Practice Problems

- Naïve Bayes Classifier
- Expressiveness of Neural Networks
- Bias Updates in 2-layer Forward Neural Network
- PCA HW08
- Kernel Perceptron HW09

Naïve Bayes Classifier

- Consider three random variables X1, X2 and Y . X1 and Y are generated according to Tables 1 and 2. Once X1 has been generated, X2 is assigned the same value as X1.
- Consider a Naïve Bayes classifier trained on features X1 and X2 to predict class label Y

	$P(Y)$
$Y = 0$	0.7
$Y = 1$	0.3

Table 1: $P(Y)$

	$X_1 = 0$	$X_1 = 1$
$Y = 0$	0.6	0.4
$Y = 1$	0.4	0.6

Table 2: $P(X_1|Y)$

Naïve Bayes Classifier

- P1: What is P(X2=x|Y=y)?
- Sol:
  - Since “Once X1 has been generated, X2 is assigned the same value as X1. ”

P(X2=x|Y=y) = P(X1=x|Y=y), we only need to copy Table 2 to Table 3.

	$X_1 = 0$	$X_1 = 1$
$Y = 0$	0.6	0.4
$Y = 1$	0.4	0.6

Table 2: $P(X_1|Y)$

	$X_2 = 0$	$X_2 = 1$
$Y = 0$	0.6	0.4
$Y = 1$	0.4	0.6

Table 3: $P(X_2|Y)$

Naïve Bayes Classifier

- P2: What is the prediction y for X1 = 0/1 and X2 = 0/1?
- Sol: Recall decision rule of Naïve Bayes Classifier
  - $\hat{y}=\underset{y=\{1,0\}}{\operatorname{argmax}} P(Y=y)P(X_1=x_1|Y=y)P(X_2=x_2|Y=y)$

	Y = 0	Y = 1
X1 = 0, X2 = 0	0.7*0.6*0.6	0.3*0.4*0.4
X1 = 0, X2 = 1	0.7*0.6*0.4	0.3*0.4*0.6
X1 = 1, X2 = 0	0.7*0.4*0.6	0.3*0.6*0.4
X1 = 1, X2 = 1	0.7*0.4*0.4	0.3*0.6*0.6

$$P(Y=y)P(X_1=x_1|Y=y)P(X_2=x_2|Y=y)$$

	$P(Y)$
$Y=0$	0.7
$Y=1$	0.3

Table 1: $P(Y)$

	$X_1=0$	$X_1=1$
$Y=0$	0.6	0.4
$Y=1$	0.4	0.6

Table 2: $P(X_1|Y)$

	$X_2=0$	$X_2=1$
$Y=0$	0.6	0.4
$Y=1$	0.4	0.6

Table 3: $P(X_2|Y)$

Naïve Bayes Classifier

- P3: Now consider a second Naive Bayes classifier trained without the duplicate feature X2. What are predictions? Do they agree?
- Sol: The predictions agree. However, in this case, ***X2 is not conditionally independent with X1 given Y***.

	Y = 0	Y = 1
X1 = 0	0.7*0.6	0.3*0.4
X1 = 1	0.7*0.4	0.3*0.6

$$P(Y = y)P(X_1 = x_1|Y = y)$$

	$P(Y)$
$Y = 0$	0.7
$Y = 1$	0.3

Table 1: $P(Y)$

	$X_1 = 0$	$X_1 = 1$
$Y = 0$	0.6	0.4
$Y = 1$	0.4	0.6

Table 2: $P(X_1|Y)$

Expressiveness of Neural Networks

- Consider neural networks built out of units that take real-valued input X1, ..., Xn where the unit output Y is given by the sigmoid function. The inputs Xi will be 0 or 1. The output Y will be real-valued, ranging between 0 and 1. We will interpret Y as a boolean value by interpreting it to be a boolean 1 if Y > 0.5, and interpreting it to be a boolean 0 otherwise.

$$Y = \frac{1}{1+exp(-(w_0+\sum_i w_i X_i))}$$

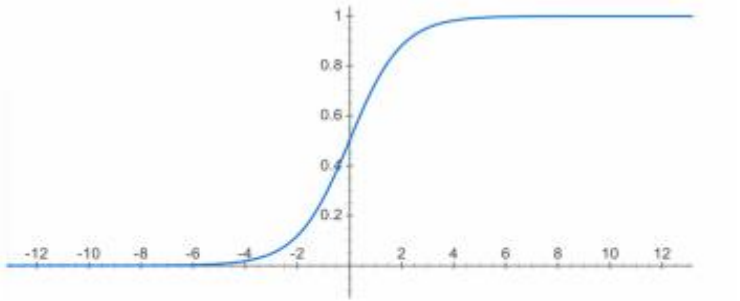


Figure 1: Non-linearity function $f(x) = \frac{1}{1+exp(-x)}$

Expressiveness of Neural Networks

- P1: Using Figure 1, give 3 weights for a single unit with input X1 and X2 that implements Y= X1 OR X2
- Sol:

	Y = X1 OR X2	Weighted sum
X1 = 0, X2 = 0	< 0.5	w0 < 0
X1 = 0, X2 = 1	> 0.5	w0 + w2 > 0
X1 = 1, X2 = 0	> 0.5	w0 + w1 > 0
X1 = 1, X2 = 1	> 0.5	w0 + w1 + w2 > 0

- One solution: w0 = -10, w1 = w2 = 20

$$Y = \frac{1}{1+exp(-(w_0+\sum_i w_i X_i))}$$

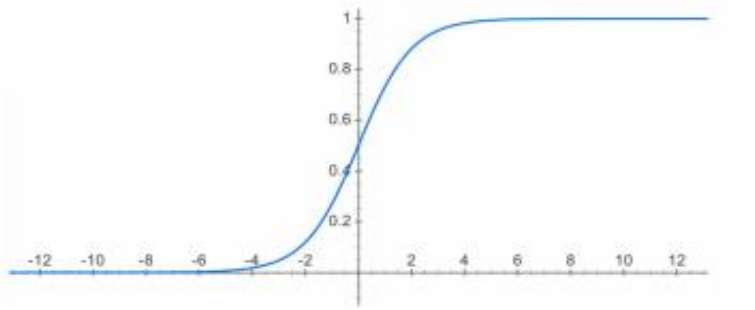


Figure 1: Non-linearity function $f(x) = \frac{1}{1+exp(-x)}$

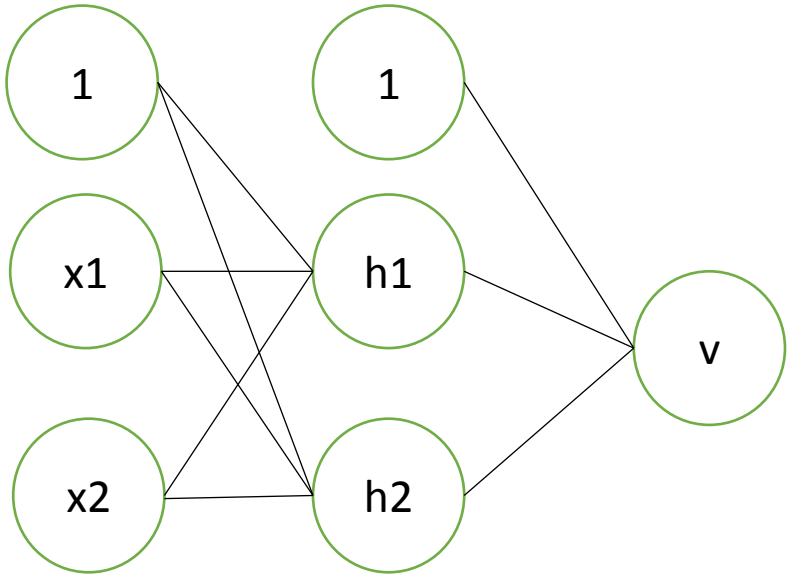
Expressiveness of Neural Network

- P2: Implement $Y = X_1 \wedge X_2$. $Y = X_1 \oplus X_2$ $Y = (A \vee \neg B) \oplus (\neg C \vee \neg D)$
- Sol: Rewrite Y to simpler expression. And then draw the table as before and pick up weight values as you wish.
  - Y = X1 AND X2.
    - Need only one unit as before.
  - Y = X1 XOR X2 = (X1 OR X2) AND (~ (X1 AND X2))
    - Two hidden units: (X1 OR X2), (~ (X1 AND X2))
    - One output unit with non-linearity: AND
  - Y = (A and ~B) xor (~C or ~D) = (A and C and D) or (~B and C and D) or (~A and B and ~C) or (~A and B and ~D)
    - Four hidden units
    - One output unit

Bias Updates in 2-layer Feedforward Neural Network

- Consider the neural network with 2 inputs x1, x2, 2 hidden units h1, h2 and 1 output unit v. We use f to denote the non-linearity. We want to train it using the backpropagation algorithm to minimize squared error on the training set of m data points.

  - $h_1 = f(b_1 * 1 + w_{11}x_1 + w_{12}x_2) = f(s_1)$
  - $h_2 = f(b_2 * 1 + w_{21}x_1 + w_{22}x_2) = f(s_2)$
  - $v = b_v * 1 + u_1 h_1 + u_2 h_2$
  - $l = \sum_m \frac{1}{2}(y_m - v_m)^2 = \sum_m l_m$



Bias Updates in 2-layer Feedforward Neural Network

- P1: Derive the update rule for bv
- Sol: Go backwards
  - $\frac{\partial l}{\partial v} = \sum_m \frac{\partial l_m}{\partial v_m} = \frac{\partial \frac{1}{2}(y_m - v_m)^2}{\partial v_m} = -\frac{1}{2} 2(y_m - v_m) = -e_m$
  - $\frac{\partial v_m}{\partial b_v} = 1$
  - $\frac{\partial l}{\partial b_v} = \sum_m \frac{\partial l_m}{\partial v_m} \frac{\partial v_m}{\partial b_v} = -\sum_m e_m$
  - $b_v \leftarrow b_v + \eta \sum_m e_m$

Bias Updates in 2-layer Feedforward Neural Network

- P2: Derive the update rule for b1, b2
- Sol: go backwards
  - $\frac{\partial l}{\partial v} = \sum_m \frac{\partial l_m}{\partial v_m} = \frac{\partial \frac{1}{2}(y_m - v_m)^2}{\partial v_m} = -\frac{1}{2} 2(y_m - v_m) = -e_m$
  - $\frac{\partial v_m}{\partial h_i} = u_i$
  - $\frac{\partial h_i}{\partial b_i} = \mathrm{f}'(\mathrm{s}_{\mathrm{im}}) = \mathrm{f}'(b_i + w_{i1} x_{1m} + w_{i2} x_{2m})$
  - $\frac{\partial l}{\partial b_i} = -\sum_m e_m u_i f'(s_{im})$
  - $b_i \leftarrow b_i + \eta \sum_m e_m u_i f'(b_i + w_{i1} x_{1m} + w_{i2} x_{2m})$

HW08 PCA

- Let's walk through a PCA example step by step.
- Consider 4 data points in a 2-d feature space: x1 = (-1,1), x2 = (0.5,-0.5), x3 = (1,1), x4 = (-0.5,0.5).

- Q3: Is the data centered?
  - Center O = (x1 + x2 + x3 + x4) / 4 = (0, 0.5), which is not (0, 0). So the data is not centered.

HW08 PCA

- Q4: What is the first principal component?
  - Center data: xi ← xi – O
    - x1 = (-1, 0.5), x2 = (0.5, -1), x3 = (1, 0.5), x4 = (-0.5, 0)
  - Compute “covariance”:
    - X = [-1, 0.5; 0.5, -1; 1, 0.5; -0.5, 0]
    - X’ * X = [2.5, -0.5; -0.5, 1.5]
  - Decompose X’*X
    - Eigenvectors v1 = [-0.3827, -0.9239] v2 = [-0.9239, 0.3827]
    - Corresponding eigenvalues 1.2929, 2.7071
  - First principal component: v2 = [-0.9239, 0.3827]

HW08 PCA

- Q5: If we project all points into the 1-d subspace defined by the second principal component, what is the variance of the project data? (round up to 4th decimal)
  - Project to second principal component
    - xi * v1 = [-0.0793, 0.7325, -0.8446, 0.1913]
  - Compute variance
    - $\frac{1}{4}\sum_i x_i v_1$=0.3232

HW09 Kernel Perceptron

- Consider the quadratic kernel $K(x, z) = (1 + x \cdot z)^2$,assume we are given 1000 training examples, each example is a 100-dimensional vector.
- Q3: We use equation 11.1 in CIML to map the features space for the quadratic kernel. What is the dimension of the feature space that K implicitly maps input vectors to?
- Sol:
  - 1 + 100 + 100 * 100 = 10101
  - (D=100 in 11.1 here)

$$\begin{aligned}\phi(x) = \langle & 1, 2x_1, 2x_2, 2x_3, \ldots, 2x_D, \\ & x_1^2, x_1x_2, x_1x_3, \ldots, x_1x_D, \\ & x_2x_1, x_2^2, x_2x_3, \ldots, x_2x_D, \\ & x_3x_1, x_3x_2, x_3^2, \ldots, x_2x_D, \\ & \ldots, \\ & x_Dx_1, x_Dx_2, x_Dx_3, \ldots, x_D^2\rangle \end{aligned} \tag{11.1}$$

HW09 Kernel Perceptron

- Q4: We explicitly apply the feature mapping to training data, and use the regular perceptron training algorithm on the resulting examples
  How many parameters do we need to estimate during training?
- Sol:
  - Explicit mapping to 10101 dimensional space, we need to estimate weight of this size, plus the bias, i.e. 10101 + 1 = 10102
- Q5: When computing the activation function for one example, what is the dimension of the vectors in the dot product?
- Sol:
  - After explicit mapping, the dimension is raised to 10101

HW09 Kernel Perceptron

- Q6: Now we use the kernel function k, to map our data to the higher dimensional space implicitly and use the kernelized perceptron algorithm for training. How many parameters do we need to estimate during training?
  - Sol
    - To apply kernel trick, weights are represented as linear combinations of 1000 training data points. The coefficients of this representation are to be learnt. Plus the bias, we get 1000 + 1 = 1001 parameters to estimate.
- Q7: When computing the activation function for one example, what is the dimension of the vectors in the dot product?
  - Sol:
    - As the mapping is implicit, we don’t need to compute dot product at high dimension. The dot product is computed using the kernel function, in which the computation is done at the original dimension, i.e. 100.