Linear Model

ITM528 Deep learning

Taemoon Jeong, Korea University

Linear Regression

Why Linear Models First?

- Let's forget about **deep** neural networks.
- Here, we consider a **shallow** neural network to focus on
  - the basics of neural network training
  - parametrizing the output layer
  - handling data
  - specifying a loss function
  - training the model

Linear Regression

- Suppose that we wish to estimate the prices of houses (in dollars) based on their area (in square feet) and age (in years).
- In machine learning terminology,
  - dataset: a **training dataset** or **training set**
  - each row, (area, age, price): an **example** (or **data point**, **instance**, **sample**)
  - price: a **label** (or **target**)
  - area and age: **features** (or **covariates**)

Model (1/3)

- We can model the price as a linear function of area and age:

$$\text{price} = w_{\text{area}} \cdot \text{area} + w_{\text{age}} \cdot \text{age} + b$$

Here w_{area} and w_{age} are called **weights**, and b is called a **bias** (or **offset**).

- The above equation is an **affine transformation** of input features, which is characterized by a linear transformation of features combined with a translation.

- Given a dataset, our goal is to choose the weights $\mathbf{w}$ and the bias b that make our model's predictions fit the true price observed in that data as closely as possible.

Model (2/3)

- When our inputs consist of d features, our prediction $\hat{y}$ becomes:

$$\hat{y} = w_1 x_1 + \ldots + w_d x_d + b$$

- Collecting all features into a vector $\mathbf{x} \in \mathbb{R}^d$ and all weights into a vector $\mathbf{w} \in \mathbb{R}^d$, we can express the model compactly via the dot product:

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

- The above equation is about a single instance and it is often convenient to refer to features of the entire dataset of n examples via the **design matrix** $\mathbf{X} \in \mathbb{R}^{n \times d}$:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

- where **broadcasting** is applied during the summation.

Model (3/3)

- We can express our model in a vector equation:

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + b$$

- Given the features of a training dataset $\mathbf{X}$ and corresponding labels $\mathbf{y}$, the goal of linear regression is to find the weight vector $\mathbf{w}$ and the bias b with the lowest prediction error.

- To this end, we need two things:
  1. a **quality measure** for some given model and data
  2. a procedure for **updating the model** to improve its quality

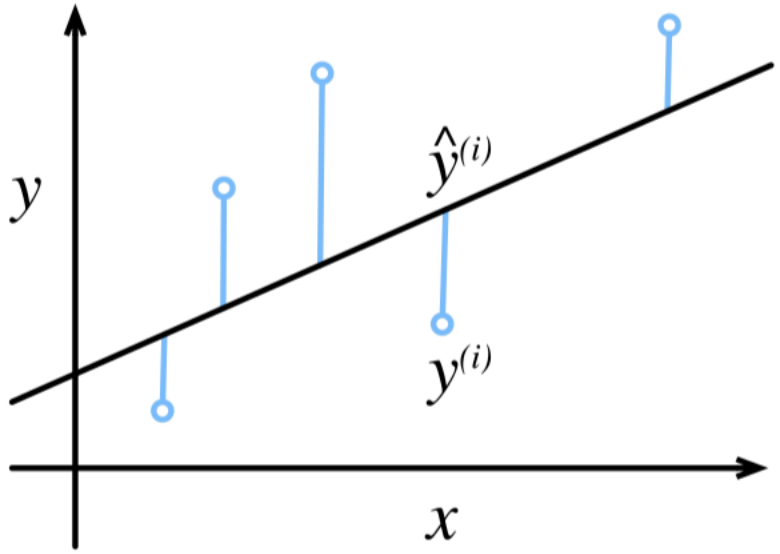
Loss Function (1/3)

- Fitting our model to the data requires that we agree on some measure of fitness and loss functions to quantify the distance between the real and predicted values of the target.

- For example, when our prediction for an i-th example is $\hat{y}^{(i)}$ and the corresponding true label is $y^{(i)}$, the squared error is given by:

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

Loss Function (2/3)



Loss Function (3/3)

- To measure the quality of a model on the entire dataset of n examples, we simply average (or sum) the losses on the training set:

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

- When training the model, we find the parameters $(\mathbf{w}^*, b^*)$ that minimize the total loss across all training examples:

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w}, b} L(\mathbf{w}, b)$$

Analytic Solution

- Unlike most of the models that we will cover, linear regression becomes a surprisingly easy optimization problem in that we can find the optimal parameters **analytically**.
- In particular, we aim to minimize

$$\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

- Hence, if the design matrix $\mathbf{X}$ has full rank and takes the derivative of the loss w.r.t. $\mathbf{w}$ and sets it to zero yields:

$$\partial_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

- Solving it for $\mathbf{w}$ provides the optimal solution:

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Stochastic Gradient Descent (1/2)

- While simple problems may admit analytic solutions, you should not get used to such good fortune.
- The key technique for optimizing nearly any deep learning model consists of iteratively reducing the error by updating the parameters in the direction that lowers the loss function, named **gradient descent**.
  - The naive application of gradient descent considers every single example in that dataset, but it can be extremely slow.
  - The other extreme is to consider only a single example at a time to take update steps, named **stochastic gradient descent** (SGD).
  - The intermediate strategy is to take a minibatch of observations, named **minibatch stochastic gradient descent**.

Stochastic Gradient Descent (2/2)

- The update rule of minibatch gradient descent becomes:

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \partial_{(\mathbf{w}),b} l^{(i)}(\mathbf{w}, b)$$

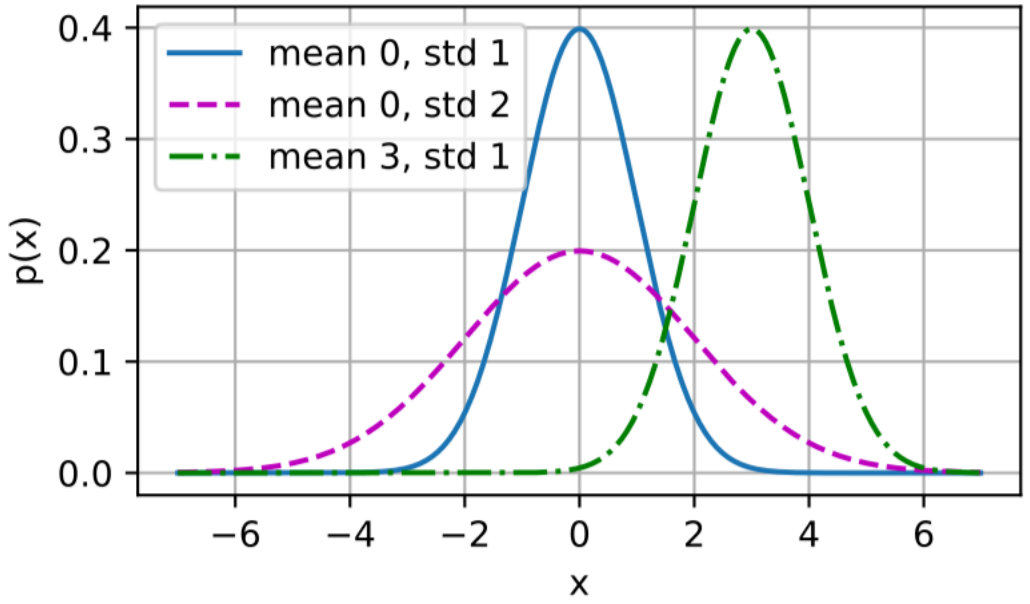
where $\mathcal{B}$ is a minibatch and η is the learning rate.

- In the end, the quality of the solution is typically assessed on a separate validation dataset.

Maximum Likelihood Learning (1/3)

- The normal distribution and linear regression with squared loss share deeper connections.
- The normal distribution with mean μ and variance σ^2 is:

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



Maximum Likelihood Learning (2/3)

- If we assume that the observations arise from noisy measurements:

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

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

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

- 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:

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

where we assume that all pairs $(\mathbf{x}^{(i)}, y^{(i)})$ were drawn independently.

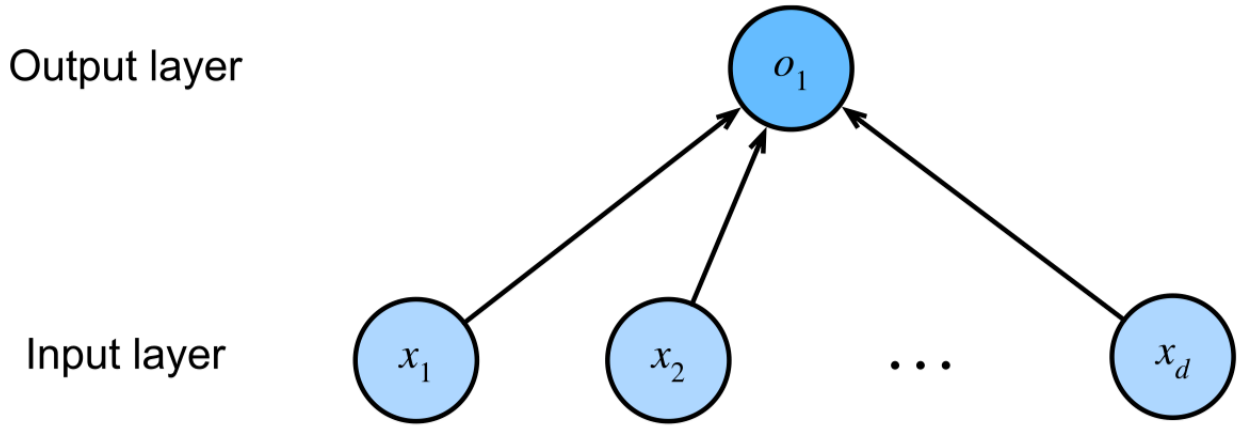
Maximum Likelihood Learning (3/3)

- Instead of maximizing the likelihood, we can minimize the negative log-likelihood, which we can express as follows:

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

- If we assume that σ is fixed, we can ignore the first term.
- In fact, as the solution does not depend on σ, minimizing the mean squared error is equivalent to the maximum likelihood estimation of a linear model under the assumption of additive Gaussian noise.

Linear Regression as a Neural Network

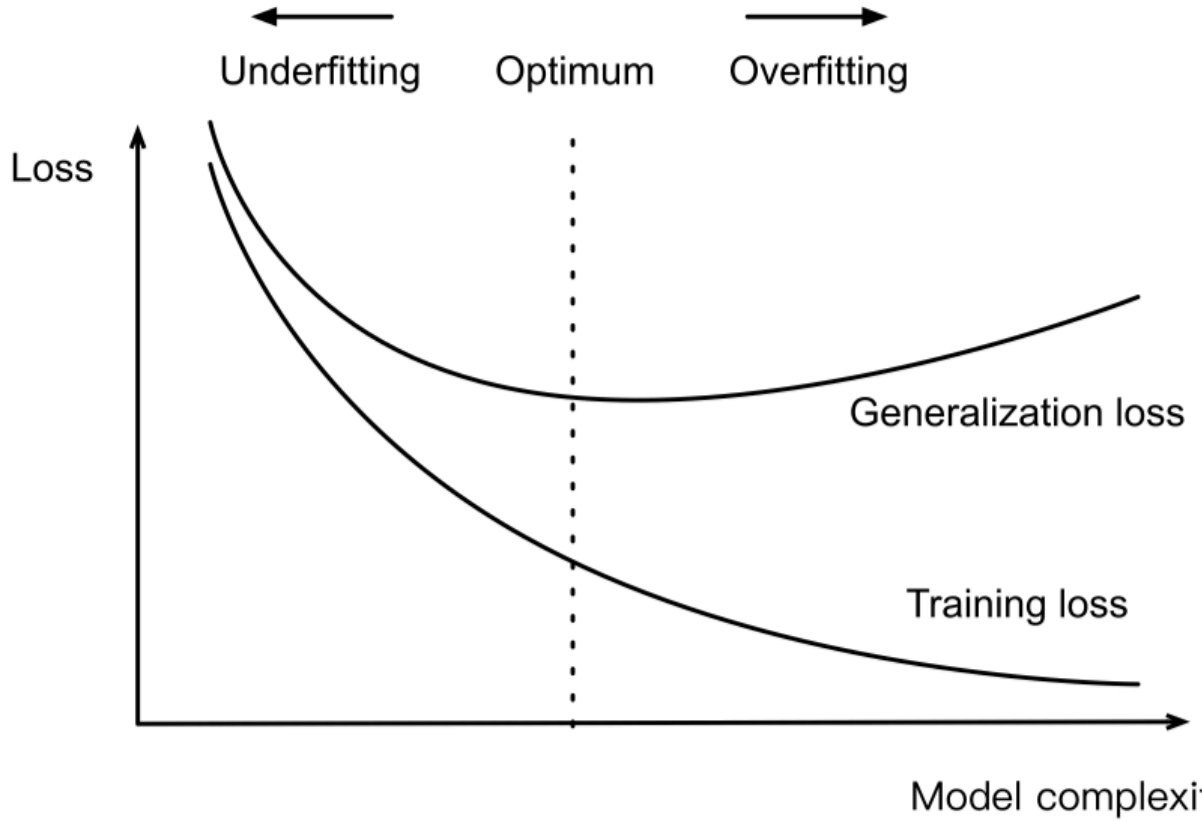


- Neural networks are rich enough to subsume linear models in which every feature is represented by an input neuron, all of which are connected directly to the output.
- The inputs are $x_1, \ldots, x_d$. We refer d as the **number of inputs** of **feature dimensionality** in the input layer.

Generalization

- Our goal is to discover patterns. But how can we be sure that we have truly discovered a **general pattern** and not simply memorized our data?
- We do not want to predict yesterday's stock prices, but tomorrow's!
- This problem of discovering patterns that **generalize** is the fundamental problem of machine learning and statistics.
- In real life, we must fit our models using a finite collection of data.
    - However, even if we have more than 100 million images (e.g., Flickr YFC100M), the number of available data points remains **infinitesimally small** compared to the space of all possible images at 1-megapixel resolution.
- Hence, we must keep in mind the risk that we might fit our training data, only to discover that we failed to discover a generalizable pattern.

Underfitting or Overfitting



- The phenomenon of fitting closer to our training data than to the underlying distribution of called **overfitting**, and techniques for combatting overfitting are often called **regularization** methods.

Training Error and Generalization Error

- In the standard supervised learning setting, we assume that the training data and the test data are drawn independently from identical distributions (aka the **IID assumption**).
- First of all, we need to differentiate between the empirical training error R_{emp} and the generalization error R:

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

and

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

- The central question of generalization is when should we expect our training error to be close to the generalization error.

Model Complexity (1/3)

- In classical learning theory, simpler models tend to have similar training and generalization (or test) errors (i.e., generalization gap to go down).
  - **Generalization gap**: a gap between training and test errors
  - **Occam's razor**: a problem-solving principle that recommends searching for explanations constructed with the smallest possible set of elements

[1] Vapnik et al., "Measuring the VC-Dimension of a Learning Machine," 1994

Model Complexity (2/3)

- In general, absent any restriction on our model class, we cannot conclude based on fitting the training data alone that our model has discovered any generalizable pattern [1].
  - On the other hand, if our model class was not capable of fitting arbitrary labels, then it must have discovered a pattern.
  - In short, what we want is a hypothesis that could not explain any observations we might conceivably make yet nevertheless happens to be compatible with those observations that we in fact make.

[1] Vapnik et al., "Measuring the VC-Dimension of a Learning Machine." 1994

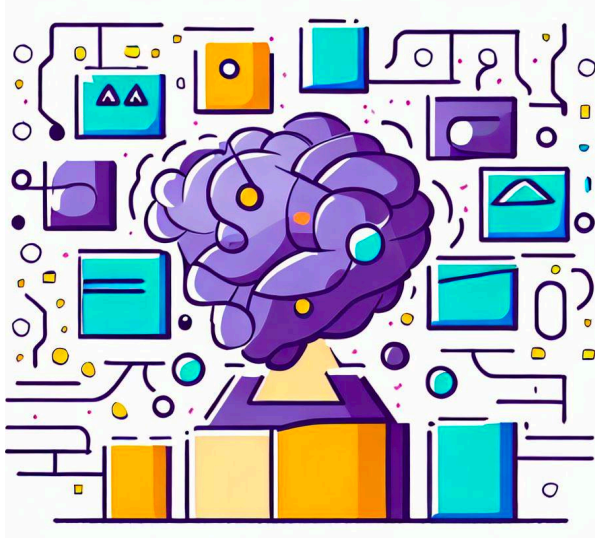
Model Complexity (3/3)

- In the context of training deep neural networks, when a model is capable of fitting arbitrary labels, **low training error** does not necessarily imply **low generalization error**.
- However, it does not necessarily imply **high generalization** error either!
- Hence, we must rely more heavily on our holdout data to certify generalization (i.e., check error on the validation set, validation error).

Dataset Size

- Another big consideration is dataset size.
- As we increase the amount of training data, the generalization gap typically decreases.
- In general, more data never hurts!

Model Selection

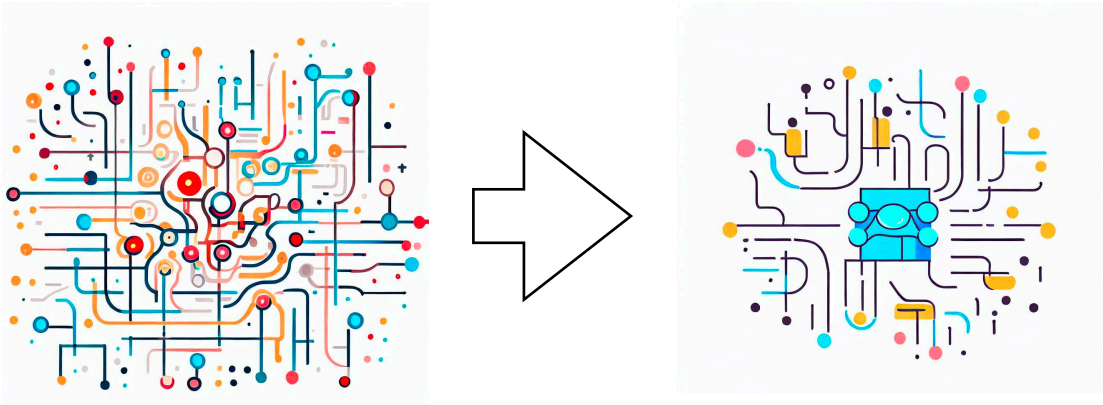


- Typically, we select our final model, only after evaluating multiple models that differ in various ways (different architectures, training objectives, selected features, learning rates, etc.).
- The common practice is to split our data in three ways, training, test, and validation sets, but it is a murky practice where the boundaries between validation and test datasets are worryingly ambiguous.

K-fold Validation

- K-fold validation is a technique to evaluate a machine learning model's performance by partitioning the original training set into k equal-sized subsets.
- One of the subsets is used as the validation set, where the remaining $k-1$ subsets are combined to form a training set.
- The process is repeated k times and the average performance across all k trials is used assess the model's overall performance.

Regularization



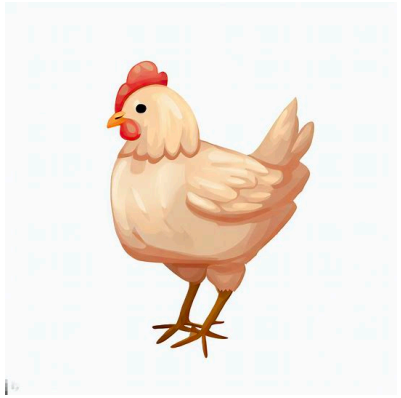
- **Regularization** is a common method for dealing with overfitting.
- Classical regularization methods add a penalty term to the loss function while training to reduce the complexity of the learned model.
  - **Weight decay**: adds a penalty proportional to the magnitude of the weights and biases
  - **Early stopping**: stops training before the model starts overfitting
  - **Dropout**: randomly sets a fraction of input units to zero

Linear Classification

Classification

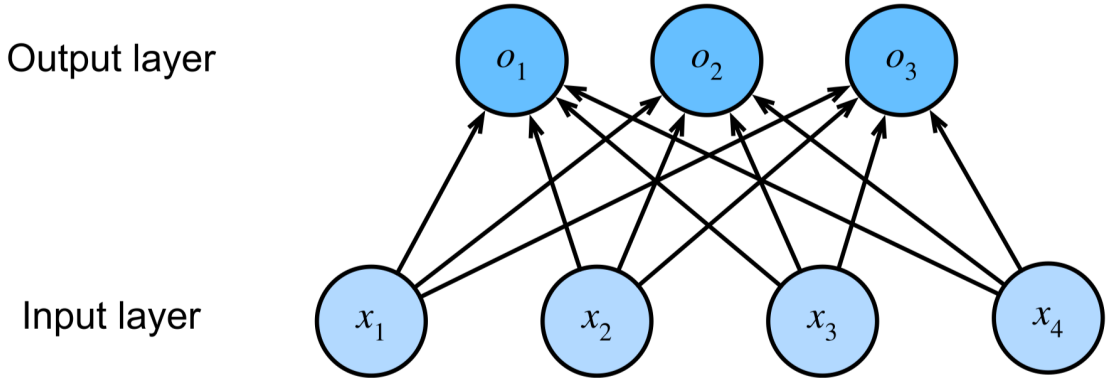
- Now, we pivot towards classification, but most of the plumbing remains the same:
  - loading the data, passing it through the model, generating output, calculating the loss, taking the gradient w.r.t. weights, and updating the model.
- The main difference between classification and regression comes from the parametrization of the output layer, and the choice of the loss function.

Labels


- Let's start with a simple image classification problem where each image belongs to either "cat", "chicken", or "dog".
- Next, we have to choose how to represent the labels.
  - Perhaps the most natural impulse would be to choose $y \in \{1,2,3\}$, where the integers represent $\{\text{dog}, \text{cat}, \text{chicken}\}$ respectively.
  - While this is a great way of storing such information on a computer, explicit ordering often hinders the performance.
  - In general, classification problems do not come with natural orderings among classes, and **one-hot encoding** is used to represent categorical data: $y \in \{(1,0,0), (0,1,0), (0,0,1)\}$.

Linear Model



- In order to estimate the categorical distribution, we need a model with multiple outputs, one per class.
- Suppose we have four features and three possible output categories:

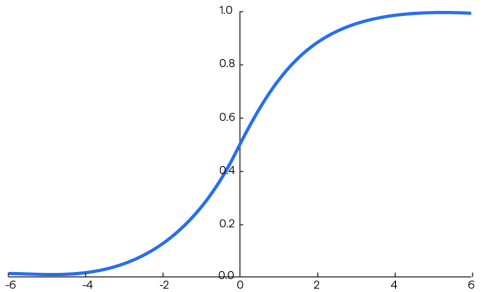
$$o_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1$$
$$o_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2$$
$$o_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3$$

- For a more concise notation, we use vectors and matrices: $\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$ where $W \in \mathbb{R}^{3\times4}$ and $b \in \mathbb{R}^3$.

Softmax (1/4)

- In the classification setting, we assume that the output follows a **categorical distribution** (i.e., nonnegative and sum up to one).
- However, the output of a neural network is simply a real-valued vector.
- Softmax utilizes an exponential function to model a categorical distribution, $P(y = i) \propto \exp(o_i)$:

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



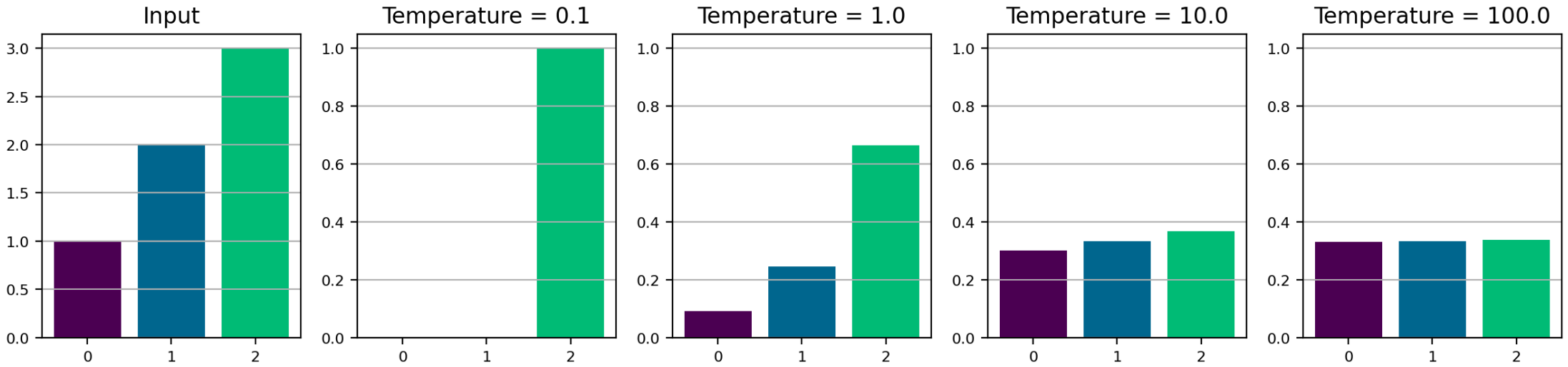
- One can also add a temperature here:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o}), \quad \text{where } \hat{y}_i = \frac{\exp(o_i/\tau)}{\sum_j \exp(o_j/\tau)} \text{ and } \tau \text{ is a temperature}$$

Softmax (2/4)

- Temperature is an important tuning parameter.

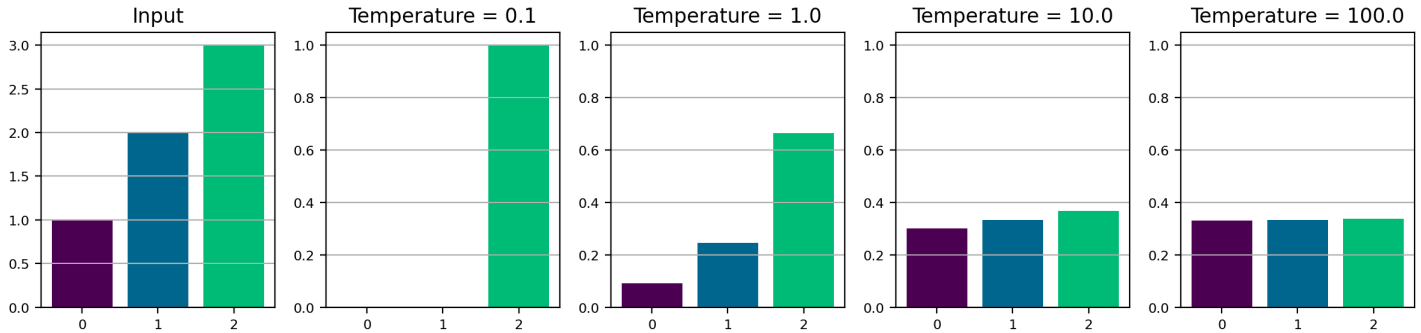
- Let's take a look at the effect of different temperatures:



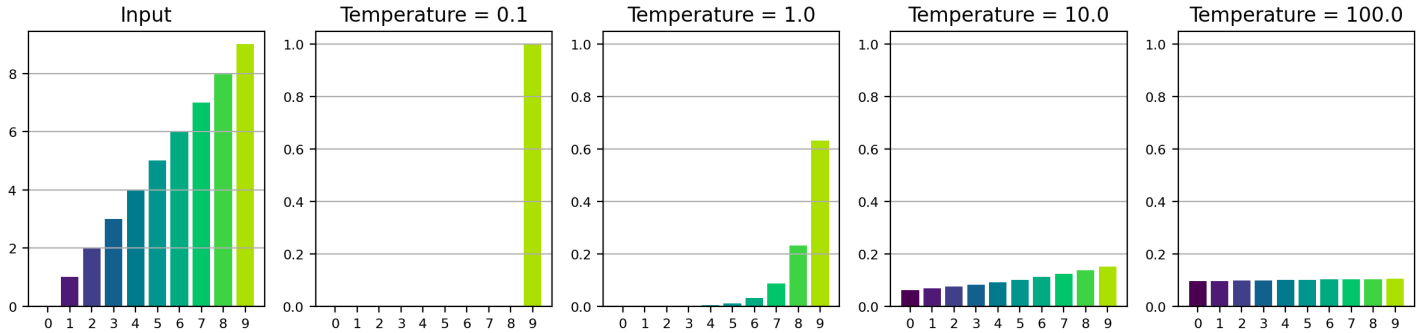
Softmax (3/4)

- What happens if the number of categories increases?

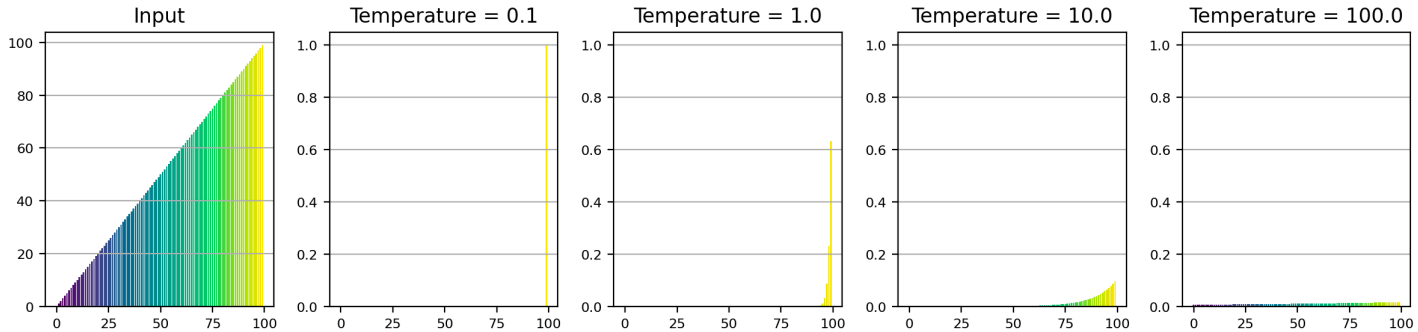
3 categories



10 categories

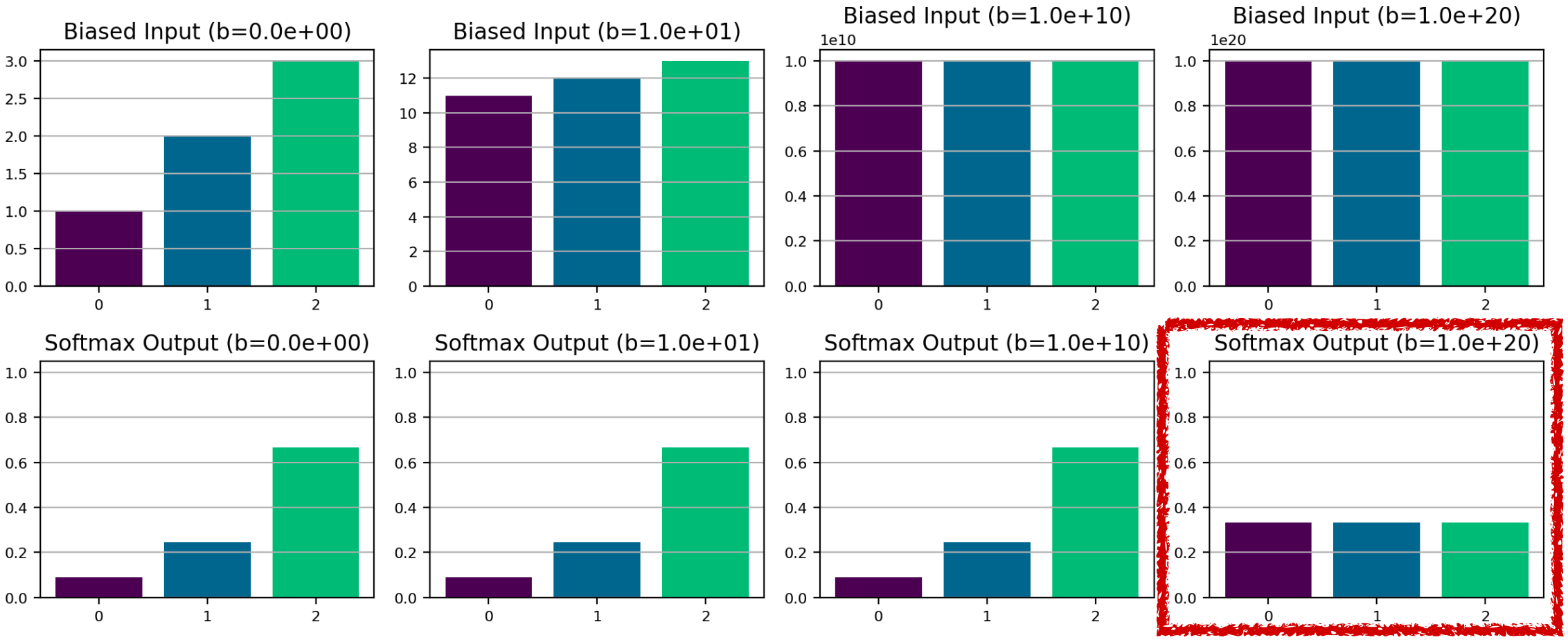


100 categories



Softmax (4/4)

- What about biases (i.e., $\hat{\mathbf{y}} = \text{softmax}(\mathbf{o} + b)$ where b is a bias)?



Why?

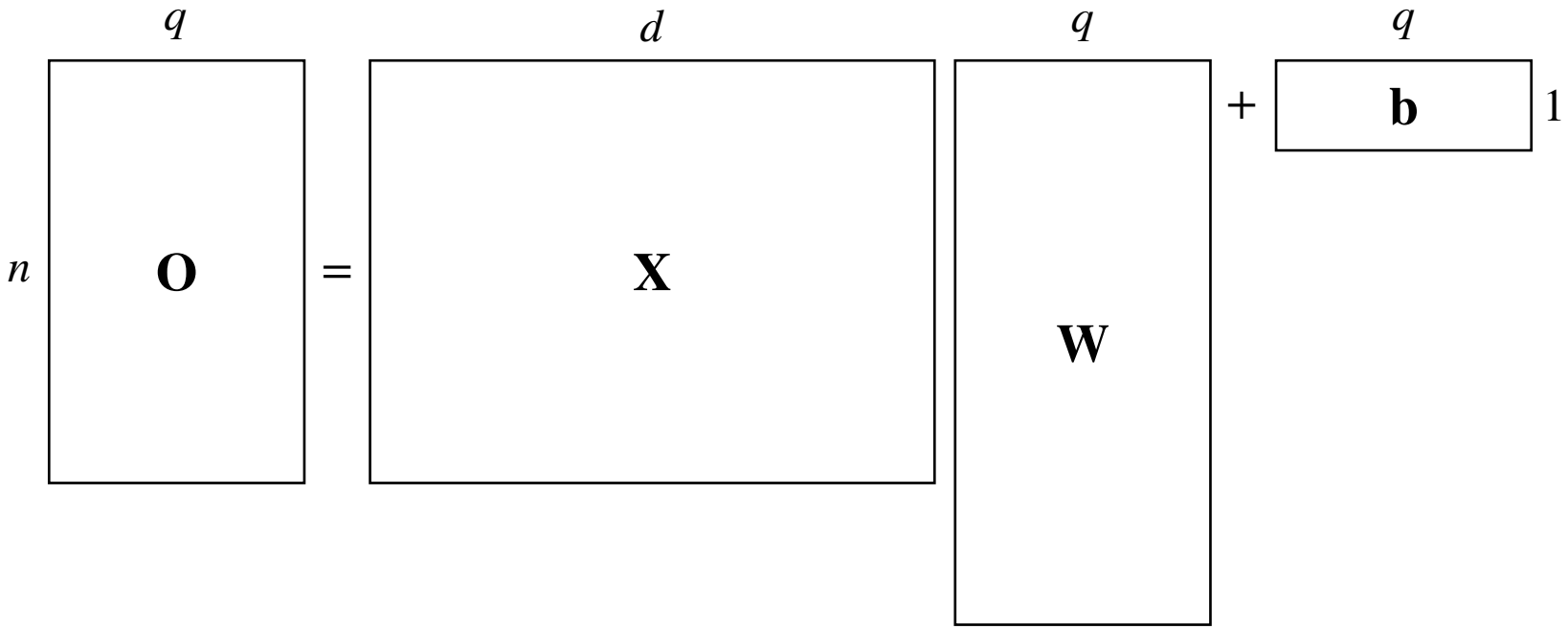
Vectorization

- To improve the computational efficiency, we vectorize calculations in minibatches of data:

$$\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b}$$

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{O})$$

where $\mathbf{O} \in \mathbb{R}^{n \times q}$, $\mathbf{X} \in \mathbb{R}^{n \times d}$, $\mathbf{W} \in \mathbb{R}^{d \times q}$, $\mathbf{b} \in \mathbb{R}^{1 \times q}$, $\hat{\mathbf{Y}} \in \mathbb{R}^{n \times q}$, and softmax($\cdot$) is computed in a row-wise manner.



Loss Function

- The softmax function gives us a vector $\hat{\mathbf{y}}$, which can be interpreted as condition probabilities of each class given an input $\mathbf{x}$ (e.g., $\hat{y}_1 = P(y = \text{cat} \mid \mathbf{x})$.
- The likelihood of the model is given by:

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

- We minimize the negative log-likelihood:

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

where $l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j$ which is commonly called the cross-entropy loss.

Cross-Entropy Loss

- Plugging the softmax output into the cross-entropy loss function:

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

- One useful fact of $\text{lse}(\mathbf{o}) = \log \sum_{k=1}^{q} \exp(o_k)$ is:

$$\max(\mathbf{o}) \leq \text{lse}(\mathbf{o}) \leq \max(\mathbf{o}) + \log q$$

- To understand a bit better, consider the derivative of $l(\mathbf{y}, \hat{\mathbf{y}})$ w.r.t. o_j:

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

- In other words, the derivative is the difference between the probability assigned by our model, $\hat{\mathbf{y}}$, and the ground truth label, $\mathbf{y}$.
  - This fact makes computing gradients easy in practice.

Information Theory

- The central idea of information theory is to quantify the amount of information contained in data.
- For a discrete distribution P, its **entropy** is defined as:

$$H[P] = \sum_j - P(j) \log P(j) = \mathbb{E}[-\log P]$$

- The entropy can be interpreted as an **expected surprisal** where the surprisal is defined as a negative log probability (i.e., lower probability, greater surprisal).
- 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 generated according to probabilities P:

$$H(P, Q) = \sum_j - P(j) \log Q(j)$$

Multilayer Perceptron

Next Week


서울과학기술대학교
SEOUL NATIONAL UNIV. OF SCIENCE & TECHNOLOGY
1910