Deep Learning Homework 2

Björn Pettersson, bjpe900 2025-81022

27-09-2025

Problem 1 : Transformer (20 Points)

1.1.a)

Computing attention score mx as:

$$S = \frac{QK^T}{\sqrt{d_k}}$$

$$d_k = 2$$

We have:

$$K^T = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$QK^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$S = \frac{QK^T}{\sqrt{d_k}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

We apply the softmax $A = softmax(S)$ row wise since each ith row in attention mx s_i contians the similarity scores between the query q_i and all keys k_j:

For row 1:

$$(s_{11}, s_{12}) = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$$

apply softmax as:

$$a_{1j} = \frac{e^{s_{1j}}}{\sum_k e^{s_{1k}}}$$

$$a_{11} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}} = \frac{1}{2} = 0.5$$

$$a_{12} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}} = \frac{1}{2} = 0.5$$

Row2:

$$(s_{21}, s_{22}) = (\frac{1}{\sqrt{2}}, 0)$$

$$a_{21} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{0}} \approx 0.670$$

$$a_{22} = \frac{e^{0}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}} \approx 0.330$$

We have A:

$$A = \begin{bmatrix} \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}}+e^{1/\sqrt{2}}} & \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}}+e^{1/\sqrt{2}}} \\ \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}}+e^{0}} & \frac{e^{0}}{e^{1/\sqrt{2}}+e^{1/\sqrt{2}}} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.670 & 0.330 \end{bmatrix}$$

1.1.b)

We compute the attention output:

$$Z = AV$$

$$Z = \begin{bmatrix} 0.5 & 0.5 \\ 0.670 & 0.330 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0.5 \\ 1.34 & 0.33 \end{bmatrix}$$

1.2.a)

We have input matrix X, we have dimention of key $d_k = 2$. X is the identity mx we have:

Head1:

$$Q_1 = XW_Q^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$K_1 = XW_K^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$V_1 = XW_V^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Head2:

$$Q_2 = XW_Q^2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$K_2 = XW_K^2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$V_2 = XW_V^2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

1.2.b)

From before we have: $Z_i = softmax(\frac{Q_i K_i^T}{\sqrt{d_k}})V_i$: We calcualte Z for each head:

HEAD1:

$$S_1 = \frac{Q_1 K_1^T}{\sqrt{d_k}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Softmax:

$$A^1 = softmax(S_1)$$

row1:

$$a_{11} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}}$$

$$a_{12} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}}$$

row2 $0, \frac{1}{\sqrt{2}}$:

$$a_{21} = \frac{e^0}{e^{1/\sqrt{2}} + e^0} = 0.3302$$

$$a_{22} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^0} = 0.6698$$

$$A^1 = \begin{bmatrix} 0.5 & 0.5 \\ 0.3302 & 0.6698 \end{bmatrix}$$

$$Z^1 = A^1 V^1 = \begin{bmatrix} 0.5 & 0.5 \\ 0.3302 & 0.6698 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.3302 & 0.6698 \end{bmatrix}$$

HEAD2:

$$S_2 = \frac{Q_2 K_2^T}{\sqrt{d_k}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

Softmax:

$$A^2 = softmax(S_2)$$

row1 $\frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}$:

$$a_{11} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}} = 0.5$$

$$a_{12} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{1/\sqrt{2}}} = 0.5$$

row2 $\frac{1}{\sqrt{2}}, 0$:

$$a_{21} = \frac{e^{1/\sqrt{2}}}{e^{1/\sqrt{2}} + e^{0}} = 0.6698$$

$$a_{22} = \frac{e^{0}}{e^{1/\sqrt{2}} + e^{0}} = 0.3302$$

$$A^2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.6698 & 0.3302 \end{bmatrix}$$

$$Z^2 = A^2V^2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.6698 & 0.3302 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.3302 & 0.6698 \end{bmatrix}$$

1.2.c)

We concatenate results:

$$Z_{multi} = [Z \mid\mid Z] = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 \\ 0.3302 & 0.6698 & 0.3302 & 0.6698 \end{bmatrix}$$

1.3.a)

What would happen if the scaling factor was removed?

Mathematically (softmax distribution):

For each QK pair we calcualte cosine similarity as $Q_i \cdot K_i = \mid\mid Q_i \mid\mid \cdot \mid\mid K_i \mid\mid cos\theta$.

Where each row in K is of consits of $d_k$elements and contains token embedding positional encoding and weight scaleing for contextuality $K_i = (TokenEmbedding + PositionEncoding) \times W_K$

We calucalte the norm as: $\mid\mid K_i \mid\mid = \sqrt{k_{i1}^2 + k_{i2}^2 ... + k_{ij}^2}$ we see that this is proporational to the square root of number of elements in $\sqrt{K_i}$ which is $\sqrt{d_k}$. We also observe that the norms in this formula are scalars: $Q_i \cdot K_i = \mid\mid Q_i \mid\mid \cdot \mid\mid K_i \mid\mid cos\theta$, we are interested in the angle between the vectors rather than the size, and one of the things that does not change regarding to the norm

between the vectors is the dimention, this is why $\sqrt{d_k}$ is a good term to use for normalisation.

EFFECT ON DISTRIBUTION:

$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1} e^{x_j}}$, example inputs:

- 1x: softmax([1,2,3]) = [0.09, 0.24, 0.67],
- 2x: softmax([2,4,6]) = [0.02, 0.12, 0.86]
- 3x: softmax([3,6,9]) = [0.00, 0.02, 0.98]

Effect on softmax distribution: We observe that for large inputs we get a sharper distirbution, almost one hot encoding with one value becomming 1 and the others 0. For a softmax output p_i

$Var_{softmax} = \sum(p_i - 1/n)^2/n$

If we envision an extreme case one hot all but one p_i is 0,

- Uniform case $([0.25, 0.25, 0.25, 0.25]) : Variance = 0$
- Sharp case $([0.97, 0.01, 0.01, 0.01]) : Variance \approx 0.18$
- One-hot case $([1.0, 0.0, 0.0, 0.0]) : Variance = 0.1875$

The effect of the scaling on the distirbution is that it causes a lower varaiance and hence smaller standard deviation σ. As inputs grow, the softmax variance approaches its maximum, creating near one-hot distributions that provide minimal gradient signals for learning.

Conceptually (gradient stability):

We would still have a valid cosine similarity but with a higher variance, the softmax would be sharper pushing most values towards 0 or 1, more similar to a one hot distribution, causing gradients to be vanishing becasue there is little difference to optimise and weak trnaing signal. : No scaling → larger scores → sharper softmax → vanishing gradients → poor learning

In terms of gradient stability we want to avoid vanishing or exploding gradients and keep them in a reasonable range. Our gradient is calcuated in regards to the softmax output and hence depends on it. Without the $\sqrt{d_k}$ division and outputs similar to one hot encoding, we would for most 0 values have very small gradients. Giving very small update signal.

1.3.b)

Why does Multi head attention allow learning subspaces of relational patterns?

Each attention head has its own projection matricesW_Q, W_K, V_V, which map the same input X X into different representational subspaces of dimension d_k. In each subspace, the model learns a different notion of similarity between tokens for example, syntactic, semantic, or positional relations.

A single attention head must encode all such relationships in one set of projections, but multiple heads can specialize: one head may focus on local dependencies ("black" → "cat"), another on long-range context or role consistency. Each head therefore captures a distinct relational pattern subspace.

When their outputs $Z_i$are concatenated ($Z_{multi} = [Z_1 \,||\, Z_2 \,||\, ...]$), the model combines these specialized relational subspaces into a richer, compositional representation. This improves expressivity, allowing the model to represent multiple independent types of relationships in parallel.

Problem 2: Learning Dynamics in a Simple RNN (20 Points)

2.1

We are looking to derive the loss function $L = \frac{1}{2}(y - t^*)^2$ as a function of w_h

we have a target output $t^* = 3$, and a prediction $y = vh_2 + c$, hence our loss is:

$$h_1 = w_h h_0 + w_x u_1 + b = 2$$

$$h_2 = w_h(w_h h_0 + w_x u_1 + b) + w_x u_2 + b = w_h 2 + 0.5$$

$$y = vh_2 + c = 4w_h + 1$$

$$L = \frac{1}{2}(y - t^*)^2 = \frac{1}{2}(4w_h - 2)^2 = 8(w_h - 0.5)^2$$

2.2.a)

to find the gradient in regards to w_h:

$$\frac{\partial L}{\partial w_h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h_2}\frac{\partial h_2}{\partial w_h}$$

We find the different partial derviateives:

$$\frac{\partial L}{\partial y} = y - 3 = y - 3$$

$$\frac{\partial y}{\partial h_2} = v = 2$$

$$\frac{\partial h_2}{\partial w_h} = 2$$

$$\begin{aligned}\frac{\partial L}{\partial w_h} &= \frac{\partial L}{\partial y}\frac{\partial y}{\partial h_2}\frac{\partial h_2}{\partial w_h} = (y-3)*2*2 \\ &= 4(y-3) \\ &= 4(4w_h+1)-12 \\ &= 16w_h - 8\end{aligned}$$

We evaluate at $w_h = 0.3$

$$\frac{\partial L}{\partial w_h} = 16 * 0.3 - 8 = -3.2$$

We update w_h, by

$$StepSize = \frac{\partial L}{\partial w_h} \cdot LearningRate = -3.2 \cdot LearningRate$$

$$w_h^{\text{new}} = w_h - StepSize = w_h + 3.2 \cdot LearningRate$$

Therefore the negative sign for $\frac{\partial L}{\partial w_h} = -3.2$, means we will increase the value of w_h deriving w_{h+1}.

2.2.b)

One step gradient decent $w_h = 0.3$:

$$\begin{aligned}w_h^{(\text{new})} &= w_h - \eta\frac{\partial L}{\partial w_h} \\ w_h^{(\text{new})} &= 0.3 - \eta(-3.2) = 0.3 + 3.2\eta\end{aligned}$$

For 3 η values:

$$\begin{aligned}w_h^{(\text{new})} &= 0.3 + 3.2(0.1) = 0.62 \\ w_h^{(\text{new})} &= 0.3 + 3.2(1.0) = 3.5 \\ w_h^{(\text{new})} &= 0.3 + 3.2(3.0) = 9.9\end{aligned}$$

Effect of LR on stability and convergence

For good learning stability we want to be moving smoothly towards the loss minimum. Here we want to avoid oscillating, or diverging. This depends on how large the step size is in relation to the curvature. With a high LR we move fast towards minimum, but we might overstep it and start oscillating by over stepping minimum each update, or stepping out of the local minima we are trying to find. This gives poor learning stability. A small LR gives small but well behaved updates, we need to find the balance bewteen speed of convergence and stabililty. In our example $w_x = 0.5$, when trained. For LR=0.1, we already overstep our minimum at 0.5, and even more so for the larger LR values. In our example especially LR 1 and 3, might struggle to converge.

2.3 Temporal Gradient Propagation in RNNs.

We have gradient:

$$\frac{\partial L}{\partial w_h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h_2}\frac{\partial h_2}{\partial w_h}$$

$$h_t = w_h h_{t-1} + w_x u_t + b$$

we observe that

$$\frac{\partial h_t}{\partial h_{t-1}} = w_h$$

more specifically

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial}{\partial w_h}[w_h h_{t-1}] + 0$$

with the product rule $g(x)f(x) = w_h h_{t-1}$:

$$\frac{\partial h_t}{\partial w_h} = h_{t-1} + w_h \frac{\partial h_{t-1}}{\partial w_h}$$

We add time steps:

$$\frac{\partial h_t}{\partial w_h} = h_{t-1} + w_h(h_{t-2} + w_h \frac{\partial h_{t-2}}{\partial w_h})$$

$$\frac{\partial h_t}{\partial w_h} = h_{t-1} + w_h(h_{t-2} + w_h(h_{t-3} + w_h(h_{t-3} + w_h \frac{\partial h_{t-3}}{\partial w_h})))$$

We notice that for each term we have a multiplication with $w_h$here we will have one term $w_h w_h w_h w_h \frac{\partial h_{t-3}}{\partial w_h}$.

When calcuating the gradient for our last term $\frac{\partial h_2}{\partial w_h}$ we need to propagate through T timesteps, resulting in a multiplication with a w_h term each time. Hence w_h is exponentially dependent on the number time steps T.

For a large nr datapoints N we have w_h^T, if $w_h > 1$ this value explodes, if $w_h < 1$ this value vanishes and if $w_h = 1$ it is stable although this might be unlikely. This is why we have LSTMs

Problem 3 : Optimizer (10 Points)

3.1 Standard SGD.

We compute $\theta^{(2)}$after two updates, starting from $\theta^{(0)} = 1$, $\eta = 0.1$:

$$\theta^{(0)} = 1$$
$$\theta^{(1)} = 1 - 0.1g^1 = 1 - 0.04 = 0.96$$
$$\theta^{(2)} = (1 - 0.1g^1) - 0.1g^2 = 0.96 + 0.02 = 0.98$$

3.2 SGD with Momentum.

$$v_t = \alpha v_{t-1} - \eta g^{(t)}$$
$$\theta_t = \theta_{t-1} + v_t$$

We compute $\theta^{(2)}$:

$$v_0 = 0$$
$$\theta_0 = 1$$

$$v_1 = \alpha v_0 - \eta g^{(1)} = 0.9 * 0 - 0.1 * 0.4 = -0.04$$
$$\theta_1 = \theta_0 + v_1 = 1 - 0.04 = 0.96$$

$$v_2 = \alpha v_1 - \eta g^{(2)} = 0.9 * (-0.04) - 0.1 * (-0.2) = -0.016$$
$$\theta_2 = \theta_1 + v_2 = 0.944$$

3.3 Discussion.

3.3.a)

For the standard SGD:

$$\theta_t = \theta_{t-1} - \eta g_t$$

for SGD with momentum:

$$\theta_t = \theta_{t-1} + \alpha v_{t-1} - \eta g^{(t)}$$

The difference is the term αv_{t-1} this terms becomes negative here because v_0 is 0, and we subtract the gradient: $v_t = \alpha v_{t-1} - \eta g^{(t)}$ which is possitive. In the second step the previous negative gradient also causes v_2 to be negative. Hence the smaller value when this term is added. Ie the previous negative momentum dominates the current upward push.

3.3.b)

Momentum is great if you have a "ravine" or oval shaped minimum which we decend into.

The momentum might help kill oscilation that otherwise would occur.

Problem 4 : Batch Normalization (20 Points)

4.1

What is a bach:

A batch is a subset of the training dataset used in a single iteration of the training process.

Why use baches instead of whole dataset at once?:

- For large datasets its hard to fit all in GPU/CPU memory, hence baching is computationally more feasible. If we have GPUs we can use some paralellisation
- We get more frequent updates to weights, leads to faster convergence.
- Estimations are a bit noicy, this is actually good because it can help us escape local minima, and acts as a form for regualisation.

Bach normalisation

Batch Normalization is a technique that normalizes the inputs to each layer by standardizing activations across a mini-batch. For each feature/channel, it, computes mean and variacne of the bach normalises it by subtracting the mean and dividing by the variances, and applies some learnable parameters γ, β for scaling and shifting the input:

(from dl.ai chapter 8.5) For a minibatch B and for $x \in B$ beeing an input to bach normalisation BN, the normalisation is defined as:

$$BN(x) = \gamma \odot \frac{x - \hat{\mu}_B}{\hat{\sigma_B}} + \beta$$

Batch normalization is applied to individual layers during training, using the statistics of the current minibatch. During prediction, it uses statistics computed from the entire dataset for stability.

Advantages of using BN in traning

- Faster convergeence: It allows for more aggressive (higher) learning rates because it puts parameters on a similar scale and prevents variable magnitudes from diverging.
- Numerical stability: Mitigates exploding / vanishing values by centrning and rescaling to a given mean and variance.
- Regulaisation effect: The noice introduced by the noicy bach mean, compared to the true mean. Can help us escape local minima and reduce overfitting.

4.2

Here is one bach continng 4 samples:

$$H_1 = \begin{bmatrix} 1 & 5 & 3 \\ 7 & 2 & 8 \end{bmatrix}, H_2 = \begin{bmatrix} 4 & 5 & 9 \\ 3 & 5 & 7 \end{bmatrix}, H_3 = \begin{bmatrix} 2 & 8 & 4 \\ 6 & 3 & 5 \end{bmatrix}, H_4 = \begin{bmatrix} 9 & 1 & 6 \\ 4 & 9 & 2 \end{bmatrix}$$

i)

We compute the mean matrix by just taking the element wise mean, f.ex for element $\mu_{0,0} = 1/n \sum x_i = (1 + 4 + 2 + 9)/4 = 4$:

$$\mu = \begin{bmatrix} 4 & 5 & 5.5 \\ 5 & 4.75 & 5.5 \end{bmatrix}$$

ii)

the standard deviation matrix, by first calcualting the variance matrix by finding the variance element wise fex the element $\sigma^2_{0,0} = 1/n \sum (x_i - \mu)^2 = [(1-4)^2 + (4-4)^2 + (2-4)^2 + (9-4)^2]/4 = 38/4 = 9.5$

Then we find the element wise standard deviation as the square root fo the vairance: $\sigma_{0,0} = \sqrt{9.5}$

$$\sigma = \begin{bmatrix} \sqrt{9.5} & \sqrt{6.5} & \sqrt{5.25} \\ \sqrt{2.5} & \sqrt{7.1875} & \sqrt{5.25} \end{bmatrix} \approx \begin{bmatrix} 3.08 & 2.55 & 2.29 \\ 1.58 & 2.68 & 2.29 \end{bmatrix}$$

iii)

Then normalised ouptuts H' for all 4 samples:

$$H'_i = \frac{H_i - \mu}{\sigma}$$

So for element $H'_{1(0,0)}$ element from the first element in H_1: $H'_{1(0,0)} = \frac{1-4}{\sqrt{9.5}} \approx -0.9733$

We repeat element wise for all 4 matrices:

$$H'^{(1)} = \begin{bmatrix} \frac{-3}{\sqrt{9.5}} & 0 & \frac{-2.5}{\sqrt{5.25}} \\ \frac{2}{\sqrt{2.5}} & \frac{-2.75}{\sqrt{7.1875}} & \frac{2.5}{\sqrt{5.25}} \end{bmatrix} = \begin{bmatrix} -0.9735 & 0 & -1.0910 \\ 1.2649 & -1.0261 & 1.0910 \end{bmatrix}$$

$$H'^{(2)} = \begin{bmatrix} 0 & \frac{1}{\sqrt{6.5}} & \frac{3.5}{\sqrt{5.25}} \\ \frac{-2}{\sqrt{2.5}} & \frac{0.25}{\sqrt{7.1875}} & \frac{1.5}{\sqrt{5.25}} \end{bmatrix} = \begin{bmatrix} 0 & 0.3922 & 1.5274 \\ -1.2649 & 0.0932 & 0.6546 \end{bmatrix}$$

$$H'^{(3)} = \begin{bmatrix} \frac{-2}{\sqrt{9.5}} & \frac{3}{\sqrt{6.5}} & \frac{-1.5}{\sqrt{5.25}} \\ \frac{1}{\sqrt{2.5}} & \frac{-1.75}{\sqrt{7.1875}} & \frac{-0.5}{\sqrt{5.25}} \end{bmatrix} = \begin{bmatrix} -0.6490 & 1.1766 & -0.6546 \\ 0.6325 & -0.6529 & -0.2182 \end{bmatrix}$$

$$H'^{(4)} = \begin{bmatrix} \frac{5}{\sqrt{9.5}} & \frac{-4}{\sqrt{6.5}} & \frac{0.5}{\sqrt{5.25}} \\ \frac{-1}{\sqrt{2.5}} & \frac{4.25}{\sqrt{7.1875}} & \frac{-3.5}{\sqrt{5.25}} \end{bmatrix} = \begin{bmatrix} 1.6225 & -1.5688 & 0.2182 \\ -0.6325 & 1.5855 & -1.5274 \end{bmatrix}$$

4.2.3

For the pre activation outputs applied element wise:

$$H'_{test} = \frac{H_{test} - \mu_{train}}{\sigma_{train}}$$

Given:

$$\mu_{train} = \begin{bmatrix} 4 & 5 & 5.5 \\ 5 & 4.75 & 5.5 \end{bmatrix}$$

$$\sigma_{train} = \begin{bmatrix} \sqrt{9.5} & \sqrt{6.5} & \sqrt{5.25} \\ \sqrt{2.5} & \sqrt{7.1875} & \sqrt{5.25} \end{bmatrix}$$

$$H^1_{test} = \begin{bmatrix} 3 & 4 & 6 \\ 5 & 4 & 7 \end{bmatrix}$$

$$H^2_{test} = \begin{bmatrix} 8 & 7 & 5 \\ 6 & 8 & 3 \end{bmatrix}$$

We get:

$$H'^{(1)} = \begin{bmatrix} \frac{-1}{\sqrt{9.5}} & \frac{-1}{\sqrt{6.5}} & \frac{0.5}{\sqrt{5.25}} \\ 0 & \frac{-0.75}{\sqrt{7.1875}} & \frac{1.5}{\sqrt{5.25}} \end{bmatrix} = \begin{bmatrix} -0.3245 & -0.3922 & 0.2182 \\ 0 & -0.2798 & 0.6546 \end{bmatrix}$$

$$H'^{(2)} = \begin{bmatrix} \frac{4}{\sqrt{9.5}} & \frac{2}{\sqrt{6.5}} & \frac{-0.5}{\sqrt{5.25}} \\ \frac{1}{\sqrt{2.5}} & \frac{3.25}{\sqrt{7.1875}} & \frac{-2.5}{\sqrt{5.25}} \end{bmatrix} = \begin{bmatrix} 1.2980 & 0.7844 & -0.2182 \\ 0.6325 & 1.2124 & -1.0910 \end{bmatrix}$$

Problem 5 : CLIP and SigLIP (10 Points)

5.1.a)

$$L = -\frac{1}{N}\sum_i log\frac{exp(S_{ii})}{\sum_j exp(S_{ij})}$$
$$L = -\frac{1}{N}\sum_i log(exp(S_{ii})) - log(\sum_j exp(S_{ij}))$$
$$L = -\frac{1}{N}\sum_i S_{ii} - log(\sum_j exp(S_{ij}))$$

Derivative with regards to S_{ik}:

$$\frac{\partial}{\partial S_{ik}}[-\frac{1}{N}\sum_i S_{ii} - log(\sum_j exp(S_{ij}))] =$$
$$-\frac{1}{N}\sum_i \frac{\partial S_{ii}}{\partial S_{ik}} - \frac{\partial}{\partial S_{ik}} log(\sum_j exp(S_{ij})) =$$

We find the parts:

$$\frac{\partial S_{ii}}{\partial S_{ik}} = 1(k = i)$$
$$\frac{\partial}{\partial S_{ik}} log \sum_j e^{S_{ij}} = \frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} : softmaxProbability \in [0, 1]$$

In all:

$$\frac{\partial L}{\partial S_{ik}} = -\frac{1}{N}\sum_i 1(k = i) - \frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}}$$
$$\frac{\partial L}{\partial S_{ik}} = \frac{1}{N}\sum_i \frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} - 1(k = i)$$

For $k = i$:

$$\frac{\partial L}{\partial S_{ik}} = \frac{1}{N}\sum_i (\frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} - 1)$$
$$\frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} - 1 \leq 0$$
$$\frac{1}{N}\sum (i\frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} - 1) \leq 0$$

For $k \neq i$:

$$\frac{\partial L}{\partial S_{ik}} = \frac{1}{N} \sum_i \frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} - 0$$
$$\frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} \geq 0$$
$$\frac{1}{N} \sum_i \frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}} \geq 0$$

When i=k, we have negative gradient, when $i \neq k$ we have possitive gradient. Ie we dont decend toward a local minima when pairs dont match, ie we "punnish" the model for beeing wrong and reward for beeing right.

5.1.b)

Effect of temperature τ on:

i) Traning stability

For small τ, the number $S_{ij} = \frac{\ddot{f_i}^{\mathrm{T}} g_j}{\tau}$ becomes large. Which makes softmax outputs close to 0 or 1. Which means gradients concentrate on a few examples and can have large magnitudes.

Large τ gives small $S_{ij} = \frac{\ddot{f_i}^{\mathrm{T}} g_j}{\tau}$ which may be better for getting probabilities closer touniform and more stable but it may make it harder to separate possitives and negatives.

A balance has to be found for an optimal τ

ii) Similarity Sharpness

Small τ give shapr similarities and larger margins between positives and negatives in embedding space.

Large τ gives smoother similarities but less separation (lower sharpness) os τ controls how peaky the models simlarity distribution is.

5.2.a) How SigLip objective changes handling of negative pares compared to

Siglip replaces bach wise softmax with independent sigmoid activations.

for CLIP we had objective:

$$L = -\frac{1}{N} \sum_i S_{ii} - log(\sum_j exp(S_{ij}))$$

For sigLip we have objective:

$$L_{SigLip} = -\frac{1}{N}\sum_{i}\left[log\sigma(S_{ii}) + \sum_{j\neq i} log(1-\sigma(S_{ij}))\right]$$

If we would look at $\frac{\partial L}{\partial S_{ik}}$ the first term $log\sigma(S_{ii})$ would vanish for $k \neq i$ since it does not have a gradient in realtion to negative paris only possitive. The second part of the expression $\sum_{j\neq i} log(1-\sigma(S_{ij}))$ does depend on the negative pairs. We can assume that the basic funcitonality of contributing to increased gradient is ture here as well, since the funciton is derived as a traning objective. However more notably we notice that rather than a group wise softmax $\frac{e^{S_{ik}}}{\sum_j e^{S_{ij}}}$ as for the CLIP objective, we have an individual level sigmoid application for each pair $\sum_{j\neq i} log(1-\sigma(S_{ij}))$. So the main difference is that CLIP uses a softmax based on contrastive loss while SigLip uses the individually applied Sigmoid in the handelling of negative pairs.

5.2.b)

i) Why SigLip objective less sensitive to bach size

Bach size in clip imacts the softmax term, since it is calcuating a probability distribution based on the bach samples. For a large number of samples the individual probability is lessend. Ie adding or removing other samples impacts the gradient for a given pair.

siglip however applies a sigmoid individually which function does not depend on the batch size. Adding or removing other samples does not imact the gradient for a given pair.

ii) what potential drawback might this introduce for global similarity calibration

Without the batch-wise softmax normalization used in CLIP, SigLIP lacks a global reference for similarity magnitudes. This can reduce global similarity calibration, similarities may not be directly comparable across batches or samples, making the absolute scale of scores less consistent for retrieval or ranking tasks.

Problem 6 : Autoencoder & Variational-Autoencoder (VAE) (20 Points)

6.1.a) Traning objective of AE and name

Training Objective: The training objective of an autoencoder is to reconstruct its input as accurately as possible. Mathematically, it minimizes the reconstruction loss.

The term "Auto" refers to the fact that the network learns in an unsupervised manner using the input data itself as the target. Unlike supervised learning

where we need separate labels, the autoencoder automatically generates its own training signal by trying to reproduce its input.

The term "Encoder" refers to the first part of the network that compresses or encodes the input data into a lower-dimensional representation.

6.1.b)

It is called the latent layer, latent space, bottleneck layer, or code layer. This layer contains the compressed representation of the input data, capturing the most essential features in a lower dimensional space.

6.1.c)

The smaller dimension creates an information bottleneck that forces the network to learn a compressed representation capturing only the most important features. If the middle layer were equal or larger in size, the network could simply learn an identity mapping without discovering meaningful patterns. Advantages are that it compresses high dimentiojanl data into compact represtations. It forces the network to extract the most important features wihtout manual feature engineering. It also prevents overfitting by limiting the capacity to meorize deatils which forces it to learn more general patterns that transfer better to unseen data.

6.2.a) Main difference AE vs VAE

AE is not generative since in between sample points in its latent space does not generally make sense. Hence we can not f.ex randomly sample from the latent space and decode into an image. Hence the latent space is not a function we can use to generate new samples. Latent space is discountinous.

However in VAE we use paramters of probability distributions to construct the latent space. This allows us to sample from it and decode into “new” generated samples following our posterior distribition approxiamtion. Latent space is continous and smooth.

The training objective differs, both uses a reconstructive L2 loss, however for the VAE we also add a KL divergence to enforce a smooth probabilistic latent space.

In all we can use AE for dimtnionality reduciton but VAE also for generation.

6.2.b) Why introduce this difference?

Because it is desireable to be able to generate images, VAE is used within modern diffusion architectures.

6.2.c) Explain bayes in context of generative models

Bayes:

$$P(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}$$

- $p(y \mid x)$: **Posterior**, what we try to approxiamte: Probability of the class (or label) y given the observed data x. This is what we want to predict e.g., the probability an image is a cat given its pixels.
- $p(x \mid y)$: **Likelihood :** Probability of observing data x given class y. Describes how the data is generated for each class e.g., how cat images are distributed.
- $p(y)$: **Prior :** Prior probability of each class y before seeing any data. Represents how common each class is e.g., how frequent cats are overall.
- $p(x)$: **Evidnce**, or margianl likelihood. : Total probability of observing x under all possible classes. Acts as a normalization factor ensuring probabilities sum to 1

In a generative model, we model p(x|y) p(x|y) and p(y) p(y) i.e how data is generated for each class and how likely each class is. Then we use Bayes' theorem to compute the posterior p(y|x) p(y|x) for prediction.

6.2.d) Loss funciton of VAE

We need to go from:

$$logp(x) = logp(x) \int q_\phi(z \mid x)$$

to:

$$logp(x) = E_{q_\phi(z|x)} \left[log \frac{p(x,z)}{q_\phi(z \mid x)} \right] + D_{KL}(q_\phi(z \mid x) \mid\mid p(z \mid x))$$

DERIVING

We have for any probability distribution probability mass=1, so for our posterior:

$$\int q(z \mid x)dz = 1$$

We hence have:

$$logp(x) = logp(x) \cdot 1 = logp(x) \int q(z \mid x)dz$$

The definition of joint probability, p observing p x and z is p of observing x, time p of observing z given that we observed x.

$$p(x,z) = p(z \mid x)p(x) \Rightarrow p(x) = \frac{p(x,z)}{p(z \mid x)}$$

We substitute:

$$\begin{aligned} logp(x) &= logp(x) \int q(z \mid x) dz \\ &= log\left(\frac{p(x,z)}{p(z \mid x)}\right) \int q(z \mid x) dz \end{aligned}$$

Since the $logp(x) = log\left(\frac{p(x,z)}{p(z|x)}\right)$ is a constant we can put it inside the integral:

$$logp(x) = \int log\left(\frac{p(x,z)}{p(z \mid x)}\right) q(z \mid x) dz$$

Since $q(z \mid x)$ has probability mass =1 we can write hte expression as a weighted average or expectation:

$$\begin{aligned} logp(x) &= E_{q(z|x)}\left[log\left(\frac{p(x,z)}{p(z \mid x)}\right)\right] \\ logp(x) &= E_{q(z|x)}\left[logp(x,z) - logp(z \mid x)\right] \end{aligned}$$

We know that $E[A - B] = E[A] - E[B]$

$$\begin{aligned} logp(x) &= E\left[logp(x,z)\right] - E[logp(z \mid x)] \\ &\quad (E[logq(z \mid x)] - E[logq(z \mid x)] = 0) \\ logp(x) &= E\left[logp(x,z)\right] - E[logp(z \mid x)] + E[logq(z \mid x)] - E[logq(z \mid x)] \\ logp(x) &= (E[logp(x,z)] - E[logq(z \mid x)]) + (E[logq(z \mid x)] - E[logp(z \mid x)]) \end{aligned}$$

We have for the first term:

$$\begin{aligned} E[logp(x,z)] - E[logq(z \mid x)] &= \\ E\left[logp(x,z) - E[logq(z \mid x)\right] &= \\ E\left[log\left(\frac{p(x,z)}{q(z \mid x)}\right)\right] &= \end{aligned}$$

For the second term:

$$\begin{aligned} E[logq(z \mid x)] - E[logp(z \mid x)] &= \\ E\left[logq(z \mid x) - logp(z \mid x)\right] &= \\ E\left[log\left(\frac{q(z \mid x)}{p(z \mid x)}\right)\right] &= \\ D_{KL}(q(z \mid x) \,||\, p(z \mid x)) &= \end{aligned}$$

Together:

$$logp(x) = (E[logp(x,z)] - E[logq(z \mid x)]) + (E[logq(z \mid x)] - E[logp(z \mid x)])$$
$$logp(x) = E\left[log\left(\frac{p(x,z)}{q(z \mid x)}\right)\right] - D_{KL}(q(z \mid x) \mid\mid p(z \mid x))$$