MACHINE LEARNING IN PHYSICS
GRAPH NEURAL NETWORKS

HARRISON B. PROSPER

PHY6938

Recap: Normalizing Flows

Let $y = f(x)$ be *vector*-valued with $x, y \in \mathbb{R}^D$. The change of variables formula is

$$p(x) = q(y)\,|\det J|$$

or

$$\log p(x) = \log q(f(x)) + \log|\det J|$$

where $J = \nabla_x f(x)$ is the Jacobian matrix of the transformation.

A normalizing flow is a sequence of bijections modeled as neural networks that approximates $\mathrm{y} = f(x)$, and its inverse $x = f^{-1}(y)$.

Recap: Diffusion Model

A diffusion model maps a vector x_1 to x_0. When x_1 is sampled from a D-dimensional Gaussian this induces a sampling of x_0 from the desired distribution, $p(x_0)$. One class of diffusion model entails solving the integro-differential equation,

$$\frac{dx_t}{dt} = \lambda_t x_t + \mu_t q(t, x_t)$$

from $t = 1$ to $t = 0$, where $\lambda_t \equiv \lambda(t)$ and $\mu_t \equiv \mu(t)$ are known functions of "time" and

$$q(t, x_t) = \int_{\mathbb{R}^D} x_0 \, \frac{p(x_t \,|x_0)}{p(x_t)} \boldsymbol{p(x_0) dx_0}$$

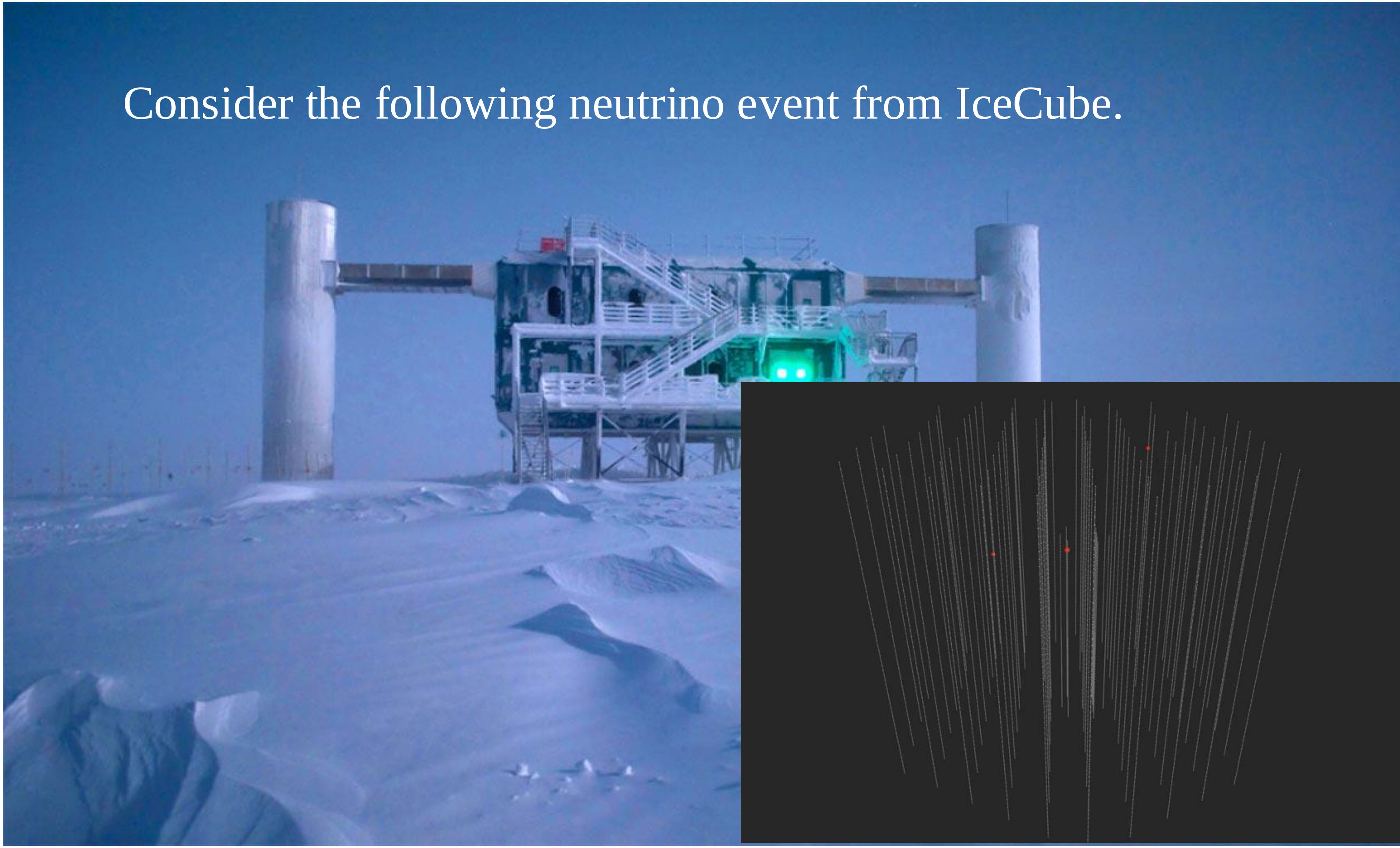
where $p(x_t \mid x_0)$ is a D-dimensional Gaussian.

Introduction

1. Convolutional neural networks (CNN)
2. Autoencoder (AE)
3. Physics-informed neural networks (PINN)
4. Flow and diffusion models
5. Graph neural networks (GNN)
6. Transformer neural networks (TNN)

Introduction

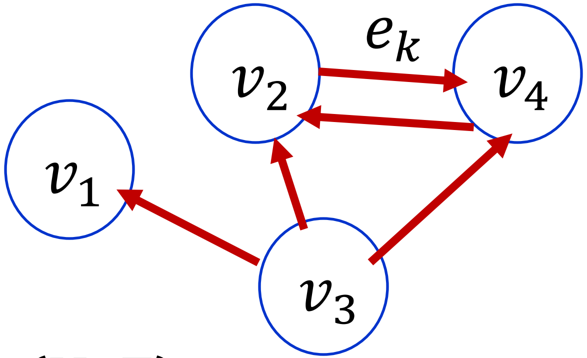
Consider the following neutrino event from IceCube.



Introduction

- The data for a single neutrino event can be modeled as a point cloud, that is, a cloud of n points in a d-dimensional space.

- Another way to represent a point cloud is as a graph.



- A graph $G = (V, E)$ comprises vertices with attributes $V = \{v_i\}$ and edges with attributes $E = \{e_k\}$.

Introduction

- Standard neural networks expect the input data to be of fixed size.

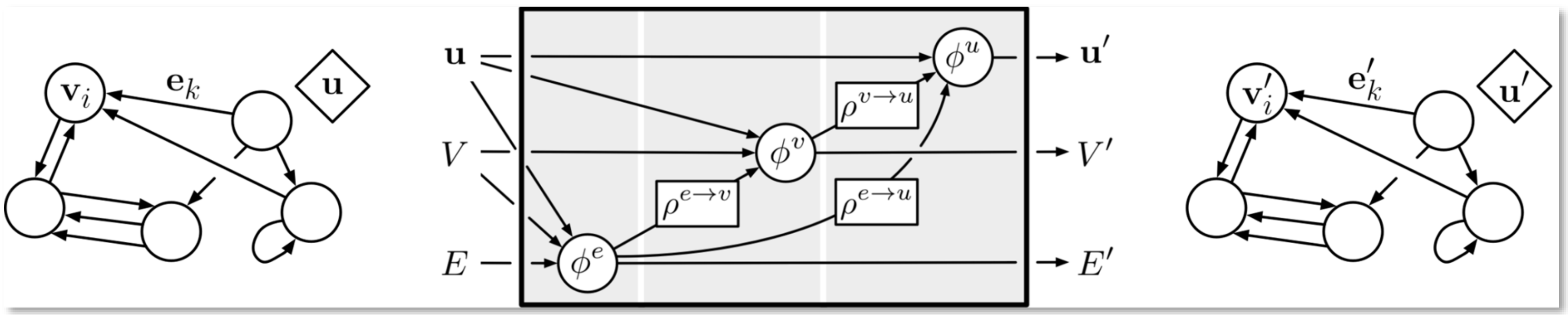
- However, the number of points in a point cloud or the number of vertices in a graph can change from one data instance to the next.

- Using the work of the IceCube collaboration, we'll show how a graph neural network can handle graphs having a varying number of vertices.

GRAPH NEURAL NETWORKS

Graph Neural Networks (GNN)

A graph neural network is designed to operate on data organized as a graph:



A graph enters a graph processor on the left and exits the processor on the right with altered graph attributes.

https://github.com/deepmind/graph_nets

IceCube: Data

The data collected by the IceCube detector consists of signals from photomultipliers*.

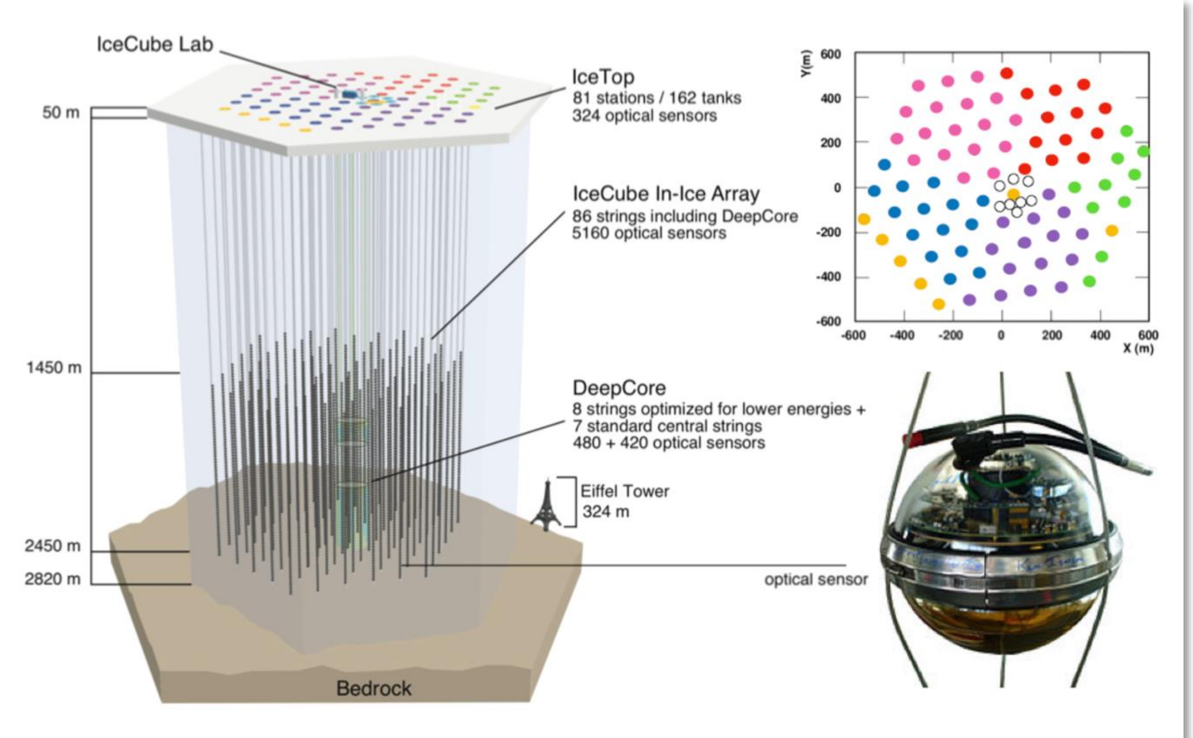


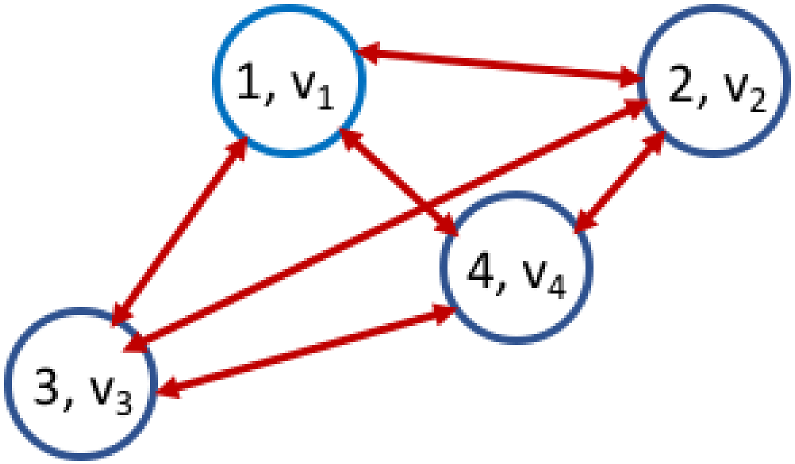
Fig. 1. The IceCube Neutrino Observatory with the in-ice array, its sub-

*N. Choma et al. IceCube collaboration, Graph Neural Networks for IceCube Signal Classification, arXiv:1809.06166v1

IceCube: Data

- IceCube data are the signals from n Digital Optical Modules (DOMs), which are modeled as the vertices of a graph.

- Each vertex is associated with a d-dimensional (row-wise) vector of attributes $v_i = (x_1, \cdots, x_d)_i$, three of which are the spatial coordinates (x, y, z) of the DOM.



IceCube: Data

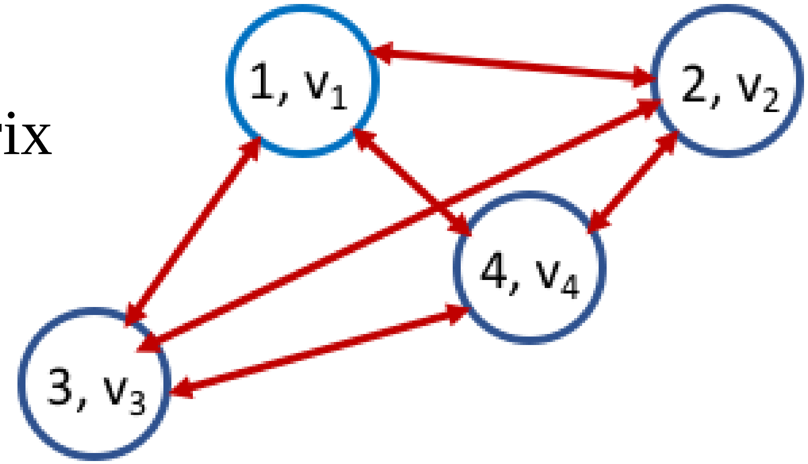
For an event with n signals, an $n \times n$ adjacency matrix, $\boldsymbol{A}$, is constructed, as follows, using the spatial data only,

$$(\boldsymbol{A})_{\boldsymbol{ij}} = \text{softmax}(d_{ij},\ \text{dim} = 1),$$

where

$$d_{ij} = \exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)$$

and σ is a free parameter. The matrix $\boldsymbol{A}$ models the edges of the graph.

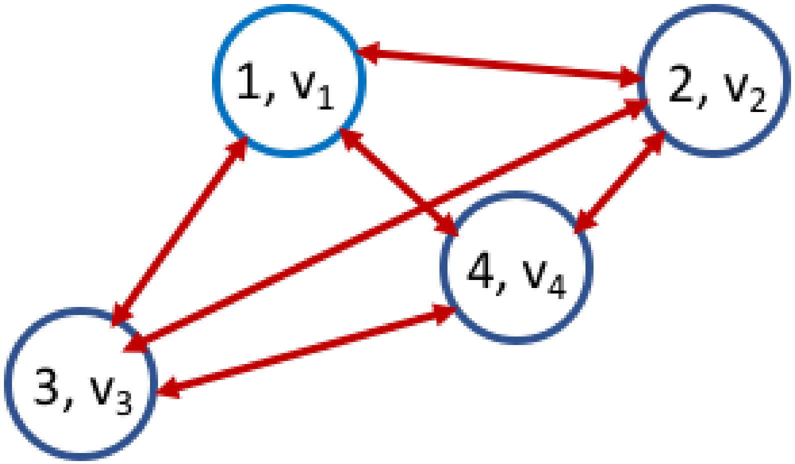


IceCube: Data

The vectors $\boldsymbol{v_i}$ are concatenated *vertically* into an $n \times d$ matrix: $\boldsymbol{X} = [v_1; \cdots; v_n]$

$$X = \begin{pmatrix} x_1 & y_1 & z_1 & \cdots \\ x_2 & y_2 & z_2 & \cdots \\ \vdots & \vdots & \vdots & \\ x_n & y_n & z_n & \cdots \end{pmatrix}$$

(Note: horizontal concatenation is denoted by $\boldsymbol{X} = [v_1, \cdots, v_n]$.)



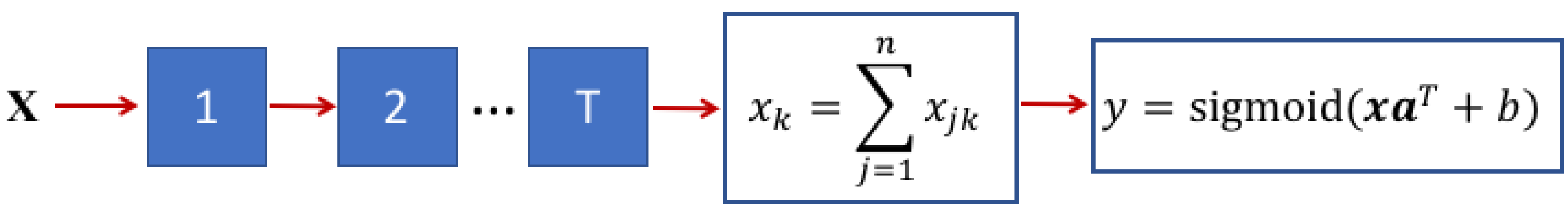
IceCube: GNN

- The $n \times d$ matrix, $\boldsymbol{X}$, passes through a sequence of T identical graph processors, each with its own parameters.

- A graph processor is parameterized by a $2d \times d/2$ matrix of parameters, w, and a single scaler b and computes:

$$\boldsymbol{Y} = [\boldsymbol{AX}, \boldsymbol{X}]w + b\boldsymbol{I}$$

- The $\boldsymbol{Y}$ operation is called a graph convolution (**GConv**) and $\boldsymbol{I}$ is an $n \times d/2$ matrix of ones.

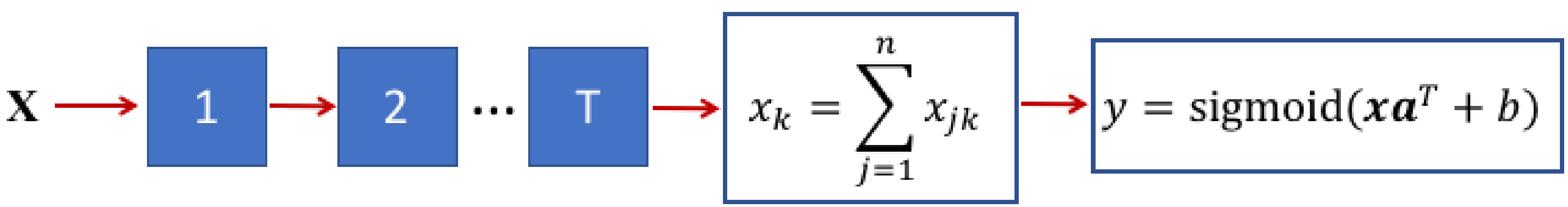


IceCube: GNN

➢ The vertices, $\boldsymbol{X}$, of the output graph $G = (\boldsymbol{X}, \boldsymbol{A})$ are constructed through *horizontal* concatenation:

$$\boldsymbol{X} = [\mathrm{ReLU}(\boldsymbol{Y}), \boldsymbol{Y}]$$

with, as usual, the ReLU function applied *element-wise*.

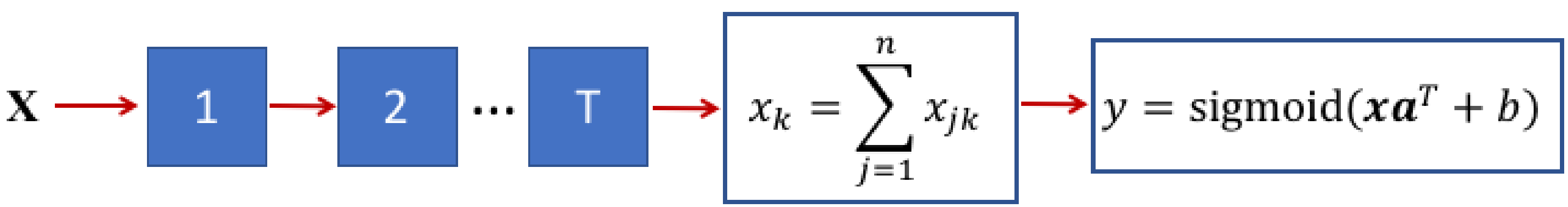


IceCube: GConv

$$\boldsymbol{Y} = [\boldsymbol{AX}, \boldsymbol{X}]w + bI,$$
$$\boldsymbol{X} = [\mathrm{ReLU}(\boldsymbol{Y}), \boldsymbol{Y}]$$

Implementation

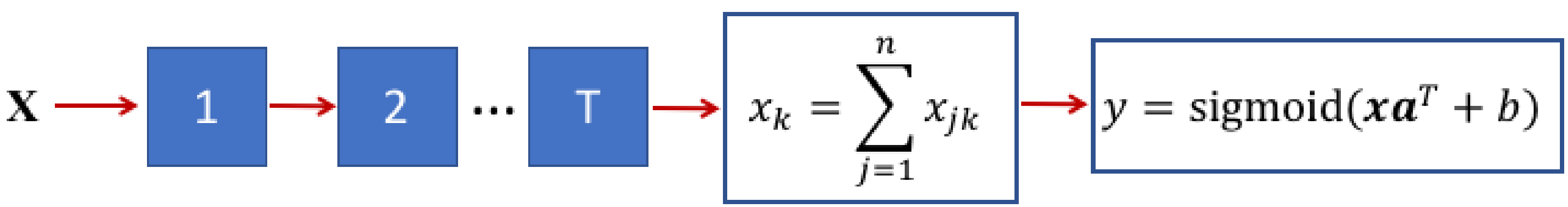
```
AX = torch.matmul(A, X)
AXX = torch.cat([AX, X], dim=1)
AXXw = torch.matmul(AXX, w)
Y = AXXw + b * torch.ones_like(AXXw)
X = torch.cat([F.relu(Y), Y], dim=1)
```



IceCube: GNN

- At the end, $\boldsymbol{X}$ is mapped to a d-dimensional vector $\boldsymbol{x}$ using a *permutation invariant map* with respect to the vertices and *invariant* with respect to the number of vertices.

- Finally, the vector $\boldsymbol{x}$ is mapped to the scalar output $0 \leq y \leq 1$ using a sigmoid.



IceCube: GNN Results

The GNN is **6.3** times more efficient than the IceCube physics baseline analysis at a signal to noise ratio that is **3** times better.

It also outperforms a 3D CNN.

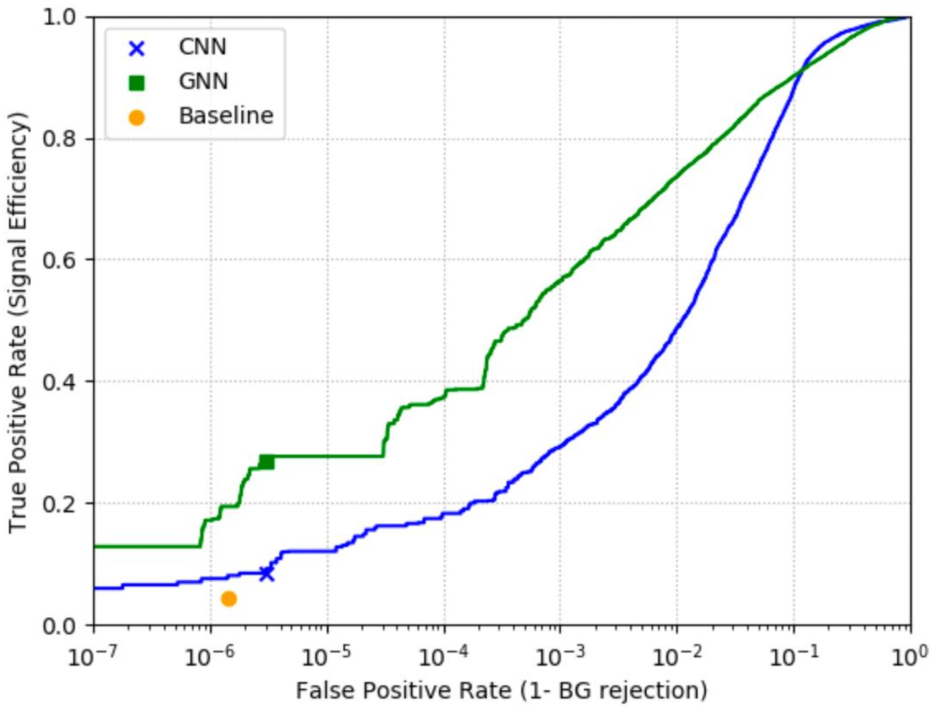


Fig. 3. Receiver operating characteristic curve for various methods considered in this paper. The green square and blue X indicate the evaluation point for the GNN and CNN, respectively.

Summary

- The models we have discussed so far are ill-suited to handle variable input data.

- But data instances in which the number of points, or vertices, can vary can be modeled with graphs $G = (V, E)$ using graph neural networks (GNN).

- We looked at a specific application of GNNs by IceCube, which, for this experiment, achieved state-of-the-art classification results.