CSE 676 Deep Learning

The Past and Continuation of Our Journey

Jue Guo[1]

University at Buffalo

[1] The material is adapted from *Dive into Deep Learning*

Contents

(Review) Linear Neural Network for Regression I

- The model:

$$y = \mathbf{w}^\top \mathbf{x} + b$$

- Combine the weight and bias into a single vector:

$$y = \mathbf{w}^\top \mathbf{x}$$

- Minimize the mean squared error between the model prediction and the ground truth:

$$L(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)} \right)^2$$

(Review) Linear Neural Network for Regression II

- Let's rewrite it in the matrix form:

$$\mathbf{X}\mathbf{w} = \begin{bmatrix} \mathbf{x}^{(1)T}\mathbf{w} \\ \mathbf{x}^{(2)T}\mathbf{w} \\ \vdots \\ \mathbf{x}^{(n)T}\mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{w}^\top\mathbf{x}^{(1)} \\ \mathbf{w}^\top\mathbf{x}^{(2)} \\ \vdots \\ \mathbf{w}^\top\mathbf{x}^{(n)} \end{bmatrix}$$

$$L(\mathbf{w}) = \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

(Review) Analytical Solution: Step by Step I

- Loss Function

$$L(\mathbf{w}) = \frac{1}{n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- Take the derivative:

$$\frac{dL}{d\mathbf{w}} = 2(\mathbf{X}\mathbf{w} - \mathbf{y}) \cdot \frac{d}{d\mathbf{w}}(\mathbf{X}\mathbf{w} - \mathbf{y})$$

$$\frac{dL}{d\mathbf{w}} = 2(\mathbf{X}\mathbf{w} - \mathbf{y})\mathbf{X}$$

- Then, rewrite,

$$\frac{dL}{d\mathbf{w}} = 2\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$$

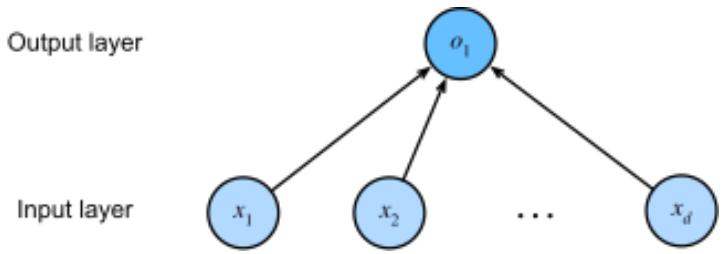
- Hence,

$$\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\mathbf{X}\mathbf{w}$$

$$\mathbf{w}^* = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

Linear Regression: A Really Shallow Network

- We can view neural network as a single-layer neural network:



- The inputs are $x_1, \ldots, x_d$. We refer to d as the number of inputs or feature dimensionality in the input layer. The output of the network is o_1.
- Because we are just trying to predict a single numerical value, we have only one output neuron. Note that the input values are all given. There is just a single computed neuron. In summary, we can think of linear regression as a single-layer fully connected neural network.

Generalization: An Overview

- As machine learning scientists, the goal is to discover patterns.
  - How can we be sure that we have truly discovered a general pattern and not simply memorized our data?
  - The problem-how to discover patterns that *generalize*-is the fundamental problem of machine learning.
- Some concepts:
  - The phenomenon of fitting closer to our training data than to the underlying distribution is called *overfitting*.
  - The techniques for combating overfitting are often called *regularization* methods.

Training Error and Generalization Error I

- Let's differentiate:
  training error:

$$R_{\text{emp}}[\mathbf{X}, \mathbf{y}, f] = \frac{1}{n} \sum_{i=1}^{n} I\left(\mathbf{x}^{(i)}, y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

  generalization error:

$$R[p, f] = E_{(\mathbf{x},y)\sim P}[I(\mathbf{x}, y, f(\mathbf{x}))] = \iint I(\mathbf{x}, y, f(\mathbf{x}))p(\mathbf{x}, y)d\mathbf{x}dy$$

Understanding Training and Generalization Error

- In the standard supervised learning setting, we assume that the training data and test data are drawn *independently* from *identical* distributions.
- Even such assumption is robust,
  - Why should we believe that training data sampled from *distribution* $P(X, Y)$ should tell us how to make predictions on test data generated by a *different distribution* $Q(X, Y)$?
  - Making such leaps turns out to require strong assumptions about how P and Q are related.
- Later, we will discuss some assumptions that allow for shifts in distribution but first we need to understand the IID case, where $P(\cdot) = Q(\cdot)$.

Traininig Error

- As stated before, training error R_{emp},

$$R_{\text{emp}}[\mathbf{X}, \mathbf{y}, f] = \frac{1}{n} \sum_{i=1}^{n} I\left(\mathbf{x}^{(i)}, y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

  - R_{emp}, which is a *statistic* calculated on the training dataset.
  - $\mathbf{x}^{(i)}$: training data
  - $y^{(i)}$: corresponding label
  - $f\left(\mathbf{x}^{(i)}\right)$: model prediction

Generalization Error

- As stated before, generalization error

$$R[p, f] = E_{(\mathbf{x},y)\sim P}[l(\mathbf{x}, y, f(\mathbf{x}))] = \iint l(\mathbf{x}, y, f(\mathbf{x}))p(\mathbf{x}, y)d\mathbf{x}dy$$

  - R, which is an expectation taken with respect to the underlying distribution.

Training Error and Generalization Error: Conclusion

- We can never calculate the generalization error:
  - No idea of the precise form of the density function $p(\mathbf{x}, y)$
  - we can not sample an infinite stream of data points
- In practice, we will evaluate our classifier on the test set that is withheld from the training set. Now the problem become simply mean estimation.

Model Complexity

- In classical theory, when we have simple models and abundant data, the training and generalization error tend to be close.
- In most cases, you will find yourself working with more complex models and/or fewer examples. Now you will find that training error go down, but the generalization gap to grow.
- We will revisit this concept in the future, but something to bear in mind:
  1. low training error does not necessarily imply low generalization error
  2. low training error does not imply high generalization error either
- In most case, we must rely more heavily on our holdout data to certify generalization after the fact. Error on the holdout data, i.e., validation set, is called the *validation error*.

A Good Habit I

- **Training Dataset**: This is the data on which the model is trained and learned. The training set forms the basis of the model's learning process. The model sees this data and adjusts its parameters (like weights and biases in the case of neural networks) to minimize the loss function.
- **Validation Dataset**: The validation set is used during the model's training process to assess the performance of the model on unseen data and to tune model hyperparameters (such as learning rate, number of layers in a neural network, etc.). The validation set provides a 'practice arena' for the model before the final test. It is used to prevent overfitting, which occurs when the model learns the training data too well and performs poorly on unseen data.

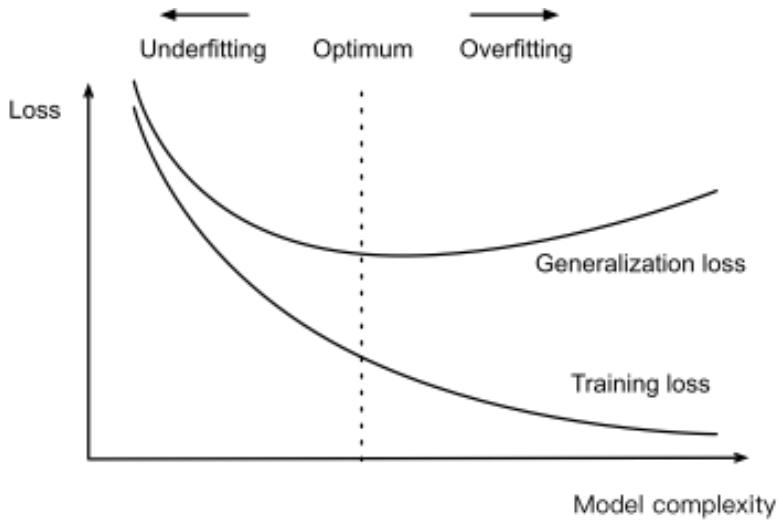
A Good Habit II

- **Testing Dataset**: After training and validation, the final assessment of the model is done on the test dataset. This data is kept separate and is not used during the training or validation processes. The performance on the test set gives us a final, unbiased estimate of our model's ability to generalize to new, unseen data.

Underfitting or Overfitting?

- When we compare the training and validation errors, we want to be mindful of two common situations.
  1. **Underfitting**: We want to watch out for cases when our training error and validation error are both substantial but there is a little gap between them. If the model is unable to reduce the training error, that could mean that our model is too simple to capture the pattern that we are trying to model. Moreover, since the *generalization gap* ($R_{\text{emp}} - R$) between our training and generalization errors is small, we have reason to believe that we could get away with a more complex model.
  2. **Overfitting**: When training error is significantly lower than our validation error. Overfitting is not necessarily a bad thing.

Visual Representation of Underfitting and Overfitting



Weight Decay: An Overview

- Now that we have characterized the problem of overfitting, we can introduce our first regularization technique.
- We can always mitigate overfitting by collecting more training data. However, that can be costly, time consuming, or entirely out of our control.

Norms and Weight Decay I

- Rather than directly manipulating the number of parameters, *weight decay*, operates by restricting the values that the parameters can take.
  Remember, the loss function for linear regression?

$$L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)} \right)^2$$

L1 vs L2 Regularisation

- L1 Regularisation (Lasso Regression)

$$\text{Loss} = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

- L2 Regularisation (Ridge Regression)

$$\text{Loss} = \text{Error}(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

L1: Take the Derivative and Gain Some Intuition

$$
\begin{aligned}
w_{\text{new}} &= w - \eta \frac{\partial L_1}{\partial w} \\
&= w - \eta \cdot \left[2x(wx + b - y) + \lambda \frac{d|w|}{dw} \right] \\
&= \begin{cases} w - \eta \cdot [2x(wx + b - y) + \lambda] & w > 0 \\ w - \eta \cdot [2x(wx + b - y) - \lambda] & w < 0 \end{cases}
\end{aligned}
$$

- **L1 Regularization**:
  - Adds a penalty term proportional to the absolute value of the weights.
  - The derivative of the absolute value function has a discontinuity at 0, which makes the update step dependent on the sign of the weight.
    - When updating the weights, a constant amount ($\eta\lambda$) is subtracted or added to the weight, which can lead to the weight reaching exactly zero, especially for small weights.
  - This is why L1 regularization can lead to sparse solutions, where some of the weights are exactly zero.

L1: An Intuition from Math

- Lasso shrinks the less important feature's coefficient to zero; thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features.

L2: Take the Derivative and Gain Some Intuition

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_2}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y) + 2\lambda w] \end{aligned}$$

- **L2 Regularization**:
  - In the given equation, the regularization term is λw^2. The derivative of this term is $2\lambda w$, which scales linearly with the weight.
  - When updating the weights, a portion of the weight itself ($2\eta\lambda w$) is subtracted. This tends to make the weights small but not exactly zero, because the amount subtracted gets smaller as the weights get closer to zero.

How do they combat over-fitting?

- **L1 Regularization** (Lasso regularization): This adds a penalty term that is proportional to the absolute value of the coefficients (the L1 norm of the weights). The effect of this is that L1 regularization tends to produce sparse solutions, pushing some weights to be exactly zero. This effectively reduces the number of features in the model, which can help prevent overfitting.
- **L2 Regularization** (Ridge regularization): This adds a penalty term that is proportional to the square of the magnitude of the coefficients (the L2 norm of the weights). This encourages the model to distribute the weight values more evenly across all input features, leading to smaller weights. Smaller weights result in a simpler model (in terms of the function it can represent), which can help prevent overfitting. Unlike L1 regularization, L2 regularization does not lead to sparse solutions and does not reduce the number of features used by the model.

Linear Neural Networks for Classification

- **Finished**: all of the mechanics that are needed to solve a machine learning problem.
- Even as we pivot towards classification, most of the plumbing remains the same:
  - Loading the data
  - Passing it through the model
  - Generating output
  - Calculating the loss
  - Taking gradients with respect to weights
  - Updating the model
- What changes?
  - the precise form of the targets
  - the parameterization of the output layer
  - the choice of loss function

Softmax Regression: An Introduction

- Regression: *how much?* or *how many?*
- Classification: *which category?*
  - Does this email belong in the spam folder or the inbox?
  - Is this customer more likely to sign up or not to sign up for a subscription service?
  - Does this image depict a donkey, a dog, a cat, or a rooster?
  - and more ...

Classification

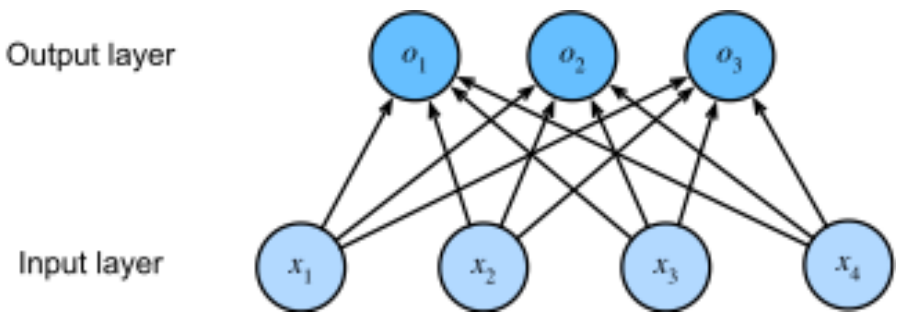
- Input: 2×2 grayscale image.
- Category: cat, chicken, dog
- One-hot Encoding: $y \in \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$

Linear Model

- To estimate the conditional probability associated with all possible classes, we need a model with multiple outputs, one per class.

$$
\begin{aligned}
o_1 &= x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1 \\
o_2 &= x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2 \\
o_3 &= x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3
\end{aligned}
$$

- A visual representation:



The Softmax I

- Rewrite the bunch of equations:

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

- Let's say, we find a suitable loss function to minimize the difference between between $\mathbf{o}$ and the labels $\mathbf{y}$.
  - There is no guarantee that the outputs o_i sum up to 1 in the way we expect probabilities to behave.
  - There is no guarantee that the outputs o_i are even nonnegative, even if their outputs sum up to 1 , or that they do not exceed 1 .
- We need to find a mechanism to "squish" the outputs. Let's borrow some idea from physics (1902):

  "prevalence of a state of energy in a thermodynamic ensemble"

The Softmax II

- To accomplish this goal, and to ensure nonnegativuity is with the help of an exponential function.

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$$

  - conditional class probability increases with increasing o_i it is monotonic, and all probabilities are nonnegative
  - the normalization process helps as it transform the values so that they add up to 1
- Note that the largest coordinate of o corresponds to the most likely class according to $\hat{\mathbf{y}}$. Moreover, because the softmax operation preserves the ordering among its arguments, we do not need to compute the softmax to determine which class has been assigned the highest probability.

$$\underset{j}{\text{argmax}} \hat{y}_j = \underset{j}{\text{argmax}} o_j.$$

Loss Function

- Now that we have a mapping from features $\mathbf{x}$ to probabilities $\hat{\mathbf{y}}$, we need a way to optimize the accuracy of this mapping.
- Before we move on, let's provide some reasoning of mean squared error loss from a probabilistic perspective:
  - the optimal parameters return the conditional expectation $E[Y \mid X]$ whenever the underlying pattern is truly linear, and the loss assigns outsize penalties for outliers.
  - We can also provide a more formal motivation for the squared loss objective by making probabilistic assumptions about the distribution of noise.

The Normal Distribution and Squared Loss I

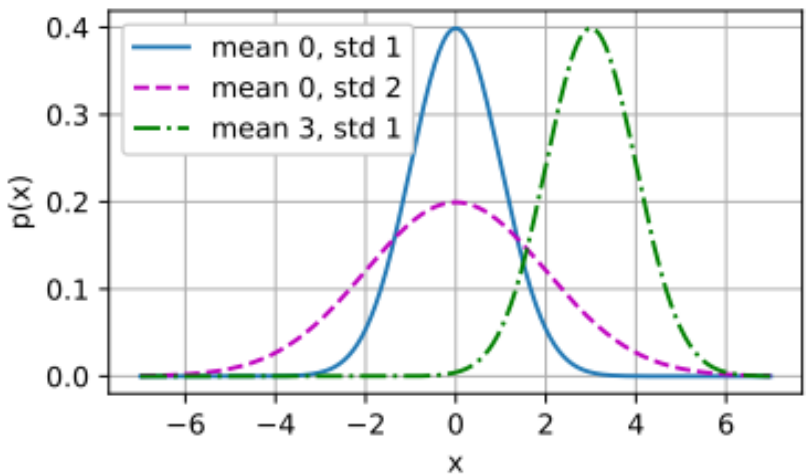
- First, something really amazing: Video
- Linear regression was invented at the turn of the 19th century. Gauss discovered the normal (Gaussian) distribution.
- Normal distribution and linear regression with squared loss share a deeper connection than common parentage.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$$

```
def normal(x, mu, sigma):
    p = 1 / math.sqrt(2 * math.pi * sigma**2)
    return p * np.exp(-0.5 * (x - mu)**2 / ↩
        sigma**2)
```

The Normal Distribution and Squared Loss II

- Visualize it:



  - changing mean corresponds to a shift along the x-axis.
  - increasing the variance spreads the distribution out, lowering its peak.

The Normal Distribution and Squared Loss III

- One way to motivate linear regression with squared loss is to assume that observations arise from noisy measurements, where the noise is normally distributed as follows:

$$y = \mathbf{w}^\top \mathbf{x} + b + \epsilon \text{ where } \epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$$

- Therefore, we can write out the *likelihood* of seeing a particular y for a given $\mathbf{x}$ via

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(y - \mathbf{w}^\top \mathbf{x} - b\right)^2\right)$$

The Normal Distribution and Squared Loss IV

- According to the *principle of maximum likelihood*, the best values of parameters $\mathbf{w}$ and b are those that maximize the likelihood of the entire dataset:

$$P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p\left(y^{(i)} \mid \mathbf{x}^{(i)}\right)$$

- Let's dive into a bit more to help with understanding.
  - This statement is based on the assumption that each data point in your dataset is independently and identically distributed (i.i.d.). This is a common assumption in many statistical models, including linear regression.

Independent and Identically Distributed

- Independent and identically distributed (i.i.d.) is a term used in statistics to describe a collection of random variables.
  - "*Independent*" means that the occurrence of one event does not affect the probability of occurrence of the other events. In the context of data, this implies that each data point does not influence or is not influenced by any other data point.
  - "*Identically distributed*" means that all the random variables in the collection have the same probability distribution. In the context of data, this implies that each data point is generated from the same underlying probability distribution.
- An example of i.i.d. data is the result of repeated tosses of a fair coin. Each coin toss is independent of the others (the outcome of one toss doesn't affect the outcomes of the other tosses), and each toss is identically distributed (the probability of heads or tails is the same for each toss).

Continue the Mind Map

- Under these assumptions, the total likelihood of observing all the data given the model parameters is the product of the likelihoods of observing each individual data point given the model parameters. This is a consequence of the definition of independent events in probability theory: the probability of independent events occurring together is the product of their individual probabilities.

Continue where we left

- Find values of $\mathbf{w}$ and b that maximize the *likelihood* of the entire dataset:

$$P(\mathbf{y} \mid \mathbf{X}) = \prod_{i=1}^{n} p\left(y^{(i)} \mid \mathbf{x}^{(i)}\right)$$

  Estimators chosen according to the principle of maximum likelihood are called *maximum likelihood estimators*.
- Now the question arises, how do we solve

$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left(y - \mathbf{w}^\top \mathbf{x} - b\right)^2\right)$$

Solution

- Take the natural logarithm of both side, we get,

$$\log P(y \mid \mathbf{x}) = -\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2$$

- For the whole dataset, rewrite, *log-likelihood*

$$\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n}\left[-\frac{1}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2\right]$$

- Optimization tend to minimize not maximize, therefore, rewrite again,

$$-\log P(\mathbf{y} \mid \mathbf{X}) = \sum_{i=1}^{n}\frac{1}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\left(y^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)} - b\right)^2$$

Some Observation and Final Justification

- If we assume that σ is fixed, we can ignore the first term, because it does not depend on w or b.
- The second term is identical to the squared error loss introduced earlier, except for the multiplicative constant $\frac{1}{\sigma^2}$.
- Fortunately, the solution does not depend on σ either. It follows that minimizing the mean squared error is equivalent to maximum likelihood estimation of a linear model under the assumption of additive Gaussian noise.

Loss Function: Log-Likelihood I

- The softmax function gives us a vector $\hat{\mathbf{y}}$, which we can interpret as (estimated) conditional probabilities of each class, given any input $\mathbf{x}$, such as $\hat{y}_1 = P(y = \text{cat} \mid \mathbf{x})$. In the following we assume that for a dataset with features $\mathbf{X}$, the labels $\mathbf{Y}$ are represented using a one-hot encoding label vector. We can compare the estimates with reality by checking how probable the actual classes are according to our model, given the features:

$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P\left(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}\right)$$

Loss Function: Log-Likelihood II

- As reasoned before, we can write equations as such:

$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P\left(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}\right) = \sum_{i=1}^{n} l\left(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}\right)$$

  - where for any pair of label $\mathbf{y}$ and model prediction $\hat{\mathbf{y}}$ over q classes, the loss function l (*cross-entropy loss*) is

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j$$

Softmax and Cross-Entropy Loss I

- Plug in:

$$
\begin{aligned}
l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} \\
&= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \\
&= \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j
\end{aligned}
$$

- Understand a bit better:

$$
\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \text{softmax}(\mathbf{o})_j - y_j
$$

Softmax and Cross-Entropt Loss II

- Let's look at the equation and gain some intuition:

$$\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \text{softmax}(\mathbf{o})_j - y_j$$

  - the derivative is the difference between the probability assigned by our model, as expressed by the softmax operation, and what actually happened, as expressed by elements in the one-hot label vector.

Information Theory Basic: Introduction

- This is a survival guide to navigate through deep learning papers and documentations.
- To make sense of them, we need some common language. *Information theory* deals with the problem of encoding, decoding, transmitting, and manipulating information (also known as data).

Information Theory Basic: Entropy

- The central idea in information theory is to quantify the amount of information contained in data.
- This places a limit on our ability to compress data. For a distribution P its entropy is defined as:

$$H[P] = \sum_j -P(j) \log P(j)$$

- One of the fundamental theorems of information theory states that in order to encode data drawn randomly from the distribution P, we need at least $H[P]$ "nats" to encode it.

Information Theory Basic: Surprisal

- You might wondering what compression has to do with prediction. Imagine that we have a stream of data we want to compress. If it is always easy for us to predict the next token, then this data is easy to compress.
- If we can not perfectly predict every event, then we might sometimes be surprised.
  - Our surprise is greater when we assigned an event lower probability.
  - Claude Shannon settled on $\log \frac{1}{P(j)} = -\log P(j)$ to quantify one's surprisal at observing an event j having assigned it a (subjective) probability $P(j)$.
  - The entropy:

$$H[P] = \sum_j -P(j) \log P(j)$$

    is then the *expected surprisal* when one assigned the correct probabilities that truly match the data-generating process.

Information Theory Basic: Cross-Entropy Revisited

- So if entropy is the level of surprise experienced by someone who knows the true probability, then you might be wondering, what is cross-entropy?
  - The cross-entropy from P to Q, denoted $H(P, Q)$, is the expected surprisal of an observer with subjective probabilities Q upon seeing data that was actually generated according to probabilities P.
- This is given by $H(P, Q) \stackrel{\text{def}}{=} \sum_j -P(j) \log Q(j)$. The lowest possible cross-entropy is achieved when $P = Q$.
  - In this case, the cross-entropy from P to Q is $H(P, P) = H(P)$.
- In short, we can think of the cross-entropy classification objective in two ways: (i) as maximizing the likelihood of the observed data; and (ii) as minimizing our surprisal (and thus the number of bits) required to communicate the labels.

The Image Classification Dataset

- Some representational dataset:
  - MNIST
  - ImageNet

Softmax Regression Implementation: Softmax I

- Let's begin with the most important part: the mapping form scalars to probabilities.
- Given a matrix X we can sum over all elements (by default) or only over elements in the same axis.

```
X = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
X.sum(0, keepdims=True), X.sum(1, keepdims=True)
```

```
(tensor([[5., 7., 9.]]),
 tensor([[ 6.],
         [15.]]))
```

Softmax Regression Implementation: Softmax II

- Computing the softmax requires three steps: (i) exponentiation of each term; (ii) a sum over each row to compute the normalization constant for each example; (iii) division of each row by its normalization constant, ensuring that the result sums to 1.

$$\text{softmax}(\mathbf{X})_{ij} = \frac{\exp(\mathbf{X}_{ij})}{\sum_k \exp(\mathbf{X}_{ik})}$$

```
def softmax(X):
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition  # The broadcasting mechanism is applied here
```

Softmax Regression Implementation: Softmax III

```
X = torch.rand((2, 5))
X_prob = softmax(X)
X_prob, X_prob.sum(1)
```

```
(tensor([[0.1560, 0.2128, 0.2260, 0.2372, 0.1680],
         [0.1504, 0.2473, 0.1132, 0.2779, 0.2112]]),
 tensor([1.0000, 1.0000]))
```

Softmax Regression Implementation: The Model I

- **Some ground work**: We now have everything that we need to implement the softmax regression model. As in our linear regression example, each instance will be represented by a fixed-length vector. Since the raw data here consists of 28×28 pixel images, we flatten each image, treating them as vectors of length 784 .
- In softmax regression, the number of outputs from our network should be equal to the number of classes. Since our dataset has 10 classes, our network has an output dimension of 10 . Consequently, our weights constitute a 784×10 matrix plus a 1×10 dimensional row vector for the biases. As with linear regression, we initialize the weights W with Gaussian noise. The biases are initialized as zeros.

Softmax Regression Implementation: The Model II

```
class SoftmaxRegressionScratch(...):
    def __init__(self, num_inputs, num_outputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W = torch.normal(0, sigma, size=(num_inputs, num_outputs),
                              requires_grad=True)
        self.b = torch.zeros(num_outputs, requires_grad=True)

    def parameters(self):
        return [self.W, self.b]
```

Softmax Regression Implementation: The Model III

- The code below defines how the network maps each input to an output. Note that we flatten each 28×28 pixel image in the batch into a vector using `reshape` before passing the data through our model.

```
def forward(self, X):
    X = X.reshape((-1, self.W.shape[0]))
    return softmax(torch.matmul(X, self.W) + ↩
        self.b)
```

Softmax Regression Implementation: The Cross-Entropy Loss I

- Now, let's implement one of the most common loss function in classification problem.

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

```
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()
```

Generalization in Classification I

- So far we have focused on how to tackle multiclass classification problems by training (linear) neural networks with multiple outputs and softmax functions.
- Interpreting our model's outputs as probabilistic predictions, we motivated and derived the cross-entropy loss function, which calculates the negative log likelihood that our model (for a fixed set of parameters) assigns to the actual labels.

Generalization in Classification II

- However, as always, our goal is to learn general patterns, as assessed empirically on previously unseen data (the test set). High accuracy on the training set means nothing.
- Why? Whenever each of our inputs is unique (and indeed this is true for most high-dimensional datasets), we can attain perfect accuracy on the training set by just memorizing the dataset on the first training epoch, and subsequently looking up the label whenever we see a new image. And yet, memorizing the exact labels associated with the exact training examples does not tell us how to classify new examples.

Some Thought-Provoking Questions

- Some burning questions that demand immediate attention:
  - How many test examples do we need to precisely estimate the accuracy of our classifiers on the underlying population?
  - What happens if we keep evaluating models on the same test repeatedly?
  - Why should we expect that fitting our linear models to the training set should fare any better than our naive memorization scheme?
- We covered overfitting and generalization in the context of linear regression, but now we will go more indepth with the some foundational ideas of statistical learning theory.

Generalization in Classification: The Test Set

- We have already begun to rely on test sets as the gold standard method for assessing generalization error, let's get started by discussing the properties of such error estimates.
- The *empirical error*:

$$\epsilon_{\mathcal{D}}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\left(f\left(\mathbf{x}^{(i)}\right) \neq y^{(i)}\right)$$

- The *population error*:

$$\epsilon(f) = E_{(\mathbf{x},y)\sim P}\mathbf{1}(f(\mathbf{x}) \neq y) = \iint \mathbf{1}(f(\mathbf{x}) \neq y)p(\mathbf{x}, y)d\mathbf{x}dy$$

Optimization: An Introduction

- What does optimization do?
  - Continue updating model parameters
  - Minimize the value of loss function (evaluating on which dataset?)
- We can not treat optimization as a black box device to minimize objective function.
  - A deeper knowledge is required. Faster optimization is equivalent to faster training time.
  - *Convex* to *Non-convex*

Optimization and Deep Learning

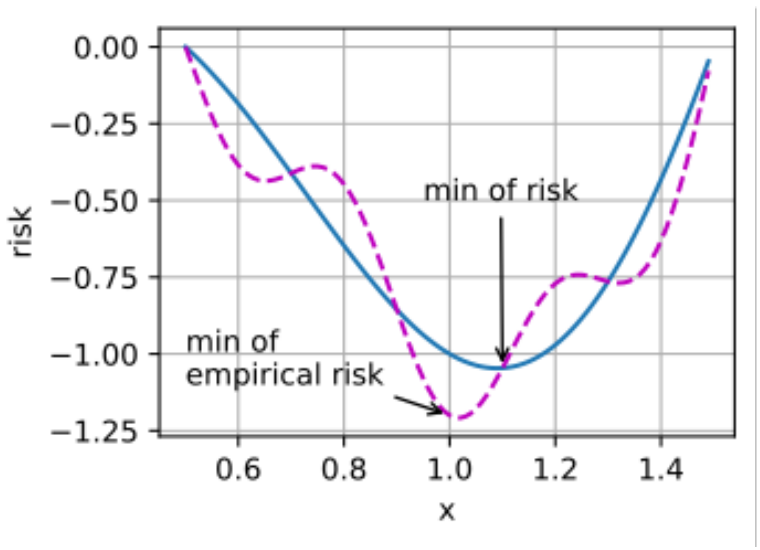
- A quick review (An analogy of Horizon 4):
  - Data (environmental data)
  - Model (your supercar)
  - Loss Function (*objective function of the optimization problem*) (the map of your choice)
  - Optimization Algorithm (v6 or v8 engine)
  - Metrics (How fast per lap)

Goal of Optimization

- The goals of optimization and deep learning are fundamentally different (**finite amount of data**):
  - *Optimization* is concerned with minimizing an objective.
  - *Deep Learning* is concerned with finding a suitable model.
- Difference:
  - The objective function of the optimization algorithm is usually a loss function based on the training dataset, the goal of the optimization is to reduce the training error.
  - The goal of deep learning is to reduce the generalization error. To accomplish this, we need to pay attention to overfitting in addition to use the optimization algorithm to reduce the training error.

Visualization

- To better understand the difference, we can see that the minimum of the empirical risk on a training dataset may be at a different location from the minimum of the risk (*generalization error*).



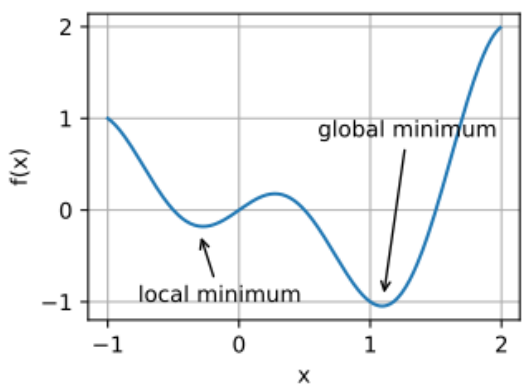
Optimization Challenges in Deep Learning

- Local Minima
- Saddle Points
- Vanishing Gradient

Local Minima I

- Definitions:
  - *For any objective function $f(x)$, if the value of $f(x)$ at x is smaller than the values of $f(\cdot)$ at any other points in the vicinity of x, then $f(x)$ could be a **local minimum**. If the value of $f(x)$ at x is the minimum of the objective function over the entire domain, then $f(x)$ is the **global minimum**.*
- Given the function:

$$f(x) = x \cdot \cos(\pi x) \text{ for } -1.0 \leq x \leq 2.0$$



Local Minima II

- **Objective Function Characteristics:**
  - The objective function of deep learning models has many local optima.
  - Near a local optimum, the numerical solution may only achieve local minimization.
  - The gradient of the objective function's solutions can approach or become zero.
- **Challenges with Local Optima:**
  - Only certain noise levels can displace the parameter from a local minimum.
- **Benefits of Minibatch Stochastic Gradient Descent:**
  - Natural variation of gradients across minibatches can move parameters away from local minima.
  - This property helps in avoiding local minimization.

Saddle Points

- **Saddle Points:**
  - A saddle point is any location where all gradients of a function vanish.
  - It is neither a global nor a local minimum.
- **Example:**
  - Consider the function $f(x) = x^3$.
  - First derivative: $f'(x) = 3x^2$.
  - Second derivative: $f''(x) = 6x$.
  - Both derivatives vanish for $x = 0$.
- **Implication:**
  - Optimization might stall at saddle points, even if they are not minima.

Mathematical Refresh

- If $f''(x) > 0$ at a point where $f'(x) = 0$, then $f(x)$ has a *local minimum* at that point.
- If $f''(x) < 0$ at a point where $f'(x) = 0$, then $f(x)$ has a *local maximum* at that point.
- If $f''(x) = 0$ at a point where $f'(x) = 0$, the test is inconclusive, and the point could be a *saddle point*, a flat region, or an inflection point.

Saddle Points at 1d and k-d

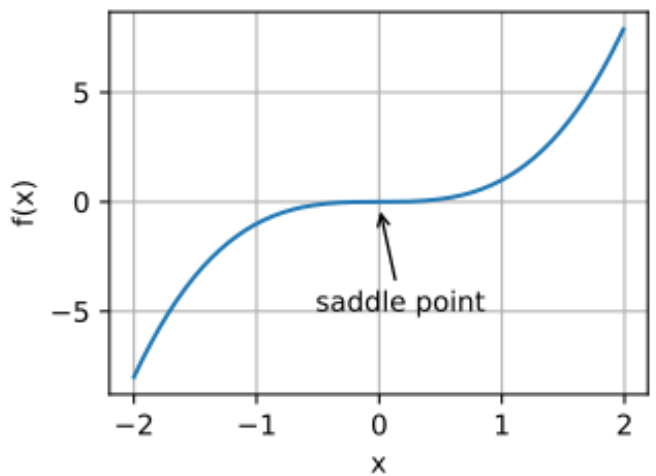


Figure: 1-D saddle points

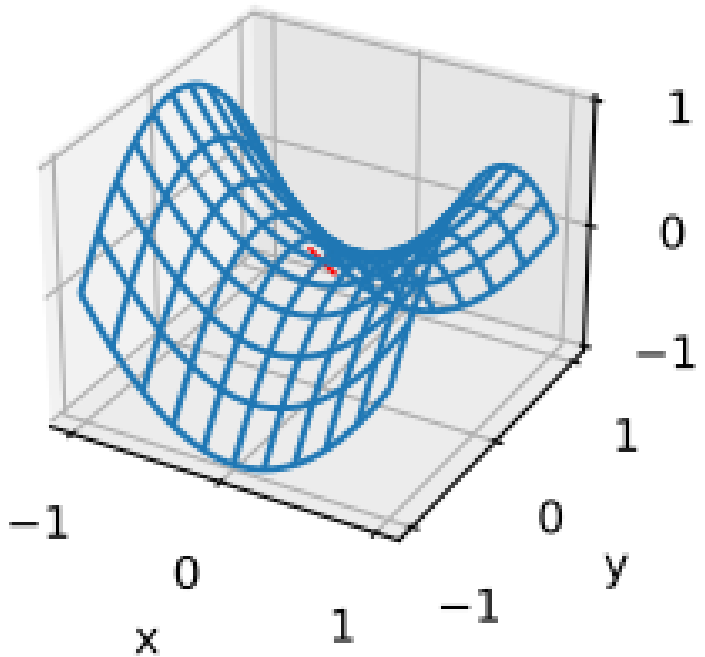


Figure: k-D saddle Points

Introduction to Hessian Matrix and Eigenvalues

- **Function Assumption:**
  - We consider a function with input as a k-dimensional vector.
  - The output of this function is a scalar.
- **Hessian Matrix:**
  - It's a $k \times k$ square matrix.
  - Contains second-order mixed partial derivatives.
  - Captures the local curvature of a function.
- **Eigenvalues:**
  - The Hessian matrix, being $k \times k$, will have k eigenvalues.
  - Eigenvalues provide insights into the curvature and nature of critical points of the function.

Determining Nature of Critical Points

- Critical points occur where the gradient of the function is zero.
- The nature of these critical points (minima, maxima, saddle) can be determined using the eigenvalues of the Hessian matrix:
  - **Local Minimum:** All eigenvalues are positive.
  - **Local Maximum:** All eigenvalues are negative.
  - **Saddle Point:** A mix of positive and negative eigenvalues.
- The eigenvalues give insights into the curvature of the function in different directions.

Saddle Points in High-Dimensional Spaces

- **High-Dimensional Complexity:**
  - With increasing dimensions, the optimization landscape of functions becomes more intricate.
  - There are more directions in which the function can exhibit curvature.
- **Eigenvalues and Curvature:**
  - Positive eigenvalues indicate upward curvature, while negative ones indicate downward curvature.
  - In high-dimensional spaces, it's more probable to have a mix of both positive and negative eigenvalues.
- **Implication for Optimization:**
  - This mixed curvature leads to the prevalence of **saddle points**, where the function curves up in some directions and down in others.
  - Saddle points are typically more common than pure local minima or maxima in high-dimensional spaces.

Convexity: A Special Case

- Convex functions have a property where the eigenvalues of the Hessian are **always non-negative**.
- This means they don't have local maxima or saddle points, only global minima.
- While most deep learning problems aren't convex, understanding convexity is crucial for optimization algorithms.

The Role of Convexity in Optimization Algorithms

- **Importance of Convexity:**
  - Convexity simplifies the analysis and testing of optimization algorithms.
  - If an algorithm struggles in the convex setting, it's unlikely to perform well in more complex scenarios.
- **Deep Learning and Convexity:**
  - While deep learning optimization problems are generally nonconvex, they can exhibit convex-like properties near local minima.
  - This observation paves the way for innovative optimization variants.

Convex Analysis: Definitions

Convex Analysis forms the foundation for many mathematical tools applied in machine learning. The core concepts include:

- Convex Sets
- Convex Functions (to be discussed later)

Convex Sets: Definition

- A set $\mathcal{X}$ in a vector space is termed **convex** if, for any two points $a, b \in \mathcal{X}$, the entire line segment connecting a and b also lies within $\mathcal{X}$.
- Mathematically, this is expressed as:

$$\lambda a + (1 - \lambda) b \in \mathcal{X} \text{ whenever } a, b \in \mathcal{X} \text{ and } \lambda \in [0, 1].$$

Visualizing Convex Sets

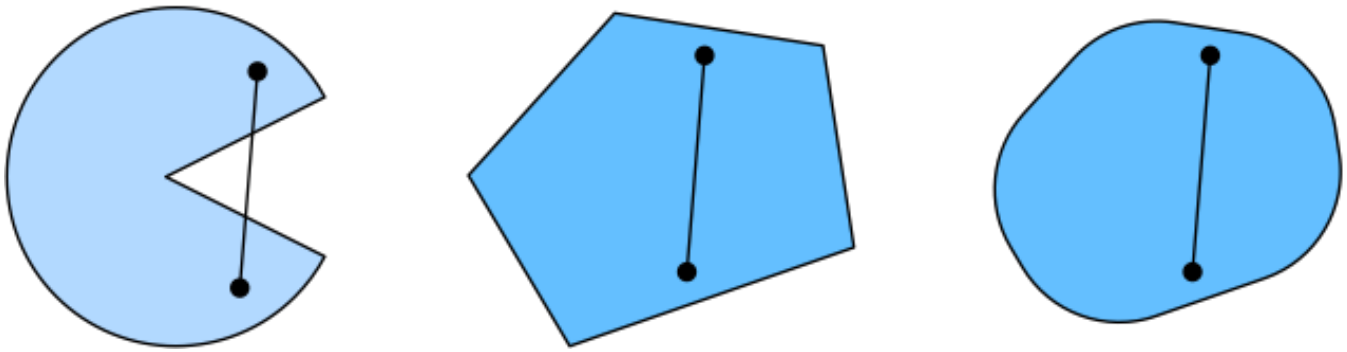


Figure: The first set is non-convex, while the latter two are convex.

A set is non-convex if there exists at least one line segment that doesn't entirely lie within the set.

Convex Sets: Practical Utility of Definitions

While definitions provide a foundation, their real utility is in the applications and properties they lead to. One such property is the **intersection** of convex sets.

Intersection of Convex Sets

- Let's consider two convex sets $\mathcal{X}$ and $\mathcal{Y}$.
- The intersection of these sets, denoted by $\mathcal{X} \cap \mathcal{Y}$, is also convex.

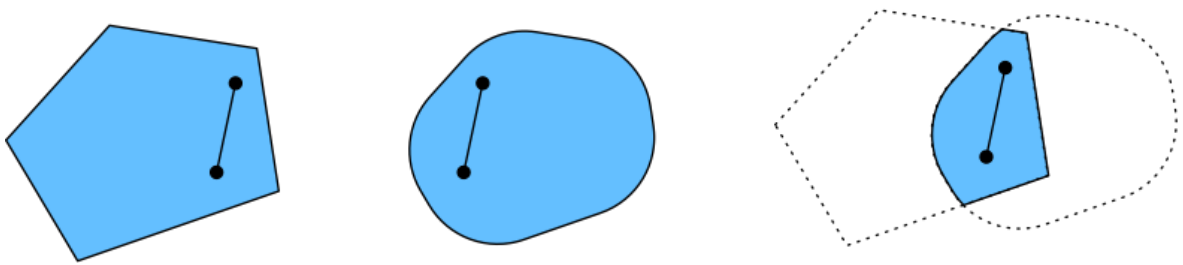


Figure: Intersection of two convex sets.

Proof of Intersection Convexity

- For any two points $a, b \in \mathcal{X} \cap \mathcal{Y}$:
  - The line segment connecting a and b lies within $\mathcal{X}$ since $\mathcal{X}$ is convex.
  - Similarly, the line segment also lies within $\mathcal{Y}$ since $\mathcal{Y}$ is convex.
- Therefore, the line segment connecting a and b must also lie within the intersection $\mathcal{X} \cap \mathcal{Y}$.
- This confirms that the intersection of convex sets is also convex.

Intersections and Unions of Convex Sets

We'll explore the properties of intersections and unions of convex sets, and understand when they retain or lose their convexity.

Intersection of Multiple Convex Sets

- Consider multiple convex sets $\mathcal{X}_i$.
- The intersection of these sets, denoted by $\cap_i \mathcal{X}_i$, is also convex.
- This is a generalization of the property that the intersection of two convex sets is convex.

Unions of Convex Sets

- Consider two disjoint sets where $\mathcal{X} \cap \mathcal{Y} = \emptyset$.
- Let $a \in \mathcal{X}$ and $b \in \mathcal{Y}$.

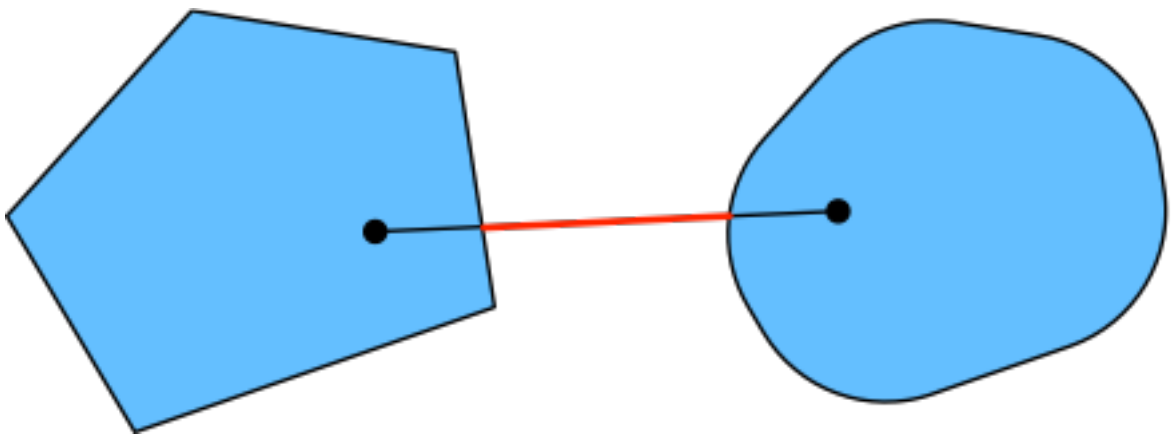


Figure: Line segment connecting a and b.

Non-Convexity of Unions

- The line segment connecting a and b contains parts not in $\mathcal{X}$ or $\mathcal{Y}$ since $\mathcal{X} \cap \mathcal{Y} = \emptyset$.
- Therefore, the line segment isn't entirely within $\mathcal{X} \cup \mathcal{Y}$.
- This demonstrates that the union of convex sets is not necessarily convex.

Convex Functions

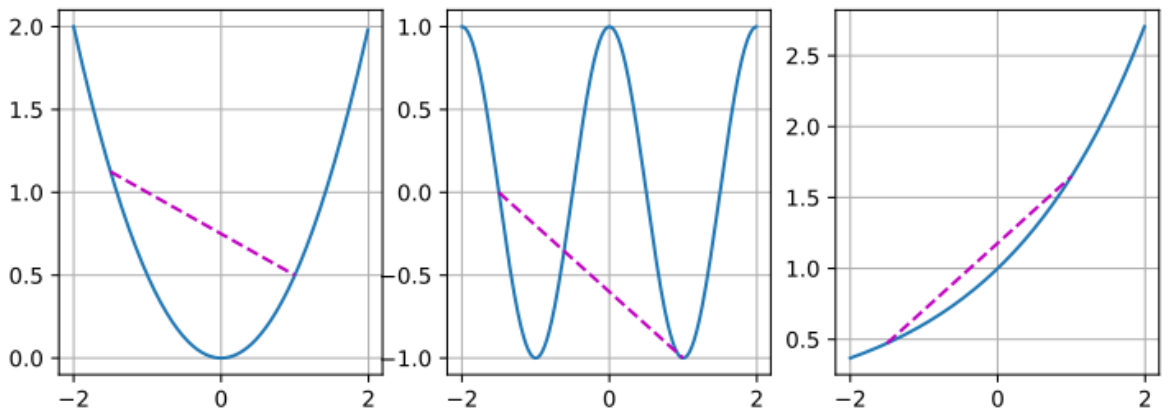
Definition

Given a convex set $\mathcal{X}$, a function $f : \mathcal{X} \to \mathbb{R}$ is termed **convex** if, for all $x, x' \in \mathcal{X}$ and for all $\lambda \in [0, 1]$, the following inequality holds:

$$\lambda f(x) + (1 - \lambda) f(x') \geq f(\lambda x + (1 - \lambda) x')$$

This definition can be visualized by plotting functions and checking which ones satisfy the above condition. Both convex and nonconvex functions can be illustrated for clarity.

Visual Representation of Convexity



Convexity of Functions

- The **cosine function** is nonconvex.
- In contrast, both the **parabola** and the **exponential function** are convex.

It's crucial to note:

- The requirement that $\mathcal{X}$ be a convex set is essential.
- Without this, the expression $f\left(\lambda x + (1-\lambda)x'\right)$ might not be well-defined.

Jensen's Inequality

Jensen's Inequality is a fundamental result for convex functions and provides a powerful tool for simplifying and bounding expressions.

Definition

Given a convex function f, Jensen's Inequality states:

$$\sum_i \alpha_i f(x_i) \geq f\left(\sum_i \alpha_i x_i\right)$$

and

$$E_X[f(X)] \geq f(E_X[X])$$

where α_i are nonnegative real numbers such that $\sum_i \alpha_i = 1$ and X is a random variable.

Interpretation of Jensen's Inequality

In simpler terms, the expectation of a convex function is no less than the convex function of an expectation. The latter is often a more straightforward expression.

- The inequality can be proven by repeatedly applying the definition of convexity to one term in the sum at a time.

Applications of Jensen's Inequality

Jensen's Inequality is frequently used to bound complex expressions by simpler ones.

Example

Consider the log-likelihood of partially observed random variables:

$$E_{Y \sim P(Y)}[-\log P(X \mid Y)] \geq -\log P(X)$$

where $\int P(Y)P(X \mid Y)dY = P(X)$. This is commonly used in variational methods.

Useful Properties of Convex Function

- Local Minima Are Global Minima
- Below Sets of Convex Functions Are Convex
- Convexity and Second Derivatives

Introduction to Below Sets

Context: Understanding the convexity of below sets of convex functions.

- A convex function f is defined on a convex set $\mathcal{X}$.
- The "below set" $\mathcal{S}_b$ captures all points in $\mathcal{X}$ where the function value is below a threshold b.
- Formally:

$$\mathcal{S}_b \stackrel{\text{def}}{=} \{x \mid x \in \mathcal{X} \text{ and } f(x) \leq b\}$$

Convexity of Below Sets

Claim: The below set $\mathcal{S}_b$ is convex.
Proof Idea:

- Take any two points $x, x' \in \mathcal{S}_b$.
- For these points, the function values are both below b: $f(x) \leq b$ and $f(x') \leq b$.
- We want to show that any point on the line segment between x and x' also lies in $\mathcal{S}_b$.

Using Convexity for Proof

For any combination $\lambda x + (1 - \lambda)x'$ where $\lambda \in [0, 1]$:
By the definition of convexity:

$$f\left(\lambda x + (1 - \lambda)x'\right) \leq \lambda f(x) + (1 - \lambda) f\left(x'\right)$$

Given our initial conditions:

$$\lambda f(x) + (1 - \lambda) f\left(x'\right) \leq b$$

Combining the two inequalities, we conclude:

$$f\left(\lambda x + (1 - \lambda)x'\right) \leq b$$

Thus, $\lambda x + (1 - \lambda)x' \in \mathcal{S}_b$, proving $\mathcal{S}_b$ is convex.

Convexity and Second Derivative

Context: Understanding the relationship between convexity and the second derivative.

- For a function $f : \mathbb{R}^n \to \mathbb{R}$, convexity can be checked using its second derivative.
- The Hessian matrix, denoted by $\nabla^2 f$, captures the second derivatives.
- Convexity requires: $\nabla^2 f \succeq 0$. (*positive semi-definite*)

One-dimensional Case: Introduction

Context: Investigating the relationship between convexity and the second derivative in one dimension.

- A function $f : \mathbb{R} \to \mathbb{R}$ is said to be convex if it curves upwards.
- The curvature can be determined using the second derivative.
- Key question: What does the second derivative tell us about convexity?

Convexity Implies $f'' \geq 0$

If f is convex, then for any two points on the function:

$$\frac{1}{2}f(x+\epsilon) + \frac{1}{2}f(x-\epsilon) \geq f\left(\frac{x+\epsilon}{2} + \frac{x-\epsilon}{2}\right) = f(x)$$

Taking the limit as $\epsilon \to 0$:

$$f''(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) + f(x-\epsilon) - 2f(x)}{\epsilon^2} \geq 0$$

This means the function is curving upwards everywhere.

$f'' \geq 0$ Implies Convexity

If the second derivative is non-negative, f' is monotonically non-decreasing.
For any three points $a < x < b$, *mean value theorem*:

$$f'(\alpha) = \frac{f(x) - f(a)}{x - a}$$

$$f'(\beta) = \frac{f(b) - f(x)}{b - x}$$

By monotonicity:

$$\lambda f(b) + (1 - \lambda) f(a) \geq f((1 - \lambda)a + \lambda b)$$

This confirms the function is convex.

One-dimensional Case: Conclusion

- The second derivative provides a clear criterion for convexity in one dimension.
- If $f'' \geq 0$, the function is convex.
- This property is fundamental for understanding the shape and behavior of functions.

Multidimensional Case: Introduction

Context: Investigating the relationship between convexity and the Hessian in multiple dimensions.

- A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be convex if it curves upwards in all directions.
- The curvature can be determined using the Hessian matrix, $\nabla^2 f$.
- Key question: What does the Hessian tell us about convexity in multiple dimensions?

Multidimensional Case: Hessian and Convexity

A multidimensional function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if:

$$\nabla^2 f \succeq 0$$

Insight:

- The Hessian matrix captures the curvature of the function in all directions.
- A positive semidefinite Hessian indicates the function is curving upwards.

Lemma on Convexity

Lemma: For a function $f : \mathbb{R}^n \to \mathbb{R}$, it is convex if and only if the function:

$$g(z) = f(z\mathbf{x} + (1 - z)\mathbf{y})$$

is convex for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $z \in [0, 1]$.

Insight:

- This lemma provides a way to test the convexity of f by examining the convexity of g.
- It essentially reduces the multidimensional problem of f to a one-dimensional problem of g.

Lemma on Convexity: Proof

Proof:

- Convexity of f implies convexity of g: If f is convex, then for any line segment in $\mathbb{R}^n$, the function value at the midpoint is less than the average of the function values at the endpoints.
- Convexity of g implies convexity of f: If g is convex, then f must also be convex since g is derived from f.

Proof of Convexity for $g(z)$

Goal: Prove $g(z) = f(z\mathbf{x} + (1-z)\mathbf{y})$ is convex if f is convex.

1. **Setup**: For convexity of g, show for all $a, b, \lambda \in [0,1]$:
$$\lambda g(a) + (1-\lambda)g(b) \geq g(\lambda a + (1-\lambda)b)$$
2. **Expand** g:
$$g(\lambda a + (1-\lambda)b) = f((\lambda a + (1-\lambda)b)\mathbf{x} + (1 - \lambda a - (1-\lambda)b)\mathbf{y})$$
3. **Rearrange**:
$$= f(\lambda(a\mathbf{x} + (1-a)\mathbf{y}) + (1-\lambda)(b\mathbf{x} + (1-b)\mathbf{y}))$$
4. **Use f's Convexity**:
$$\lambda f(a\mathbf{x} + (1-a)\mathbf{y}) + (1-\lambda)f(b\mathbf{x} + (1-b)\mathbf{y})$$
$$\geq f(\lambda(a\mathbf{x} + (1-a)\mathbf{y}) + (1-\lambda)(b\mathbf{x} + (1-b)\mathbf{y}))$$
5. **Relate to** g:
$$\Rightarrow \lambda g(a) + (1-\lambda)g(b) \geq g(\lambda a + (1-\lambda)b)$$

Conclusion: $g(z)$ is convex if f is convex.

Proof of Convexity for f using g

Goal: Prove f is convex if g is convex.

$$\begin{aligned} & f(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \\ & = g(\lambda \cdot 1 + (1-\lambda) \cdot 0) \\ & \leq \lambda g(1) + (1-\lambda) g(0) \\ & = \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}) \end{aligned}$$

Conclusion: If $g(z) = f(z\mathbf{x} + (1-z)\mathbf{y})$ is convex, then f is also convex.