Imperial College London

Department of Mathematics

Title of the Thesis

FIRSTNAME LASTNAME

CID: 01234567

Supervised by SUPERVISORNAME and COSUPERVISORNAME

1 May 2022

Submitted in partial fulfilment of the requirements for the MSc in Statistics of Imperial College London

The work contained in this thesis is my own work unless otherwise stated.

Signed: STUDENT'S NAME Date: DATE

Abstract

ABSTRACT GOES HERE

Acknowledgements

ANY ACKNOWLEDGEMENTS GO HERE

Contents

Notation

$\boldsymbol{X}$ is a matrix

y is a vector

Abbreviations

DAIN Deep Adaptive Input Normalization

RDAIN Robust Deep Adaptive Input Normalization

EDAIN Extended Deep Adaptive Input Normalization

EDAIN-KL Extended Deep Adaptive Input Normalization, optimised with Kullback–Leibler divergence

BIN Bilinear Input Normalization

pdf probability density function

KL-divergence Kullbeck-Leibler divergence

1 Introduction

The introduction section goes here[1].

[1] Tip: write this section last.

2 Background

TODO: introduction to this chapter

2.1 Deep learning

TODO: write details

2.1.1 Sequence models

2.2 Data preprocessing

Origin pa

Koval (2018) does some data preprocessing for neural networks, and Nawi et al. (2013) also investigate the effect of data preprocessing on neural network. Also looked at effect on classification performance by Singh and Singh (2020). Moreover, been studied as early as 1997 by (Sola and Sevilla, 1997).

2.2.1 Static distribution transformations

2.2.2 Adaptive distribution transformations

DAIN

The Deep Adaptive Input Normalization (DAIN) method, proposed by Passalis et al. (2019).

RDAIN

We have Passalis et al. (2021)

BiN

We have Tran et al. (2021)

2.3 Normalizing flows

TODO: write details

3 Methods

TODO: introduction to this chapter

3.1 EDAIN

My first contribution is the Extended Deep Adaptive Input Normalization (EDAIN) layer. This adaptive preprocessing layer is inspired by the likes of DAIN_REF and BIN_REF, but unlike the aforementioned methods, the EDAIN layer also supports normalizing the data in a *batch-agnostic* fashion, whereas the DAIN, Robust Deep Adaptive Input Normalization (RDAIN) and Bilinear Input Normalization (BIN) layers are all *batch-aware*. Additionally, the EDAIN layer extends the other layers with two new operations: An outlier removal operation that is designed to reduce the negative impact of high-tail observations, as well as a power-transform operation that is designed to transform non-normal data to be more normal.

3.1.1 Notation

Let $\{\mathbf{X}^{(i)} \in \mathbb{R}^{d \times T}; i = 1, \ldots, N\}$ denote a set of N multivariate time-series, each composed of T d-dimensional feature vectors. We also let $\mathbf{x}_t^{(i)} \in \mathbb{R}^d$, where $t = 1, \ldots, T$, denote the tth feature vector at time-step t in the time-series.

3.1.2 Architecture

An overview of the layer's architecture is shown in figure TODO_FIG. First, outlier removal. Then scale. Then shift. Then power transform. This order because TODO.

TODO: explain comparison to DAIN, RDAIN and BIN, and how not batch-aware, but rather aims to learn characteristics of global distribution, as it's the global distribution that might be very irregular...

Outlier removal

Handling outliers and extreme values in the dataset can increase predictive performance if done correctly (citation needed). Two common ways of doing this are omission and winsorization (Nyitrai and Virág, 2019). With the former, observations that are deemed to be extreme are simply removed during training. With the latter, all the data is still

used, but observations lying outside a certain number of standard deviation from the mean, or below or above certain percentiles, are *clamped down* to be closer to the mean or median of the data. For example, if winsorizing data using 3 standard deviation, all values less than $\mu - 3\sigma$ are set to be exactly $\mu - 3\sigma$. Similarly, the values above $\mu + 3\sigma$ are *clamped* to this value. Winsorization can also be done using percentiles, where common boundaries are the first and fifth percentiles Nyitrai and Virág (2019). However, the type of winsorization, as well as the number of standard deviation or percentiles to use, might depend on the dataset. Additionally, it might not be necessary to winsorize the data at all if the outliers turn out to not negatively affect performance. All this introduces more hyperparameters to tune during modelling. The outlier removal presented here aims to automatically both determine whether winsorization is necessary for a particular feature, and determine the threshold at which to apply winsorization.

For input vector $\mathbf{x} \in \mathbb{R}^d$, the adaptive outlier removal operation is defined as:

$$\boldsymbol{\alpha}' \odot \underbrace{\left(\boldsymbol{\beta}' \odot \tanh\left\{(\mathbf{x} - \hat{\boldsymbol{\mu}}) \oslash \boldsymbol{\beta}'\right\} + \hat{\boldsymbol{\mu}}\right)}_{\text{smooth adaptive centred winsorization}} + \underbrace{\left(1 - \boldsymbol{\alpha}'\right) \odot \mathbf{x}}_{\text{residual connection}}, \tag{3.1}$$

where $\odot$ is the element-wise multiplication, $\oslash$ is element-wise division, $\boldsymbol{\alpha}' \in [0, 1]^d$ is a parameter controlling how much winsorization to apply to each feature, and $\boldsymbol{\beta}' \in [\beta_{\min}, \infty)^d$ controls the winsorization threshold for each feature, that is, the maximum absolute value of the output, thus controlling the range of the output. The effect of the two parameters is illustrated in fig. 3.1. The $\hat{\boldsymbol{\mu}}$ parameter is an estimate of the mean of the data, and is used to ensure the winsorization is centred. When setting the EDAIN layer in *batch-aware* mode, it is simply the mean of the batch:

$$\hat{\mu}_k = \frac{1}{|\mathcal{B}|T} \sum_{i \in \mathcal{B}} \sum_{t=1}^{T} x_{j,k}^{(i)}, \tag{3.2}$$

while if using the *batch-agnostic* mode, it is iteratively updated using a cumulative moving average estimate at each forward pass of the layer. This is to better approximate the global mean of the data. The unknown parameters of the model are $\boldsymbol{\alpha} \in \mathbb{R}^d$ and $\boldsymbol{\beta} \in \mathbb{R}^d$, and they are transformed into the constrained parameters $\boldsymbol{\alpha}'$ and $\boldsymbol{\beta}'$, as used in eq. (3.1) through the following element-wise mappings:

$$\boldsymbol{\alpha}' = \frac{e^{\boldsymbol{\alpha}}}{1 + e^{\boldsymbol{\alpha}}} \qquad \boldsymbol{\beta}' = \beta_{\min} + e^{\boldsymbol{\beta}}. \tag{3.3}$$

For ease of notation, we let $\mathbf{W}_1 = (\boldsymbol{\alpha}, \boldsymbol{\beta})$ denote the $2d$ unknown parameters that are optimised for the adaptive outlier removal layer.

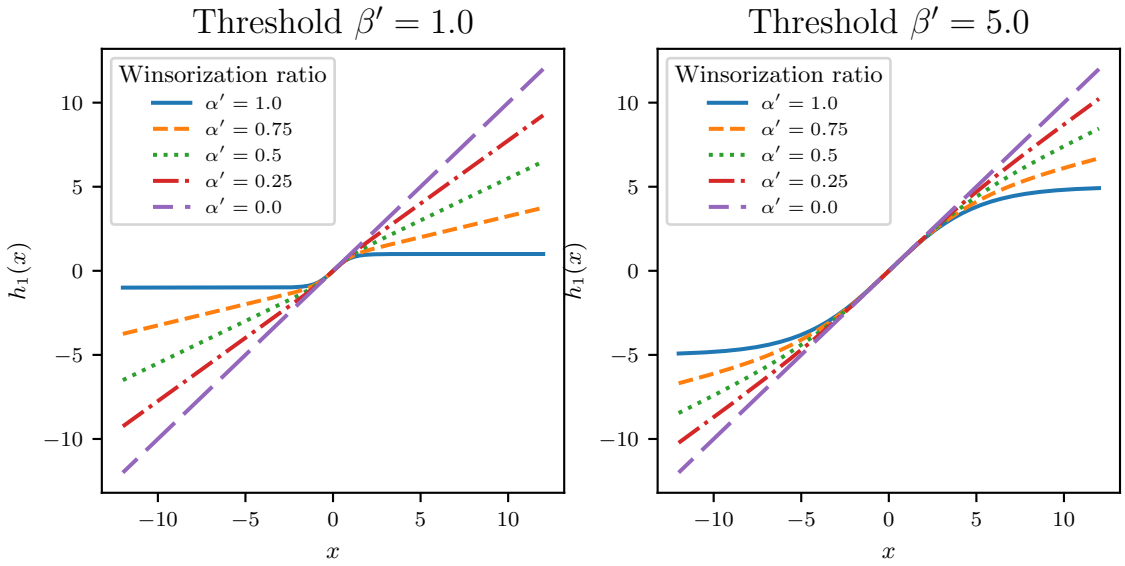

Figure 3.1: Plot of the adaptive outlier removal operation for different combinations of parameter values for α' and β'.

Scale and shift

The adaptive shift and scale layer, combined, simply performs the operation

$$(\mathbf{x} \oplus \boldsymbol{\gamma}) \odot \boldsymbol{\lambda}, \tag{3.4}$$

with input $\mathbf{x}$ and unknown parameters $\boldsymbol{\gamma} \in \mathbb{R}^d$ and $\boldsymbol{\lambda} \in (0, \infty)^d$ when the EDAIN is set to batch-agnostic mode. This makes the scale-and-shift layer a generalised version of Z-score scaling, or standard scaling, as setting

$$\boldsymbol{\gamma} := -\frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \mathbf{x}_t^{(i)} \tag{3.5}$$

and

$$\boldsymbol{\lambda} := \left(\frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(\mathbf{x}_t^{(i)} \oplus \boldsymbol{\gamma} \right)^2 \right)^{-1} \tag{3.6}$$

makes the operation in eq. (3.4) equivalent to Z-score scaling. This *batch-agnostic* mode is useful if the distribution is similar across batches and constitute a global unimodal distribution that should be centred.

However, some datasets might have multiple modes arising from significantly different data generation mechanisms. Attempting to scale and shift each batch to a global mean and standard deviation might hurt performance in such cases. Instead, CITE_AUTHOR_DAIN

propose basing the scale and shift on a *summary representation* of the current batch, allowing each batch to be normalized according the specific mode that batch of data might have come from. This gives

$$(\mathbf{x} \oplus [\boldsymbol{\gamma} \odot \mu_{\mathbf{x}}]) \odot [\boldsymbol{\lambda} \odot \sigma_{\mathbf{x}}], \tag{3.7}$$

where the summary representations $\sigma_{\mathbf{x}}$ and $\mu_{\mathbf{x}}$ are computed through reduction of the temporal dimension for each observation:

$$\mu_{\mathbf{x}}^{(i)} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{x}_t^{(i)} \in \mathbb{R}^d \tag{3.8}$$

$$\sigma_{\mathbf{x}}^{(i)} = \sqrt{\frac{1}{T} \sum_{t=1}^{T} \left(\mathbf{x}_t^{(i)} - \mu_{\mathbf{x}}^{(i)} \right)^2} \in \mathbb{R}^d. \tag{3.9}$$

With this mode, it is difficult for the layer to generalise Z-score scaling, but it becomes more able to normalize in a *mode-aware* fashion.

Power transform

Many real-world datasets exhibit significant skewness, which is often treated using power transformations (citation needed). The most common transformation is the Box-Cox transformation, but this is only valid for positive values, so it is not applicable to most real-world datasets TODO_CITE_BOX_COX. An alternative is a transformation proposed by TODO_CITE_YEO_Johnson, who proposed to following transformation:

$$f_{\text{YJ}}(x) = \begin{cases} \frac{(x+1)^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0, x \geq 0; \\ \log(x+1), & \text{if } \lambda = 0, x \geq 0; \\ \frac{(1-x)^{2-\lambda} - 1}{\lambda - 2}, & \text{if } \lambda \neq 2, x < 0; \\ -\log(1-x), & \text{if } \lambda = 2, x < 0. \end{cases} \tag{3.10}$$

Like the Box-Cox transformation, transformation f_{YJ} only has one unknown parameter, λ, but it works for any $x \in \mathbb{R}$, not just positive values.

The power transform layer simply applies the transformation in eq. (3.10) along each dimension of the input, that is for each $i = 1, \ldots, N$ and $t = 1, \ldots, T$,

$$\left[\text{power\_transform} \left(\mathbf{x}_t^{(i)} \right) \right]_j = f_{\text{YJ}}(x_{t,j}^{(i)}), \quad j = 1, \ldots, d. \tag{3.11}$$

The unknown parameters is the vector $\boldsymbol{\lambda} \in \mathbb{R}^d$.

3.1.3 Optimisation through back-propagation

Can simply feed values forward through the 4 layers, as shown in TODO_FIG. Additionally, weights are updated through standard gradient descent simultaneously while training the model

$$\Delta(\mathbf{W}_\alpha, \beta) = -\eta \left(\eta_{pt} \frac{\partial \mathcal{L}}{\partial \mathbf{W}_a}, \dots \right) \tag{3.12}$$

Similarly to DAIN_REF, convergence is unstable if use same learning rate for RNN model and preprocessing layer weights, so seperate learning rates for each parameter, and these are set using hyperparameter tuning search, described later in section TODO.

3.2 EDAIN-KL

TODO: introduction something something alterative inspired by normalizing flow, and mention bijectors.

3.2.1 Architecture

The Extended Deep Adaptive Input Normalization, optimised with Kullback–Leibler divergence (EDAIN-KL) layer has a very similar architecture to the EDAIN layer, described in section 3.1, but the outlier removal transformation has been simplified to ensure its inverse is tractable. Additionally, the scale and shift can no longer be in batch-aware mode, as this would make the inverse intractable. The EDAIN-KL transformations are:

$$\text{(Outlier removal)} \qquad h_1\left(\mathbf{x}_t^{(i)}\right) = \boldsymbol{\beta}' \odot \tanh\left\{ (\mathbf{x}_t^{(i)} - \hat{\boldsymbol{\mu}}) \oslash \boldsymbol{\beta}' \right\} + \hat{\boldsymbol{\mu}} \tag{3.13}$$

$$\text{(shift)} \qquad h_2\left(\mathbf{x}_t^{(i)}\right) = \mathbf{x}_t^{(i)} \oplus \boldsymbol{\gamma} \tag{3.14}$$

$$\text{(scale)} \qquad h_3\left(\mathbf{x}_t^{(i)}\right) = \mathbf{x}_t^{(i)} \odot \boldsymbol{\lambda} \tag{3.15}$$

$$\text{(power transform)} \qquad h_4\left(\mathbf{x}_t^{(i)}\right) = \left[f_{\text{YJ}}^{\lambda_1}\left(x_{t,0}^{(i)}\right) \quad f_{\text{YJ}}^{\lambda_2}\left(x_{t,1}^{(i)}\right) \quad \cdots \quad f_{\text{YJ}}^{\lambda_d}\left(x_{t,d}^{(i)}\right) \right], \tag{3.16}$$

where $f_{\text{YJ}}^{\lambda_i}(\cdot)$ is defined in eq. (3.10).

3.2.2 Optimisation through Kullback-Leibler divergence

This optimisation method is inspired by normalizing flow, of which Kobyzev et al. provide a great overview of.

Brief background on normalizing flow

Consider a random variable $\mathbf{Z} \in \mathbb{R}^d$ with a known and analytic expression for the probability density function (pdf) $p_{\mathbf{z}} : \mathbb{R}^d \to \mathbb{R}$, which we call the *base distribution*. The idea behind normalizing flows is defining a arbitrarily complicated parametrised bijector $\mathbf{g}_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$—an invertible function—and transforming the simple base distribution into a new arbitrarily complicated probability distribution: $\mathbf{Y} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{Z})$. The pdf of the transformed distribution can then be computed using the change of variable formula (Kobyzev et al., 2021):

$$\begin{aligned} p_{\mathbf{Y}}(\mathbf{y}) &= p_{\mathbf{Z}}(\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\mathbf{y})) \cdot \left|\det \mathbf{J}_{\mathbf{Y}\to\mathbf{Z}}(\mathbf{y})\right| \\ &= p_{\mathbf{Z}}(\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\mathbf{y})) \cdot \left|\det \mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}\left(\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\mathbf{y})\right)\right|^{-1}, \end{aligned} \tag{3.17}$$

where $\mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}$ is the Jacobian matrix for the forward mapping $\mathbf{Y} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{Z})$. Taking logs on both sides, it follows that

$$\log p_{\mathbf{Y}}(\mathbf{y}) = \log p_{\mathbf{Z}}(\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\mathbf{y})) - \log \left|\det \mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}\left(\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\mathbf{y})\right)\right|. \tag{3.18}$$

One common application of normalizing flows is density estimation (Kobyzev et al., 2021); Given a dataset $\mathcal{D} = \{\mathbf{y}^{(i)}\}_{i=1}^N$ with samples from some unknown complicated distribution, we want to estimate its unknown pdf, $p_{\mathcal{D}}$. This can be done with likelihood-based estimation, where we assume the data points come from parametrised distribution $\mathbf{Y} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{Z})$ and optimise $\boldsymbol{\theta}$ to maximise the data log-likelihood:

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p_{\mathbf{Y}}(\mathbf{y}^{(i)}|\boldsymbol{\theta}) \tag{3.19}$$

$$\overset{eq.\ (3.18)}{=} \sum_{i=1}^{N} \log p_{\mathbf{Z}}\left(\mathbf{g}_{\boldsymbol{\theta}}^{-1}\left(\mathbf{y}^{(i)}\right)\right) - \log \left|\det \mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}\left(\mathbf{g}_{\boldsymbol{\theta}}^{-1}\left(\mathbf{y}^{(i)}\right)\right)\right|. \tag{3.20}$$

This is equivalent to minimising the Kullbeck-Leibler divergence (KL-divergence) be-

tween the empirical distribution $\mathcal{D}$ and the transformed distribution $\mathbf{Y} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{Z})$:

$$\arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p_{\mathbf{Y}}\left(\mathbf{y}^{(i)}\middle|\boldsymbol{\theta}\right) \tag{3.21}$$

$$= \frac{1}{N}\sum_{i=1}^{N} \log p_{\mathcal{D}}\left(\mathbf{y}^{(i)}\right) + \arg\max_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N} \log p_{\mathbf{Y}}\left(\mathbf{y}^{(i)}\middle|\boldsymbol{\theta}\right) \tag{3.22}$$

$$= \arg\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N} \log p_{\mathcal{D}}\left(\mathbf{y}^{(i)}\right) - \frac{1}{N}\sum_{i=1}^{N} \log p_{\mathbf{Y}}\left(\mathbf{y}^{(i)}\middle|\boldsymbol{\theta}\right) \tag{3.23}$$

$$= \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} p_{\mathcal{D}}\left(\mathbf{y}^{(i)}\right) \log p_{\mathcal{D}}\left(\mathbf{y}^{(i)}\right) \tag{3.24}$$

$$- \sum_{i=1}^{N} p_{\mathcal{D}}\left(\mathbf{y}^{(i)}\right) \log p_{\mathbf{Y}}\left(\mathbf{y}^{(i)}\middle|\boldsymbol{\theta}\right) \tag{3.25}$$

$$= \arg\min_{\boldsymbol{\theta}} D_{\mathrm{KL}}\left(\mathcal{D} \,||\, (\mathbf{Y} \mid \boldsymbol{\theta})\right). \tag{3.26}$$

When training an normalizing flow model, we adjust $\boldsymbol{\theta}$ to minimize the above KL-divergence. This requires computing all the terms in eq. (3.20), which requires analytic and differentiable expressions for the inverse transformation $\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\cdot)$, the pdf of the base distribution $p_{\mathbf{Z}}(\cdot)$ and the log determinant of the Jacobian matrix for $\mathbf{g}_{\boldsymbol{\theta}}^{-1}$, $\log|\det \mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}|$. Using a result stated in Kobyzev et al., the following can be shown:

Theorem 3.2.1. Let $\mathbf{g}_1, \ldots, \mathbf{g}_n : \mathbb{R}^d \to \mathbb{R}^d$ all be bijective functions, and consider the composition of these functions, $\mathbf{g} = \mathbf{g}_n \circ \mathbf{g}_{n-1} \cdots \circ \mathbf{g}_1$. Then, $\mathbf{g}$ is a bijective function with inverse

$$\mathbf{g}^{-1} = \mathbf{g}_1^{-1} \circ \cdots \circ \mathbf{g}_{n-1}^{-1} \circ \mathbf{g}_n^{-1}, \tag{3.27}$$

and the log of the absolute value of the determinant of the Jacobian is given by

$$\log\left|\det \mathbf{J}_{\mathbf{g}^{-1}}(\cdot)\right| = \sum_{i=1}^{N} \log\left|\det \mathbf{J}_{\mathbf{g}_i^{-1}}(\cdot)\right|. \tag{3.28}$$

Similarly,

$$\log|\det \mathbf{J}_{\mathbf{g}}(\cdot)| = \sum_{i=1}^{N} \log|\det \mathbf{J}_{\mathbf{g}_i}(\cdot)|. \tag{3.29}$$

Application to EDAIN-KL

Like with the EDAIN layer, we want to compose the outlier removal, shift, scale and power transform transformations into one operation, which we do by defining

$$\mathbf{g}_{\boldsymbol{\theta}} = \mathbf{h}_1^{-1} \circ \mathbf{h}_2^{-1} \circ \mathbf{h}_3^{-1} \circ \mathbf{h}_4^{-1}, \tag{3.30}$$

where $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\lambda}, \boldsymbol{\lambda}^{(\mathrm{YJ})})$. Notice that we apply all the operations in reverse order, compared to the EDAIN layer. This is because we will use $\mathbf{g}_{\boldsymbol{\theta}}$ to transform our base distribution $\mathbf{Z}$ into a distribution that is as close to our dataset $\mathcal{D}$ as possible. Then, when we want to normalize the dataset, we apply

$$\mathbf{g}_{\boldsymbol{\theta}}^{-1} = h_4 \circ h_3 \circ h_2 \circ h_1 \tag{3.31}$$

to each sample. It can be shown that all the transformations defined in eqs. (3.13) to (3.16) are invertible. Using theorem 3.2.1, it follows that $\mathbf{g}_{\boldsymbol{\theta}}$, as defined in eq. (3.30), is bijective and that its inverse is given by eq. (3.31). As we will see in later in section 3.2.2, the inverse transformation in eq. (3.31) has a tractable and differentiable expression, so $\mathbf{g}_{\boldsymbol{\theta}}$ can be used as a normalizing flow bijection.

Making the input data as Gaussian as possible usually increases performance of deep sequence models (citation needed), so a suitable base distribution is the standard multivariate Gaussian distribution

$$\mathbf{Z} \sim \mathcal{N}(0, \mathbf{I}_d), \tag{3.32}$$

whose pdf has a tractable and differentiable expression, so it is suitable for our needs.

We have that both $p_{\mathbf{Z}}(\cdot)$ and $\mathbf{g}_{\boldsymbol{\theta}}^{-1}(\cdot)$ have analytic expressions and are differentiable, so we have almost everything that we need in order to use eq. (3.20) to optimise $\boldsymbol{\theta}$. The only part remaining is an expression for the log of the determinant of the Jacobian of the forward transformation given by $\mathbf{g}_{\boldsymbol{\theta}}^{-1}$, which we will derive in the next section. Once we have that, $\boldsymbol{\theta}$ can be optimised using back-propagation as described in TODO, using the negation of eq. (3.20) as the loss function $\mathcal{L}(\boldsymbol{\theta})$.

Derivations of inverse log determinant Jacobians

Recall that the EDAIN-KL architecture is just a bijector that is composed of 4 other bijective functions. Using the result in theorem 3.2.1, we get

$$\log \left|\det \mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}(\cdot)\right| = \sum_{i=1}^{4} \log \left|\det \mathbf{J}_{h_i^{-1}}(\cdot)\right|. \tag{3.33}$$

Considering the transformations in eqs. (3.13) to (3.16), we notice that all the transformation happen element-wise, so for $i \in \{1, 2, 3, 4\}$, we have $\frac{\partial h_i^{-1}(x_j)}{\partial x_k} = 0$ for $k \neq j$. Therefore, the Jacobians are diagonal matrices, so the determinant is just the product

of the diagonal entries, giving

$$\log|\det \mathbf{J}_{\mathbf{Z}\to\mathbf{Y}}(\mathbf{x})| = \sum_{i=1}^{4} \log\left|\prod_{j=1}^{d} \frac{\partial h_i^{-1}(x_j)}{\partial x_j}\right| \tag{3.34}$$

$$= \sum_{i=1}^{4}\sum_{j=1}^{d} \log\left|\frac{\partial h_i^{-1}(x_j)}{\partial x_j}\right|. \tag{3.35}$$

We now proceed to deriving the derivatives appearing on the right-hand side for h_1^{-1}, h_2^{-1}, h_3^{-1}, and h_4^{-1}.

Shift We first consider $h_2(x_j;\gamma_j) = x_j + \gamma_j$. Its inverse is $h_2^{-1}(z_j;\gamma_j) = z_j - \gamma_j$, and it follows that

$$\log\left|\frac{\partial h_2^{-1}(z_j;\gamma_j)}{\partial z_j}\right| = \log 1 = 0. \tag{3.36}$$

Scale We now consider $h_3(x_j;\lambda_j) = x_j \cdot \lambda_j$, whose inverse is $h_3^{-1}(x_j;\lambda_j) = \frac{z_j}{\lambda_j}$. It follows that

$$\log\left|\frac{\partial h_3^{-1}(z_j;\gamma_j)}{\partial z_j}\right| = \log\left|\frac{1}{\lambda_j}\right| = -\log|\lambda_j|. \tag{3.37}$$

Outlier removal We now consider $h_1(x_j;\beta_j') = \beta_j' \tanh\left\{\frac{(x_j-\hat{\mu}_j)}{\beta_j'}\right\} + \hat{\mu}_j$. Its inverse is

$$h_1^{-1}(z_j;\beta_j') = \beta' \tanh^{-1}\left\{\frac{z_j-\hat{\mu}_j}{\beta_j'}\right\} + \hat{\mu}_j. \tag{3.38}$$

It follows that

$$\log\left|\frac{\partial h_1^{-1}(z_j;\beta_j')}{\partial z_j}\right| = \log\left|\frac{1}{1-\left(\frac{z_j-\hat{\mu}_j}{\beta_j'}\right)^2}\right| = -\log\left|1-\left(\frac{z_j-\hat{\mu}_j}{\beta_j'}\right)^2\right|. \tag{3.39}$$

Power transform By considering the expression in eq. (3.16), it can be shown that for fixed λ, negative inputs are always mapped to negative values and vice versa, making the Yeo-Johnson transformation invertible. Additionally, in $\mathbf{h}_4(\cdot)$ the Yeo-Johnson transformation is applied element-wise, so we get

$$\mathbf{h}_4^{-1}(\mathbf{z}) = \begin{bmatrix}\left[f_{\mathrm{YJ}}^{\lambda_1}\right]^{-1}(z_1) & \left[f_{\mathrm{YJ}}^{\lambda_2}\right]^{-1}(z_2) & \cdots & \left[f_{\mathrm{YJ}}^{\lambda_d}\right]^{-1}(z_d)\end{bmatrix}, \tag{3.40}$$

where it can be shown that the inverse Yeo-Johnson transformation for a single element is given by

$$\left[f_{\mathrm{YJ}}^{\lambda}\right]^{-1}(z) = \begin{cases} (z\lambda+1)^{1/\lambda} - 1, & \text{if } \lambda \neq 0, z \geq 0; \\ e^{z} - 1, & \text{if } \lambda = 0, z \geq 0; \\ 1 - \{1 - z(2-\lambda)\}^{1/(2-\lambda)}, & \text{if } \lambda \neq 2, z < 0; \\ 1 - e^{-z}, & \text{if } \lambda = 2, z < 0. \end{cases} \tag{3.41}$$

The derivative with respect to z then becomes

$$\frac{\partial\left[f_{\mathrm{YJ}}^{\lambda}\right]^{-1}(z)}{\partial z} = \begin{cases} (z\lambda+1)^{(1-\lambda)/\lambda}, & \text{if } \lambda \neq 0, z \geq 0; \\ e^{z}, & \text{if } \lambda = 0, z \geq 0; \\ \{1 - z(2-\lambda)\}^{(\lambda-1)/(2-\lambda)}, & \text{if } \lambda \neq 2, z < 0; \\ e^{-z}, & \text{if } \lambda = 2, z < 0. \end{cases} \tag{3.42}$$

It follows that

$$\log\left|\frac{\partial\left[f_{\mathrm{YJ}}^{\lambda}\right]^{-1}(z)}{\partial z}\right| = \begin{cases} \frac{1-\lambda}{\lambda}\log(z\lambda+1), & \text{if } \lambda \neq 0, z \geq 0; \\ z, & \text{if } \lambda = 0, z \geq 0; \\ \frac{\lambda-1}{2-\lambda}\log\{1 - z(2-\lambda)\}, & \text{if } \lambda \neq 2, z < 0; \\ -z, & \text{if } \lambda = 2, z < 0, \end{cases} \tag{3.43}$$

which we use as the expression for $\log\left|\frac{\partial h_4^{-1}(z_j;\lambda^{(\mathrm{YJ})})}{\partial z_j}\right|$ for $z = z_1, \ldots, z_d$.

3.3 PREPMIX-CAPS

TODO: write details

4 Results

TODO: introduction to this chapter

4.1 Evaluation methodology

Small introduction

4.1.1 Sequence model architecture

4.1.2 Fitting the models

Mention scheduling, early stopping, optimizer used, learning rate etc.

4.1.3 Tuning adaptive preprocessing model hyperparameters

Details on the tuning for all the methods presented

4.1.4 Evaluation metrics

4.1.5 Cross-validation

4.2 Simulation study

Small introduction, including motivation

4.2.1 Multivariate time-series data generation algorithm

4.2.2 Negative effects of irregularly-distributed data

4.2.3 Preprocessing method experiments

4.3 American Express default prediction dataset

4.3.1 Description

4.3.2 Preprocessing method experiments

5 Discussion

TODO: introduction to this chapter

5.1 EDAIN

5.2 EDAIN-KL

5.3 PREPMIX-CAPS

6 Conclusion

6.1 Summary

Conclusion goes here.

6.2 Main contributions

6.3 Future work

References

Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, nov 2021. doi: 10.1109/tpami.2020.2992934. URL https://doi.org/10.1109%2Ftpami.2020.2992934.

Stanislav I. Koval. Data preparation for neural network data analysis. In *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 898–901, 2018. doi: 10.1109/EIConRus.2018.8317233.

Nazri Mohd Nawi, Walid Hasen Atomi, and M.Z. Rehman. The effect of data preprocessing on optimized training of artificial neural networks. *Procedia Technology*, 11: 32–39, 2013. ISSN 2212-0173. doi: https://doi.org/10.1016/j.protcy.2013.12.159. URL https://www.sciencedirect.com/science/article/pii/S2212017313003137. 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013.

Tamás Nyitrai and Miklós Virág. The effects of handling outliers on the performance of bankruptcy prediction models. *Socio-Economic Planning Sciences*, 67:34–42, 2019. ISSN 0038-0121. doi: https://doi.org/10.1016/j.seps.2018.08.004. URL https://www.sciencedirect.com/science/article/pii/S003801211730232X.

Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Deep adaptive input normalization for time series forecasting. *arXiv preprint arXiv:1902.07892*, 2019.

Nikolaos Passalis, Juho Kanniainen, Moncef Gabbouj, Alexandros Iosifidis, and Anastasios Tefas. Forecasting financial time series using robust deep adaptive input normalization. *Journal of Signal Processing Systems*, 93(10):1235–1251, Oct 2021. ISSN 1939-8115. doi: 10.1007/s11265-020-01624-0. URL https://doi.org/10.1007/s11265-020-01624-0.

Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2019.105524. URL https://www.sciencedirect.com/science/article/pii/S1568494619302947.

J. Sola and Joaquin Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on*, 44:1464 – 1468, 07 1997. doi: 10.1109/23.589532.

Dat Thanh Tran, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Bilinear input normalization for neural networks in financial forecasting. *arXiv preprint arXiv:2109.00983*, 2021.