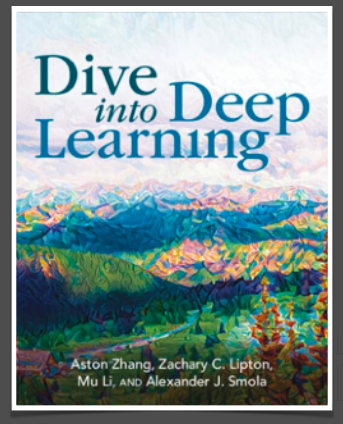
ROBOT INTELLIGENCE LAB

Introduction to Deep Learning

Linear Model (+Machine Learning in General)

Sungjoon Choi, Korea University



Content

ROBOT INTELLIGENCE LAB

- Linear Regression
- Linear Classification (+Machine Learning in General)

ROBOT INTELLIGENCE LAB

Linear Regression

Why Linear Models First?

ROBOT INTELLIGENCE LAB

- 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

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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)

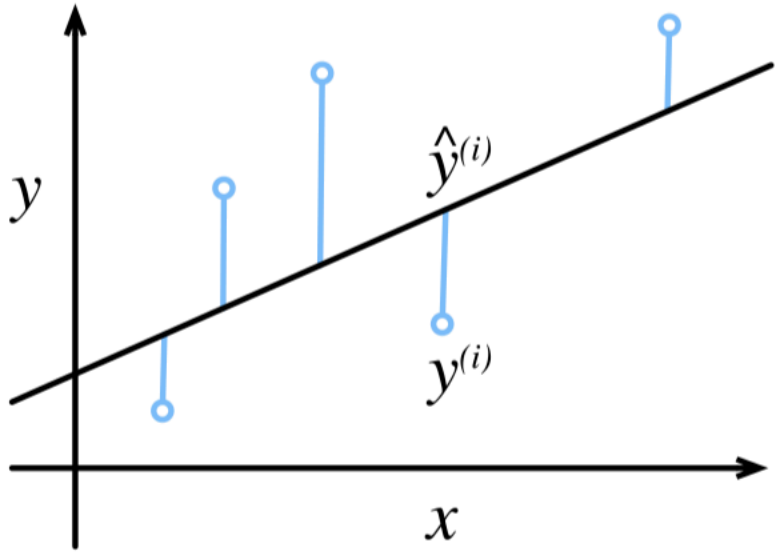
ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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.

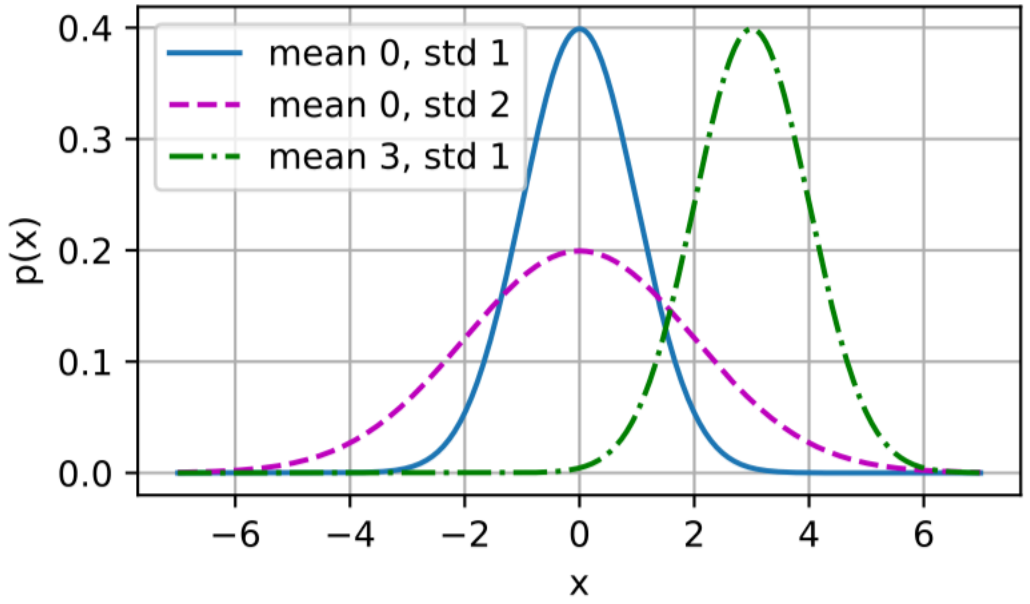
- In the end, the quality of the solution is typically assessed on a separate validation dataset.

Maximum Likelihood Learning (1/3)

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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)

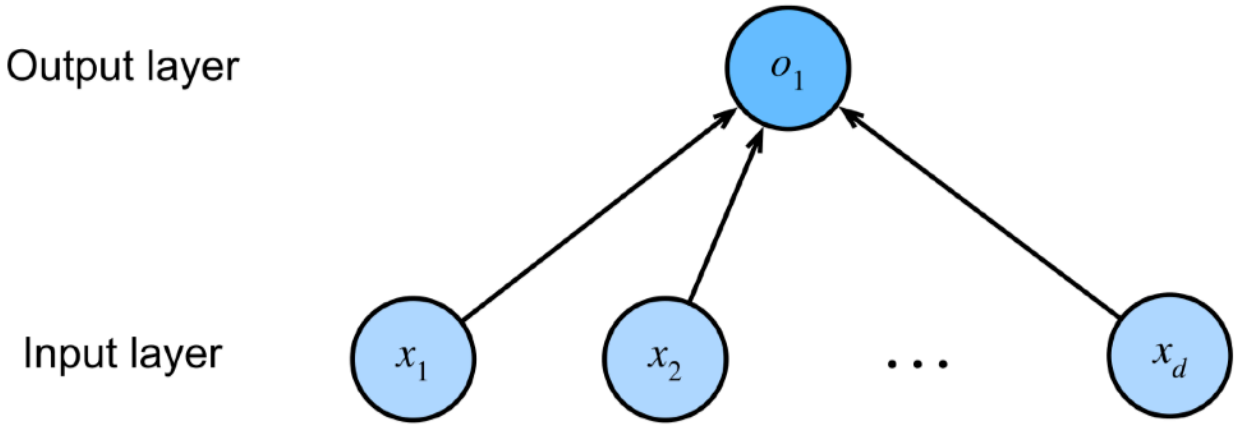
ROBOT INTELLIGENCE LAB

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

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

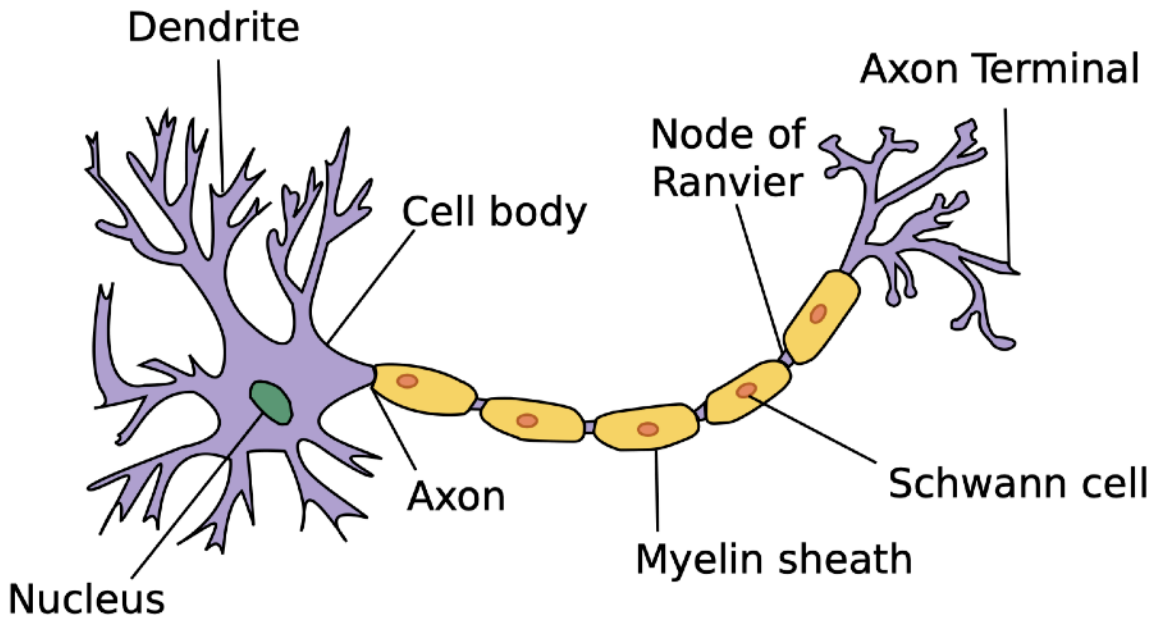
- If we assume that σ is fixed, we can ignore the first term.
- In fact, as the solution does not depend on σ, minimizing the mean squared error is equivalent to the maximum likelihood estimation of a linear model under the assumption of additive Gaussian noise.

Linear Regression as a Neural Network



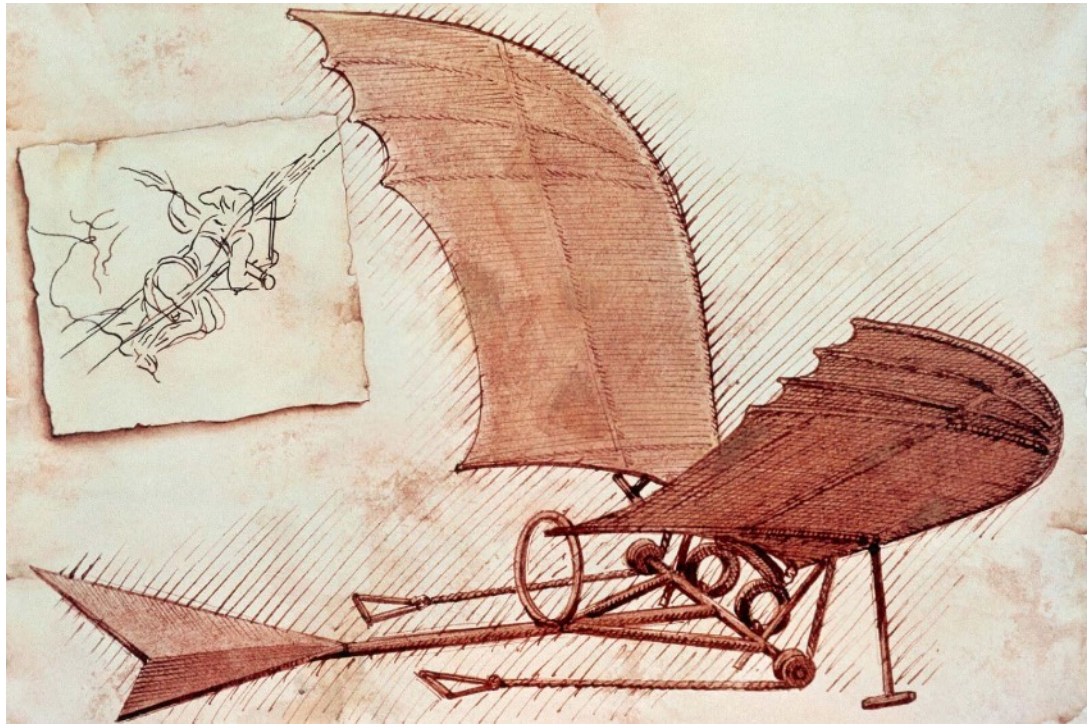
- Neural networks are rich enough to subsume linear models in which every feature is represented by an input neuron, all of which are connected directly to the output.
- The inputs are $x_1, \ldots, x_d$. We refer d as the **number of inputs** of **feature dimensionality** in the input layer.

Biology (1/2)



- The above biological neuron consists of
  - dendrites (input terminals)
  - nucleus (CPU)
  - axon (output wire)
  - axon terminals (output terminals)
  - synapses (connections)

Biology (2/2)



Leonardo Da Vinci's Glider Sketch

Lockheed Martin F-35A

- Russel and Norvig [1] pointed out that although airplanes might have been inspired by birds, ornithology (study of birds) has not been the primary driver of aeronautics innovation.

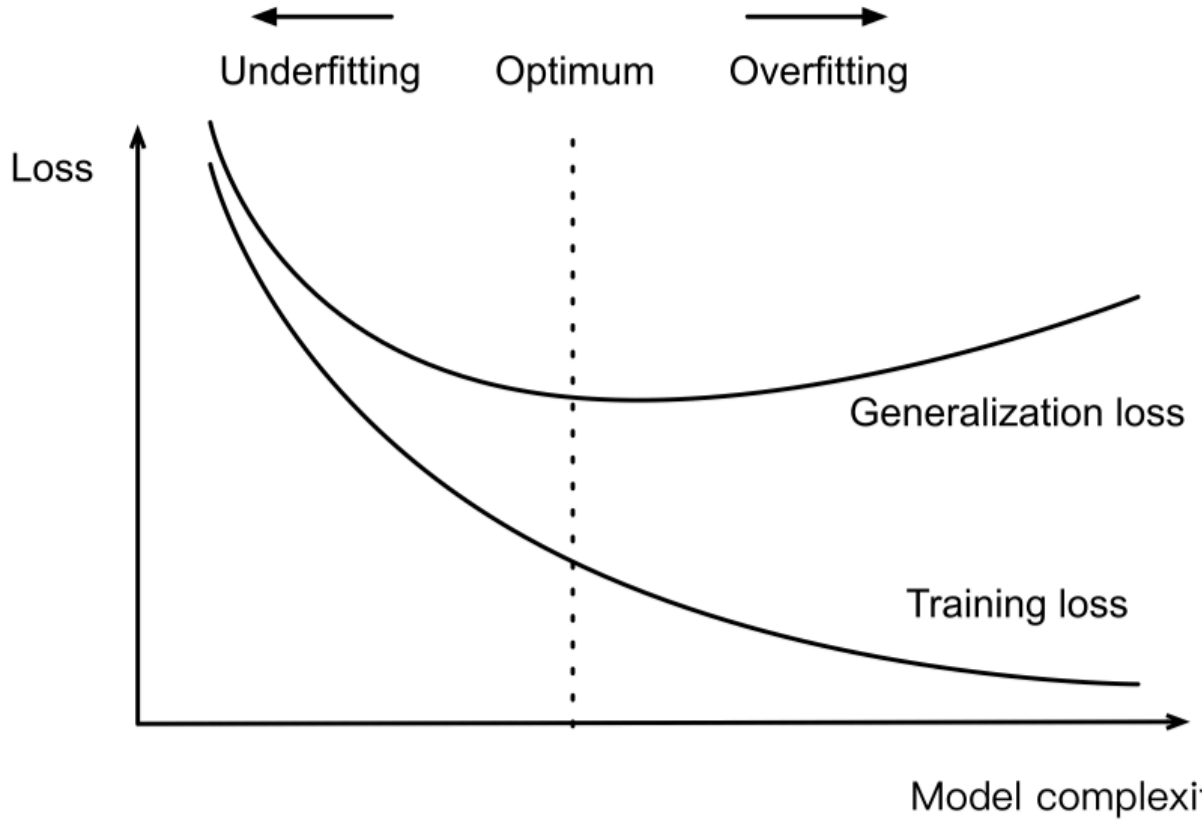
[1] "Artificial Intelligence: A Modern Approach"

Generalization

ROBOT INTELLIGENCE LAB

- 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

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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)

ROBOT INTELLIGENCE LAB

- 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

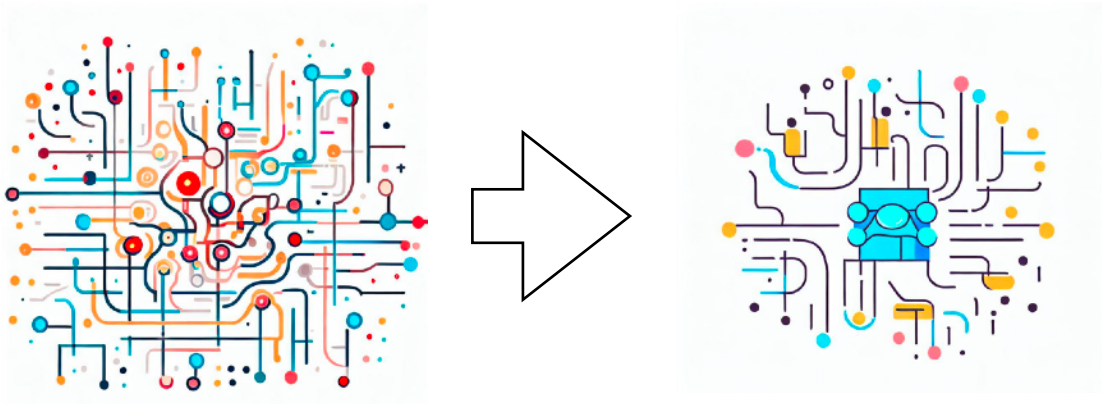
- Typically, we select our final model, only after evaluating multiple models that differ in various ways (different architectures, training objectives, selected features, learning rates, etc.).
- The common practice is to split our data in three ways, training, test, and validation sets, but it is a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

K-fold Validation

ROBOT INTELLIGENCE LAB

- 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

ROBOT INTELLIGENCE LAB

Linear Classification

Classification

ROBOT INTELLIGENCE LAB

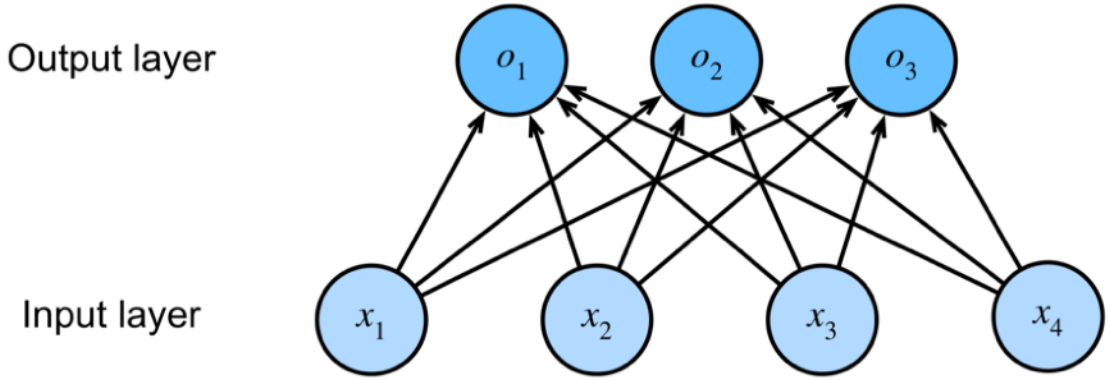
- Now, we pivot towards classification, but most of the plumbing remains the same:
  - loading the data, passing it through the model, generating output, calculating the loss, taking the gradient w.r.t. weights, and updating the model.
- The main difference between classification and regression comes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

ROBOT INTELLIGENCE LAB

- 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, $P(y = i) \propto \exp o_i$:

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

- One can also add a temperature here:

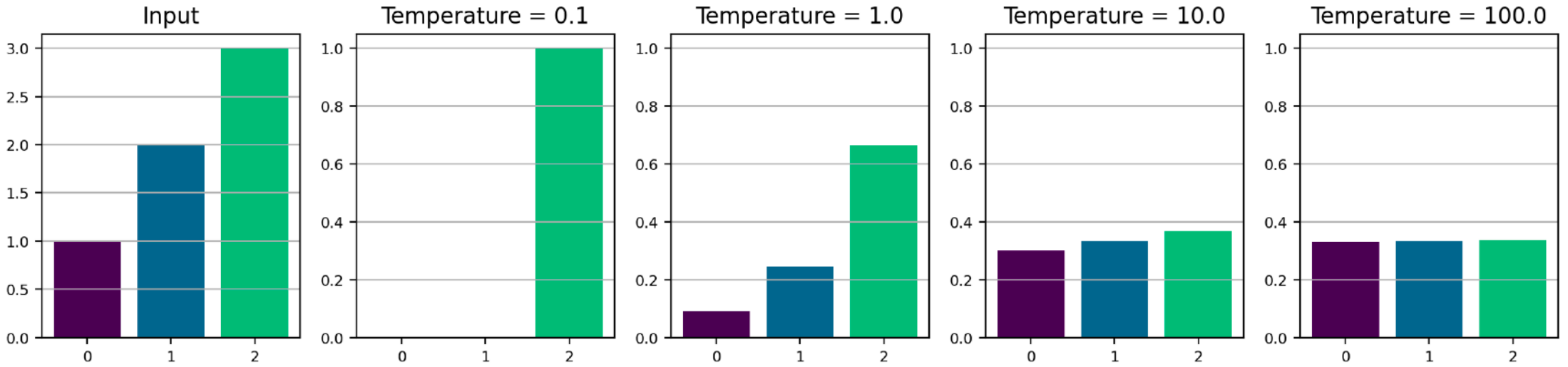
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{o}) \text{ where } \hat{y}_i = \frac{\exp(o_i/\tau)}{\sum_j \exp(o_j/\tau)} \text{ and } \tau \text{ is a temperature}$$

Softmax (2/4)

- Temperature is an important tuning parameter.

```python
def plot_softmax_temperatures(x,temperatures=1.0,SKIP_XTICKS=False):
    def softmax(x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=0)
    indices     = range(len(x))
    colors      = plt.cm.viridis(np.linspace(0,1,len(x)+1))
    fig, axarr = plt.subplots(1, len(temperatures)+1, figsize=(12,3))
    axarr[0].bar(indices, x, color=colors)
    axarr[0].set_title('Input')
    if not SKIP_XTICKS: axarr[0].set_xticks(indices)
    axarr[0].grid(True, axis='y')
    for i, t in enumerate(temperatures):
        probs = softmax(x/t)
        axarr[i+1].bar(indices, probs, color=colors)
        axarr[i+1].set_title(f'Temperature = {t}')
        if not SKIP_XTICKS: axarr[i+1].set_xticks(indices)
        axarr[i+1].set_ylim(0, 1.05)
        axarr[i+1].grid(True, axis='y')
    plt.tight_layout()
    plt.show()
```

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

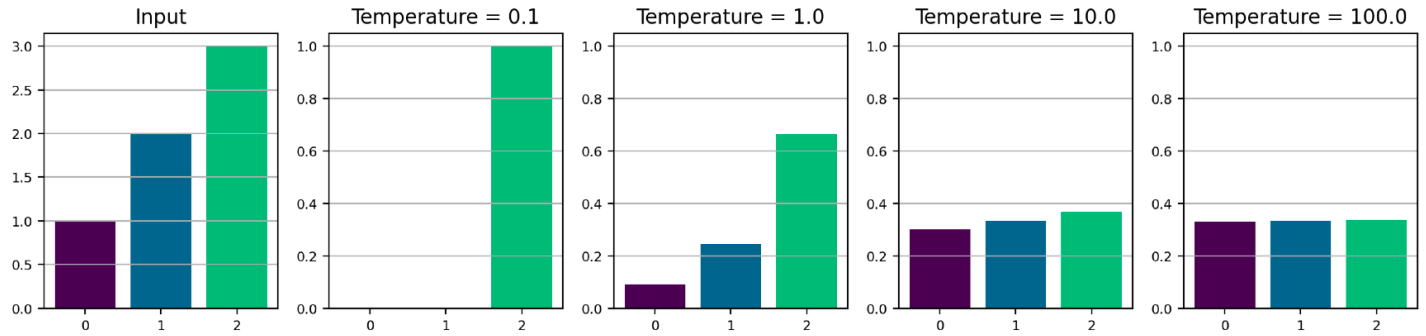


Softmax (3/4)

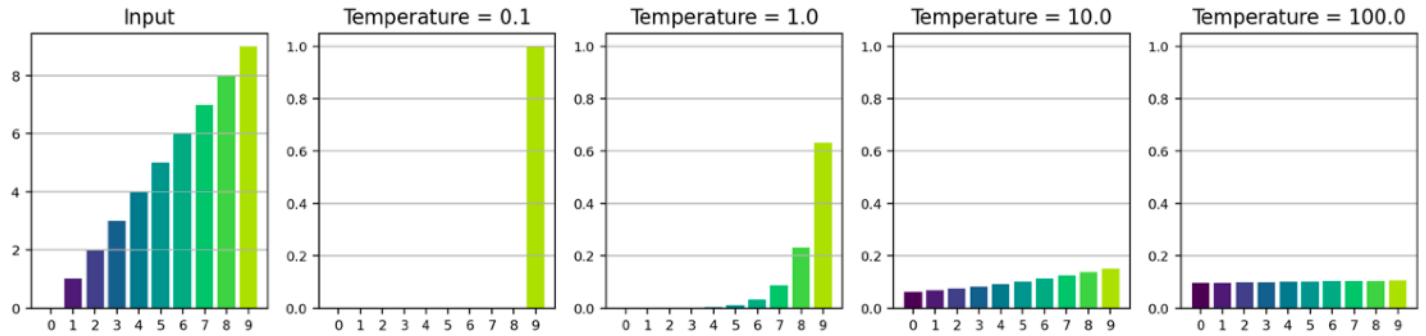
ROBOT INTELLIGENCE LAB

- What happens if the number of categories increases?

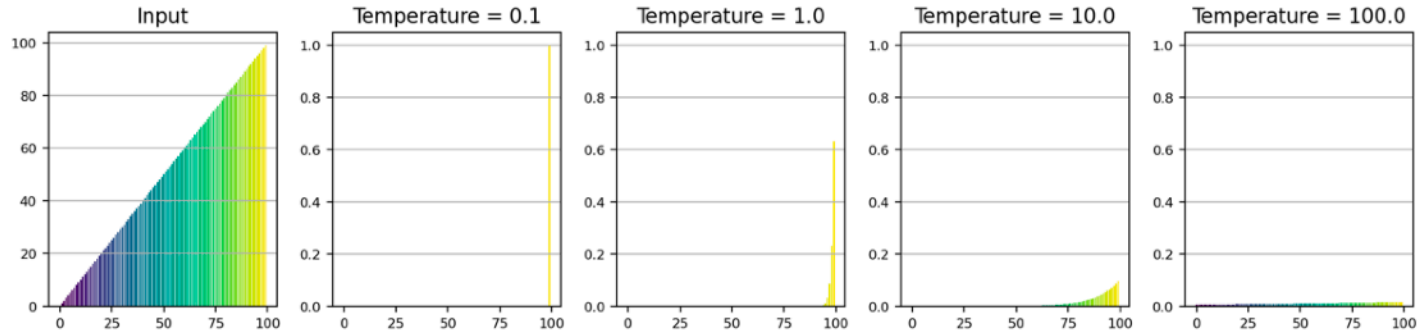
3 categories



10 categories



100 categories

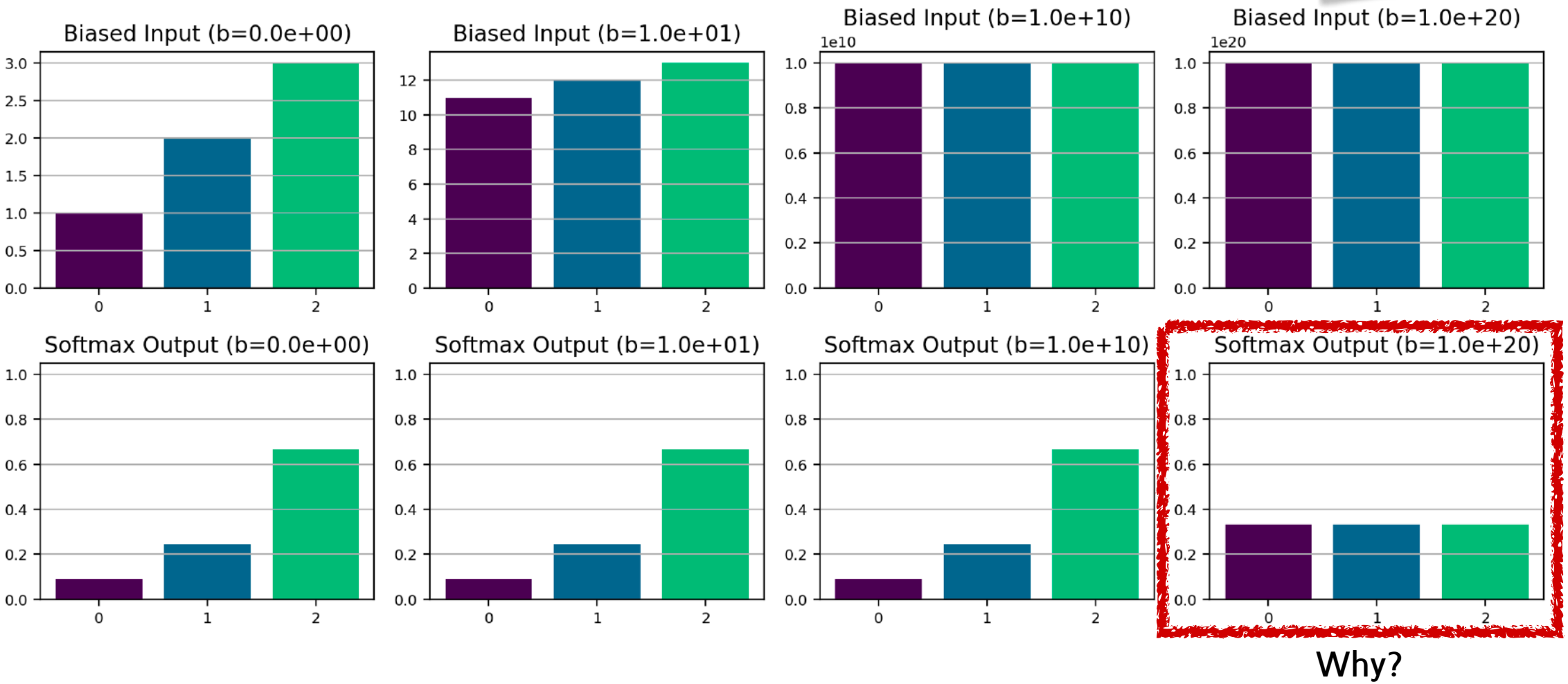


Softmax (4/4)

```
def plot_softmax_biases(x,biases=[0.0,10.0,1e10,1e20]):
    def softmax(x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=0)
    indices     = range(len(x))
    colors      = plt.cm.viridis(np.linspace(0,1,len(x)+1))
    fig, axarr = plt.subplots(2,len(biases),figsize=(12,5))
    for i,b in enumerate(biases):
        axarr[0,i].bar(indices, x+b, color=colors)
        axarr[0,i].set_title('Biased Input (b=%.1e)'%(b))
        axarr[0,i].set_xticks(indices)
        axarr[0,i].grid(True, axis='y')

    for i,b in enumerate(biases):
        probs = softmax(x+b)
        axarr[1,i].bar(indices, probs, color=colors)
        axarr[1,i].set_title('Softmax Output (b=%.1e)'%(b))
        axarr[1,i].set_xticks(indices)
        axarr[1,i].set_ylim(0, 1.05)
        axarr[1,i].grid(True, axis='y')
    plt.tight_layout()
    plt.show()
```

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



Why?

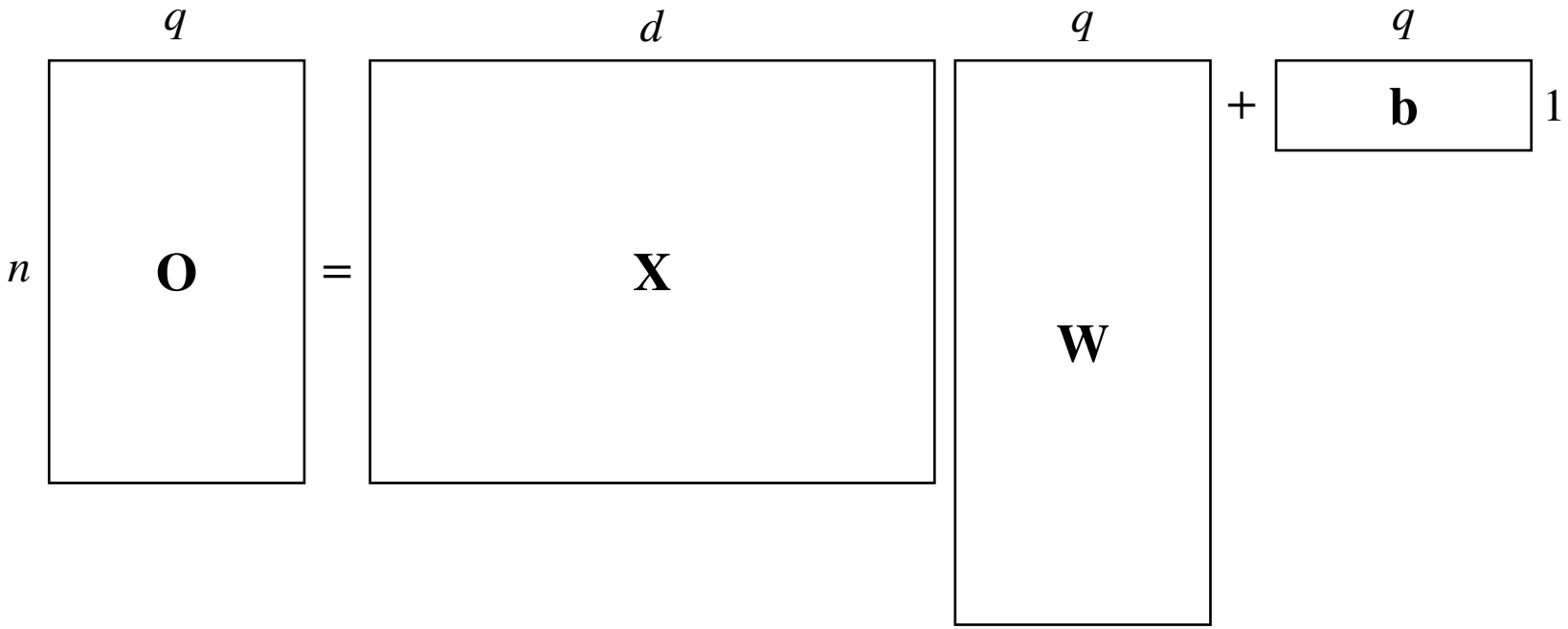
Vectorization

- 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

ROBOT INTELLIGENCE LAB

- 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 joint probability 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

ROBOT INTELLIGENCE LAB

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

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

- 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

ROBOT INTELLIGENCE LAB

- 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:

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

Generalization in Classification (1/2)

ROBOT INTELLIGENCE LAB

- As our goal is to learn **general patterns**, high accuracy on the training set means nothing.
- Some open questions:
  - How many test examples do we need to estimate the test accuracy of the model?
  - What happens if we keep evaluating models on the same test?
  - Why should we expect that fitting our linear models to the training set should fare any better than our naive memorization scheme?

Generalization in Classification (2/2)

ROBOT INTELLIGENCE LAB

- It turns out that we often can guarantee generalization **a priori**:
  - for many models, and for any desired upper bound on the generalization gap ϵ, we can often determine some required number of samples n such that if our training set contains at least n samples, then our empirical error will lie within ϵ of the true error, for any data generating distribution.
- Unfortunately, such guarantees are of limited practical utility to the deep learning practitioner (i.e., we might need trillions or more examples).
- Empirically, deep neural networks typically **generalize remarkably well** with far fewer examples (thousands).
  - Why?

The Test Set (1/3)

ROBOT INTELLIGENCE LAB

- We rely on test sets as the gold standard method for assessing generalization error. Then, what are the properties of such error estimates?
- Suppose we have a fixed classifier f and a fresh dataset $\mathscr{D} = (\mathbf{x}^{(i)}, y^{(i)})_{i=1}^{n}$ that were not used to train the classifier f.
  - The empirical error of our classifier f on $\mathscr{D}$ is simply the fraction of instances for which the prediction $f(\mathbf{x}^{(i)})$ disagrees with the true label $y^{(i)}$, and if given by the following expression:

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

- By contrast, the population is the expected fraction of examples in the underlying population characterized by the probability density function $p(\mathbf{x}, y)$ for which our classifier disagrees with the true label:

$$\epsilon(f) = \int\int \mathbf{1}(f(\mathbf{x}) \neq y) p(\mathbf{x}, y) d\mathbf{x} dy$$

The Test Set (2/3)

ROBOT INTELLIGENCE LAB

- While $\epsilon(f)$ is the quantity that we actually care about, we cannot observe it directly.
- Because our test set $\mathcal{D}$ is statistically representative of the underlying population, we can view $\epsilon_{\mathcal{D}}(f)$ as a **statistical estimator** of the population error $\epsilon(f)$.
    - Moreover, because our quantity of interest $\epsilon(f)$ is an expectation and $\epsilon_{\mathcal{D}}(f)$ is the sample average, estimating the population error is simply the classic problem of **mean estimation**.
- The **central limit theorem** tells us that whenever we possess n random samples $a_1, \ldots, a_n$ drawn from any distribution with mean μ and standard deviation σ, as the number of samples n approaches infinity, the sample average $\hat{\mu}$ follows $\mathcal{N}\left(\mu, \sigma^2/n\right)$.
    - This tells us that as n increases, $\epsilon_{\mathcal{D}}(f)$ should approach $\epsilon(f)$ at a rate of $O(1/\sqrt{n})$.
    - In general, such a rate of $O(1/\sqrt{n})$ is often the best we can hope in statistics.

The Test Set (3/3)

ROBOT INTELLIGENCE LAB

- The random variable of interest $\mathbf{1}(f(X) \neq Y)$ can only take values 0 and 1 and thus is a Bernoulli random variable, characterized by a parameter p indicating the probability that it takes value 1 (e.g., $\mathbf{1}(f(X) \neq Y) \sim \text{Bern}(p)$).
  - Here, 1 means that our classifier made an error, so the parameter p is actually the true error rate $\epsilon(f)$.
- For a Bernoulli random variable, we can obtain finite sample bounds by applying **Hoeffding's inequality**:

$$P(\epsilon_{\mathscr{D}}(f) - \epsilon(f) \geq t) < \exp(-2nt^2)$$

  - For $t = 0.01$ and 95% confidence, n should exceed 15,000.
  - For $t = 0.01$ and 99% confidence, n should exceed 23,000.
  - For $t = 0.001$ and 99% confidence, n should exceed 2,300,000.

Test Set Reuse (1/2)

ROBOT INTELLIGENCE LAB

- Suppose you have your model f_1 following all our protocols (k-fold validation, hyperparameter optimization, optimization) evaluate your model f_1 on the test set, and report an unbiased estimate of the population error.
- So far so good. However, that night you wake up at 3am with a brilliant idea for a new modeling approach.
  - The next day, you code up your model, tune its hyperparameters on the validation set, and not only are you getting your new model f_2 to work but its error rate appears to be much lower than f_1's.
  - Are you all happy?

Test Set Reuse (2/2)

ROBOT INTELLIGENCE LAB

- Even though the original test set $\mathscr{D}$ is still sitting on your server, you cannot rely on the test error. Why?
  - Recall that our analysis of test set performance rested on the assumption that the classifier was chosen absent any contact with the test set!
  - However, the subsequent function f_2 was chosen after you observed the test set performance of f_1, which is often called an adaptive overfitting problem.

Statistical Learning Theory (1/2)

ROBOT INTELLIGENCE LAB

- It may seem that the test sets are all that we really have. But what can we say when a classifier is trained and evaluated on the same dataset?
- Suppose that our learned classifier f_S trained on the set S must be chosen among some pre-specified set of functions $\mathcal{F}$.
  - We focus on **uniform convergence**, i.e., with high probability, the empirical error rate for every classifier in the class $f \in \mathcal{F}$ will simultaneously converge to its true rate.
  - In other words, we seek a theoretical principle that would allow us to state that with probability at least $1 - \delta$ no classifier's error rate $\epsilon(f)$ will be misestimated by more than some small amount α.

Statistical Learning Theory (2/2)

ROBOT INTELLIGENCE LAB

- Vapnik and Chervonenkis presented the Vapnik-Chervonenkis (VC) dimension, which measures the complexity (flexibility) of a model class.
- The key result of the VC dimension bounds the difference between the empirical error and the population error as a function of the VC dimension and the number of samples:

$$P(R[p, f] - R_{\text{emp}}[\mathbf{X}, \mathbf{Y}, f] < \alpha) \geq 1 - \delta \text{ for } \alpha \geq c\sqrt{(\text{VC} - \log \delta)/n}$$

  where $\delta > 0$ is the probability that the bound is violated, α is the upper bound on the generalization gap, n is the dataset size, and c is a constant that depends only on the scale of the loss.

- For example,
  - linear models on d-dimensional inputs have VC dimension $d + 1$.
  - multilayer perceptions with l layers and d_i input and latent dimensions have VC dimension $O(N \log_2 N)$ where $N = \sum_{i=1}^{l} \sum_{j=1}^{d_{i-1}} (d_{i-1} + 1)$.

Distribution Shift

ROBOT INTELLIGENCE LAB

- Distribution shift indicates the scenario where our training data was sampled from $p_S(\mathbf{x}, y)$ but our test data will consist of examples drawn from a different distribution $p_T(\mathbf{x}, y)$.
- **Covariate shift**: $P(Y \mid \mathbf{X})$ remains the same, but $P(\mathbf{X})$ changes.
  - The distribution of inputs changes, but the labeling function does not change.
  - In training, we see many photorealistic images, but in the test, we see many cartoonish images.
- **Label shift**: $P(\mathbf{X} \mid Y)$ remains the same, but $P(Y)$ changes.
  - This label shift may cause label imbalance problems, i.e., in training, we see many "dogs", but in the test, we see many "cats".
- **Concept shift**: $P(\mathbf{X})$ remains the same, but $P(Y \mid \mathbf{X})$ changes.
  - The labels associated with input images vary, e.g., is it okay to drink cocaine?

Correction of Distribution Shift (1/3)

ROBOT INTELLIGENCE LAB

- **Empirical Risk**
  - What we usually do is the following:

$$\min_f \frac{1}{n} \sum_{i=1}^{n} l(f(\mathbf{x}_i), y_i)$$

- **Covariate Shift Correction**
  - The input distribution of the source, $q(\mathbf{x})$, and the target, $p(\mathbf{x})$, are different.

$$\int\int l(f(\mathbf{x}), y) p(y \mid \mathbf{x}) p(\mathbf{x}) d\mathbf{x} dy = \int\int l(f(\mathbf{x}), y) p(y \mid \mathbf{x}) q(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} dy$$

  - Hence, we can train our model using weighted empirical risk minimization:

$$\min_f \frac{1}{n} \sum_{i=1}^{n} \beta_i l(f(\mathbf{x}_i), y_i)$$

where $\beta_i = \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}$ is the ratio of the probability that it would have been drawn from the correct distribution to that from the wrong one.

Correction of Distribution Shift (2/3)

ROBOT INTELLIGENCE LAB

- **Covariate Shift Correction**
  - But how can we compute $\beta_i = \dfrac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)}$?
    - We assume that we have an equal number of instances from both $p(\mathbf{x})$ and $q(\mathbf{x})$, respectively, and the labels for $p(\mathbf{x})$ are 1 and labels for $q(\mathbf{x})$ are -1.
    - Then, the optimal classifier between $p(\mathbf{x})$ and $q(\mathbf{x})$ is:

$$P(z=1\,|\,\mathbf{x}) = \frac{p(\mathbf{x})}{p(\mathbf{x})+q(\mathbf{x})} \text{ and hence } \frac{P(z=1\,|\,\mathbf{x})}{P(z=-1\,|\,\mathbf{x})} = \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

  - If we use a logistic regression, $P(z=1\,|\,\mathbf{x}) = \dfrac{1}{1+\exp(-h(\mathbf{x}))}$, and

$$\beta_i = \frac{1/(1+\exp(-h(\mathbf{x}_i)))}{\exp(-h(\mathbf{x}_i))/(1+\exp(-h(\mathbf{x}_i)))} = \exp(h(\mathbf{x}_i))$$

Correction of Distribution Shift (3/3)

ROBOT INTELLIGENCE LAB

- **Label Shift Correction**
  - Since we assume that labels are shifted, we can estimate the test set label distribution by solving a simple linear system:

$$\mathbf{C}p(\mathbf{y}) = \mu(\hat{\mathbf{y}})$$

where $\mathbf{C} \in \mathbb{R}^{k \times k}$ is the confusion matrix, where each column corresponds to the ground truth label and each row corresponds to our model's predicted category.

  - If the confusion matrix is invertible, we get a solution:

$$p(\mathbf{y}) = \mathbf{C}^{-1}\mu(\hat{\mathbf{y}})$$

Fairness in Machine Learning

ROBOT INTELLIGENCE LAB

- We should be careful about how prediction systems can lead to feedback loops.
- Consider predictive policing systems, which allocate patrol officers to areas with high forecasted crime.
    1. Neighborhoods with more crime get more patrols.
    2. Consequently, more crimes are discovered in these neighborhoods.
    3. Exposed to more positives, the model predicts yet more crime in such neighborhoods.
    4. In the next iteration, the updated model target the same neighborhood even more heavily leading to yet more crime discovered.
- These are often called runaway feedback loops.
- "Should the news that an individual encounters be determined by the set of Facebook pages they have liked?"

Title Text