DeepMind

Topics on Attention in Deep Learning

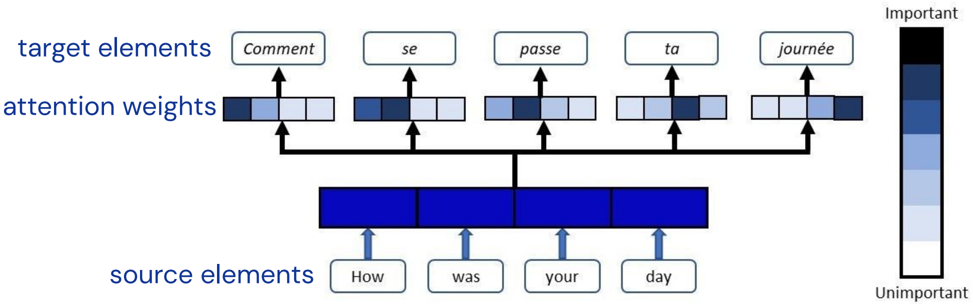
Hyunjik Kim

Link for slides: hyunjik11.github.io

06/08/2020

What is Attention?

- Given source & target sets
  - for every target element, assign **weight** to each source element.
  - high weight = source element is important/related to target element
  - low weight = source element is unimportant/unrelated (as far as the task is concerned)
- Early works on attention focused on Machine Translation (Bhadanau '15, Luong 15'), but also have works on vision tasks (Xu '15)



A woman is throwing a frisbee in a park.

Sources for diagram & picture: https://blog.floydhub.com/attention-mechanism/ , Show, Attend and Tell (Xu '15)
Summary of history of attention in ML: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention & Self-Attention

- Attention described mathematically:

source: key, value   target: query   query value

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

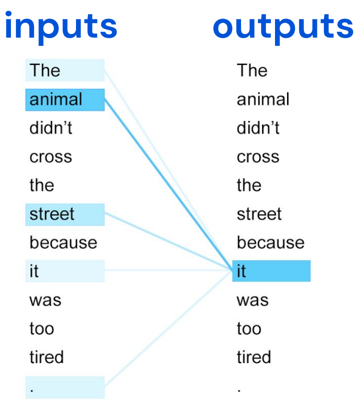
attention weight $w_i = K(q, k_i)$ or $w_{1:N} = \mathrm{softmax}(K(q, k_{1:N}))$

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

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

- 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

DeepMind

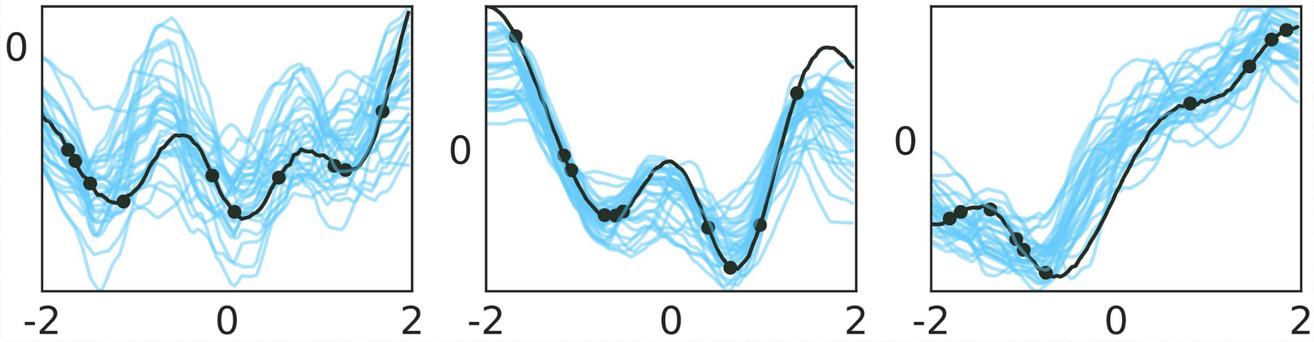
1 Attentive Neural Processes

Presented @ ICLR '19

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

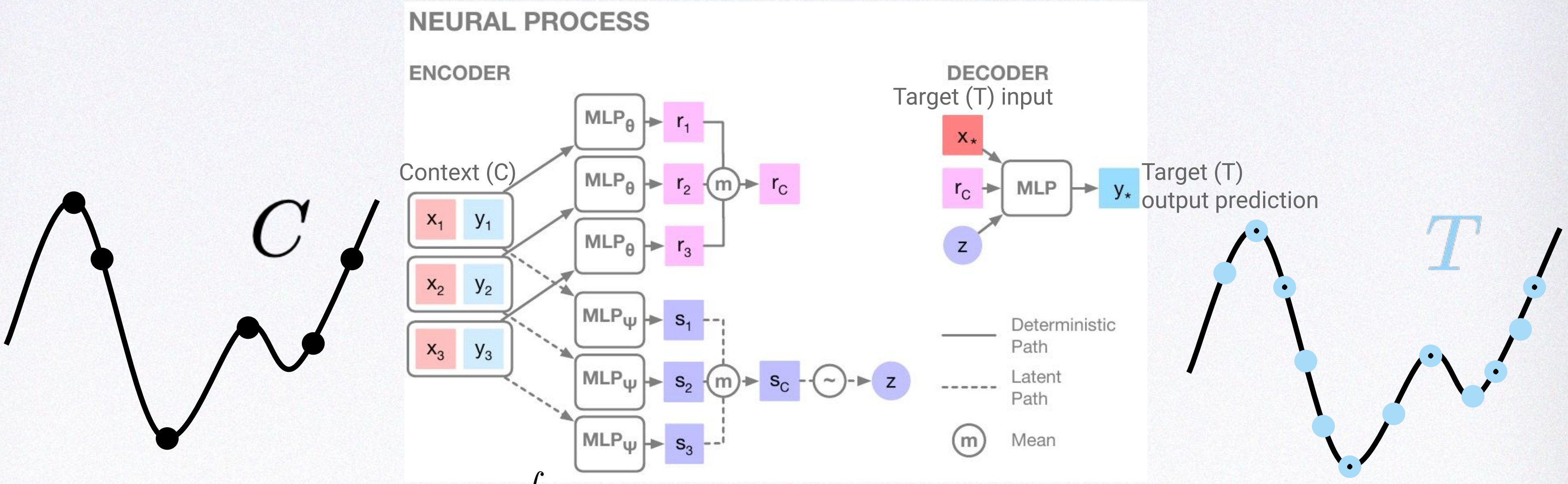
Introduction to Neural Processes (NPs)

- 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

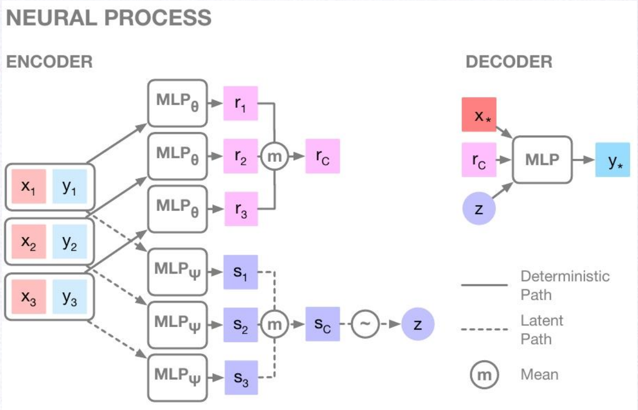
NPs



- 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 $C \subset T$

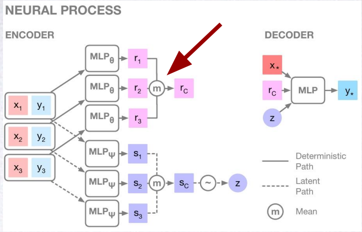
Desirable Properties of NPs



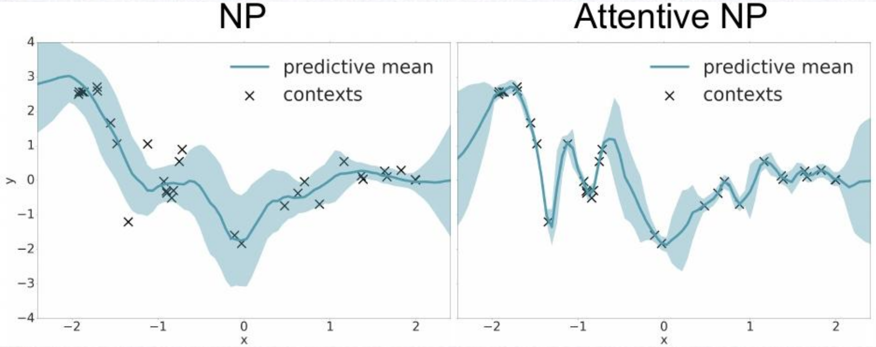
- **Linear scaling: O(*n+m*)** for *n* contexts and *m* targets at train and prediction time
- **Flexibility:** defines a very wide family of distributions, where one **can condition on an arbitrary number of contexts** to predict an arbitrary number of targets.
- **Order invariant** in the context points (due to aggregation of r_i by taking mean)

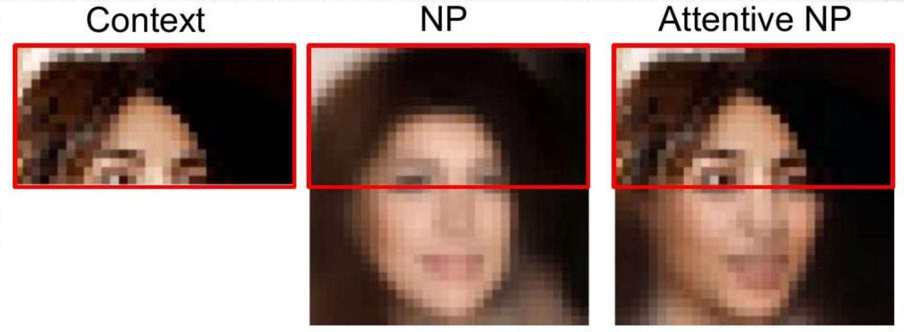
DeepMind

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.

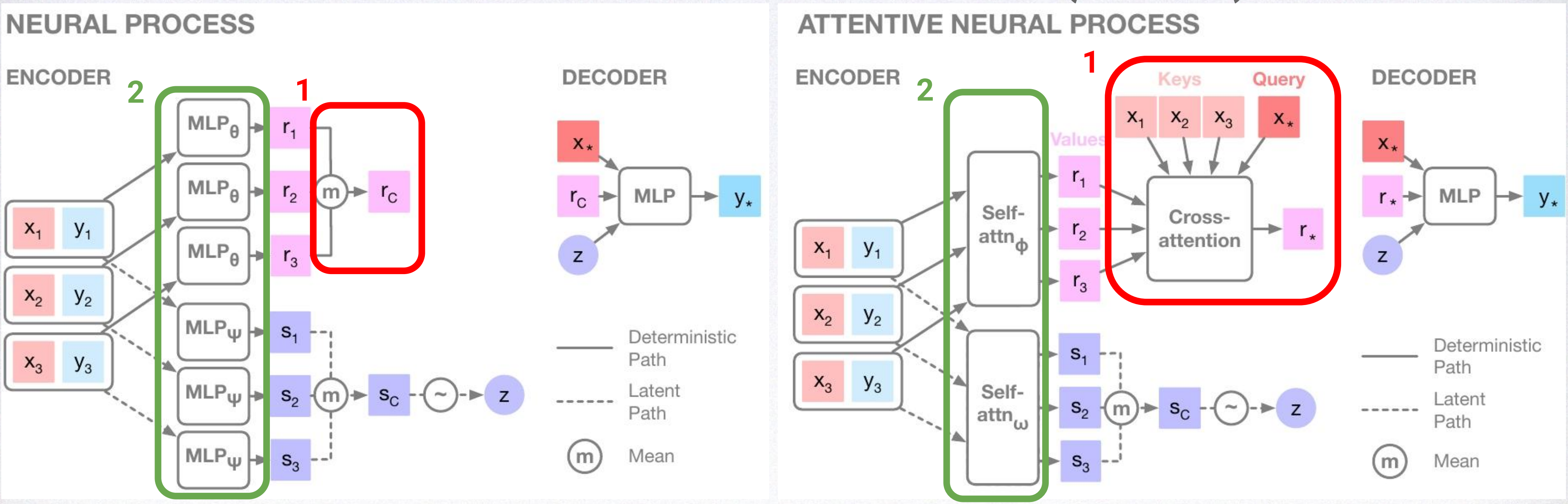
- To help NP learn this, we can **bake into NP an attention mechanism,** and this inductive bias may e.g. help avoid underfitting, enhance expressiveness of NPs, and help it learn faster.

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

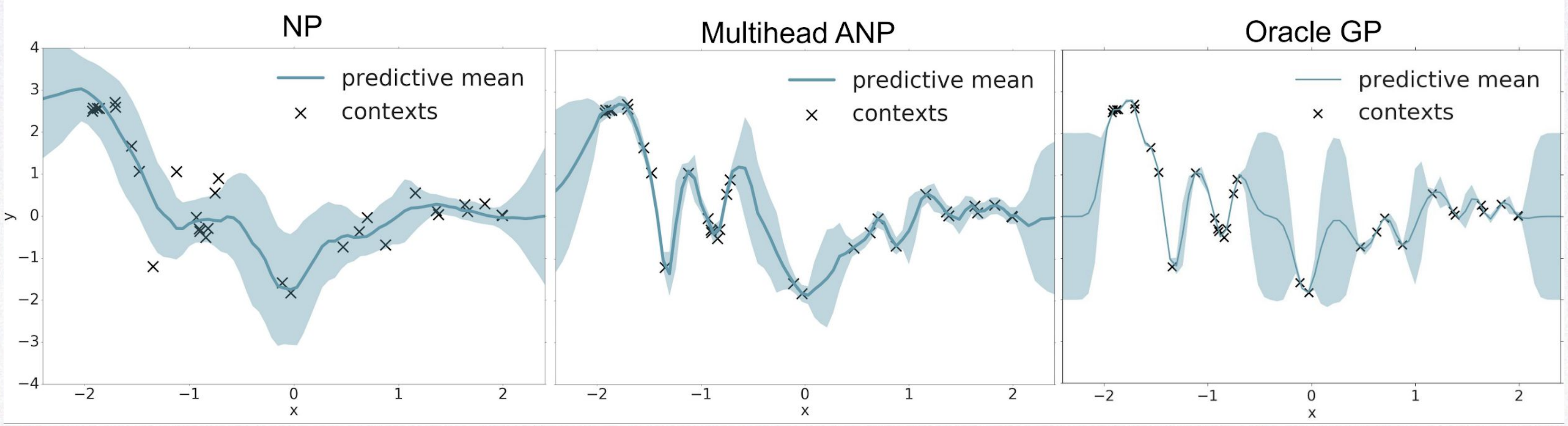
Attentive Neural Processes (ANPs)



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

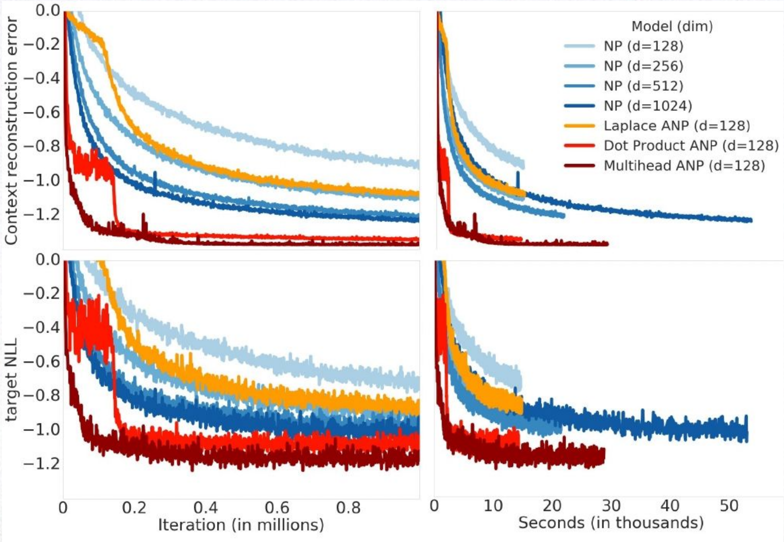
1D Function regression on GP data

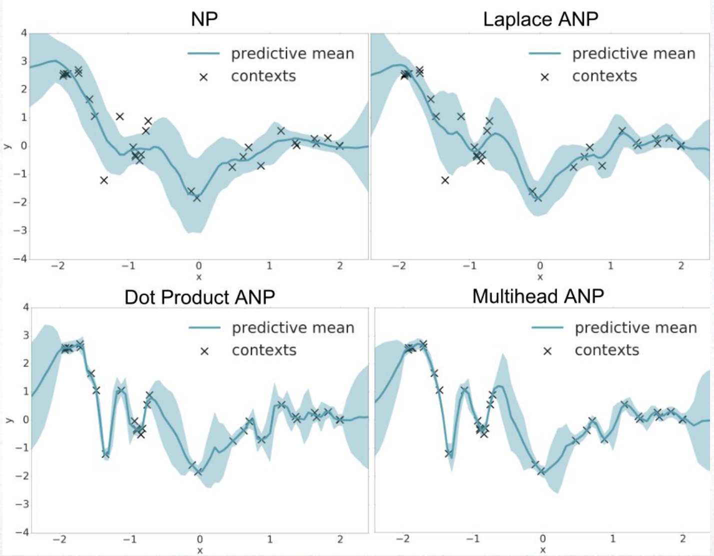
- At every training iteration, draw curve from a GP with random kernel hyperparameters (that change at every iteration).
- Then choose random points on this curve as context and targets, and optimise mini-batch loss



1D Function regression on GP data

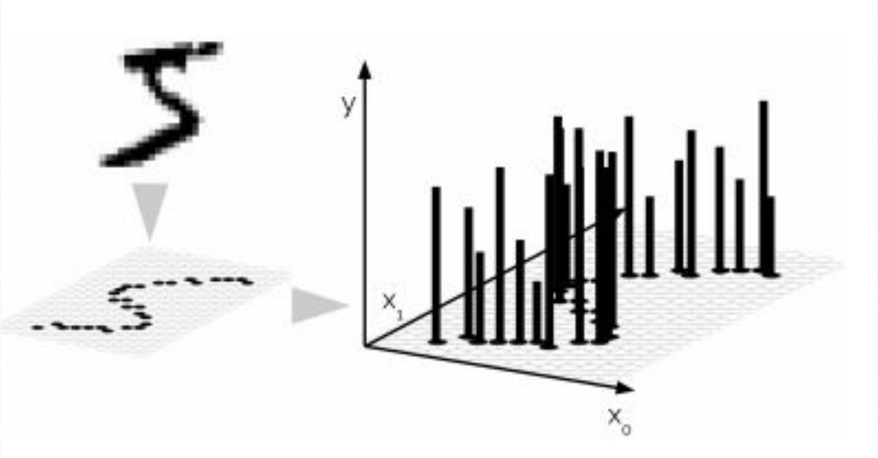
- At every training iteration, draw curve from a GP with random kernel hyperparameters (that change at every iteration).
- Then choose random points on this curve as context and targets, and optimise mini-batch loss

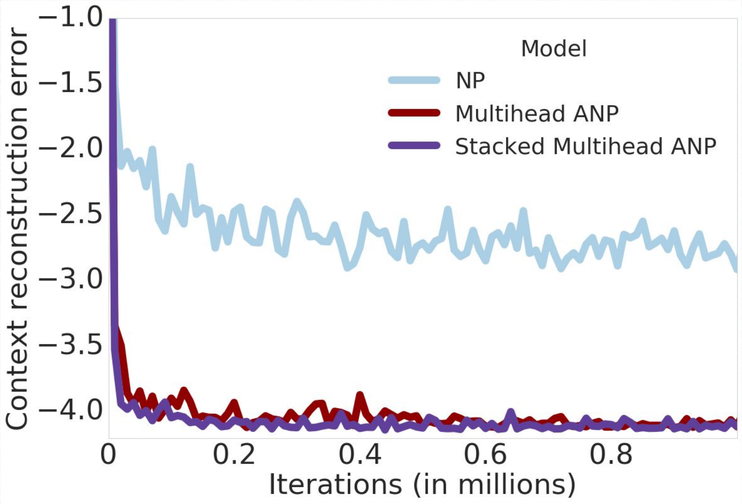




2D Function Regression on Image data

- 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.

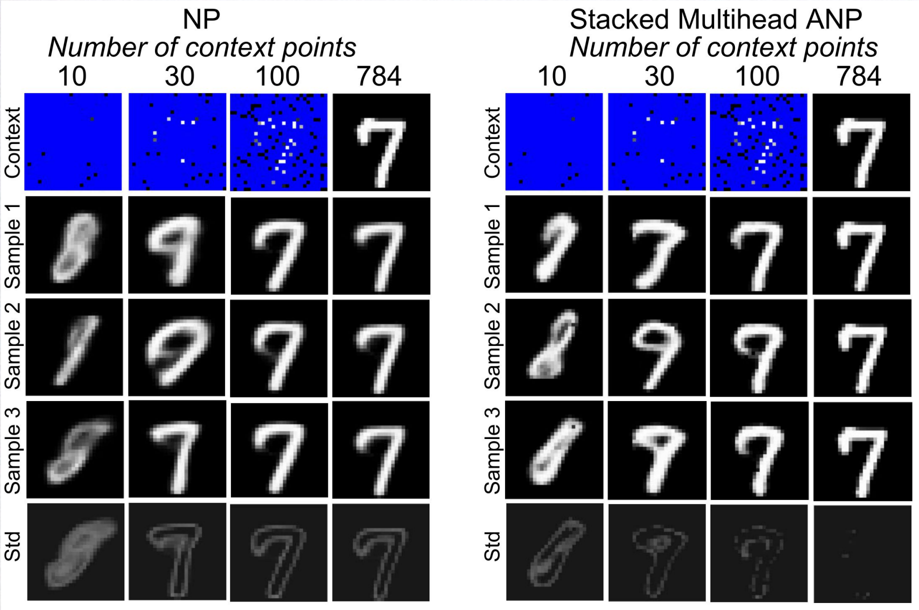


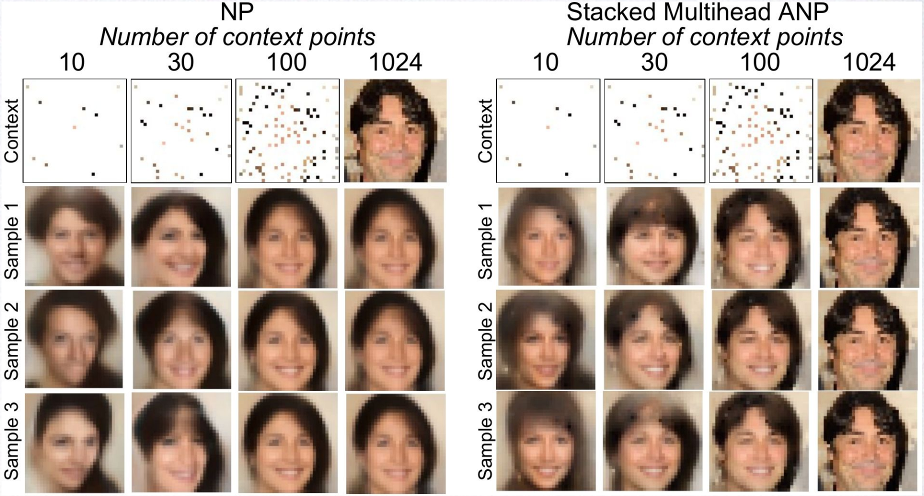


DeepMind

2D Function Regression on Image data

Arbitrary Pixel Inpainting



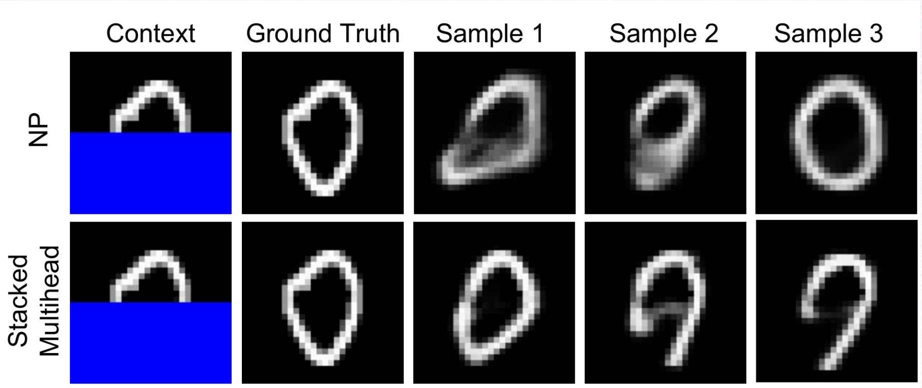


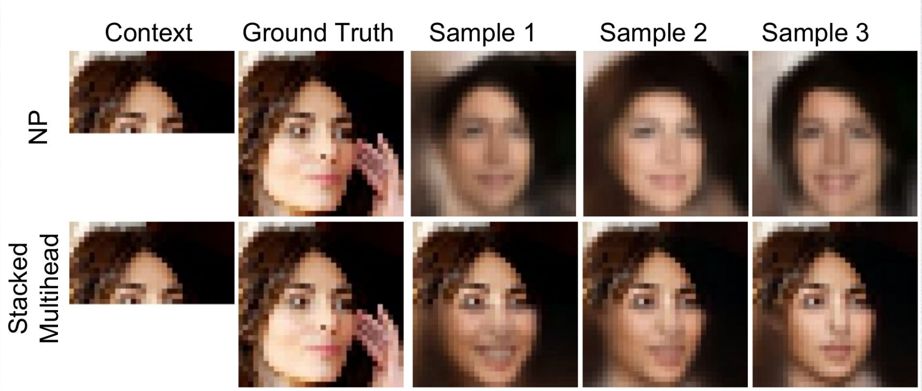
DeepMind

2D Function Regression on Image data

Bottom half prediction

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



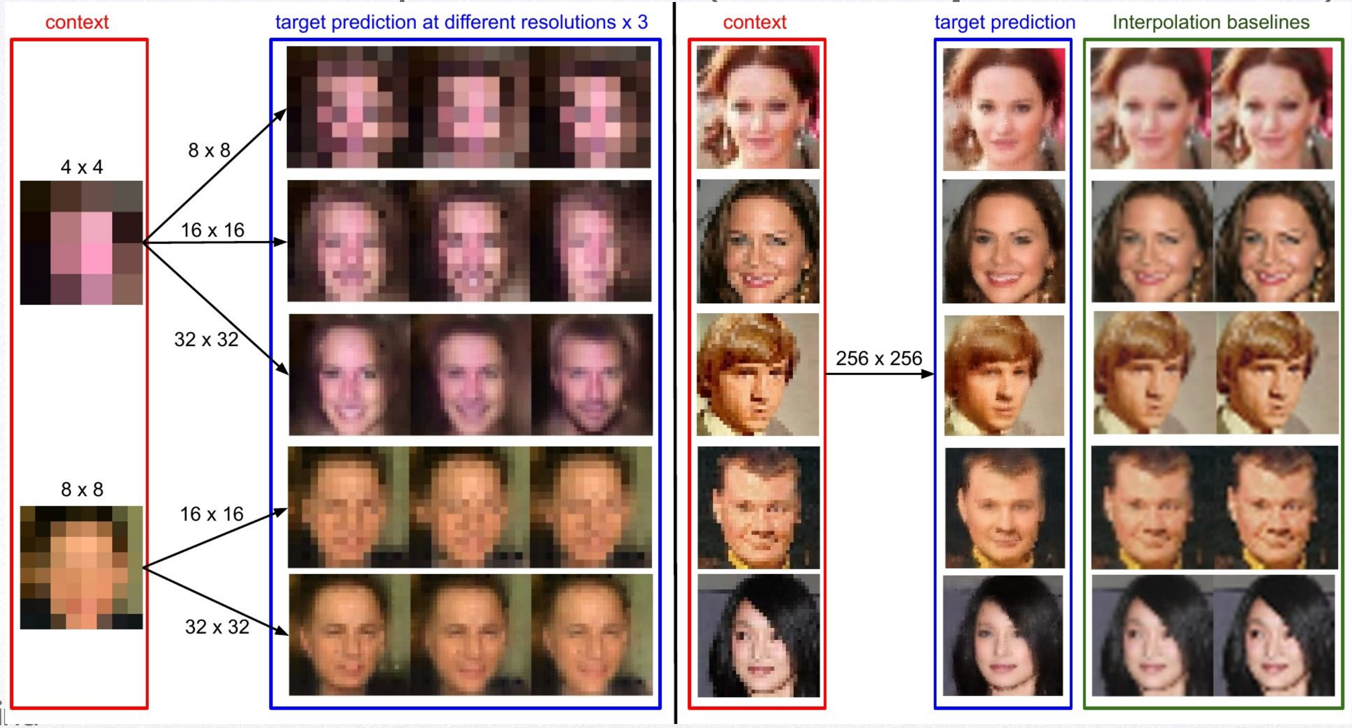


DeepMind

2D Function Regression on Image data

Mapping between arbitrary resolutions

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

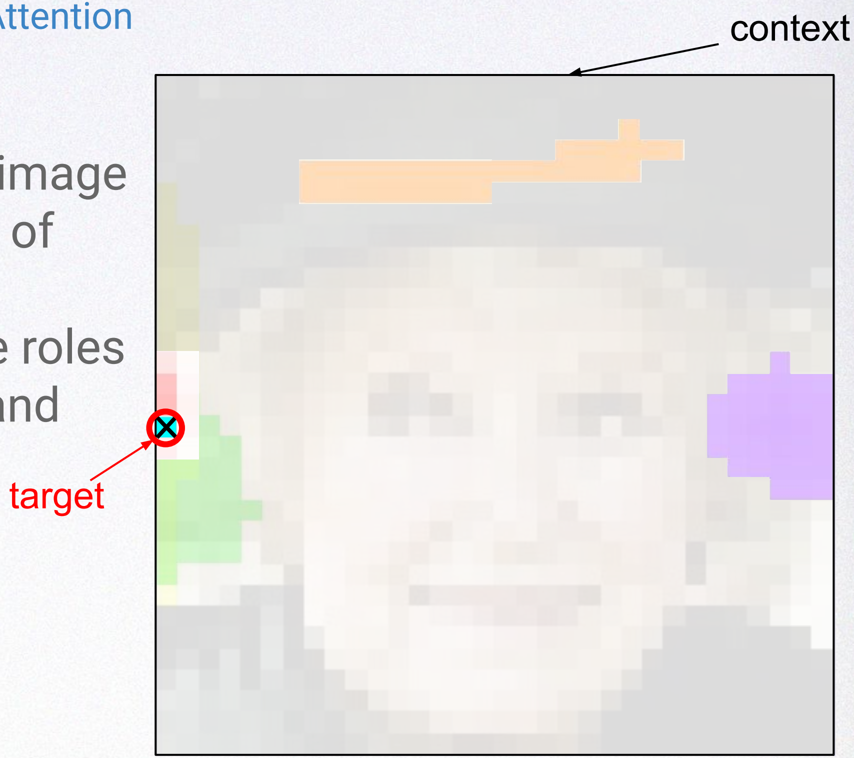


DeepMind

2D Function Regression on Image data

Visualisation of Attention

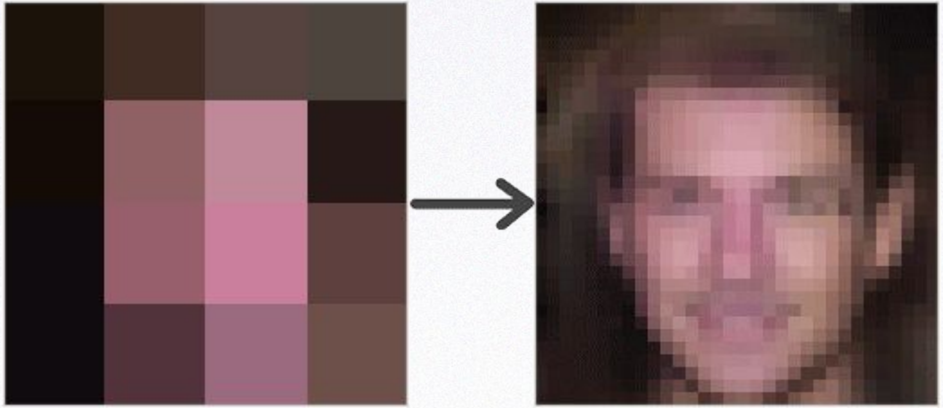
- Visualisation of Multihead Attention:
- Target is pixel with cross, context is full image
- Each **colour corresponds to** the weights of **one head of attention**.
- **Each head has different roles**, and these roles are consistent across different images and different target points.



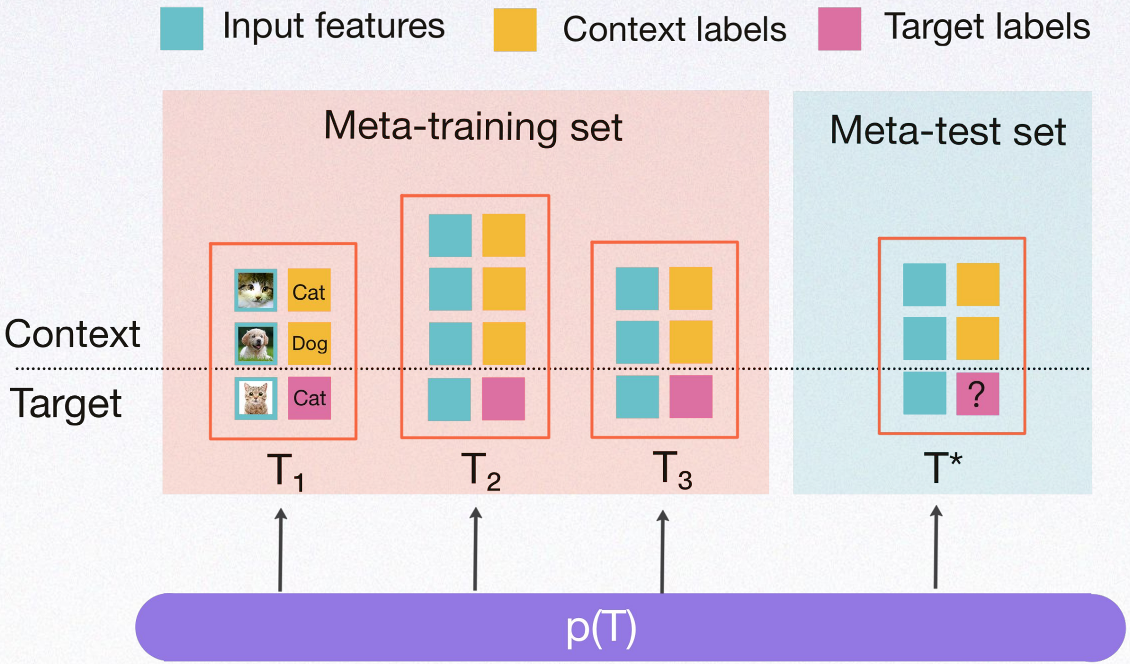
Varying predictions with varying Latents

Bottom half prediction

Super-resolution



NPs for Meta-Learning



- [MetaFun: Meta-Learning with Iterative Functional Updates]() (Xu et. al, ICML 2020)

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!

DeepMind

2 The Lipschitz Constant of Self-Attention

arXiv: https://arxiv.org/abs/2006.04710 (in submission)

Hyunjik Kim, George Papamakarios, Andriy Mnih

Lipschitz constant: Motivation

When are **Lipschitz constants** useful in Deep Learning?

- provable adversarial robustness (Cisse et al. '17, Tsuzuku et al. '18)
- generalisation bounds (Sokolić et al. '17)
- estimating Wasserstein distance (Peyré & Cuturi '18)
- stabilising training *e.g. spectral normalization* (Miyato et al. '18)
- parameterising a Neural ODE (Chen et al. '18)
- formulating invertible neural nets (Berhmann et al. '19)

Lipschitz constant: Definition

Given two metric spaces $(\mathcal{X}, d_{\mathcal{X}})$ and $(\mathcal{Y}, d_{\mathcal{Y}})$, a function $f : \mathcal{X} \to \mathcal{Y}$ is called ***Lipschitz (continuous)*** if there exists $K \geq 0$ such that

$$d_{\mathcal{Y}}(f(x), f(x')) \leq K d_{\mathcal{X}}(x, x') \quad \forall x, x' \in \mathcal{X}$$

Lipschitz constant=smallest K

If $\mathcal{X} = \mathcal{Y}, d_{\mathcal{X}} = d_{\mathcal{Y}}$ and is induced by a norm $||\cdot||$, the above is equivalent to:

$$\sup_{x \neq x' \in \mathcal{X}} \frac{||f(x) - f(x')||}{||x - x'||} \leq K$$

Focus on case where $\mathcal{X}$ is Euclidean and $||x|| = ||x||_p := (\sum_i |x_i|^p)^{\frac{1}{p}}$

Note $||x||_\infty = \max_i |x_i|$

Lipschitz constant: Computation

The following theorem (e.g. Federer 1969) is useful for computing Lip(f):

Theorem 2.1 (Federer, 1969). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be differentiable and Lipschitz continuous under a choice of p-norm $\|\cdot\|_p$. Let $J_f(x)$ denote its total derivative (Jacobian) at x. Then* $\mathrm{Lip}_p(f) = \sup_{x \in \mathbb{R}^n} \|J_f(x)\|_p$ *where $\|J_f(x)\|_p$ is the induced operator norm on $J_f(x)$.*

- Hence if f is a linear map represented by matrix W, then

$$\mathrm{Lip}(f) = ||W||_p := \sup_{x:||x||_p=1} ||Wx||_p = \begin{cases} \sigma_{\max}(W), & \text{if } p = 2 \\ \max_i \sum_j |W_{ij}|, & \text{if } p = \infty \end{cases}$$

- Also using Lip(g∘h) ≤ Lip(g)·Lip(h), we can easily bound Lip(f) where f is a fully-connected/convolutional layer.

- How about self-attention?

Main result 1: Dot-product self-attention is NOT Lipschitz

Input $X \in \mathbb{R}^{N \times D}$ (sequence of N inputs $x_i \in \mathbb{R}^D$).

Single head of (dot-product) self-attention:

each x_i linearly transformed by W^V

$$DP(X) = \boxed{\text{softmax}\left(\frac{XW^Q(XW^K)^\top}{\sqrt{D/H}}\right)X}\boxed{XW^V} \in \mathbb{R}^{N \times \frac{D}{H}}$$

each output is a linear combination of the x_i

where $W^Q, W^K, W^V \in \mathbb{R}^{D \times \frac{D}{H}}$

<u>Theorem 1</u> Dot-product self-attention is **NOT** Lipschitz under $||\cdot||_p \quad \forall p \in [1, \infty]$

<u>Proof outline</u> Some terms of the Jacobian become arbitrarily large when one $x_i = 0$ and $x_{j \neq i}$ grows to infinity. By Thm 2.1, dot-product self-attention is not Lipschitz under $||\cdot||_\infty$. By equivalence of p-norms, it is not Lipschitz under $||\cdot||_p \quad \forall p \in [1, \infty]$

Main result 2: L2 self-attention - a Lipschitz variant

Dot-product self-attention: $W_{ij} \propto \exp\left(\frac{x_i^\top W^Q (x_j^\top W^K)^\top}{\sqrt{D/H}}\right)$

When $x_i = 0,\ W_{ij} = \frac{1}{N}\ \forall j$ => Not Lipschitz

L2 self-attention: $W_{ij} \propto \exp\left(\frac{-||x_i^\top W^Q - x_j^\top W^Q||_2^2}{\sqrt{D/H}}\right)$

We can prove that the **resulting L2 self-attention map is Lipschitz.**

2 Changes: 1. Dot product replaced by negative squared L2 distance.

2. Tied W^Q and W^K (otherwise not Lipschitz).

Main result 2: Lipschitz bounds on Multihead L2 self-attention

Theorem 2 Let each head of L2 self-attention be:

non-linear function of X

$$L2^h(X) := \boxed{W^h(X)} X \boxed{A^h} W^{V,h} \in \mathbb{R}^{N \times \frac{D}{H}}$$

$$A^h := W^{Q,h} W^{Q,h^\top} / \sqrt{D/H}$$

And let L2 multihead self-attention (L2-MHA) be:

$$f(X) = [L2^1(X), \ldots, L2^H(X)] W^O \text{ where } W^O \in \mathbb{R}^{D \times D}$$

Then under $||\cdot||_\infty$, we can obtain an $O(\log N)$ bound on Lip(f):

$$\boxed{\mathrm{Lip}(f) \leq \max_h ||W^{Q,h^\top}||_\infty ||W^{Q,h}||_\infty ||W^{V,h}||_\infty (4 \log N + \frac{1}{\sqrt{D/H}}) ||W^O||_\infty}$$

Under $||\cdot||_2$, we have a looser $O(\sqrt{N} \log N)$ bound.

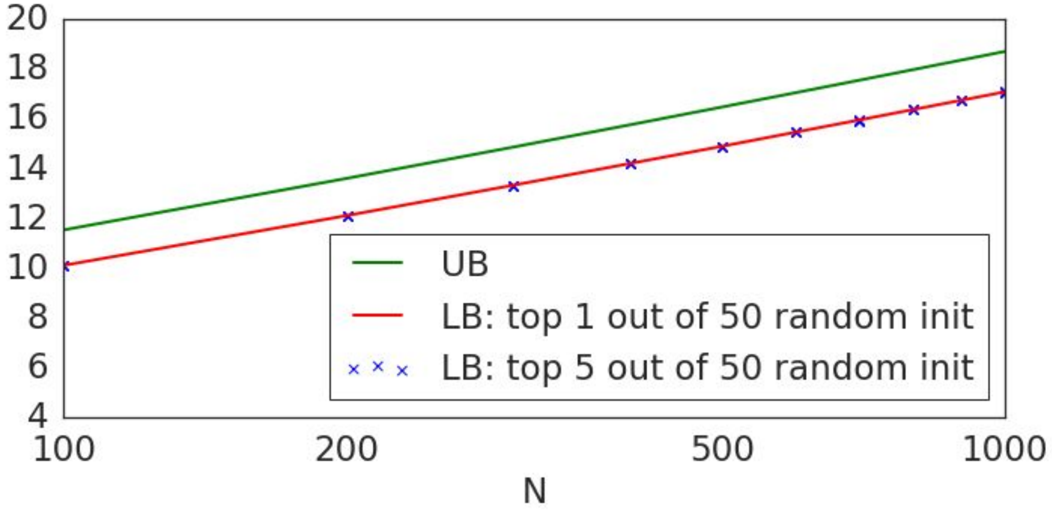
Proof See paper.

Empirical Evidence for Asymptotic Tightness

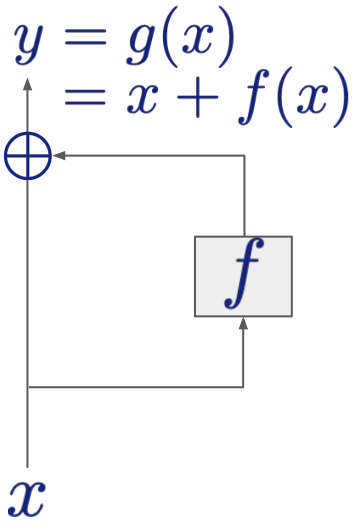
Recall:

Theorem 2.1 (Federer, 1969). *Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be differentiable and Lipschitz continuous under a choice of p-norm $\|\cdot\|_p$. Let $J_f(x)$ denote its total derivative (Jacobian) at x. Then* $\mathrm{Lip}_p(f) = \sup_{x \in \mathbb{R}^n} \|J_f(x)\|_p$ *where $\|J_f(x)\|_p$ is the induced operator norm on $J_f(x)$.*

Hence we can obtain a *lower bound* on $\mathrm{Lip}_p(f)$ by optimising $\|J_f(x)\|_p$ wrt x. For $p = \infty$:



Invertible Residual Networks (Berhmann et al. '19) **& Invertible Self-Attention**



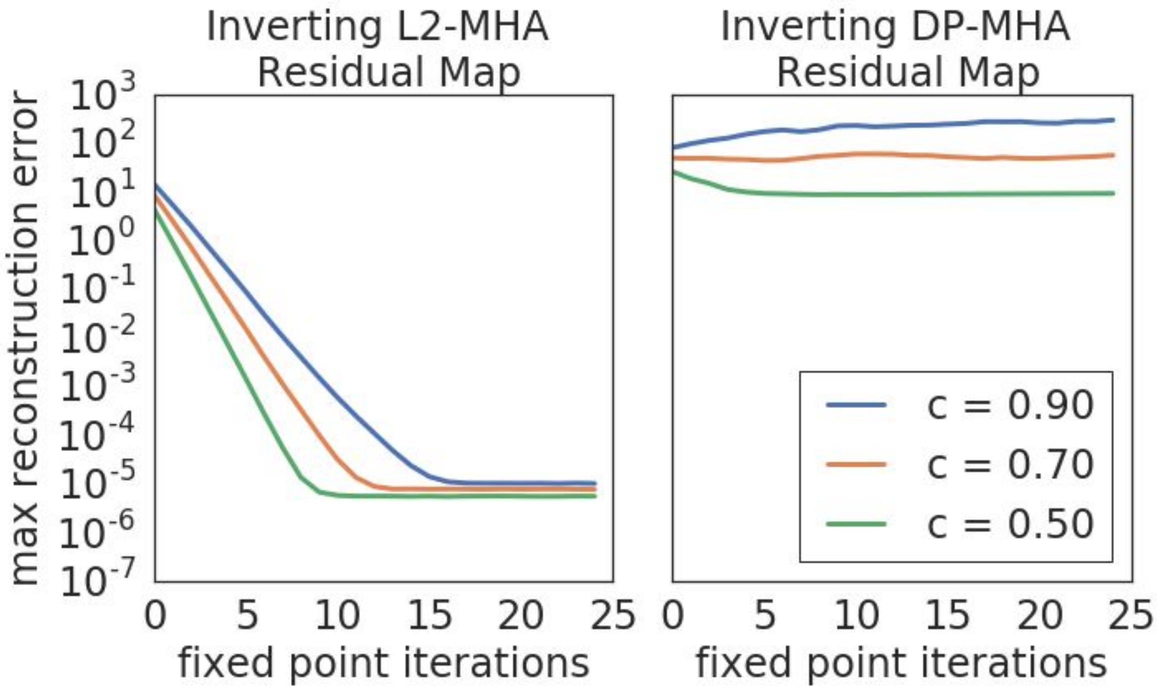
Lemma If **f** has Lipschitz constant less than 1 (i.e. contraction), then the mapping $g : x \mapsto x + f(x)$ is invertible.

Proof The iteration $x \leftarrow y - f(x)$ converges to a unique fixed point (by Banach's fixed point theorem), which is $g^{-1}(y)$.

- So if **f** are convolutions, we can divide **f** by an upper bound on Lip(**f**) to obtain an invertible resnet.
- Similarly if **f** is L2 self-attention, we can divide **f** by the upper bound on Lip(**f**) to obtain **invertible self-attention**.

Invertibility of L2-MHA vs DP-MHA Residual Map

We check numerical invertibility of $g(x) = x + cf(x)$ via fixed point iteration for different values of c.

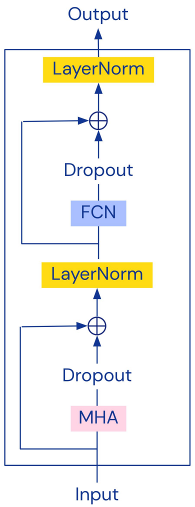


How does expressiveness of invertible self-attention compare to the original self-attention?

To test this, we look at:

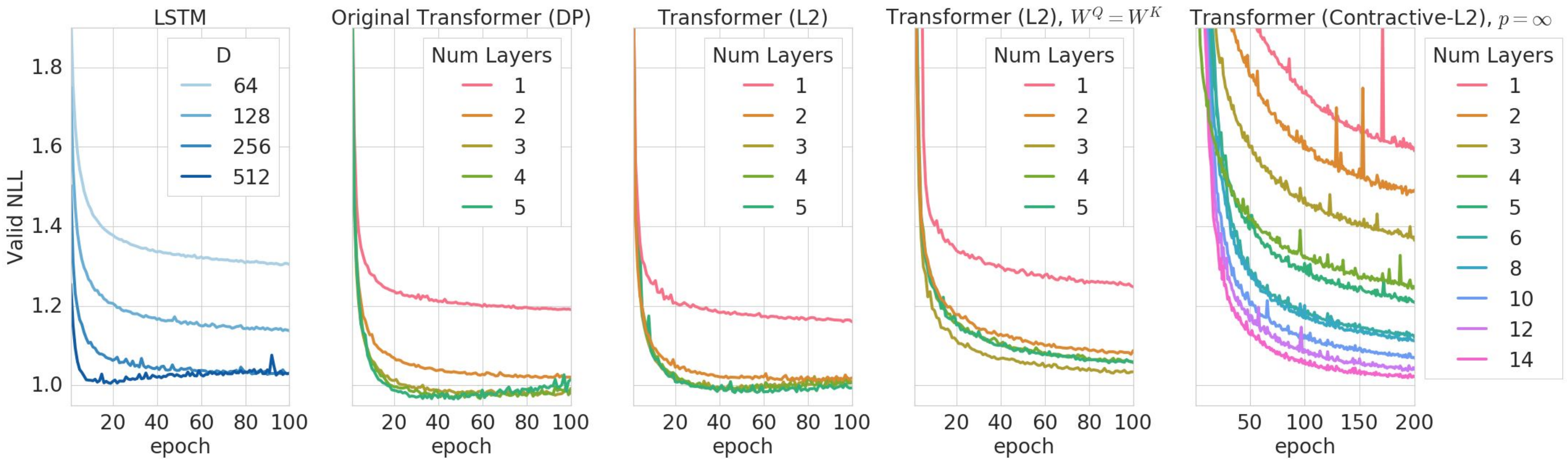
- validation log likelihood of the Transformer on character level language modelling (dataset: Penn Treebank) – **i.e. task: predict next character**
- making one change at a time from DP-MHA to invertible self-attention
    - Recall that the changes for MHA are:

1. Replace dot-product with **negative squared L2 distance**
2. Tie weights W^Q and W^K
3. Post-multiply each head by $A^h := W^{Q,h} {W^{Q,h}}^\top / \sqrt{D/H}$
4. Divide MHA by UB(Lip(MHA))



a single Transformer Block

Validation performance on character-level LM on PTB



Conclusions

- Showed that standard **dot-product multi-head self-attention is NOT Lipschitz**.
- **Proposed L2 self-attention**, an alternative formulation of multi-head self-attention **that is Lipschitz**.
- **Derived upper bounds on the Lipschitz constant of L2 self-attention**, with empirical evidence for asymptotic tightness.
- Showed that **Lipschitz-constrained L2 self-attention can give reasonable predictive performance** on character-level language modelling on PTB, **but does come at cost of expressivity.**

Questions are welcome!