CS 1810 Spring 2026 Section 2 Notes
(Model Selection)

1 Model Selection

1.1 Bias-Variance Decomposition

Bias-variance decomposition is a way of understanding how different sources of error (bias and variance) can affect the final performance of a model. A tradeoff between bias and variance is often made when selecting models to use.

1.2 Exercise: Bias-Variance Decomposition

Decompose the generalization error (or mean squared error) into the sum of squared bias (systematic error), variance (sensitivity of prediction), and noise (irreducible error) by following the steps below. You will find the following notation useful:

- f_D: The trained model, $f_D : \mathcal{X} \mapsto \mathbb{R}$.
- D: The data, a random variable sampled from some distribution $D \sim F^n$.
- $\mathbf{x}$: A new input.
- y: The true result of input $\mathbf{x}$. Conditioned on $\mathbf{x}$, y is a r.v. (there may be noise involved.)
- $\overline{y}$: The true conditional mean, $\overline{y} = \mathbb{E}_{y|\mathbf{x}}[y|\mathbf{x}]$.
- $\overline{f}(\mathbf{x})$: The predicted mean from the model, $\overline{f}(\mathbf{x}) = \mathbb{E}_D[f_D(\mathbf{x})]$.
- $E_D(\cdot)$ is the expectation with respect to the data (for a quantity that depends on the data, and over the data's distribution). $E_{y|\mathbf{x}}(\cdot)$ is the expectation of a quantity over the conditional distribution of y given $\mathbf{x}$.

1. Start with the equation for the mean squared error or generalization error:

$$\mathbb{E}_{D,\,y|\mathbf{x}}[(y - f_D(\mathbf{x}))^2]$$

and use the linearity of expectation to derive an equation of the form:

$$\underbrace{\mathbb{E}_{y|\mathbf{x}}[(y-\overline{y})^2]}_{\text{noise}} + \underbrace{\mathbb{E}_D[(\overline{y} - f_D(\mathbf{x}))^2]}_{\text{bias+var}} + \boxed{???} \tag{1}$$

where the $\boxed{???}$ denotes a third term. What is this term? **Hint**: add and subtract $\overline{y}$.

2. Show that this third term is equal to 0 (**Hint**: take advantage of the fact that $\overline{y}$ and $f_D(\mathbf{x})$ do not depend on $y|\mathbf{x}$).

3. The first term in (1) is the noise. We therefore want to decompose the second term into the bias and variance. Again, using the linearity of expectation, re-write the second term in equation (1) in the form:

$$\underbrace{(\overline{y} - \overline{f}(\mathbf{x}))^2}_{\text{bias squared}} + \underbrace{\mathbb{E}_D[(\overline{f}(\mathbf{x}) - f_D(\mathbf{x}))^2]}_{\text{variance}} + 2\mathbb{E}_D[(\overline{y} - \overline{f}(\mathbf{x}))(\overline{f}(\mathbf{x}) - f_D(\mathbf{x}))] \tag{2}$$

Show that the third term is equal to 0.

4. Plug the results of part 3 back into (1) to show that we have decomposed the error into noise, bias, and variance.

Solution:

1. Follow the hint:

$$\begin{aligned}
&\mathbb{E}_{D,y|\mathbf{x}}[(y - f_D(\mathbf{x}))^2] \\
&= \mathbb{E}_{D,y|\mathbf{x}}[(y - \overline{y} + \overline{y} - f_D(\mathbf{x}))^2] \\
&= \underbrace{\mathbb{E}_{y|\mathbf{x}}[(y - \overline{y})^2]}_{\text{noise}} + \underbrace{\mathbb{E}_D[(\overline{y} - f_D(\mathbf{x}))^2]}_{\text{bias+var}} + \underbrace{2\mathbb{E}_D[(y - \overline{y})(\overline{y} - f_D(\mathbf{x}))]}_{0}
\end{aligned}$$

2. Using the hint to apply the law of iterated expectations (also known as Adam's law):

$$2\mathbb{E}_D[(\overline{y} - f_D(\mathbf{x})) \cdot \mathbb{E}_D[(y - \overline{y})|\mathbf{x}]] = 2\mathbb{E}_{y|\mathbf{x}}[(\overline{y} - f_D(\mathbf{x})) \cdot 0] = 0.$$

3. Following a similar procedure as in part 1:

$$\begin{aligned}
&\mathbb{E}_D[(\overline{y} - f_D(\mathbf{x}))^2] \\
&= \mathbb{E}_D[(\overline{y} - \overline{f}(\mathbf{x}) + \overline{f}(\mathbf{x}) - f_D(\mathbf{x}))^2] \\
&= \underbrace{(\overline{y} - \overline{f}(\mathbf{x}))^2}_{\text{bias squared}} + \underbrace{\mathbb{E}_D[(\overline{f}(\mathbf{x}) - f_D(\mathbf{x}))^2]}_{\text{variance}} + \underbrace{2\mathbb{E}_D[(\overline{y} - \overline{f}(\mathbf{x}))(\overline{f}(\mathbf{x}) - f_D(\mathbf{x}))]}_{0}
\end{aligned}$$

where the third term is 0 by:

$$2(\overline{y} - \overline{f}(\mathbf{x}))\mathbb{E}_D[\overline{f}(\mathbf{x}) - f_D(\mathbf{x})] = 2(\overline{y} - \overline{f}(\mathbf{x}))(0) = 0.$$

4. Substituting (2) back into (1), we have:

$$\begin{aligned}
&\mathbb{E}_{D,y|\mathbf{x}}[(y - f_D(\mathbf{x}))^2] \\
&= \mathbb{E}_{y|\mathbf{x}}[(y - \overline{y})^2] + (\overline{y} - \overline{f}(\mathbf{x}))^2 + \mathbb{E}_D[(\overline{f}(\mathbf{x}) - f_D(\mathbf{x}))^2] \\
&= \text{noise}(\mathbf{x}) + \text{bias}^2(f(\mathbf{x})) + \text{Var}_D(f_D(\mathbf{x})).
\end{aligned}$$

Considering the expectation over $\mathbf{x}$ (you are not asked to do this in the exercises), the generalization error is:

$$\mathbb{E}_{\mathbf{x}}\left[\text{noise}(\mathbf{x}) + \text{bias}^2(f(\mathbf{x})) + \text{Var}_D(f_D(\mathbf{x}))\right]$$

1.3 Exercise: Bias-Variance Tradeoff in Estimating an Unknown Parameter

We consider a very simple example where the data is a univariate Gaussian, with $x_i \sim \mathcal{N}(\mu, 1)$ with known variance but unknown mean. In this case, we want to estimate the true mean μ - and we are dealing with only one variable, i.e. the x_i's. A very simple hypothesis, for example, is the sample mean

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

for data $(x_1, \ldots, x_n) \in \mathbb{R}^n$. Calculate the bias and variance for the following two hypotheses:

1. Estimate 1: Use the same mean of data D.
2. Estimate 2: Use the constant hypothesis, 0.

Solution:

1. The bias is $\mu - E_D[\overline{x}] = \mu - E_D[(1/n)\sum x_i] = 0$. The variance is:

$$\begin{aligned} Var(\overline{x}) &= Var\left(\frac{x_1 + ... + x_n}{n}\right) \\ &= \frac{1}{n^2} Var\left(x_1 + ... + x_n\right) \\ &= \frac{1}{n^2} \cdot nVar(x_1) \\ &= \sigma^2/n \end{aligned}$$

a standard result for the variance of sample mean on n i.i.d. examples.

2. Assuming the mean is 0 is an arbitrary choice, so the bias equals μ, the expected difference between the estimator and the true value. However, this prediction is constant, so the variance of our prediction is 0.

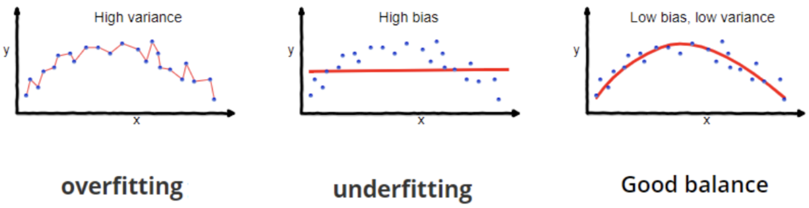

Figure 1: A helpful image to illustrate this tradeoff taken from here. The blue points are the training data points, and the red line is the model we fit.

1.4 Limitations

How can you test the "variance" of your estimate/model (i.e., how it would vary if you estimated it a bunch of times on **a bunch of datasets drawn from the same distribution**), when in practice you only have access to **one dataset**?

We can solve this problem by ”bootstrapping” (i.e. sampling a bunch of ”synthetic” datasets from the data we have, with replacement). But there are easier ways to solve this problem.

1.5 Validation Set

A *validation set* contains data that are separate from our training set used to fit the regression. Here is a sample process:

1. Separate our full dataset into a training set and validation set (say in a 90/10 split).
2. Train your models with different parameters on the training set. Each time, check the performance on the validation set.
3. This gives you an optimal value for the parameter.

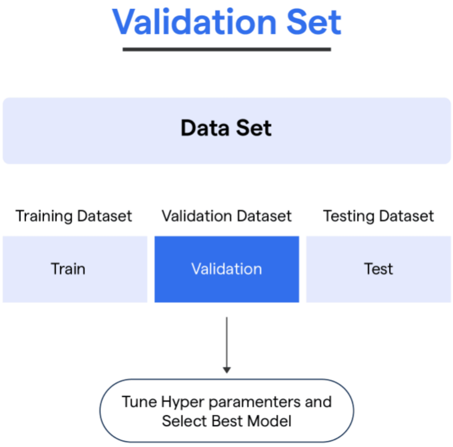

Figure 2: A helpful image to illustrate the validation set taken from here.

1.6 Cross Validation

Cross validation is a more sophisticated technique for obtaining validation losses.

1. In k-fold cross validation, we split our data into k equal chunks.
2. For each chunk, we set it to be the validation set and use the rest of the $k-1$ chunks together as the training data to fit our model.
3. Then, we obtain a validation loss on our current chunk.
4. Averaging loss over the k chunks gives us a final validation loss.

Now, we have an improved way of computing validation losses by averaging. This reduces the variance in the resulting validation loss - since we’ve used every data point in doing so!

See this notebook for an interactive demo of how cross validation could be used.

1.7 Ensemble Methods

Ensemble methods take advantage of multiple models to obtain better predictive accuracy than with a single model alone. The two most common types are bagging and boosting.

1.7.1 Bootstrap aggregating (Bagging)

- In bagging, we fit each individual model on a random sample of the training set.
- To predict data in the test set, we either use an average of the predictions from the individual models (for regression) or take the majority vote (for classification).
- This tends to lower variance without changing bias, since it's an average of models!
- **Example:** Random forest, which is an average of predictions from decision trees!

1.7.2 Boosting

- In boosting, we train the individual models sequentially. After training the i^{th} model on a sample of the training set, we train the $(i+1)^{th}$ model on a new sample based on the performance of the i^{th} model.
- Thus, examples classified incorrectly in the previous step receive higher weights in the new sample, encouraging the new model to focus on those examples.
- During testing, we take a weighted average or weighted majority vote of the models' predictions based on their respective training accuracies on their reweighted training data (i.e. higher models have larger weights).
- **Example:** The Adaboost algorithm is a common example.

1.8 Exercise: Model Selection Using Bias and Variance

You have a dataset about n dogs. Your want to model weight of a dog (dependent var.) using age (independent var.). You fit and evaluate three models using the same train-test split, a linear model, a quadratic model, and a cubic model. Below is a table of the train and test accuracies.

	Linear	Quadratic	Cubic
Train	0.60	0.82	0.93
Test	0.62	0.73	0.54

1. For each model, would you choose to regularize? What would regularization do?
2. For each of the three models, what would be the effect of adding more data and why?
3. How would each of these models perform on a freshly drawn set of dogs? Assume that the draws across both data sets are i.i.d. (i.e. using same breeds, etc. in both data sets).
4. Based on these results, which model do you think is the most appropriate for this data?

Solution:

1. The linear model is already fitting quite well as seen by the fact that the test performance exceeds the train performance, so no regularization is needed. In fact, regularization would probably only serve to reduce its test performance, making the model even weaker than it is.

   The quadratic and cubic models both appear to do worse on the test than the train set, suggesting that they are both overfit and would therefore benefit from regularization. If we regularize, the training accuracies in these cases will likely decrease, but this is not a problem since the test accuracies should consequently increase.

2. The linear model will see no significant difference in performance with more data, since it is already not overfit. In fact, it might be somewhat underfit.

   The quadratic model, as mentioned before, is overfit and would likely benefit from more data, since the addition of data tends to reduce overfitting.

   Similarly, the cubic model appears even more overfit than the quadratic model, in which case data augmentation will be beneficial in fitting a more representative model.

3. Since both the quadratic and cubic models are overfit, there is likely to be high variance and therefore substantially different performance results if they are applied to another data set. It will be difficult to say which of the quadratic and cubic models will perform better, but it can be fair to presume that the cubic model will have higher variance due to its more extreme overfitting.

   The linear model, on the other hand, might actually be somewhat underfit, and therefore be quite biased. Based on what we know of bias-variance relationships, that suggests that the linear model will have little variance across data sets and therefore perform similarly.

4. The quadratic model almost certainly looks better than the linear model, due to the higher test performance, even if the quadratic model is overfit. However, it is unlikely but still possible that the cubic model will outperform the quadratic model once proper regularization techniques are applied.

2 Regularization

2.1 Linear Regression

Suppose we have data $\{(x_i, y_i)\}_{i=1}^N$, with $x_i, y_i \in \mathbb{R}$, and we want to fit polynomial basis functions:

$$\boldsymbol{\phi}(x)^\top = [\phi_1(x) = 1, \phi_2(x) = x, \ldots, \phi_{d+1}(x) = x^d]$$

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(x)$$

That is, we fit a degree d polynomial. With a small dataset and too high of a d, we get overfitting. Obviously, this will generalize poorly to new data points. How can we solve this problem?

Recall from Homework 1 that for bases:

- $\phi_j(x) = \cos(f(x)/j)$ for $j = 1 \ldots 9$
- $\phi_j(x) = \cos(f(x)/j)$ for $j = 1 \ldots 49$

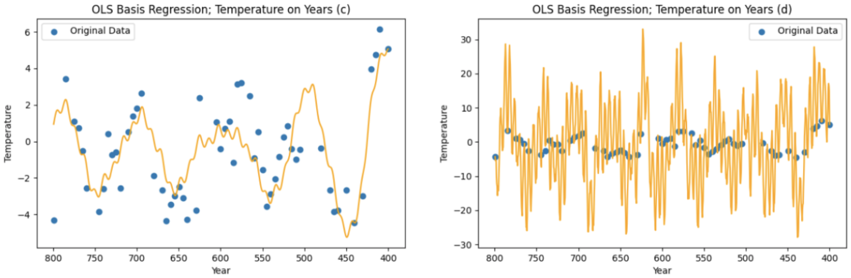

Figure 3: Plots from Hw1 Part 3 Basis Transformations c and d

2.2 Penalized Loss Function

Recall that the standard linear regression problem, known as *ordinary least squares (OLS)*, uses the following loss function (which is just the mean squared error):

$$\mathcal{L}_{OLS}(D) = MSE = \sum_{i=1}^N (y_i - f(x_i; \mathbf{w}))^2$$

Regularization refers to the general practice of modifying the model-fitting process to avoid overfitting. Linear models are typically regularized by adding a *penalization term* to the loss function. The penalization term is simply any function R of the weights $\mathbf{w}$ scaled by a penalization factor λ. The loss then becomes:

$$\mathcal{L}_{reg}(D) = \sum_{i=1}^N (y_i - f(x_i; \mathbf{w}))^2 + \lambda R(\mathbf{w})$$

There are some common choices for $R(\mathbf{w})$ that will be discussed. They frequently leverage the idea of a vector norm, where $||\mathbf{w}||_p$ represents the L_p-norm of the vector $\mathbf{w}$ for $p \geq 1$:

$$||\mathbf{w}||_p = \left(\sum_d |\mathbf{w}_d|^p \right)^{1/p}$$

A note on gradient descent. Referring to section 3.4 of the textbook, we can use gradient descent to update our weight parameters $\mathbf{w}$ using our loss function $\mathcal{L}(\mathbf{w})$. More specifically, the weight parameters at each time step t can be found using the following update rule:

$$\mathbf{w}^t = \mathbf{w}^{t-1} - \alpha \nabla \mathcal{L}(\mathbf{w}^{t-1})$$

where $\alpha > 0$ is the learning rate.

2.3 LASSO Regression

One common choice for a penalization term is simply $R(\mathbf{w}) = ||\mathbf{w}||$. This is just the L_1-norm of the weights vector, which quite naively means that the penalization term here is just the sum of the magnitudes of all the weights for the model. This form of regularized regression is known as *LASSO (Least Absolute Shrinkage and Selection Operator)* regression. The full modified loss is then:

$$\mathcal{L}_{LASSO}(D) = \sum_{i=1}^{N} (y_i - f(x_i; \mathbf{w}))^2 + \lambda ||\mathbf{w}||$$

There are some notable properties of LASSO regression. One main disadvantage is that it does not have a closed-form solution, meaning that it cannot be analytically solved. Therefore, it needs to be numerically solved through an iterative process, which can be much slower.

> **Concept Question**: Why do you think LASSO has no closed-form solution? Try to solve for it using the same process as for the OLS solution; what goes wrong?

Solution: The difficulty arises from the L_1 penalty term $||\mathbf{w}||_1 = \sum_d |w_d|$, which is not differentiable at $w_d = 0$. In contrast to OLS or ridge regression, the objective function is therefore not everywhere smooth. When attempting to solve the optimization problem by setting the gradient to zero, the derivative of $|w_d|$ is undefined at zero and discontinuous around it. As a result, the normal-equation approach used for OLS breaks down.

2.4 Ridge Regression

Another solution to overfitting linear regression is through ridge regression, which minimizes a modified least squares loss function:

$$\mathcal{L}(D) = \sum_{i=1}^{N} (y_i - f(x_i; \mathbf{w}))^2 + \frac{\lambda}{2} ||\mathbf{w}||^2$$

Ridge regression is used to *regularize* a model, making it simpler and allowing it to generalize better to new data. Indeed, the extra term penalizes overly large weights in $\mathbf{w}$, leading to smaller coefficients for a "flatter" polynomial.

Unlike LASSO, ridge regression has a closed form solution, which makes the solution much more computationally efficient. While it does not shrink coefficients to zero, it has other intuitive properties, such as connection to a Normal prior. The analytical solution is:

$$\mathbf{w}_{ridge} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y}$$

The above expression can be compared to the OLS solution. The only additional term is $\lambda\mathbf{I}$, which looks like a "ridge" of λ values (hence the name ridge regression). This also helps avoid problems with singular data.

Concept Question: Solve for the closed form solution to ridge regression.

Solution: Starting from the ridge loss

$$\mathcal{L}(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^\top\mathbf{w},$$

we differentiate with respect to $\mathbf{w}$ and set the gradient to zero:

$$-2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} = 0.$$

Rearranging terms yields

$$(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = \mathbf{X}^\top\mathbf{y}.$$

Assuming $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ is invertible, the solution is

$$\widehat{\mathbf{w}}_{ridge} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}.$$

2.5 Exercise: Ridge Regression

Suppose we have some data matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ and targets $\mathbf{y} \in \mathbb{R}^n$. Suppose the data are orthogonal*, i.e. satisfies $\mathbf{X}^\top\mathbf{X} = \mathbf{I}$. Show that if $\widehat{\mathbf{w}}$ is the solution to linear regression, and $\widehat{\mathbf{w}}_{ridge}$ is the solution to ridge regression, then

$$\widehat{\mathbf{w}}_{ridge} = \frac{1}{1+\lambda}\widehat{\mathbf{w}}$$

This explicitly illustrates the phenomenon of weight shrinkage.

Solution:

Recall that the linear regression solution is

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$$

and we have that the ridge regression solution is

$$\widehat{\mathbf{w}}_{ridge} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$$

If $\mathbf{X}^\top\mathbf{X} = \mathbf{I}$, we see that

$$\widehat{\mathbf{w}} = \mathbf{X}^\top\mathbf{y}$$
$$\widehat{\mathbf{w}}_{ridge} = \frac{1}{1+\lambda}\mathbf{X}^\top\mathbf{y}$$

hence giving us the result.

* Orthogonal data is a very special case in which the inner product between any two distinct features is zero. Normally we expect features to be correlated. But it is used to gain this clean illustration of the effect of ridge regression. Technically, we have $\mathbf{X} = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$ where $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are n dimensional, orthogonal column vectors.

2.6 Exercise: Ridge with Intercept

1. An important consideration when fitting a regularized linear model like the ridge regression is to not penalize the intercept term. Why is this the case?

2. Below we write the ridge objective function without a penalty on the intercept term:

$$\mathcal{L}_{ridge} = \sum_{n=1}^{N}(y_n - \mathbf{w}^\top \mathbf{x}_n)^2 + \lambda \sum_{d=2}^{D} w_d^2$$

   How can we express this purely with matrices/vectors?

   Hint: you'll want to define some $\mathbf{A}$ such that

$$\mathbf{w}^\top \mathbf{A}\mathbf{w} = \sum_{d=2}^{D} w_d^2$$

3. Now solve for $\hat{\mathbf{w}}$ (in terms of matrices/vectors) that minimizes the above loss function.

Solution:

1. Imagine the true data generating process has a very large intercept term. Then a ridge model may not be able to fit this data well because it would penalize large fitted values for the intercept.

2. Following the hint, we define

$$\mathbf{A} = \begin{bmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{I}_{D-1} \end{bmatrix}$$

   It follows that

$$\mathbf{w}^\top \mathbf{A}\mathbf{w} = \begin{bmatrix} 0 & \mathbf{w}_{2:D}^\top \end{bmatrix} \mathbf{w} = \sum_{d=2}^{D} w_d^2$$

   Thus, our loss function can be written as

$$\mathcal{L}_{ridge} = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^\top \mathbf{A}\mathbf{w}$$

3. We differentiate the loss from the previous part and set it equal to 0, then solve for $\mathbf{w}$:

$$\frac{\partial \mathcal{L}_{ridge}}{\partial \mathbf{w}} = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda \mathbf{A}\mathbf{w} = 0$$

$$\implies (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{A})\mathbf{w} = \mathbf{X}^\top \mathbf{y} \implies \hat{\mathbf{w}} = \boxed{(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{A})^{-1}\mathbf{X}^\top \mathbf{y}}$$

2.7 Geometric Intuition: Ridge vs. LASSO

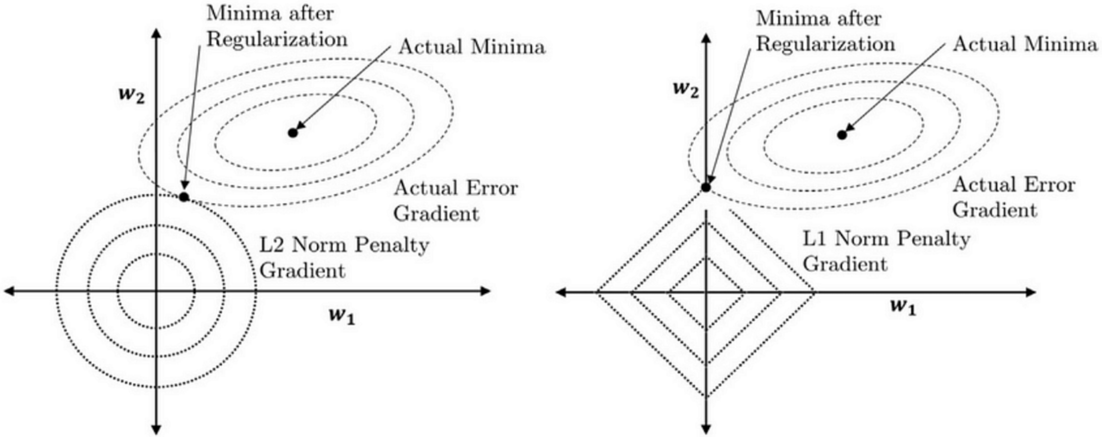

Figure 4: Geometric interpretation of L_2 (ridge) and L_1 (LASSO) regularization in two dimensions. Elliptical contours represent level sets of the unregularized least-squares loss. Left: The circular L_2 constraint leads to smooth shrinkage of coefficients. Right: The diamond-shaped L_1 constraint has corners on the coordinate axes, increasing the likelihood of sparse solutions.

Both ridge regression and LASSO can be interpreted geometrically as constrained optimization problems. In particular, minimizing a regularized loss is equivalent to minimizing the OLS loss subject to a constraint on the size of the weights:

$$\min_{\mathbf{w}} \ \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \quad \text{subject to} \quad \|\mathbf{w}\|_p \leq c,$$

where $p = 2$ corresponds to ridge regression and $p = 1$ corresponds to LASSO.

In two dimensions, the L_2 constraint region is a circle, while the L_1 constraint region is a diamond with corners aligned to the coordinate axes. The contours of the OLS loss (i.e. the lines that show where the parameters are equally optimal) are ellipses centered at the unconstrained least-squares solution.

For ridge regression, the circular constraint typically intersects the loss contours at a point where both coordinates are nonzero, leading to smooth shrinkage of all coefficients. But with lasso regression the sharp corners of the L_1 constraint are the furthest away from the origin, so this increases the probability that the optimum occurs exactly on an axis, which is exactly where one parameter has been set to zero. This effect is even more pronounced in higher dimensions.

This geometric difference explains why ridge regression encourages small but nonzero coefficients, while LASSO naturally performs variable selection, i.e. setting certain parameters to zero.

3 Gradient Descent

3.1 Motivation: Why Gradient Descent?

Most machine learning training problems can be written as minimizing a loss function over model parameters. While some special cases (e.g. ordinary least squares) admit closed-form solutions, in general we cannot solve for the minimizer analytically. This is especially true in modern ML, where models may have millions or billions of parameters and training sets can be extremely large. As a result, training is typically performed via *iterative optimization*, making repeated small parameter updates that reduce the loss. A great video explanation for gradient descent is this one by StatQuest.

3.2 Learning as Optimization (Empirical Risk Minimization)

In supervised learning we choose parameters θ to minimize the *training loss* (empirical risk):

$$\min_{\theta} \mathcal{L}(\theta) \quad \text{where} \quad \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i).$$

Here, f_θ is the model, $\ell(\cdot, \cdot)$ is a per-example loss, and $\mathcal{L}(\theta)$ defines a surface (a "loss landscape") over parameter space. Optimization can be viewed as moving "downhill" on this landscape. The geometry of the landscape (smoothness, curvature, convexity, and the presence of saddle points) strongly affects how hard optimization is.

3.3 Core Idea: Follow the Slopes

The gradient $\nabla_\theta \mathcal{L}(\theta)$ points in the direction of steepest *increase* of the loss. Therefore, to decrease the loss we should move in the opposite direction, the direction of steepest *descent*:

$$-\nabla_\theta \mathcal{L}(\theta).$$

This motivates gradient descent: an iterative algorithm that repeatedly takes small steps downhill.

3.4 Gradient Descent Algorithm

Gradient descent proceeds by initializing parameters and repeatedly updating them using the gradient of the loss:

$$\theta_0 \text{ given}, \qquad \theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t),$$

where $\eta > 0$ is the *step size* (also called the *learning rate*).

Step size / learning rate. The learning rate controls how large each update is. If η is too small, progress can be very slow. If η is too large, the updates can overshoot low-loss regions, causing oscillation or even divergence. A practical takeaway is that being slower but stable is often preferable to taking overly aggressive steps.

3.5 Batch, Stochastic, and Mini-Batch Gradient Descent

So far, the update uses the gradient of the full training loss $\mathcal{L}(\theta)$, which requires processing all N training examples each step. This is called *batch* (or *full-batch*) gradient descent. It is stable and deterministic given the initialization, but can be computationally expensive for large datasets.

Stochastic gradient descent (SGD). Instead of using all data each step, SGD uses a single randomly sampled training example (x_i, y_i) to form a stochastic estimate of the gradient:

$$\theta_{t+1} \;=\; \theta_t \;-\; \eta\,\nabla_\theta\,\ell(f_{\theta_t}(x_i),\, y_i)\,.$$

This update is cheaper but noisy; runs can follow different trajectories even with the same initialization. The noise can sometimes help optimization, e.g. by preventing stagnation near saddle points.

Mini-batch gradient descent. Mini-batch methods use a small subset (batch) B of size $|B|$ each iteration:

$$\theta_{t+1} \;=\; \theta_t \;-\; \eta\,\frac{1}{|B|}\sum_{i\in B}\nabla_\theta\,\ell(f_{\theta_t}(x_i),\, y_i)\,.$$

This balances the stability of batch GD with the scalability of SGD, and is the default choice in most modern deep learning systems (enabling efficient vectorization on GPUs/TPUs).

3.6 Optimizer Variants

Different optimizers use the same underlying objective and gradients but change the update dynamics.

Vanilla gradient descent.

$$\theta_{t+1} \;=\; \theta_t \;-\; \eta\, g_t \quad \text{where} \quad g_t = \nabla_\theta \mathcal{L}(\theta_t).$$

Momentum. Momentum uses a running average of gradients instead of just the gradient of the loss function at the current step:

$$m_t \;=\; \beta m_{t-1} + (1-\beta) g_t, \qquad \theta_{t+1} \;=\; \theta_t - \eta\, m_t.$$

Adam. Adam combines momentum with adaptive per-parameter scaling, which rescales the size of each parameter's individual update.

$$m_t \;=\; \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad v_t \;=\; \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \qquad \theta_{t+1} \;=\; \theta_t - \eta\,\frac{m_t}{\sqrt{v_t}+\epsilon}.$$

3.6.1 More information on Adam.

Adam augments gradient descent by maintaining *two running statistics for each parameter*: a first-moment estimate (mean of gradients) and a second-moment estimate (mean of squared gradients). Let $g_t = \nabla_\theta \mathcal{L}(\theta_t)$ denote the gradient at iteration t.

- **First moment (m_t).**

  $$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

  This is an exponentially weighted moving average of past gradients. It acts like *momentum*, smoothing noisy gradients and accumulating consistent directions of descent.

- **Second moment (v_t).**

  $$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

  This is an exponentially weighted moving average of squared gradients, computed elementwise. It estimates the typical *magnitude* (or variability) of the gradient for each parameter.

- The parameter update is then given by

  $$\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}.$$

- How the update works. The update divides the (momentum-smoothed) gradient m_t by $\sqrt{v_t}$, producing a *per-parameter rescaling*:

  – If a parameter consistently receives *large gradients*, then v_t will be large, shrinking the effective step size.

  – If a parameter receives *small or infrequent gradients*, then v_t will be small, increasing the effective step size.

  Thus, Adam assigns each parameter its own adaptive learning rate, while still following the overall direction of descent.

- **Role of ϵ.** The constant $\epsilon > 0$ prevents division by zero and improves numerical stability when v_t is very small.

3.7 Exercise: Comparing Optimizer Updates

Let the gradient at iteration t be

$$g_t = \begin{bmatrix} 4 \\ 1 \end{bmatrix},$$

and suppose the running second-moment estimate used by Adam is

$$v_t = \begin{bmatrix} 16 \\ 1 \end{bmatrix}.$$

Assume no momentum (i.e. $m_t = g_t$), ignore bias correction, and let $\epsilon = 0$.

1. Write the update step for **vanilla gradient descent**.
2. Write the update step for **Adam**.
3. Compare the sizes of the updates applied to the two coordinates, for both Vanilla GD and Adam.

4. Optimization behavior and geometry. Based on the update directions you computed above, explain how *vanilla gradient descent* and *Adam* would behave differently on a loss surface with highly unequal curvature across coordinates (like an elongated, narrow valley). In particular, address:

   (a) Which optimizer is more sensitive to the scale of the gradients?

   (b) Which optimizer is likely to make faster progress along the narrow direction of the valley, and why?

Solution:

1. **Vanilla gradient descent.** The update rule is

$$\theta_{t+1} = \theta_t - \eta g_t = \theta_t - \eta \begin{bmatrix} 4 \\ 1 \end{bmatrix}.$$

   So the update vector is

$$\Delta\theta_t^{(GD)} := \theta_{t+1} - \theta_t = -\eta \begin{bmatrix} 4 \\ 1 \end{bmatrix}.$$

2. **Adam (under the stated simplifications).** With $m_t = g_t$ and $\epsilon = 0$, the update rule becomes

$$\theta_{t+1} = \theta_t - \eta \frac{m_t}{\sqrt{v_t} + \epsilon} = \theta_t - \eta \frac{g_t}{\sqrt{v_t}}.$$

   Here $\sqrt{v_t}$ is taken elementwise:

$$\sqrt{v_t} = \begin{bmatrix} \sqrt{16} \\ \sqrt{1} \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}.$$

   Therefore,

$$\frac{g_t}{\sqrt{v_t}} = \begin{bmatrix} 4/4 \\ 1/1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

   and so

$$\theta_{t+1} = \theta_t - \eta \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

   The update vector is

$$\Delta\theta_t^{(Adam)} := \theta_{t+1} - \theta_t = -\eta \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

3. **Comparison / interpretation.** For vanilla GD, the magnitude of the update in coordinate 1 is 4 times the magnitude of the update in coordinate 2 (since it is proportional to $(4, 1)$). For Adam, the two coordinates receive updates of equal magnitude (proportional to $(1, 1)$), because the update is rescaled by $\sqrt{v_t}$ elementwise.

   This illustrates **adaptive per-parameter scaling**: parameters with consistently larger gradients (hence larger v_t) get their effective step sizes reduced, while parameters with smaller gradients get relatively larger effective step sizes.

4. **Solution.** (a) Vanilla gradient descent is sensitive to the scale of the gradients, since its update is directly proportional to the raw gradient vector. In this example, the first coordinate has a gradient four times larger than the second, so vanilla gradient descent takes steps that are four times larger in that direction. On a loss surface with highly unequal curvature, this can lead to oscillations in steep directions. We would need a small learning rate for stable convergence.

   (b) Adam, on the other hand, rescales each coordinate of the gradient by an estimate of its second moment. As a result, coordinates with consistently larger gradients receive smaller effective step sizes, while coordinates with smaller gradients receive relatively larger step sizes. In this example, this normalization leads to updates of equal magnitude across coordinates (as we saw).

   Consequently, Adam is less sensitive to curvature and is able to make more balanced progress along narrow or flat directions of the loss landscape. This typically allows Adam to converge faster than vanilla gradient descent on ill-conditioned optimization problems.