CS181 Spring 2022 Midterm Review Section

Sean Ty, Oliver Cheng

February 2022

1 Regression

1.1 Linear Regression

Consider data points $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where the $\mathbf{x}$'s have p dimensions. In linear regression, we think there is a linear relationship between the predictors and the response: noise $+ w_0 + \mathbf{w}^T\mathbf{x}_n = y_n$.

Bias Trick For ease of notation, we sometimes add an intercept term to each observation $\mathbf{x}$ instead, so that the model becomes noise $+ \mathbf{w}^T\mathbf{x}_n = y_n$. For instance, if originally $\mathbf{x}_n = (3.14, 2.71)$ then if we use the bias trick, it becomes $\mathbf{x}_n = (1, 3.14, 2.71)$. We do this so that we can absorb the w_0 term into the vector $\mathbf{w}$. In this handout we will be automatically using the bias trick.

Loss Function Purely a function of the observed response and the model predictions, $\mathcal{L}(y, \hat{y})$. Intuitively, we can think of this as how much the model messed up. For linear regression, we use the sum of squared errors: $\mathcal{L} = \sum_{n=1}^N (y_n - \hat{y}_n)^2$. Sometimes we use the absolute errors $\sum_{n=1}^N |y_n - \hat{y}_n|$, which is less sensitive to outliers, but there is no closed form.

Deriving Optimal Weights Note that we predict $\hat{y}_n = \mathbf{w}^T\mathbf{x}_n$. We then want to minimize the loss $\sum_{n=1}^N (y_n - \mathbf{w}^T\mathbf{x}_n)^2$. Taking the derivative of this with respect to $\mathbf{w}$ and setting to zero, we get $\mathbf{w}^* = (X^TX)^{-1}X^Ty$. Note that X here is the model matrix, where each row corresponds to each $\mathbf{x}_n^T$, and each element of y is y_n.

Basis Functions These give our model flexibility by using transformed versions of the predictors. For instance, if we believed that there is a quadratic relationship between x and y, we would use the basis $1, x, x^2$ and form a linear regression with these features instead.

1.2 K-Nearest Neighbors

We predict the response of a new data point $(\mathbf{x}^*, y^*)$ to be the average of the responses of the K observations that are "closest" to the new data point. We select K, which is sometimes called the *hyperparameter*.

A few things to consider:

1. Need a notion of distance between the $\mathbf{x}$'s. Usually, we use the Euclidean metric. Note that because of this, using KNN on datasets with categorical predictors can be tricky.
2. KNN interpolates in a more "steppy" manner, and is constant when we extrapolate.

1.3 Kernelized Regression

We predict a new data point $(\mathbf{x}^*, y^*)$ with a kernel $K(x, x^*)$ on our training data $\{(\mathbf{x_i}, y_i)\}_{i=1}^N$.

$$\mathbf{y}^* = \sum_{i=1}^{N} K(\mathbf{x_i}, \mathbf{x}^*) y_i.$$

This technique is essentially equivalent to taking a weighted average of your data to find y^*, where the weights are determined by $K(x, x^*)$.

1.4 Regularization

Trades off bias and variance, which could lower MSE as a result. Note that we **do not penalize the intercept term**. Intuitively, can think of higher regularization as trading off more bias for less variance.

1. **Ridge regression**: Adds a penalty term $\lambda \sum_{i=1}^{p} w_i^2 = \lambda \|\mathbf{w}\|_2^2$ (do **not** penalize intercept term).
   (a) Results in smoother regularization, where parameters are *shrunk* towards zero as regularization rate λ increases.
   (b) Closed form solution for optimal $\mathbf{w}$ exists: $\mathbf{w}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y$.
2. **Lasso regression**: Adds penalty term $\lambda \sum_{i=1}^{p} |w_i| = \lambda \|\mathbf{w}\|_1$ (again, do **not** penalize intercept term).
   (a) Results in more discretized regularization, where parameters are *sent* to zero as regularization rate λ increases.
   (b) Closed form solution does not exist. Can use methods such as gradient descent to find optimal $\mathbf{w}$.

Addendum for practical applications: We usually normalize (i.e. center and scale to a sample variance of 1) each feature before regularizing. Can you see why?

2 Bayesian Regression

2.1 Regularization

Connection to frequentist:

1. For ridge regression, use normal prior on $\mathbf{w}$: $\mathbf{w} \sim \mathcal{N}(0, \tau^2 I)$. Compute posterior of $\mathbf{w}$ based on observed data, and posterior mode (MAP) becomes ridge estimate.
2. For lasso regression, the same idea except with a Laplace prior.

3. Useful to note normal-normal conjugacy, and beta-binomial conjugacy.

2.2 More Bayesian

1. Posterior of $\mathbf{w}$: we update our confidence on $\mathbf{w}$ as we have more observations, given by

$$p(\mathbf{w}|D) = \frac{p(\{y_n\}|\{\mathbf{x}_n\}, \mathbf{w})p(\mathbf{w})}{p(\{y_n\}|\{\mathbf{x}_n\})}.$$

2. Posterior predictive: how can we use observed data to better predict? We have a probability distribution for response given data:

$$p(y|\mathbf{x}, D) = \int_{\mathbf{w}} p(y|\mathbf{x}, \mathbf{w})p(\mathbf{w}|D)\ d\mathbf{w}.$$

3. Marginal likelihood: compute

$$p(D) = \int_{\mathbf{w}} p(D|\mathbf{w})p(\mathbf{w})\ d\mathbf{w}.$$

   If considering too many possiblke models $\mathbf{w}$, then $p(\mathbf{w})$ will be spread out, which makes integral low. If $p(w)$ does not contain good models, then $p(D|\mathbf{w})$ would be very low. This is how Bayesian methods penalize complicated models.

3 Classification

Common Loss Functions . Loss functions are used to calculate how "wrong" our set of parameters are.

1. **0/1 Loss**: Loss that is 0 if classified correctly, and 1 otherwise. Is this a good loss?
2. **Hinge Loss**: Loss that is 0 if classified correctly, and "how wrong" we are otherwise. Comments: Gradient is computable here, where the only computable terms will be those that were incorrectly classified and not on the decision boundary.
   (a) ReLU function: $\text{ReLU}(z) = \max(0, z)$. This experimentally seems to be able to provide broader generalizeability (especially in NN as we will see).
3. **Logistic Loss**: This also referred to as *cross-entropy* loss or *log loss* or *negative log likelihood*. The idea behind this is to maximize the likelihood function,

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} y_n \log(p(y_n)|\mathbf{x}) + (1 - y_n)\log(1 - p(y_n|\mathbf{x})).$$

   This come directly from the task of trying to maximize the likelihood

$$p(\mathbf{y}|\mathbf{w}) = \prod_n p(y_n = 1|\mathbf{x_n})^{y_n} \left(1 - p(y_n = 1|\mathbf{x_n})\right)^{1-y_n}.$$

The multi-class analogue comes from taking the negative log of the following expression for K classes,

$$p(\mathbf{y}|\mathbf{W}) = \prod_{n=1}^{N}\prod_{k=1}^{K} p(\mathbf{y}_n = C_k|\mathbf{x_n})^{y_{nk}}.$$

To be clear, $\mathbf{y}_n$ is generally encoded as a *one-hot vector*, and equality to class C_k just means that this data point is classified as class k.

Stochastic Gradient Descent All this involves is computing the gradient on a subset of our data. This subset at each iteration will be random, and updates will only be on this subset. The main purpose for SGD is to increase efficiency, especially if data is clustered, then computing several gradients at similar points is not particularly helpful. Section 3.4.2 Textbook.

Note that one case of stochastic gradient descent is the *perceptron algorithm*, where you only compute the derivative on one single datapoint. This will always converge as each iteration slightly tilts our hyperplane to classify better, but only if our data is *linearly separable*. This condition is necessary as otherwise the algorithm would not stop, and in general all logistic regressors build a linear decision boundary with respect to their basis. A proof and more detailed explanation by Kalyanakrishnan can be found here.

Generative Models Instead of modeling $p(y|\mathbf{x})$, we model the joint $p(\mathbf{x}, y)$–i.e. we model how the data is being generated. We usually assume that data point is modeled by first choosing the class y, then generating the feature values $\mathbf{x}$. We use

$$p(y^* = C_k|\mathbf{x}^*) \propto p(\mathbf{x}^*|y^* = C_k)p(y^* = C_k).$$

Note that in particular this requires us to provide a prior for the classes.

Generative vs. Discriminative The key difference between them is that discriminative models the conditional $p(y|\mathbf{x})$ (i.e. it "discriminates" based on the features). Meanwhile, generative models the joint $p(\mathbf{x}, y) \propto p(\mathbf{x}|y)p(y)$–i.e. it models how the data are generated.

Examples: For discriminative, we could be classifying the brands of cars based on features like price, number of seats, engine, etc. For generative, we could be doing sentiment analysis in NLP, where we are modeling the probability of a specific word appearing together with a response sentiment (this is also a great example of Naive Bayes in applications).

Naive Bayes A simplifying assumption of conditional independence, so that

$$p(\mathbf{x}^*|y^* = C_k) = \prod_{i=1}^{D} p(x_i^*|y^* = C_k).$$

This assumption significantly simplifies calculations.

4 Model Selection

Bias and Variance of a model Important to note here that the data D was generated randomly. Since we fit a model onto the data, our trained model f_D is also random. Averaging over all possibilities of D, we obtain a mean model $\overline{f}$. Let's say we want to predict the value given a predictor $\mathbf{x}$.

1. **Bias**: Quantifies the intuition: "Given enough data, how far will our model be from the true mean?" Given by the quantity

$$\text{Bias} = \overline{f}(\mathbf{x}) - \overline{y}.$$

2. **Variance**: Quantifies the intuition: "How much does our model change if our data set changes?" Given by the quantity

$$\text{Variance} = \mathbf{E}_D[(f_D(\mathbf{x}) - \overline{f}(\mathbf{x}))^2)].$$

3. **Combining together**: Expected generalization error for a new data point $\mathbf{x}$ is given by $\text{Error} = \text{bias}^2 + \text{variance} + \text{noise}$.

4. **Bias-Variance Tradeoff**: Controllable factors for generalization error are bias and variance of model. Usually a tradeoff between bias and variance–opting for less bias means more variance, and vice-versa.

Cross-Validation Used to prevent overfitting, when tuning hyperparameter values. We essentially create artificial copies of train and validation sets, to better see how a specific set of hyperparameters perform under "new data." The best set of hyperparameters is evaluated on the test set.

1. **K-fold Cross Validation**: Divide training data into K disjoint (roughly) equally sized sets. For each set, we can allocate it as the validation set and train the model with the remaining $K-1$ sets. Compute the loss for each iteration and average, choosing the hyperparameter setting with least average loss.

5 Neural Networks

Recall in linear regression the importance of adding a basis in order to improve the predictive ability of our model. Fundamentally, neural networks were built upon this idea.

5.1 Activation Functions

If we did not have activation functions, there would be no interesting basis (why?). Four functions

- Linear: $\phi(\mathbf{x}) = \mathbf{ax}$
- Rectified Linear Unit (ReLU): $\phi(\mathbf{x}) = \max(0, \mathbf{x})$. This ends up being the gold standard for activation functions.
- Sigmoid: $\phi(\mathbf{x}) = \dfrac{1}{1+e^{-\mathbf{x}}}$. Motivation for finite range is that a weight should most likely never tend to infinity.

- Tanh: $\phi(\mathbf{x}) = \dfrac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}}$. Compared to sigmoid, this performs better due to stronger (steeper) gradients and being centred around 0.

5.2 Forward Pass

Given a data set, we can compute our predictions via a forward pass. Due to the power of matrix multiplication, we can do this all extremely quickly. Namely, if layer ℓ has N_ℓ nodes, and the weight matrix is $W = (w_{ij}^{(\ell)})$, we can calculate the value for each node as according to the diagram.

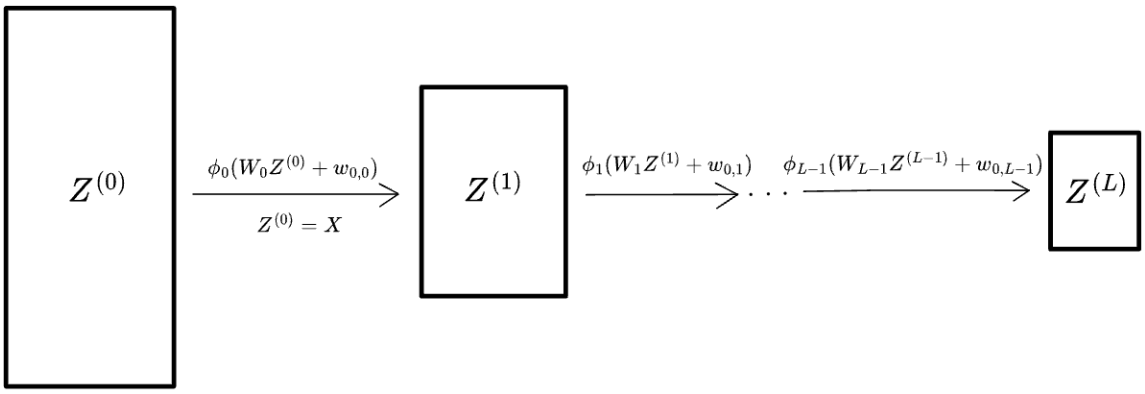

Figure 1: Forward pass where $Z^{(\ell)}$ $(N_\ell \times 1)$ represents the nodes in the ℓ layer.

In particular, the value of the subsequent nodes $z_i^{(\ell+1)}$ in the forward pass is

$$z_i^{(\ell+1)} = \phi_\ell(w_{0,\ell} + \sum_{k=1}^{N_\ell} z_k^{(\ell)} w_{ik}^{(\ell)}).$$

Where ϕ_ℓ is our activation function for the ℓ layer.

5.3 Loss Function

There are two choices, for regressions we use least squares, while for classification we use softmax/logistic regression.

5.4 Backpropagation

Backpropagation is the technique we will use to update the parameters of our model. It is the equivalent to gradient descent, but the update is for each layer of the Neural Network. We want to compute the gradient of the loss with respect to the weights, but doing this directly is infeasible, so set $W_\ell Z^{(\ell)} + w_{0,\ell} = A^{(\ell)}$ and we will use the chain rule as follows:

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = \frac{\partial \mathcal{L}}{\partial Z^{(\ell)}} \frac{\partial Z^{(\ell)}}{\partial A^{(\ell)}} \frac{\partial A^{(\ell)}}{\partial W_\ell}.$$

We have three terms here.

- $\frac{\partial \mathcal{L}}{\partial Z^{(\ell)}}$. At the last layer, this can be directly computed from the loss function. To compute the error term at layer L.

$$\frac{\partial \mathcal{L}}{\partial Z^{(\ell)}} = \frac{\partial \mathcal{L}}{\partial Z^{(L)}} \frac{\partial Z^{(L)}}{\partial Z^{(L-1)}} \cdots \frac{\partial Z^{(\ell+1)}}{\partial Z^{(\ell)}}.$$

  Where $\frac{\partial Z^{(\ell+1)}}{\partial Z^{(\ell)}} = \phi'_\ell \left(A^{(\ell)}\right) W_\ell$

- $\frac{\partial Z^{(\ell)}}{\partial A^{(\ell)}}$, this is simply the gradient of the activation function ϕ_ℓ.
- $\frac{\partial A^{(\ell)}}{\partial W_\ell}$. We can compute this directly, since it is a linear combination, we have $\frac{\partial A^{(\ell)}}{\partial W_\ell} = W^{(\ell)}$.

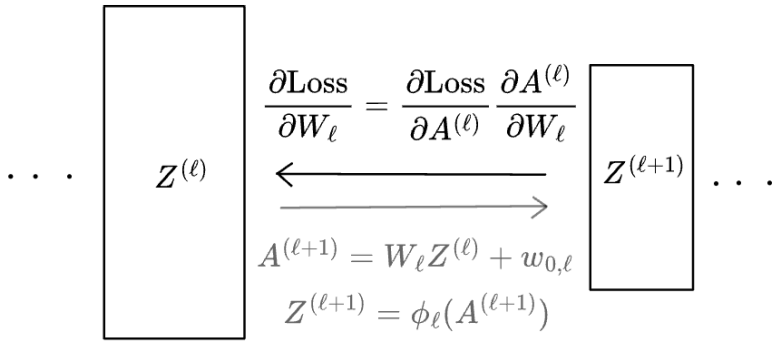

Figure 2: Backpropagation between layer ℓ and $\ell + 1$