A Additional experimental results and implementation details

Datasets The statistics for the largest connected components of the experimental datasets are given in Table 1.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Classes</th>
<th>Features</th>
<th>Nodes</th>
<th>Edges</th>
<th>Label rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>citation</td>
<td>7</td>
<td>1433</td>
<td>2485</td>
<td>5069</td>
<td>0.056</td>
</tr>
<tr>
<td>Citeseer</td>
<td>citation</td>
<td>6</td>
<td>3703</td>
<td>2120</td>
<td>3679</td>
<td>0.057</td>
</tr>
<tr>
<td>PubMed</td>
<td>citation</td>
<td>3</td>
<td>500</td>
<td>19717</td>
<td>44324</td>
<td>0.003</td>
</tr>
<tr>
<td>Coauthor CS</td>
<td>co-author</td>
<td>15</td>
<td>6805</td>
<td>18333</td>
<td>81894</td>
<td>0.016</td>
</tr>
<tr>
<td>Computers</td>
<td>co-purchase</td>
<td>10</td>
<td>767</td>
<td>13381</td>
<td>245778</td>
<td>0.015</td>
</tr>
<tr>
<td>Photos</td>
<td>co-purchase</td>
<td>8</td>
<td>745</td>
<td>7487</td>
<td>119043</td>
<td>0.021</td>
</tr>
<tr>
<td>OGB-Arxiv</td>
<td>citation</td>
<td>40</td>
<td>128</td>
<td>169343</td>
<td>1166243</td>
<td>0.005</td>
</tr>
</tbody>
</table>

Table 1: Dataset Statistics

Replication of results and hyper-parameters Code to regenerate our experimental results together with hyperparameters for all datasets is provided. The hyperparameters are listed in best_params.py. Refer to the README.md for instructions to run the experiments.

Numerical ODE solver We use the library torchdiffeq [2] to discretise the continuous time evolution and learn the system dynamics. The Pontryagin maximum / adjoint method is used to replace backpropagation for all datasets, with the exception of Cora and Citeseer due to the high memory complexity of applying backpropagation directly through the computational graph of the numerical integrator.

Decoupling the terminal integration time between inference and training At training time we use a fixed terminal time $T$ that is tuned as a hyperparameter. For inference, we treat $T$ as a flexible parameter to which we apply a patience and measure the validation performance throughout the integration.

Data splits We report transductive node classification results in Tables 2 and 3 in the main paper. To facilitate direct comparison with many previous works, Table 2 uses the Planetoid splits given by [8]. As discussed in e.g. [7], there are many limitations with results based on these fixed dataset splits and so we report a more robust set of experimental results in Table 3. Here we use 100 random splits with 20 random weight initializations. In each case 20 labelled nodes per class are used at training time, with the remaining labels split between test and validation. This methodology is consistent with [7].
We additionally conducted an ablation study using these functions, results are given in Table 2.

<table>
<thead>
<tr>
<th>Method</th>
<th>Cora</th>
<th>Citeseer</th>
<th>PubMed</th>
<th>CoauthorCS</th>
<th>Computers</th>
<th>Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>cosine_sim</td>
<td>83.8 ± 0.5</td>
<td>73.5 ± 1.1</td>
<td>79.0 ± 2.3</td>
<td>92.8 ± 0.1</td>
<td>84.8 ± 0.7</td>
<td>93.2 ± 0.4</td>
</tr>
<tr>
<td>exp_kernel</td>
<td>83.6 ± 1.8</td>
<td>75.0 ± 1.3</td>
<td>79.9 ± 1.4</td>
<td>92.8 ± 0.2</td>
<td>84.8 ± 0.5</td>
<td>93.2 ± 0.5</td>
</tr>
<tr>
<td>pearson</td>
<td>83.7 ± 1.5</td>
<td>75.2 ± 1.3</td>
<td>80.0 ± 1.2</td>
<td>92.8 ± 0.2</td>
<td>84.8 ± 0.5</td>
<td>93.5 ± 0.3</td>
</tr>
<tr>
<td>scaled dot</td>
<td>84.8 ± 0.9</td>
<td>75.9 ± 1.3</td>
<td>79.5 ± 1.4</td>
<td>92.9 ± 0.2</td>
<td>86.9 ± 0.6</td>
<td>92.9 ± 0.6</td>
</tr>
</tbody>
</table>

Table 2: Performance with different attention functions. Scaled dot is best performing in four of the six experiments.

**Positional encodings** In section 5.2 of the main paper three types of positional encoding are described: **DIGL PPR** [4], **DeepWalk** [6] and **hyperbolic embeddings** [1][5]. We provide the latter two as preprocessed pickle files within our repo. The DIGL PPR positional encodings are too large and so these are automatically generated and saved whenever code that requires them is run.

**Diffusivity (attention) function** In section 4 of the main paper one choice of attention function is described. Additionally four alternatives were considered, which achieved roughly equivalent performance:

- Scaled dot
  \[
  a(z_i, z_j) = \text{softmax} \left( \frac{(W_K z_i)^	op W_Q z_j}{d_k} \right)
  \]
  (1)

- Cosine similarity
  \[
  a(z_i, z_j) = \text{softmax} \left( \frac{(W_K z_i)^	op W_Q z_j}{\|W_K z_i\| \|W_Q z_j\|} \right)
  \]
  (2)

- Pearson correlation
  \[
  a(z_i, z_j) = \text{softmax} \left( \frac{(W_K z_i - W_K z_i)^	op (W_Q z_j - W_Q z_j)}{\|W_K z_i - W_K z_i\| \|W_Q z_j - W_Q z_j\|} \right)
  \]
  (3)

- Exponential kernel
  \[
  a(z_i, z_j) = \text{softmax} \left( (\sigma_u \sigma_x)^2 e^{-\|W_K u_i - W_Q u_j\|^2/2d_k^2} e^{-\|W_K x_i - W_Q x_j\|^2/2d_k^2} \right)
  \]
  (4)

We additionally conducted an ablation study using these functions, results are given in Table 2.

**B Theory of Beltrami flow**

This section proves Theorem 1 in the main paper:

**Theorem 1.** Under structural assumptions on the diffusivity, graph Beltrami flow

\[
\frac{\partial z_i(t)}{\partial t} = \sum_{j : (i, j) \in E} a(z_i(t), z_j(t))(z_j(t) - z_i(t)) \quad z_i(0) = z_i; \quad i = 1, \ldots, n; \quad t \geq 0
\]

is the gradient flow of the discrete Polyakov functional.

**B.1 Polyakov action and Beltrami flow on manifolds**

In this section we briefly review the analysis in [3] to motivate the introduction of a discrete Polyakov action on graphs. Assume that \((\Sigma, g)\) and \((M, h)\) are Riemannian manifolds with coordinates \(\{x_\mu\}\) and \(\{y_\ell\}\) respectively. The Polyakov action for an embedding \(Z : (\Sigma, g) \rightarrow (M, h)\) can be written as

\[
S[Z, g, h] = \sum_{\mu, \nu = 1}^{\dim(\Sigma)} \sum_{\ell, m = 1}^{\dim(M)} \int_{\Sigma} h_{\ell m}(Z) \partial_\mu Z^\ell \partial_\nu Z^m g^{\mu \nu} \text{dvol}(g),
\]
We consider the graph counterpart to the problem of minimizing a generalized Dirichlet energy $Z$ with respect to the embedding. Let $G$ where the operator $\Delta$ is the graph Laplacian. Therefore, from a high-level perspective, the Beltrami flow derived from the Polyakov action consists of a family of nonnegative maps $(\psi_{ij}^\ell)$, with $\psi_{ij}^\ell : \mathbb{R}^{(d' + d)} \rightarrow \mathbb{R}$, $1 \leq i, j \leq n$ and $1 \leq \ell \leq d' + d$, we define

$$S[Z, \psi] := \frac{1}{2} \sum_{\ell=1}^{d' + d} \sum_{i,j=1}^{n} A_{ij} \psi_{ij}^\ell (Z),$$

with $\text{dvol}(g)$ the volume form on $\Sigma$ associated with the metric $g$. We restrict to the case where $(M, h)$ is the $d$-dimensional Euclidean space $\mathbb{R}^d$, so that the functional becomes

$$S[Z, g] = \sum_{\ell=1}^{d} \sum_{i,j=1}^{n} \int_{\Sigma} \partial_\mu Z^\ell \partial_\nu Z^\ell g^{\mu\nu} \text{dvol}(g).$$

The quantity above can be rewritten more geometrically as

$$S[Z, g] = \sum_{\ell=1}^{d} \int_{\Sigma} ||\nabla_g Z^\ell||^2 \text{dvol}(g). \quad (6)$$

From equation (6) we see that the Polyakov action is measuring the smoothness of the embedding - more precisely its Dirichlet energy measuring the smoothness of a graph embedding - defined by varying both $Z$ and $g$ we find that the metric $g$ must be the metric induced on $\Sigma$ by the map $Z$, namely its pullback. In local coordinates, this amounts to the constraint below:

$$g_{\mu\nu} = \sum_{\ell=1}^{d} \partial_\mu Z^\ell \partial_\nu Z^\ell. \quad (7)$$

Once the condition on $g$ is satisfied, the Euler-Lagrange equations for each feature channel become

$$\sum_{\mu,\nu} \frac{1}{\sqrt{\text{det}(g)}} \partial_\nu (\sqrt{\text{det}(g)} g^{\mu\nu} \partial_\mu Z^\ell) = \text{div}_g \nabla_g Z^\ell = \Delta_g Z^\ell = 0, \quad (8)$$

where the operator $\Delta_g$ is the Laplace-Beltrami operator on $\Sigma$ associated with the metric $g$. As a specific example, if we embed an image $\Sigma \subset \mathbb{R}^2$ into $\mathbb{R}^3$ via a grey color mapping $I \times x : (u_1, u_2) \mapsto (u_1, u_2, x(u_1, u_2))$, the gradient flow associated with the functional $S$ and hence the Euler-Lagrange equation (8) for the grey channel is exactly the one reported in Section 2.

Therefore, from a high-level perspective, the Beltrami flow derived from the Polyakov action consists in minimizing a functional $\tilde{S}$ representing the Dirichlet energy of the embedding $Z$ computed with respect to a metric $g$ depending precisely via the pullback on the embedding itself.

### B.2 A discrete Polyakov action

We consider the graph counterpart to the problem of minimizing a generalized Dirichlet energy with respect to the embedding. Let $G = (\mathcal{V}, \mathcal{E})$ be a simple, undirected and unweighted graph with $|\mathcal{V}| = n$. Also let $L^2(\mathcal{V})$ and $L^2(\mathcal{E})$ denote the space of signal functions $y : \mathcal{V} \rightarrow \mathbb{R}$ and $Y : \mathcal{E} \rightarrow \mathbb{R}$ respectively. Recall that the graph gradient of $y \in L^2(\mathcal{V})$ is defined by

$$\nabla_g y \in L^2(\mathcal{E}) : (i, j) \mapsto y_j - y_i.$$ 

Its adjoint operator is called graph divergence and satisfies

$$\text{div}_g Y \in L^2(\mathcal{V}) : i \mapsto \sum_{j : (i,j) \in \mathcal{E}} Y_{ij}.$$ 

Assume now we have a graph embedding $Z : \mathcal{V} \rightarrow \mathbb{R}^{d'} \times \mathbb{R}^d$ of the form $z_i = (u_i, \alpha \chi_i)$ for some scaling $\alpha \geq 0$, with $1 \leq i \leq n$. The coordinates $u_i$ and $\chi_i$ are called positional and feature coordinates respectively. In analogy with the Polyakov action (6), we introduce a modified Dirichlet energy measuring the smoothness of a graph embedding $Z$ across its different channels: given a family of nonnegative maps $(\psi_{ij}^\ell)$, with $\psi_{ij}^\ell : \mathbb{R}^{(d' + d)} \rightarrow \mathbb{R}$, $1 \leq i, j \leq n$ and $1 \leq \ell \leq d' + d$, we define

$$S[Z, \psi] := \frac{1}{2} \sum_{\ell=1}^{d' + d} \sum_{i,j=1}^{n} A_{ij} \psi_{ij}^\ell (Z), \quad (9)$$

3
where $A$ is the graph adjacency matrix. Defining the following $\psi$-weighted norm for each $\ell \in [1, d' + d]$

$$||\nabla \ell \tilde{z}^\ell||^2_\psi := \sum_{j=1}^{n} A_{ij} \psi^\ell_{ij}(Z),$$

then (9) can be written as

$$S[Z, \psi] = \frac{1}{2} \sum_{\ell=1}^{d'+d} \sum_{i,j=1}^{n} ||\nabla \ell \tilde{z}^\ell||^2_\psi. \quad (10)$$

Beyond the notational similarity with (6), the quantity $S[Z, \psi]$ sums the integrals over all channels $1 \leq \ell \leq d' + d$ - i.e. summations on the vertex set - of the norms of the gradients of $\tilde{z}^\ell$ exactly as for the continuum Polyakov action. The dependence of such norms on the embedding is to take into account that in the smooth case the Beltrami flow imposes the constraint (7), meaning that the metric $g$ with respect to which we compute the gradient norm of the embedding depends on the embedding itself. We note that the choice

$$\psi^\ell_{ij}(Z) = (\tilde{z}_j^\ell - \tilde{z}_i^\ell)^2 = (\nabla g z^\ell (i, j))^2; \quad (11)$$

yields the classic Dirichlet energy of a multi-channel graph signal

$$S[Z] = \frac{1}{2} \sum_{\ell=1}^{d'+d} \sum_{i,j=1}^{n} A_{ij} (\nabla g z^\ell (i, j))^2,$$

From now on we refer to the function $S : (Z, \psi) \mapsto S[Z, \psi]$ in (9) as the discrete Polyakov action.

### B.3 Proof of Theorem 1

We recall that given an embedding $Z : \mathcal{V} \rightarrow \mathbb{R}^{d'+d}$, we are interested in studying a discrete diffusion equation of the form

$$\frac{\partial z_i(t)}{\partial t} = \sum_{j : (i,j) \in \mathcal{E}} a(z_i(t), z_j(t)) (z_j(t) - z_i(t)) \quad z_i(0) = z_i; \quad i = 1, \ldots, n; \quad t \geq 0. \quad (12)$$

The coupling $a(z_i(t), z_j(t))$ is called diffusivity. We now prove that the differential system above is the gradient flow of the discrete Polyakov action $S[Z, \psi]$ under additional assumptions on the structure of the diffusivity. We restate Theorem 1 in a more precise way:

**Theorem 1.** Let $(\psi^\ell_{ij})$ be a family of maps $\psi^\ell_{ij} : \mathbb{R}^n \times (d'+d) \rightarrow \mathbb{R}$ satisfying the assumptions

(i) There exist a family of maps $(\tilde{\psi}_{ij})$, with $\tilde{\psi}_{ij} : \mathbb{R}^n \rightarrow \mathbb{R}$, such that

$$\psi^\ell_{ij}(Z) = \tilde{\psi}_{ij} (||z_i - z_1||^2, \ldots, ||z_i - z_n||^2) (z_j^\ell - z_i^\ell)^2. \quad (13)$$

(ii) If we write $\tilde{\psi}_{ij} : (p_1, \ldots, p_n) \mapsto \tilde{\psi}_{ij}(p_1, \ldots, p_n)$, then we require

$$\partial_{p_k} \tilde{\psi}_{ij}(p) = 0, \text{ if } (i, k) \notin \mathcal{E}.$$

Then, the gradient flow associated with $S[Z, \psi]$ is given by (12), with the diffusivity $a$ satisfying

$$a(z_i(t), z_j(t)) = (\tilde{\psi}_{ij} + \tilde{\psi}_{ji})(Z(t)) + \sum_{k : (i,k) \in \mathcal{E}} \partial_\ell \tilde{\psi}_{ik}(Z(t)) ||z_k(t) - z_i(t)||^2 + \sum_{k : (j,k) \in \mathcal{E}} \partial_\ell \tilde{\psi}_{jk}(Z(t)) ||z_k(t) - z_j(t)||^2$$

where the dependence of $\tilde{\psi}$ on $Z(t)$ is as in (13).

**Remark.** We observe that by taking $\tilde{\psi}_{ij}$ to be the constant function one, we recover the classical case (11). In fact, for such choice the diffusivity satisfies $a(z_i(t), z_j(t)) = 2$ and the gradient flow is given by the graph Laplacian, namely

$$\frac{\partial z^\ell_i(t)}{\partial t} = (2\Delta z^\ell_i(t)).$$
We now use the assumption (ii) to see that

\[ \partial S = \frac{1}{2} \sum_{\ell=1}^{d} \sum_{i,j=1}^{n} A_{ij} \partial z^{\ell}_{ij}(Z). \]  

(14)

From the assumptions we can expand the partial derivative of the map \( \psi^{\ell}_{ij} \) as

\[ \partial z^{\ell}_{ij}(Z) = 2 \sum_{s=1}^{n} \sum_{q=1}^{d} \partial s \psi^{\ell}_{ij}(Z)(z_{ij}^s - z_{ij}^q) \delta_{rq}(\delta_{ik} - \delta_{ik})(z_{ij}^r - z_{ij}^q)^2 + 2 \psi^{\ell}_{ij}(Z)(z_{ij}^r - z_{ij}^q) \delta_{rk}(\delta_{jk} - \delta_{jk}), \]

where we have simply written \( \psi^{\ell}_{ij} \) as \( \tilde{\psi}^{\ell}_{ij} \). Therefore, we can write (14) as

\[ \frac{\partial S[Z]}{\partial z^{\ell}_{kk}} = \sum_{j,s=1}^{n} A_{kj}(\partial s \tilde{\psi}^{\ell}_{kj}(Z))(z_{ks}^j - z_{ks}^q)\norm{z_j - z_k}^2 \]

\[ + \sum_{i,j=1}^{n} A_{ij}(\partial k \tilde{\psi}^{\ell}_{ij}(Z))(z_{ik}^j - z_{ik}^r)\norm{z_i - z_j}^2 \]

\[ + \sum_{j=1}^{n} A_{kj}(\tilde{\psi}^{\ell}_{kj} + \tilde{\psi}^{\ell}_{jk})(Z)(z_{jk}^j - z_{jk}^r). \]

Proof. Once we choose the family of maps \( \psi^{\ell}_{ij} \) as in the statement, the discrete Polyakov action is a map \( S : \mathbb{R}^{n \times (d' + d)} \rightarrow \mathbb{R} \). To ease the notation, given a vector \( Z \in \mathbb{R}^{n \times (d' + d)} \) we write \( Z = (z_1^1, \ldots, z_n^1, \ldots, z_1^{d' + d}, \ldots, z_n^{d' + d}) \). To prove the result we now simply need to compute the gradient of the functional \( S[Z] \): given \( 1 \leq r \leq d' + d \) and \( 1 \leq k \leq n \) we have

\[ \frac{\partial S[Z]}{\partial z^{\ell}_{kk}} = \sum_{j:(j,k) \in E} ((\tilde{\psi}^{\ell}_{kj} + \tilde{\psi}^{\ell}_{jk})(Z) + \sum_{p:(k,p) \in E} \partial j \tilde{\psi}^{\ell}_{kp}(Z)\norm{z_p - z_k}^2 + \sum_{p:(j,p) \in E} \partial k \tilde{\psi}^{\ell}_{jp}(Z)\norm{z_p - z_j}^2)(z_{jk}^j - z_{jk}^r). \]

By inspection, we see that the right hand side can be rewritten as

\[ \frac{\partial S[Z]}{\partial z^{\ell}_{kk}} = \sum_{j:(j,k) \in E} a(z_k, z_j)(z_{jk}^j - z_{jk}^r), \]

with \( a \) the diffusivity given in the statement of Theorem 1. Therefore, the gradient flow associated with \( S[Z] \) is

\[ \frac{\partial z^{\ell}_{kk}(t)}{\partial t} = -\frac{\partial S[Z]}{\partial z^{\ell}_{kk}} = \sum_{j:(j,k) \in E} a(z_k(t), z_j(t))(z_{jk}^j(t) - z_{jk}^r(t)), \]

for each \( 1 \leq k \leq n \) and \( 1 \leq r \leq d' + d \). This completes the proof. □

C Implementation Details

Runtimes We include below the training and inference runtimes of BLEND and BLEND-kNN, compared with corresponding runtimes of GAT. We use a standard GAT implementation (also used to generate results in the main paper) with two layers and eight heads (all other hyperparameters are tuned). Experiments used a Tesla K80 GPU with 11GB of RAM. The relative runtimes of BLEND and BLEND-kNN are largely driven by the respective edge densities and so we also report these in the table. For the small graph BLEND-kNN significantly densifies the graphs (as is consistent with DIGL) and runtimes are longer. For the large graphs the effect is less pronounced and even reversed in the case of Computers.

Computational cost can be computed using from the runtimes based on the AWS rental cost of a Tesla K80 GPU, which is currently less than $1 / hour.
<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
<th>CoauthorCS</th>
<th>Computers</th>
<th>Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLEND (s)</td>
<td>26.79</td>
<td>27.74</td>
<td>168.71</td>
<td>183.29</td>
<td>230.20</td>
<td>69.85</td>
</tr>
<tr>
<td>BLEND_kNN (s)</td>
<td>44.65</td>
<td>83.84</td>
<td>387.86</td>
<td>216.94</td>
<td>157.77</td>
<td>73.86</td>
</tr>
<tr>
<td>Av. degree</td>
<td>4.1</td>
<td>3.5</td>
<td>4.5</td>
<td>8.9</td>
<td>36.70</td>
<td>31.8</td>
</tr>
<tr>
<td>Av. degree kNN</td>
<td>32</td>
<td>64</td>
<td>61.1</td>
<td>12.4</td>
<td>22</td>
<td>38.2</td>
</tr>
<tr>
<td>GAT (s)</td>
<td>1.68</td>
<td>1.76</td>
<td>9.85</td>
<td>18.96</td>
<td>27.12</td>
<td>10.65</td>
</tr>
<tr>
<td>BLEND / GAT</td>
<td>15.95</td>
<td>15.72</td>
<td>17.12</td>
<td>9.67</td>
<td>8.49</td>
<td>6.56</td>
</tr>
<tr>
<td>BLEND_kNN / GAT</td>
<td>26.59</td>
<td>47.51</td>
<td>39.36</td>
<td>11.44</td>
<td>5.82</td>
<td>6.93</td>
</tr>
</tbody>
</table>

Table 3: Training times (s) for 100 epochs

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
<th>CoauthorCS</th>
<th>Computers</th>
<th>Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLEND (s)</td>
<td>0.0712</td>
<td>0.0878</td>
<td>0.2955</td>
<td>0.449</td>
<td>0.4603</td>
<td>0.2071</td>
</tr>
<tr>
<td>BLEND_kNN (s)</td>
<td>0.114</td>
<td>0.2458</td>
<td>0.9463</td>
<td>0.4945</td>
<td>0.3029</td>
<td>0.2246</td>
</tr>
<tr>
<td>Av. degree</td>
<td>4.1</td>
<td>3.5</td>
<td>4.5</td>
<td>8.9</td>
<td>36.7</td>
<td>31.8</td>
</tr>
<tr>
<td>Av. degree kNN</td>
<td>32</td>
<td>64</td>
<td>61.1</td>
<td>12.4</td>
<td>22</td>
<td>38.2</td>
</tr>
<tr>
<td>GAT (s)</td>
<td>0.0031</td>
<td>0.0039</td>
<td>0.0070</td>
<td>0.0326</td>
<td>0.0165</td>
<td>0.0093</td>
</tr>
<tr>
<td>BLEND / GAT</td>
<td>22.72</td>
<td>22.46</td>
<td>42.5</td>
<td>13.8</td>
<td>27.91</td>
<td>22.35</td>
</tr>
<tr>
<td>BLEND_kNN / GAT</td>
<td>36.40</td>
<td>62.88</td>
<td>136.08</td>
<td>15.19</td>
<td>18.37</td>
<td>24.23</td>
</tr>
</tbody>
</table>

Table 4: Inference times (s)

**Hyperparameter Tuning**  We tuned the neural ODE based methods using Ray Tune. The remaining results were taken from the Pitfalls of GNNs paper. As this paper applied a thorough hyperparameter search and we replicated their experimental methodology we did not feel that it was necessary to independently tune these methods.

We did not apply Jacobian or kinetic regularisation for CGNN, GODE or GDE. We used regularisation to reduce the number of function evaluations made by the solver, which made training faster and more stable, but did not improve performance.

**Softmax versus Squareplus**  In some cases we found it beneficial to replace the softmax operator with squareplus which replaces $e^x$ with $\frac{1}{2}(x + \sqrt{x^2 + 4})$. This normalises logits (like the softmax), but has a gradient approaching one for large $x$, preventing one edge from dominating the diffusivity function.

**References**


