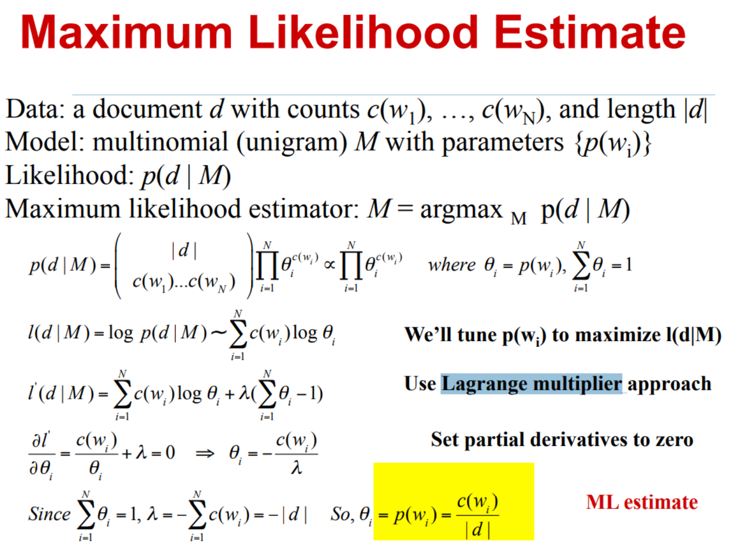
1. Uni-gram language model and Maximum Likelihood Estimation (1%)

The language model is a highly related application of machine learning. For a language model, the uni-gram is a classical one, which ignores the correlation between words and treats the distribution of all words independently. For a document d consisting N types of words, w_1, ..., w_N, each word w_i corresponds to their counts $c(w_i)$, and its length $|d|$ is equivalent to $\sum_{i=1}^{N} c(w_i)$. Based on the uni-gram model, we can make the probability of any sampled document d be modeled as a multinomial distribution, shown as follows:

$$P(d|\theta_1, \ldots, \theta_N) = \binom{|d|}{c(w_1)\ldots c(w_N)} \prod_{i=1}^{N} \theta_i^{c(w_i)}$$

where θ_i represents the probability of w_i, that is, $\theta_i = P(w_i)$. Generally, we would not know the real value of $P(w_i)$ so that we need to estimate it from the sampled document d with the constraint, $\sum_{i=1}^{N} \theta_i = 1$, since we set θ_i as a probability. The Maximum Likelihood Estimation (MLE) method estimates θ_i by deriving $\text{argmax}_{\theta_1, \ldots, \theta_N}\{P(d|\theta_1, \ldots, \theta_N)\}$. Please show the detailed derivation of $\theta_i = \frac{c(w_i)}{|d|}$ from MLE. (Hint: You might need to use the Lagrange multiplier.)

Sol:



Maximum Likelihood Estimate

Data: a document d with counts $c(w_1)$, …, $c(w_N)$, and length $|d|$

Model: multinomial (unigram) M with parameters $\{p(w_i)\}$

Likelihood: $p(d \mid M)$

Maximum likelihood estimator: $M = \text{argmax}_M \; p(d \mid M)$

$$p(d \mid M) = \binom{|d|}{c(w_1)\ldots c(w_N)} \prod_{i=1}^{N} \theta_i^{c(w_i)} \propto \prod_{i=1}^{N} \theta_i^{c(w_i)} \quad \textit{where } \theta_i = p(w_i), \sum_{i=1}^{N} \theta_i = 1$$

$$l(d \mid M) = \log p(d \mid M) \sim \sum_{i=1}^{N} c(w_i) \log \theta_i$$

We'll tune p(w$_i$) to maximize l(d|M)

$$l'(d \mid M) = \sum_{i=1}^{N} c(w_i) \log \theta_i + \lambda\left(\sum_{i=1}^{N} \theta_i - 1\right)$$

Use Lagrange multiplier approach

$$\frac{\partial l'}{\partial \theta_i} = \frac{c(w_i)}{\theta_i} + \lambda = 0 \quad \Rightarrow \quad \theta_i = -\frac{c(w_i)}{\lambda}$$

Set partial derivatives to zero

$$\textit{Since } \sum_{i=1}^{N} \theta_i = 1, \; \lambda = -\sum_{i=1}^{N} c(w_i) = -|d| \quad \textit{So, } \theta_i = p(w_i) = \frac{c(w_i)}{|d|}$$

ML estimate

2. LSTM Cell (1%)

In this exercise, we will simulate the forward pass of a simple LSTM cell. Figure.1 shows a single LSTM cell, where z is the cell input, z_i, z_f, z_o are the control inputs of the gates, c is the cell memory, and f, g, h are activation functions. Given an input x, the cell input and the control inputs can be calculated by the following equations :

- $z = w \cdot x + b$
- $z_i = w_i \cdot x + b_i$
- $z_f = w_f \cdot x + b_f$
- $z_o = w_o \cdot x + b_o$

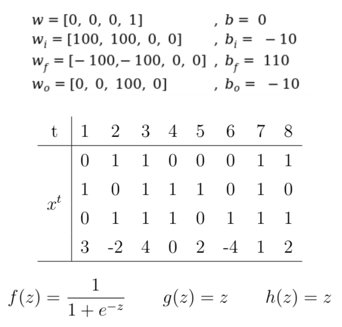
Where w, w_i, w_f, w_o are weights and b, b_i, b_f, b_o are biases. The final output can be calculated by

$y = f(z_o)\,h(c')$

where the value stored in cell memory is updated by

$c' = f(z_i)g(z) + cf(z_f)$

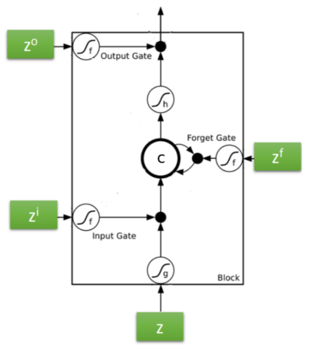
Given an input sequence x^t $(t = 1, 2, \ldots, 8)$, please derive the output sequence y_t. The input sequence, the weights, and the activation functions are provided below. The initial value in cell memory is 0. **Please note that your calculation process is required to receive full credit.**



$w = [0,\ 0,\ 0,\ 1]$, $b = 0$
$w_i = [100,\ 100,\ 0,\ 0]$, $b_i = -10$
$w_f = [-100, -100,\ 0,\ 0]$, $b_f = 110$
$w_o = [0,\ 0,\ 100,\ 0]$, $b_o = -10$

t	1	2	3	4	5	6	7	8
	0	1	1	0	0	0	1	1
x^t	1	0	1	1	1	0	1	0
	0	1	1	1	0	1	1	1
	3	-2	4	0	2	-4	1	2

$f(z) = \dfrac{1}{1+e^{-z}}$ $\qquad g(z) = z$ $\qquad h(z) = z$

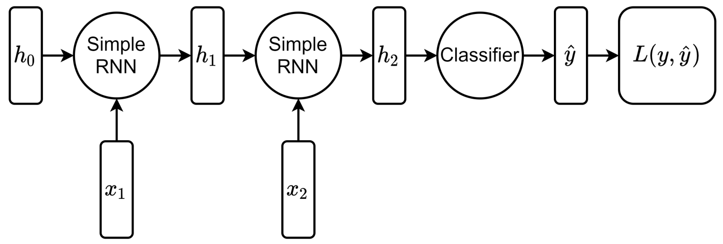


Sol:

t	1	2	3	4	5	6	7	8
z_i	90	90	190	90	90	-10	190	90
$f(z_i)$	1	1	1	1	1	0	1	1
z_f	10	10	-90	10	10	110	-90	10
$f(z_i)$	1	1	0	1	1	1	0	1
z_o	-10	90	90	90	-10	90	90	90
$f(z_o)$	0	1	1	1	0	1	1	1
z	3	-2	4	0	2	-4	1	2
c'	3	1	4	4	6	6	1	3
y	0	1	4	4	0	6	1	3

3. Backpropagation through time via Simple RNN (1%)

Backpropagation through time is a critical concept to know as we train a recurrent network. Here, we set a toy case of binary classification problem.



The Simple RNN module has two kinds of weights, W_i and W_h, such that $h_t = \tanh(W_i x_t + W_h h_{t-1})$, where t represents the index of steps. The Classifier module has the weight W_o such that $\hat{y} = \sigma(W_o h_2)$, where $\sigma(W_o h_2) = \frac{1}{1+\exp(-W_o h_2)}$. The initial state h_0 is set to be 0. The sequential input only contains $\{x_1, x_2\}$; the label is y;

the objective function is binary cross entropy. Please derive $\frac{\partial L(y,\hat{y})}{\partial W_o}$, $\frac{\partial L(y,\hat{y})}{\partial W_h}$, $\frac{\partial L(y,\hat{y})}{\partial W_i}$ in terms of x_1, x_2, h_0, h_1, h_2, W_i, W_o, and W_h.
Sol:
ps: Hadamard and Kronecker product operations can be replaced with standard multiplication while the inputs are single variables.
Since Hadamard and Kronecker product operations are not in our course range, we decided to simplify this multivariate problem as a single-variable problem (The dimension of the inputs are only one). That is, you will get full grades if your answer is almost correct, but only the multiplication parts are wrong.
Let z be $W_o h_2$. By the chain rule, we can derive the results as follows:
$\frac{\partial L(y,\hat{y})}{\partial W_o} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial W_o}$
$\frac{\partial L(y,\hat{y})}{\partial W_h} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial h_2} \odot (\frac{\partial h_2}{\partial h_1} \odot \frac{\partial h_1}{\partial W_h} + \frac{\partial h_2}{\partial W_h})$
$\frac{\partial L(y,\hat{y})}{\partial W_i} = \frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial h_2} \odot (\frac{\partial h_2}{\partial h_1} \odot \frac{\partial h_1}{\partial W_i} + \frac{\partial h_2}{\partial W_i})$
where $\odot$ denotes the Hadamard product operation.
(1) $\frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z}$:
$\frac{\partial L(y,\hat{y})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = -[y \cdot \frac{1}{\hat{y}} + (1-y) \cdot \frac{-1}{1-\hat{y}}] \cdot \hat{y}(1-\hat{y}) = \hat{y} - y$
(2) $\frac{\partial z}{\partial W_o}$:
$\frac{\partial z}{\partial W_o} = h_2^T$
By (1)(2), we can get $\frac{\partial L(y,\hat{y})}{\partial W_o} = (\hat{y} - y)h_2^T$
(3) $\frac{\partial z}{\partial h_2}$: W_o^T
(4) $\frac{\partial h_2}{\partial h_1}$:
By taking the result that $\frac{\partial \tanh(x)}{\partial x} = \mathrm{sech}^2(x)$, we can get
$\frac{\partial h_2}{\partial h_1} = (\mathrm{sech}^2(W_i x^2 + W_h h_1) \otimes 1) \odot W_h$, where $\otimes$ denotes the Kronecker product opereation.
(5) $\frac{\partial h_1}{\partial W_h}$:
$\frac{\partial h_1}{\partial W_h} = \mathrm{sech}^2(W_i x_1 + W_h h_0) \otimes h_0$, given that $h_0 = 0$, then $\frac{\partial h_1}{\partial W_h} = 0$
(6) $\frac{\partial h_2}{\partial W_h}$:
$\frac{\partial h_2}{\partial W_h} = \mathrm{sech}^2(W_i x_2 + W_h h_1) \otimes h_1$
By (1)(3)(4)(5)(6), we can get
$\frac{\partial L(y,\hat{y})}{\partial W_h} = (\hat{y} - y)W_o^T \odot (\mathrm{sech}^2(W_i x_2 + W_h h_1) \otimes h_1)$.
(7) $\frac{\partial h_1}{\partial W_i}$:
$\frac{\partial h_1}{\partial W_i} = \mathrm{sech}^2(W_i x_1 + W_h h_0) \otimes x_1$
(8) $\frac{\partial h_2}{\partial W_i}$:
$\frac{\partial h_2}{\partial W_i} = \mathrm{sech}^2(W_i x_2 + W_h h_1) \otimes x_2$
By (1)(3)(4)(7)(8), we can get
$\frac{\partial L(y,\hat{y})}{\partial W_i} = (\hat{y} - y)W_o^T \odot (((\mathrm{sech}^2(W_i x^2 + W_h h_1) \otimes 1) \odot W_h) \odot (\mathrm{sech}^2(W_i x_1 + W_h h_0) \otimes x_1) + \mathrm{sech}^2(W_i x_2 + W_h h$
.

4. Multiclass AdaBoost (1%)

Let X be the input space, F be a collection of multiclass classifiers that map from X to $[1, K]$, where K denotes the number of classes. Let $\{(x_i, \hat{y}_i)\}_{i=1}^n$ be the training data set, where $x_i \in R^m$ and $\hat{y}_i \in [1, K]$. Given $T \in N$, suppose we want to find functions

$$g_T^k(x) = \sum_{t=1}^{T} \alpha_t f_t^k(x), \quad k \in [1, K]$$

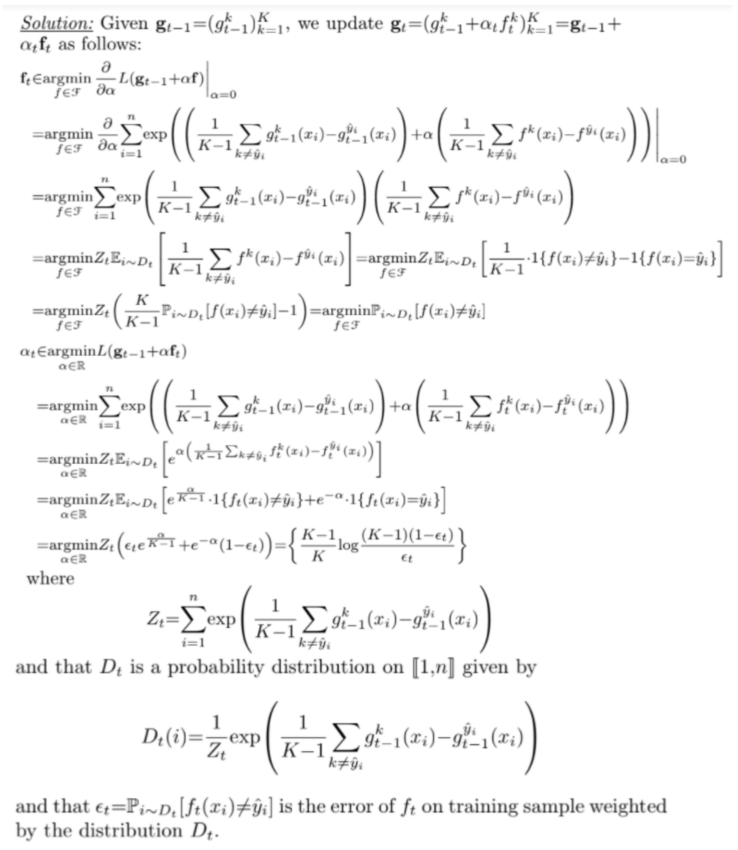
where $f_t \in F$ and $\alpha_t \in R$ for all $t \in [1, T]$. Here for $f \in F$, we denote $f^k(x) = \{f(x) = k\}$ as the k-th element in the one-hot representation of $f(x) \in [1, K]$. The aggregated classifier $h : X \to [1, K]$ is defined as

$$h(x) = \operatorname*{argmax}_{1 \le k \le K} g_T^k(x)$$

For the multi-class adaBoost, closed form update rules can be derived if for each t, we individually update α_t^k for $k = 1, \ldots, K$, one at a time. Please apply gradient boosting to show how the functions f_t and coefficients α_t^k are computed with an aim to minimize the following loss function

$$L(g_T^1, \ldots, g_T^K) = \sum_{i=1}^{n} \exp\left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} g_T^k(x_i) - g_T^{\hat{y}_i}(x_i) \right)$$

SOL:



Solution: Given $\mathbf{g}_{t-1} = (g_{t-1}^k)_{k=1}^K$, we update $\mathbf{g}_t = (g_{t-1}^k + \alpha_t f_t^k)_{k=1}^K = \mathbf{g}_{t-1} + \alpha_t \mathbf{f}_t$ as follows:

$$\mathbf{f}_t \in \operatorname*{argmin}_{f \in \mathcal{F}} \frac{\partial}{\partial \alpha} L(\mathbf{g}_{t-1} + \alpha \mathbf{f}) \Big|_{\alpha=0}$$

$$= \operatorname*{argmin}_{f \in \mathcal{F}} \frac{\partial}{\partial \alpha} \sum_{i=1}^{n} \exp\left(\left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} g_{t-1}^k(x_i) - g_{t-1}^{\hat{y}_i}(x_i) \right) + \alpha \left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} f^k(x_i) - f^{\hat{y}_i}(x_i) \right) \right) \Big|_{\alpha=0}$$

$$= \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{i=1}^{n} \exp\left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} g_{t-1}^k(x_i) - g_{t-1}^{\hat{y}_i}(x_i) \right) \left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} f^k(x_i) - f^{\hat{y}_i}(x_i) \right)$$

$$= \operatorname*{argmin}_{f \in \mathcal{F}} Z_t \mathbb{E}_{i \sim D_t} \left[\frac{1}{K-1} \sum_{k \ne \hat{y}_i} f^k(x_i) - f^{\hat{y}_i}(x_i) \right] = \operatorname*{argmin}_{f \in \mathcal{F}} Z_t \mathbb{E}_{i \sim D_t} \left[\frac{1}{K-1} \cdot 1\{f(x_i) \ne \hat{y}_i\} - 1\{f(x_i) = \hat{y}_i\} \right]$$

$$= \operatorname*{argmin}_{f \in \mathcal{F}} Z_t \left(\frac{K}{K-1} \mathbb{P}_{i \sim D_t}[f(x_i) \ne \hat{y}_i] - 1 \right) = \operatorname*{argmin}_{f \in \mathcal{F}} \mathbb{P}_{i \sim D_t}[f(x_i) \ne \hat{y}_i]$$

$$\alpha_t \in \operatorname*{argmin}_{\alpha \in \mathbb{R}} L(\mathbf{g}_{t-1} + \alpha \mathbf{f}_t)$$

$$= \operatorname*{argmin}_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \exp\left(\left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} g_{t-1}^k(x_i) - g_{t-1}^{\hat{y}_i}(x_i) \right) + \alpha \left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} f_t^k(x_i) - f_t^{\hat{y}_i}(x_i) \right) \right)$$

$$= \operatorname*{argmin}_{\alpha \in \mathbb{R}} Z_t \mathbb{E}_{i \sim D_t} \left[e^{\alpha \left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} f_t^k(x_i) - f_t^{\hat{y}_i}(x_i) \right)} \right]$$

$$= \operatorname*{argmin}_{\alpha \in \mathbb{R}} Z_t \mathbb{E}_{i \sim D_t} \left[e^{\frac{\alpha}{K-1}} \cdot 1\{f_t(x_i) \ne \hat{y}_i\} + e^{-\alpha} \cdot 1\{f_t(x_i) = \hat{y}_i\} \right]$$

$$= \operatorname*{argmin}_{\alpha \in \mathbb{R}} Z_t \left(\epsilon_t e^{\frac{\alpha}{K-1}} + e^{-\alpha}(1 - \epsilon_t) \right) = \left\{ \frac{K-1}{K} \log \frac{(K-1)(1-\epsilon_t)}{\epsilon_t} \right\}$$

where

$$Z_t = \sum_{i=1}^{n} \exp\left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} g_{t-1}^k(x_i) - g_{t-1}^{\hat{y}_i}(x_i) \right)$$

and that D_t is a probability distribution on $[\![1, n]\!]$ given by

$$D_t(i) = \frac{1}{Z_t} \exp\left(\frac{1}{K-1} \sum_{k \ne \hat{y}_i} g_{t-1}^k(x_i) - g_{t-1}^{\hat{y}_i}(x_i) \right)$$

and that $\epsilon_t = \mathbb{P}_{i \sim D_t}[f_t(x_i) \ne \hat{y}_i]$ is the error of f_t on training sample weighted by the distribution D_t.