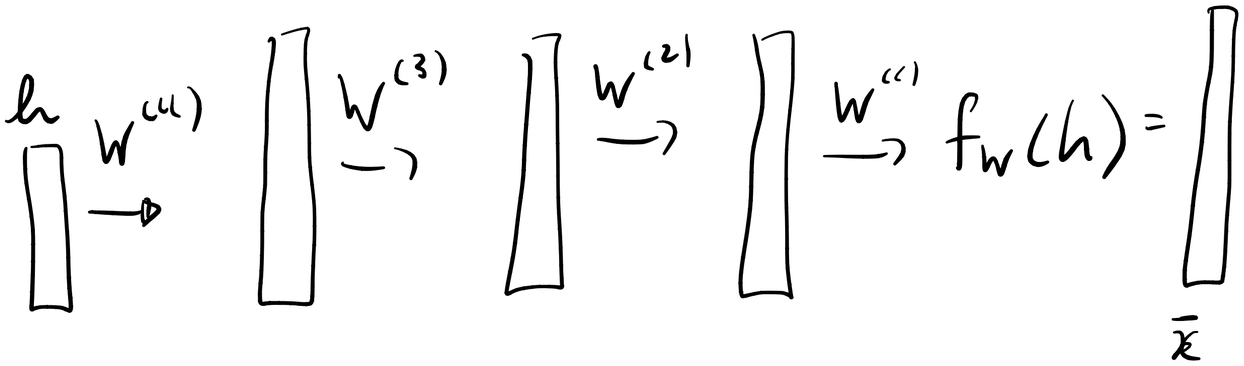
return $\alpha_k^t, \boldsymbol{\theta}_k^{(t)}$ for all $k \in \{1,2,\ldots,m\}$

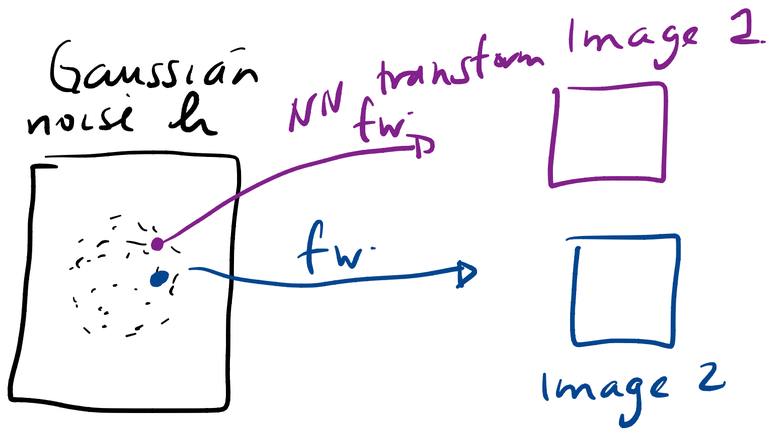
Which part is the E step (update $p(z_i \mid \mathbf{x}_i)$) and which is the M step (update $\mathbf{w}$)?

Chapter 13: Generative Models using Data Reps

- Understand that both PPCA and VAEs make the assumption that
- $p(\mathbf{x}) = \int p(\mathbf{x}\,|\,\mathbf{h})p(\mathbf{h})d\mathbf{h}$ with $p(\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Understand that PPCA assumes a linear relationship between $\mathbf{x}$ and $\mathbf{h}$
  $p(\mathbf{x}\,|\,\mathbf{h}) = \mathcal{N}(\mathbf{hD}, \sigma^2\mathbf{I})$
- And that VAE generalizes to a nonlinear relationship, using NN $f_{\mathbf{W}}$ to give
  $p(\mathbf{x}\,|\,\mathbf{h}) = \mathcal{N}(f_{\mathbf{W}}(\mathbf{h}), \sigma^2\mathbf{I})$
- Understand how to sample from a VAE
  - Step 1: Sample $\mathbf{h} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then
  - Step 2: Return $f_{\mathbf{W}}(\mathbf{h})$ (where $f_{\mathbf{W}}$ is the decoder part of the network)

Visualizing this function





- VAE generalizes to a nonlinear relationship, using NN $f_{\mathbf{W}}$ to give $p(\mathbf{x} \mid \mathbf{h}) = \mathcal{N}(f_{\mathbf{W}}(\mathbf{h}), \sigma^2 \mathbf{I})$
- Understand how to sample from a VAE
  - Step 1: Sample $\mathbf{h} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then
  - Step 2: Return $f_{\mathbf{W}}(\mathbf{h})$ or sample $\mathbf{x} \sim \mathcal{N}(f_{\mathbf{W}}(\mathbf{h}), \sigma^2 \mathbf{I})$

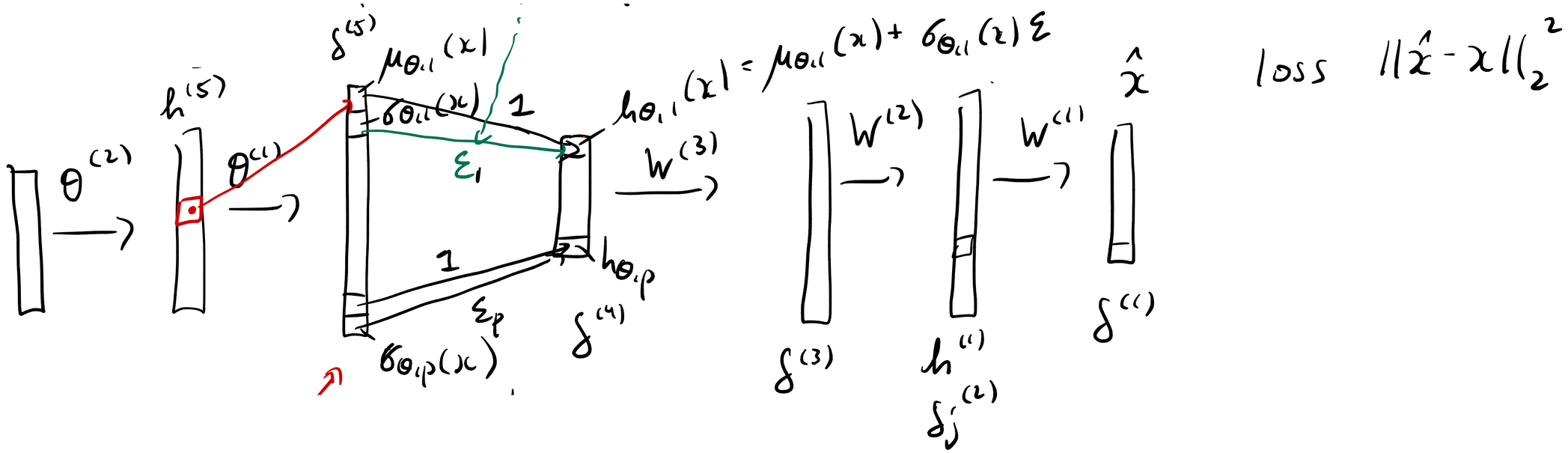
Chapter 12: VAEs (cont)

- Understand our goal is to minimize $\sum_{i=1}^{n} -\ln p(\mathbf{x}_i \,|\, \mathbf{W})$
  - But that this is hard to because $p(\mathbf{x}_i \,|\, \mathbf{W})$ involves an integral over hidden $\mathbf{h}$
- Understand that we learn the encoder $q(\mathbf{h} \,|\, \mathbf{x})$ only as part of the optimization, to help us learn $p(\mathbf{x} \,|\, \mathbf{h})$; we do not need $q(\mathbf{h} \,|\, \mathbf{x})$ itself for deployment
- Understand that we derive the VAE objective by starting with $-\ln p(\mathbf{x} \,|\, \mathbf{W}) + D_{KL}(q_\theta(\,\cdot\,|\,\mathbf{x}) \,||\, p(\,\cdot\,|\,\mathbf{x}, \mathbf{W}))$ and rearranging terms to get $D_{KL}(q_\theta(\,\cdot\,|\,\mathbf{x}) \,||\, \mathcal{N}(0, \mathbf{I})) - \mathbb{E}_{\mathbf{h} \sim q_\theta(\cdot|\mathbf{x})}[\ln p(\mathbf{x} \,|\, \mathbf{h}, \mathbf{W})]$ (the negative ELBO)

Discussion questions

- Why is our minimization $\min_{\mathbf{W},\boldsymbol{\theta}} -\ln p(\mathbf{x}\,|\,\mathbf{W}) + D_{KL}(q_\theta(\,\cdot\,|\,\mathbf{x})\,||\,p(\,\cdot\,|\,\mathbf{x},\mathbf{W}))$?
- (And why do we write $q_\theta(\,\cdot\,|\,\mathbf{x})$ instead of $q(\,\cdot\,|\,\mathbf{x},\theta)$?)
- Why is it equivalent to instead minimize the negative ELBO?
  $\min_{\mathbf{W},\boldsymbol{\theta}} D_{KL}(q(\,\cdot\,|\,\mathbf{x})\,||\,\mathcal{N}(0,\mathbf{I})) - \mathbb{E}_{\mathbf{h}\sim q(\cdot|\mathbf{x})}[\ln p(\mathbf{x}\,|\,\mathbf{h},\mathbf{W})]$
- How does this look if I maximize the ELBO?
- If we change $p(\mathbf{x}\,|\,\mathbf{h}) = \mathcal{N}(f_{\mathbf{W}}(\mathbf{h}), \sigma^2\mathbf{I})$ to $p(\mathbf{x}\,|\,\mathbf{h}) = \text{Bernoulli}(f_{\mathbf{W}}(\mathbf{h}))$, then do we also have to change the distribution $p(\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$?

Exercise: which part is q and p?



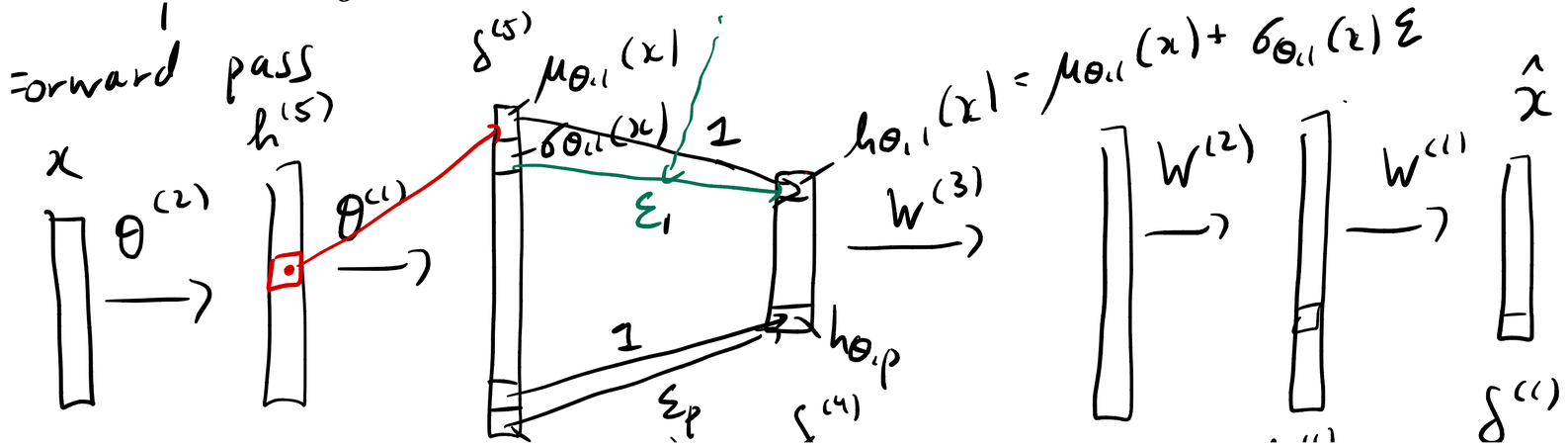
- Recall we parameterized our encoder as $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(f_{\mu,\theta}(\mathbf{x}), f_{\sigma,\theta}(\mathbf{x}))$

Chapter 12: VAEs (cont)

- Understand our goal is to minimize $\sum_{i=1}^{n} -\ln p(\mathbf{x}_i \,|\, \mathbf{W})$
  - But that this is hard to because $p(\mathbf{x}_i \,|\, \mathbf{W})$ involves an integral over hidden $\mathbf{h}$
- Understand that we learn the encoder $q(\mathbf{h} \,|\, \mathbf{x})$ only as part of the optimization, to help us learn $p(\mathbf{x} \,|\, \mathbf{h})$; we do not need $q(\mathbf{h} \,|\, \mathbf{x})$ itself for deployment
- Understand that we derive the VAE objective by starting with $-\ln p(\mathbf{x} \,|\, \mathbf{W}) + D_{KL}(q_\theta(\,\cdot\,|\,\mathbf{x}) \,||\, p(\,\cdot\,|\,\mathbf{x}, \mathbf{W}))$ and rearranging terms to get $D_{KL}(q_\theta(\,\cdot\,|\,\mathbf{x}) \,||\, \mathcal{N}(0, \mathbf{I})) - \mathbb{E}_{\mathbf{h} \sim q_\theta(\cdot|\mathbf{x})}[\ln p(\mathbf{x} \,|\, \mathbf{h}, \mathbf{W})]$ (the negative ELBO)
- Understand why we use the reparameterization trick to get the gradient

Exercise: Reparameterization trick

- Recall we parameterized our encoder as $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(f_{\mu,\theta}(\mathbf{x}), f_{\sigma,\theta}(\mathbf{x}))$
- We used reparameterization
  $\mathbb{E}_{\mathbf{h}\sim q(\cdot|\mathbf{x})}[\ln p(\mathbf{x} \mid \mathbf{h}, \mathbf{W})] = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\mathbf{I})}[\ln p(\mathbf{x} \mid h_j = \mu_j(\mathbf{x}) + \sigma_j(\mathbf{x})\boldsymbol{\epsilon}_j, \mathbf{W})]$
- Q1: What is the dimension of $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$? How is this sampled?
- Q2: What if we decided we wanted discrete hidden h and used $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \text{softmax}(f_\theta(\mathbf{x}))$? How does the NN change? Can we still use reparam?



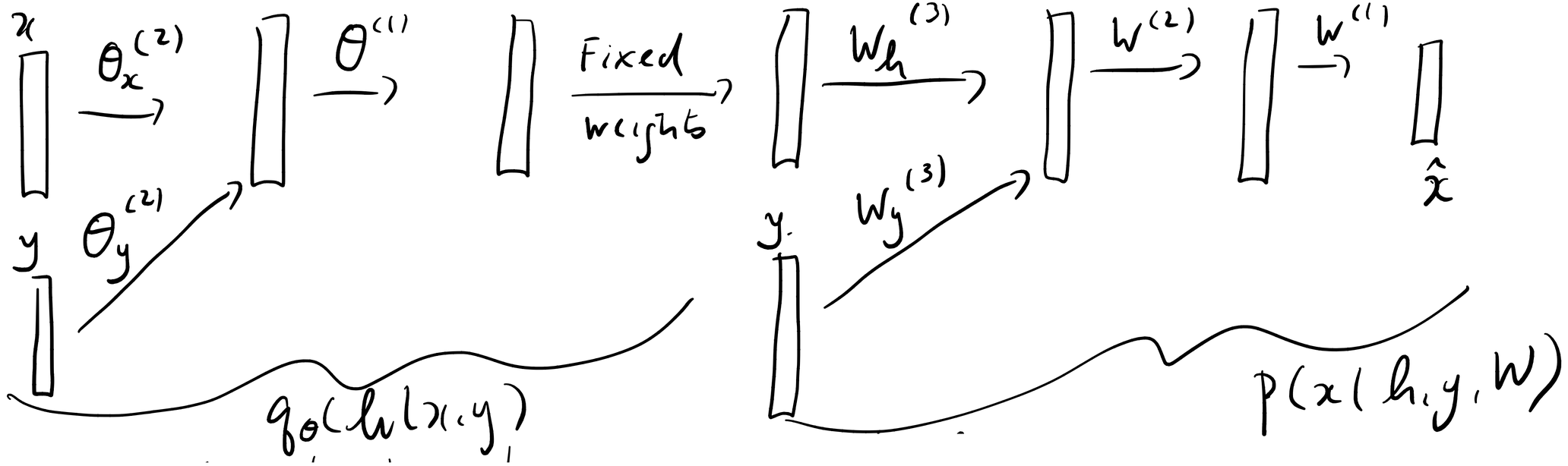
Exercise: Reparameterization trick

- Recall we parameterized our encoder as $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(f_{\mu,\theta}(\mathbf{x}), f_{\sigma,\theta}(\mathbf{x}))$
- We used reparameterization
  $\mathbb{E}_{\mathbf{h}\sim q(\cdot|\mathbf{x})}[\ln p(\mathbf{x} \mid \mathbf{h}, \mathbf{W})] = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,\mathbf{I})}[\ln p(\mathbf{x} \mid h_j = \mu_j(\mathbf{x}) + \sigma_j(\mathbf{x})\boldsymbol{\epsilon}_j, \mathbf{W})]$
- Q1: What is the dimension of $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$? How is this sampled?
  - **Ans**: size of h (size k), independently get k samples from a normal distribution $\mathcal{N}(0,1)$
- Q2: What if we decided we wanted discrete hidden h and used
  $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \text{softmax}(f_\theta(\mathbf{x}))$? How does the NN change? Can we still use reparam?
  - **Ans**: If the number of discrete items is m (h in {1, 2, …, m}), the hidden dimension is k*m sized, i.e., softmax outputs repeated k times (one for each h1, …, hk that will be sampled)
  - It is not longer straightforward to use reparameterization

Exercise

- To sample from a VAE, we use
  - Step 1: Sample $\mathbf{h} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and then
  - Step 2: query the decoder part of the VAE network $f_{\mathbf{W}}(\mathbf{h})$
- Why don't we sample $\mathbf{h}$ from $q(\mathbf{h} \mid \mathbf{x})$?

Chapter 13: Conditional VAEs



Chapter 13: VAEs (cont)

- **Will not be directly tested**
  - Memorizing the VAE objective (I will give you formulas)
  - Memorizing the gradient update for the VAE
  - The connection to Expectation-Maximization (9.3)
  - Deriving the gradient with the reparameterization trick

Chapter 13: Evaluating Generative Models

- Understand goal is to estimate $\hat{\text{GE}}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{x_i \in \mathcal{D}_{\text{test}}} -\ln p(\mathbf{x}_i \mid \boldsymbol{\theta})$
- Understand that can easily compute this for a mixture model, as long as easy to compute $-\ln p(\mathbf{x}_i \mid \boldsymbol{\theta}_k)$ for each component
- Understand that this is hard to compute for VAEs, must be estimated, and that we again leveraged $q_\theta(\mathbf{h} \mid \mathbf{x})$ to do so
- **Will not be directly tested**
  - the exact importance sampling algorithm to estimate this GE for VAEs

Exercise

- How would you use k-fold CV to pick the number of centers for a Gaussian Mixture Model (GMM)?

Exercise

- How would you use k-fold CV to pick the number of centers for a Gaussian Mixture Model (GMM)?
- Answer: You would decide on the set of numbers to select from,
  e.g., H = {2, 4, 8, 16}
- After partitioning the data into k folds, for each hyper m in H and each fold f
  - Learn the GMM $\hat{p}$ on all but fold f
  - Evaluate on fold f, by computing the negative log likelihood on the data
    $$\sum_{x \in \text{fold f}} -\ln \hat{p}(x)$$

Exercise

- Can we take a mixture over VAE components?

Exercise

- Can we take a mixture over VAE components?
  - Yes! Not sure its useful, but seems fun. Go back to the MM algorithms and see if you can figure out how to do it by

Algorithm 10: EM for any component distribution

Input: number of components m, with components distributions $p(\cdot|\boldsymbol{\theta}_1), \ldots, p(\cdot|\boldsymbol{\theta}_m)$
Initialize $\theta_k^{(0)}$ and $\alpha_k^{(0)}$ for all $k \in 1$ to m, $t = 0$
while not converged **do**
  $p_t[i,k] = \frac{\alpha_k^{(t)} p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t)})}{\sum_{j=1}^m \alpha_j^{(t)} p(\mathbf{x}_i|\boldsymbol{\theta}_j^{(t)})}$ for all $i \in \{1,2,\ldots,n\}$, $k \in \{1,2,\ldots,m\}$
  Compute $p_t[k] \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^n p_t[i,k]$
  for $k \in \{1,2,\ldots,m\}$ **do**
    $\alpha_k^{(t+1)} = p_t[k]$
    $\boldsymbol{\theta}_k^{(t+1)} = \text{argmin}_{\boldsymbol{\theta}_k} - \sum_{i=1}^n p_t[i,k] \ln p(\mathbf{x}_i|\boldsymbol{\theta}_k)$
  $t \leftarrow t+1$
return $\alpha_k^t, \boldsymbol{\theta}_k^{(t)}$ for all $k \in \{1,2,\ldots,m\}$

Exercise

- Can we take a mixture over VAE components?
  - Yes! Not sure its useful, but seems fun. Go back to the MM algorithms and see if you can figure out how to do it by
- Is this still easy to evaluate, in terms of negative log-likelihood?

Chapter 15: Missing Data

- Understand how to do imputation using PCA (matrix factorization), including training with missing data and using the model for new data
- Understand how to do imputation using an autoencoder, including training with missing data and using the model for new data
- Understand the difference between the two stage approach (impute then hand to learning algorithm) versus the direct approach (missingness indicator)
- **Will not be tested**
  - Connections to the transductive and semi-supervised settings

Notation check

- What does $\mathbf{x}_{\mathcal{A}_i}$ mean?

Notation check

- What does $\mathbf{x}_{\mathcal{A}_i}$ mean?
- The entries in $\mathbf{x}$ that correspond to the indices in $\mathcal{A}_i$
- We write $\mathbf{x}_{\mathcal{M}_i}$ to refer to the part of the array where values are missing. We use this to talk about directly optimizing for $\mathbf{x}_{\mathcal{M}_i}$

Exercise: PCA (matrix completion)

- In PCA we solved for $\min_{\mathbf{h}_1,\mathbf{h}_2,\ldots,\mathbf{h}_n \in \mathbb{R}^p, \mathbf{D} \in \mathbb{R}^{p \times d}} \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - \mathbf{h}_i \mathbf{D}_{:j})^2$
- In PCA with missing data, $\min_{\mathbf{h}_1,\mathbf{h}_2,\ldots,\mathbf{h}_n \in \mathbb{R}^p, \mathbf{D} \in \mathbb{R}^{p \times d}} \sum_{i=1}^{n} \sum_{j \in \mathcal{A}_i} (x_{ij} - \mathbf{h}_i \mathbf{D}_{:j})^2$
- Q: Why didn't we just set $\mathbf{x}_{\mathcal{M}_i} = \mathbf{0}$ (set unavailable values to zero) and call PCA? Will we get back the same h's and D as using the above optimization?

Exercise: PCA (matrix completion)

- In PCA we solve for $\min_{\mathbf{h}_1,\mathbf{h}_2,\ldots,\mathbf{h}_n\in\mathbb{R}^p,\mathbf{D}\in\mathbb{R}^{p\times d}} \sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij}-\mathbf{h}_i\mathbf{D}_{:j})^2$
- In PCA with missing data, $\min_{\mathbf{h}_1,\mathbf{h}_2,\ldots,\mathbf{h}_n\in\mathbb{R}^p,\mathbf{D}\in\mathbb{R}^{p\times d}} \sum_{i=1}^{n}\sum_{j\in\mathscr{A}_i}(x_{ij}-\mathbf{h}_i\mathbf{D}_{:j})^2$
- Q: Why didn't we just set $\mathbf{x}_{\mathscr{M}_i}=\mathbf{0}$ (set unavailable values to zero) and call PCA? Will we get back the same h's and D as using the above optimization?
  - No. But we do if we explicitly minimize over $\mathbf{x}_{\mathscr{M}_i}$

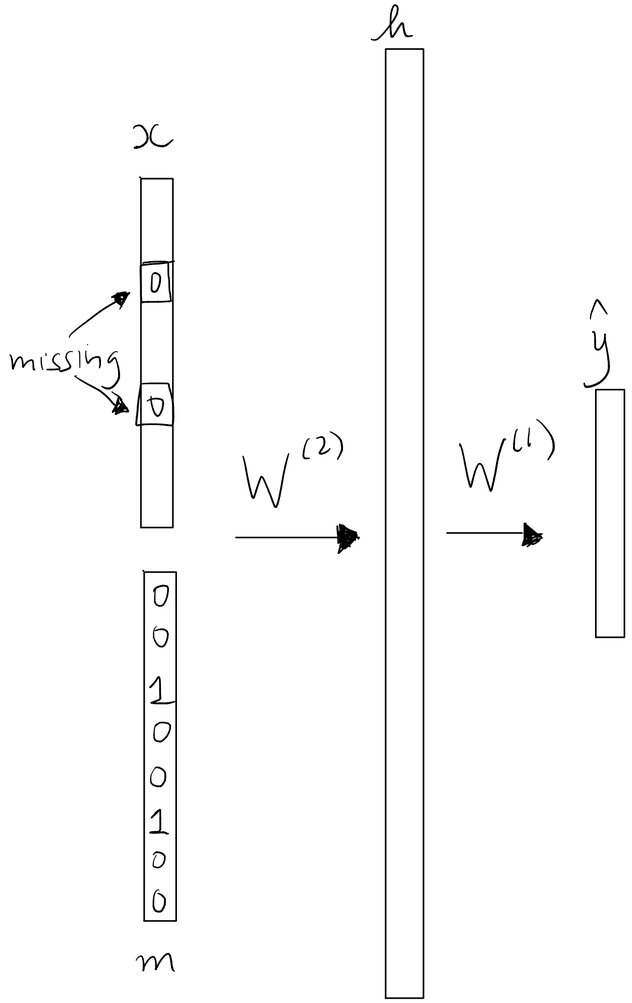
More general loss we use

- In PCA we solve for $\min_{\mathbf{x}_{1,\mathscr{M}_1},\ldots,\mathbf{x}_{n,\mathscr{M}_n}} \min_{\mathbf{h}_1,\mathbf{h}_2,\ldots,\mathbf{h}_n \in \mathbb{R}^p, \mathbf{D} \in \mathbb{R}^{p\times d}} \sum_{i=1}^{n}\sum_{j=1}^{d} (x_{ij} - \mathbf{h}_i \mathbf{D}_{:j})^2$
- For autoencoder, $\min_{\mathbf{x}_{1,\mathscr{M}_1},\ldots,\mathbf{x}_{n,\mathscr{M}_n}} \min_{\mathbf{W}} \sum_{i=1}^{n}\sum_{j=1}^{d} (x_{ij} - f_{\mathbf{W}}(\mathbf{x}_i))^2$
- We need this more general purpose loss for the autoencoder, because we cannot use the following objective. **Why**?

  $\min_{\mathbf{W}} \sum_{i=1}^{n}\sum_{j\in\mathscr{A}_i} (x_{ij} - f_{\mathbf{W}}(\mathbf{x}_i))^2$

Exercise

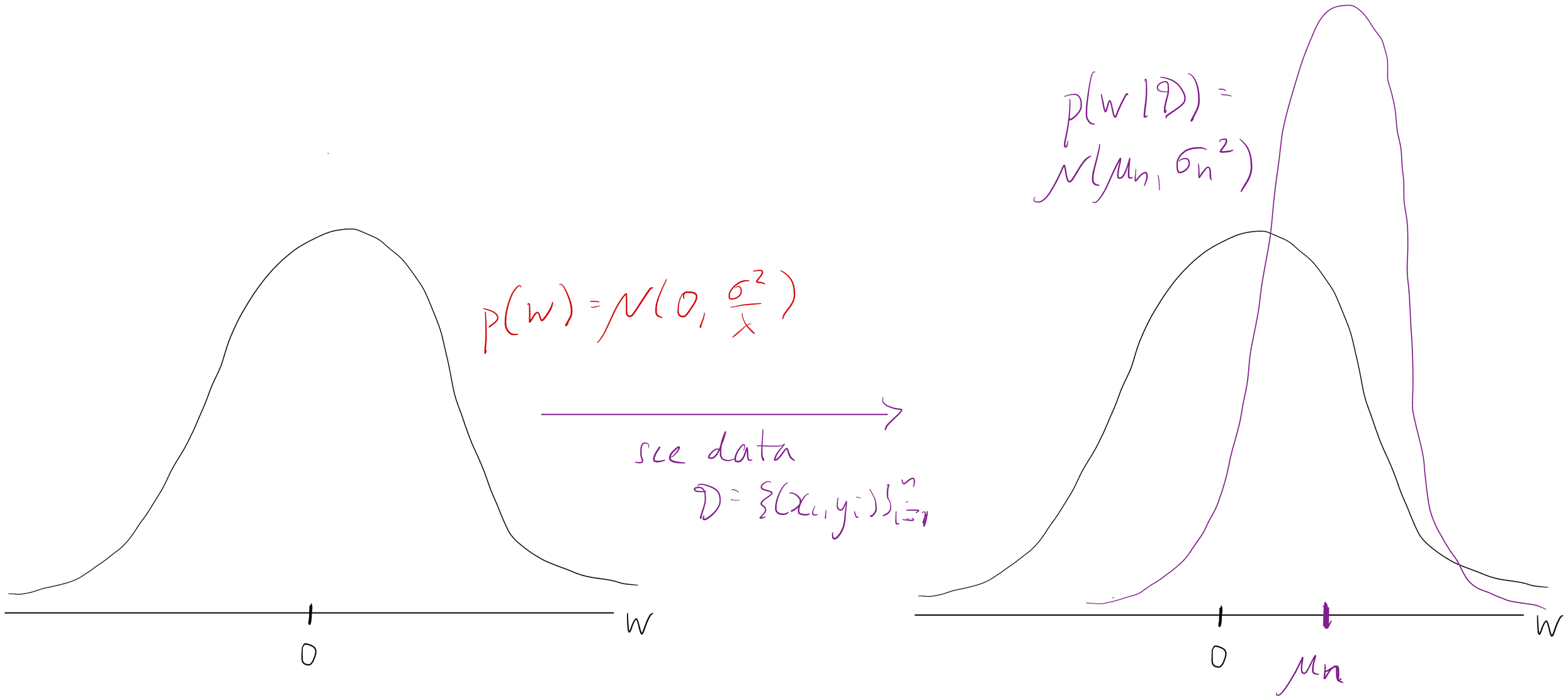
- Our goal in direct prediction was to learn an NN that conditions on $(\mathbf{x}_{\mathscr{A}}, \mathscr{M})$
- What if we knew that we would only ever see feature 1 or feature 2 missing. We decide to train three NNs: one using only features 2 to d, one using only features 1, 3 to d and one using all features.
- Q: In deployment, how do you use these three models?
- Q: How might you train these?
- Q: Is this also learning a model that conditions on $(\mathbf{x}_{\mathscr{A}}, \mathscr{M})$?



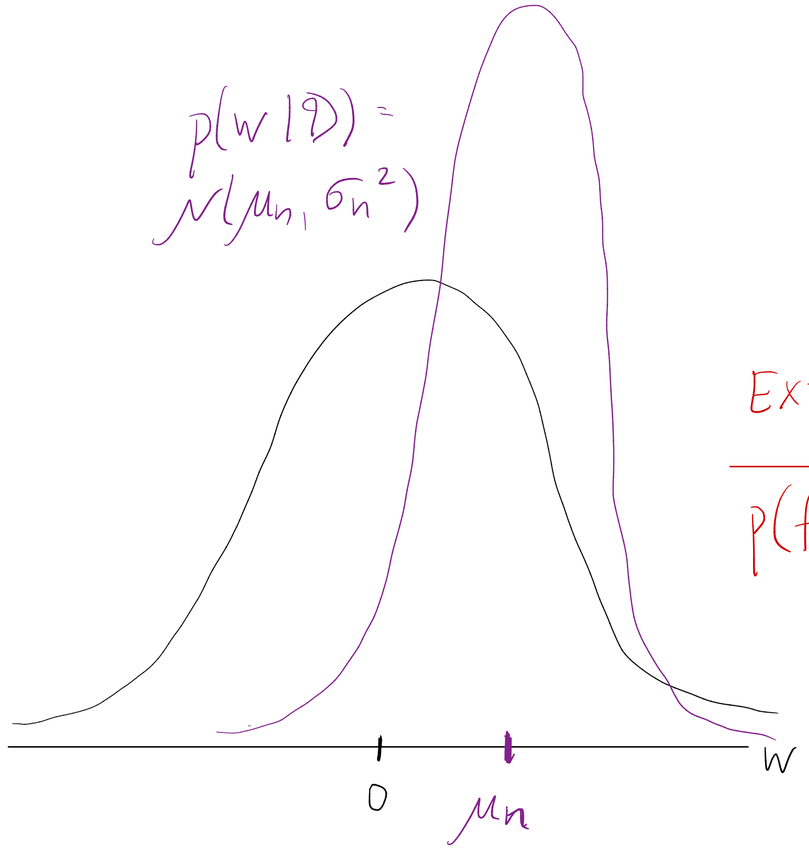
Chapter 15: Uncertainty estimation

- Understand that we might want to know distribution over plausible values of $\mathbf{w}$, given the evidence (data)
- Understand that this allows us to also obtain a distribution over our predictions, and so construct credible intervals $[f_{\mathbf{w}}(\mathbf{x}) - \epsilon, f_{\mathbf{w}}(\mathbf{x}) + \epsilon]$
- Understand why the posterior and credible interval shrink with growing n
- **Will not test you on**
  - Memorizing the formulas for Bayesian linear regression
  - The Normal-inverse gamma distribution

Shrinking posterior

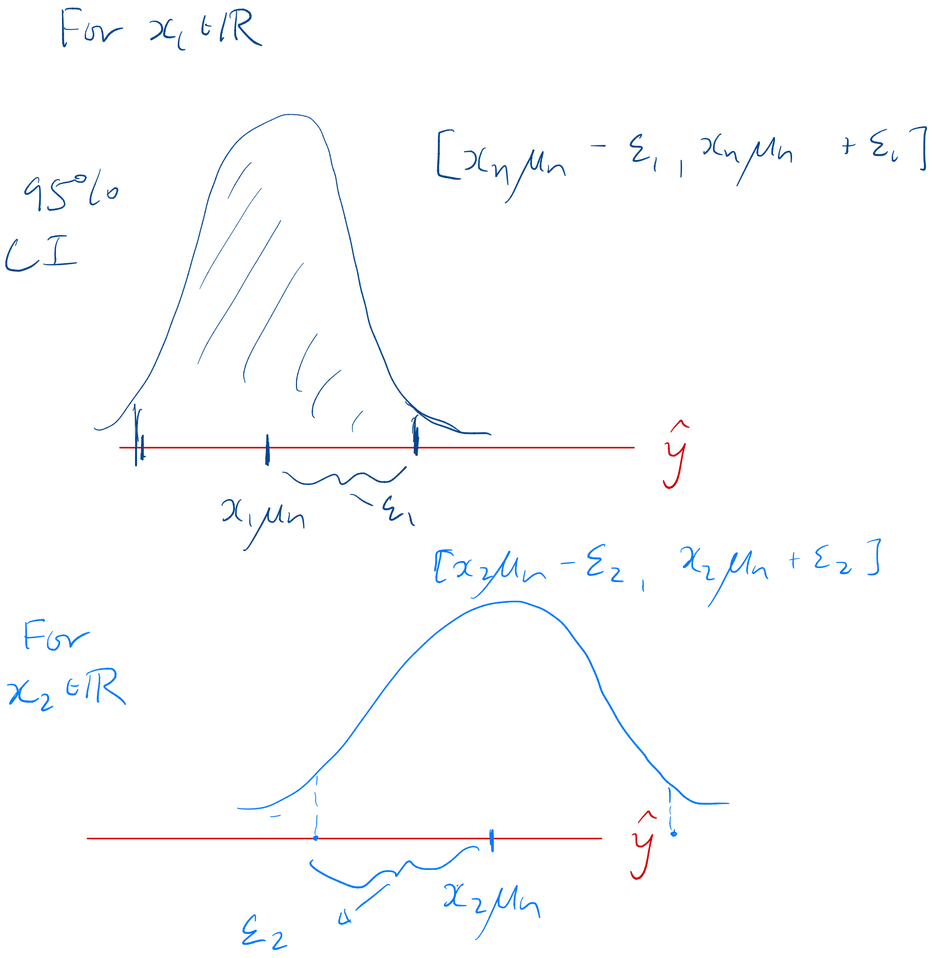


Credible Interval for Predictions



Extract

$p(f_w(x)|D)$

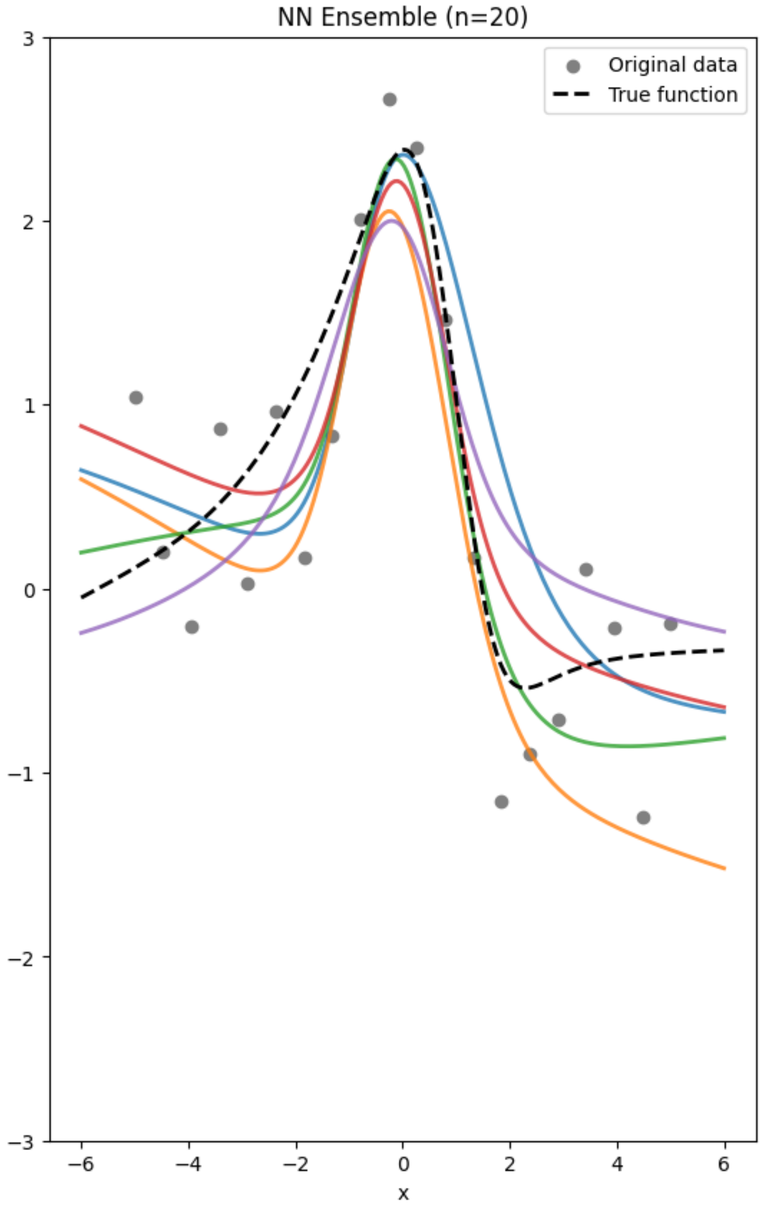


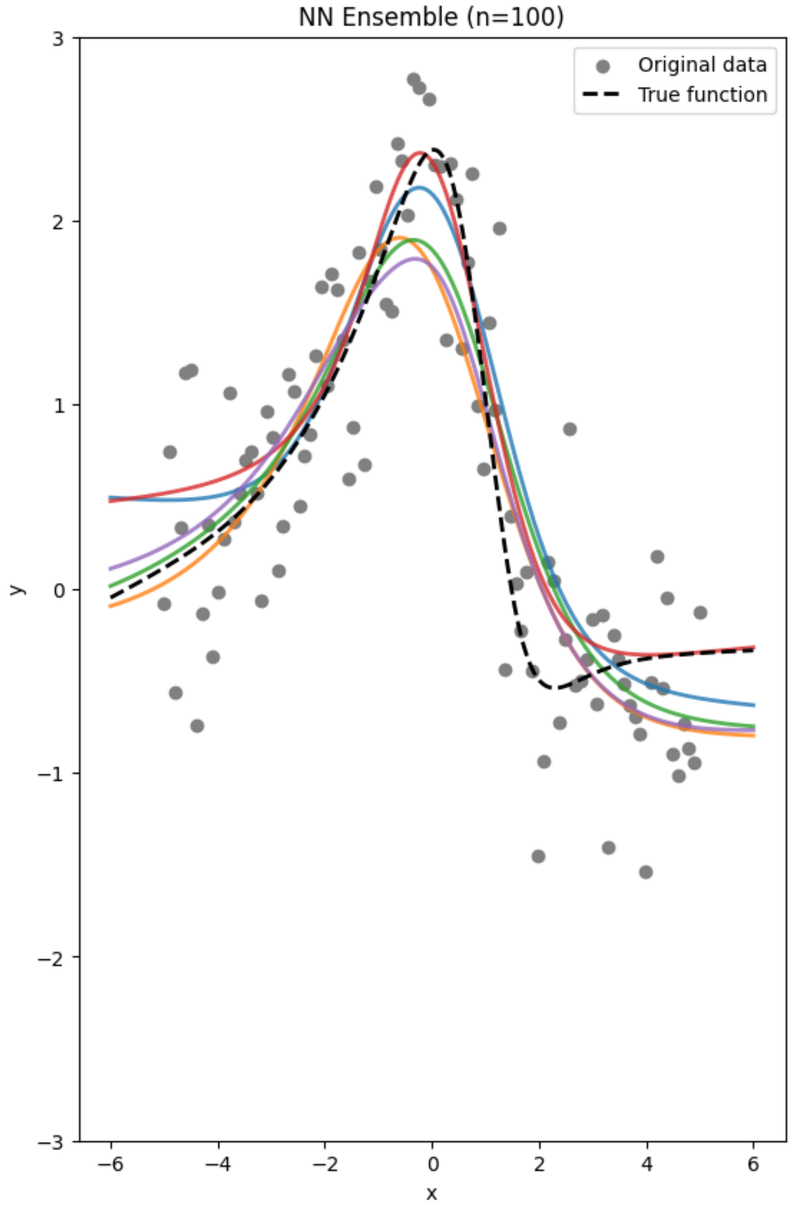
Chapter 16: Uncertainty estimation

- Understand that we can use fixed representations + Bayesian linear regression to get nonlinear regressors + credible intervals
- Understand that we can use bootstrap resampling to estimate uncertainty for neural networks
  - Understand that we sample m datasets with replacement from all the training data, and that we train an NN $f_{\mathbf{W}_k}$ from scratch on each dataset k
  - We use this ensemble of NNs to get a set of predictions to compute intervals for our predictions, $\hat{y}_1 = f_{\mathbf{W}_1}(\mathbf{x}), \ldots, \hat{y}_m = f_{\mathbf{W}_m}(\mathbf{x})$
- **Will not test:** how to compute percentile CIs

Discussion

- Looking at the ensembles learned, what can you say? Can you hypothesize what subset of points were used to train the yellow line?
- Why are the lines so much more similar for n = 100, even though the data is just as variable?
- If we had n = 10,000, what might you expect to see for our ensemble of m = 5?





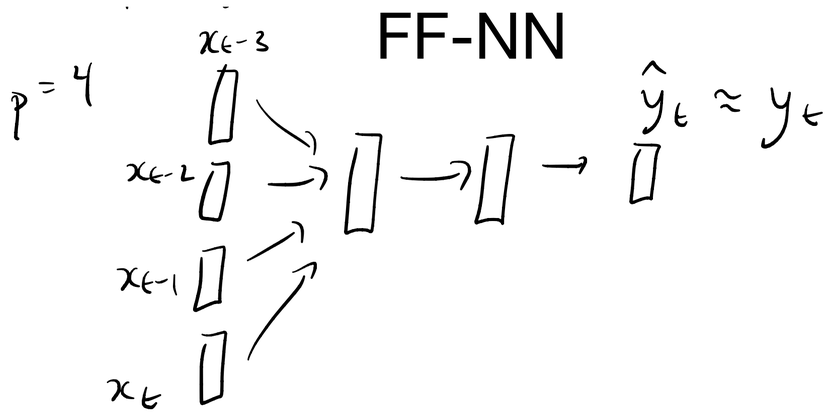
Chapter 17: Learning on temporal data

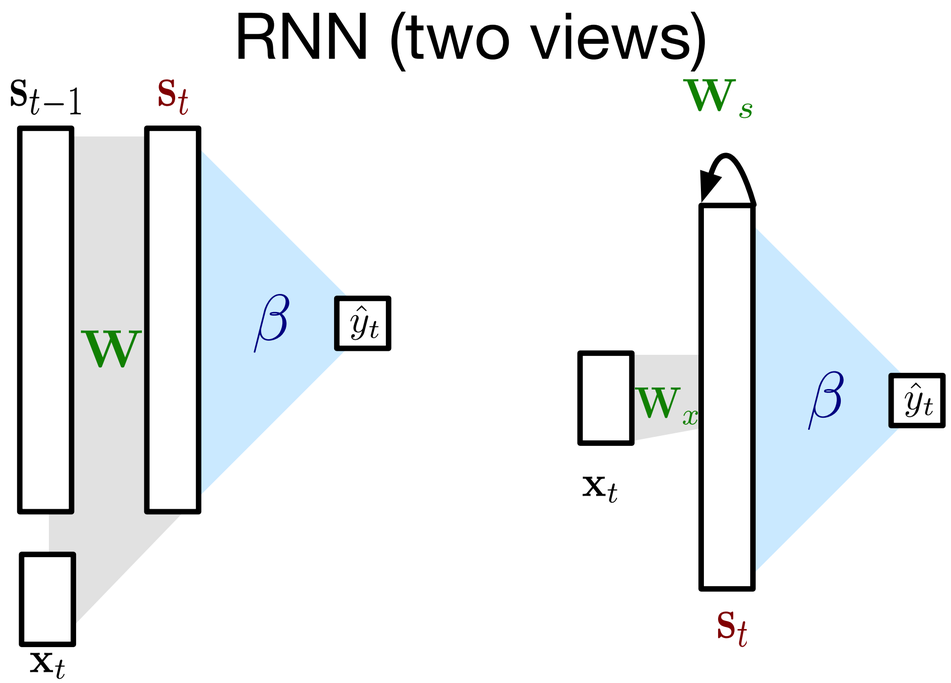
- Understand that data is of the form $\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots (\mathbf{x}_t, \mathbf{y}_t)\}$ where now index indicates time: i-1 happened before i, which happened before i+1, etc.
- Understand that these temporal relationships allow us to mitigate partial observability by conditioning on the history rather than just the immediate observation: predicting $\mathbf{y}_t$ using $\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t$ rather than just $\mathbf{x}_t$
- Understand our two key strategies:
  - conditioning explicitly on the last p points, inputting them into our function (NN)
  - using recurrence to incrementally summarize and construct state
- Understand transformers use first approach, but allow for large p by construction

Exercise

- Consider the three NN choices we considered
  - **Option 1**: stack the last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ and input to a standard feedfoward NN (e.g., multiple ReLU layers)
  - **Option 2**: use a recurrent neural network (RNN) (e.g., simple single hidden recurrent layer with ReLU, or more advanced RNNs like GRU, LSTM)
  - **Option 3**: the last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ into a transformer (a particular NN architecture)

The three architectures





Transformer subblock

before linearly combining and sending to layernorm+ReLU

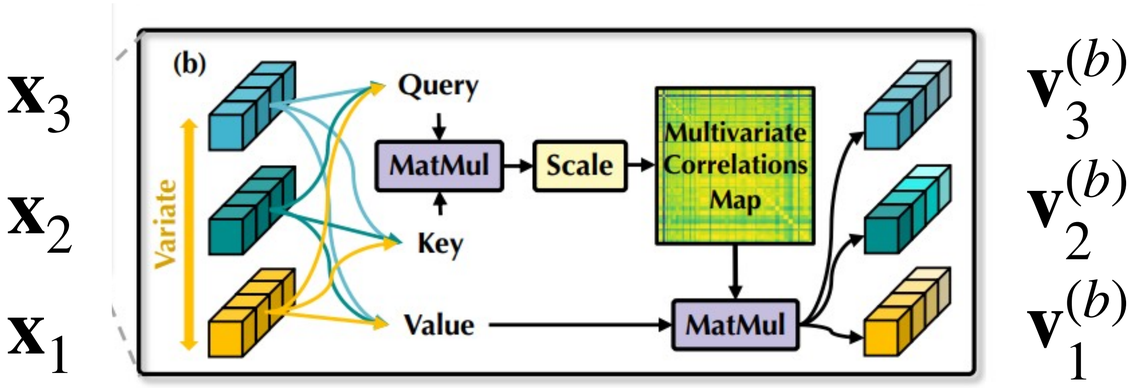


*image from
https://www.analyticsvidhya.com/blog/2024/05/a-comprehensive-guide-on-i-transformer/

Exercise

- Consider the weights for our three NN choices we considered
  - Option 1: stack the last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ and input to a standard feedfoward NN (e.g., multiple ReLU layers)
  - Option 2: use a recurrent neural network (RNN) (e.g., simple single hidden recurrent layer with ReLU, or more advanced RNNs like GRU, LSTM)
  - Option 3: the last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ into a transformer (a particular NN architecture)
- Q: Does the number of parameters grow with increasing p for the FFNN, the RNN, and/or the Transformer?
  How about with increasing observation dimension d?

Exercise

- Consider the weights for our three NN choices we considered
  - Option 1: stack the last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ and input to a standard feedfoward NN (e.g., multiple ReLU layers)
  - Option 2: use a recurrent neural network (RNN) (e.g., simple single hidden recurrent layer with ReLU, or more advanced RNNs like GRU, LSTM)
  - Option 3: the last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ into a transformer (a particular NN architecture)
- Q: Does the number of parameters grow with increasing p for the FFNN, the RNN, and/or the Transformer? How about with increasing observation dimension d?
- **Ans**: #parameters grows with increasing d for all three, #parameters grows with increasing p for FNN, but not for RNN or Transformer

Exercise

- All of our approaches created a new representation that summarizes the history
  - Option 1: input last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ to feedfoward NN produces final hidden layer $\mathbf{h}^{(1)}$
  - Option 2: Recurrence is a state update function, that provides current compact summary of sequence:
  - Option 3: input last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ to Tranformer produces $\mathbf{z}_{t-p}, \ldots, \mathbf{z}_t$ where we used $\mathbf{z}_t$ as our new representation
- Q: How do we use this new representation? Do we always use it to predict $\mathbf{x}_{t+1}$?

Exercise

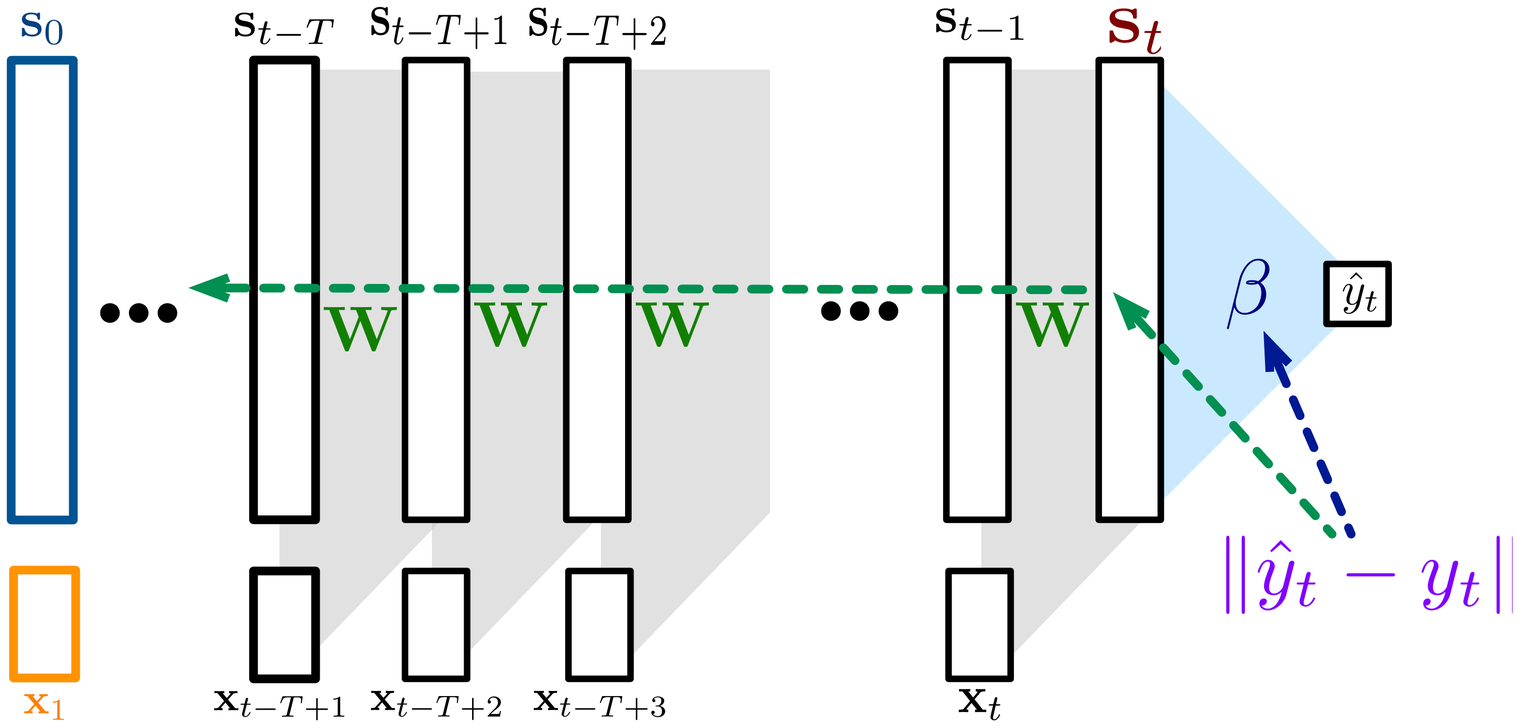
- All of our approaches created a new representation that summarizes the history
  - Option 1: input last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ to feedfoward NN produces final hidden layer $\mathbf{h}^{(1)}$
  - Option 2: Recurrence is a state update function, that provides current compact summary of sequence:
  - Option 3: input last p observations $\mathbf{x}_{t-p}, \ldots, \mathbf{x}_t$ to Tranformer produces $\mathbf{z}_{t-p}, \ldots, \mathbf{z}_t$ where we used $\mathbf{z}_t$ as our new representation
- Q: How do we use this new representation? Do we always use it to predict $\mathbf{x}_{t+1}$?
  - No. Like any NN, we can use it to predict any target we want. Can predict $\mathbf{x}_{t+10}$ or could predict something not in the time series, such as a classification target (e.g., does the image in a sequence contain a cat or not)

Chapter 17: Learning on temporal data (cont.)

- Understand all three approaches are trained using gradient descent (backprop), but that for an RNN this involves unrolling the recurrence (computing the gradients back in time, called backprop through time)
- Understand a predicted sequence can be generated autoregressively: predict $\hat{\mathbf{x}}_{t+1}$, then use your own prediction to predict $\hat{\mathbf{x}}_{t+2}$, etc.
- **Will not be directly tested**
  - The Transformer architecture and any specific RNN architectures
  - Do not need to know nuances around training RNNs with backprop, including how to pick truncation, how to use mini-batches, etc.

Discussion: BPTT for an RNN vs backprop for a transformer

- Why don’t we say a transformer is using BPTT? It is also looking at the sequence back in time?
- Why might an RNN with BPTT be more prone to vanishing or exploding gradients?
- For a feedforward NN, do we use BPTT or standard backprop?



Chapter 18: Gen error beyond iid

- Entire chapter will not be tested