ECE/CS 559: Neural Networks **Fall 2024**

Mock Final

Wednesday December 11, 2024 — There are **7 questions**; the actual exam will have 5.

Note: ***Please write your name on every page!*** *You are expected to work on your own on this exam. This exam is* ***closed notes*** *and no calculators are allowed. Other electronic devices or communication with others are also* ***not*** *allowed. Include your reasoning, not just the final answer. Be clear and concise. Watch your time. If stuck, move on then come back later.* ***Good luck!***

NAME: Solutions

— You may use this page and backs of pages for scratch work. —

Lecture 17

HW7

1. Consider the following data points in $\mathbb{R}^2$: $\overset{x_1}{\begin{bmatrix}-0.01\\1\end{bmatrix}}, \overset{x_2}{\begin{bmatrix}1\\1\end{bmatrix}}, \overset{x_3}{\begin{bmatrix}0.01\\-1\end{bmatrix}}, \overset{x_4}{\begin{bmatrix}-1\\-1\end{bmatrix}}$.

   (a) (3 pts) Perform, step by step, Lloyd's algorithm for k-means with $k = 2$ until convergence, starting with the second and third points as initial centers. Express the final centers clearly. Draw the initial and final Voronoi cells (along with the data points) on the same plot.

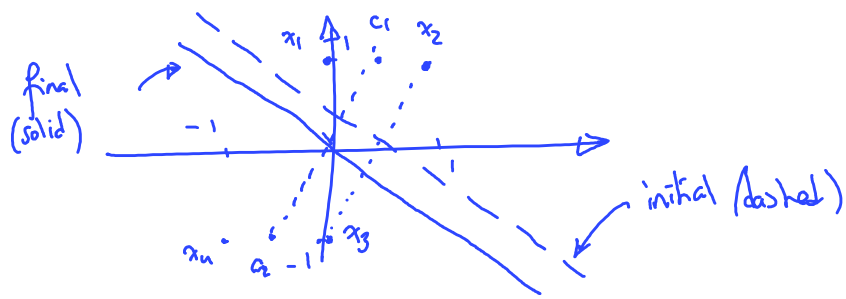
Initialization: $c_1 = \begin{bmatrix}1\\1\end{bmatrix}$ $c_2 = \begin{bmatrix}0.01\\-1\end{bmatrix}$

Step 1 : $V_1 = \left\{ \begin{bmatrix}-0.01\\1\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix} \right\}$, $V_2 = \left\{ \begin{bmatrix}0.01\\-1\end{bmatrix}, \begin{bmatrix}-1\\-1\end{bmatrix} \right\}$

(closer to c_1) (closer to c_2)

updated centers : $c_1 = \begin{bmatrix}\frac{0.99}{2}\\1\end{bmatrix}$ $c_2 = \begin{bmatrix}\frac{-0.99}{2}\\-1\end{bmatrix}$

Step 2 : V_1 the same V_2 the same $\rightarrow$ algorithm stops



   (b) (2 pts) Suggest a different initialization that will not converge to the same final centers.

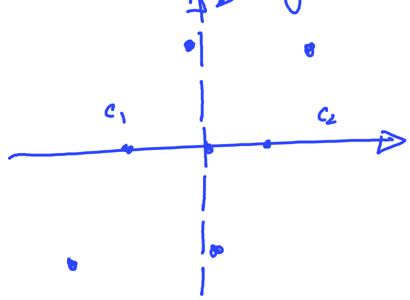
To not converge the same way we can use try and group the points differently in a way that stays consistent with proximity. In particular, group $(x_1 \,\&\, x_4)$ and $(x_2$ and $x_3)$.

Their centers are then $c_1 = \begin{bmatrix}\frac{-1.01}{2}\\0\end{bmatrix}$ and $c_2 = \begin{bmatrix}\frac{1.01}{2}\\0\end{bmatrix}$

If we initialize these, $V_1 = \{x_1, x_4\}$ and $V_2 = \{x_2, x_3\}$

thus the algorithm will not update and stop!

y-axis is the Voronoi separator.



Lecture 18

2. Consider the following data points in $\mathbb{R}^2$: $\begin{bmatrix}0\\1\end{bmatrix}$, $\begin{bmatrix}1\\1\end{bmatrix}$, $\begin{bmatrix}0\\-1\end{bmatrix}$, $\begin{bmatrix}-1\\-1\end{bmatrix}$.

   (a) (1 pt) State the competitive (winner-take-all) learning rule in clear mathematical notation. What "kind" of rule is this? It is a Hebbian rule.

   $$i = \max_{i' \text{(neurons)}} w_{i'}^T x$$

   $$\zeta = w_i^T x \quad \text{(output of winner)}$$

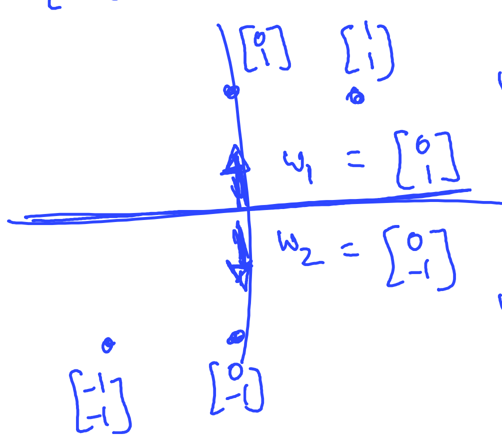
   update: (only winner)

   $$w_i \leftarrow w_i + \eta \zeta_i (x - w_i) = (1 - \eta\zeta_i) w_i + \eta \zeta_i x$$

   (b) (4 pts) Perform this learning rule, with one difference: *after every update, round weights to the nearest integer*. Process the data in the given order, for 2 epochs. Use $\eta = \frac{1}{2}$ and initialize with weights $\begin{bmatrix}-1\\2\end{bmatrix}$ and $\begin{bmatrix}2\\-1\end{bmatrix}$. Give the clustering interpretation for the final weights. (this means we have 2 neurons. will make this explicit)

w_1 w_2 before iteration	Epoch Iteration	Data Point	winner info	update
$\begin{bmatrix}-1\\2\end{bmatrix}\begin{bmatrix}2\\-1\end{bmatrix}$	Ep 1, It 1	$x=\begin{bmatrix}0\\1\end{bmatrix}$	winner $i=1$, $\zeta=2$	$w_1 = (1-\frac{1}{2}2)w_1 + \frac{1}{2}2\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}2\\-1\end{bmatrix}$	Ep 1, It 2	$x=\begin{bmatrix}1\\1\end{bmatrix}$	winner $i=1$, $\zeta=1$ (tie, break either way)	$w_1 = (1-\frac{1}{2}1)w_1 + \frac{1}{2}\cdot 1\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\\1\end{bmatrix} \xrightarrow{\text{round}} \begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}2\\-1\end{bmatrix}$	Ep 1, It 3	$x=\begin{bmatrix}0\\-1\end{bmatrix}$	winner $i=2$, $\zeta=1$	$w_2 = \frac{1}{2}w_2 + \frac{1}{2}\begin{bmatrix}0\\-1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}2\\-2\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix}$	Ep 1, It 4	$x=\begin{bmatrix}-1\\-1\end{bmatrix}$	winner $i=2$, $\zeta=0$	$w_2 = 1\,w_2 + 0\begin{bmatrix}-1\\-1\end{bmatrix} = \begin{bmatrix}1\\-1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix}$	Ep 2, It 5	$x=\begin{bmatrix}0\\1\end{bmatrix}$	winner $i=1$	(no change, since $w_1 = x_1$)
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix}$	Ep 2, It 6	$x=\begin{bmatrix}1\\1\end{bmatrix}$	winner $i=1$, $\zeta=1$	$w_1 = \frac{1}{2}w_1 + \frac{1}{2}\begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\\1\end{bmatrix} \xrightarrow{\text{round}} \begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix}$	Ep 2, It 7	$x=\begin{bmatrix}0\\-1\end{bmatrix}$	winner $i=2$, $\zeta=1$	$w_2 = \frac{1}{2}w_2 + \frac{1}{2}\begin{bmatrix}0\\-1\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\\-1\end{bmatrix} \xrightarrow{\text{round}} \begin{bmatrix}0\\-1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}0\\-1\end{bmatrix}$	Ep 2, It 8	$x=\begin{bmatrix}-1\\-1\end{bmatrix}$	winner $i=2$, $\zeta=1$	$w_2 = \frac{1}{2}w_2 + \frac{1}{2}\begin{bmatrix}-1\\-1\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\\-1\end{bmatrix} \xrightarrow{\text{round}} \begin{bmatrix}0\\-1\end{bmatrix}$

Clustering:



winners with w_1

winner with w_2

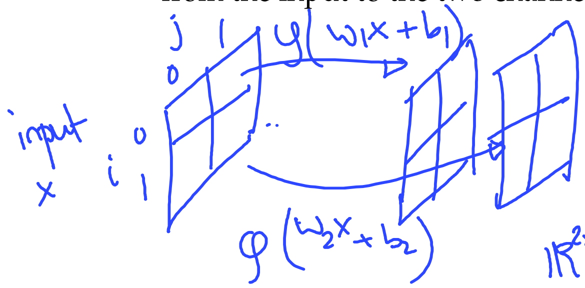
2 clusters

Lecture 16

HW 6

3. Consider a simple single-layer CNN, which takes as input a 2×2 image and produces two 2×2 channels, by applying kernels of width 1 with bias.

   (a) (2 pts) Describe the parameters of this CNN and write the forward equations from the input to the two channels.



output channels f^1, f^2

channel 1: w_1, b_1 (both $\mathbb{R}^1$)

channel 2: w_2, b_2 (both $\mathbb{R}^1$)

Forward equations:

$$\overset{\mathbb{R}^{2\times2}}{f^1} = \varphi(\overset{\mathbb{R}^1}{w_1}\ \overset{\mathbb{R}^{2\times2}}{x} + \overset{\mathbb{R}^1}{b_1})$$

$$f^2 = \varphi(w_2 x + b_2)$$

(per coordinate)

   (b) (3 pts) Write the backpropagation equations for this CNN layer, starting with the gradients of the loss at the output. (You don't need to know the loss.)

We assume we have $\frac{\partial \ell}{\partial f^1_{ij}}$ and $\frac{\partial \ell}{\partial f^2_{ij}}$

at each coordinate of f^1 & f^2 $i \in \{0,1\}$, $j \in \{0,1\}$

$$\frac{\partial \ell}{\partial w_1} = \frac{\partial}{\partial_1}\, \ell(\underset{w_1 x_{00} + b_1}{f^1_{00}}, \underset{w_1 x_{01} + b_1}{f^1_{01}}, \underset{w_1 x_{10} + b_1}{f^1_{10}}, \underset{w_1 x_{11} + b_1}{f^1_{11}})$$

$$= \sum_{ij} \frac{\partial \ell}{\partial f^1_{ij}} \cdot \frac{\partial f^1_{ij}}{\partial w_1} = \sum_{ij} \frac{\partial \ell}{\partial f^1_{ij}} \cdot x_{ij} \cdot \varphi'(w_1 x_{ij} + b_1)$$

Similarly:

$$\frac{\partial \ell}{\partial w_2} = \sum_{ij} \frac{\partial \ell}{\partial f^2_{ij}} \cdot x_{ij} \cdot \varphi'(w_2 x_{ij} + b_2)$$

$$\frac{\partial \ell}{\partial b_1} = \sum_{ij} \frac{\partial \ell}{\partial f^1_{ij}} \cdot 1 \cdot \varphi'(w_1 x_{ij} + b_1) \qquad \frac{\partial \ell}{\partial b_2} = \sum_{ij} \frac{\partial \ell}{\partial f^2_{ij}} \cdot 1 \cdot \varphi'(w_2 x_{ij} + b_2)$$

Gaussian Autoencoder Model

Gaussian/Normal
mean covariance
$z = \mathcal{N}(0, I)$ (prior)

P_{prior}
(forced, not optimized)

$z|x = g(x) + \mathcal{N}(0,I)$
$= \mathcal{N}(g(x), I)$ (encoder)

P_{enc}
(optimized via g)

$\hat{x}|z = f(z) + \mathcal{N}(0,\Sigma)$
$= \mathcal{N}(f(z), \Sigma)$ (decoder)

P_{dec}
(optimized via f and Σ)

Lecture 20, 21

HW7

4. Consider an autoencoder, with input x, embedding z, and output $\hat{x}$. Model the prior of the embedding to be a standard Gaussian vector. Let the encoder be a neural network parametrized by W producing an output $g(x; W)$ plus a standard Gaussian noise. Let the decoder be a neural network parametrized by W producing an output $f(z; W)$ plus a Gaussian with a *learnable covariance matrix* Σ. (W has both sets of parameters)

(a) (2.5 pts) Write the general form of the ELBO loss and specialize it to this autoencoder, simplifying it in the same way as the Gaussian autoencoder in class.

green = Same as lecture

The only thing that changes is the decoder's distribution. It becomes:

$$P_{dec}(x|z) \propto \frac{1}{(\det \Sigma)^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(x - f(z;w))^T \Sigma^{-1} (x - f(z,w))\right]$$

The general ELBO loss is $-\mathbb{E}\left[\log \frac{P_{prior}(z) P_{dec}(x|z)}{P_{enc}(z|x)}\right]$. Here, it becomes:

$$\mathbb{E}[-\log P_{prior}(z) + \log P_{enc}(z|x)] = \mathbb{E}\left[\text{constant} + \frac{1}{2}\|z\|^2 - \frac{1}{2}\|z - g(x)\|^2\right]$$

$$= \mathbb{E}\left[\text{constant} + \cancel{\frac{1}{2}\|z\|^2} - \cancel{\frac{1}{2}\|z\|^2} + z^T g(x) - \frac{1}{2}\|g(x)\|^2\right]$$

drop constants as they don't affect optimization

ignore constants (don't affect optimization)

$$= \mathbb{E}\left[\underbrace{\mathbb{E}\left[z^T g(x) - \frac{1}{2}\|g(x)\|^2 \,|\, x\right]}_{\text{since } \mathbb{E}[z|x] = g(x)}\right] = \mathbb{E}\left[g(x)^T g(x) - \frac{1}{2}\|g(x)\|^2\right] = \frac{1}{2}\mathbb{E}\|g(x)\|^2 \quad ①$$

- $\mathbb{E}[-\log P_{dec}(x|z)] = \text{constant} + \frac{1}{2}\mathbb{E}\left[(x - f(z))^T \Sigma^{-1} (x - f(z))\right]$

$$= \frac{1}{2}\mathbb{E}\left[(x - f(g(x)+N))^T \Sigma^{-1} (x - f(g(x)+N))\right] \quad ②$$

ELBO = ① + ②

(b) (2.5 pts) In your homework, you concatenated z and the reconstruction error $x - \hat{x}$. Explain why this won't work here. Then, use the fact that you can write $\Sigma^{-1} = M^T M$ to suggest a simple modification that would work.

In our code, z referred to $g(x)$.

- In class, we had ELBO = $\mathbb{E}[\|g(x)\|^2] + \mathbb{E}[\|x - \underbrace{f(g(x)+N)}_{\hat{x}}\|^2]$ (ignoring constants and constant factors)

  z in the code

- The first term is still the same, but the second is changed to:

$$\mathbb{E}\left[(x - \hat{x})^T \Sigma^{-1} (x - \hat{x})\right]$$

- By using the hint, we can reparametrize $\Sigma^{-1} = M^T M$. By substituting:

$$\mathbb{E}\left[(x - \hat{x})^T M^T M (x - \hat{x})\right]$$

- We recognize this as the square norm of $M(x - \hat{x})$.

$$\mathbb{E}\left[\|M(x - \hat{x})\|^2\right]$$

- So, if instead of giving $x - \hat{x}$ to the MSE Loss, we give $M(x - \hat{x})$, concatenated with z (in this case $g(x)$), then we get ELBO.

- If M is learnable ("requires-gradients" in Pytorch) part of the model, then M will be optimized along the way, as desired, via backpropagation.

In office hours there was a question of where exactly we use the idealized GAN loss.
(a) In Lecture 23 ("analyzing GANs"), we considered this same idealized GAN loss (with expectation instead of sum). We said that, because the discriminator is optimizing the cross-entropy between P(S|X) and P_disc(S|X), the best thing to do is to set D(x) = P_disc(S=1|X=x) = P(S=1|X=x). So, finding the optimal D is equivalent to find this posterior distribution.
(b) We also use the idealized loss by using the fact that with the optimal discriminator above, the generator is optimizing the Jensen-Shannon divergence between P(X) (the true/population) and P(G(Z)). This is almost a "distance" and it is smallest when zero, which is achieved by setting them to be equal. In other words, it is achieved by making G(Z) have the same distribution as X, as the question states.

Lectures 22,23

5. Consider a GAN to generate a real-valued random variable X. Say the population of X is distributed in $[0,1]$ with density $f_X(x) = 2-2x$. Use a noise Z uniformly distributed in $[0,1]$. Use the idealized GAN loss we considered in class:

$$\mathop{\mathbf{E}}_{X,Z}\left[-\log D(X) - \log(1-D(G(Z)))\right]$$

(a) (2.5 pts) Say the initial generator is just the identity function $G(z)=z$. What is the optimal discriminator in this case?

We saw in class that the optimal discriminator is:

$$D(x) = \mathbb{P}(S=1|X=x) = \frac{\mathbb{P}(S=1)\cdot\mathbb{P}(X=x|S=1)}{\mathbb{P}(S=0)\,\mathbb{P}(X=x|S=0)+\mathbb{P}(S=1)\,\mathbb{P}(X=x|S=1)}$$

$\mathbb{P}(X=x|S=1) = f_X(x) = 2-2x$

$\mathbb{P}(X=x|S=0) = f_Z(x) = 1$ (since G is passing z).

$\mathbb{P}(S=0)=\mathbb{P}(S=1)=\frac{1}{2}$

$$\Rightarrow D(x) = \frac{2-2x}{2-2x+1} = \frac{2-2x}{3-2x}$$

(b) (2.5 pts) What is the optimal generator G? That is, find G such that $G(Z)$ has the same distribution as X.

Lecture 19 "desired distributions"

G is not necessarily unique, but let's choose it monotonically increasing. This, in particular, means that it's invertible: $x=G(z)$, $z=G^{-1}(x)$.

$$G(Z)\sim X \Rightarrow \mathbb{P}(G(Z)\le x) = \mathbb{P}(X\le x) = F_X(x) = \int_0^x f_X(t)\,dt$$
$$= \int_0^x 2-2t\,dt = 2t-t^2\Big]_0^x = 2x-x^2$$

But also $\mathbb{P}(G(Z)\le x) = \mathbb{P}(Z\le \underbrace{G^{-1}(x)}_{z}) = F_Z(z) = z$ (uniform)

$$z = G^{-1}(x) = 2x-x^2 \iff z = 2G(z)-G(z)^2 \Rightarrow 1-z = [1-G(z)]^2$$
$$\Rightarrow G(z) = 1-\sqrt{1-z}$$

(the other solution produces $G(z)\notin(0,1)$, which isn't acceptable!)

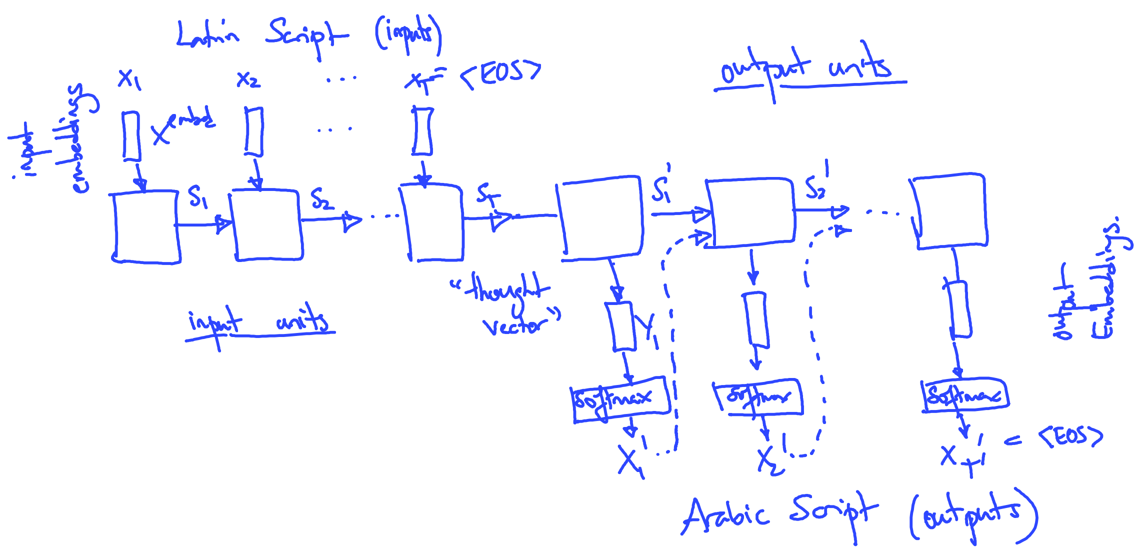
Lecture 24

6. Say we would like to build an ML model that takes as input Arabic return in Latin script, and outputs the same text in Arabic script. When written in Latin script, many Arabic letters require multiple characters to represent,

   (a) (1 pt) What kind of sequence model is most appropriate for this problem? Explain why.

   This is an asynchronous sequence to sequence model, because the output is a sequence and the input and output don't line up exactly.

   (b) (4 pts) Sketch an RNN architecture that you could use for this model. Draw a diagram with embeddings, units, inputs, and outputs clearly marked. Write a simple input-state-output equation for each unit.



- Input embedding: $x_t^{embd} = A x_t$, where x_t = one-hot
- Input units: $s_t = \varphi(W^x x_t + W^s s_{t-1} + b)$
- Output units: $s_t' = \varphi(V^s s_{t-1} + \underbrace{V^x x_{t-1}'}_{\text{optional but will make the RNN aware of what's generated}} + c)$
- Output embedding: $Y_t' = \varphi(U s_t' + d)$
- Outputs: $x_t' \sim$ Sample from Softmax(Y_t).

$$= \frac{e^{Y_t[j]}}{\sum_{j'} e^{Y_t[j']}}$$

I should have named this differently to not confuse with values.

Lectures 25, 26

7. Say you have a graph $\mathcal{G} = (V, E)$ consisting of vertices $v \in V$ and edges $E \subset V^2$, the subset of pairs of vertices that are connected. You want to make a binary classification (e.g., is there a disturbance or not) that depends on the inputs x_v, sitting at each node.

(a) (2 pts) You decide to design a *graph* self-attention, by remaining consistent with the graph at any given layer, i.e., uses only inputs connected to each vertex. Write the self-attention equations at a node v in clear mathematical notation.

The only thing we need to change is to make the attention over neighbors, i.e. v' s.t. $(v,v') \in E$

$Q_v = W^Q x_v$, $K_{v'} = W^K x_{v'}$, attention at v:

$$\alpha_{v,v'} = \underset{v' : (v,v') \in E}{\mathrm{Softmax}}\left(\frac{1}{\sqrt{k}} Q_v^T K_{v'}\right) = \begin{cases} 0 & ; \text{ if } (v,v') \notin E \\ \dfrac{e^{\frac{1}{\sqrt{k}} Q_v^T K_{v'}}}{\sum_{v' : (v,v') \in E} e^{\frac{1}{\sqrt{k}} Q_v^T K_{v'}}} & ; \text{ if } (v,v') \in E \end{cases}$$

(b) (3 pts) Explain (1) how you would build transformer layers using this modified self-attention, (2) whether or not the outputs of the last layer will only depend on neighboring inputs in the graph, and (3) how you would use the last layer outputs to make the binary classification with a clear description of the architecture and loss function.

(1) Calculate value at each node $V_{v'} = W^V x_{v'}$.

Combine according to attention: $S_v = \sum_{v' : (v,v') \in E} \alpha_{v,v'} V_{v'}$

Possibly create multiple heads + concatenate

Pass through add+norm, FF, add+norm, get x_v^{new}.

Repeat, to obtain multiple layers.

(2) The output at higher layers can depend on nodes further than the immediate neighbors, e.g. on the second layer, they will depend on neighbors of neighbors. (information propagates)

(3) Say x_v^{last} are the outputs of the last layer.

One approach is to add a linear layer + sigmoid

- $f = \sigma\left(\sum_v W_v x_v^{out} + b\right) \in (0,1)$

We can then use binary cross entropy, if data is (x,y) binary

- $\ell(f,y) = y \log \frac{1}{f} + (1-y) \log \frac{1}{1-f}$.