# Deep Prior

Ivan Anokhin, Egor Nuzhin
2018

# Introduction

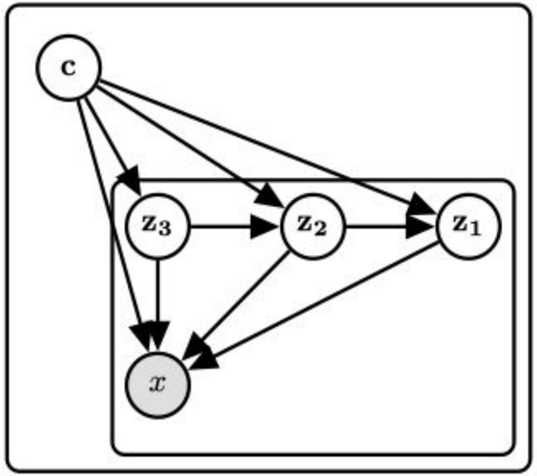
- The machine learning community is well-practised at learning representations of data-points and sequences.
- Efficient learner is one who reuses what they already know to tackle a new problem.
- We considered two approaches to understanding the similarities amongst datasets:
    - Towards a Neural Statistician (TNS)
    - Deep Prior
- Our goal is to discuss and compare described approaches

# TNS. Problem Statement

- Given datasets $D_i$ for $i \in \mathcal{I}$
- Each dataset $D_i = \{x_1, \ldots, x_{k_i}\}$ consists of a number of i.i.d samples from an associated distribution $p_i$
- We assume there is a common underlying generative process such tha $p_i = p(\cdot|c_i)$ drawn from known distribution $p(c)$
- The main goal is to find a posterior generative mode $p(x|D_t)$ - origin distribution of the target dataset

# TNS. Model assumptions

- Context $p(c)$ is normal distribution with zero mean and unit variance
- Context define latent variables $z_i$
- Each latent variable $z_i$ is defined by context c and $z_{i-1}$ and has normal distribution
- Origin predicted random variable x is also normal and defined by contex c and each $z_i$
- Conditional dependencies are embedded through NN
- Model parameters are estimated as MLE




$$\theta^* = arg \max_\theta \prod_i p(D_i)$$

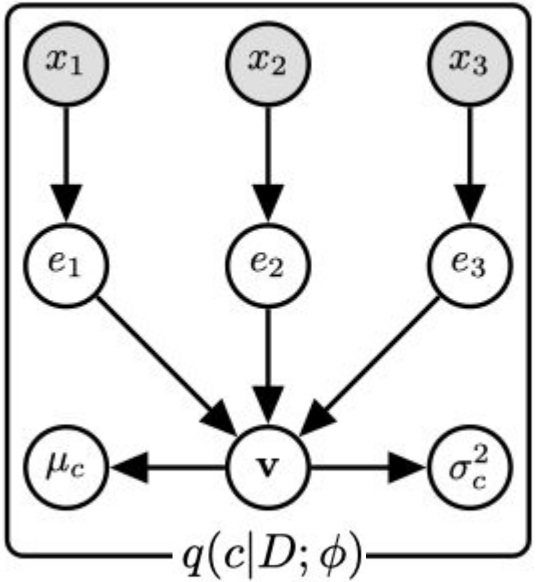
$$p(D) = \int p(c) \prod_{x \in D} \int p(x|c, z_{1:L}; \theta) p(z_L|c; \theta) \prod_{i=1}^{L-1} p(z_i|z_{i+1}, c; \theta)\, dz_{1:L}\, dc$$

# TNS. Variational amortization

- The full approximate posterior factorizes analogously
- Approximate posterior distribution of z (inferiance) $q(z_i|z_{i+1}, x, c; \phi)$ is defined by context c, sample x and $z_i$ .It is normal with conditional dependencies through a NN
- Approximate posterior context $q(c|D; \phi)$ is also normal and consist of three blocks:
    - Instance encoder E
    - Instance pooling (mean, sum, etc,)
    - Post-pooling network
- Approximate posterior is estimated through minimization of KL divergence

$$\phi^* = arg\min KL(q(c, z_{1:L}|D, \phi)||p(c, z_{1:L}|D, \theta^*)$$

$$q(c, z_{1:L}|D; \phi) = q(c|D; \phi) \prod_{x \in D} q(z_L|x, c; \phi) \prod_{i=1}^{L-1} q(z_i|z_{i+1}, x, c; \phi)$$



# TNS. ELBO optimization

$$\mathcal{L}_D = R_D - C_D - L_D \text{ with}$$

$$R_D = \mathbb{E}_{q(c|D;\phi)} \sum_{x \in D} \mathbb{E}_{q(z_{1:L}|c,x;\phi)} \log p(x|z_{1:L}, c; \theta)$$

$$C_D = D_{KL}\left(q(c|D;\phi) \| p(c)\right)$$

$$L_D = \mathbb{E}_{q(c,z_{1:L}|D;\phi)} \Bigg[ \sum_{x \in D} D_{KL}\left(q(z_L|c,x;\phi) \| p(z_L|c;\theta)\right) + \sum_{i=1}^{L-1} D_{KL}\left(q(z_i|z_{i+1}, c, x; \phi) \| p(z_i|z_{i+1}, c; \theta)\right) \Bigg]$$

# TNS. Data generation

- Using full model we can generate new example from test dataset:

$$p(x|D_t) = \int p(c|D_t)p(z_{1:L}|c, D_t)p(x|c, z_{1:L})dcdz_{1:L} \approx \int q(c|D_t) \prod_{x \in D_t} q(z_{1:L}|c, D_t) \prod_i q(z_i|z_{i-1}c, x_t)p(x|c, z_{1:L})dcdz_{1:L}$$

- But we can approximate it just using MLE estimate of context $q(c|D;\phi)$ i.e mean value.

**Algorithm 2** Sampling a dataset of size $k$ conditioned on a dataset of size $m$

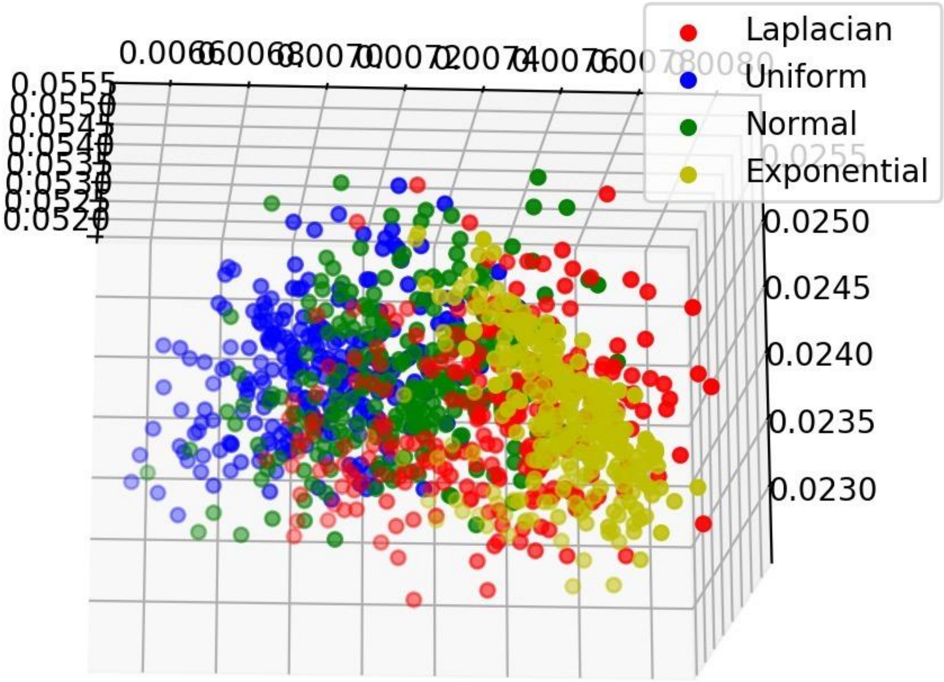
$\mu_c, \sigma_c^2 \leftarrow q(c|x_1, \ldots, x_m; \phi)$ {Calculate approximate posterior over $c$ using statistic network.}
$c \leftarrow \mu_c$ {Set $c$ to be the mean of the approximate posterior.}
**for** $i = 1$ **to** $k$ **do**
  sample $z_{i,L} \sim p(z_L|c;\theta)$
  **for** $j = L - 1$ **to** $1$ **do**
    sample $z_{i,j} \sim p(z_j|z_{i,j+1}, c;\theta)$
  **end for**
  sample $x_i \sim p(x|z_{i,1}, \ldots, z_{i,L}, c;\theta)$
**end for**

# TNS. Additional applications

- Clustering
- Transferring generative models to new datasets
- Selecting representative samples of datasets
- Classifying previously unseen classes

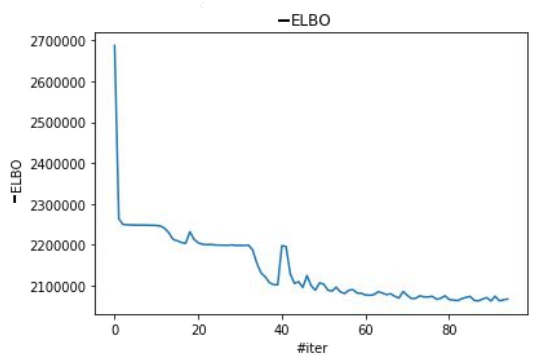
# TNS. Experiments. Clusterization. Untrained

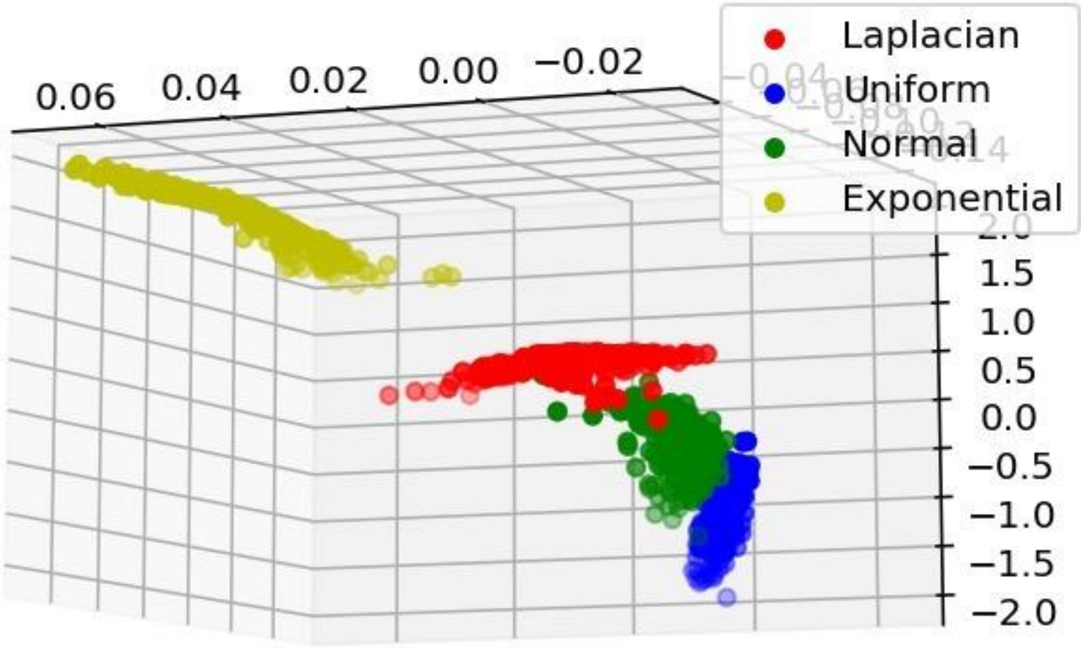
- Each point is the mean of the approximate posterior over the context $q(c|D;\phi)$
- Silhouette score: 0.11



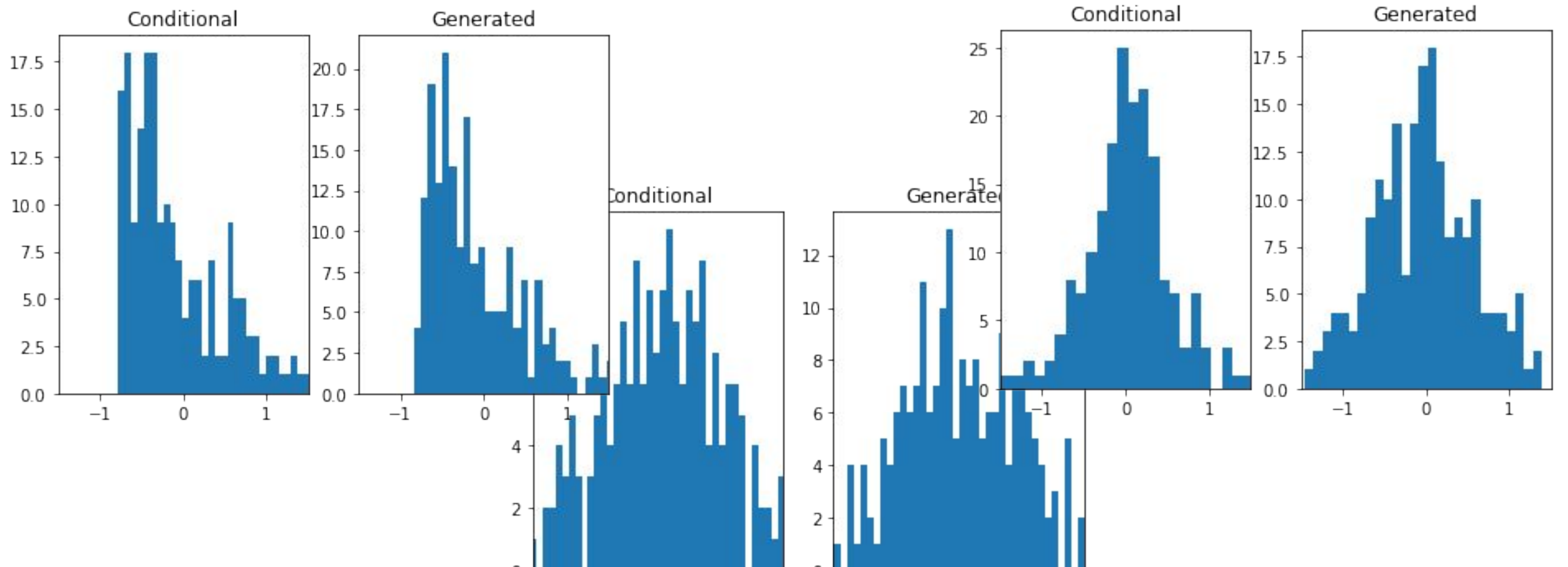
# TNS. Experiments. Clusterization. Trained

- Silhouette score: 0.57

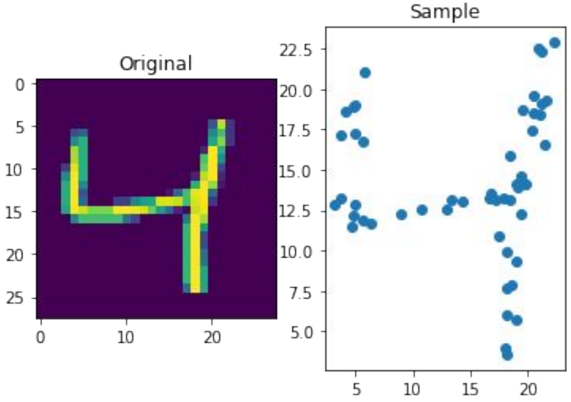


# TNS. Experiments. Generation



# TNS. Image generation



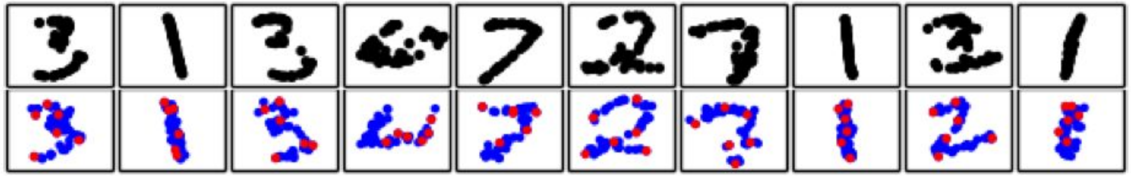



Figure 4: Conditioned samples from spatial MNIST data. Blue and red digits are the input sets, black digits above correspond to samples given the input. Red points correspond to a 6-sample summary of the dataset

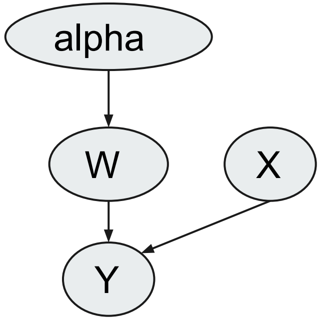
# Deep prior model

The main goal is Learning the prior distribution over neural network parameters.

Observe dataset, generate W, make prediction

How? Maximize p(D| alpha)

ELBO approximation:

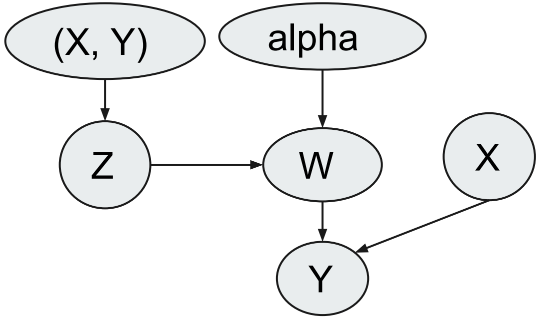


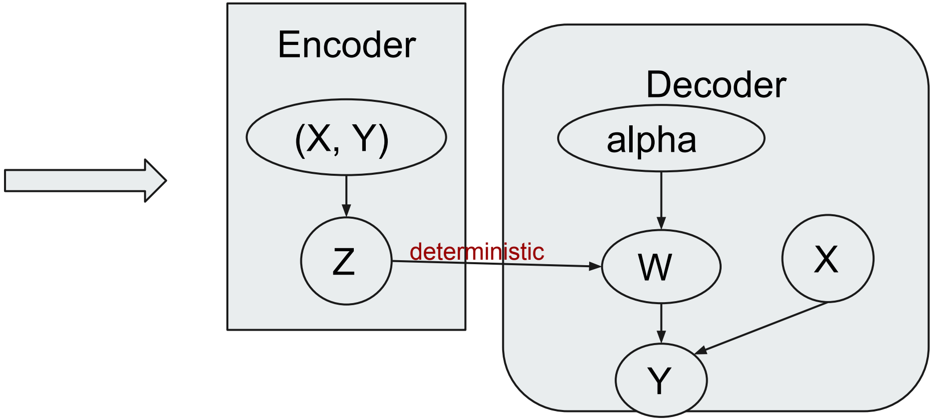

True distributions:

$$p(z_j) = \mathcal{N}(0, I)$$
$$p(z_j, w_j|\alpha) = p(z_j)\delta_{h_\alpha(z_j)-w_j}$$
$$p(z_j, w_j|\alpha, S_j) = p(z_j|\alpha, S_j)\delta_{h_\alpha(z_j)-w_j}$$

Approximation:

$$q(\{w_j, z_j\}_{j=1}^N) = \prod_j q_{\theta_j}(z_j|S_j)\delta_{h_\alpha(z_j)-w_j}$$



# Deep prior, ELBO

$$
\begin{aligned}
\ln p(\mathcal{D}) &\geq \mathop{\mathbb{E}}_{q(\{w_j, z_j\}_{j=1}^{N})} \sum_{j=1}^{N} \sum_{i=1}^{n_j} \ln p(y_{ij} | x_{ij}, w_j) - \mathrm{KL}\left(q \parallel p\right), \\
&= \sum_{j=1}^{N} \mathop{\mathbb{E}}_{q_{\theta_j}(z_j|S_j)} \sum_{i=1}^{n_j} \ln p(y_{ij} | x_{ij}, h_\alpha(z_j)) - \sum_{j} \mathop{\mathbb{E}}_{q(w_j, z_j)} \ln \frac{q_{\theta_j}(z_j|S_j)}{p(z_j)} \frac{\delta_{h_\alpha(z_j)-w_j}}{\delta_{h_\alpha(z_j)-w_j}} \\
&= \sum_{j=1}^{N} \mathop{\mathbb{E}}_{q_{\theta_j}(z_j|S_j)} \sum_{i=1}^{n_j} \ln p(y_{ij} | x_{ij}, h_\alpha(z_j)) - \sum_{j} \mathrm{KL}\left(q_{\theta_j}(z_j|S_j) \parallel p(z_j)\right)
\end{aligned}
$$

$z_j$ - latent variable that describes dataset $j$

$S_j$ - dataset $j$

# Deep prior model

The task now consists in jointly learning a function $h_\alpha$ common to all tasks and a posterior distribution $p(z_j|\alpha, S_j)$ for each task. At inference time, predictions are performed by marginalizing $z$ i.e.: $p(y|x, \mathcal{D}) = \mathop{\mathbb{E}}_{z \sim p(z_j|\alpha, S_j)} p(y|x, h_\alpha(z))$

Since we no longer need to explicitly calculate the KL on the space of $w$, we can simplify the likelihood function to the following $p(y_{ij}|x_{ij}, z_j, \alpha)$, which can be a deep network, parameterized by $\alpha$ taking both $x_{ij}$ and $z_j$ as inputs. This contrasts with the previous formulation where $h_\alpha(z)$ produces all the weights of a network, yielding a really high dimensional representation and slow training.

Learning $p(w_j|\alpha, S_j)$

We have latent variable $z$ and a deterministic function projecting the noise $z$ to the space of neural nets weights $w$ i.e. $w = h_\alpha(z)$

we have $p(w|\alpha) = \int_z p(z)p(w|z, \alpha)dz = \int_z p(z)\delta_{h_\alpha(z)-w}dz$

As we can not marginolized directly expression above for general $h_\alpha$ we cosider two latent variable $z, w$ at inference time.

# Encoder

# Normalizing Flows (Planar flows)

Deep prior

Sample from simple distribution: $z_0 \in \mathbb{R}^D$

$$z_0 \sim q_0(z\,|\,x) = \mathcal{N}(z\,|\,\mu(x), \mathrm{diag}(\sigma^2(x)))$$

Apply sequence of invertible transformations: $f_k : \mathbb{R}^D \mapsto \mathbb{R}^D$

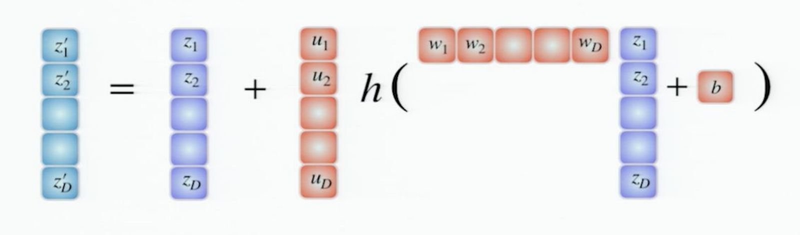
$$z_K = f_K \circ \ldots \circ f_2 \circ f_1(z_0) \qquad z_0 \to z_1 \to \ldots \to z_K$$

For each transformation: $z_k = f_k(z_{k-1})$

$$q_k(z_k) = q_{k-1}(z_{k-1}) \left| \det \frac{\partial f_k^{-1}}{\partial z_k} \right| = q_{k-1}(z_{k-1}) \left| \det \frac{\partial f_k}{\partial z_{k-1}} \right|^{-1}$$

Planar flow:

$$\mathbf{f}_t(\mathbf{z}_{t-1}) = \mathbf{z}_{t-1} + \mathbf{u}h(\mathbf{w}^T\mathbf{z}_{t-1} + b)$$



# Inverse Autoregressive Flow (IAF)

IAF: $z'_i = \mu_i(z_{1:i-1}) + \sigma_i(z_{1:i-1})\, z_i$

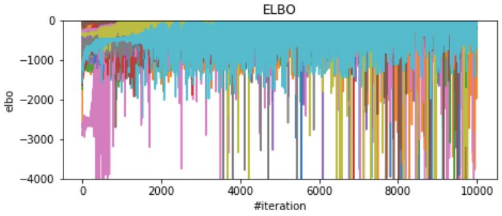
Stable version:

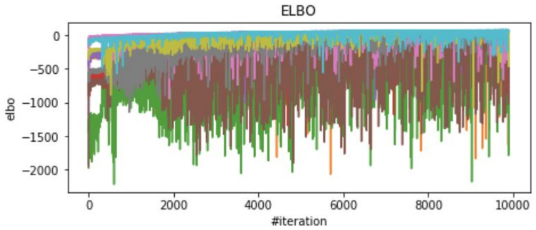
$[\boldsymbol{\mu}, \boldsymbol{\sigma}, \mathbf{h}] \leftarrow \texttt{EncoderNN}(\mathbf{x}; \boldsymbol{\theta})$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$
$\mathbf{z} \leftarrow \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} + \boldsymbol{\mu}$
$l \leftarrow -\texttt{sum}(\log \boldsymbol{\sigma} + \frac{1}{2}\boldsymbol{\epsilon}^2 + \frac{1}{2}\log(2\pi))$
**for** $t \leftarrow 1$ **to** $T$ **do**
  $[\mathbf{m}, \mathbf{s}] \leftarrow \texttt{AutoregressiveNN}[t](\mathbf{z}, \mathbf{h}; \boldsymbol{\theta})$
  $\boldsymbol{\sigma} \leftarrow \texttt{sigmoid}(\mathbf{s})$
  $\mathbf{z} \leftarrow \boldsymbol{\sigma} \odot \mathbf{z} + (1 - \boldsymbol{\sigma}) \odot \mathbf{m}$
  $l \leftarrow l - \texttt{sum}(\log \boldsymbol{\sigma})$
**end**

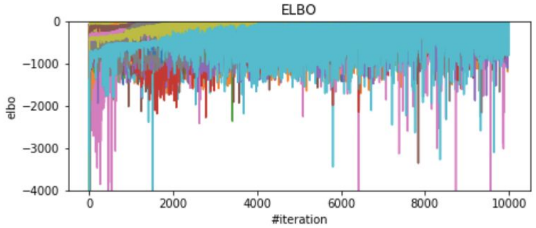
$$\ln q_K(z_k) = \ln q(z_0|x) - \sum_t \ln \det \frac{dz_t}{dz_{t-1}} = \ln q(z_0|x) - \sum_t \ln \det \sigma_t$$

# Different flows comparison

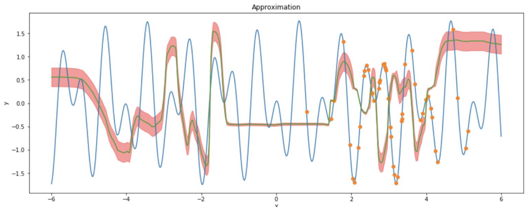
| Standart Reparemetrisation trick | Planar flow | IAF |
|---|---|---|

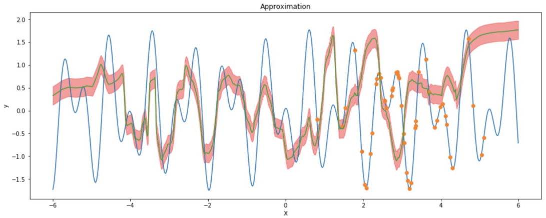


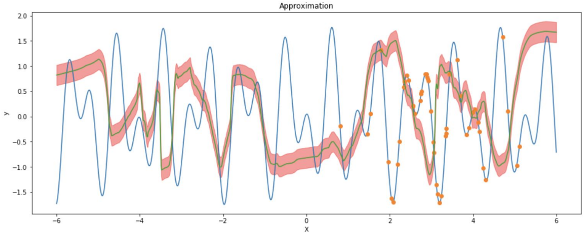




Test

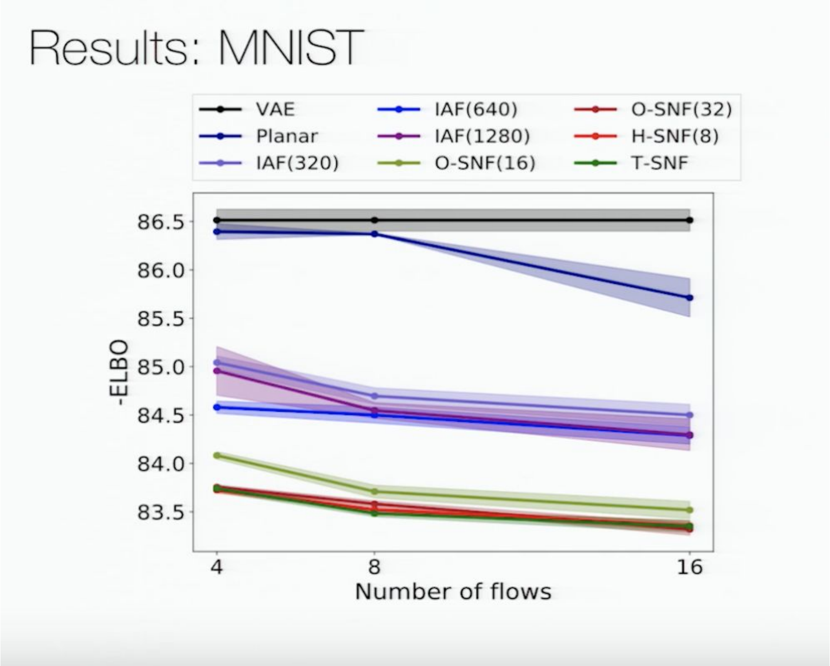



| | | | | |
|---|---|---|---|---|
| MSE | Test | 2.689 | 2.484 | 2.315 |
| | Train | 4.851 | 4.093 | 4.165 |

# Different flows comparison




the picture is from Rianne van den Berg et al. Sylvester Normalizing Flows for Variational Inference (2018)

# Results

**Deep prior**

Result from the deep prior paper:

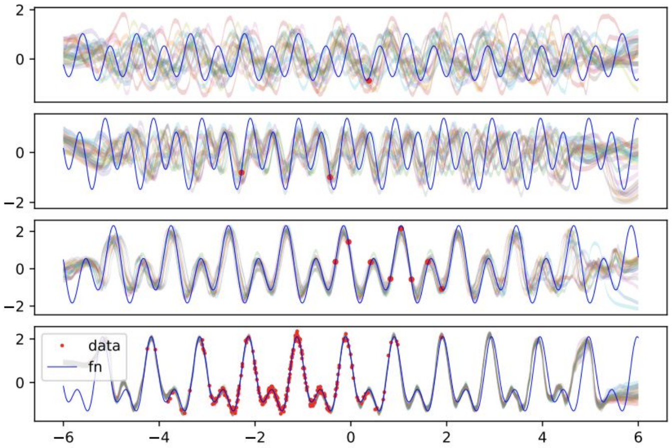
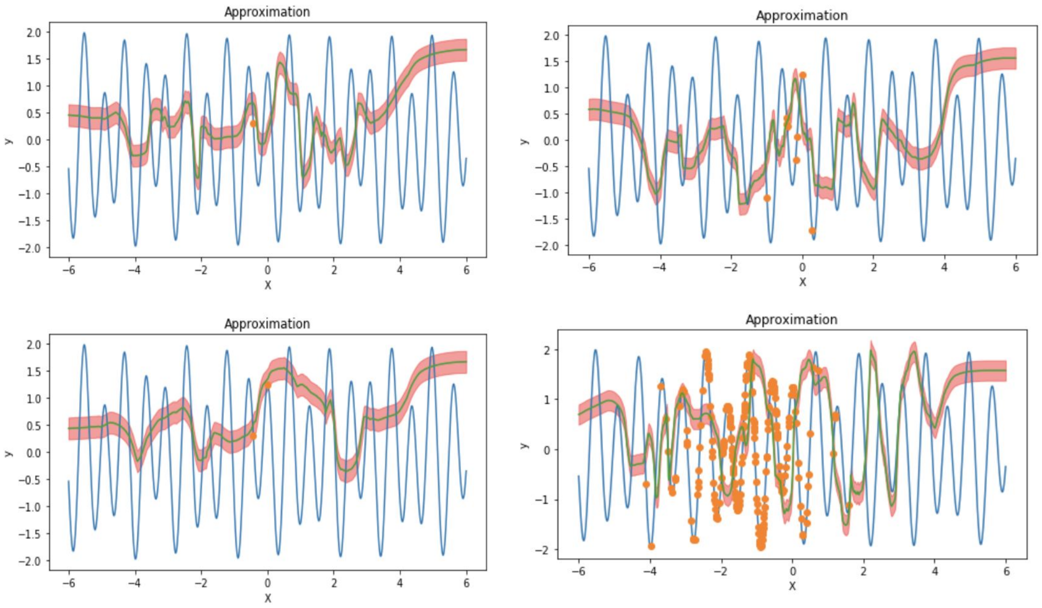



Figure 1: Preview of a few tasks (blue line) with increasing amount of training samples (red dots). Samples from the posterior distribution are shown in semi-transparent colors. The width of each samples is two standard deviations (provided by the predicted heteroskedastic noise).

Our results of the experiment (IAF):



# Deep prior vs TNS

- Both models try to extract knowledge from datasets
- Deep prior developed for prediction (discriminative model), TNS for generation (generative model)
- Deep prior uses one latent variable w, TNS instead uses two (c and z)
- Deep prior model latent variable w as arbitrary function (NN) of unit Normal distribution but TNS epxploit two random variables normal distributions with complex dependences of mean and variance
- Deep prior model distribution of weight of predictive normal model given dataset, TNS models distribution of datasets, given a test dataset and distribution of samples in this dataset

# References


1. NN architectures:
    a. Towards a Neural Statistician (2016)
    b. Deep Prior (2017)
2. Normalizing flows:
    a. Variational Inference with Normalizing Flows (2016)
    b. Improved Variational Inference with Inverse Autoregressive Flow (2017)
    c. Sylvester Normalizing Flows for Variational Inference (2018)

# Thank you

Ivan.Anokhin@skoltech.ru
Egor.Nuzhin@skoltech.ru