Temporally Precise Forecasting for State-Based Violent Conflict in Afghanistan

Alan Xiao alanxiao211@gmail.com

Abstract

War and political conflict has been a staple of world development in the 21st century, with millions of lives endangered due to political, religious, or social conflicts. In the post Cold-War era, war and political conflict has been particularly damaging for those living in the Middle East, especially in the country of Afghanistan, where aid groups often fail to catch up with the rising torrent of deadly conflict. Although much research has been published on creating data-based conflict prediction systems for countries in Africa and Asia, the Middle East is often overlooked as a center of conflict for such systems. Therefore, we propose the usage of LSTM networks to produce a conflict prediction system for Afghanistan by forecasting the future estimated death toll due to state-based political conflict in the country. In addition, we propose to perform the forecasting on a higher temporal resolution than done before by predicting the death toll in a time window of weeks rather than months or years. We experiment with multiple traditional time series models before comparing their performances to our LSTM network, and we show that our LSTM network performs reasonably at the task of fine-grained conflict prediction. In addition, we suggest further exploration into the possibilities of fine-grained conflict prediction and point to ways to improve upon the limits of our LSTM model.

Keywords: Time Series Forecasting, ARIMA, LSTM Networks, Deep Learning

1. Introduction

Even in a post-Cold War world, the dream of world peace is still far from reach. Many countries struggle with threats of invasion, civil unrest, and political violence or genocide. These events are destructive because they are often unprecedented; however, many events also follow predictably from previous violence and unrest. The need for aid groups to predict future bouts of political violence in order to efficiently allocate resources gave rise to the field of conflict prediction and the development of conflict prediction systems.

In the past ten years, conflict prediction systems have started to take advantage of machine learning algorithms to make predictions. Machine learning models can predict highly complex patterns in the data compared to traditional statistical techniques (eg. a simple linear regression) [5] and are thus well suited to conflict prediction. In particular, time series forecasting models have been shown to have an even larger advantage when compared to pure regressional models [6].

In this paper, we will apply machine learning methods to conflict prediction in the context of time series forecasting by training and evaluate several time series forecasting models in order to predict the death toll caused by violent conflicts in the country of Afghanistan. We propose a new way to frame the conflict prediction problem by making predictions on more fine-grained time steps (15 day per step) than before. The data used will be extracted from the UCDP-GED (Uppsala Conflict Data Program Georeferenced Event Dataset) dataset, which contains comprehensive conflict data collected from a variety of reliable independent sources. We propose the usage of LSTM networks to effectively fit long sequences of fine-grained conflict data, and we will evaluate the performance of our proposed LSTM network model on the UCDP-GED conflict dataset against a variety of traditional time series forecasting models, such as ARIMA, Linear Exponential Smoothing, and Damped Trend Linear Exponential Smoothing. Moreover, we will discuss the implications of our findings in terms of the feasibility of fine-grained conflict prediction and the viability of LSTM networks for performing fine-grained conflict prediction. Finally, we will point out the possibilities for expanding upon our results.

2. Related Work

The current literature employs both simpler statistical models and more complex deep learning models in making predictions on the time dependent conflict data. Because these models are only trained on data from past conflicts, with no external variables such as socioeconomic indicators, these models must capture complex time-dependent features of conflict in order to make accurate predictions from conflict data alone. Because of this, deep learning models commonly outperform simpler models in terms of MSE. [6] Over long sequences of time-series data, traditional time series methods such as ARIMA and Vector Autoregression usually fail to capture long-term dependencies within the data. [5]

In the current literature, LSTM networks have been shown to be one of the most effective models. [10] LSTMs are a variety of traditional RNN (Recurrent Neural Networks) that can better capture long term patterns and features in a sequence, thus "[remedying] some of the shortcomings of RNNs." [6] Radford et. al. used a variation of LSTMs known as "ConvLSTM" which aimed to capture both the spatial and temporal features of conflict dynamics. [10] In addition to LSTMs, other deep learning methods such as a feed-forward network with hyperparameters tuned by AutoML systems [4] have shown comparable results.

In order to evaluate their models, the current research utilizes a variety of metrics. Some of these include metrics that are standard in the machine learning field, such as MSE (Mean Squared Error). In a paper published by Hegre in 2022, the usage of MSE was described as "standard… for continuous predictions." [13] This reflects the state of the current research; however, there also have been multiple attempts at creating effective non-standard metrics tailored to evaluate time-series models solving this task in particular. For instance, multiple papers in the current research reference the "TADDA (Targeted Absolute Distance with Direction Augmentation)" score [10] [3], which is optimized to "rewards predictions for both matching the sign and mag-nitude of the actual change [in fatalities]." [6], allowing for better modeling of "escalation and de-escalation." The current literature also acknowledges that more research needs to be done to create effective metrics that accurately reflect the ability of a model to "predict different stages of violence… and … reward them for accurately predicting changes in fatalities where they occur rather than zero in peaceful settings." [2] This difficulty in creating an effective metric inevitably stems from the peculiarities of the problem of forecasting time-dependent conflict, which are described above; however, the current literature still acknowledges MSE as a generally effective metric for this problem.

We build upon the current literature by experimenting with new evaluation metrics, namely MAE (Mean Absolute Error), and using state-of-the-art models to solve the conflict prediction problem on a much finer temporal resolution than before. Most previous works attempt to predict the future death toll due to violent conflict for either monthly or yearly [6] intervals, while we attempt predict the same target variable for 15-day timesteps.

3. Methodology

3.1 Process Outline

In order to produce accurate and consistent experimental results, we carefully designed a training and evaluation pipeline to prevent data leakage and ensure consistent evaluation methods for each model. In our overall experimental process, we first train and evaluate our baseline models before using the pipeline to train, fine-tune and evaluate our LSTM against the baseline models. We will elaborate on the details of this outline in the Methodology section. The training and evaluation pipeline is as follows:

1. Preprocess the dataset and format the data appropriately for the fine-grained time series forecasting task.
2. Perform feature engineering on the dataset.
3. Perform a simple train/val/test split on the dataset. For time series forecasting, it is important to split the data by chronological order rather than shuffle the dataset to make sure consecutive timesteps are still sequenced properly.
4. In the case of the deep learning models, each of the train/val/test datasets are split into sequences of length N matched with a scalar label which indicates the value of the target variable at the (N + 1)th index. The sequences are split using a sliding window method with an offset of 1. The datasets of sequence-label pairs are then batched and will later be used for training/evaluation.
5. Fit the training dataset on the model.
6. Make predictions on the evaluation dataset.
7. Tune the model's hyperparameters if applicable.
8. Repeat steps **5** through **7**

The experiments were performed on a desktop computer running Ubuntu 22.0.4 with an Intel i7-8700K processor, 16 gigabytes of RAM, and an NVIDIA GTX 1080 GPU with CUDA support. The experimental code was developed and run using Python 3.10 and the scikit-learn, statsmodels, and Tensorflow 2.10 packages.

3.2 UCDP-GED Dataset

The dataset used is the UCDP-GED dataset (Uppsala Conflict Data Program Georeferenced Event Dataset) [12]. Each entry in the dataset describes one "event". According to the UCDP, an event is defined as "An incident where armed force was used by an organized actor against another organized actor, or against civilians, resulting in at least 1 direct death at a specific location and a specific date." In addition, the UCDP clarifies that "armed force" refers to any lethal use of "manufactured weapons... but also sticks, stones, fire, water, etc." and that "organized groups" refer to organizations that "consciously conduct and plan" events of armed conflict. Moreover, in order for an armed conflict to count as an event, the specific geographical coordinates of the event must be given, along with a specified time period.

The UCDP-GED dataset tracks several features for each event. These include geographical features, such as region, country, and longitude/latitude, and temporal feature, such as date and time. The dataset also specifies the type of violence for each event, which can either be state based, non-state based, or one-sided violence, and the estimated death toll for each event.

3.2.1 Data Analysis and Visualization

In order to effectively train and evaulate models on the dataset, we first analyzed the data to gain an understanding of the target and predictor variables.

We can start with our target variable, which we can visualize using a line plot:

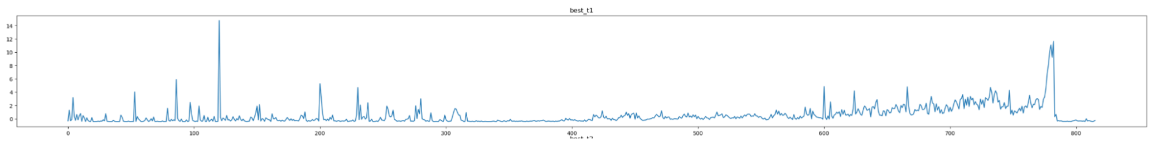


Figure 1: State-Based Death Toll per Timestep

We can notice a few characteristics of the target data. First, there are one or two prominent outliers with spikes indicating timesteps with abnormally high death tolls. We have the choice to either remove or keep these outliers. However, because standardization will decrease the prominence of such outliers and because those outliers represent correct and accurate data, we choose to keep the outliers in our dataset. In addition, the majority of the death toll values seem to be around zero. This indicates that the data is quite sparse, which makes sense, as conflicts are (fortunately) sparse most of the time. This is evident if we visualize the death toll datapoints using a histogram.

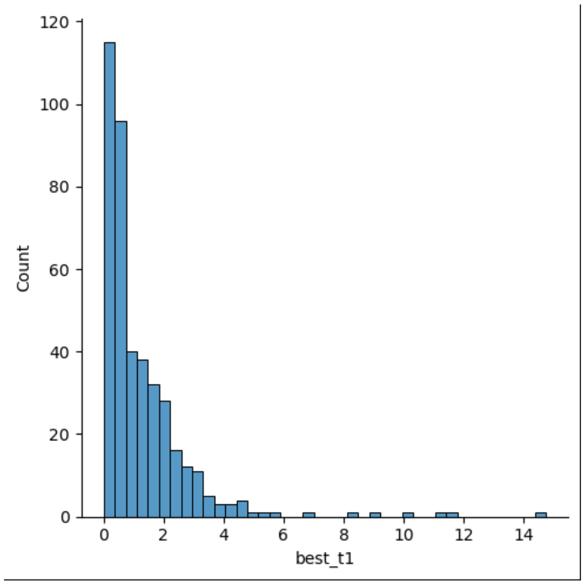


Figure 2: Distribution of Target Variable

Through analysing the data, we also found that several more features were highly correlated to the target variable (state-based violence related death toll). These were specifically the counts and death tolls of state-based, non-state-based, and one-sided violence.

3.2.2 Data Preprocessing

In our experiments, we took a subset of predictor features from the UCDP-GED dataset (a total of 5), in addition to the target feature, to train and evaluate the models. We first reformatted the data into a suitable format for time series analysis by dividing the total time period of the dataset (1989 - 2020) into small 15-day segments. In each time segment, we grouped the events by their type of violence and country. Using these groupings, we then calculated the aggregate sum of the estimated death tolls ("best") and total count of events for each violence type/country cross-section per bimonthly time period. The features of the dataset were then standardized to have a zero mean and unit variance using the scikit-learn library's StandardScaler method.

3.3 Baseline Model Training

After reformatting the dataset and performing basic feature engineering on the reformatted data, we trained 4 traditional time series forecasting models to serve as baseline models for our proposed LSTM network to be evaluated against. The first model is a trivial model that predicts a value of zero for every timestep, while the other 4 are time series regression models commonly used in the existing literature.

3.3.1 Evaluation Metrics

In order to evaluate and compare the performance of our different models, we decided on using two metrics: MAE and MSE.

MAE(Mean Absolute Error) The MAE between the predicted target variable $\hat{y}$ and the target labels y for a series of size N is given as:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |\hat{y} - y|$$

MSE (Mean Squared Error) The MSE between the predicted target variable $\hat{y}$ and the target labels y for a series of size N is given as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (\hat{y} - y)^2$$

3.3.2 Constant Predictor

The trivial constant predictor model simply predicts $y_t = 0$ for all t.

3.3.3 Linear Exponential Smoothing

The linear exponential smoothing model is essentially a moving average model over the second order difference of the target series.

3.3.4 Autoregression Model

Autoregression models use the previous values of the target variable to predict the value of the target at the current timestep through regression. An n^{th} order autoregression would predict the variable y_t using the values $y_{t-1}, y_{t-2} \dots y_{t-n}$ using the following equation:

$$y_t = \phi + \sum_{i=1}^{n} \theta_i y_{t-i}$$

, Where θ_k is the k^{th} model parameter and ϕ is the bias.

3.3.5 Damped Trend Linear Exponential Smoothing

Damped trend LES is similar to normal LES, except that a LES model will typically make predictions based on the assumption of a constant trend in the time series data while a damped trend LES will allow the predicted trend to "dampen" or decay over time.

3.3.6 ARIMA (Autoregressive Integrated Moving Average) Model

The ARIMA model consists both of an autoregressive component, in which target values from the past are used to fit a regression for the next target value, and a moving average component, which performs a weighted sum on past error terms in order to predict the target variable. The parameters of the ARIMA model are estimated using maximum log-likelihood estimation, which essentially attempts to find the model parameters which result in the maximum probability of yielding the given data. Moreover, ARIMA models work primarily on forecasting for stationary time series data, meaning that if the series contains any trend or seasonality, the data has to be preprocessed through differencing or transformations to make the mean and variance roughly constant throughout the entire time interval.

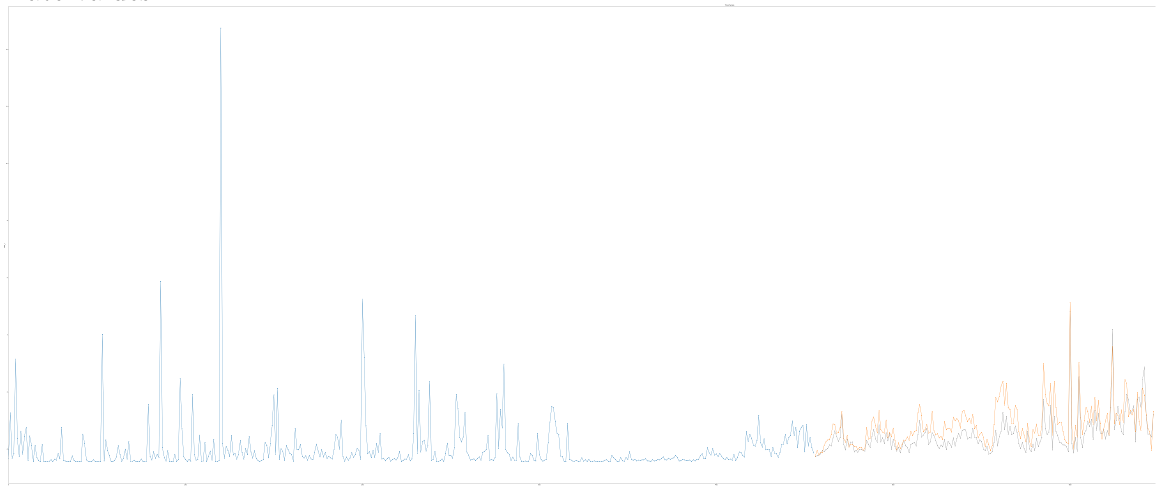
The main hyperparameters of the ARIMA model are p, d, and q, where p is the number of autoregressive terms, d is the order of differencing, and q is the number of moving average terms. In order to determine the p and q terms, we look at the PACF and ACF graphs of the target variable data. The ACF, or autocorrelation function, measures the linear correlation of the time series with the lagged values of itself. The k^{th} autocorrelation coefficient of a time series can be calculated as such:

$$r_i = \frac{\sum_{t=i+1}^{L}(y_t - y_{avg})(y_{t-k} - y_{avg})}{\sum_{t=1}^{L}(y_t - y_{avg})^2}$$

where L is the length of the time series and y_{avg} is the average y value throughout the whole series.

3.3.7 Baseline Evaluation

After training the baseline models, we evaluated each model on the evaluation data with MAE and MSE. Out of all 4 baselines, the Linear Exponential Smoothing model performed the best, with an MSE of 0.25 and MAE of 0.40, while the constant predictor, as expected, performed the most poorly. In order to visualize the effectiveness of the best performing baseline, we superposed graphs of the predicted and original death toll estimates per timestep. The LES model is shown in the graph to reasonably follow the trend of death toll estimate values.



3.4 LSTM Network Training

After training and evaluating the baseline models, we trained and fine-tuned our LSTM network model with the goal of outperforming our baselines on the evaluation data. In particular, we fine-tuned our LSTM model by experimenting with 5 different aspects of the model: model architecture, optimizers, loss functions, activations, and regularization methods.

3.4.1 Advantages of the LSTM

The LSTM [11] is based on the traditional RNN (Recurrent Neural Network) model. Like the traditional RNN, LSTM cells take input from both recurrent connections and new input signals, and just like an RNN unit, the LSTM unit updates its weights using two special forms of backpropagation: backpropagation through time (BPTT) and real time recurrent learning (RTRL). [11] However, the LSTM aims to solve the problem of vanishing gradients that RNNs tend to have over long time series, which is done through the combination of a carefully protected cell state and gates that regulate the modification of the LSTM cell state. This allows the LSTM to have an advantage compared to both statistical models and RNNs when being trained on longer time series datasets. Because of this, we proposed that LSTMs would have an advantage in the task of fine-grained conflict prediction, as the shorter timesteps in the data would cause complex features in the data to be spread over long sequences.

Constant Error Carousel (CEC) In order to solve the vanishing gradient problem in RNN training, LSTMs implement a mechanism called the CEC [11], or Constant Error Carousel. Essentially, the LSTM contains a “memory block” which is guaranteed to yield a constant error flow during backpropagation through recurrent connections to itself.

Consider a memory block b that only is connected to itself through a single recurrent connection. The recurrent connection takes the cell's activation $y_b(t)$ and multiplies it with a weight w, before activating it with a function f to yield $y_b(t+1)$, or the cell's activation at time $t+1$. Thus, the forward step of an isolated memory cell looks like this:

$$y_b(t+1) = f(y_b(t) * w)$$

During backpropagation, the error signal, or the partial derivative of the loss with respect to z, is equal to $\frac{\partial E(t)}{\partial z(t)}$, where $z = y_b(t) * w$, is passed backward to the cell itself through the recurrent connection is given as follows:

$$\frac{\partial E(t)}{\partial z(t)} = \frac{df(z)}{dz} * w * \frac{\partial E(t+1)}{\partial z(t+1)}$$

In order to set $\frac{\partial E(t)}{\partial z(t)} = \textbf{const.}$ for all t, we can simply set the activation to $f(x) = x$ and the weight $w = 1.0$. With this configuration, it is possible for the cell in isolation to have a constant error flow and avoid the vanishing gradient problem.

LSTM Cell Gates In order to be able to use the CEC in practice, the LSTM uses 3 gates to restrict the inflow and outflow of information into each CEC memory cell. [9] These are the input, output, and forget gates. The function of the input gate is to regulate the effects of new inputs on the existing cell state by scaling the relevant and irrelevant signals accordingly, and the output gate regulates the outputted parts of the cell's state in order to protect other cells from irrelevant signals coming from itself. The forget gate, on the other hand, exists to protect the CEC's cell state from accumulating, which could stop the model from learning from new input signals.

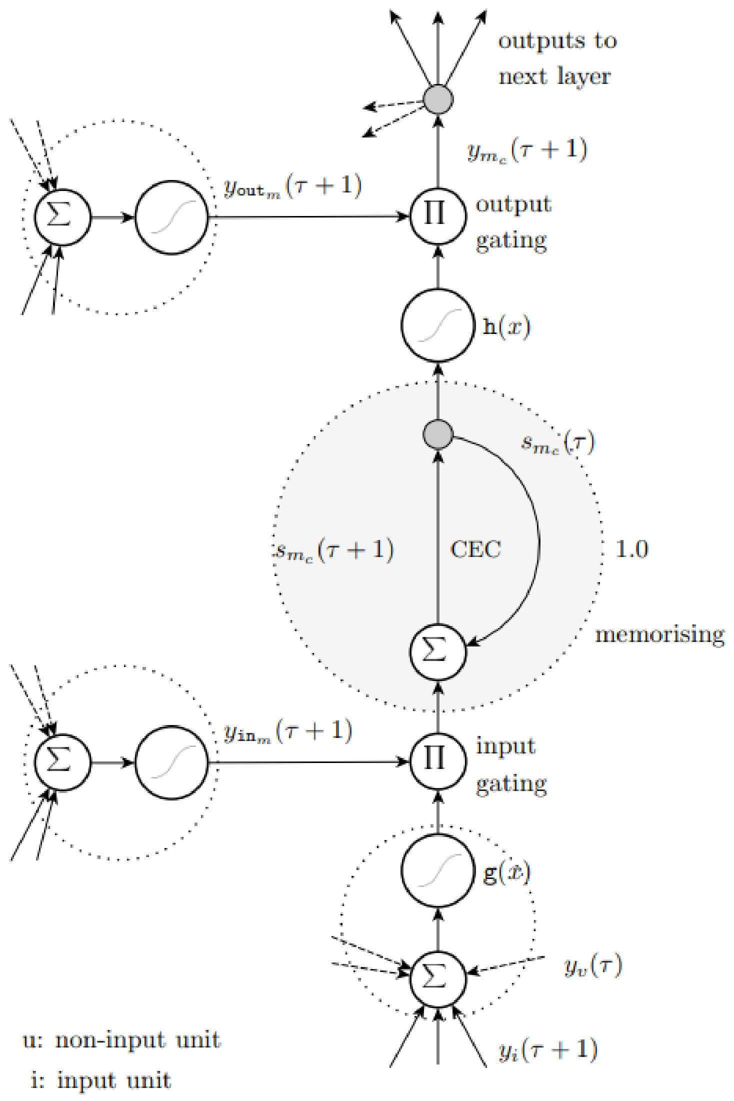


Figure 3: Full LSTM cell layout, with CEC, input/output/forget gates, and recurrent/input connections [11]

Forward Step Each LSTM cell performs the forward step using 3 gates: the input gate, recurrent gate, and output gate. The propagation of signals through the cell is regulated by the three gates, each of which have their own set of input and recurrent weights for the input and recurrent connections of the LSTM cell respectively. The values for the gates are calculated as follows to produce scalar gate values:

$$I(t) = \sigma({W_I}^T x(t) + {R_I}^T y(t-1))$$

$$o(t) = \sigma({W_o}^T x(t) + {R_o}^T y(t-1))$$

$$f(t) = \sigma({W_f}^T x(t) + {R_f}^T y(t-1))$$

As shown above, the values of the gates are produced using the previous cell output and the input at the current timestep. In addition, there are two sets of weights W and R, which are the input and recurrent weights respectively. The function of the three gates will become more clear once the flow of information through the LSTM cell is clarified. Sequential inputs first are combined with results from the previous time step to produce the input activation:

$$z(t) = a({W_z}^T x(t) + {R_z}^T y(t-1))$$

The input activation $z(t)$ is then multiplied with the input gate $I(t)$

Where the activation function a is usually either a sigmoid or tanh function. Other than $y(t-1)$ and $x(t)$, the LSTM cell at timestep t also takes in the previous cell state $c(t-1)$ as input. The cell state stores the features of the data over time, which the cell learns during training. A problem with LSTM cell states is that they tend to grow quickly with respect to the length of time series data, which interferes with future learning. The forget gate $f(t)$ aims to solve this issue by allowing the cell to adaptively reset irrelevant features from its cell state at every timestep so that the cell can both retain long term information and be able to learn new features at the same time. Just like the input gate and input activation, the previous cell state and forget gate are multiplied together element-wise. The two values are then added together to produce the current cell state $c(t)$:

$$c(t) = f(t) \times c(t-1) + I(t) \times z(t)$$

The cell output is then calculated from the cell state by activating $c(t)$ with the cell activation function and multiplying element-wise by the output gate:

$$y(t) = \sigma(t) \times a(c(t))$$

Overall, the six equations listed above constitute the feedforward step of the LSTM. In the case of conflict prediction, the LSTM was chosen due to its ability to retain both long term and short term features of the time series data. In particular, the use of the forget and input gates allow the LSTM to tune which features are relevant for prediction. This allows the LSTM to have a significant advantage over traditional RNN models, which often fail to maintain relevant long-term features when being fit on time series data.

3.5 LSTM Fine-Tuning

In order to achieve the best possible performance with the LSTM network, we experimented with several variables, including loss functions, optimizers, batch sizes, model architecture, and feature engineering.

3.5.1 Model Architecture

Out of all the model sizes we explored, our best-performing LSTM network contained 4 LSTM layers with 1000, 800, 600, and 400 units respectively, plus one dense layer with one unit. Because of the complexity and size of the data, it was estimated that an LSTM network with 2 to 3 layers and about 100 units each would be more than enough. However, it was shown that even larger models had better performance than the estimated model size.

	MAE
1 layer (20)	5.11
1 layer (500)	0.51
2 layers (300, 300)	0.46
3 layers (300, 300, 300)	0.43
4 layers(300, 300, 300, 300)	0.45
4 layers (800, 600, 600, 400)	0.41
4 layers (800, 600, 600, 400) + 1 dense	0.40

Table 1: LSTM Model Sizes

Note that we have not applied any regularization or dropout layers to the LSTM network during this experiment, causing the network to overfit, which is especially prominent with the larger model sizes. In addition, we also found that ”tapering” the layer sizes, making each layer smaller than the last, resulted in better predictive performance than keeping each layer size constant. During this experiment, Each model was trained with default weight and bias initialization, MAE loss, ADAM optimizer with 1e-3 learning rate, no LR decay, and 100 epochs with no early stopping. The initial batch size was set to 32 and N (input length) = 24.

3.5.2 Optimizers

We considered 2 candidates for optimizers for our LSTM: RMSProp [7] and ADAM [8]. In all of our other experiments, we defaulted to using ADAM with a learning rate of 0.001. However, we eventually found through experimentation that our model performed the best with RMSProp at a learning rate of 0.0001. Both ADAM and RMSProp are extensions of the SGD (Stochastic Gradient Descent) algorithm, which is governed by the update equation:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta)$$

where α is the learning rate and θ_t is the set of model parameters at time t. Essentially, SGD estimates the gradient using a set of data points and moves the model parameters in the direction of the negative gradient in order to minimize the loss function.

RMSProp (Root-Mean-Square Propagation) RMSProp is similar to SGD in that the loss gradient is used to update the model parameters; moreover, the RMSProp optimizer attempts to improve upon SGD by implementing a mechanism that can help the model converge quickly on optimal solutions and move slowly towards less certain solutions. The optimizer keeps track of an exponential moving average v of the squared gradients, which is updated as:

$$v_{t+1} = \beta v_t + (1 - \beta) \nabla L(\theta_t)^2$$

The moving average v_{t+1} is then used to update the model parameters by scaling the learning rate by the inverse of the square root of v as given:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_{t+1}}} \nabla L(\theta_t)$$

Adam The ADAM optimizer is similar to RMSProp, except that the optimizer keeps track of an exponential moving average of both the squared gradient and the gradient itself, both of which are used to perform gradient descent in place of the gradient $\nabla L(\theta_t)$.

3.5.3 Loss Functions

The three loss functions we chose to experiment with were MSE (Mean Squared Error), MAE (Mean Absolute Error), and MSLE (Mean Squared Logarithmic Error). We chose MSE because it was already a well established loss function in the conflict prediction literature, and the MAE and MSLE because we theorized they would perform well given their robustness to outliers, which are heavily present in the data.

Loss functions are important because the initial "source" of backpropagation is the estimated gradient $\nabla L(\theta)$ of the loss with respect to the model parameters. In other words, the goal of backpropagation is to optimize the model parameters against some loss function L. A good loss function that allows for the model to converge quickly to an optimal solution is necessary for high predictive performance.

MSLE (Mean Squared Logarithmic Error) The MSLE, much like MAE, tends to be more resilient to outliers and respond more leniently to abnormally large errors. The MSLE between the predicted target variable $\hat{y}$ and the target labels y for a series of size N is given as:

$$MSLE = \frac{1}{N} \sum_{i=1}^{N} (log(1+\hat{y}) - log(1+y))^2$$

Tuning Results We found that the MAE loss function allowed the model to achieve a lower MAE than the other models, though at the expense of the MSE performance (Table 2), which was expected. However, because our primary metric is MAE, meaning we optimize the MAE first before attempting to minimize the MSE, we chose the MAE loss function in our final model.

	MAE	MSE
MAE	0.36	0.58
MSE	0.40	0.48
MSLE	0.43	0.54

Table 2: Loss Functions Comparison

3.5.4 Regularization

In order to regularize the LSTM model, we experimented with using dropout and 3 different types of kernel regularization.

Dropout Dropout helps prevent overfitting by randomly setting to zero a given proportion of the input weight matrix W_I at each timestep. This causes the inputs corresponding to the zeroed weights to be "dropped out", which stops the model from overfitting on the training features.

Kernel Regularization Kernel regularization helps prevent overfitting by placing a penalty on large values in the input weights W_I. This is done by adding a regularization term to the loss function so that the model learns to optimize for lower weight values, which prevents overfitting. We experimented with three different kernel regularization functions: L1L2, L2, and Orthogonal Regularizer.

L2 Kernel Regularization The regularization term of L2 regularization is the sum of the magnitudes of the weights:

$$L_{reg} = \sum_{i=1}^{N} w_i^2$$

When applying to L2 regularization, large weights are punished heavily and the magnitude of weights closer to zero are likely to never reach zero. This is because the derivative of the regularization term with respect to each weight is proportional to the magnitude of the weight, which causes smaller weights to be punished less.

L1L2 Kernel Regularization L1L2 regularization combines the L2 regularization term with the L1 regularization term, which is simply a sum of the magnitudes of the weights. Thus, the L1L2 regularization term is given as:

$$L_R = \sum_{i=1}^{N} w_i + \beta \sum_{i=1}^{N} w_i^2$$

Orthogonal Kernel Regularization Orthogonal regularization [1] works by rewarding weight matrices for being orthogonal, which is achieved when a matrix's transpose is equal to its inverse matrix. Orthogonal regularization is widely used in convolutional networks, where the repeated matrix multiplications with nonorthogonal matrices can often lead to unnaturally large or small signals. The regularization term for orthogonal regularization is given as:

$$L_R = \sum(|WW^T - ID|)$$

Where ID is the identity matrix for W.

Tuning Results The experiments for LSTM regularization were performed with the highest-performing 4-layer LSTM network, MAE loss function, RMSProp optimizer, batch size of 3, and input length of 24. The results showed that a low dropout value of 0.1 combined with L1L2 kernel regularization gave the best results. This matches up with the existing notion in the machine learning literature that high dropout does not work well in LSTM networks [15]. Moreover, the low performance of the L1 kernel regularization may be due to the L1 regularization causing the weights to converge to zero too quickly, causing the model to underfit.

4. Results

The highest performing LSTM network after tuning consisted of 4 layers, with a tapering architecture (800 - 600 - 400 - 200 units), with an additional single-unit dense layer with identity activation. The model was trained with the RMSProp optimizer and MAE loss function, and used a dropout of 0.1 for each LSTM layer, along with L1L2 kernel regularization.

Overall, the LSTM network outperformed every other model in terms of MAE, our main metric (see Table 1). However, the MSE of the LSTM network lagged behind a few of the others, which could indicate that the LSTM network was less robust to outliers and failed to predict the spikes in conflict at several points in the evaluation data, causing the MSE to suffer heavily. This result was surprising; although we expected the LSTM model to be more sensitive to outliers than the autoregressive and exponential smoothing models, we did not expect the other models to outperform the LSTM network in terms of MSE. The huge advantage that the LSTM network has in terms of MAE would also indicate that the discrepancy in MSE must be due to reasons other than poor predictive performance.

	Test MAE	Test MSE
Constant	0.531	0.699
ARIMA	0.424	0.274
Autoregression	0.421	0.270
Linear Exponential Smoothing	0.403	0.257
Damped Trend LES	0.431	0.549
LSTM Network	0.354	0.390

Table 3: Model Comparison

5. Discussion

Our experimental results have several implications for the viability of increasing the temporal resolution in conflict prediction systems. As shown from the initial data analysis, the conflict death toll data for Afghanistan appears to be quite sparse in some areas; with smaller timesteps, the data will become even more sparse as periods of time with zero conflict will contain even more timesteps than before. This unusual characteristic of conflict data presents a serious problem for time series forecasting models - and is amplified when increasing the temporal resolution. However, we still want models to learn and recognize the sparseness of the conflict data when making predictions while not allowing large peaceful periods to cause the models to "forget" long term patterns in the data. From this reasoning, we expected the LSTM model to be more suitable for this task than other time series forecasting models, which was confirmed in the results. The effectiveness of the LSTM model in performing the task of fine-grained conflict prediction shows that effectively performing predictions at a higher temporal resolution is actually possible.

5.1 Prediction Analysis

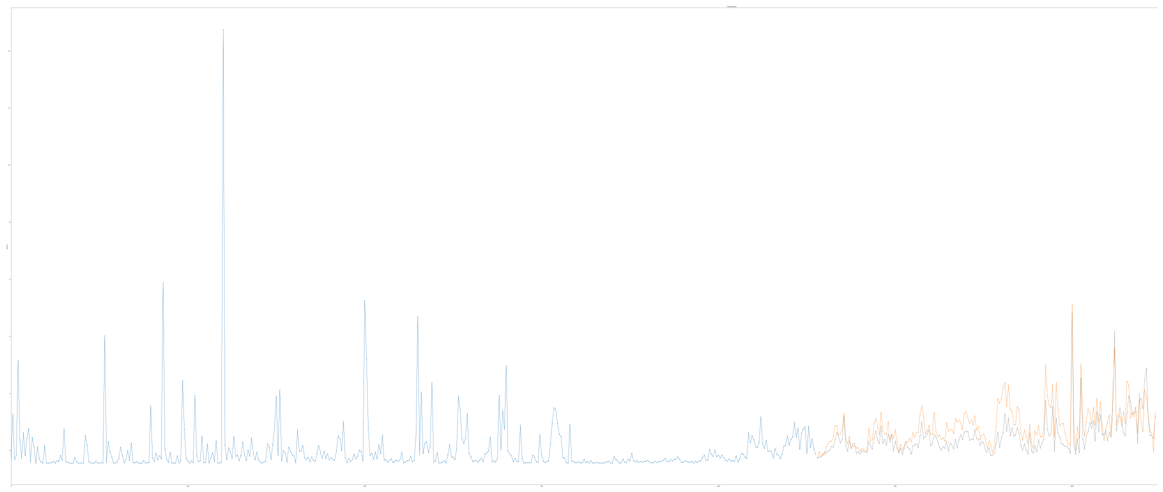


Figure 4: Test Set Predictions (LES)

In the line plot above, the blue part of the line plot represents the training data, while the orange part represents the evaluation data, with the grey line being the predicted data points. From this plot, we see that the LSTM network effectively captures many of the features of the data, as expected. For example, the conflict data does have a significant seasonal component, as the death tolls constantly escalate and de-escalate, peaking periodically.

Moreover, the model also picks up on the trend of the data points in the evaluation set, as the mean of the predicted values over time follows the same pattern as the mean of the label values over time. This was unexpected due to the differences between the trend in the training data versus the evaluation data; due to the drastic change in political climate between the time period corresponding to the training and evaluation sets, the nature of the conflict represented in the data would also differ, which led to different trends in the training and evaluation data. Our LSTM performed well despite this problem, which was unexpected. This discrepancy could also explain the unexpected decrease in performance as we increased the input lengths for LSTM training, because the model would see and overfit on the ”flat” trend of the training data and perform poorly on the evaluation data as a result.

Our model was also shown to be robust to outliers. Removing or capping the outliers actually decreased our performance, which would indicate that the presence of the outliers actually contributed a performance benefit. This could be because of the nature of conflict data, which would inherently have sporadic spikes due to large and sudden outbursts of violence. The LSTM model learned to recognize when and where these ”spikes” could occur, which is shown in the prediction labels when the LSTM model accurately forecasted the two major spikes in violence in the evaluation data.

6. Conclusion and Future Work

By using LSTMs to predict violence conflict in the near future at a finer temporal resolution than done before, we showed that LSTM networks are a viable method of performing time series forecasting even in the case where the data may be particularly sparse or erratic, as conflict data is. We compared the performance of LSTM networks to traditional machine learning models and showed that the LSTM outperformed every other model when evaluated on our main metric of MAE. The strong performance of LSTM networks suggests that it is possible in the future to perform conflict prediction on an even more fine-grained temporal resolution than simply predicting death tolls for monthly or yearly intervals.

Future work in this area could focus on improve upon the limitations of our study. For instance, more feature engineering techniques should be explored in detail and experimented with, as the quality of the data remains a huge limiting factor in conflict prediction performance. More models should also be explored; our strong results with the LSTM imply that other deep learning-based models, such as attention based networks [14], could be effective in fine-grained conflict prediction. Due to time and hardware limitations, we were only able to train smaller-sized LSTM networks, but larger and more complex deep learning models could also be considered in the future.

References

[1] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016.

[2] Lars-Erik Cederman and Nils B Weidmann. Predicting armed conflict: Time to adjust our expectations? *Science*, 355(6324):474–476, 2017.

[3] Thomas Chadefaux. A shape-based approach to conflict forecasting. *International Interactions*, 48(4):633–648, 2022.

[4] Vito D'Orazio and Yu Lin. Forecasting conflict in africa with automated machine learning systems. *International Interactions*, 48(4):714–738, 2022.

[5] Håvard Hegre, Håvard Mokleiv Nygård, and Peder Landsverk. Can we predict armed conflict? how the first 9 years of published forecasts stand up to reality. *International Studies Quarterly*, 65(3):660–668, 2021.

[6] Håvard Hegre, Paola Vesco, and Michael Colaresi. Lessons from an escalation prediction competition. *International Interactions*, 48(4):521–554, 2022.

[7] Jason Huang. RMSProp - Cornell University Computational Optimization Open Textbook - Optimization Wiki — optimization.cbe.cornell.edu. `https://optimization.cbe.cornell.edu/index.php?title=RMSProp`, 2020. [Accessed 29-08-2023].

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Christopher Olah. Understanding lstm networks–colah's blog. *Colah. github. io*, 2015.

[10] Benjamin J Radford. High resolution conflict forecasting with spatial convolutions and long short-term memory. *International Interactions*, 48(4):739–758, 2022.

[11] Ralf C Staudemeyer and Eric Rothstein Morris. Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[12] Ralph Sundberg and Erik Melander. Introducing the ucdp georeferenced event dataset. *Journal of peace research*, 50(4):523–532, 2013.

[13] Paola Vesco, Håvard Hegre, Michael Colaresi, Remco Bastiaan Jansen, Adeline Lo, Gregor Reisch, and Nils B Weidmann. United they stand: Findings from an escalation prediction competition. *International Interactions*, 48(4):860–896, 2022.

[14] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[15] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.