Self-Attention Between Datapoints: Going Beyond individual Input-Output Pairs in Deep Learning

Jannik Kossen, Neil Band, Clare Lyle, Aidan Gomez, Tom Rainforth, Yarin Gal ~ 2021 (NeurIPS)

presented by Ruard van Workum at DL seminar/MDS

Parametric vs. non-parametric modelling

Parametric $p(\mathbf{y}^*|\mathbf{x}^*;\theta)$

- Fixed number of parameters, e.g. linear regression
- With non-optimal capacity we can never reach the lowest possible error (Bayes error)
- Utilize assumptions about data distribution to reduce number of parameters
- Can’t utilize direct relationships between datapoints

Non-Parametric $p(\mathbf{y}^*|\mathbf{x}^*;D_{train})$

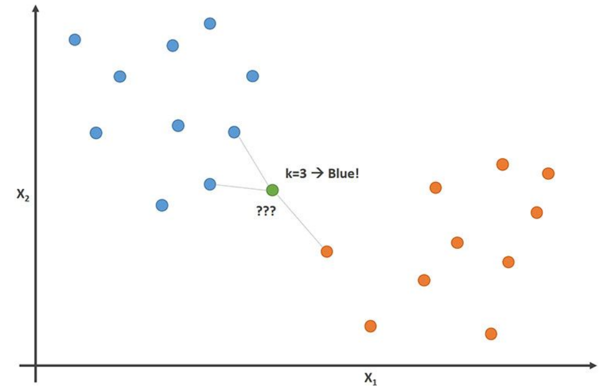
- Number of parameters scales with training data, e.g. nearest neighbor regression
- With increasing training set size, we eventually reach Bayes error
- Minimize assumptions about data distribution with the cost of computational complexity
- Can utilize direct relationships between datapoints

Hybrid forms - example with KNN regression

K-NN (non-parametric): compute label as an average of the labels of the K-nearest neighbouring datapoints, according to a euclidean distance metric in feature space.

K-NN (hybrid form):

1) Compute averages over K-nearest neighbours in an embedding space, where the linear mapping from features to embedding is parametrized by a neural net.
2) Compute averages over K-nearest neighbours in feature space, according to a distance metric which is parametrized by a neural network.



This work: Non-Parametric Transformers (NPT’s)

Challenges the strictly parametric modelling practices in modern deep learning, by extending models with the additional flexibility to use training data to make predictions during test time.

$$P(\mathbf{y}^*|\mathbf{x}^*; D_{train}; \theta)$$

The NPT model models both:
1) interactions between features or **representation learning** (think of this as learning an embedding of the training data -> K-NN example)
2) interactions between data points (think of this as learning custom distance functions -> K-NN example)

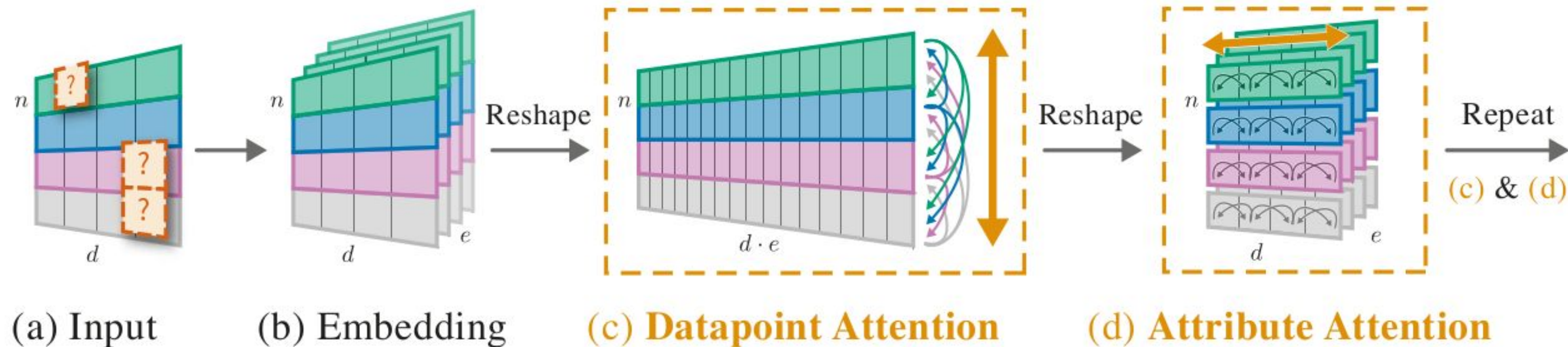
By utilizing end-to-end training, the model can naturally learn from the data how to balance these two.

This work: Non-Parametric Transformers

"*Non-Parametric Transformers explicitly learn relationships between data points to improve predictions.*" 3 most important ingredients for to achieve this:

1) The model is provided with the **entire dataset as input**. If the dataset size exceeds computational constraints, this is approximated with minibatches.
2) They use **self-attention between features** and **between data-points.**
3) The **NPT's training objective is to reconstruct a corrupted version of the input dataset**, in which some fields are masked (similar to NLP). This means that random features and/or targets are masked & loss is minimized on the predictions at these masked entries.

Architecture

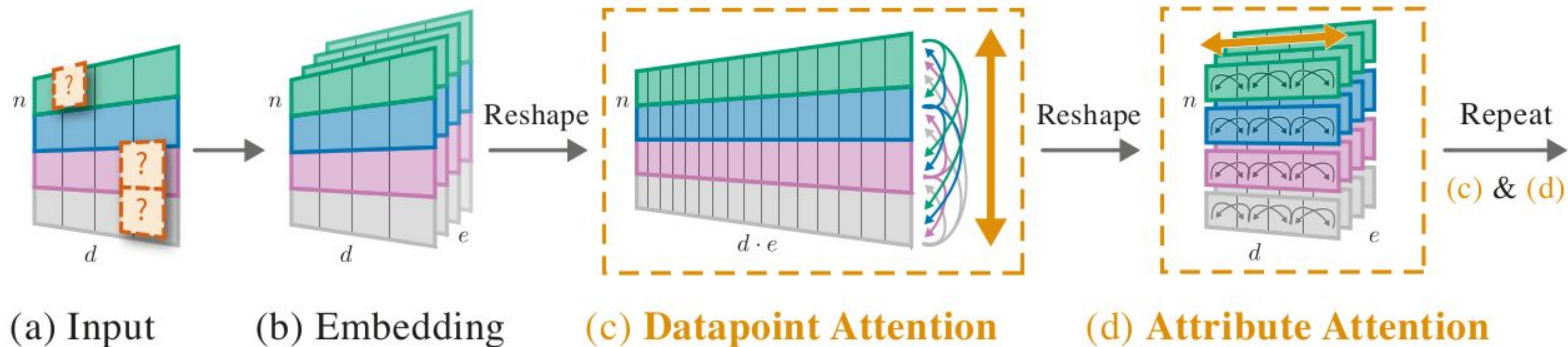


NPTs take as input the entire dataset $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. The datapoints are stacked as the rows of this matrix $\{\boldsymbol{X}_{i,:} \in \mathbb{R}^{d} \mid i \in 1 \ldots n\}$, and we refer to the columns as attributes $\{\boldsymbol{X}_{:,j} \in \mathbb{R}^{n} \mid j \in 1 \ldots d\}$. Each

There is also a binary Mask matrix M with size n by d, indicating masked values.

Therefore NPT takes as input (X, M) and outputs a matrix X* for values masked at input.

Architecture



The input matrix X is first linearly embedded into a matrix of size n x d x e.

Then Multi-Head Self-Attention layers between data points & attributes/features are applied repeatedly.

Note: NPT's are equivariant with respect to permutations in datapoints (along vertical axis in the image)

Architecture: Multi-Head Self-Attention

Attention: **a weighted sum with data-dependent learned weights**

$$Z_i = \sum_i^T \alpha_{ij} h_j$$

For example:

- We can compute a hidden state of a word in a sentence as a weighted sum of other words in the same sentence (where the weights depend on the other words)
- We can compute a hidden state of a node in a graph as a weighted sum of neighbouring nodes/edges in the graph (where the weights depend on the other nodes/edges)

That leaves the question: **How do we compute the data-dependent weights?** (also called alignment weights)

Since our datapoints are represented as vectors, we can utilize all standard vector similarity measures:

- Cosine-similarity
- Dot-product
- Scaled dot-product (used in the original Transformers paper)

Architecture: Multi-Head Self-Attention

In the case of self-attention, we allow the model to learn different mappings from the initial representation to:
(1) **Query** vectors (**Q**) & **Key** vectors (**K**) (the vectors we use to calculate alignment scores)
(2) **Value** vectors (**V**) (the vectors we use to take this weighted sum over)

Which allows you to update all representations using just Matrix multiplication:

$$\mathrm{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^T / \sqrt{h})\boldsymbol{V}.$$

Where Q, K & V are now matrices obtained by stacking the Q, K & V vectors of all datapoints

Architecture: Multi-Head Self-Attention

Multi-head dot-product attention concatenates a series of k independent *attention heads*

$$\text{MHAtt}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \underset{\text{axis}=h}{\text{concat}}(\boldsymbol{O}_1, \ldots, \boldsymbol{O}_k)\boldsymbol{W}^O, \text{ where} \tag{2}$$

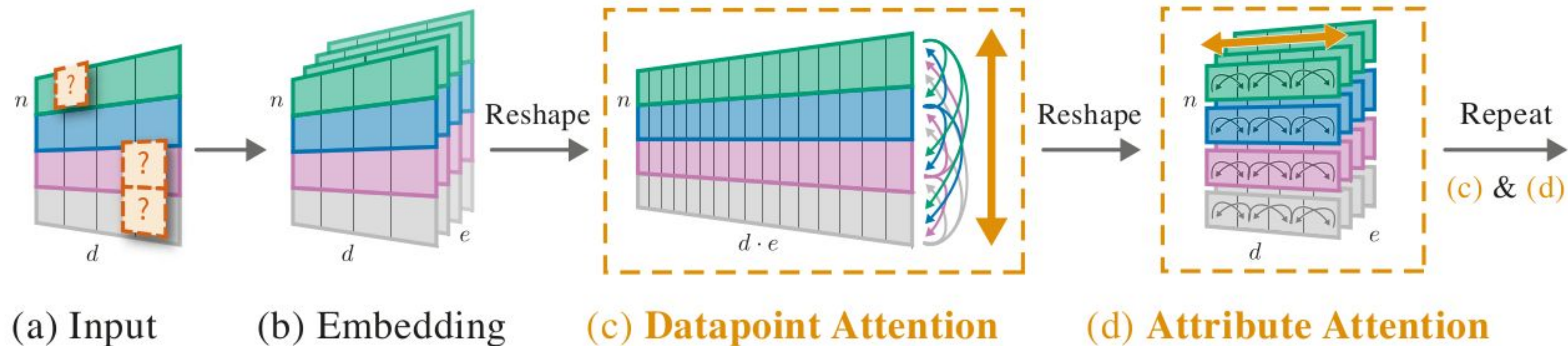
$$\boldsymbol{O}_j = \text{Att}(\boldsymbol{Q}\boldsymbol{W}_j^Q, \boldsymbol{K}\boldsymbol{W}_j^K, \boldsymbol{V}\boldsymbol{W}_j^V). \tag{3}$$

We learn embedding matrices $\boldsymbol{W}_j^Q, \boldsymbol{W}_j^K, \boldsymbol{W}_j^V \in \mathbb{R}^{h \times h/k}, j \in \{1, \ldots, k\}$ for each head j, and $\boldsymbol{W}^O \in \mathbb{R}^{h \times h}$ mixes outputs from different heads. Here, we focus on multi-head *self*-attention,

Self-Attention in this case refers to the fact that Q = K = V = H_i, where H_i represents the hidden representation at layer i.

Multi-Head attention allows for easy parallelization.

Architecture



NPT Objective. During training, we compute the negative log-likelihood loss at training targets $\mathcal{L}^{\text{Targets}}$ as well as the auxiliary loss from masked-out features $\mathcal{L}^{\text{Features}}$. We write the NPT training objective as $\mathcal{L}^{\text{NPT}} = (1 - \lambda)\mathcal{L}^{\text{Targets}} + \lambda\mathcal{L}^{\text{Features}}$, where λ is a hyperparameter. At test time, we only mask and compute a loss over the targets of test points. See Appendix C.5 for optimization details.

Results on standard supervised-learning tasks

Tabular data: Average rank-order on UCI benchmarks for binary class. , multi-class class. & regression tasks

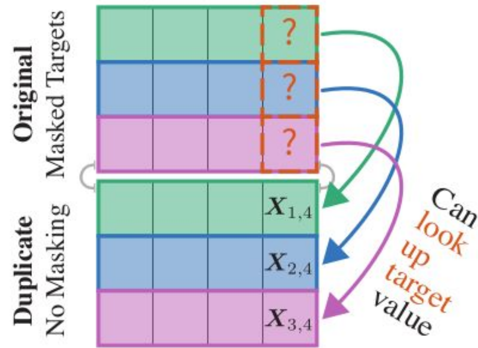
Method	AUROC
NPT	**2.50 ± 0.87**
CatBoost	2.75 ± 0.85
LightGBM	3.50 ± 1.55
XGBoost	4.75 ± 1.25
Gradient Boosting	5.00 ± 0.71
MLP	5.75 ± 1.49
Random Forest	6.00 ± 0.71
TabNet	6.50 ± 1.32
k-NN	8.25 ± 0.48

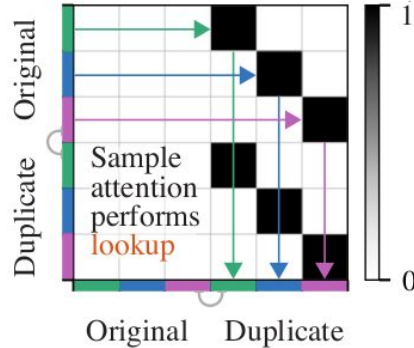
Method	Accuracy
NPT	**2.50 ± 0.50**
XGBoost	**2.50 ± 1.50**
MLP	3.00 ± 2.00
CatBoost	3.50 ± 0.50
Gradient Boosting	3.50 ± 1.50
Random Forest	6.50 ± 0.50
TabNet	7.50 ± 0.50
LightGBM	7.50 ± 1.50
k-NN	8.50 ± 0.50

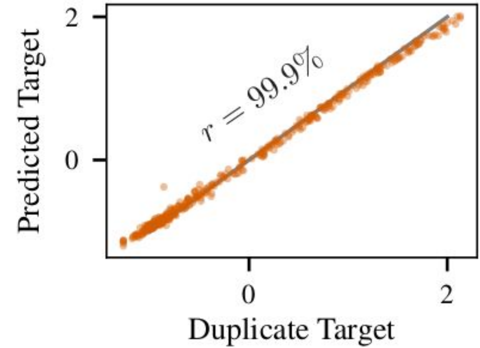
Method	RMSE
CatBoost	**3.00 ± 0.91**
XGBoost	3.25 ± 0.63
NPT	3.25 ± 1.31
Gradient Boosting	4.00 ± 1.08
Random Forest	4.50 ± 0.87
MLP	5.00 ± 1.22
LightGBM	6.50 ± 1.55
TabNet	6.75 ± 0.95
k-NN	8.75 ± 0.25

Image data: 93.7% on CIFAR-10 (using CNN embedding) & 98.3% on MNIST.

NPT's learn to predict using attention between datapoints







They perform experiments where each mini-batch consists of the original data + duplicate original data with targets masked (this way they hope that the model is forced to perform datapoint lookup).

(They call this a semi-synthetic task, I believe it's just synthetic)

NPT's learn to predict using attention between datapoints (on real data)

Another experiment: At test time, randomize the data for all other datapoints by independently shuffling each of their attributes across the rows. Since this corrupts the information from all datapoints except the one that is tested, this should deteriorate performance if the model correctly learned relationships between data.

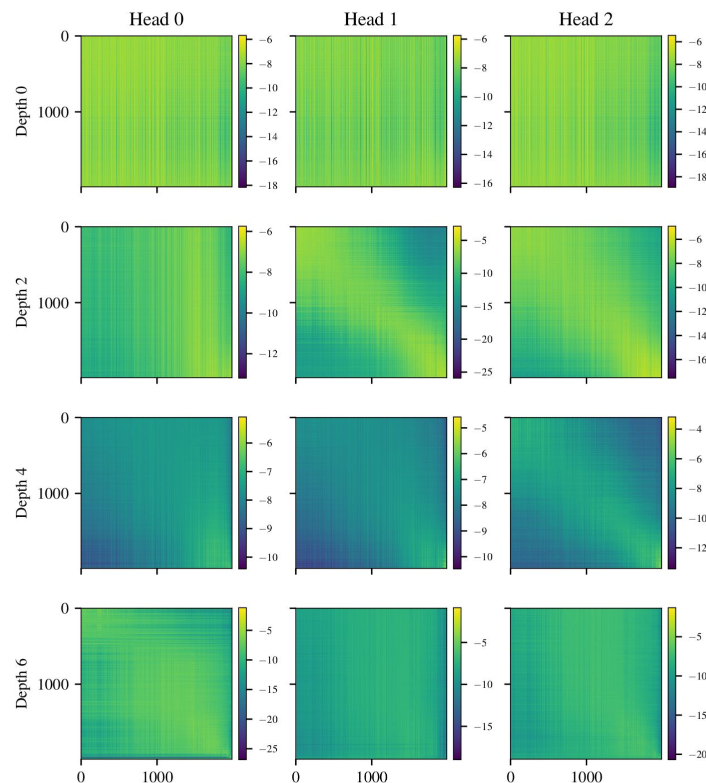
Table 2: Drop in NPT performance after destroying information from other datapoints. Shown are changes in test set performance, where negative values indicate worse performance after corruption.

Δ *Accuracy*	CIFAR-10	Poker	Income	Higgs	MNIST	Forest	Kick	Breast Cancer
	–1.2	–1.1	–1.1	–0.5	–0.4	–0.1	–0.1	0.0
$\Delta RMSE / RMSE$ (%)	Yacht	Protein	Boston	Concrete				
	–52%	–21%	–20%	–7%				

NPT’s learn to rely on similar datapoints for predictions on real data

Qualitative: Looking at the ABD attention weight maps on datapoints sorted by feature space distance -> “diagonal structure”

Quantitative: Removing data points from dataset until performance starts to deteriorate very fast -> “kept” datapoints are closer then “removed” datapoints (in feature space)



Final statement

NPT’s seem to a successful attempt at combining parametric & non-parametric ML approach into 1 model using state-of-the-art attention mechanism and a very flexible training strategy.

Another setting I think would be interesting is if we have very few labelled datapoints & high amount of unlabelled datapoints.