Solutions to 4F10 Deep Learning and Structured Data, 2026

1. *Mixture Models and ML training*

(a) If $\mathbf{f}(x)$ has the form given then considering only the exponential terms

$$\alpha_1 x + \alpha_2 x^2 = -\frac{x^2}{2\sigma^2} + \frac{x\mu}{\sigma^2}$$

Hence

$$\begin{aligned} \alpha_1 &= \frac{\mu}{\sigma^2} \\ \alpha_2 &= -\frac{1}{2\sigma^2} \end{aligned}$$

The value of Z will consist of both the standard normalisation term and the final quadratic term from the Gaussian, hence

$$Z = \frac{\sqrt{2\pi\sigma^2}}{\exp(-\mu^2/2\sigma^2)}$$

[25%]

(b)(i) The variance for each component is the same:

$$\sigma^2 = -\frac{1}{2\lambda}$$

The mean is component specific

$$\mu_m = -\frac{\alpha_m}{2\lambda}$$

Using the expression earlier

$$Z_m = \frac{\sqrt{-\pi/\lambda}}{\exp\left(\alpha_m^2/(4\lambda)\right)}$$

[15%]

(b)(ii) The auxiliary function for the general mixture model may be written as

$$\mathcal{Q}(\boldsymbol{\alpha}, \hat{\boldsymbol{\alpha}}) = \sum_{m=1}^{M}\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)\left[\log\left(c_m\right) - \frac{1}{2}\log(-\pi/\lambda) + \hat{\alpha}_m x + \frac{\hat{\alpha}_m^2}{4\lambda} + \lambda x^2\right]$$

Differentiating with respect to $\hat{\alpha}_m$ and equating to zero yields

$$\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)\left[x_i + \frac{\hat{\alpha}_m}{2\lambda}\right] = 0$$

Yielding

$$\hat{\alpha}_m = -2\lambda\frac{\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)x_i}{\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)}$$

[30%]

(b)(iii) Differentiating the auxiliary function with respect to λ and equating to zero yields

$$\sum_{m=1}^{M}\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)\left[\frac{1}{2\hat{\lambda}} - \frac{\hat{\alpha}_m^2}{4\hat{\lambda}^2} + x_i^2\right] = 0$$

Need to substitute the estimation in the previous stage so that solution can be expressed in terms of suffieient statistics

$$\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)\left[\frac{1}{2\hat{\lambda}} - \left(\frac{\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)x_i}{\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)}\right)^2 + x_i^2\right] = 0$$

which yields

$$\frac{1}{2\hat{\lambda}} = -\frac{1}{\sum_{m=1}^{M}\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)} \sum_{m=1}^{M}\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)\left[x_i^2 - \left(\frac{\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)x_i}{\sum_{i=1}^{N} P(\omega_m|x_i,\alpha)}\right)^2\right]$$

[20%]

(b)(iv) Both representations can yield closed form solutions based on the same set of sufficient statistics. [Any sensible comments on the two estimation approaches acceptable] [10%]

Examiner's Comments: This questions examined the students' knowledge of Expectation Maximisation (EM) and the parameterisation of distributions. It was the most popular of the questions. Part (a) was generally well done with students being able to relate the parameters of a log-linear model to a Gaussian distribution. Despite deriving connections with the Gaussian distribution, many candidates could not derive the normalisation term for the log-linear model. Most candidates understood the form of EM estimation.

2. *Support Vector Machines and Kernels*

(a)(i) For a general kernel the form is

$$\sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b = 0$$

where α are the Lagrange multipliers for each training sample i. Comments to make:

- large dimensional feature-spaces can be generated with kernels
- SVMs are based on large-margin training, good generalisation for high dimensions
- computational cost is determined by the number of support vectors, where $\alpha_i > 0$

[15%]

(b) From the answer to part (a)

$$\sum_{i=1}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b = \sum_{i=1}^{N} \alpha_i y_i \sum_{j=1}^{n} k_j(\mathbf{x}_i, \mathbf{x}) + b = 0$$

[10%]

(c)(i) The kernel operation is the dot product of the vectors in the feature-space. So

$$\begin{aligned} k(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) &= \boldsymbol{\phi}\left(\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix}\right)^{\mathsf{T}} \boldsymbol{\phi}\left(\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix}\right) \\ &= \begin{bmatrix} x_1^{(1)3} \\ \sqrt{3}x_1^{(1)2}x_2^{(1)} \\ \sqrt{3}x_1^{(1)}x_2^{(1)2} \\ x_2^{(1)3} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} x_1^{(2)3} \\ \sqrt{3}x_1^{(2)2}x_2^{(2)} \\ \sqrt{3}x_1^{(2)}x_2^{(2)2} \\ x_2^{(1)3} \end{bmatrix} \\ &= x_1^{(1)3}x_1^{(2)3} + 3x_1^{(1)2}x_2^{(1)}x_1^{(2)2}x_2^{(2)} + 3x_1^{(1)}x_2^{(1)2}x_1^{(2)}x_2^{(2)2} + x_2^{(1)3}x_2^{(2)3} \\ &= (x_1^{(1)}x_1^{(2)} + x_2^{(1)}x_2^{(2)})^3 \end{aligned}$$

[25%]

(c)(ii) The dimensionality of the inhomogeneous polynomial kernel can be expressed as a sum of homogeneous ones (from question). Thus the complete dimensionality is the sum of the dimensionalities of all the individual homogeneous ones. Since the dimensionality for power c is simply $c+1$ for $d=2$, then the complete dimensionality is

$$\sum_{c=0}^{c}(c+1) = \frac{1}{2}(c+1)(c+2) = (c+2)!/(c!2!)$$

[20%]

(d)(i) The form of the Gaussian kernel (from lecture notes)

$$k(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\frac{1}{2s}||\mathbf{x} - \mathbf{x}_i||^2\right)$$

where $s > 0$ is the free parameter associated with the kernel. [10%]

(d)(ii) The effective dimensionality can be obtained from the rank of the Gram matrix. As $s \to 0$ the dimensionality becomes very large, bounded by the number of training samples N. As $s \to \infty$ all distances become 1, so the Gram matrix only has a dimensionality of 1. [20%]

Examiner's Comments: This questions examined kernels and the impact that the form of the kernel has on the dimensionality of the induced feature-space. Most candidates showed knowledge of polynomial feature-spaces. However, the requested proof often had vague steps in the derivation.

3. *Neural Network Training*

(b) The ReLu activation function can be written as, for a single element

$$\texttt{relu}(x_i) = \begin{cases} x_i & x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

For the softmax the i-th value given by

$$[\texttt{soft}(\mathbf{x})]_i = \frac{\exp(x_i)}{\sum_{j=1}^{K} \exp(x_j)} = \phi_i(\mathbf{x})$$

The ReLU function provide a non-linearity so that the network yields non-linear decision boundaries. The softmax on the output layer is suited for classification tasks as this results in values that sum to 1 and are all positive, so can be treated as probabilities. [15%]

(c)(i) The Jacobian associated with the softmax function input $\mathbf{x}$ to output $\mathbf{y}$

$$\begin{aligned} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} &= \begin{bmatrix} \phi_1(\mathbf{x})(1-\phi_1(\mathbf{x})) & -\phi_1(\mathbf{x})\phi_2(\mathbf{x}) & \dots & -\phi_1(\mathbf{x})\phi_K(\mathbf{x}) \\ -\phi_2(\mathbf{x})\phi_1(\mathbf{x}) & \phi_2(\mathbf{x})(1-\phi_2(\mathbf{x})) & \dots & -\phi_2(\mathbf{x})\phi_K(\mathbf{x}) \\ \vdots & \ddots & \ddots & \vdots \\ -\phi_K(\mathbf{x})\phi_1(\mathbf{x}) & -\phi_K(\mathbf{x})\phi_2(\mathbf{x}) & \dots & \phi_K(\mathbf{x})(1-\phi_K(\mathbf{x})) \end{bmatrix} \\ &= \begin{bmatrix} \phi_1(\mathbf{x}) & 0 & \dots & 0 \\ 0 & \phi_2(\mathbf{x}) & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \phi_K(\mathbf{x}) \end{bmatrix} - \begin{bmatrix} \phi_1(\mathbf{x}) \\ \phi_2(\mathbf{x}) \\ \vdots \\ \phi_K(\mathbf{x}) \end{bmatrix} \begin{bmatrix} \phi_1(\mathbf{x}) & \phi_2(\mathbf{x}) & \dots & \phi_K(\mathbf{x}) \end{bmatrix} \end{aligned}$$

The VJP for is then given by

$$\begin{aligned} \texttt{soft\_vjp}(\mathbf{a}, \mathbf{b}) &= \mathbf{b}\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\bigg|_{\mathbf{a}} \right) \\ &= \mathbf{b} \odot \texttt{soft}(\mathbf{a}) - \mathbf{b}\texttt{soft}(\mathbf{a})^{\mathsf{T}}\texttt{soft}(\mathbf{a}) \\ &= \mathbf{b} \odot \texttt{soft}(\mathbf{a}) - \texttt{sum}(\mathbf{b} \odot \texttt{soft}(\mathbf{a}))\texttt{soft}(\mathbf{a}) \end{aligned}$$

where $\mathbf{a}$ is obtained by passing the observation through the network upto the softmax function in the forward pass,

$$\mathbf{b} = \frac{\partial \texttt{ce}(y_i, \mathbf{y}(\mathbf{x}))}{\partial \mathbf{y}}\bigg|_{\texttt{soft}(\mathbf{a})}$$

[35%]

(c)(ii) Again the starting point for the VJP is given by the differential over the linear transformation

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{A}^{(l)\mathsf{T}}$$

Also need to compute how the matrix $\mathbf{A}^{(l)}$ changes (these are computing the parameters of the model) $\frac{\partial \mathbf{z}}{\partial \mathbf{A}}$. Considering just z_i

$$\frac{\partial z_i}{\partial \mathbf{A}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{x}^{(l)\mathsf{T}} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

This is repeated for all elements of the model. For the VJP the input is

$$\mathbf{c} = \left.\frac{\partial \mathtt{ce}(y, \mathbf{y}(\mathbf{x}))}{\partial \mathbf{z}^{(l)}}\right|_{\mathbf{z}^{(l)}}$$

The output becomes

$$\begin{aligned} \mathbf{c}\left.\frac{\partial \mathtt{ce}(y, \mathbf{y}(\mathbf{x})}{\partial \mathbf{x}^{(l)}}\right|_{\mathbf{x}_i^{(l)}} &= \mathbf{c}\mathbf{A}^{(l)\mathsf{T}} \\ \mathbf{c}\left.\frac{\partial \mathtt{ce}(y, \mathbf{y}(\mathbf{x}))}{\partial \mathbf{A}^{(l)}}\right|_{\mathbf{A}^{(l)}} &= \mathbf{c}^{\mathsf{T}}\mathbf{x}^{(l)} \end{aligned}$$

[25%]

(c)(iii) The first stage is to do a forward pass through the network to get the "current" values of $\mathbf{x}_i^{(l)}$ and $\mathbf{z}_i^{(l)}$ for all $L+1$ layers (including softmax). The loop involving the VJP values goes in the reverse direction. For the top-layer there the following loop is applied

(a) compute $\mathbf{b}_i$ given the current parameters and $\mathbf{z}_i^{(L+1)}$

(b) $\mathbf{da}^{(L+1)} = \mathtt{soft\_vjp}(\mathbf{z}_i^{(L+1)}, \mathbf{b}_i)$

(c) $\mathbf{dx}^{(L+1)}, \mathbf{dA}^{(L+1)} = \mathtt{mul\_vjp}(\mathbf{x}_i^{(L+1)}, \mathbf{A}^{(L+1)}, \mathbf{da}^{(L+1)})$

This yields the gradient for the linear transformation of the softmax layer. For the hidden layer there is following expressions moving backwards through the layers, considering here the l-th layer

(a) $\mathbf{da}^{(l)} = \mathtt{relu\_vjp}(\mathbf{z}_i^{(l)}, \mathbf{dx}^{(l+1)})$

(b) $\mathbf{dx}^{(l)}, \mathbf{dA}^{(l)} = \mathtt{mul\_vjp}(\mathbf{x}_i^{(l)}, \mathbf{A}^{(l)}, \mathbf{da}^{(l)})$

[25%]

Examiner's Comments: This question examined the understanding of use of the vector-Jacobian product (VJP) to estimate the parameters of a deep neural network. This topic was introduced to the course last year, and was the first exam question on the topic. It was the least popular question despite the question being closely related to one that was given in an examples paper. It was disappointing that performance was generally poor, with many candidates unable to give the appropriate forms for the use of VJP.

4. *Transformer Architecture for Language Modelling*

(a) The simplest form is the standard cross-entropy optimisation

$$\arg\max \left\{ \sum_{i=1}^{N} \sum_{j=1}^{T} \log(P(\omega_j^{(i)} | \omega_{1:j-1}^{(i)})) \right\}$$

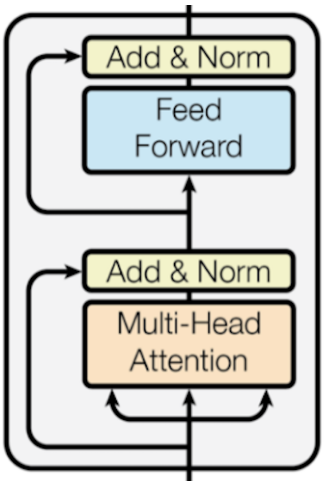
The language model is trained to model $P(\omega|\omega_{1:j-1}^{(i)})$. This allows any length of sentence to be modelled. An `<eos>` token needs to be added to the token-list (sometimes the start of sentence `<sos>` is also added). [20%]

(b) An iterative approach is used to generate sentences. To generate the i-th token given the generated word sequence $\hat{\omega}_{1:i-1}$

$$\hat{\omega}_i \sim P(\omega|\hat{\omega}_{1:i-1})$$

Either sampling or selecting the maximum is acceptable. This word is then added to the generated sequence. The process is repeated until an end of sentence symbol, `<eos>`, is generated and the process terminated. [15%]

(c)(i) The description of the model means that an encoder style transformer is needed



The layers of the block are

- **Multi-Head Attention** layer Multihead attention looks like

$$\begin{aligned} \alpha_{\tau i}^{(j)} &= \frac{1}{Z} \exp\left(\mathbf{k}_{\tau}^{(j)\mathsf{T}} \boldsymbol{q}_i^{(j)} / \sqrt{d_k^{(j)}} \right) \\ \boldsymbol{c}_i^{(j)} &= \sum_{\tau=1}^{T} \alpha_{\tau i}^{(j)} \mathbf{W}_{\mathtt{v}}^{(j)} \boldsymbol{v}_{\tau} \\ \boldsymbol{c}_i &= = \left[\begin{array}{ccc} \boldsymbol{c}_i^{(1)\mathsf{T}} & \ldots & \boldsymbol{c}_i^{(J)\mathsf{T}} \end{array} \right]^{\mathsf{T}} \end{aligned}$$

and for self-attention

$$\begin{aligned} \mathbf{k}_i^{(j)} &= \mathbf{W}_{\mathtt{k}}^{(j)} \boldsymbol{v}_i; \\ \mathbf{q}_i^{(j)} &= \mathbf{W}_{\mathtt{q}}^{(j)} \boldsymbol{v}_i; \end{aligned}$$

This layer enables the network to focus on the part of the sequence that is most relevant, and enables any length of sequence to be handled

- **Residual Network & Layer Norm** The residual connection simply connects the input and the output directly. Layer-norm simply normalises the values using the activation function means and variance. These stages both aid training.
- **Feed-Forward network** This layer is applied to each of elements of the sequence and supplies the non-linearity to enable complex decision boundaries/
- **Residual Network & Layer Norm** (see above)

[40%]

(c)(ii) For the L-th layer only a single token output is required, rather than the T elements from all the other layers. Typically the last token, here the i-th token when predicting the $i-1$-th token, is used, $\mathbf{y}_{i-1}^{(L)}$. This output is then passed through a linear transform and a softmax to yield the estimate of the probability.

$$\hat{P}(\omega|\omega_{1:j-1}^{(i)}) = \texttt{softmax}(\mathbf{W}\mathbf{y}_{i-1}^{(L)} + \mathbf{b})$$

[15%]

(d) The model is based on "teacher forcing" training. The reference back-history is used rather than the history generated by the language model. This makes training far more efficient. [10%]

Examiner's Comments: This question examined the use of a "decoder-only" transformer architecture for language modelling. Most students showed knowledge of the transformer architecture, but the connection for its use to prediction the conditional probability for the LM was missed by many students.