Well Logs Interpolation and Uncertainty Analysis Using Neural Processes

Meng Jia
Mentored by Haibin Di

Geophysics Techonology Center, Schlumberger, Houston, Texas

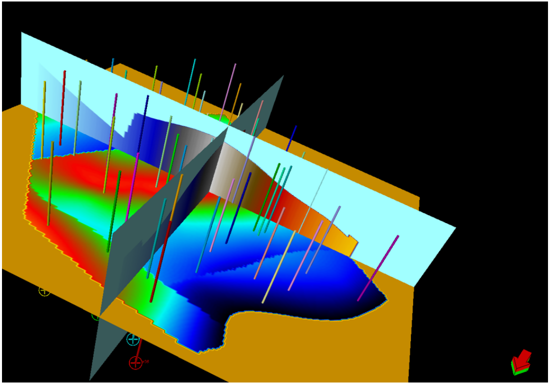
MJia@slb.com

July 30, 2021

Overview

Problem Description

- We aim to predict (or interpolate) the well log properties for an entire site cube from sparsely available well logs and to use seismic data as reference.
- In addition, we want to quantify the uncertainty of the predictions.



Problem Description

- We can formulate the problem as finding $\hat{f}$, a good approximation of f, the true function. We then use $\hat{f}$ for prediction and quantify the uncertainty

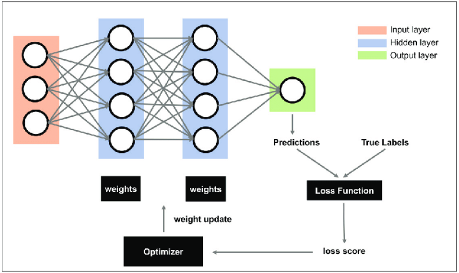
$$y = f(\mathrm{x}) = f(i, j, k, s)$$

where y is some well property, i, j, k are the coordinates, s denotes the seismic data.

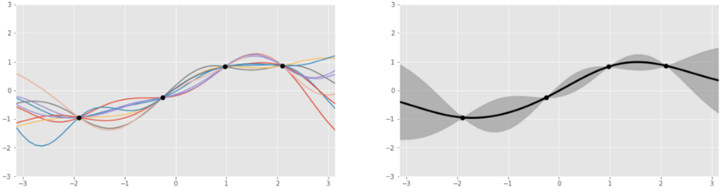
- The task is two-fold:
  - Find $\hat{f}$, a good approximation of f which is used for prediction.
    - Neural networks
  - Quantify the uncertainty of new predictions, i.e., $SD(\hat{f}(\mathrm{x}^*))$, where x^* denotes a new observation point.
    - MC dropouts
    - Gaussian processes

Neural Networks and Gaussian Processes

- A neural network (NN) is a parameterized function that can be tuned via gradient descent to approximate the ground truth.
- Parameters (or weights) are updated in a back-propagation pass. New predicted value is computed in a forward pass.



- A Gaussian process (GP) is a probabilistic model that defines a distribution over possible functions, and is updated in light of data via the rules of probabilistic inference.



Neural Networks and Gaussian Processes

- A GP can also be considered as a random function

$$F : \mathcal{X} \mapsto \mathcal{Y}$$

such that for any finite sequence $\mathrm{x}_{1:n} = (\mathrm{x}_1, \ldots, \mathrm{x}_n)$, $(F(\mathrm{x}_1), \ldots, F(\mathrm{x}_n))$ follows a multivariate normal distribution.

- A GP is fully determined by
  - a mean function $\mathrm{m}(x) \in R^n$ and
  - a covariance or kernel function $K(x, x')) \in R^{n \times n}$

written as:

$$F \sim GP(\mathrm{m}(x), K(x, x'))$$

- For a particular instantiation $\mathrm{x}_{1:n} = (\mathrm{x}_1, \ldots, \mathrm{x}_n)$:

$$(F(\mathrm{x}_1), \ldots, F(\mathrm{x}_n)) \sim MN(\mathrm{m}(\mathrm{x}_{1:n}), K(\mathrm{x}_{1:n}, \mathrm{x}'_{1:n}))$$

Neural Networks and Gaussian Processes

- A GP defines a prior over functions, which can be converted into a posterior over functions once we have seen some data.

Theorem

Let $f = (F(x_1), \dots F(x_n))$ be the data we have seen (or called context points) and $f^* = (F(x_1^*), \dots, F(x_m^*))$ the data we want to predict (or called target points). In addition, the prior over $(f, f^*)^T$ is given by:

$$\begin{pmatrix} f^* \\ f \end{pmatrix} \sim MN\left(\begin{pmatrix} \mu_* \\ \mu \end{pmatrix}, \begin{pmatrix} K(\mathrm{x}_i, \mathrm{x}_j) & K(\mathrm{x}_i, \mathrm{x}_k^*) \\ K(\mathrm{x}_k^*, \mathrm{x}_i), & K(\mathrm{x}_k^*, \mathrm{x}_l^*) \end{pmatrix}\right)$$

Then the posterior for the unseen data f^* conditioned on f is given by:

$$f^*|f \sim MN(\mu_{f^*|f}, \Sigma_{f^*|f})$$

where

$$\mu_{f^*|f} = K(\mathrm{x}_k^*, \mathrm{x}_i)K^{-1}(\mathrm{x}_i, \mathrm{x}_j)\mathrm{f}$$

$$\Sigma_{f^*|f} = K(\mathrm{x}_k^*, \mathrm{x}_l^*) - K(\mathrm{x}_k^*, \mathrm{x}_i)K^{-1}(\mathrm{x}_i, \mathrm{x}_j)K(\mathrm{x}_i, \mathrm{x}_k^*)$$

Neural Networks and Gaussian Processes

Neural Network

Pros

- Predictions are accurate.
- Computational complexity is $O(n)$ in the inference process where n is the number of data points.

Cons

- Only one single approximated function is found.
- Cannot be adapted once the training phase is done.

Neural Networks and Gaussian Processes

Neural Network

Pros

- Predictions are accurate.
- Computational complexity is $O(n)$ in the inference process where n is the number of data points.

Cons

- Only one single approximated function is found.
- Cannot be adapted once the training phase is done.

Gaussian Processes

Pros

- A distribution over potential functions is produced $\rightarrow$ uncertainty analysis
- Shift some of the workload from training to testing time $\rightarrow$ more model flexibility

Cons

- The kernel needs to be given prior to fitting the GPs model.
- Computational complexity is $O(n^3)$ in the inference process.

Attentive Neural Processes

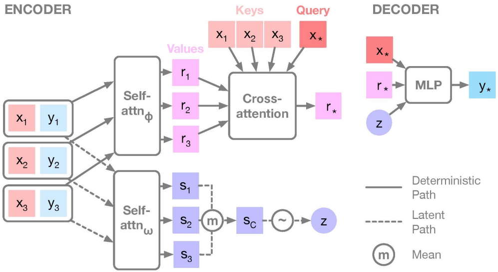
- **Basic Idea**
  Since a Gaussian process can be considered as a random function, we approximate a Gaussian process $y \sim \mathcal{GP}(\mathrm{x})$ by:

  $$y = g(\mathrm{x}, \mathrm{z})$$

  where z is a high-dimensional latent random vector capturing all the randomness in the GP, and is assumed to be multivariate normal; $g()$ is a fixed and learnable function. Our goal is to find

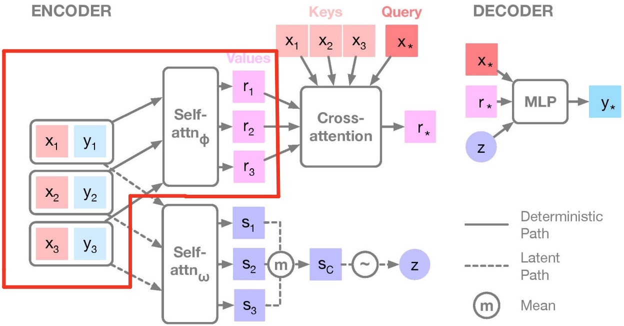
  $$p(y|\mathrm{x}, \mathrm{z})$$

- **Model Overview**
  - Encoder
    - Deterministic Encoder
    - Cross Attention
    - Latent Encoder
  - Decoder

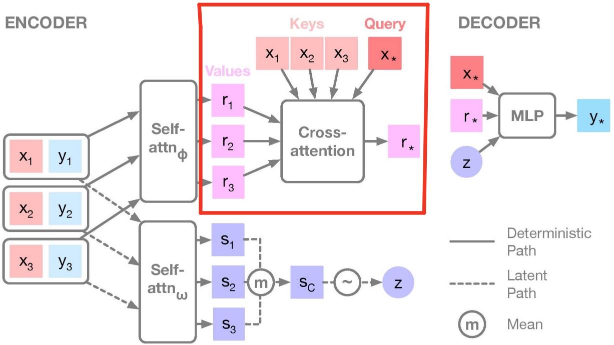


Kim et.al. 2019

- **Model - Deterministic Encoder**
  Compute representations of each each (x, y) pair in the context set through a self attention model, i.e., $r_i = h_\phi(x_i, y_i), i \in C$

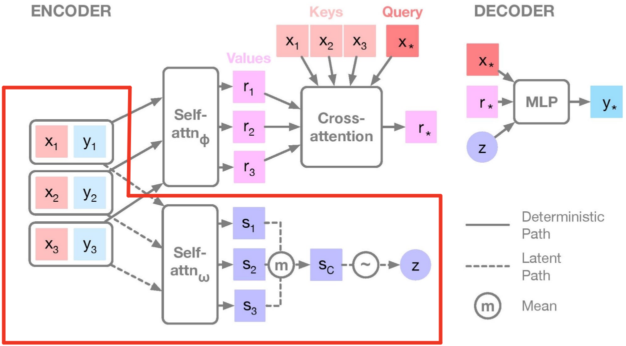


- **Model - Cross Attention**
  Compute the representation for each target query x_* by attending to all the context $x_C = (x_i)_{i \in C}$ using cross-attention mechanism, i.e., $r_* = r^*(x_C, r_C, x_*)$
  - Here we choose multihead as our cross-attention model.

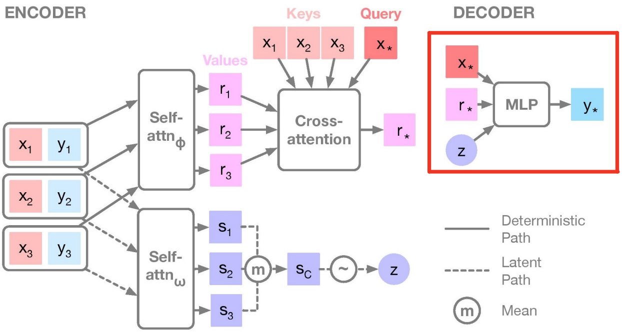


Attentive Neural Processes

- **Model - Latent Encoder**
  Find the distribution of the high-dimensional random vector z representing the global randomness of the system.
  - Here we assume z follows a multivariate normal distribution.



- **Model - Decoder**
  Compute the distribution of the target output $p(y_*|x_*, r_C, z)$
  - $p(y_*|x_*, r_C, z)$, by definition, is a multivariate normal distribution.



Attentive Neural Processes

- **Loss Function**

$$\max\{\log p(y_T|x_T, x_C, y_C)\}$$

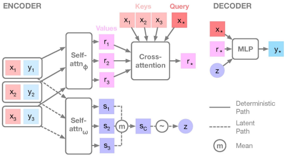
which, by variational inference, is equivalent to

$$\max\{\mathbb{E}_{q(z|x_C,y_C)}[\log p(y_T|x_T, z)] - \mathbf{KL}\left(q(z|x_T, y_T)\|q(z|x_C, y_C)\right)\}$$

- **Data Description**
  We use a $700 \times 327 \times 397$ (vertical $\times$ crossline $\times$ inline) cube of synthetic data to benchmark our model. There are 36 well logs in the cube recording porosity values.
  - # points in total $= 90,873,300$
  - # seismic point $= 18,167,447$
  - # porosity point $= 12159$ (in 36 well logs)
- **Input & Output**
  - Input: $\mathrm{x} = (i, j, k, \mathrm{flatten}(s_{\mathrm{cube}}))$
  - Output: y

Variable Name	Description
$(i, j, k) \in \mathbb{R}^3$	Location indices of the target point, vertical, crossline, and inline, respectively
$s \in \mathbb{R}$	Seismic value at the target point
$s_{\mathrm{cube}} \in \mathbb{R}^{7\times5\times5}$	A $7\times5\times5$ (vertical $\times$ crossline $\times$ inline) seismic cube centered at the target point
$y \in \mathbb{R}_+$	Porosity value at the target point

- **Model Architecture**



Model Component	Structure
Deterministic Encoder	sizes = [128, 128, 128, 128] activation function = [relu, relu, relu, dense]
Cross Attention	sizes = [128] activation function = [Conv1D] # heads = 16
Latent Encoder	sizes = [128, 128, 128, 128] activation function = [relu, relu, relu, dense]
Decoder	sizes = [128, 128, 2] activation function = [relu, relu, dense]

- **Other Setups**
    - Scale i, j, k to range $[0, 1]$ and amplify seismic values by 10.
    - Set 0.001 as an lower bound of $SD(\hat{y}_*)$
    - # training iterations = 500,000

Results - Training

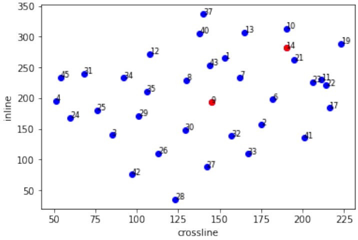
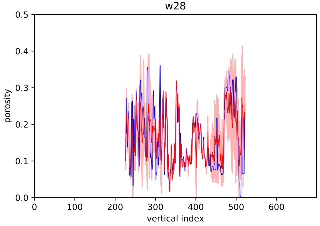
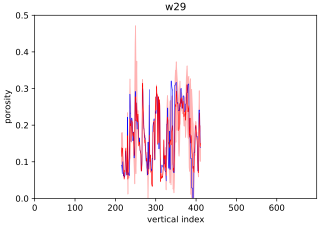
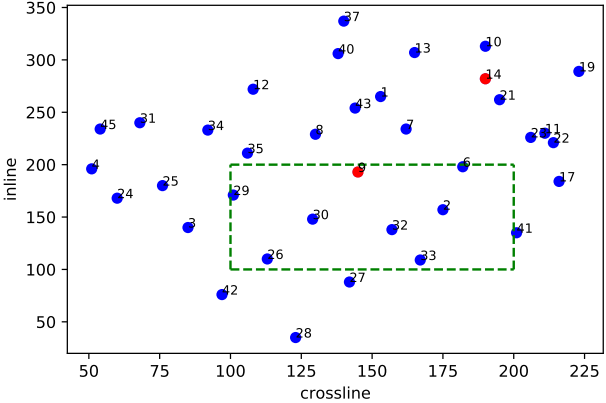


Figure: Training and validation data distribution

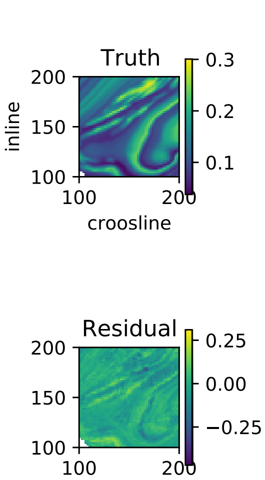




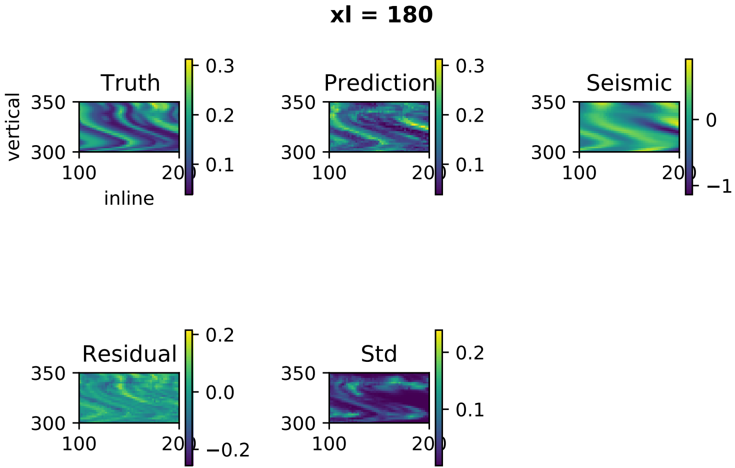
- Test Data Distribution
  Depth range = [300, 350]



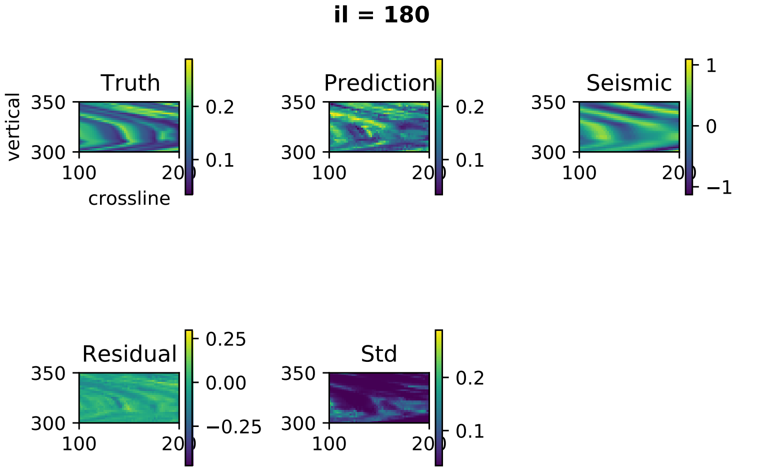
- Example - a slice at depth $= 310$



- Example - a slice at crossline = 180

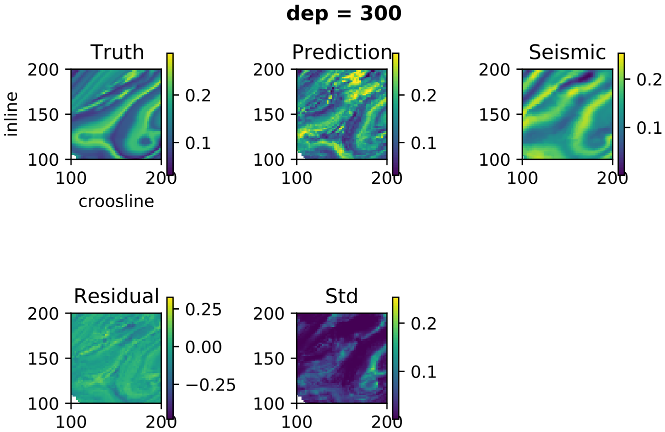


- Example - a slice at inline = 180

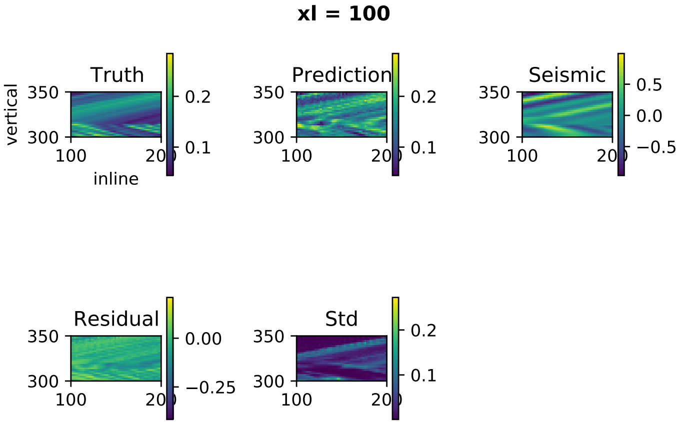


Results - Slices

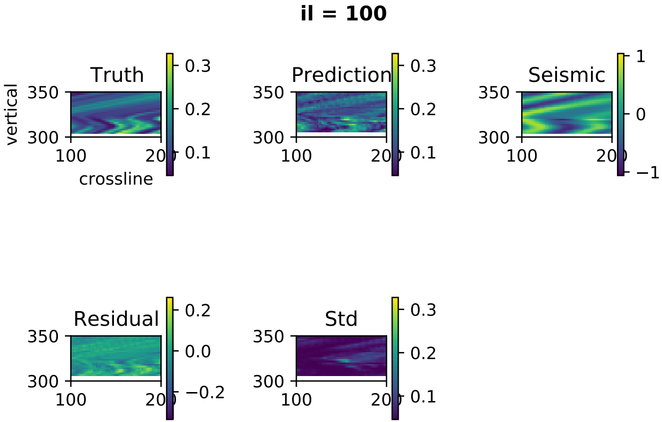
- Results at all depths



- Results at all crossline indices

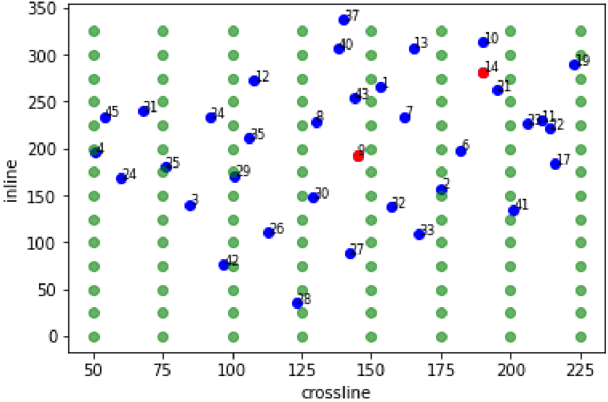


- Results at all inline indices



Results - Vertical Logs

- Test Data Distribution



- Examples

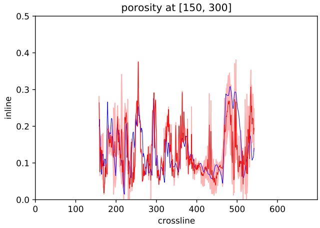


Figure: A good example

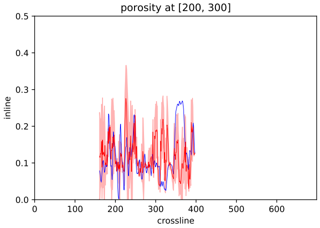
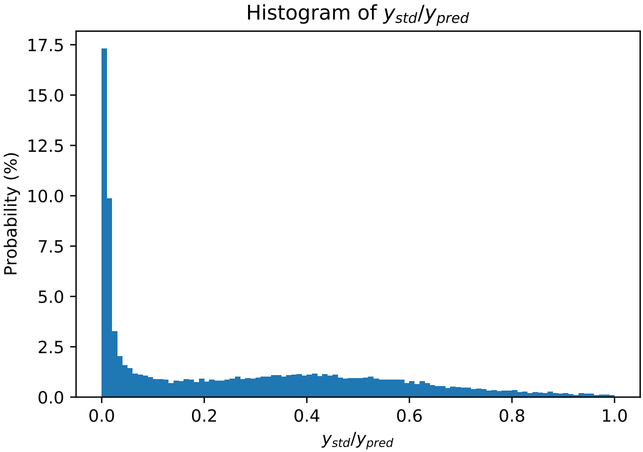


Figure: A bad example

Results - Uncertainty Analysis

- Histogram of y_{std}/y_{pred}



- Confidence Interval (CI) Coverage

$$y_{true} \in [y_{pred} \pm z_{\alpha} * y_{std}]$$

Confidence Level (α)	Z_{α}	Coverage (%)
.95	1.960	89.75
.90	1.645	85.32
.68	1.000	59.39

Conclusions and Future Directions

- Conclusions
  - Attentive neural processes is an adaptive and computationally efficient model for prediction and the corresponding uncertainty analysis.
  - The use of seismic patch as input and cross-attention improve the prediction accuracy.
  - The standard deviation of the predicted values is highly correlated with the residuals, which is unexpected in Gaussian processes-based models.
  - The estimated standard deviation is reasonable in terms of the CI coverages.
- Future Work
  - More inspections on the standard deviation of the predictions.
  - Run inference for the entire cube (computational issues occurred on Google Cloud Computation Platform)
  - Apply the model to Groningen data
  - Adapt the model for few-shots regression
  - Convolutional neural processes

References

- Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D.J., Eslami, S.M. and Teh, Y.W., 2018. Neural processes. arXiv preprint arXiv:1807.01622.
- Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O. and Teh, Y.W., 2019. Attentive neural processes. arXiv preprint arXiv:1901.05761.
- `https://github.com/deepmind/neural-processes`

Acknowledgement

- Haibin, for being a good and kind mentor and all the help along the way
- Zhun, for all the discussions and precious suggestions
- Hiren, Aria, for insights into my project
- Anisha, Tao, for setting up computational resources
- Everyone in the group, for every interesting topics in the casual meetings

Thank you!

Appendix - Multihead Attention

Let Q, K, V denote the query, keys, and values in the query system, respectively.

$$\begin{aligned} \textbf{MultiHead}(Q,K,V) &= \text{concat}(\text{head}_1,\dots,\text{head}_H)W \\ \text{head}_h &= \textbf{DotProduct}(QW_h^Q, KW_h^K, VW_h^V) \\ \textbf{DotProduct}(Q,K,V) &= \text{softmax}(QK^T/\sqrt{d_k})V \end{aligned}$$