THST: A Bayesian Approach To Long Term Climate Prediction

Rilwan A. Adewoyin[1,2,4], **Yulan He**[1], **and Ritabratta Dutta**[3,4]

[1]Department of Computer Science, University of Warwick, UK
[2]Department of Computer Science, Shezhen Univeristy, Shenzhen, China
[3]Alan Turing Institute, London, UK
[4]Department of Statistics, University of Warwick, UK

Initially, we have designed a BNN implementation of CNN based DeepSD for the closely related task of Precipitation Downscaling, an advancement on the previously used Dropout method.

As our focus is in computing the probability of extreme events in the future, we are developing a framework using Bayesian neural Networks (BNN) (eg. dropout, flip-out etc).

We have used our flip-out adjusted implementation of DeepSD (1) for the task of downscaling Precipitation in the USA region. We achieve improved performance. Our next goal is to extend this Bayesian framework to predict future extreme rainfall events.

To that end, we propose a novel architecture, Temporal Hierarchical Spatial Transformer (THST), a ConvLSTM based model designed to efficiently utilise long and short temporal relationships and spatial relationships in model fields.

This is an ongoing work carried out by Rilwan Adewoyin with supervision from Ritabrata Dutta and Yulan He.

Bayesian Neural Networks | Super-resolution | Convolutional Networks | Network Compression | ConvLSTM | Spatio-temporal modelling

Correspondence: ***rilwan.adewoyin@warwick.ac.uk***

Contents

1: Bayesian Neural Networks

Variational Inference. In a Bayesian Neural Network (BNN), a prior distribution p is placed on the weights $\omega \sim p(\omega)$ of the neural network. During training an optimal posterior distribution q over the weights of the neural network is learned. When evaluating unseen data the learnt posterior distribution q facilitates the stochastic predictions. These stochastic predictions can be used to determine prediction intervals as well as point predictions.

Formally, given a neural network, $f(x_n, \omega)$, with weights ω and taking input x_n, and a training set $D = (x_n, y_n)_{n=1}^{N}$, we aim to learn a predictive distribution 1 for unseen data (x^*, y^*)

$$p(y^*|f(x^*,\omega)) = \int f(y^*|f(x^*|x^*,\omega) \cdot p(\omega|D)d\omega \quad \textbf{(1)}$$

In 1 the rightmost term can not be calculated since its decomposition contains the intractable term $p(D)$ 2

$$p(\omega|D) = \frac{p(D|\omega \cdot p(\omega)}{p(D)} \quad \textbf{(2)}$$

To bypass this intractable term, we learn an approximation q for the weights 3, where θ represents the latent variables of q. When training the BNN, we maximise the Variational Free Energy (VFE), the motivation for which can be found in A.

$$q(\omega|D,\theta) \approx p(\omega|D) \quad \textbf{(3)}$$

$$\begin{aligned} VFE &= -ELBO \\ ELBO &= \mathbb{E}_{q(z|x)}[\ln p(x|z)] - D_{KL}[q(z|x)\|p(z)] \end{aligned} \quad \textbf{(4)}$$

4 shows that that the optimization goal is to minimize the predictive loss, through the LHS, and minimize the KL-divergence between $q(\omega|D,\theta)$ and $p(\omega|D)$ through the RHS.

The various types of Bayesian networks differ in their specification of the prior $p(W)$ and approximating posterior $q(\omega)$. Careful specification of the prior $p(\omega)$ allows problem-specific beliefs to be encoded into the model, such as sparsity or regularised weights. Furthermore, the structure of $q(\omega)$ is essential for capturing complex relationships such as correlation of nodes within a layer.

Practical Implementation of Variational Inference. For generic variational inference within a BNN we require a Monte Carlo inference scheme to estimate the expected log likelihood term in 4. Specifically, given a set of latent variables θ and a posterior $q(\omega|D,\theta)$, the VFE as can be approximated using M posterior samples w_i for the weights of the network. 5

$$\mathcal{F}(\mathcal{D},\boldsymbol{\theta}) \approx \frac{1}{M}\sum_{i=1}^{M}[\log q\left(\mathbf{w}^{(i)}|\boldsymbol{\theta}\right) - \log p\left(\mathbf{w}^{(i)}\right) - \log p\left(\mathcal{D}|\mathbf{w}^{(i)}\right)] \quad \textbf{(5)}$$

To evaluate the log likelihood, rightmost, term in 5 we use the M posterior samples ${w_i}_{i=1}^{M}$ and f to create M estimates of each true target $\{y_{n,i}\}_{n=1,i=1}^{N,M}$.
We denote the M predictions for a single y_n as $\mathbf{\hat{y}_n}$. Estimates for the mean and variance are calculated from $\mathbf{\hat{y}_n}$. These estimates are used in the rightmost term of 5. Alternatively, during prediction the mean and variance are used to generate both point prediction and confidence intervals for any given prediction.

A drawback of this Monte Carlo BNN training framework is that M forward passes must be performed for each backward propagation, a relatively expensive process compared to normal back-propogation. It is common for the likelihood in 5 to be a distribution that contains a (scale) parameter which is estimated using the empirical second central moment. As such the scale parameter can not be calculated from one sample.

Crude methods to avoid this penalization on efficiency include using a fixed variance term or a variance term that scales proportional to the loss. An alternative approach (2) is to place trainable distributions on variance term.

2: Horseshoe Prior

Motivation. When producing probabilistic predictions from a BNN it is ideal for the prediction intervals to be directly independent of the structural characteristics of the network, and only directly dependent on the networks ability to estimate the correct value.

Ghosh (2) highlighted the following drawback of existing BNNs. Given two fully connected Bayesian networks b_1, b_2, both containing one hidden layer ,with N, 10N nodes respectively, and a simple function to learn $y = f(x)$. In the scenario where the prediction accuracy of b_1 and b_2 are similar the prediction interval for b_2 will significantly exceed that of b_1. This can be attributed to the proportional relationship between the variance of a network's stochastic prediction and the number of stochastic parameters in a BNN. The Horseshoe Prior equipped with weight pruning addresses this issue by removing excess capacity for a given neural network model.

Standard Horseshoe Prior. Utilising a well documented method in the statistics field of Sparse Bayesian Regression, Ghosh (2) extended the Horseshoe prior to a fully connected network by placing a grouped distribution 6 over the network weights. We use $\omega_{kl} \in \mathbb{R}$ to denote the set of all weights incident into unit k of hidden layer l.

$$\begin{aligned} \omega_{kl}|\tau_{kl}, \nu_l &\approx \mathcal{N}(0, (\tau_{kl}^2, \nu_l^2)\mathbb{I}) \\ \tau_{kl} &\sim C^+(0, b_0) \\ \nu_l^2 &\sim C^+(0, b_g) \end{aligned} \tag{6}$$

Where C^+ represents the Half Cauchy distribution.
This distribution is leptokurtic and has a large mass at zero. During network training this has the effect of pulling all weights, ω_{kl}, towards zero, except for significant weights. Further, all weights within a layer experience uniform pull to 0 through the layerwise ν_l. τ_{kl}^2 allows individual weights within the layer to resist this shrinkage.

Extending upon this, three adaptations of the Horseshoe have been proposed (3) to improve its ability to converge.

Fully Factorized Horseshoe Prior (FFHP) The FFHP regularizes the weights, through an intermediary $\tilde{\tau}$ 7

$$\begin{aligned} \omega_{kl}|\tau_{kl}, \nu_l, c &\sim \mathcal{N}(0, (\tilde{\tau}_{kl}^2, \nu_l^2)\mathbb{I}) \\ \tilde{\tau}_{kl}^2 &= \frac{c^2 \tau_{kl}^2}{c^2 + \tau_{kl}^2} \\ c^2 &\sim \texttt{InvGamma}(c_a, c_b) \end{aligned} \tag{7}$$

Factorized-Tied Horseshoe Prior (FTHP) The FTHP addresses the downsides of the relationship between ω_{kl} and the pair τ_{kl}, ν_l. This dependence 7 often leads to posteriors that lay on a restrictive manifold, leading to a poorly explored sample space. Decomposing the distribution for ω_{kl} via the introduction of an independent Guassian term β 8 alleviates this issue.

$$\omega_{kl} = \tilde{\tau}_{kl} \nu_l \beta_{kl}, \quad \beta_{kl} \sim \mathcal{N}(0, \mathbb{I}) \tag{8}$$

Structured Horseshoe Prior (SHSP) SHSP allows the posterior to capture correlation between units in a fully connected layer. To achieve this a non-factorized distribution is used for all ω_{kl} the within a layer l. Let $\beta_l \in \mathbb{R}^{K, N_l}$ denote the matrix of all β_{kl} between layers $l-1$ and l and $\ln(\tau_l) = \left[\ln(\tau_{1l}), \ldots, \ln(\tau_{K_l})l\right]^T$.
A Matrix Normal Distribution is defined over, D_l, the concatenation of β_l and $\ln(\tau)_l$ as

$$\begin{aligned} D_l &= \begin{bmatrix} \beta_l \\ \ln(\tau)_l^T \end{bmatrix} \\ \nu_l &= \left[\nu_{1l}, \ldots, \nu_{K_l} l\right]^T \\ q\left(D_l|\phi_{B_l}\right) &= \mathcal{MN}\left(D_l|M_l, U_l, V_l\right) \end{aligned}$$

Where M_l represents the mean matrix, U_l, the correlation between weights incident on the same nodes and V_l, the correlation related to the weights originating from a single node. The implementation of either of these three adaptations has been showed to improve upon the generic Horseshoe prior (4) (3) (5).

3: Downcasting Task Introduction

Precipitation Downscaling Data Framework. Our approach to the precipitation downscaling draws parallels to the approach used for precipitation nowcasting in (6)

Denote our time varying input data as, $\{X_t\}_{t=1}^{T}$. X_t is the low resolution 3-dimensional matrix (H,W,C) for representing C model fields, where each model field d_t^c is defined on a (H,W) grid. The first two dimensions of $d_t^c and X_t$ represent a specific position/region on a map.

The target, denoted P_t, isa 2 dimensional array with shape $(s_1 * H, s_2 * W)$, for $s_1, s_2 > 1$. This represents the high resolution precipitation for an arbitrary timestep t.

Formally, the task is to learn a neural network $f(X_t, \omega)$ that outputs a prediction for precipitation on a pre-defined higher resolution grid $(s_1 * H, s_2 * W)$ given data defined at on a lower resolution on a (H,W) grid.
In this case, the region of interest is the USA.

Inter-Stacking X_t. The matrix X_t contains information on the coarse grid precipitation and information from the fine grid model fields and geography. Information on the coarse grid will be defined on a matrix of size (H,W) while information on the find grid is defined on a matrix of size $(s_1 * H, s_2 * W)$.
To resolve this shape mismatch, (1) expands each coarse grid datum in X_t to a matrix of size $(s_1 * H, s_2 * W)$ using Bi-linear Interpolation. However, the credibility of using Bi-linear Interpolation is questionable, furthermore it leads a dimensionality increase in the input data.
To improve on this method we used Inter-Stacking. For every d_c in X_t defined on a fine grid we reshape it from $(s_1 * H, s_2 * W)$ to (H, W, s_3) where $s_3 = s_1 * s_2$. Practically this involves selecting m 'central points'A on the $(s_1 * H, s_2 * W)$ and stacking all other points in the $s_1 * s_2$ region surrounding the central points.
This method does not create the spurious data points created by bi-linear interpolation and it also leads to X_t having a smaller matrix size.

4: Introducing DeepSD Structure

The DeepSD model (1) approaches the precipitation downscaling task as an image hyper resolution task. DeepSD is structured as a Stacked Super Resolution CNN (SSRCNN).

Stacked Super Resolution CNN. Introduced by Dong (7), the Super Resolution Convolution Neural Network contains three sub-modules 1 which aim to learn a function mapping a low resolution image X to a high resolution image Y. The three sub-modules are formulated as

Module 1 :$F_1(\mathrm{X}) = Relu(0, W_1 * \mathrm{X} + B_1)$

Module 2 :$F_2(\mathrm{X}) = Relu(0, W_2 * \mathrm{X} + B_2)$

Module 3 :$F_3(\mathrm{X}) = W_3 * \mathrm{X} + B_3$

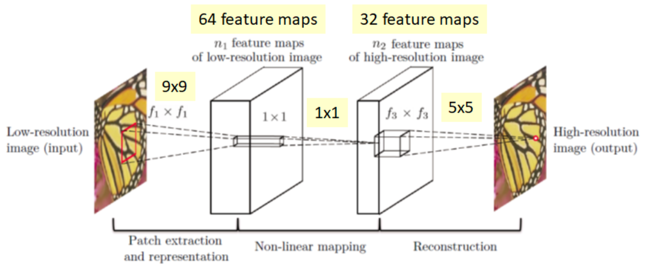

64 feature maps
32 feature maps
n_1 feature maps of low-resolution image
n_2 feature maps of high-resolution image
9x9
$f_1 \times f_1$
1×1
1x1
$f_3 \times f_3$
5x5
Low-resolution image (input)
High-resolution image (output)
Patch extraction and representation
Non-linear mapping
Reconstruction

Fig. 1. SRCNN

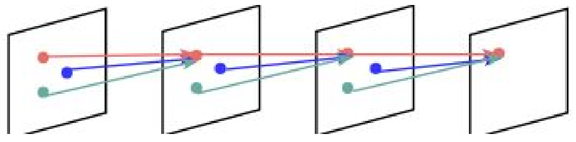
Where W_i is the set of kernels comprising a 2D-Convolution, B_i the corresponding bias units and $*$ represents the convolution operation. The number of filters in W_1 and W_2 can vary, however W_3 must only contain one filter as the output of F_3 is an estimate for the high resolution image Y.

Stacked SRCNN. It is common in image hyper-resolution for this SRCNN unit to provide a 2-4x enhancement in image resolution. Within Precipitation downscaling, it is often the case that our coarse grid field will require an 16x to 32x image super-resolution.
To achieve this resolution increase from r_1 to r_n the n SRCNN modules are stacked, each module increasing the resolution by a factor $\frac{r_{i+1}}{r_i}$ prediction. Each SRCNN module is trained independently. During inference, the input data X_t is input to the first SRCNN and for each other SRCNN the prediction from the previous module is passed as the input to the subsequent SRCNN module.

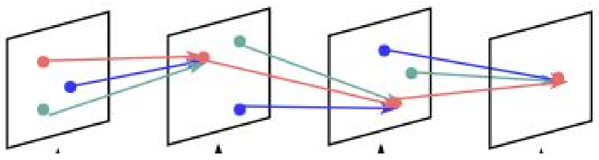
A. Model Improvements. To improve upon the model used in (1) the following features were utilized.

Alternative Bayesian Priors. Three distributions/methods for Bayesian inference were tested, namely;

- Standardized Structured Horseshoe Prior
- Standardized Unstructured Horseshoe Prior
- Flipout Estimator



(a) Generic Convolution



(b) Convolution w/ Spatial Transformer
Fig. 2. Plots of input region to convolution operation

Spatial Transformers. Within weather/climate prediction, translation invariance is of great importance to our convolutional layers. When viewed on a spatial grid most weather patterns do not occur in the exact same shape, but instead as rotations/inversions/reflections. For example prevailing wind flows often form elongated convex patterns.

A drawback of using the traditional convolution structure is that the convolution operation always operates on a fixed grid, defined by the shape of the kernel. Suggested by Jaderberg et al.(8) Spatial Transformers (STs) allow a transformation of the input grid prior to the convolution operation. Again using the example of wind patterns, the ST would be able to ensure the convolution kernels are applied to the relevant region of the grid as shown in 2. Alternative successful approaches to achieve translation invariance for image related tasks include max-pooling and data-augmentation. However, max-pooling also leads to information loss, while there is limited scope of valid data-augmentation when considering precipitation forecasting.

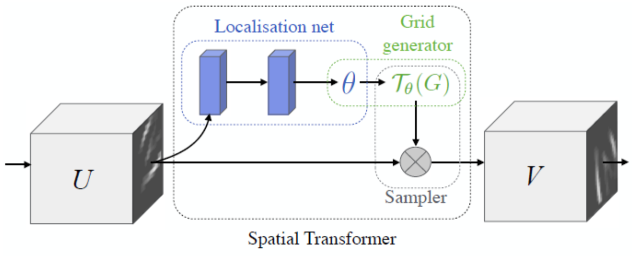


Fig. 3. Spatial Transformer Network

Network Description The Spatial Transformer network precedes the convolution operation and is composed of a Localisation Net, Grid Generator and Sampler. The following description refers to the **??**.

Localisation Net Let U denote an input feature map with dimensions (H,W,C). The outputs of the localisation net are θ. These are the parameters to be used in the Grid Generator Translation step. The localisation net can be limited to scaling and translation 9 or any affine translation 10.

$$A_\theta = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \end{bmatrix} \tag{9}$$

$$A_\theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \tag{10}$$

Grid Generator The Grid generator applies the transformation T_θ, from the Localisation Net, to the input U. The output of this step is a set of *continuous* co-ordinates (x_i, y_j, c) on the original input U.

Sampler The values of U at these continuous co-ordinates $W = (x_i, y_j, c)$ will constitute the new warped matrix. As the co-ordinates are continuous we use bi-linear sampling to generate values from the matrix U for (x_i, y_i, c).
Finally the convolution operation is applied to the sampled feature map V.

5: Structured Horseshoe Prior Attempt

In attempts to use the Horseshoe Prior the significant issue related to underflow.

A. Underflow. Consider the set of n convolution filters $F = \{f_1, f_2, \cdots, f_n\}$ for one SCRNN module in the DeepSD model. Both the input data X_t and intermediate features are 3-dimensional matrices with general shape (H,W,C). For the case of the input data, C is equal to the number of types of weather features / model fields. However the intermediate features may have a large value for C, for example 64 in the original paper for SRCNN (7) and 64 in the deep network extension of SRCNN (9).

Suppose the input to F has dimension $(H_1, W_1, 32)$. As each kernel f_i defines a 2D convolution, the number of weights w^{f_i} belonging to the kernel f_i is $h^{f_i} * W^{f_i} * 32$. Specifically, in the task used in (1) $h^{f_i} = 9$ and $w^{f_i} = 9$.

In the Structured Horseshoe prior formulation, there is a joint prior distribution for the weights w^{f_i}. As such the back-propagation step requires the calculation the log-probability of the w^{f_i} originating from the joint prior distribution. In this scenario w^{f_i} has $2{,}592$ elements. Given tensorflow enforces operations are limited to 32 bit values, an overflow error can easily be reached during this log-probability calculation due to the size of the domain of the joint distribution. This is further exacerbated if the weights naturally evolve to an empirical distribution that is far from the prior distribution

An obvious solution involves significantly reducing the number of filters in the convolution kernels, thus reducing the value of C in other layers. However, this reduced kernel size would significantly impair prediction results as the model capacity would be significantly reduced.

Afterword. References to this problem have been faced by other researchers

1. NaN results in Bayesian Networks

2. Referring to the paper in which Structured Horseshoe Prior was suggested (3), I later noted that Structured Horseshoe Prior is used in fully connected layers with a kernel with up to $10{,}000$ weights. To ensure that I had not made a coding error, I checked source code by Soumya Ghosh for Horseshoe Prior. However, this source code only provides an implementation of the factorized Horseshoe Prior

Unfortunately, a decent amount of time was spent tinkering with this model, in attempts to alleviate the problem mentioned. Following this I decided to forgo the Structured Horseshoe Prior in favour of the Factorized-Tied HS 8

B. Non Structured HorseShoe Prior. During training, the Non Structured Horseshoe Prior (NSHSP) did not face underflow problems. However, convergence issues stemming from performing BNN monte-carlo schemes with low sample size.

During the gradient update step for Bayesian Neural Networks, the Variational Free Energy $\mathcal{F}$ contains an expectation of the log-likelihood term lk; the second term on the right of 11.

$$\mathcal{F}(\mathcal{D},\boldsymbol{\theta}) = \mathrm{KL}(q(\mathbf{w}|\boldsymbol{\theta})\|p(\mathbf{w})) - \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(\mathcal{D}|\mathbf{w}) \quad \textbf{(11)}$$

$$\begin{aligned} lk &= \mathbb{E}_{q(\mathbf{w}|\theta)} \log p(\mathcal{D}|\mathbf{w}) && \textbf{(12)} \\ &= l\left(\hat{\mu}, \hat{\sigma}^2; y_1, \ldots, y_n\right) && \textbf{(13)} \\ &= -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln\left(\hat{\sigma}^2\right) - \frac{1}{2\hat{\sigma}^2}\sum_{j=1}^{n}(y_j - y^*)^2 && \textbf{(14)} \end{aligned}$$

As discussed in 1, the monte carlo sample of predictions is used to calculate the the mean $\hat{\mu}$ and variance $\hat{\sigma}^2$ terms for Guassian Distributions 14.

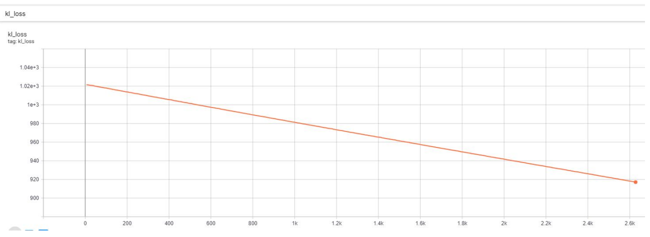
Improving 1 step Forward Pass Inference To reduce the computational burden of this monte-carlo training scheme, I was aiming to perform stable inference with one forward pass. Under this one sample monte-carlo scheme the $\hat{\mu} = y_1$, while the $\hat{\sigma}^2$ was determined using two methods:

1. Fixed at an appropriate value deduced through trial and error

2. Randomly sampled from a distribution. This method is used by Shoumya for regression tasks. (3)

Method 1, produced relatively stable results. One issue that occurred related to the coupling of long tailed distributions and one sample monte-carlo training scheme. Specifically, the NSHSP contains Half-Cauchy distributions 8. During training, I traced my KL divergence term on tensorboard. The KL divergence term in **??** would experience sporadic jumps **??**.

TODO: Insert image showing NSHSP convergence



(b) Smooth Convergence of Flip out

This issue was partially addressed by lowering the initialisation values for the scale terms of the hyper-parameters which were drawn from a Half-Cauchy Distribution. This significantly reduced the occurrence of sporadic jumps in the KL divergence term. However, the odd sporadic jump in KL divergence still occurred. Due to this I choose to advance with a Flipout implementation of DeepSD, until I could formulate a completely stable version of this NSHSP.

6: Flipout Estimator

Motivation. All the aforementioned variants of Horseshoe Prior fall are all methods that utilise the Reparameterization Trick. One drawback of all Reparameterization Trick methods is that for any given training mini-batch, identical weights are sampled for every sample in the mini-batch. Since ,the gradient update $\delta\omega_B$ from a batch of data containing N samples is calculated as the average of the gradient from each sample **??**, there exists correlation between the gradients for each example within the mini-batch. Flipout, varies the pertubes the weights to each example within the mini-batch leading to lower variance in gradient during training, while also only needing one forward stochastic pass during training.

$$\Delta\omega_{MB} = \frac{1}{N}\sum_{i=1}^{N}\Delta\omega_i \tag{15}$$

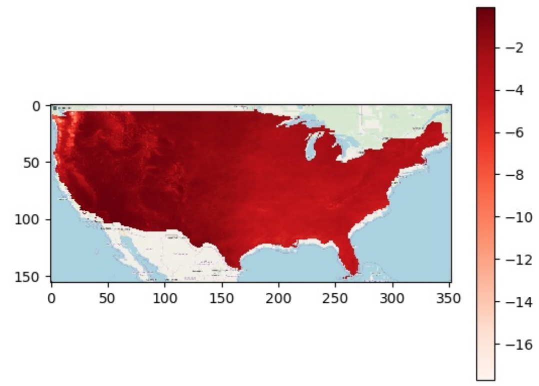
The Local Reparameterization Trick extends the Reparameterization trick by introducing this independence among weights applied to examples within a mini-batch. However extensive tests, have found that Flipout outperforms Local Reparameterization (10).

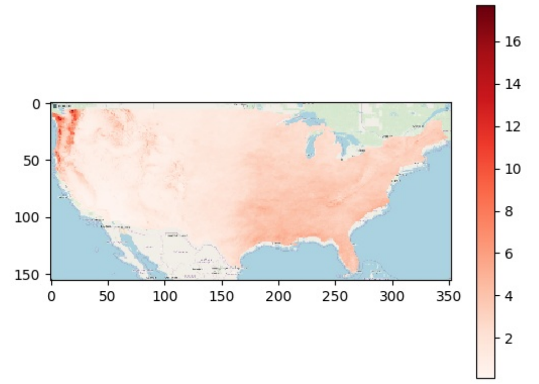
Formulation. TODO: Provide informative summarized mathmatical epxlantion of Flipout

Results. As it provided the most stable training convergence, I focused on producing results using DeepSD with Flipout. To evaluate the predictions I have produced three illustrations: average bias; average rmse and hit rate.

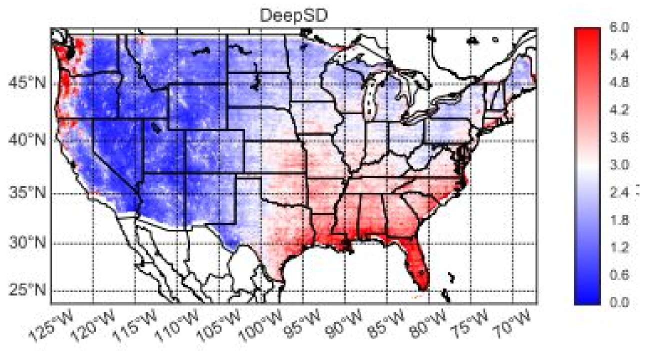
Average Bias and Average RMSE The units of the 5 are mm/day. 5c shows Vandals RMSE Results. All diagrams indicate that the Midwestern region of USA is the easiest to predict, while coastal areas such as San Francisco and Florida which have more varied weather are harder to predict. From 5a and 5b it can be seen that the Rocky Mountains.
To provide a more detailed comparison of models, I will recreate the DeepSD model and produce prediction results.

Hit Rate Using our BNN, we can produce prediction intervals. Hit Rate is the proportion of prediction intervals which contain the true values.
To provide a more detailed comparison of my Hit rate performance and Vandals





(b) Average RMSE



(c) Vandal RMSE

Fig. 5. Evaluation Plots

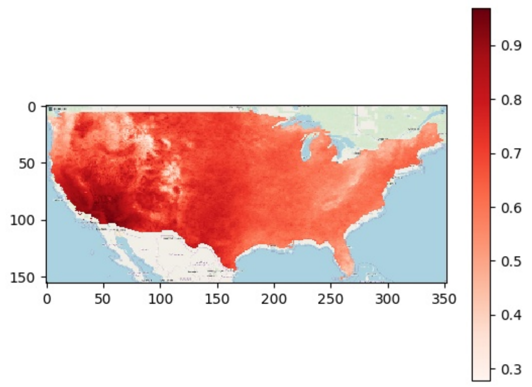


Fig. 6. Hit Rate

7: Miscellaneous

SECTION TO BE COMPLETED AFTER HORSESHOE PRIOR 2ND IMPLEMENTATION

Gradual Distributional Pruning . After training with the horsheshoe prior, there will many weights with small relative magnitude but that are not 0. A common approach to address this is to use a heuristic boundary for values of weights, under which the weights are pruned. [reference] uses the 5 percentile as a boundary. In a non Bayesian setting to introduce some dependence on the value [so and so] have suggested basing the value on the gradients. However, this often.....

Adaptive KL loss weighting scheme. **??** uses an evolutionary weighting scheme, wherein each iteration i has the KL loss scaled by $s_i = \frac{2^{M-i}}{2^M - 1}$. Essentially, this method ensures that when training begins, the prior kl divergence, is large similar to the log likelihood. The exponential decay of this function mimics the typical exponential decay present in NN loss decrease. However, this is not computationally feasible when training on large batches of data. Provide a method where the next weight for the KL loss is dependent on the variance of the gradients at the previous step, in a manner similar to ADAM

8: Next Steps

A. Unstructured Horseshoe Prior.

- After discussion with Dr.Dutta, less focus will be placed on performing BNN training with low monte carlo sampling size. Drawing larger samples will improve stability of the Unstructured Horeshoe Prior.
- Produce Illustrative evaluations for Horeshoe prior, similar to produced for Flipout Estimator
- Implement weight pruning method, prior to producing prediction results

B. Spatial Transformer.

- Build General Spatial Transformer module, with Bayesian layers as opposed to non stochastic layers
- Add to the Horseshoe implementation of DeepSD

C. Comparison To Vandal.

1. Vandal evaluates two metrics R20 Error and SDII Error besides Bias and RMSE. To evaluate the performance benefit of my model compares to him, I must evaluate these metrics. However, Vandal does not produce results for Hit Rate.
2. Vandal also uses three variants of the likelihood term: Guassian, DC-Guassian and DC-LogNormal. I think it would be worthwhile to also evaluate the difference in my model with alternative likelihoods.

9: Progress Update

Comparison with Vandal I have successfully implemented an version of a Bayesian DeepSD which uses the Flipout methodology, widely regarded as an improvement on the Dropout methodology used in Vandal's paper. To quantify this improvement, I will download Vandal's implementation of DeepSD https://github.com/tjvandal/deepsd. However, Vandal's github does not contain a repository for his paper in which he used Dropout to produce prediction bands. As such, Vandal's deterministic DeepSD is the only model I can compare against. However, deterministic models always produce lower biases that their stochastic counterparts.

Spatial Transformer Implementation As well as model Spatial Transformer's to the model and inducing stability in the Unstructured Horeshoe Prior.

Behind Schedule I am behind in the following areas:

- Utilizing large monte carlo sample training with the Unstructured Horseshoe Prior as discussed with Dr Dutta

10: Long-Term Precipitation Forecasting Task

Overview The aim of long term rain forecasting is to use sequences of model field data to forecast a fixed length time window of the future rainfall. These model fields are either based on observed values or generated using systems of differential equations. The model fields can be defined on a coarse or fine grid.

Formulation Suppose we observe model field data over a spatial region. Each type of climate variable is defined by a $M \times N$ grid which varies over time. Given P different types of climate variable, we have a tensor $\mathcal{X} \in \mathbf{R}^{M \times M \times P}$ where $\mathbf{R}$ signifies the domain of observed climate variables. The observations of this matrix $\mathcal{X}$ occur at six hour intervals, producing a sequence of tensors $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_t$.
In this task we aim to produce daily predictions for precipitation $\mathcal{Y}_t \in \mathbf{R}^{M \times M}$ on the same $M \times N$ spatial grid.

In producing predictions for $\mathcal{Y}_t$, the THST model a uses a length J time window of 6 hour model field data $\mathcal{X}_t$ to predict, daily precipitation in the same time window 16.

$$\tilde{\mathcal{Y}}_t, \ldots, \tilde{\mathcal{Y}}_{t-J} = \underset{\mathcal{Y}_t, \ldots, \mathcal{Y}_{t-J}}{\arg\max}\, p\left(\mathcal{Y}_t, \ldots, \mathcal{Y}_{t-J} | \mathcal{X}_t, \mathcal{X}_{t-1}, \ldots, \mathcal{X}_{t-J}\right) \tag{16}$$

11: Background and Related Work

Long Short-Term Memory. A typical RNN structure is an ordered chain of N RNN cells $D_1, D_2, \ldots, D_N$. Each of the $t \in N$ cells has a memory matrix M_t and takes as input the memory matrix M_{t-1} from the previous cell and an exogenous input $\mathcal{X}_t$. In the Long Short-Term Memory (LSTM) structure 7 17 the single memory matrix M_t, is replaced by two matrices; the cell state matrix $\mathcal{C}_t$ and the hidden state matrix $\mathcal{H}_t$.

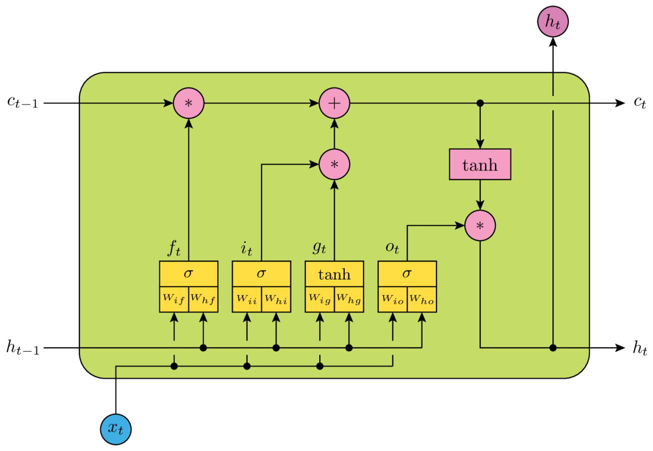


Fig. 7. LSTM Cell Illustration

$$\begin{aligned} i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\ f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\ o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\ \tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\ C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\ h_t &= \tanh\left(C_t\right) * o_t \end{aligned} \tag{17}$$

ConvLSTM. To enable the propagation of spatial information throughout the LSTM cell during inference, Shi proposed the ConvLSTM (11). In ConvLSTM the matrices and operations defining the cell operation belong to the three-dimensional space 18; namely the inputs $\mathcal{X}_1, \ldots, \mathcal{X}_t$, hidden states $\mathcal{H}_1, \ldots, \mathcal{H}_t$ and cell states $\mathcal{C}_1, \ldots, \mathcal{C}_t$ and the gates/functions i_t, f_t, o_t.

$\circ$ denotes the Hadamard product

$*$ denotes the convolution operator

$$\begin{aligned} i_t &= \sigma\left(W_{x,i} * \mathcal{X}_t + W_{h,i} * \mathcal{H}_{t-1} + W_{c,i} \circ \mathcal{C}_{t-1} + b_i\right) \\ f_t &= \sigma\left(W_{x,f} * \mathcal{X}_t + W_{h,f} * \mathcal{H}_{t-1} + W_{c,f} \circ \mathcal{C}_{t-1} + b_f\right) \\ \mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh\left(W_{x,c} * \mathcal{X}_t + W_{h,c} * \mathcal{H}_{t-1} + b_c\right) \\ o_t &= \sigma\left(W_{x,o} * \mathcal{X}_t + W_{h,o} * \mathcal{H}_{t-1} + W_{c,o} \circ \mathcal{C}_t + b_o\right) \\ \mathcal{H}_t &= o_t \circ \tanh\left(\mathcal{C}_t\right) \end{aligned} \tag{18}$$

Cross Attention Module. Suppose we have three sets of time dependent vectors, $\{\mathcal{A}\}_{i=1}^{T}, \{\mathcal{B}\}_{i=1}^{T} and \{\mathcal{C}\}_{i=1}^{T}$, with dimensions of d_a, d_b and d_c respectively. Furthermore, each triplet $(\mathcal{A}_i, \mathcal{B}_i, \mathcal{C}_i)$ are related.

Cross Attention allow us to learn an improved vector representation of $\mathcal{C}_i$, as a weighted average of $\{\mathcal{C}_j\}, j \in \mathbb{N}_T \setminus i$ based on the 'similarity' between $\mathcal{A}_i$ and $\{\mathcal{B}_j\}, j \in \mathbb{N}_T \setminus i$.

A cross attention module has three trainable weight matrices, $W_Q^{(d_q,d_a)}, W_K^{(d_k,d_b)}, W_V^{(d_v,d_c)}$. For each set of vectors denote the matrix $\mathcal{A} = \text{Concat}[\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_T]$, $\mathcal{B} = \text{Concat}[\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_T]$ and the matrix $\mathcal{C} = \text{Concat}[\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_T]$.
First the Key(K), Query(Q) and Value(V) Matrices are calculated.

$$* \text{ denotes matrix multiplication}$$

$$K = W_Q * \mathcal{A} \quad \textbf{(19)}$$

$$Q = W_K * \mathcal{B} \quad \textbf{(20)}$$

$$V = W_V * C \quad \textbf{(21)}$$

Following this, for each $\mathcal{A}_i$, *T-1* attention (similarity) scores, $\mathcal{S}_{ij}$ are calculated. Each one encoding the similarity between $\mathcal{A}_i$ and $\{\mathcal{B}_j\}, j \in \mathbb{N}_T \setminus i$ 22.
Then the new representation of $\mathcal{C}_i$ is calculated 23. In the THST model the score operator represents a scaled dot product operation(12).

$$\mathcal{S}_{ij} = \frac{\exp(\text{score}(\mathcal{A}_i, \mathcal{B}_j))}{\sum_{s'=1}^{S} \exp(\text{score}(\mathcal{A}_i, \mathcal{B}_j))} \quad \textbf{(22)}$$

$$\mathcal{C}_i^{(new)} = \sum_j \mathcal{S}_{ij} * \mathcal{C}_j \quad \textbf{(23)}$$

$$\textbf{(24)}$$

Bi-Directional LSTM.

Models Related to THST. The aim of Precipitation Nowcasting is to predict precipitation in a relatively small time horizon. The input features (model fields) $\{\mathcal{X}_i\}_{i=1}^{T}$ are at time points preceding the targets (precipitation) $\{\mathcal{Y}_i\}_{i=T+1}^{T+K}$. For this task of Precipitation Nowcasting, Shi proposed the encoder-forecaster structure (6), Illustrated in 8.

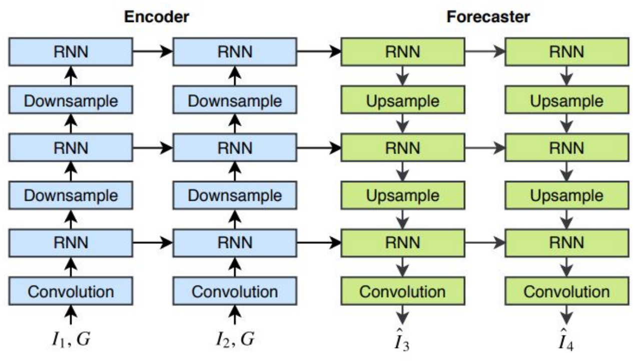


Fig. 8. Encoder-Forecaster Model

The stacked ConvLSTM layers include spatial downsampling. This allows the network to learn larger scale spatial patterns. THST provides a natural extension/adaptation of this model to long term prediction. First THST includes temporal upsampling/downsampling within the hierarchy of LSTM layers. This ensures the network is aware of temporal patterns at a short and long timescale. Further, THST adopts an encoder-decoder structure as opposed to an encoder-forecaster structure.

12: Temporal Hierarchical Spatial Transformer

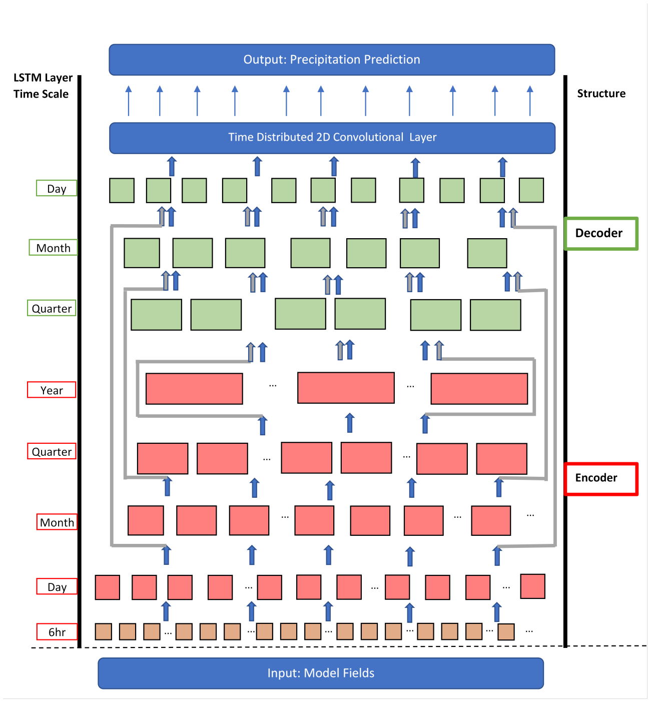


Fig. 9. THST Model

The THST model can be divided into the following three parts, to be introduce in the following sections:

1. Input Layer
2. Encoder
3. Decoder and Output Layer

Model Size TODO: Complete when a preliminary size for model is pinned down

Input Layer. The Input Layer, shaded brown, is a simple Bi-Directional ConvLSTM, taking a sequence of $\mathcal{X}_i$ as input and passing hidden states to the the first layer of the encoder.

Encoder: Stacked Sparsity LSTM.

Overview The Encoder is the section shaded red in 9. The Encoder is formed by stacking multiple LSTM layers. Each constituent horizontal layer of red squares represents one Bi-directional LSTM layer. Within this stack, each layer outputs hidden states, which become inputs to the cells in the layer above. Referring again to 9, the 'LSTM Time Scale' indicates the temporal information encoded by the corresponding LSTM layer. For example, the encoder's 2nd lowest LSTM layer, labelled 'Month', will contain information relating to a monthly representation of precipitation.Intuitively, each LSTM cell in the 'Month' LSTM layer receives hidden states from the 30 corresponding and consecutive LSTM cells in the 'Day' LSTM layer. The hidden states transferred received by LSTM cells in layer 'Month' are mutually exclusive as shown in 10.

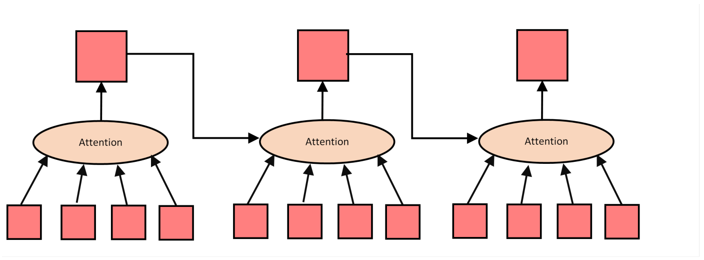


Fig. 10. Attention Mechanism between LSTM layers

Mechanism Generalizing to any Encoder layer $l, l \in \{1-4\}$, denote the i^th LSTM cell of l as D_{li}. The hidden state from cell D_{li} as $\mathcal{H}_{li}$ and the input to D_{li} as $\mathcal{X}_{li}$. Finally, the $M_l \in \mathbb{N}$ hidden states that D_{li} consumes from cells, $\{d_{l-1,j}\}_{j \in I_{li}}$, in the lower layer are denoted $\{\mathcal{H}_{lj}\}_{j \in I_{li}}$; where I_{li} is the set of indexes j for cells $D_{(l-1)j}$ which send their hidden state to D_{lj}

As explained in 18, a generic Convolutional LSTM cell takes one input. For cell $D_{(l-1)j}$ cross attention is used to calculate the one input $\mathcal{X}_{li}$. Referring to 19, 20, 21

- $X_{l,i}$ flattened is the new representation, $C_i^{(new)}$
- A down-scaled (3D-Average-Pool) version of the hidden state $\mathcal{H}_{l,i-1}$ flattened, as $\mathcal{A}$, is used to calculate the Key matrix K.
- Down-scaled versions (3D-Average-Pool) of $\{\mathcal{H}_{l,j}\}_{j \in I_(li)}$ flattened takes the role of $\mathcal{B}$.

The down-scaling performed on $\mathcal{H}_{l,i-1}$ and $\{\mathcal{H}_{l,j}\}_{j \in I_(li)}$ is 3D Average Pooling, wherein the pooling filter has depth 1. The 3D Average Pooling is applied before the matrix is flattened.

Decoder and Output.

Overview The Decoder is the section shaded green in 9. The Decoder is formed by stacking multiple LSTM layers, each constituent horizontal layer of green squares representing one bi-directional LSTM layer. Within the stack, each layer takes information from two sources, the previous layer and the corresponding layer, by time scale, in the encoder. The input of two features into the decoder layers is indicated by the grey and blue arrows 9. The final layer of the Decoder is the 'Day' LSTM layer; the hidden states from this layer are passed to a Time Distributed Convolutional Layer (TDCL). The output of the TDCL will be daily precipitation predictions.

Mechanism To allow the LSTM cell's in the decoder to effectively factor in information from two different time scales, we propose a two cell state variant of the LSTM cell 25.. Specifically, there are 2 sets of gates and cell states; one for each time scale. However, it is still the case that one hidden state is produced by each cell.

$$\begin{aligned}
&j \in 1, 2 \\
i_t^j &= \sigma\left(W_{x,i}^j * \mathcal{X}_t W_{h,i} * \mathcal{H}_{t-1} + W_{c,i}^j \circ \mathcal{C}_{t-1}^j + b_i^j\right) \\
f_t^j &= \sigma\left(W_{x,f}^j * \mathcal{X}_t^j + W_{h,f} * \mathcal{H}_{t-1} + W_{c,f}^j \circ \mathcal{C}_{t-1}^j + b_f^j\right) \\
\mathcal{C}_t^j &= f_t^j \circ \mathcal{C}_{t-1}^j + i_t^j \circ \tanh\left(W_{x,c}^j * \mathcal{X}_t^j + W_{h,c} * \mathcal{H}_{t-1} + b_c^j\right) \\
o_t^j &= \sigma\left(W_{x,o}^j * \mathcal{X}_t^j + W_{h,o} * \mathcal{H}_{t-1} + W_{c,o}^j \circ \mathcal{C}_t^j + b_o^j\right) \\
\mathcal{H}_t &= 0.5 * \left[o_t^1 \circ \tanh\left(\mathcal{C}_t^1\right) + o_t^2 \circ \tanh\left(\mathcal{C}_t^2\right)\right]
\end{aligned} \tag{25}$$

13: Bayesian Neural Network

To introduce uncertainty to the model, the convolutional and dense layers in THST, were replaced with their Bayesian counterparts. TODO:Finish this section when Bayesian THST completed

A. Results.

Dropout

Flipout

Unstructured Horseshoe Prior

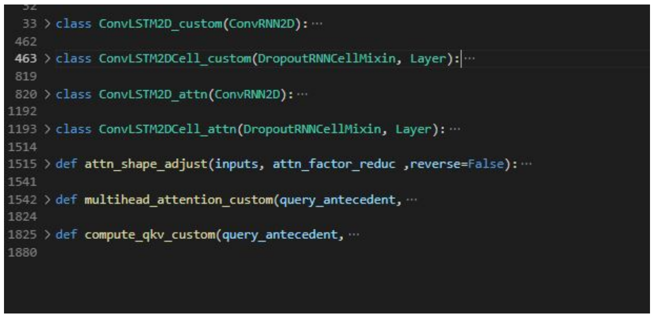
14: Next Steps

1. Debug improved system for attention in the decoder layer trains
2. Ensure non-BNN THST trains
3. Produce Predictions with non-BNN THST using full 30 years of ATI data
4. Add Spatial Transformers to appropriate parts of the LSTM layers in the encoder and decoder.
5. Ensure non-BNN THST trains
6. Create Unstructured Horseshoe Prior version of THST
7. Create Flipout version of THST
8. Discuss w/ Sherman, the best methods to evaluate model performance
9. Produce Predictions

15: Progress Update

Completing code for novel structures The github repository is accessible at this link https://github.com/Akanni96/Bnn_CNN_Horshoe/invitations for Dr Dutta.
The two input ConvLSTM and the Cross-Attention assisted ConvLSTM (CACL) were both novel structures, created to adapt the encoder-forecaster structure. (6) to a structure appropriate to this prediction task. The bulk of new code I wrote for these two structures can be observed in the layers_ConvLSTM2D.py

Memory Issues Related To Attention on 3D Tensors As discussed with Dr. He, my initial implementation of the Cross-Attention assisted ConvLSTM lead to memory issues. Since Attention operates on vectors, I was initially flattening 3D tensors. However this lead to large vectors. On Friday, I formulated a solution involving 3D Average Pooling using filters of size $(n, m, 1)$ and stride $(n, m, 1)$. On the most weekend I revised my code for the CACL. This involved creating a modified method for multi-head cross attention since Tensorflows' implementation of Attention does not allow multiple heads, while Tensor2Tensor's implementation of Attention does not allow cross attention.



```
> class ConvLSTM2D_custom(ConvRNN2D):···

> class ConvLSTM2DCell_custom(DropoutRNNCellMixin, Layer):···

> class ConvLSTM2D_attn(ConvRNN2D):···

> class ConvLSTM2DCell_attn(DropoutRNNCellMixin, Layer):···

> def attn_shape_adjust(inputs, attn_factor_reduc ,reverse=False):···

> def multihead_attention_custom(query_antecedent,···

> def compute_qkv_custom(query_antecedent,···

```

Fig. 11. Caption

Unfortunately, I am behind in the following areas:

- Inspecting predictive performance of a single LSTM layer as discussed with Dr Dutta
- Producing predictions with the preliminary version of the THST model
- Creating a completely presentable, well detailed description/introduction to all relevant neural network structures used in THST
- Including a description of Generic LSTM model prior to introducing the ConvLSTM model as discussed with Dr Dutta
- Adapt my key parts of my code-base to be interoperable with Sherman's codebase. In particular training, prediction and prediction evaluation.

I will prioritize steps related to producing a provisional step of predictions for the ATI data using the THST data. Following, this I will work on implementinf the Horseshoe variant of Deep SD. (1)

Appendix

A: 1

ELBO LOSS.

$$
\begin{aligned}
& \ln p(x) \\
=& \ln \int_z p(x,z) \\
=& \ln \int_z p(x,z) \frac{q(z|x)}{q(z|x)} \\
=& \mathbb{E}_{q(z|x)} \left[\ln \frac{p(x,z)}{q(z|x)} \right] \\
\geq & \ \mathbb{E}_{q(z|x)} [\ln p(x|z)] + \int_z q(z|x) \ln \frac{p(z)}{q(z|x)} \Bigg] \\
=& \mathbb{E}_{q(z|x)} [\ln p(x|z)] + \int_{KL} [q(z|x) \| p(z)] \\
=& \text{ likelihood } - KL
\end{aligned}
\tag{26}
$$

Central Points. Given two matrices X_1 and X_2 of size $(m_1 * H, m_2 * W)$ and (H, W) , the central points ofX_1 relative to X_2 are those points which relate to the same position on the spatial map represented by both matrices. If X_1 is defined on the upscaled granuality of the same map as X_2, then this central points of X_1 can be located by taking (m_1, m_2) strides along X_1

Bibliography

1. Thomas Vandal, Evan Kodra, Sangram Ganguly, Andrew Michaelis, Ramakrishna Nemani, and Auroop R. Ganguly. Deepsd: Generating high resolution climate change projections through single image super-resolution. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1663–1672, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348874. doi: 10.1145/3097983.3098004.
2. Soumya Ghosh and Finale Doshi-Velez. Model selection in bayesian neural networks via horseshoe priors, 2017.
3. Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors, 2018.
4. Hiske Overweg, Anna-Lena Popkes, Ari Ercole, Yingzhen Li, José Miguel Hernández-Lobato, Yordan Zaykov, and Cheng Zhang. Interpretable outcome prediction with sparse bayesian neural networks in intensive care, 2019.
5. Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors, 2016.
6. Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Deep learning for precipitation nowcasting: A benchmark and a new model. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5617–5627. Curran Associates, Inc., 2017.
7. Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015.
8. Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.
9. Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. *CoRR*, abs/1511.04587, 2015.
10. Hector J. Hortua, Riccardo Volpi, Dimitri Marinelli, and Luigi Malagò. Parameters estimation for the cosmic microwave background with bayesian neural networks, 2019.
11. Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 802–810. Curran Associates, Inc., 2015.
12. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

Word Counts

This section is *not* included in the word count.

Statistics on word count

```
File: 00_Article_Merge.tex
Sum count: 5095
Words in text: 4596
Words in headers: 207
Words outside text (captions, etc.): 48
Number of headers: 80
Number of floats/tables/figures: 14
Number of math inlines: 229
Number of math displayed: 15
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  30+9+0 (1/0/0/0) _top_
  143+1+0 (1/0/0/0) Abstract
  13+0+0 (0/0/0/0) main
  0+3+0 (1/0/0/0) Section: Bayesian Neural Networks
  214+2+0 (1/0/19/3) Subsection: Variational Inference
  239+5+0 (1/0/9/0) Subsection: Practical Implementation of Variational Inference
  0+2+0 (1/0/0/0) Section: Horseshoe Prior
  133+1+0 (1/0/6/0) Subsection: Motivation
  270+16+0 (4/0/26/3) Subsection: Standard Horseshoe Prior
  0+3+0 (1/0/0/0) Section: Downcasting Task Introduction
  118+4+0 (1/0/13/0) Subsection: Precipitation Downscaling Data Framework
  148+1+0 (1/0/15/0) Subsection: Inter-Stacking $X_t$
  201+9+1 (3/1/15/0) Section: Introducing DeepSD Structure
  322+14+16 (7/2/12/2) Subsection: Model Improvements
  13+4+0 (1/0/0/0) Section: Structured Horseshoe Prior Attempt
  234+1+0 (1/0/18/0) Subsection: Underflow
  134+1+0 (1/0/1/0) Subsection: Afterword
  254+10+8 (2/3/6/0) Subsection: Non Structured HorseShoe Prior
  0+2+0 (1/0/0/0) Section: Flipout Estimator
  134+1+0 (1/1/2/0) Subsection: Motivation
  8+1+0 (1/0/0/0) Subsection: Formulation
  138+8+10 (3/2/0/0) Subsection: Results
  9+1+0 (1/0/0/0) Section: Miscellaneous
  75+3+0 (1/0/0/0) Subsection: Gradual Distributional Pruning
  91+5+0 (1/0/2/0) Subsection: Adaptive KL loss weighting scheme
  0+2+0 (1/0/0/0) Section: Next Steps
  51+3+0 (1/0/0/0) Subsection: Unstructured Horseshoe Prior
  21+2+0 (1/0/0/0) Subsection: Spatial Transformer
  68+3+0 (1/0/0/0) Subsection: Comparison To Vandal
  127+10+0 (4/0/0/0) Section: Progress Update
  155+6+0 (3/0/11/1) Section: Long-Term Precipitation Forecasting Task
  0+4+0 (1/0/0/0) Section: Background and Related Work
  60+3+3 (1/1/8/1) Subsection: Long Short-Term Memory
  44+1+0 (1/0/4/1) Subsection: ConvLSTM
  109+3+0 (1/0/19/2) Subsection: Cross Attention Module
  0+2+0 (1/0/0/0) Subsection: Bi-Directional LSTM
  110+4+2 (1/1/2/0) Subsection: Models Related to THST
  36+6+2 (2/1/0/0) Section: Temporal Hierarchical Spatial Transformer
  28+2+0 (1/0/1/0) Subsection: Input Layer
  257+6+5 (3/1/27/0) Subsection: Encoder: Stacked Sparsity LSTM
  164+5+0 (3/0/0/1) Subsection: Decoder and Output
  27+3+0 (1/0/0/0) Section: Bayesian Neural Networ
  0+6+0 (4/0/0/0) Subsection: Results
  69+2+0 (1/0/0/0) Section: Next Steps
  274+15+1 (3/1/2/0) Section: Progress Update
  0+1+0 (1/0/0/0) Chapter: Appendix
  0+1+0 (1/0/0/0) Section: 1
  0+2+0 (1/0/0/1) Subsection: ELBO LOSS
  54+2+0 (1/0/11/0) Subsection: Central Points
  0+0+0 (1/0/0/0) Subsection:
  9+2+0 (1/0/0/0) Section: Word Counts
  12+4+0 (1/0/0/0) Section: Statistics on word cou
```