4F10 Deep Learning and Structured Data, 2022

1. *EM algorithm and HMMs*

   (a) Log-likelihood of the training data is

$$\begin{aligned}\log(P(x_1,\ldots,x_n|\lambda_1,\ldots,\lambda_M)) &= \log\left(\sum_{\boldsymbol{\theta}\in\boldsymbol{\Theta}}\left(\prod_{i=1}^{n}a_{\theta_{i-1}\theta_i}\right)\left(\prod_{i=1}^{n}\sum_{m=1}^{M}c_m\lambda_m^{x_i}(1-\lambda_m)^{(1-x_i)}\right)\right)\\ &= \sum_{i=1}^{n}\log\left(\sum_{m=1}^{M}c_m\lambda_m^{x_i}(1-\lambda_m)^{(1-x_i)}\right)\end{aligned}$$

   where $\boldsymbol{\Theta}$ is the set of all possible state sequences, the simplification is not necessary to get the marks, but simplifies the answer to Part (c)(i). [15%]

   (b)(i) EM is an iterative approach to estimating the model parameters. Given the current estimates of the model parameters, $\boldsymbol{\lambda}$, the new estimates, $\hat{\boldsymbol{\lambda}}$, are found using

   - Compute component posteriors, $P(\omega_m|x_i,\boldsymbol{\lambda})$, using current parameters.
   - Using the Auxiliary function, $\mathcal{Q}(\boldsymbol{\lambda},\hat{\boldsymbol{\lambda}})$, compute the new parameters.

   [15%]

   (b)(ii) Substituting in the expression for the likelihood to the auxiliary function

$$\mathcal{Q}(\boldsymbol{\lambda},\hat{\boldsymbol{\lambda}}) = \sum_{i=1}^{n}\sum_{m=1}^{M}P(\omega_m|x_i,\boldsymbol{\lambda})\left(x_i\log(\hat{\lambda}_m)+(1-x_i)\log(1-\hat{\lambda}_m)\right)$$

   Differentiate this with respect to $\hat{\lambda}_q$ give

$$\frac{\partial\mathcal{Q}(\boldsymbol{\lambda},\hat{\boldsymbol{\lambda}})}{\partial\hat{\lambda}_q} = \sum_{i=1}^{n}P(\omega_q|x_i,\boldsymbol{\lambda})\left[\frac{x_i}{\hat{\lambda}_q}-\frac{(1-x_i)}{(1-\hat{\lambda}_q)}\right]$$

   Equating to zero gives

$$(1-\hat{\lambda}_q)\sum_{i=1}^{n}P(\omega_q|x_i,\boldsymbol{\lambda})x_i = \hat{\lambda}_q\sum_{i=1}^{n}P(\omega_q|x_i,\boldsymbol{\lambda})(1-x_i)$$

   Rearranging yields

$$\hat{\lambda}_q = \frac{\sum_{i=1}^{n}P(\omega_q|x_i,\boldsymbol{\lambda})x_i}{\sum_{k=1}^{n}P(\omega_j|x_k,\boldsymbol{\lambda})}$$

   [30%]

(c)(i) Without the additional observation z, the distribution associated with each state does not depend in the state, hence the simplification in Part (a). This means that the model in Part (a) doe s not models sequences, it's a simple GMM. [15%]

(c)(ii) Again EM is used. Plugging in the expressions yields

$$Q(\boldsymbol{\lambda}, \hat{\boldsymbol{\lambda}}) = K + \sum_{i=1}^{n}\sum_{j=1}^{J}\sum_{m=1}^{M} P(\mathtt{s}_j, \omega_m | x_i, z_i, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \log(P(x_i, z_i | \mathtt{s}_j, \omega_m, \hat{\lambda}_m, \hat{\alpha}_j))$$

This can then be split into the part involving $\boldsymbol{\alpha}$

$$\begin{aligned} Q(\boldsymbol{\lambda}, \hat{\boldsymbol{\lambda}}) &= K^\star + \sum_{i=1}^{n}\sum_{j=1}^{J}\sum_{m=1}^{M} P(\mathtt{s}_j, \omega_m | x_i, z_i, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \left(z_i \log(\hat{\alpha}_j) + (1 - z_i)\log(1 - \hat{\alpha}_j)\right) \\ &= K^\star + \sum_{i=1}^{n}\sum_{j=1}^{J} P(\mathtt{s}_j | x_i, z_i, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \left(z_i \log(\hat{\alpha}_j) + (1 - z_i)\log(1 - \hat{\alpha}_j)\right) \end{aligned}$$

Thus the update rule has a similar form

$$\hat{\alpha}_j = \frac{\sum_{i=1}^{n} P(\mathtt{s}_j | x_i, z_i, \boldsymbol{\lambda}, \boldsymbol{\alpha}) z_i}{\sum_{i=1}^{n} P(\mathtt{s}_j | x_i, z_i, \boldsymbol{\lambda}, \boldsymbol{\alpha})}$$

This can be further simplified as the state posterior isn't impacted by x_i. [25%]

Comments This question examined Hidden Markov Models (HMM) and the use of the EM algorithm. It was disappointing that the standard form of the log-likelihood of the HMM could not be derived by some candidates. The use and description of the EM algorithm was generally good, with many students being able to derive the required update formulae. Very few students commented on the fact that the original state output distribution was not a function of the HMM state, resulting in the model being a GMM rather than an HMM.

2. *Neural Network Training*

(a)(i) The values are

$$\begin{aligned} \mathbf{b} &= \boldsymbol{\nabla} E(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^{(\tau)}} \\ \mathbf{A} &= \boldsymbol{\nabla}^2 E(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}^{(\tau)}} \end{aligned}$$

the gradient and the Hessian respectively at the current model parameters. [15%]

(a)(ii) Letting

$$\boldsymbol{\Delta\theta}^{(\tau)} = (\boldsymbol{\theta} - \boldsymbol{\theta}^{(\tau)})$$

gives the following differential expression

$$\frac{\partial}{\partial \boldsymbol{\Delta\theta}^{(\tau)}} E(\boldsymbol{\theta}) = \mathbf{b} + \mathbf{A}\boldsymbol{\Delta\theta}^{(\tau)})$$

Equating this to zero gives

$$\boldsymbol{\Delta\theta}^{(\tau)} = -\mathbf{A}^{-1}\mathbf{b}$$

Thus the value is given by

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(\tau)} - \mathbf{A}^{-1}\mathbf{b}$$

This is only true if the Hessian is positive definite

$$\mathbf{v}^{\mathsf{T}}\mathbf{H}\mathbf{v} > 0$$

This needs to be checked prior to updating the model parameters. If the Hessian is not positive-definite need to back-off to, for example, standard gradient descent approaches. [25%]

(b)(i) Computing the Hessian for the set-up

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_j} = \sum_{i=1}^{N} (f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)}) \frac{\partial f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})}{\partial \theta_j}$$

and

$$\frac{\partial^2 E(\boldsymbol{\theta})}{\partial \theta_j \partial \theta_k} = \sum_{i=1}^{N} \frac{\partial f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})}{\partial \theta_k} \frac{\partial f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})}{\partial \theta_j} + \sum_{i=1}^{N} (f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) - y^{(i)}) \frac{\partial^2 f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})}{\partial \theta_j \partial \theta_k}$$

For the conditions described the network will train so that

$$f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) = y^{(i)}$$

When $\boldsymbol{\theta}$ is close to a solution, and the resulting predictions are accurate, then the second term will be close to zero and the approximation is good [20%]

(b)(ii) From the approximation given, the recursion for $\mathbf{A}_n$ can be expressed as

$$\mathbf{A}_n = \mathbf{A}_{n-1} + \mathbf{g}^{(n)}(\mathbf{g}^{(n)})^\mathsf{T}; \quad \mathbf{g}^{(n)} = \left.\frac{\partial f(\boldsymbol{x}^{(n)};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}^{(\tau)}}$$

Using the inverse equality given in the question

$$\begin{aligned}
\mathbf{A}_n^{-1} &= \left(\mathbf{A}_{n-1} + \mathbf{g}^{(n)}(\mathbf{g}^{(n)})^\mathsf{T}\right)^{-1} \\
&= \mathbf{A}_{n-1}^{-1} - \mathbf{A}_{n-1}^{-1}\mathbf{g}^{(n)}\left(1 + \mathbf{g}^{(n)\mathsf{T}}\mathbf{A}_{n-1}^{-1}(\mathbf{g}^{(n)})\right)^{-1}(\mathbf{g}^{(n)})^\mathsf{T}\mathbf{A}_{n-1}^{-1} \\
&= \mathbf{A}_{n-1}^{-1} - \frac{\mathbf{A}_{n-1}^{-1}\mathbf{g}^{(n)}(\mathbf{g}^{(n)})^\mathsf{T}\mathbf{A}_{n-1}^{-1}}{1 + \mathbf{g}^{(n)\mathsf{T}}\mathbf{A}_{n-1}^{-1}(\mathbf{g}^{(n)})}
\end{aligned}$$

[20%]

(c) There are a number of issues with using the approximation in Part (a)

i. The evaluation of the Hessian may be computationally expensive as $\mathcal{O}(N^2)$ parameters must be accumulated for each of the n training samples.

ii. The Hessian must be inverted to find the direction, $\mathcal{O}(N^3)$. This gets very expensive as N gets large.

iii. The direction given need not head towards a minimum - it could head towards a *maximum* or *saddle point*. This occurs if the Hessian is not *positive-definite* i.e.

$$\mathbf{v}^\mathsf{T}\mathbf{H}\mathbf{v} > 0$$

for all $\mathbf{v}$. The Hessian may be made positive definite using

$$\tilde{\mathbf{H}} = \mathbf{H} + \lambda\mathbf{I}$$

If λ is large enough then $\tilde{\mathbf{H}}$ is positive definite.

iv. If the surface is highly non-quadratic the step sizes may be too large and the optimisation becomes unstable.

The approach in Part (b) both reduces the space required accumulation, only gradients are required, as well as not requiring the inverse of the Hessian, the scheme directly calculates the inverse. An initial value is needed for this scheme ($\mathbf{A}_0$). The simplest approach is to use a diagonal matrix with small values on the leading diagonal (easy to invert and will not distort the final results).

If SGD is being used, then the approximation allows only depends on the size of the batch, not the dimesionality of the parameter space. [20%]

Comments This questions examined neural network optimisation using second-order approaches. This was the most popular question. The question was generally well answered with candidates showing a good understanding of second-order approaches and the form of approximation provided.

3. *Kernels for Structured data*

(a)(i) The following steps are used in Speaker Verification with SVMs

- Train the general GMM on all the enrolment data.
- For each enrolled speaker compute the Fisher score-space. To obtain "negative" examples use other speaker's data with the general GMM.
- Train the SVM
- During verification, extract the SVM for the claimed identity and recognise. [20%]

(a)(ii) The log-likelihood may be expressed as

$$\log(p(\mathbf{X}^{(i)}|\theta)) = \sum_{j=1}^{T^{(i)}} \log\left(\sum_{m=1}^{M} c_m \mathcal{N}(\mathbf{x}_j^{(i)}; \boldsymbol{\mu}_m, \Sigma_m)\right)$$

Standard problem to compute the score-space (described in lectures) Considering just the means of a GMM

$$\boldsymbol{\phi}(\boldsymbol{X}^{(i)}) = \begin{bmatrix} \sum_{t=1}^{T^{(i)}} P(\omega_1|\boldsymbol{x}_t^{(i)}, \hat{\boldsymbol{\theta}})\hat{\boldsymbol{\Sigma}}_1^{-1}(\boldsymbol{x}_t^{(i)} - \hat{\boldsymbol{\mu}}_1) \\ \vdots \\ \sum_{t=1}^{T^{(i)}} P(\omega_M|\boldsymbol{x}_t^{(i)}, \hat{\boldsymbol{\theta}})\hat{\boldsymbol{\Sigma}}_M^{-1}(\boldsymbol{x}_t^{(i)} - \hat{\boldsymbol{\mu}}_M) \end{bmatrix}$$

This is a $M \times d$ features vector. [30%]

(b)(i) For the linear kernel, the sequence kernel looks like, $\boldsymbol{\mu}^{(i)}$ is the mean of the vectors in $\mathbf{X}^{(i)}$

$$\begin{aligned} k(\mathbf{X}^{(i)}, \mathbf{X}^{(s)}) &= \sum_{j=1}^{T^{(i)}} \sum_{k=1}^{T^{(s)}} \mathbf{x}_j^{(i)\mathsf{T}} \mathbf{x}_k^{(s)} \\ &= \left(\sum_{j=1}^{T^{(i)}} \mathbf{x}_j^{(i)\mathsf{T}}\right)\left(\sum_{k=1}^{T^{(s)}} \mathbf{x}_k^{(s)}\right) \\ &= T^{(i)}T^{(s)}\boldsymbol{\mu}^{(i)\mathsf{T}}\boldsymbol{\mu}^{(s)} \end{aligned}$$

Compare this to the Fisher kernel with a single component (global mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$)

$$\begin{aligned} &\left(\sum_{j=1}^{T^{(i)}} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_j^{(i)} - \boldsymbol{\mu})\right)^{\mathsf{T}} \left(\sum_{k=1}^{T^{(s)}} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_k^{(s)} - \boldsymbol{\mu})\right) \\ &\quad = T^{(i)}T^{(s)}\left(\boldsymbol{\mu}^{(i)\mathsf{T}}\boldsymbol{\Sigma}^{-2}\boldsymbol{\mu}^{(s)} - \boldsymbol{\mu}^{\mathsf{T}}\boldsymbol{\Sigma}^{-2}\left(\boldsymbol{\mu}^{(i)} + \boldsymbol{\mu}^{(s)}\right) + \boldsymbol{\mu}^{\mathsf{T}}\boldsymbol{\Sigma}^{-2}\boldsymbol{\mu}\right) \end{aligned}$$

The covariance matrix for the component should be an identity matrix for the two kernels to yield the same values. Also the global mean, $\boldsymbol{\mu}$, needs to be zero. This equates to sphering the data prior to constructing the classifiers. [25%]

(b)(ii) Gaussian kernel has the form

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$$

The following points should be mentioned

- Fisher kernel requires explicit mapping into the feature-space, this is not necessary (or possible) for the Gaussian kernel.
- The computation cost for the Fisher kernel is a function of the number of components M.
- Both schemes use non-linear transformations to derive the feature-space.
- Computational costs are
  - Fisher kernel, $\mathcal{O}(T^{(i)} + T^{(s)})$ to derive posteriors - Md dot-product.
  - Sequence kernel, $\mathcal{O}(T^{(i)}T^{(s)})$ as all combinations of observations examined.

[25%]

Comments This question examined the use of kernels for sequence data and Support Vector Machine (SVM) training. This was the least popular question and there were a number of poor answers. The first part of the question was closely related to am Examples Paper question, but a number of students gave very poor answers. Few students could correctly analyse the cost implications of the two approaches.

4. *Deep Learning and Sequence models*

(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) The following equations can be used to describe attention

$$\begin{aligned} \mathbf{h} &= \sum_{j=1}^{L} \alpha_j \mathbf{x}_j^{(i)} \\ \alpha_j &= \frac{\exp(e_j)}{\sum_{l=1}^{L} \exp(e_l)} \end{aligned}$$

The forms of attention vary depending on how e_j is calculated. For both attention self-attention is required to be used.

- *additive attention*:

$$e_j = \mathbf{w}^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{W}_\alpha \mathbf{x}_j^{(i)} + \mathbf{b}_\alpha)$$

  where $\boldsymbol{\phi}(.)$ is a non-linear activation function.
- *dot-product attention*:

$$e_j = \mathbf{x}_j^{(i)\mathsf{T}} \mathbf{W}_\alpha \mathbf{x}_j^{(i)}$$

Both mechanisms yield a PMF over the L elements. A non-linearity prior to the attention mechanism is useful to mention. [25%]

(c)(i) Multi-head attention mechanisms are usually described in terms of dot-product attention. For a K-head system

$$\mathbf{h}^{(k)} = \sum_{j=1}^{L} \alpha_j^{(k)} \mathbf{W}_{\mathrm{v}}^{(k)} \mathbf{x}_j^{(i)}$$

where

$$
\begin{aligned}
e_j^{(k)} &= \mathbf{x}_j^{(i)\mathsf{T}} \mathbf{W}_{\mathtt{k}}^{(k)\mathsf{T}} \mathbf{W}_{\mathtt{q}}^{(k)} \mathbf{x}_j^{(i)} \\
\alpha_j^{(k)} &= \frac{\exp(e_j^{(k)})}{\sum_{l=1}^{L} \exp(e_l^{(k)})}
\end{aligned}
$$

The final form of $\mathbf{h}$ then becomes

$$
\mathbf{h} = \begin{bmatrix} \mathbf{h}^{(1)} \\ \vdots \\ \mathbf{h}^{(K)} \end{bmatrix}
$$

[20%]

(c)(ii) Each of the heads of the multi-head attention defines a PMF over the input sequence in the same way as the single attention mechanism. Each of the K attention mechanisms should extract complementary information as they are all trained simultaneously.

The main issue with using multi-head attention mechanism is the increase in the number of model parameters, there are K times as many parameters in the attention mechanism, and the input to $\mathbf{g}()$ will be K times as large. [15%]

(c)(iii) Dropout de-activates nodes of a network. For the form of attention mechanism this results in randomly setting values of the weight matrices to zero. Other forms are possible where elements of $\boldsymbol{x}_j^{(i)}$ are randomly set to zero.

This is expected to improve performance as using multi-head attention increases the number of model parameters. Dropout acts as a form of regularisation. [15%]

Comments This questions examined the students' knowledge of deep learning for sequence data. Many candidates showed a good knowledge of the deep-learning approaches discussed in the course. Some candidates described the use of RNNs, rather than attention mechanisms, as requested in the question.