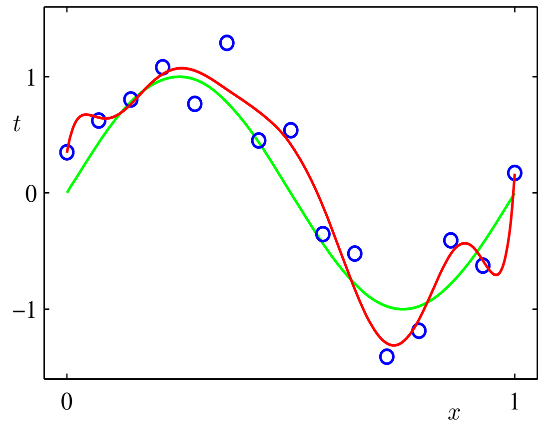
Probabilistic Linear Regression

Uncertainty in Decision Making

- You make a system predicting stock prices. Suppose that the system predicts that the price of stock X will increase 5% tomorrow. Will you bet all your money?
- A binary classifier predicting the existence of a cancer says TRUE (exist) will you go to a surgery?
  - What if this decision is made based on the certainty of 51% of true and 49% of false.
- Therefore, we not only need to predict a point but also need to learn the uncertainty (probability) of the prediction



P(y|x) vs y = f(x)

- That is, we want a probabilistic model **p(y|x)** instead of a deterministic model **y = f(x)**

- Variable vs random variable & function vs distribution

  - In the former, y is a ***random variable***, thus we need to model its ***distribution*** p(y|x)

  - In the latter, y is a deterministic (non-random) variable, and thus we need to model a **function** y = f(x)

How to construct a probabilistic model

- The observed data contains the observation noise *e* and we can model this as follows:

$$y = f(x;w) + \boldsymbol{e}$$

  - Here, we assume additive noise *e*
  - Note that the MSE loss for regression we considered last week do not consider this noise term
- For modeling, we need to make an assumption about what kind of distribution does the noise *e* follows.
- By assuming that the noise can be either positive or negative and small noise is more likely than large noise, we can use a Gaussian distribution

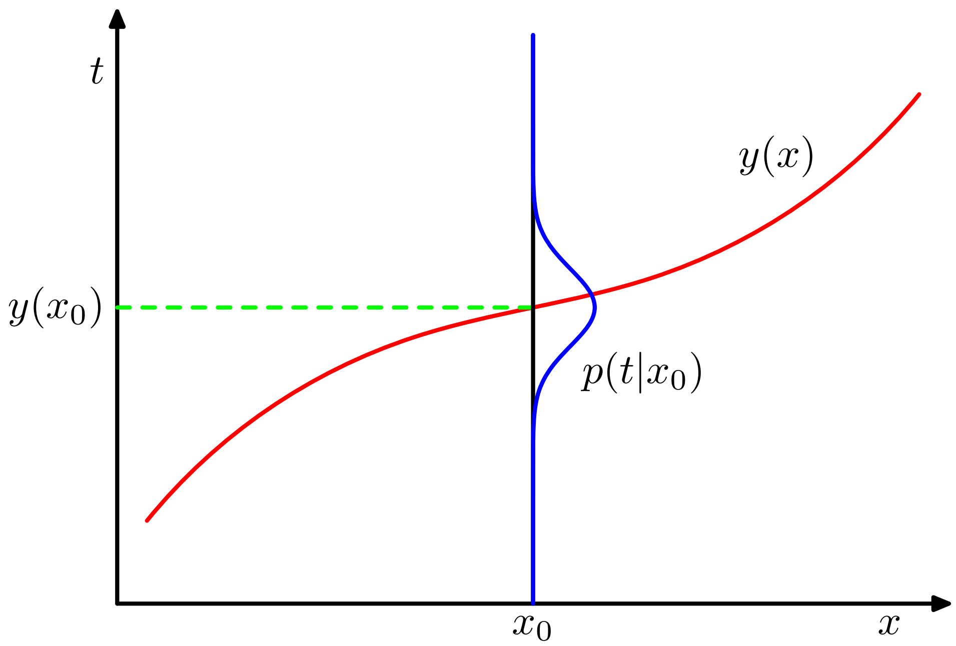
$$e \sim N(0,\ \sigma^2)$$

- Then we can construct a probabilistic model

$$p(y|x) = N(\mu, \sigma^2)$$

where $\mu = f(x; w)$ and $f(x; w) = \boldsymbol{x}^\top \boldsymbol{w}$ is the deterministic linear regression function.

- How can we learn this model $N(f(x; w), \sigma^2)$ from data?



Maximum Likelihood Estimation (MLE)

- Such a function modeling a distribution with a model parameter w is called a ***likelihood function***
- The objective of the maximum likelihood estimation is to maximize the likelihood of the data w.r.t. the parameter w
    - But we will see that these two can be the same in certain conditions

Gaussian MLE for Regression

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

- By setting $\mu = wx + b$ and σ as a hyperparameter
- We have the Gaussian linear regression model

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

Loss Function of Probabilistic Linear Regression

- According to the *maximum likelihood principle,* the best parameters are those that maximize the likelihood of the entire dataset which we can write as follows:

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

- Thus, we maximize this w.r.t. the model parameter w. How?

Negative Log-Likelihood (NLL)

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

- Maximizing the product of many exponential functions becomes simple by taking log (and negative sign to turn it to a minimization problem)
  - As log() is a monotone function, taking log does not change the solution
- This gives the **Negative Log-Likelihood** (NLL) objective

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

  - Compare this objective to MSE. What is similar and different?
- For regression, the solution of Gaussian MLE is equivalent to that of MSE loss.

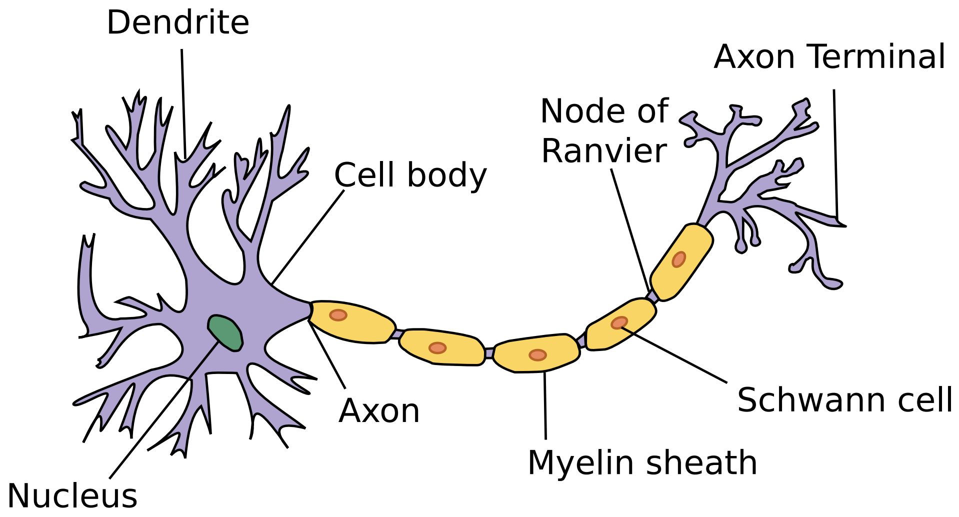
$$P(y \mid \mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(y - \mathbf{w}^\top \mathbf{x} - b)^2\right)$$

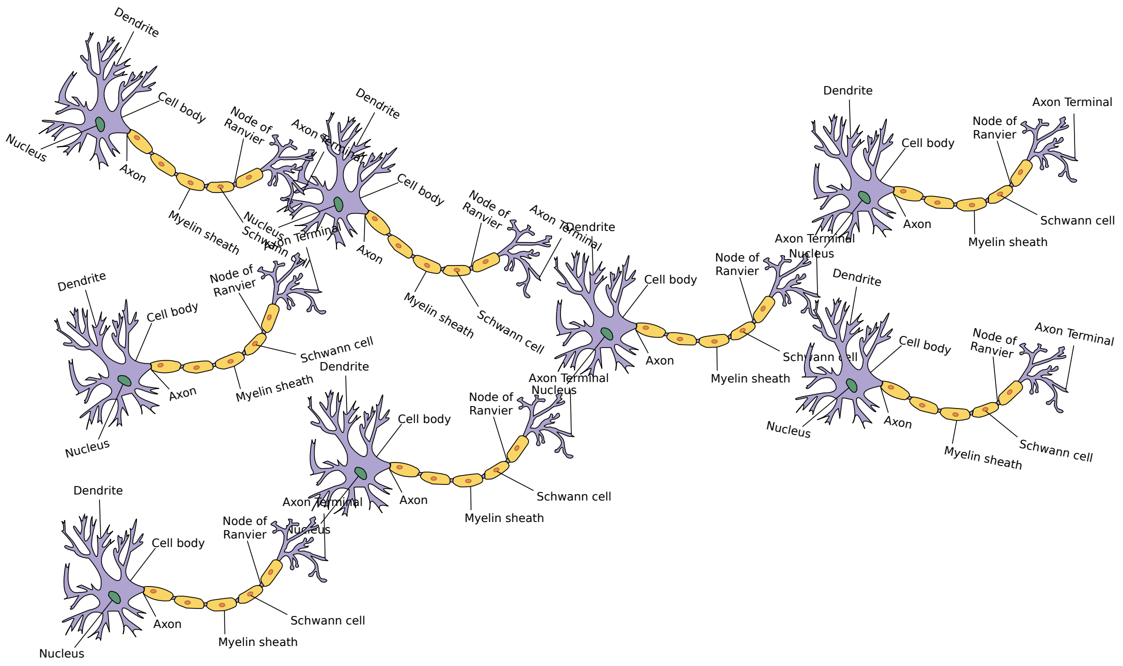
Training in Gaussian MLE

- We saw that when the mean prediction is a linear model, Gaussian MLE is equivalent to MSE of a linear model.
- This means that we can find the solution in both ways: using the normal equation or using gradient descent

Linear Regression to Neural Networks

Biological Neuron and Neural Network

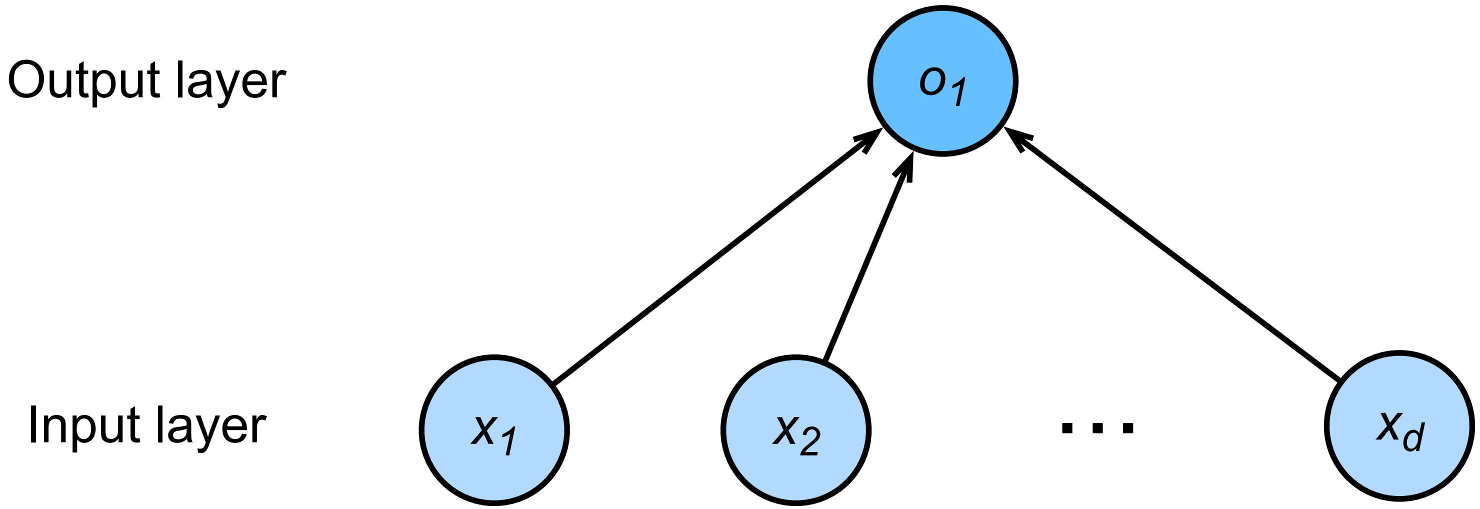


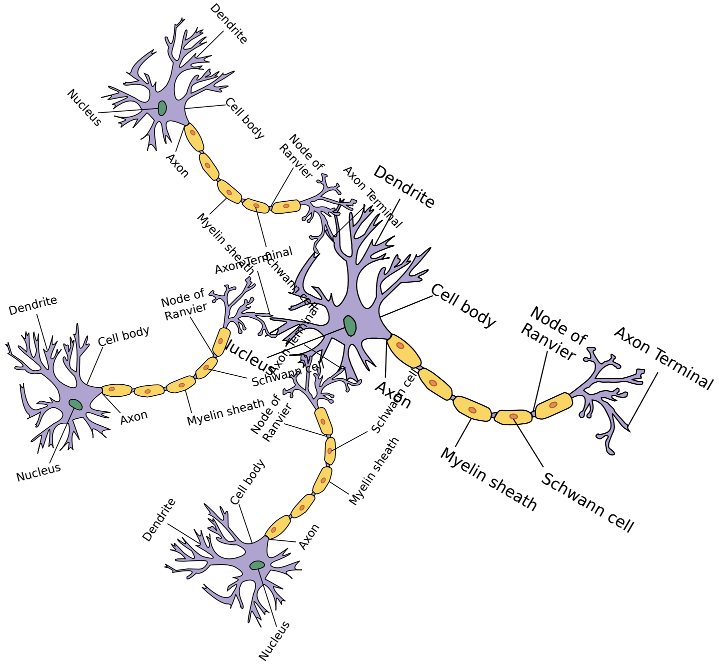


Neural Network Interpretation of Linear Regression

- Weights can be considered as the

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





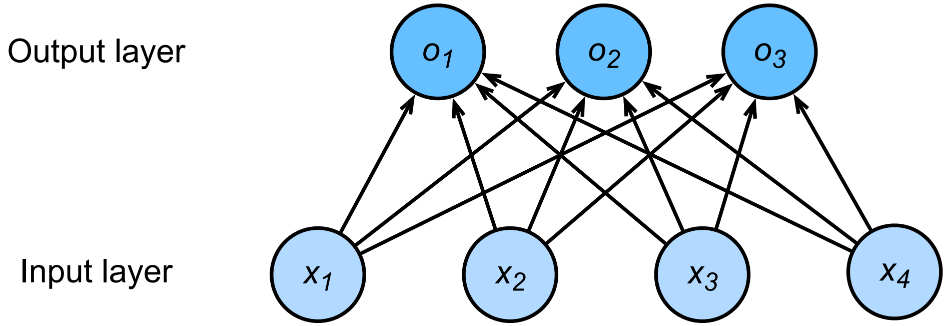
Linear Classification with Softmax

One-hot vector

- The class label is nominal (no order). So, how can be represent class 1,2,3 or a,b,c, or "cat", "chicken", "dog"?
- In classification, it is convenient to express a label (target) by the **one-hot vector**
  - Class 1 (or "chicken") : 100
  - Class 2 (or "dog") : 010
  - Class 3 (or "cat") : 001
  - Not that the order class 1,2,3 is arbitrary
- If we build a model predicting next word given past words, we have as many classes as the size of vocabulary (say, 100k). In this case, a word is expressed a one-hot vector which contains 1 only in one place in the vector of dimension 100k, and 0 in all other places.

Linear Regression with Multiple Outputs

- Assume that the total number of classes are C
- We can simply extend the linear regression model to predict C outputs



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

- We want to make the output o_1 to represent the **probability** of class 1 to be the answer
  - (Cat, chicken, dog) = (0.2, 0.7, 0.1)

Softmax Operation

- This means that we need to normalize the output so that its sum becomes 1 and each output is nonnegative
- Softmax function does this

nonnegative

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

Sum to 1

Loss Function for Classification

- Cross-Entropy Loss: Maximum-Likelihood for Classification

$$P(Y \mid X) = \prod_{i=1}^{n} P(y^{(i)} \mid x^{(i)}) \text{ and thus } -\log P(Y \mid X) = \sum_{i=1}^{n} -\log P(y^{(i)} \mid x^{(i)}).$$

- where

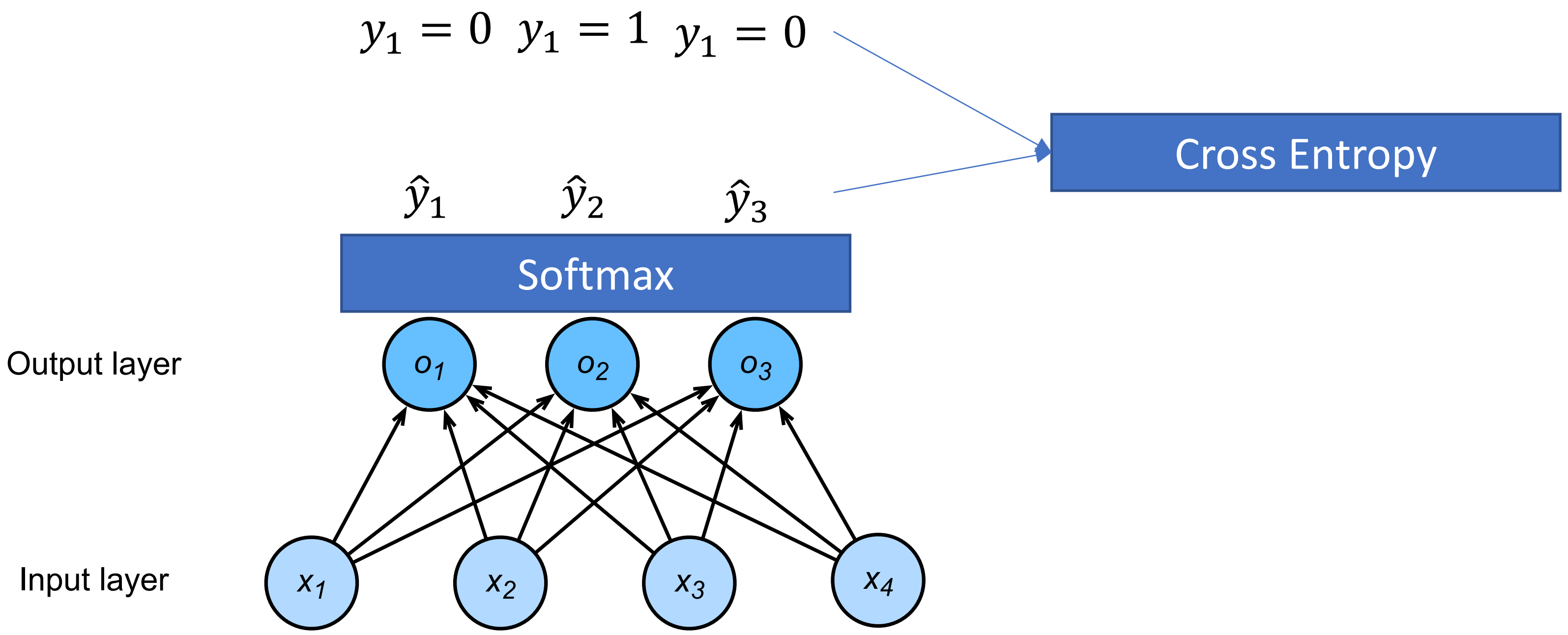
Predicted probability of class j

$$l = -\log P(y \mid x) = -\sum_j y_j \log \hat{y}_j.$$

Actual probability of class j

Cross-Entropy

$$l = -\log P(y \mid x) = -\sum_j y_j \log \hat{y}_j.$$



The Gradients of The Cross Entropy Loss

- Softmax is a non-linear function. And thus we don’t have a close form solution. We need use gradient descent
- Compute the gradient of the cross-entropy loss