B. Data Structures, Symmetries, and Neural Network Architectures

B.1 Tabular data: MLPs, residual connections, and autoencoders
B.2 Grid-structured data: CNNs
B.3 Sequential data: RNNs, LSTMs, and GRUs
B.4 Graph-structured data: GNNs and message passing
B.5 Set-structured data: Deep Sets

Tiam Heydari, PhD
Postdoctoral Fellow, UBC

Sets: Basics

- Assume we have a set **X**=$\{x_1, \quad x_N\}$, is made out of N elements.
- Each element can exist in R^d dimensions.
- Now our task is to make predictions about the “set”. Note that our predictions are about the set and not individual elements.
- Because of this, any sort of prediction task (f) that we do over the full set **X**, should be permutation invariant. Why? Because the order of elements does not matter in sets.
- We have already learned about permeation equivariance and invariances.

f(**X**)= f(P**X**), where P is permutation matrix

Example1: F($\{x_1, x_2, x_3\}$) = F($\{x_3, x_2, x_3\}$)

Example2: This point cloud is referring to number (3), regardless of in which orde we store the points coordinates

DeepSets: Basics

- Deep set theorem suggests that the most powerful function (that will be able to perform permutation invariance) over the set **X**=$\{x_1, \quad x_N\}$ can be always written as bellow:

$$f(X) = \rho\left(\sum_i \phi(x_i)\right)$$

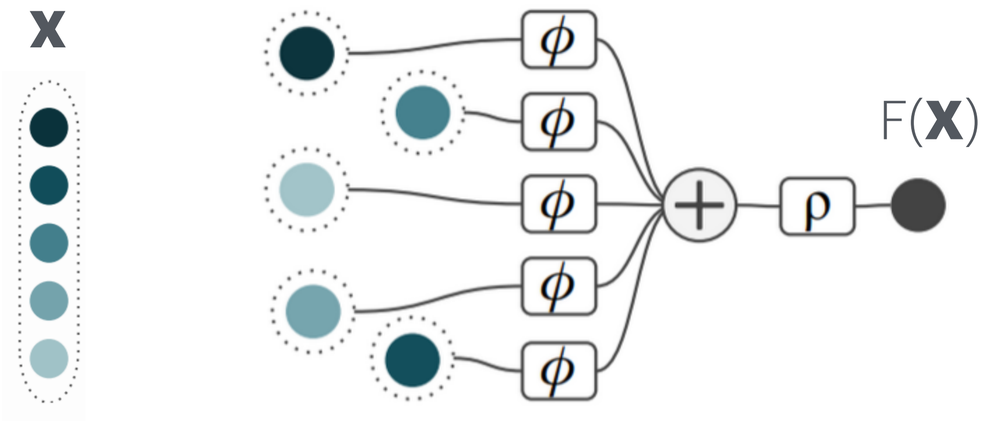
- Here ϕ is a function that encodes individual elements (x_i), then the outputs of all elemntwise operations $\varphi(x_i)$ will be summed.
- Note that this term $\left(\sum_i \phi(x_i)\right)$ is permutation invariant because of the sum.
- ρ is an arbitrary function that transforms the pooled sum into the final result.

- We can imagine that we can just replace the ϕ and ρ with simple MLPs and make this a deep learning model.

Deep set paper: https://arxiv.org/abs/1611.08998

DeepSets: Basics

- We can visually illustrate the computation as:



- **However**! This will not work the best in practice. Why?
  - There are two levels when we think about a set, individual elements and their relationship with each other (relational reasoning).
  - In the deepest formula, individual reasoning is being performed by ϕ and relational reasoning by ρ. This is because only ρ can see all elements at the same time together
  - However, this in practice usually does not work the best, and people found that is better to rewrite the f(**X**) in a slightly different way to work better in practice.

From https://fabianfuchsml.github.io/learningonsets/

DeepSets: Basics

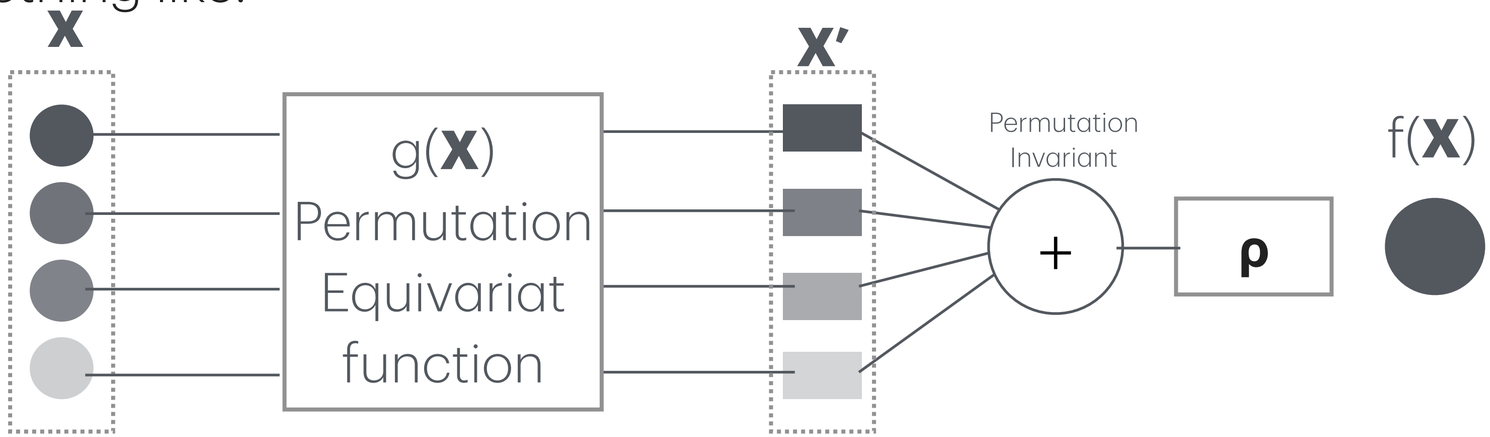
- An example of importance of relational reasoning in deep sets:

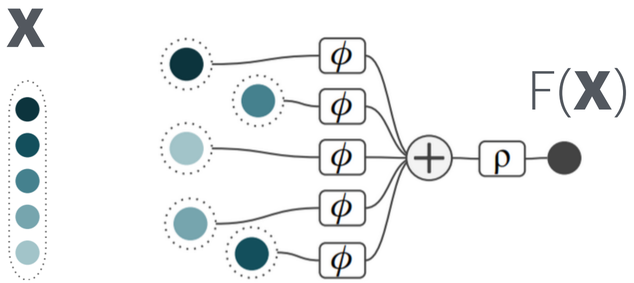
> “To illustrate this with a simple example, consider the task of assessing how well a set of ingredients go together for cooking a meal. In the original form of deepset, the function ϕ can take into account relevant individual attributes, but will be unable to spot any clashes between ingredients (like garlic and vanilla).
>
> However note that the formula these interactions into account implicitly. This is because the ρ gets the encoded set as its input, sees the whole set at once. So, in principle, it’s capable of taking into accout interactions between any number of elements, and could learn these implicitly. However, the version of the set that it sees is a learned encoding, so it’s **harder** to design the model so that it makes use of these interactions properly.

Adapted From https://fabianfuchsml.github.io/learningonsets/

DeepSets:

- What would be an alternative function form that we can put there to work better in practice?
- Turns out that following suggestion would work better:
  "First having multiple permutation equivaraint transformation functions. Then an invariant function"
- Graphically it is something like:





- Comparing the new computation graph with the old one, you see that before pooling (+), we had a chance to create embeddings **X'=** g(**X**) for individual elements
- g(**X**), knows about all elements when makes the embedding
- The only catch is that it should be permutation equivariant

Deep set paper: https://arxiv.org/abs/1611.08998

DeepSets:

- Now the question is, what should the g(**X**) be based on the constraints we set?
- If g(**X**) is a non-linear NN, it should be combination of linear (W) and non linear (V) transformations.
- Let's look at a linear maps on a set of N elements that is permutation equivariant.

$$X = (x_1, \ldots, x_N), \quad x_i \in \mathbb{R}^d$$

- A linear layer acting on the set can be written as $Y = WX$
- Permutation equivariance requires $W(PX) = P(WX)$
- The set of matrices commuting with the symmetric group S_n (symmetric group on n n elements) is:

$$Y = WX = \gamma X + \lambda(\mathbf{1}\mathbf{1}^T)X$$

Where $Y = y_1, y_2, \ldots, y_n$. Note that:

Note that:

$$(\mathbf{1}\mathbf{1}^T)X = \begin{pmatrix} \sum_{j=1}^N x_j \\ \sum_{j=1}^N x_j \\ \vdots \\ \sum_{j=1}^N x_j \end{pmatrix}$$

Therefore:

$$y_i = \gamma x_i + \lambda \sum_{j=1}^N x_j$$

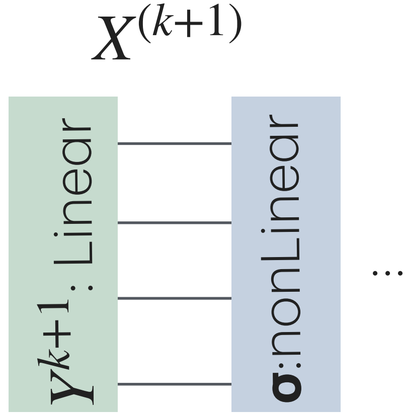
Or, recentered: (this works better, sees interaction better- elements talk to each other via the subtracted from mean pool)

$$y_i = \gamma x_i + \lambda(x_i - \bar{x})$$

$$\bar{x} = \frac{1}{N} \sum_{j=1}^N x_j$$

DeepSets: A basic full model

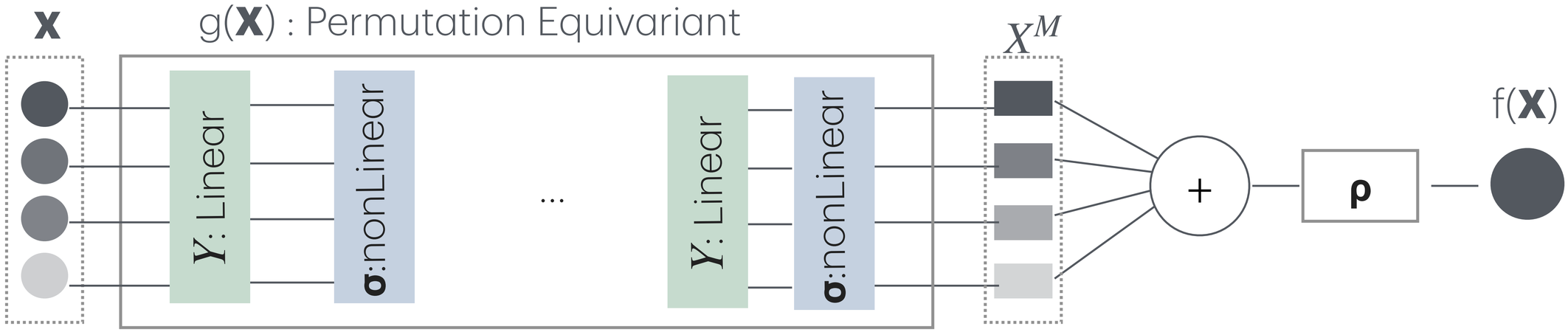
We can this of this (**Y**) as a linear block that we use inside g(**X**)
We will then add a non linear layer (**σ**) right after and make a specific for of MLP that is Permutation Equivariant, that will be our g(X)!



$$X^{(k+1)} = \sigma\Big(Y^{(k+1)} \Big) = \sigma\Big(\Gamma^{(k)} X^{(k)} + \Lambda^{(k)} \big(X^{(k)} - \bar{X}^{(k)} \big) \Big)$$

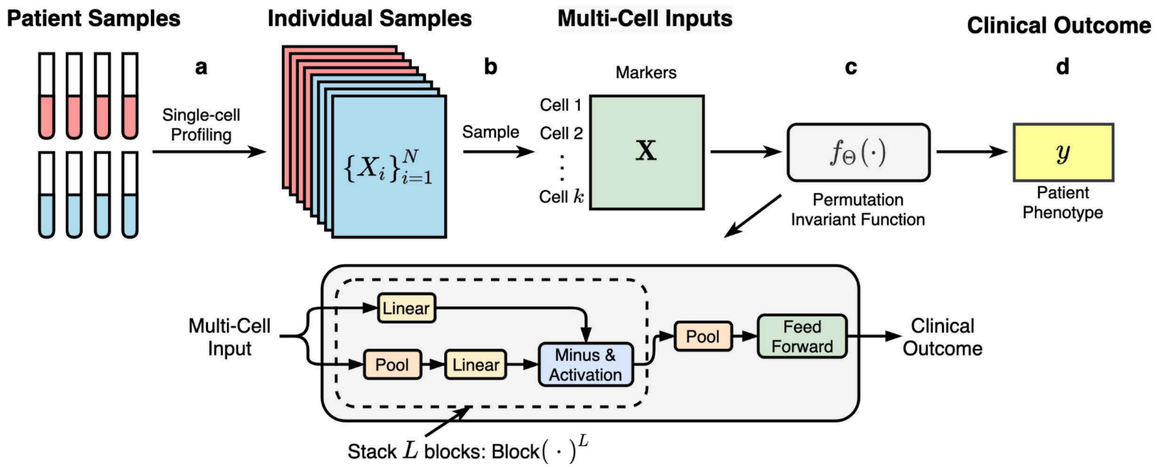
So the final form of f(X) will be (After M layers of deepest stacked layers):

$$f(X) = \rho \left(\frac{1}{N} \sum_i g(X)_i \right)$$



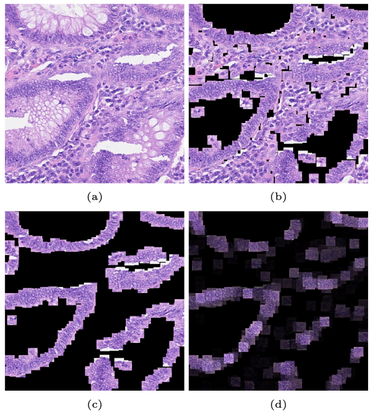
DeepSets: Example applications in biological systems:

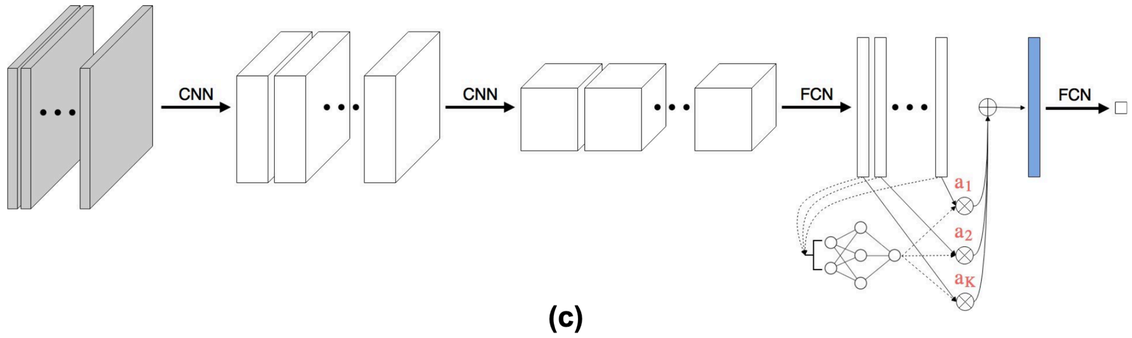
1.Predict patient-level clinical outcome from a cytometry sample containing many single cells.



Yi, Haidong, and Natalie Stanley. "Cytoset: Predicting clinical outcomes via set-modeling of cytometry data." *Proceedings of the 12th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. 2021.

2.Problem: Predict a slide-level diagnosis from many image patches, even though only the whole slide has a label.





Ilse, Maximilian, Jakub Tomczak, and Max Welling. "Attention-based deep multiple instance learning." *International conference on machine learning*. PMLR, 2018.

Recipe Card: **Deep Set Model**

Type of Learning: **Supervised**

Data set: **Sets**

Data

1- Load libraries: Basics + PyTorch

2-Load your Set Dataset **D**:

-Make sure **D** is a list and each element of **D** (**D**[i]=X_i) is a set

-So: $X_i = x_1, \ldots x_n$, where x_i is a vector

-If you have a custom Dataset class, the two core things are required for torch DataLoader to be able to process them:

a)the class should have __len__ (How many samples exist)

b)the class should have __getitem__ (to get one sample by index)

3-Now set up the DataLoader for training/validation/testing:

-if the size of sets is not uniform, make a costume costum_batch. costum_batch will downsample or add zero padding if needed

-use torch's random_split function to split data to three parts

- A usual split is (train_data: 0.6, validate_data: 0.2, test_data 0.2)

- Now load these splitter data pieces to torch loader via torch DataLoader use the costum_batch function if needed!

NN model

4-Building the DeepSet neural network network

-Define the DeepSet linear layer $\Gamma^{(k)}X^{(k)} + \Lambda^{(k)}\left(X^{(k)} - \bar{X}^{(k)}\right)$ as a Module. Adding batch correction at the end of each linear layer is good.

-Define DeepSet network as a Module. It has the form:

Input, X_i**→[(DeepSetLinear)→(Relu)]→ … →[(DeepSetLinear)→(Relu)] →[+Pool]→ out,** $\hat{Y}_i$

-note that dimension of out, depends on the dimension of the supervised task (label). The goal of the whole network is to predict output $\hat{Y}_i$ for each set, X_i (with real value Y_i)

-If Y_i is a continuous variable (regression) then the last layer a simple linear layer with [dim]=[dim] of Y_i

-f Y_i is categorical, then the last layer should be output raw logits with dim = Number of categories in Y_i

Training

5-Define a proper loss function based on Y_i

-If Y_i is continuous, then use MSELoss or L1Loss

-If Y_i is categorical, use CrossEntropyLoss.
Note that we feed raw logits directly to Loss.

-Add regularization to prevent overfitting

-The loss will be the sum of all relevant losses

6-Define the train and test functions:

-train:

a) the model will be trained in this block.

b) it gets model, DataLoader, and optimizer(Adam)

b) set model.train() at the beginning

c) for loop over all batcher:

i-Send data to device

ii-optimizer.zero_grad()

iii-Calculate loss

iv-Back propagate via loss.backward() to find derivatives then optimizer.step() to update

d) at the end, test function return loss

-test: same as train but, set model.test() and backer

7-Define training and run the training:

-Choose number of epochs, For each epoch:

a) run train() for one epoch on the training loader

b)run test() on thetraining/validation loader to get losses

-At the end:a) load the best saved model plot training and validation loss curves if needed

C. Attention and Transformers

C.1 Origins of attention

C.2 Attention as a standalone operator

C.3 Transformer architectures

C.4 Transformers across different data structures

C.5 Training transformers