DeepMind

Attention: the Analogue of Kernels in Deep Learning

Hyunjik Kim

27/09/2019

Kernels and Deep Learning

Some recent works @ intersection of Kernels & Deep Learning (DL)

- Deep Gaussian Processes (GPs) (Damianou et al., 2013)
- Deep Kernel Learning (Wilson et al., 2015)
- Convolutional GPs (Van der Wilk et al., 2017)

Kernels and Deep Learning

Some recent works @ intersection of Kernels & Deep Learning (DL)

- Deep Gaussian Processes (GPs) (Damianou et al., 2013)
- Deep Kernel Learning (Wilson et al., 2015)
- Convolutional GPs (Van der Wilk et al., 2017)

–> Ideas from DL incorporated into Kernel Methods

Kernels and Deep Learning

Ideas from Kernels incorporated in DL?

Kernels and Deep Learning

Ideas from Kernels incorporated in DL?

“ Attention ”

Kernels and Deep Learning

Ideas from Kernels incorporated in DL?

“ Attention ”

keys, values   query -> query value

$$(k_i, v_i)_{i \in \mathcal{I}}, q \mapsto v_q = \sum_i w_i v_i$$

weight   kernel

$$w_i = K(q, k_i)$$

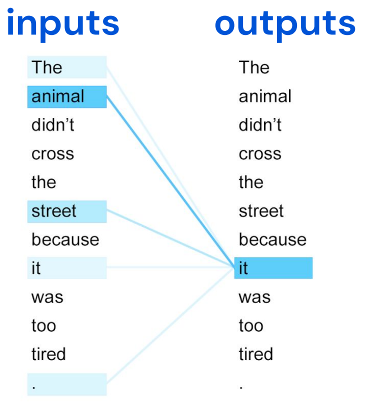
or $$w_{1:N} = \mathrm{softmax}(K(q, k_{1:N}))$$

What is Self-Attention?

- keys = values = queries = sequence of inputs $(x_i)_{i=1}^{N}$

$$q = x_i \mapsto \sum_{j=1}^{N} W_{ij} x_j$$

- $W \in \mathbb{R}^{N \times N}$ is the attention weight matrix
  - Analogous to kernel Gram matrix
- Self-attention maps N inputs to N outputs
  - These layers are stacked to form deep architectures e.g. **Transformer** (Vaswani et al., 2018)



Source for diagram: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Attentive Neural Processes

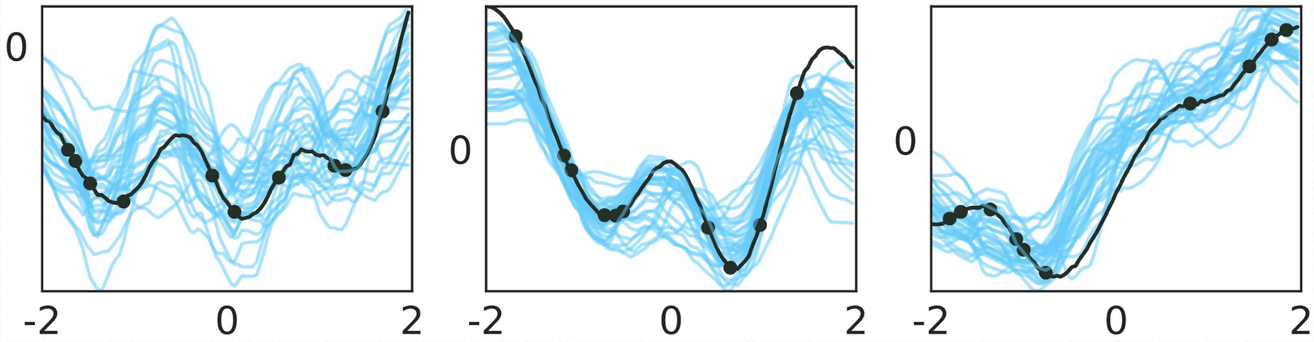
Presented @ ICLR '19

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo,
Ali Eslami, Dan Rosenbaum, Oriol Vinyals, Yee Whye Teh

DeepMind

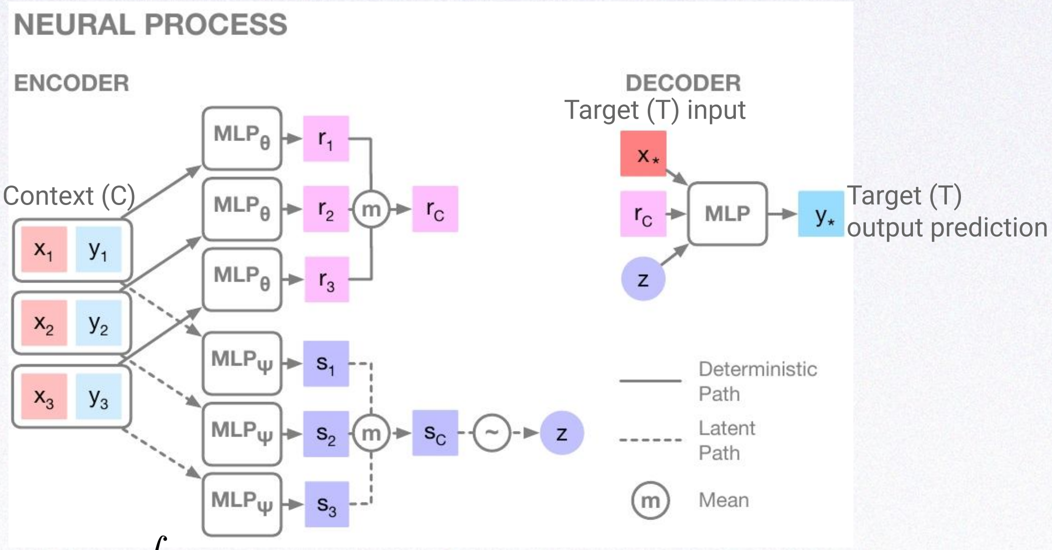
Introduction to Neural Processes (NPs)

- We explore the use of NPs for **regression.**
- Given observed $(x_i, y_i)_{i \in C}$ pairs (**context**), NPs model the function f that maps arbitrary target input x_* to the **target** output y_*.
- Specifically, **NPs learn a distribution over functions** f (i.e. stochastic process) that can explain the context data well while also giving accurate predictions on arbitrary target inputs.



DeepMind

NPs

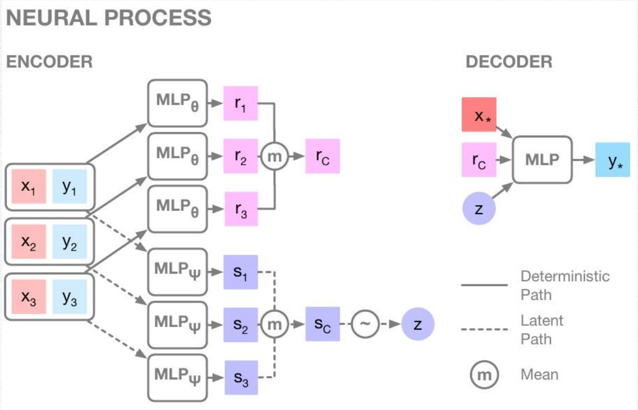


- Define: $p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{x}_C, \boldsymbol{y}_C) := \int p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C, \boldsymbol{z})q(\boldsymbol{z}|\boldsymbol{s}_C)d\boldsymbol{z}$
- Learn by optimising: $\log p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{x}_C, \boldsymbol{y}_C) \geq \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{s}_T)}[\log p(\boldsymbol{y}_T|\boldsymbol{x}_T, \boldsymbol{r}_C, \boldsymbol{z})] - D_{\mathrm{KL}}(q(\boldsymbol{z}|\boldsymbol{s}_T)\|q(\boldsymbol{z}|\boldsymbol{s}_C))$

  with randomly chosen $C \subset T$

DeepMind

Desirable Properties of NPs



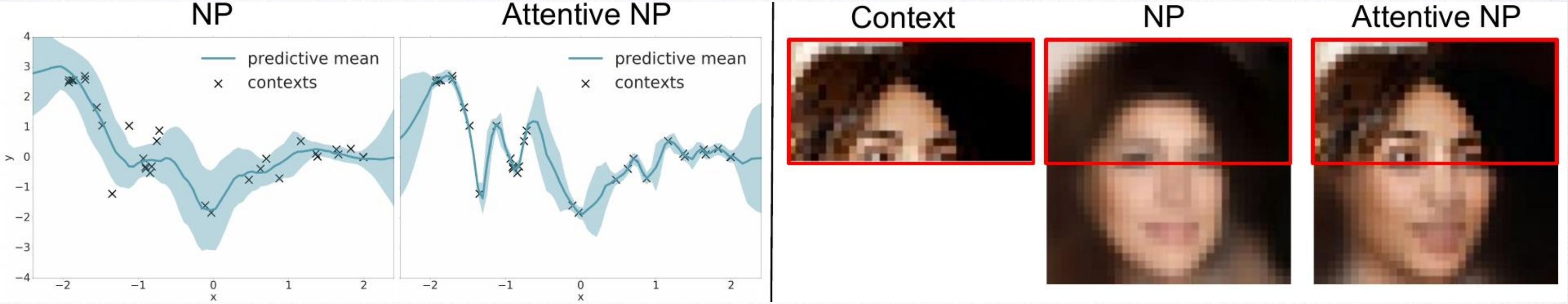
- **Linear scaling: O(*n+m*)** for *n* contexts and *m* targets at train and prediction time

- **Flexibility:** defines a very wide family of distributions, where one **can condition on an arbitrary number of contexts** to predict an arbitrary number of targets.

- **Order invariant** in the context points (due to aggregation of r_i by taking mean)

Problems of NPs

- Signs of **underfitting** in NPs: **inaccurate predictions at inputs of the context**
- **mean-aggregation step in encoder acts as a bottleneck**
  - **Same weight given to each context point**, so difficult for decoder to learn which contexts are relevant for given target prediction.



DeepMind

Desirable properties of GPs

- Kernel tells you which context points x_i are relevant for a given target point x_*
  - $x_* \approx x_i \Rightarrow \mathbb{E}[y_*] \approx y_i$, $\mathbb{V}[y_*] \approx 0$
  - x_* far from all $x_i \Rightarrow \mathbb{E}[y_*] \approx$ prior mean, $\mathbb{V}[y_*] \approx$ prior var
  - i.e. no risk of underfitting.

- In the land of Deep Learning, we can use differentiable ***Attention*** that **learns to attend to contexts relevant to given target**

Attention

- Attention is used when we want to map query x_* and a set of key-value pairs $(x_i, y_i)_{i \in O}$ to output y_*

- It learns which (x_i, y_i) are relevant for the given x_*, which is ultimately what we want the NP to learn.

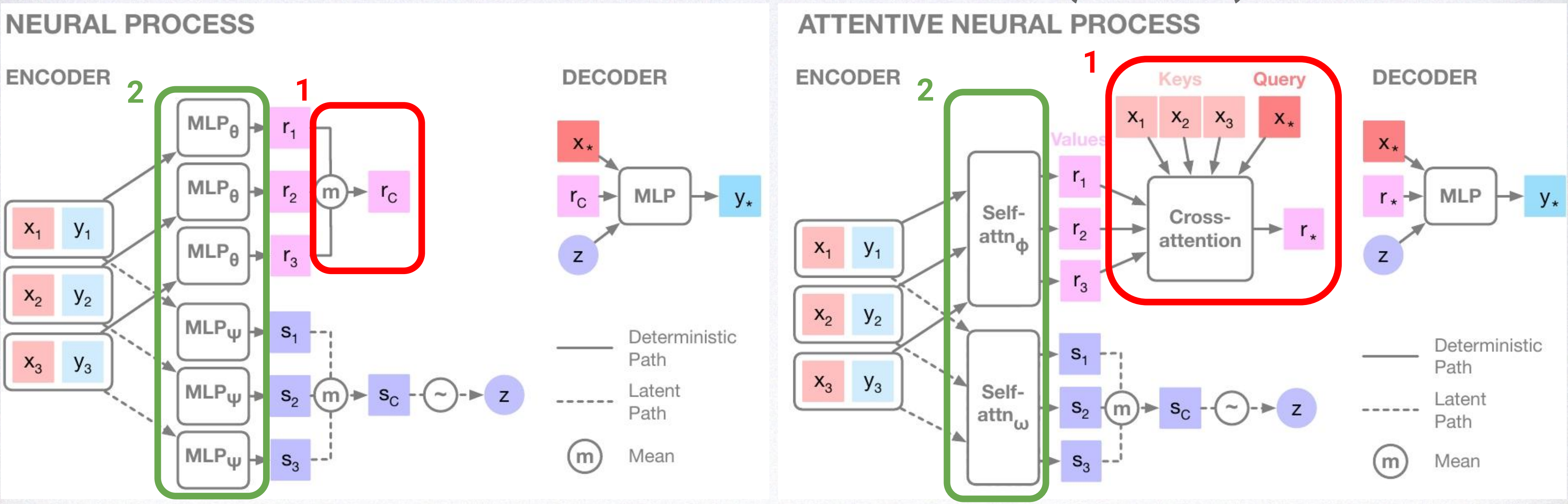
- To help NP learn this, we can **bake into NP an attention mechanism,** and this inductive bias may e.g. help avoid underfitting, enhance expressiveness of NPs, and help it learn faster.

Types of Attention

- **Laplace**: $(w_i)_{i \in C} = softmax[(-||x_i - x_*||_1)_{i \in C}], \quad r_* = \sum_{i \in C} w_i r_i$
- **Dot product**: $(w_i)_{i \in C} = softmax[(\frac{f_\theta(x_i)^\top f_\theta(x_*)}{\sqrt{d}})_{i \in C}], \quad r_*^\theta = \sum_{i \in C} w_i r_i$
  where $f_\theta = MLP_\theta, \quad d = dim(f_\theta(x))$
- **Multihead**: $r_* = Linear(Concat([r_*^{\theta_1}, \ldots, r_*^{\theta_H}]))$

DeepMind

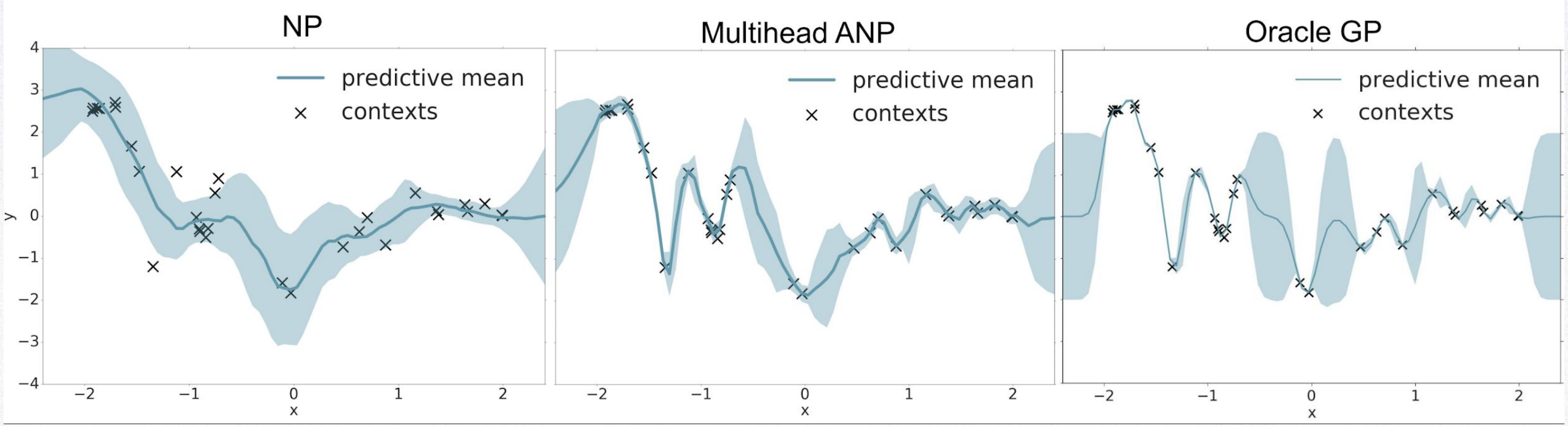
Attentive Neural Processes (ANPs)



- Computational complexity risen to O(*n(n+m)*) but still fast using mini-batch training.

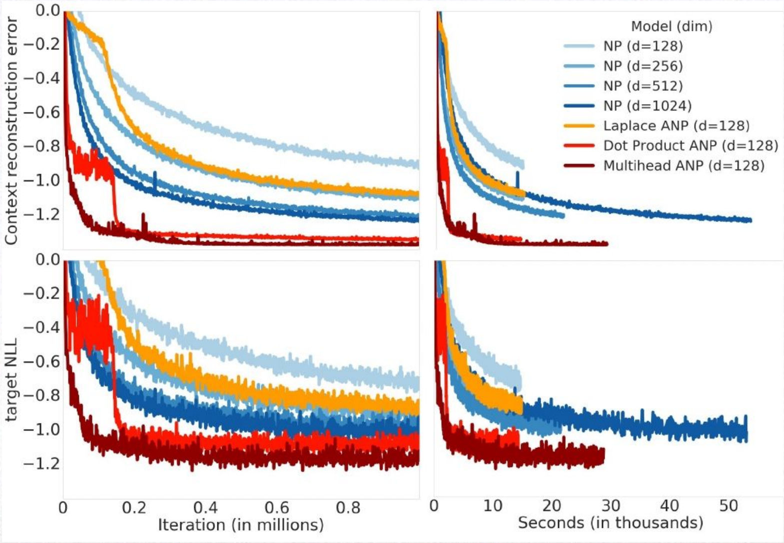
1D Function regression on GP data

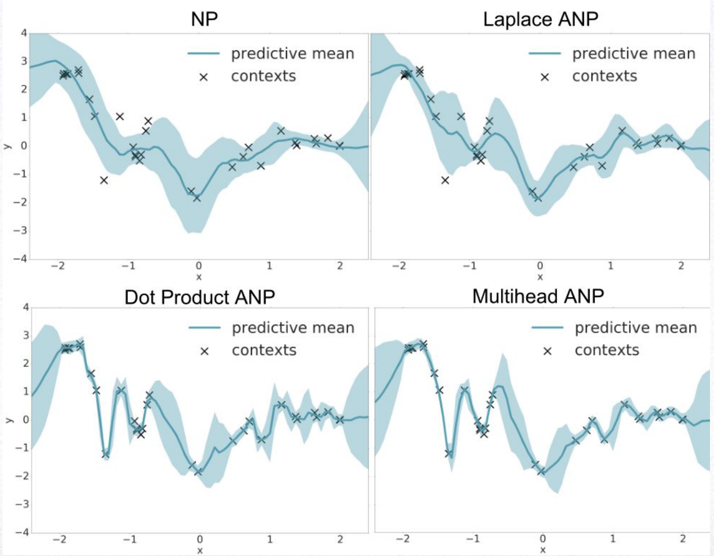
- At every training iteration, draw curve from a GP with random kernel hyperparameters (that change at every iteration).
- Then choose random points on this curve as context and targets, and optimise mini-batch loss



1D Function regression on GP data

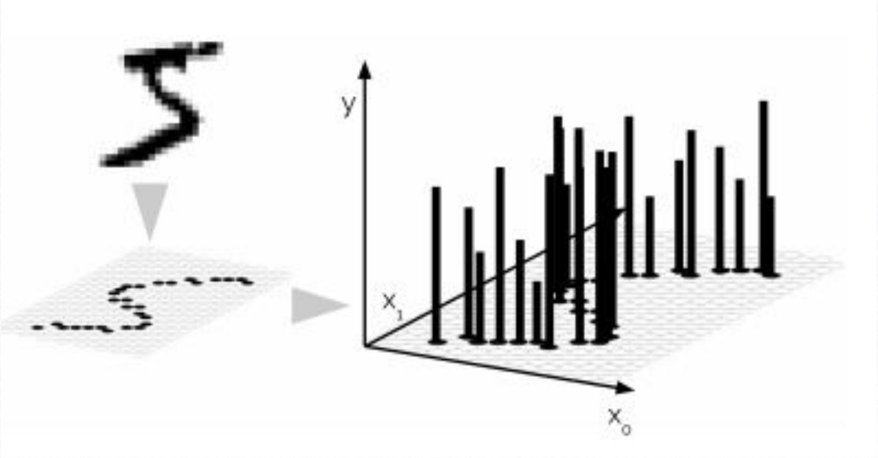
- At every training iteration, draw curve from a GP with random kernel hyperparameters (that change at every iteration).
- Then choose random points on this curve as context and targets, and optimise mini-batch loss

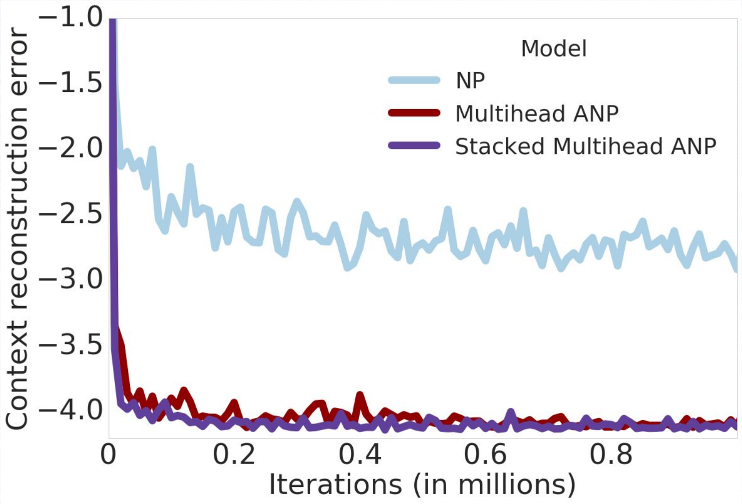




2D Function Regression on Image data

- x_i: 2D pixel coordinate, y_i: pixel intensity (1d for greyscale, 3d for RGB)
- At each training iteration, draw a random image and choose random pixels to be context and target, and optimise mini-batch loss.

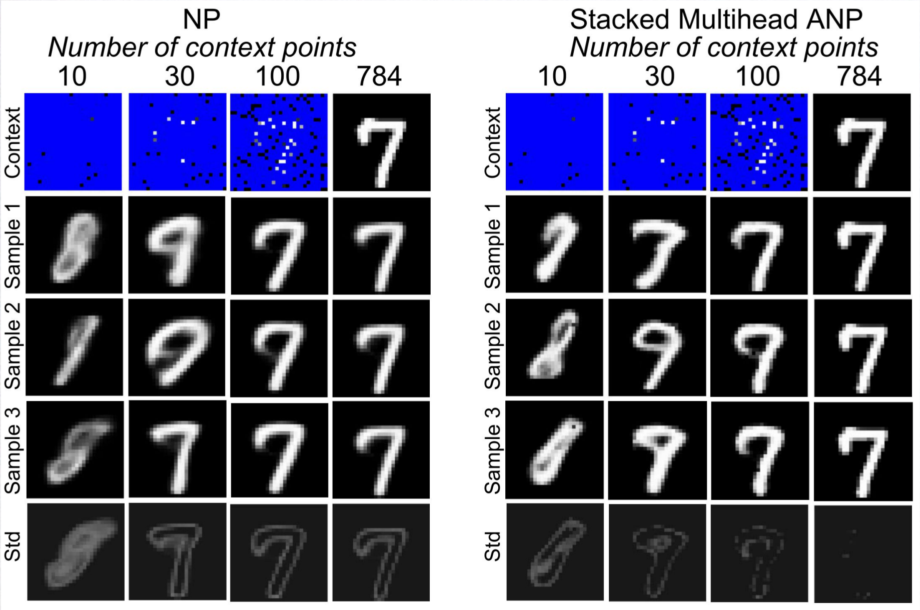


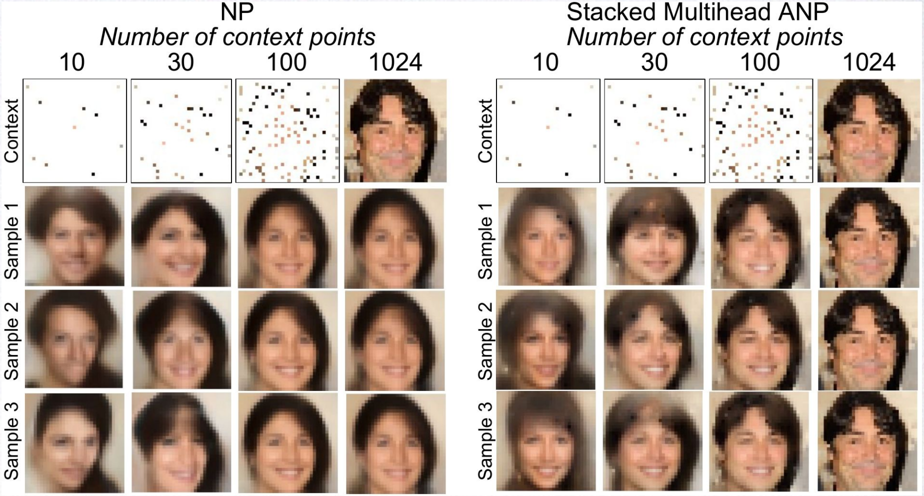


DeepMind

2D Function Regression on Image data

Arbitrary Pixel Inpainting



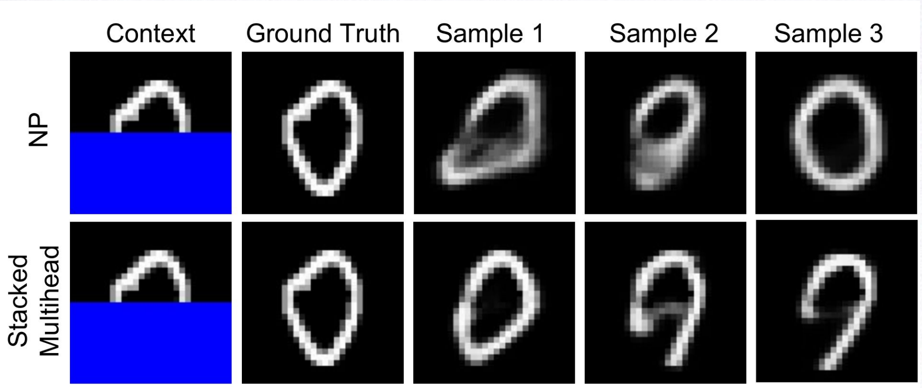


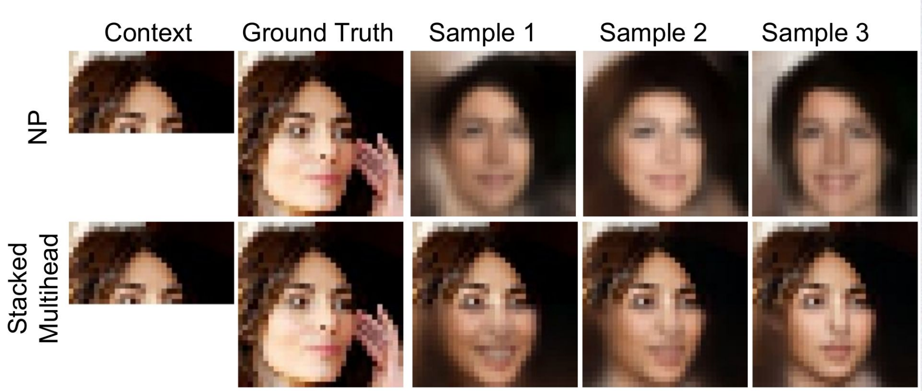
DeepMind

2D Function Regression on Image data

Bottom half prediction

Using **same model** as previous slide (with **same parameter values**):



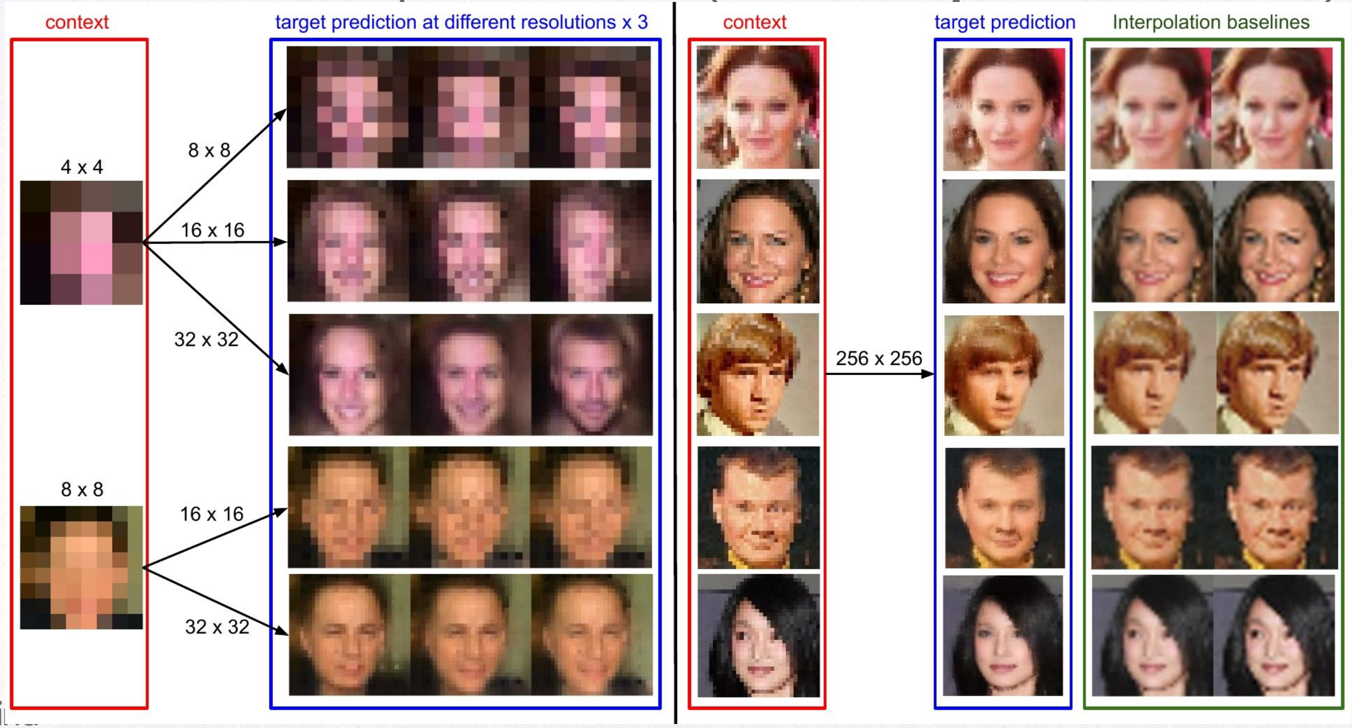


DeepMind

2D Function Regression on Image data

Mapping between arbitrary resolutions

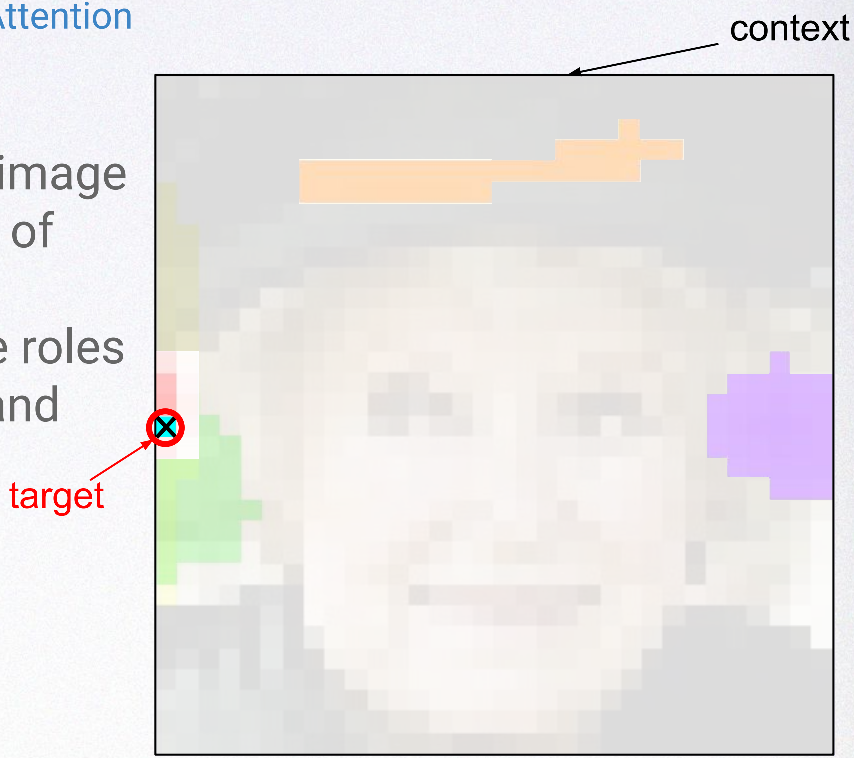
Using **same ANP model** as previous slide (with **same parameter values**):



2D Function Regression on Image data

Visualisation of Attention

- Visualisation of Multihead Attention:
- Target is pixel with cross, context is full image
- Each **colour corresponds to** the weights of **one head of attention**.
- **Each head has different roles**, and these roles are consistent across different images and different target points.

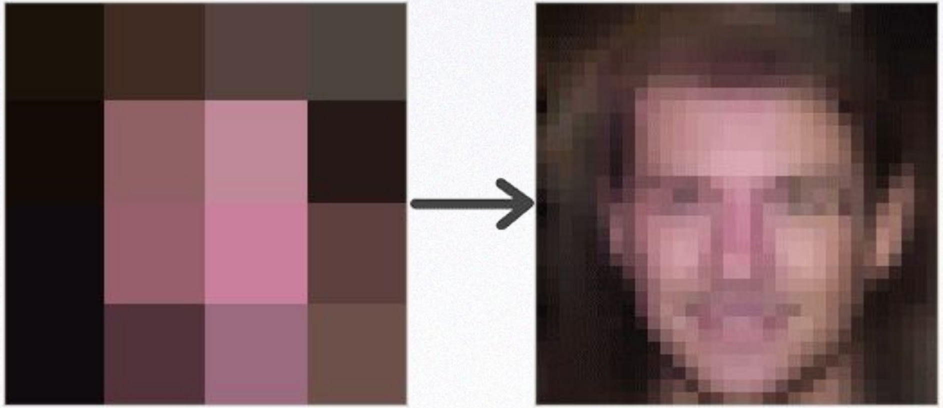


DeepMind

Varying predictions with varying Latents

Bottom half prediction

Super-resolution



Conclusion

Compared to NPs, ANPs:

- Greatly improve the accuracy of context reconstructions and target predictions.

- Allow faster training.

- Expand the range of functions that can be modelled.

with the help of attention (kernels)!

DeepMind