A Appendix: Partial Variational Autoencoders

A.1 Partial Variational Autoencoders

For our experiments, we base our model on the Partial Variational Autoencoder (P-VAE) [20] - this model combines a traditional variational autoencoder (VAE) model with a PointNet-style set encoder [27], allowing it to efficiently encode and reconstruct partially observed data points. The P-VAE is based on the observation that typically the features in a VAE are assumed to be conditionally independent when conditioned on the latent variable $\mathbf{z}$. That is,

$$p(\mathbf{x}|\mathbf{z}) = \prod_j p(x_j|\mathbf{z})$$

Then, given a data point $\mathbf{x}$ with observed features $\mathbf{x}_O$ and unobserved features $\mathbf{x}_U$, we have that

$$p(\mathbf{x}_U|\mathbf{x}_O, \mathbf{z}) = p(\mathbf{x}_U|\mathbf{z})$$

Hence, if we can infer a posterior distribution over $\mathbf{z}$ from the observed features, we can use this to estimate $p(\mathbf{x}_U|\mathbf{x}_O)$. The P-VAE infers a variational posterior distribution over $\mathbf{z}$ using an amortized inference network (or *encoder* network) $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}_O)$ and approximates the conditional data distribution given a value of $\mathbf{z}$ using a *decoder* network $p_{\boldsymbol{\phi}}(\mathbf{x}_O, \mathbf{x}_U|\mathbf{z})$.

In our model, we extend the decoder to decode the value of a new feature x_n by initialising an additional subnetwork in the decoder which we term a *decoder head*, with parameters $\boldsymbol{\phi}_n$, to extend its output dimension by one. In principal this head could be of any architecture which takes as input the output of the shared layers of the decoder, but in practice we found that simply extending the final layer of weights and biases to accommodate a new output dimension yielded good results while remaining parameter-efficient as the number of output features grows.

A.2 Training P-VAEs

The P-VAE is trained to reconstruct observed features in the partially-observed data point, and in the process learn to infer a variational posterior $q_\theta(\mathbf{z}|\mathbf{x}_O)$ over the latent variable $\mathbf{z}$. The P-VAE is given batches of data points where features from both the meta-train and meta-test sets are hidden from the model. Additionally, each time a particular data point is input, some additional features are also randomly hidden from the model using a Bernoulli mask, in order to ensure the model is robust to different sparsity patterns in the data. The P-VAE is then trained by maximising the Evidence Lower-Bound (ELBO) [21]:

$$\begin{aligned}
\log p(\mathbf{x}_O) &\geq \log p(\mathbf{x}_O) - D_{\mathrm{KL}}(q(\mathbf{z}|\mathbf{x}_O)||p(\mathbf{z}|\mathbf{x}_O)) \\
&= \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x}_O)}\left[\log p(\mathbf{x}_O, \mathbf{x}_U|\mathbf{z})\right] - D_{KL}\left[q(\mathbf{z}|\mathbf{x}_O)||p(\mathbf{z})\right] \\
&= \mathscr{L}_{partial}(\mathbf{x}_O)
\end{aligned}$$

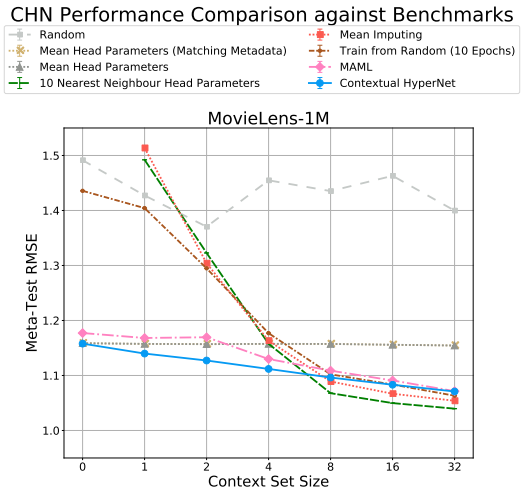


Figure 6: k-shot learning performance on MovieLens-1M where features are arranged into training, meta-training and meta-testing set chronologically by movie release data from oldest to newest.

B Appendix: Chronological Feature Ordering

Throughout our experiments training CHNs, we use random splits of each dataset's features into training, meta-training and meta-testing. While we do not believe that this represents information leakage from future to past in an asymmetric way, we performed an additional experiment on MovieLens-1M where the training, meta-training and meta-testing sets are arranged chronologically by movie release date from oldest to newest. Figure 6 shows the results of this experiment. We see that the overall performance is somewhat worse for all baselines (although this may simply be random noise), but that the relative ordering of the baselines appears largely unchanged.

C Appendix: Baselines

Here we provide additional details and results for the baselines used in our experiments.

C.1 MAML

We adapt the Model-Agnostic Meta Learning [4] technique as a baseline. The decoder head parameters $\boldsymbol{\theta}_n$ are adapted using the MAML algorithm in the 'meta-training' stage. Each new feature $\mathcal{X}$ is viewed as a separate MAML task, with some observed and unobserved values. We sample the tasks in batches of size M and train the inner (a.k.a. fast) model over N steps. The inner model training loss is the ELBO of the PVAE on the observations $\mathcal{L}_{\mathcal{X}_O}$. The meta-model (a.k.a. the slow or outer model) is trained by being given the context set observations, and computing a reconstruction loss on the target set, $\hat{\mathcal{L}}_{\mathcal{T},\mathcal{C}}(f_{\theta'}, \mathcal{X})$. The gradient for the meta-model update is taken over the batch reconstruction losses mean. The full algorithm is detailed in Algorithm 1.

Algorithm 1 Feature-wise Model-Agnostic Meta-Learning with PVAE

Input:
$p(\mathcal{X})$: distribution over features.
α, β: learning rate hyperparameters.
M: meta-batch size, N: number inner iterations.
Initialize θ
while not done **do**
  Sample M features $\mathcal{X}_i \sim p(\mathcal{X})$.
  for all $\mathcal{X}_i$ **do**
    $\theta_{i,0} \leftarrow \theta$
    for $j \leftarrow 0, N$ **do**
      Evaluate ELBO gradient $\nabla_{\theta_{i,j}} \mathcal{L}_{\mathcal{X}_O}(f_{\theta_{i,j}}, \mathcal{X}_i)$ w.r.t. observations in K examples
      Optimize inner model parameters: $\theta_{i,j+1} \leftarrow \theta_{i,j} - \nabla_{\theta_{i,j}} \mathcal{L}_{\mathcal{X}_O}(f_{\theta_{i,j}}, \mathcal{X}_i)$
    end for
    $\theta'_i \leftarrow \theta_{i,N}$
  end for
  Evaluate gradient of mean reconstruction error $\nabla_\theta \frac{1}{M} \sum_{\mathcal{X}_i \sim p(\mathcal{X})} \hat{\mathcal{L}}_{\mathcal{T}_i,\mathcal{C}_i}(f_{\theta'_i}, \mathcal{X}_i)$
  Optimize meta-model parameters: $\theta \leftarrow \theta - \nabla_\theta \frac{1}{M} \sum_{\mathcal{X}_i \sim p(\mathcal{X})} \hat{\mathcal{L}}_{\mathcal{T}_i,\mathcal{C}_i}(f_{\theta'_i}, \mathcal{X}_i)$
end while
Output: θ

Notably, since MAML aims to fit parameters that adapt quickly to new tasks, it allows for fine-tuning at evaluation time, that is, training the model for several iterations from the MAML parameter initialization. Here, we evaluate the model with and without fine-tuning.

In the MAML baseline experiments we use $M = 4$, $N = 10$, ADAM [14] with learning rate $\alpha = \beta = 10^{-2}$ for inner and outer model optimization. The model fine-tuned performance is evaluated over $\{1, 3, 5, 10\}$ epochs and the best results are used. We make use of the higher order optimization facilitated by the `higher` library [10] in the implementation of this baseline.

Figure 7 shows the performance of the MAML baseline for different numbers of fine-tuning epochs and with no fine-tuning. As expected, the baseline with no fine-tuning is outperformed by those where fine-tuning is employed. For the Neuropathic Pain and E-learning datasets, the increase in the number of fine-tuning epochs corresponds to improvement in performance (greater AUROC), whereas in case of MovieLens-1M, performance drops (RMSE increases) with longer fine-tuning, particularly for the smaller context set sizes.

C.2 k-Nearest Neighbour Head Parameters

We consider k-Nearest Neighbour Head Parameters baselines for the values $k \in \{1, 5, 10\}$. Figure 8 shows the performance of this baseline for the different values of k across a range of context set sizes. We expect that as k is increased further, and the number of head parameters averaged over grows,

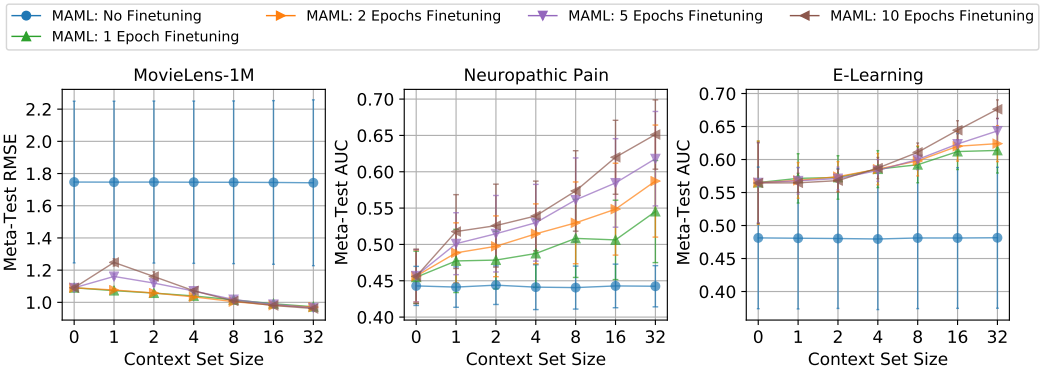


Figure 7: MAML baseline performance comparison for $\{1, 2, 5, 10\}$ fine-tuning epochs and with no fine-tuning. Left plot shows RMSE (lower is better), center and right plots show AUROC (higher is better).

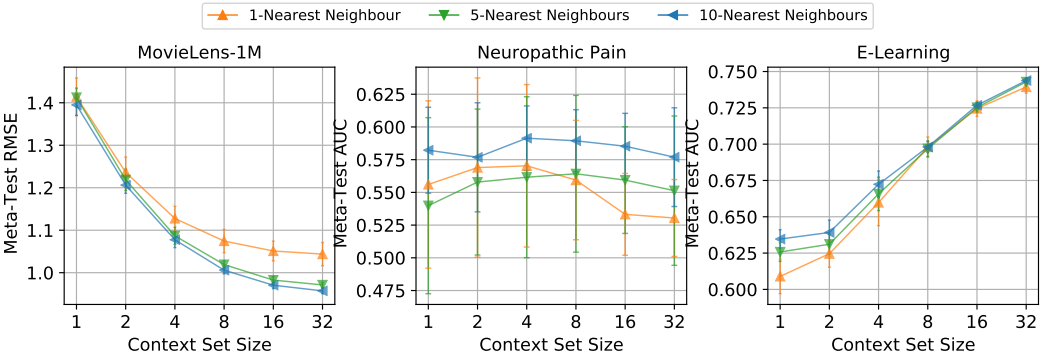
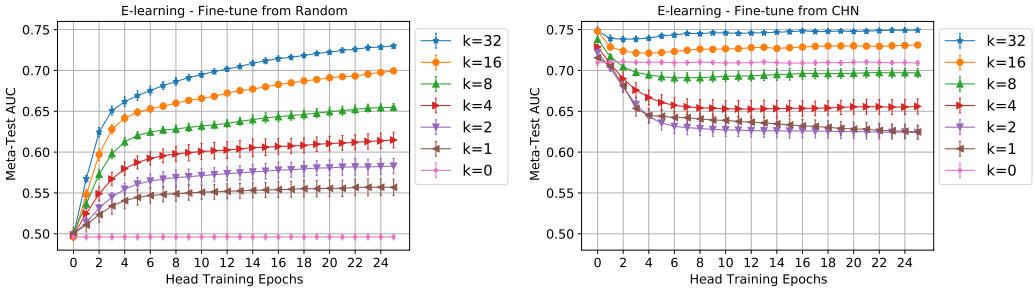


Figure 8: k-Nearest Neighbour Head Parameters baseline performance for $k \in \{1, 5, 10\}$. Context set size here corresponds to the number of observed values used when determining the nearest neighbour heads.

the behaviour will approach that of the mean head parameter baselines. In the main text, 10-Nearest Neighbours is used throughout, as it yields good performance in both the low and high-data regimes.



(a) Train decoder heads from random initialisation (b) Train decoder heads from CHN initialisation

Figure 9: Comparing the predictive performance when training decoder new head parameters in a P-VAE on a range of context set sizes k, on the E-learning dataset (higher is better).

C.3 Fine-Tuning

In our experimental results, we show the performance of training the new decoder heads on their context sets from randomly initialized parameters for 10 epochs, in order to provide a trade-off between predictive accuracy and computational cost. In Figure 9a, we show the predictive performance of the P-VAE on the meta-test set after training randomly initialized head parameters for an increasing number of epochs, for a range of context set sizes k. We see that the performance improves with training in all cases, with better performance achieved as the context set size k increases, and thus the effect of over-fitting is lessened.

Furthermore, in Figure 9b, we perform the same experiment but instead initialising the heads with the CHN parameters. We see that in all cases except $k = 0$ and $k = 32$, training by gradient descent leads to a decrease in performance due to over-fitting, suggesting that the CHN has an implicit regularising effect on the parameter initialisation. We note also that in all cases, the untrained CHN parameters substantially outperform those trained from the random initialisation for all values of k, even after 25 training epochs, with many of the training curves appearing to approach convergence.

Table 2: Hyperparameters and architecture details for the P-VAE and CHN used on each dataset. Feed-forward neural networks are represented by a list of the dimensions of their hidden layers.

		MovieLens-1M	Neuropathic Pain	E-learning
Training				
	Epochs	200	1000	50
	Batch Size	1000	1000	1000
	Learning Rate	1e-3	1e-2	1e-3
	Weight Decay	0	0	0
Meta-Training				
	Epochs	100	300	20
	Batch Size	256	128	128
	Learning Rate	1e-4	1e-3	1e-3
	Weight Decay	1e-3	1e-3	1e-3
Set Encoder				
	Feature Embedding Dim.	50	30	50
	Set Embedding Dim.	30	30	30
Encoder				
	Latent Dim.	150	20	150
	Layers	[200]	[30]	[200]
Decoder				
	Shared Layers	[200]	[30]	[200]
	Output Variance	0.1	-	-
CHN				
	Data point Embedding Dim.	50	25	50
	Context Encoding Dim.	50	25	50
	Context Encoder Layers	[128]	[50]	[50]
	Metadata Encoding Dim.	5	-	20
	Metadata Encoder Layers	[10]	-	[20]
	Param. Pred. Net Layers	[256,256,256]	[64,64]	[50,100,150]

D Appendix: Experiment Details

All models were implemented in PyTorch [25]. All experiments were performed on a single Nvidia Tesla K80 GPU. For training both the P-VAE and the CHN's parameters, the ADAM[14] optimizer was used with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. Training and evaluating a CHN for the specified number of epochs took around 3 minutes on the Neuropathic Pain dataset, around 1.5 hours on the E-learning dataset, and around 8 hours on MovieLens-1M.

Details of hyperparameters and model architectures used for each dataset can be found in Table 2.