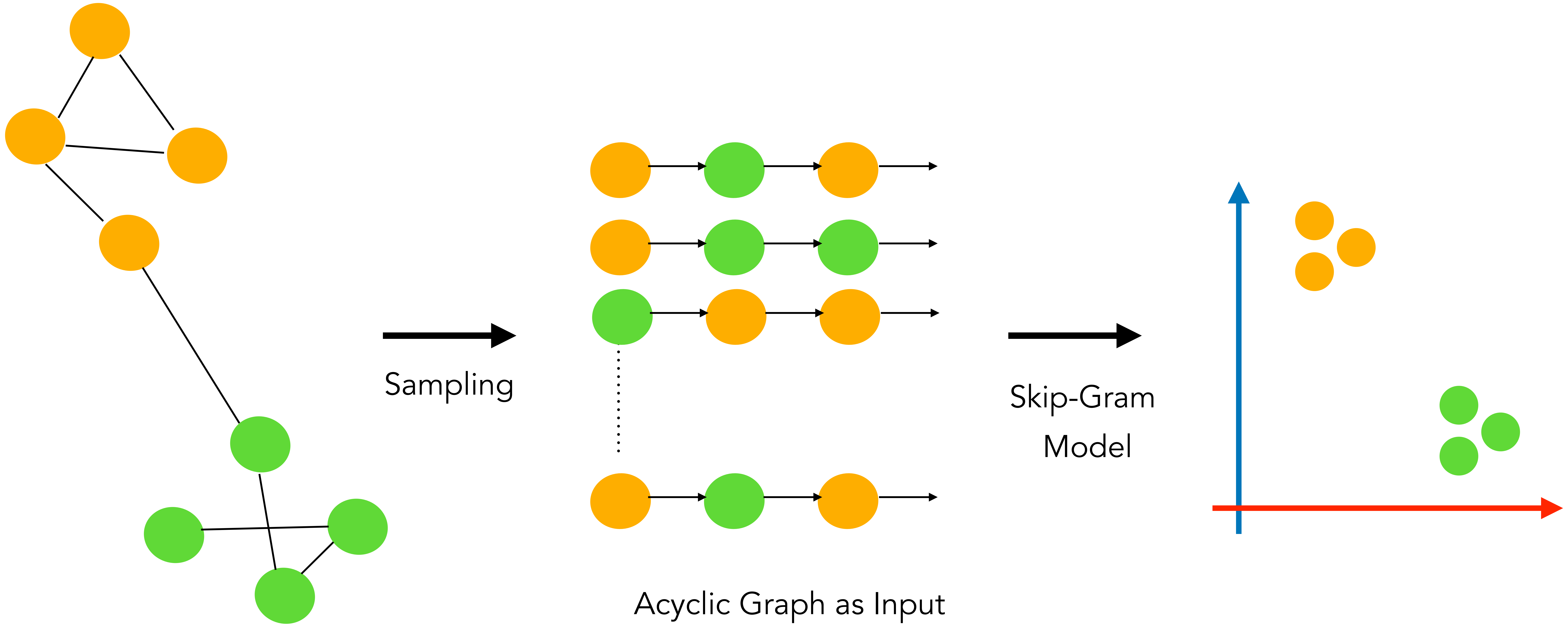
Node2Vec

B. N. Srimouli
Mtech - DSE
BITS Pilani



$$F : G(V, E, W) \longrightarrow R^N$$

Mathematical view:

Feature Learning

For every source node $u \in V$, we define $N_S(u) \subset V$ as a *network neighbourhood* of node u generated through a neighbourhood sampling strategy S.

Optimisation of the following objective function, which maximises the log-probability of observing a network neighbourhood $N_S(u)$ for a node u conditioned on its feature representation, given by f:

$$\max_f \sum_{u \in V} \log Pr(N_S(u)|f(u))$$

Why do we use Log Space?

Avoid Underflow

Adding is faster than multiplying

Assumptions:

To make optimization problem traceable

- Conditional Independence

applying markov assumption

$$Pr(N_S(u) \mid f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i \mid f(u))$$

we are approximating every component in the product. We are factorizing the likelihood by assuming the likelihood of observing neighbourhood node.

Assumptions:

To make optimization problem traceable

- Symmetry in feature Space

we are modeling conditional likelihood of every source neighbourhood node pair as softmax unit parametirised by dot product of features

$$Pr(n_i | f(u)) = \frac{e^{f(n_i) \cdot f(u)}}{\sum_{v \in V} e^{f(v) \cdot f(v)}}$$

Why Softmax?

Hierarchical Softmax (HS) is a solution to the enormous compute cost of training a neural network which uses the **Softmax** activation and has thousands of outputs. This allows us to decompose calculating the probability of one word into a sequence of probability calculations, which saves us from having to calculate the expensive normalization over all words

In case of Node2Vec, we have one output for a vertex in our corpus and we can have any 100s of thousands of outputs.

$$p(n_i \mid f(u)) = \frac{\exp(f(n_i) \, . f(u))}{\sum_{v \in V} \exp(f(v) \, . f(u))}$$

Using above assumptions, we get.

$$\max_{f} \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

$$Z_u = \sum_{v \in V} \exp(f(u) \cdot f(v))$$

Using this softmax layer, the model tries to maximize the probability of predicting the correct word at every timestep. The whole model thus tries to maximize the averaged log probability of the whole corpus:

$$J_\theta = \frac{1}{T} \log p(w_1, \cdots, w_T)$$

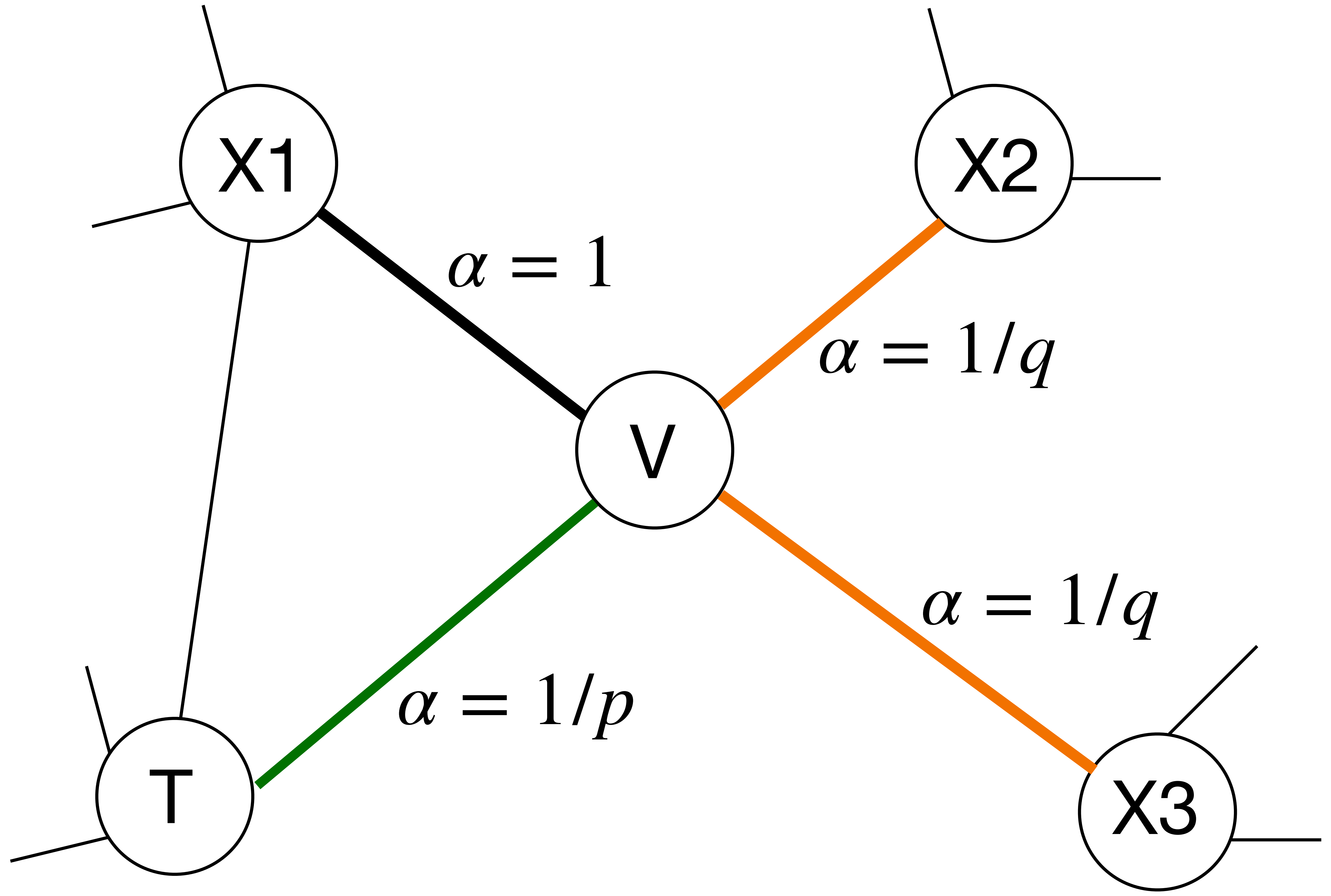
Analogously, through application of the chain rule, it is usually trained to maximize the average of the log probabilities of all vertexes in the corpus given their previous

Sampling Strategy

- Number of Walks
- Walk Length
- P : Return Hyper parameter
- Q : In-out Hyper parameter

Random Walk:

A graph-theoretic measure for ranking nodes as well as similarity: for example, two entities are similar, if lots of short paths between them. Random walks have proven to be a simple, but powerful mathematical tool for extracting this information



$\alpha = SearchBias$

$$P(C_i = x \mid C_{i-1} = v) = \begin{cases} \pi_{vx}/Z & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

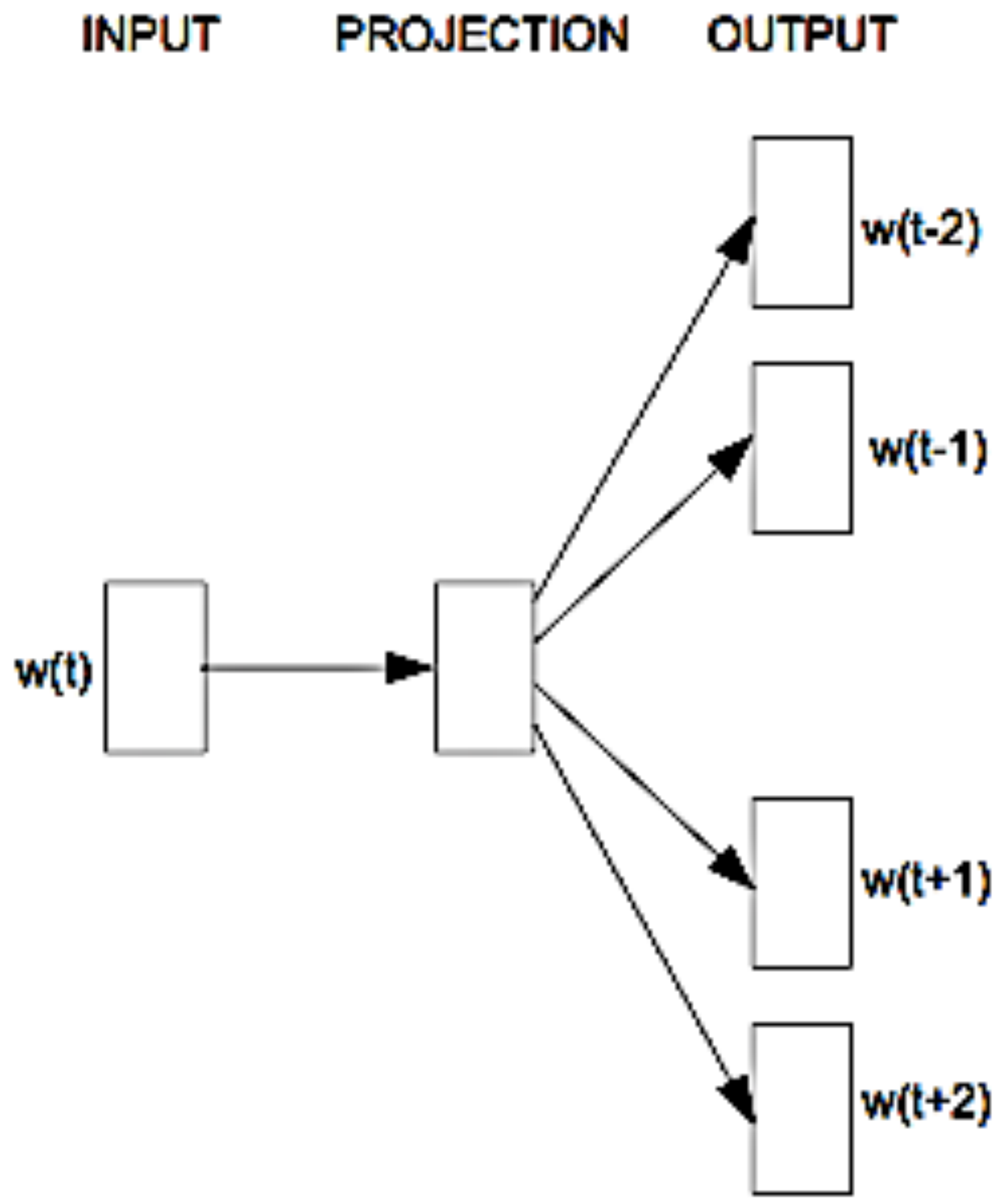
$\pi_{vx} \rightarrow$ Unnormalized transitional probability

$Z \rightarrow$ Normalizing constant

$\pi_{vx} =$ Its a vector we can pre compute these values, this saves time of execution

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$$

$$\alpha_{pq}(t,x) = \begin{cases} 1/p & d_{tx} = 0 \\ 1/q & d_{tx} = 2 \\ 1 & d_{tx} = 1 \end{cases}$$



Skip Gram Model

Negative Sampling

SGD with Negative Sampling

- From the entire training corpus of list of words, and we chose about 5 negative samples by picking randomly from the list.

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} \left(f(w_j)\right)}$$

- These words are intended to make the output = 0, the most frequent words are likely to be selected as negative samples.Negative sampling also uses a logistic loss function to minimise the negative log-likelihood of words in the training set.

- In order to mitigate this computational burden, of iterations needed to calculate the many calculations for update of one single weight in word2vec the Stochastic Gradient Descent (SGD) was used for parameter optimization in designing node2vec

$$\theta^{(new)} = \theta^{(old)} - \eta \cdot \nabla_{J(\theta;w^{(t)})}$$

Node2vec combines the benefit of both Breadth first Search which has edge in assessing neighbouring nodes and the benefits fo Depth for search to asses the distant nodes making it a hybrid algorithm to explore the graph