Homework 4

Problem1

$\frac{\partial y}{\partial \vec{x}}$, $\frac{\partial \vec{y}}{\partial x}$, $\frac{\partial \vec{y}}{\partial \vec{x}}$, where $x, y \in R$, $\vec{y} \in R^m$, $\vec{x} \in R^n$

$$\frac{\partial y}{\partial \vec{x}} = \left[\frac{\partial y}{\partial x_1} \ \frac{\partial y}{\partial x_2}, \cdots, \frac{\partial y}{\partial x_n} \right] \in R^{1\times n}$$

$$\frac{\partial \vec{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \vdots \\ \frac{\partial y_n}{\partial x} \end{bmatrix} \in R^{m\times 1}$$

$$\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_n} \end{bmatrix} \in R^{m\times n}$$

Problem 2

- basic info : name : DeepSeek-R1

  version : R1

  purpose : A large language model used for tasks like text generation, reasoning, mathematics, and code generation

  realease date : January 10, 2025

  reference : https://en.wikipedia.org/wiki/DeepSeek_%28chatbot%29?utm_source=chatgpt.com

- size : number of parameters : DeepSeek-R1 contains 670 billion parameters,

  architecture : Deepseek-R1 is built upon the Transformer architecture, enhanced with a Mixture of Experts (MoE) design. The model comprise 61 Transformer layers in total. It also adopts Multi-Head Latent Attention (MLA) across all layers.

  reference : https://pub.towardsai.net/deepseek-r1-model-architecture-853fefac7050

• training data:

Pretraining Data: DeepSeek-R1 was built upon DeepSeek-V3-Base, which underwent pretraining on a multilingual corpus comprising approximately 14.8 trillion tokens. This dataset predominantly featured English and Chinese texts and included a higher proportion of mathematics and programming-related content compared to previous versions.

Fine-Tuning Data: Following pretraining, DeepSeek-R1 underwent Supervised Fine-Tuning (SFT) using a dataset of 1.5 million instruction examples. This dataset encompassed both reasoning tasks (such as mathematics, programming, and logic) and non-reasoning tasks (including creative writing, roleplay, and simple question answering). The reasoning data was generated by specialized "expert models," while the non-reasoning data was produced by DeepSeek-V2.5 and validated by human reviewers.

reference: https://arxiv.org/pdf/2501.12948 (DeepSeek-R1 Technical Report)

• training time: In H800 GPU Hours: Total: 2788 K

Pre-Training: 2664K, Contex Extension: 119K Post-Training: 5K

• Hardward:

Training hardware: 2048 Nvidia H800 GPUs

GPU Acquisition Costs: $35.8M-143.4M (Market price for H800 GPU $17500 to $70000)

reference https://www.linkedin.com/pulse/unveiling-true-costs-behind-deepseek-r1-gwri-pennar-l0yye/

https://en.wikipedia.org/wiki/DeepSeek_%28chatbot%29?utm_source=chatgpt.com

energy cost: DeepSeek-R1 achieve training with just $5.58M

From U.S. Energy Information Administration,

average us. industrail electricity price = $0.085 per KWh.

Total Energy $= \frac{5,580,000}{0.085} = 65,647,059$ KWh

Each family electricity cost = 899 KWh per month

$65,647,059 \div 899 \times 12 \approx 6085$ Families electricity cost per year

Cost of operation: electricity, GPU, software, staff, ...

GPU Acquisition Costs: $31.8M - 143.4M

Operational Costs: including infrastructure, energy consumption, data preprocessing, etc.

Development cost: expenses for staff, software, and research further inflate the budget

Multiple Training Runs: Assuming 5-10 runs at $6M each

Combining these factors, the true cost of developing DeepSeek-R1 likely exceeds $100M to $150M.

reference: https://www.linkedin.com/pulse/unveiling-true-costs-behind-deepseek-r1-gwri-pennar-l0yye/

- interesting find:

One particular suprising fact is the extreme cost efficient of DeepSeek-R1. Despite its massive scale (671B), the total training cost was only $5.58M which is over 90% cheaper than GPT-4's estimated $500M training cost. And its performance is higher than GPT-4 and it performs on par with the previous version of o1 model.

Problem 3.

(a) $x = 2$, $z^{(0)} = \begin{pmatrix}1\\2\end{pmatrix}$

$a_1^{(1)} = 0.1\times 1 + 0.3\times 2 = 0.7$

$z_1^{(1)} = \tanh(a_1^{(1)}) = \tanh(0.7) \approx 0.604$

$a_2^{(1)} = 0.2\times 1 + 0.4\times 2 = 1$

$z_2^{(1)} = \tanh(a_2^{(1)}) = \tanh(1) \approx 0.762$

$a^{(1)} = \begin{pmatrix}0.7\\1\end{pmatrix}$ $\quad z^{(1)} = \begin{pmatrix}0.604\\0.762\end{pmatrix}$

$a^{(2)} = 0.2\times 1 + 1\times 0.604 + (-3)\times 0.762 = -1.482$

$z^{(2)} = \tanh(a^{(2)}) = \tanh(-1.482) = -0.902$

(b) $y = 1$

squared loss : $L = \frac{1}{2}(z^{(2)} - y)^2 = \frac{1}{2}\times(-0.902-1)^2 = 1.808$

$\delta^{(2)} = \frac{\partial L}{\partial a^{(2)}} = (z^{(2)} - y)\cdot(1 - \tanh^2(a^{(2)}))$

$= (-0.902-1)(1-(-0.902)^2) = -0.355$

$\delta_1^{(1)} = (1-\tanh^2(0.7))\times 1\times(-0.355) = (1-(0.604)^2)\times(-0.355) = -0.225$

$\delta_2^{(1)} = (1-\tanh^2(1))\times(-3)\times(-0.355) = (1-0.762^2)\times 1.065 = 0.447$

$\Delta W^{(2)} = \delta^{(2)}(z^{(1)})^T = -0.355\times[0.604 \quad 0.762] = [-0.214 \quad -0.271]$

$\Delta b^{(2)} = \delta^{(2)} = -0.355$

$z^{(0)} = \begin{pmatrix}1\\2\end{pmatrix}$

$\Delta W^{(1)} = \delta^{(1)}(z^{(0)})^T = \begin{pmatrix}-0.225\\0.447\end{pmatrix}(1, 2) = \begin{pmatrix}-0.225 & -0.450\\0.447 & 0.894\end{pmatrix}$

$\Delta b^{(1)} = \begin{pmatrix}-0.225\\0.447\end{pmatrix}$

Problem 4

(a) From the diagram, the final layer L contains m_L units. Thus, the number of classes is m_L.

For multi-class classification, we can use the softmax function at the output layer. It converts raw scores into a probability distribution $z_i^{(L)} = \frac{e^{a_i^{(L)}}}{\sum_{j=1}^{m_L} e^{a_i^{(L)}}}$, each $z_i^{(L)} \in (0.1)$. $\sum_{i=1}^{m_L} z_i^{(L)} = 1$

So, the output vector $\vec{z}^{(L)}$ is a valid probability vector

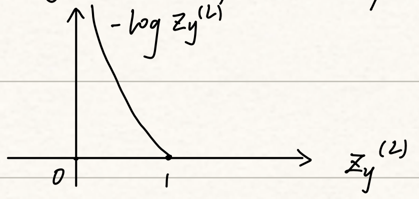
$\vec{z}^{(L)} = (z_1^{(L)}, z_2^{(L)}, \cdots, z_{m_L}^{(L)})$

(b) $\vec{a}^{(L)} = W^{(L)} \vec{z}^{(L-1)} + \vec{b}^{(L)}$

$\vec{z}^{(L)} = g_L(\vec{a}^{(L)}) = g_L(W^{(L)} \vec{z}^{(L-1)} + \vec{b}^{(L)})$

$\hat{y}_i = \arg\max_j z_j^{(L)}$

(c) $L(\vec{x}, y) = -\sum_{i=1}^{m_L} I_{[i=y]} \log z_i^{(L)} = -\log z_y^{(L)}$, where y is the true class label, and $z_y^{(L)}$ is the predicted probability for class y.



(d) $\delta_i^{(L)} = \frac{\partial L}{\partial a_i^{(L)}} = z_i^{(L)} - I_{[i=y]}$

$\frac{\partial L}{\partial W_{ij}^{(L)}} = \frac{\partial L}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial W_{ij}^{(L)}} = (z_i^{(L)} - I_{[i=y]}) \cdot z_j^{(L)}$

Hence, the weight update rule is $W_{ij}^{(L)} \leftarrow W_{ij}^{(L)} - \eta (z_i^{(L)} - I_{[i=y]}) \cdot z_j^{(L)}$

where η is the learning rate.

(e) Using the chain rule, we have $\delta_j^{(L-1)} = \frac{\partial L}{\partial a_j^{(L-1)}} = \sum_{k=1}^{m} \frac{\partial L}{\partial a_k^{(L)}} \cdot \frac{\partial a_k^{(L)}}{\partial z_j^{(L-1)}} \cdot \frac{\partial z_j^{(L-1)}}{\partial a_j^{(L-1)}}$

since $a_k^{(L)} = \sum_j W_{kj}^{(L)} z_j^{(L-1)} + b_k^{(L)}$, $z_j^{(L-1)} = g_{L-1}(a_j^{(L-1)})$

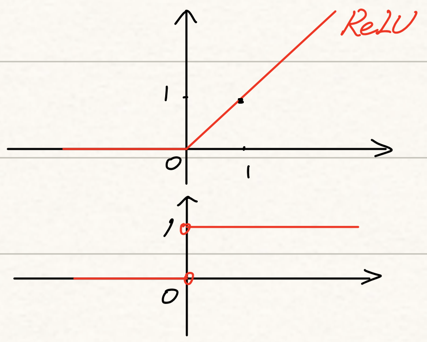
$\frac{\partial a_k^{(L)}}{\partial z_j^{(L-1)}} = W_{kj}^{(L)}$, $\frac{\partial z_j^{(L-1)}}{\partial a_j^{(L-1)}} = g_{L-1}'(a_j^{(L-1)})$

Thus, $\delta_j^{(L-1)} = (\sum_{k=1}^{m_L} \delta_k^{(L)} \cdot W_{kj}^{(L)}) \cdot g_{L-1}'(a_j^{(L-1)})$

Problem 5

(a) $\mathrm{ReLU}(x) = \begin{cases} 0, & x \le 0 \\ x, & x > 0 \end{cases}$

$\frac{d}{dx}\mathrm{ReLU}(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$



(b) The vanishing gradient problem refers to a phenomenon in neural networks where, during backpropagation, the gradients following back to earliers layers become extremely small. As the gradients become very small in earlier layers, the parameters in those layers barely update, making training extremely slow or stuck.

ReLU largely mitigate the vanishing gradient problem. When $x<0$ the ReLU derivate is 1, which means there is no gradient attenuation from positive inputs. When $x<0$, the derivative is 0, and it sets that neuron's gradient to zero outright. This effect does not typically cause vanishing across many layers. it instead results in some neurons never activating if their inputs remain negative.

(c) A dying neuron is one whose output is consistently zero during training. ReLU suffers from this issue. For ReLU-based networks, if a neuron's input becomes negative for all training examples, that neuron's output remains zero for those inputs, because $\frac{d}{dx}\mathrm{ReLU}(x)=0$ for $x<0$. Once a neuron becomes consistently negative, the gradient through that neuron is 0; so its weights will not update to revive it back into a positive input region.

(d) Replace ReLU with leaky ReLU, where leaky ReLU = $\begin{cases} \alpha x, & x<0 \\ x, & x \geq 0 \end{cases}$,

where α is a small positive slope. This ensures that even for negative x, the derivate is α instead of 0. Hence, the neuron never truely dies, as gradients can still propagate even when the unit is in the negative region.

Problem 6

(a) $f(x) = x^3$, $x=2$, $\varepsilon = 0.001$

$$\frac{\delta f}{\delta x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}$$

$$f'(x) = 3x^2, \qquad f'(2) = 3 \cdot 2^2 = 12$$

$$f(2+0.001) = f(2.001) = 2.001^3 \approx 8.012006001$$

$$f(2-0.001) = f(1.999) = 1.999^3 \approx 7.988005999$$

$$\frac{f(2.001) - f(1.999)}{2 \times 0.001} = \frac{8.012006001 - 7.988005999}{0.002} = 12.000001$$

(b) $f(w_{11}^{(l)}, \cdots, w_{m_{l-1}m_l}^{(l)}, b)$

$$\frac{\partial f}{\partial w_{ij}^{(l)}} = \lim_{\varepsilon \to 0} \frac{f(w_{11}^{(l)}, \cdots, w_{ij}^{(l)}+\varepsilon, \cdots, b) - f(w_{11}^{(l)}, \cdots, w_{ij}^{(l)}-\varepsilon, \cdots, b)}{2\varepsilon}$$

$$\frac{\partial f}{\partial b_k} = \lim_{\varepsilon \to 0} \frac{f(w, \cdots, b_k+\varepsilon, \cdots) - f(w, \cdots, b_k-\varepsilon, \cdots)}{2\varepsilon}$$