4F10 Deep Learning and Structured Data, 2019

1. *Training Logistic Regression and the use of the Hessian*

(a) Linear decision boundaries passing through the origin. [10%]

(b)(i) Differentiating

$$\begin{aligned}\frac{\partial}{\partial \mathbf{b}} P(\omega_1|\mathbf{x},\mathbf{b}) &= \frac{\exp(-\mathbf{b}^{\mathsf{T}}\mathbf{x})}{(1+\exp(-\mathbf{b}^{\mathsf{T}}\mathbf{x}))^2}\mathbf{x} \\ &= P(\omega_1|\mathbf{b},\mathbf{x})(1-P(\omega_1|\mathbf{b},\mathbf{x}))\mathbf{x}\end{aligned}$$

Thus

$$\begin{aligned}\frac{\partial}{\partial \mathbf{b}} \mathcal{L}(\mathbf{b}) &= \sum_{i=1}^{n} \mathbf{x}_i \left(y_i(1-P(\omega_1|\mathbf{b},\mathbf{x}_i)) - (1-y_i)P(\omega_1|\mathbf{b},\mathbf{x}_i)\right) \\ &= \sum_{i=1}^{n} \mathbf{x}_i \left(y_i - P(\omega_1|\mathbf{b},\mathbf{x}_i)\right)\end{aligned}$$

This can be used in a gradient style approach where

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + \eta \left.\frac{\partial}{\partial \mathbf{b}} \mathcal{L}(\mathbf{b})\right|_{\mathbf{b}^{(k)}}$$

[25%]

(b)(ii) Element j,k of the Hessian is

$$h_{jk} = \frac{\partial^2}{\partial b_j \partial b_k} \mathcal{L}(\mathbf{b})$$

Using the above expression

$$\frac{\partial}{\partial b_j}\left(\sum_{i=1}^{n} \left(y_i - P(\omega_1|\mathbf{b},\mathbf{x}_i)\right) x_{ik}\right) = -\sum_{i=1}^{n} P(\omega_1|\mathbf{b},\mathbf{x}_i)(1-P(\omega_1|\mathbf{b},\mathbf{x}_i)) x_{ij} x_{ik}$$

This can be expressed in matrix form as

$$\mathbf{H} = -\left[\mathbf{x}_1,\ldots,\mathbf{x}_n\right] \begin{bmatrix} P(\omega_1|\mathbf{b},\mathbf{x}_1)(1-P(\omega_1|\mathbf{b},\mathbf{x}_1)) & \ldots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \ldots & P(\omega_1|\mathbf{b},\mathbf{x}_n)(1-P(\omega_1|\mathbf{b},\mathbf{x}_n)) \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{bmatrix}^{\mathsf{T}}$$

Thus

$$\begin{aligned}\mathbf{S} &= \left[\mathbf{x}_1,\ldots,\mathbf{x}_n\right]^{\mathsf{T}} \\ \mathbf{R} &= \begin{bmatrix} P(\omega_1|\mathbf{b},\mathbf{x}_1)(1-P(\omega_1|\mathbf{b},\mathbf{x}_1)) & \ldots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \ldots & P(\omega_1|\mathbf{b},\mathbf{x}_n)(1-P(\omega_1|\mathbf{b},\mathbf{x}_n)) \end{bmatrix}\end{aligned}$$

[25%]

(b)(iii) The Hessian may be used for optimisation as

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} - \mathbf{H}^{-1} \frac{\partial}{\partial \mathbf{b}} \mathcal{L}(\mathbf{b}) \Big|_{\mathbf{b}^{(k)}}$$

where the Hessian is evaluated at the $\mathbf{b}^{(k)}$. Should discuss

- No need to compute η which is a major issue with gradient descent
- If error surface is quadratic - straight to solution
- Hessian may involve computing a large number of parameters (if feature-space is large).

[15%]

(b)(iv) The Hessian is negative-definite for this problem. This implies that the error function is a concave function so has a unique maximum. [10%]

(c) The first derivative of the regularisation term is simply

$$\frac{\partial \mathcal{R}}{\partial \mathbf{b}} = 2\lambda \mathbf{b}$$

The Hessian then has the form of

$$\frac{\partial^2 \mathcal{R}}{\partial \mathbf{b}^2} = 2\lambda \mathbf{I}$$

This is the simplest form of L_2 regularisation where network parameters are regularised to zero, discouraging the network from training large weights.

Note for this to be sensible $\lambda < 0$ so that there is a penalty in having large weights. [15%]

Comment: This question examined training of a logistic regression classifier using first and second order approaches. In general the students showed a good understanding of the underlying theory of the optimisation approaches, but there were a range of mistakes in deriving the form of Hessian given. It was disappointing that few students could state the implications on optimisation for the form of Hessian given.

2. *Sequence Data with Deep Learning*

(a) A standard network configuration

- output needs to K-dimensional where each element is associated with one of the sentiment classes;
- a soft-max activation function should be used for the last layer of the network so it has the form of posterior class distribution;
- multiple layers should be used to allow for non-linear decision boundaries;
- any form of non-linear activation function valid for other layers.

The form of network uses the standard training criterion is cross entropy

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{k=1}^{K} \delta(\omega_k, y^{(i)}) \log(t^{(i)})$$

where $\mathbf{t}^{(i)}$ is the final later output for review i. [25%]

(b)(i) The important aspect of this model is map the variable length sequence (the review) into a fixed-length vector. The easiest way (described in lectures) is to use the final hidden unit of the sequence for $\mathbf{h}$. The equations associated with this are:

$$\mathbf{h}_t = \mathbf{f}(\mathbf{h}_{t-1}, \mathbf{x}_t^{(i)}) = \boldsymbol{\phi}(\mathbf{W}_{\mathtt{r}}\mathbf{h}_{t-1} + \mathbf{W}_{\mathtt{f}}\mathbf{x}_t^{(i)} + \mathbf{b})$$

$\boldsymbol{\phi}$ can be any form of non-linear activation function. Then the output is given by

$$\mathbf{h} = \mathbf{h}_L$$

for the L-length sequence. An additional non-linear transform prior to the recurrent layer can improve performance (maps the standard word-embedding into a "better" space for the recurrent network to operate. [20%]

(b)(ii) For a bidirectional model there is a backward recurrency as well. Here:

$$\tilde{\mathbf{h}}_t = \mathbf{f}(\tilde{\mathbf{h}}_{t+1}, \mathbf{x}_t^{(i)}) = \boldsymbol{\phi}(\tilde{\mathbf{W}}_{\mathtt{r}}\tilde{\mathbf{h}}_{t+1} + \tilde{\mathbf{W}}_{\mathtt{f}}\mathbf{x}_t^{(i)} + \tilde{\mathbf{b}})$$

The form of output then can be written as

$$\mathbf{h} = \begin{bmatrix} \mathbf{h}_L \\ \tilde{\mathbf{h}}_1 \end{bmatrix}$$

[15%]

(c) Using the standard additive form of attention the following equations can be used to describe attention

$$\begin{aligned} \mathbf{h} &= \sum_{j=1}^{L} \alpha_j \mathbf{x}_j^{(i)} \\ \alpha_j &= \frac{\exp(e_j^{(i)})}{\sum_{k=1}^{L} \exp(e_k^{(i)})} \\ e_j^{(i)} &= \boldsymbol{\phi}(\mathbf{W}\mathbf{x}_j^{(i)} + \mathbf{b}) \end{aligned}$$

This is a self-attention mechanism that yields a PMF over the L elements. Again a non-linearity prior to the attention mechanism is useful to mention. [20%]

(d) Answer should include

- (b)(i) is the simplest, but is biased to elements at the end of the sequence.
- (b)(ii) removes the bias to the end, but now middle elements of the sequence are deweighted. Requires more model parameters than the simple network.
- (c) allows all elements to be used, but only a simple key is used. Effectively each word is treated in isolation (depends on whether a recurrent network is included in the attention mechanism network).
- For all the recurrent network models these could be improved for long sequences by using LSTMs or GRUs.

[20%]

Comment: This questions examined how a deep neural network can be configured to classify sequence data. This was the least popular question. The question was well answered with students showing a good knowledge of recurrent networks and attention mechanisms, and the limitations of the approaches.

3. *Mixture Models and the Exponential Family*

(a) Z must satisfy

$$Z = \int \exp\left(\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{f}(\mathbf{x})\right) dx$$

this ensures that the PDF integrates to 1. [10%]

(b) It is possible to rewrite the expression as

$$\begin{aligned} p(\mathbf{x}|\boldsymbol{\mu}) &= \frac{1}{Z}\prod_{i=1}^{d}\mu_i^{x_i} \\ &= \frac{1}{Z}\exp\left(\sum_{i=1}^{d} x_i \log(\mu_i)\right) \end{aligned}$$

By analogy with the exponential distribution

$$\lambda_i = -\log(\mu_i)$$

The values are then

$$\begin{aligned} \boldsymbol{\alpha} &= \begin{bmatrix} \log(\mu_1) \\ \vdots \\ \log(\mu_d) \end{bmatrix} \\ \mathbf{f}(\mathbf{x}) &= \mathbf{x} \\ Z &= \frac{1}{\prod_{i=1}^{d}\left(-\log(\mu_d)\right)} \end{aligned}$$

[There is a standard additional restriction, not mentioned in the question, that the solution will satisfy $0 < \mu < 1$ to yield a valid form for the exponential distribution.] [30%]

(c)(i) The expression for the log-likelihood is

$$l(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\log\left(\sum_{m=1}^{M} c_m \frac{1}{Z_m}\exp\left(\boldsymbol{\alpha}_m^{\mathsf{T}}\mathbf{f}(x_i)\right)\right)$$

[15%]

(c)(ii) Substituting in the expression for the exponential family

$$\begin{aligned} \mathcal{Q}(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) &= \sum_{i=1}^{n}\sum_{m=1}^{M} P(\omega_m|\mathbf{x}_i, \boldsymbol{\alpha})\left[-\log(\hat{Z}_m) + \hat{\boldsymbol{\alpha}}_m^{\mathsf{T}}\mathbf{f}(\mathbf{x}_i)\right] \\ &= \sum_{m=1}^{M}\left(-\log(\hat{Z}_m)\left(\sum_{i=1}^{n} P(\omega_m|\mathbf{x}_i, \boldsymbol{\alpha})\right) + \hat{\boldsymbol{\alpha}}_m^{\mathsf{T}}\left(\sum_{i=1}^{n} P(\omega_m|\mathbf{x}_i, \boldsymbol{\alpha})\mathbf{f}(\mathbf{x}\right)\right) \end{aligned}$$

Sufficient statistics for auxiliary function is simply

$$\sum_{i=1}^{n} P(\omega_m|x_i, \boldsymbol{\alpha}); \quad \sum_{i=1}^{n} P(\omega_m|x_i, \boldsymbol{\alpha})\mathbf{f}(\mathbf{x}_i)$$

for each of the components of the model. [25%]

(c)(ii) Differentiating the auxiliary function yields

$$\begin{aligned}\frac{\partial}{\partial \hat{\boldsymbol{\alpha}}_m}\mathcal{Q}(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) &= \sum_{i=1}^{N} P(\omega_m|\mathbf{x}_i, \boldsymbol{\alpha})\frac{\partial}{\partial \hat{\boldsymbol{\alpha}}_m}\left(\log(c_m) - \log(Z_m) + \hat{\boldsymbol{\alpha}}_m^{\mathsf{T}}\mathbf{f}(\mathbf{x}_i)\right) \\ &= \sum_{i=1}^{N} P(\omega_m|\mathbf{x}_i, \boldsymbol{\alpha})\left[-\frac{1}{Z_m}\frac{\partial}{\partial \hat{\boldsymbol{\alpha}}_m}Z_m + \mathbf{f}(\mathbf{x}_i)\right]\end{aligned}$$

In the general case the normalisation term will not be linear, so general optimisation approaches are required, for example gradient descent. [20%]

Comment: This questions examined the students' knowledge of probability density functions (the exponential family) and the EM algorithm. This was the most popular question. Very few students were able to derive the statistics required to optimise the auxiliary function, many students simply described the general process for EM.

4. *Support Vector Machines and Kernels*

a) Many linear machine learning algorithms operate on data using dot products between data points. You can obtain a non-linear version of the algorithm by replacing each dot product $\mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m$ in the algorithm with $k(\mathbf{x}_n, \mathbf{x}_m)$, where $k(\cdot, \cdot)$ is a kernel which implicitly computes the dot product in a higher-dimensional space. [15%]

b)

i) The constraints are $t_n\mathbf{w}^{\mathrm{T}}\mathbf{x}_n \geq 1$ for $n = 1, \ldots, N$. [10%]

ii) The resulting Lagrangian is

$$L(\mathbf{w}, a_1, \ldots, a_N) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{n=1}^{N} a_n\{t_n\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - 1\}\,.$$

[10%]

iii) The expression is obtained by equating the gradient of $L(\mathbf{w}, a_1, \ldots, a_N)$ to zero and then solving for $\mathbf{w}$:

$$\frac{\partial L(\mathbf{w}, a_1, \ldots, a_N)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} a_n t_n \mathbf{x}_n = 0 \Leftrightarrow \mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n\,.$$

[15%]

c) [As no bias is given the form of algorithm given in the question just terminates at the trivial solution $\mathbf{w} = 0$. This did not cause any confusion.]

i) The constraints are $t_n\mathbf{w}^{\mathrm{T}}\mathbf{x}_n \geq 0$ for $n = 1, \ldots, N$. The constraints for the perceptron are similar to the SVM ones, enforcing that the classifier separates the data, but now there is no constraint on the scale of $\mathbf{w}$. The SVM is preferred because it finds a classifier that separates the data but which has highest margin (by minimizing $0.5||\mathbf{w}||^2$ subject to scale constraints), which often results in improved predictive accuracy. [20%]

ii) The initial value of $\mathbf{w}$ is zero. After that, whenever $\mathbf{w}$ makes a mistake classifying $\mathbf{x}_n$, we add $t_n\mathbf{x}_n$ to $\mathbf{w}$. Therefore, at convergence, $\mathbf{w}$ satisfies $\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n$, where a_n is the number of mistakes that the algorithm made when classifying $\mathbf{x}_n$ during its execution. [15%]

iii) The pseudocode is

Algorithm 1 Kernelised Perceptron

Input: Dataset $\{(\mathbf{x}_n, t_n)\}_{n=1}^N$ **Output**: $a_1, \ldots, a_N$
$a_1, \ldots, a_N \leftarrow 0$
repeat
    Errors $\leftarrow$ False
    for $n = 1, \ldots, N$ **do**
        if $t_n \left(\sum_{m=1}^N a_m t_m k(\mathbf{x}_m, \mathbf{x}_n) \right) < 0$ **then**
            $a_n \leftarrow a_n + 1$
            Errors $\leftarrow$ True
until Not Errors

Predictions are now done using $y(\mathbf{x}) = \sum_{n=1}^N a_n t_n k(\mathbf{x}_n, \mathbf{x})$ [15%]

Comment: This question examined Support Vector Machines (SVMs) and the impact of kernels. The students showed a good understanding of the constraints associated with the SVM, and the constraint required for the perceptron algorithm to terminate. Very few students could obtain the correct form of the kernelised perceptron algorithm, many simply replaced the standard perceptron algorithm with one where the transformed features.