Предсказание уровня воды на реке Амур с помощью sequence to sequence модели с использованием GRU

Конюшенко Иван Денисович
Email: idkonyushenko@gmail.com
Telegram: @idkon
mobile: +79858551757

Абстрактно

В этом документе представлено описание решения к треку NoFloodWithAI: паводки на реке Амур в рамках AI Journey 2020. Решение размещено в рамках номинации AIJ Junior. Некоторые части этого пояснения являются переводом статьи [0].

1. Введение

Изначально проблема сформулирована как необходимость предсказать одномерный временной ряд (значения уровня воды на целевых гидропостах) по входному многомерному ряду признаков. В решении были использованы методы глубокого обучения (Deep Learning). В качестве основной архитектуры была использована модель sequence to sequence (seq2seq). Благодаря своей производительности и крайне малом количестве гиперпараметров модель удобно масштабировать для решения аналогичных задач на других гидрологических постах, а также расширять перечень признаков (фичей).

2. Описание GRU и модели

2.1 Gated Recurrent Units (GRU)

GRU – упрощенная версия LSTM, впервые представленная в [1], относится к классу *gated RNN*, но отличается от LSTM тем, что функции forget gate и input gate заложены в один единственный gate, что позволяет потреблять меньшие, в сравнении с LSTM, ресурсы. На схеме 1 представлена архитектура GRU с одной ячейкой.

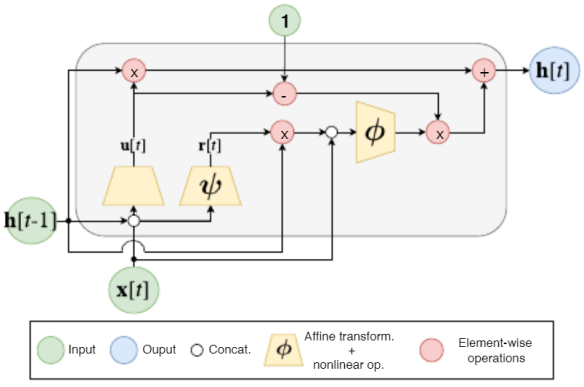


Схема 1

А также ниже приведено описание работы ячейки:

$$\mathbf{u}[\mathbf{t}] = \psi\left(\mathbf{W_u h}[t-1] + \mathbf{U_u x}[t]\right)$$
$$\mathbf{r}[\mathbf{t}] = \psi\left(\mathbf{W_r h}[t-1] + \mathbf{U_r x}[t]\right)$$
$$\widetilde{\mathbf{h}}[\mathbf{t}] = \phi\left(\mathbf{W_c}\left(\mathbf{r}[t] \odot \mathbf{h}[t-1]\right) + \mathbf{U_c x}[t]\right)$$
$$\mathbf{h}[\mathbf{t}] = \mathbf{u}[t] \odot \mathbf{h}[t-1] + (1 - \mathbf{u}[t]) \odot \widetilde{\mathbf{h}}[t],$$

где $\mathbf{W_u}, \mathbf{W_r}, \mathbf{W_c} \in \mathbb{R}^{n_H \times n_H}$, $\mathbf{U_u}, \mathbf{U_r}, \mathbf{U_c} \in \mathbb{R}^{n_H \times d}$ - обучаемые параметры модели, $\psi(\cdot)$ - сигмоида, в то время как $\phi(\cdot)$ может быть любой нелинейной функцией активации (в оригинале – гиперболический тангенс), $\mathbf{u}[t]$, $\mathbf{r}[t]$ - update и reset gates соответственно. GRU был выбран именно по причине того, что в задании требовалась “легкая” в плане потребляемых ресурсов архитектура.

2.2 Seq2seq

Seq2seq модель изначально была разработана для решения проблемы невозможности RNN/LSTM/GRU выдавать последовательность произвольной длины. Модель зарекомендовала себя в разных областях: машинный перевод, распознавание речи, предсказание временных рядов.

Главная идея модели в том, чтобы соединить две нейронные сети и получить энкодер-декодер архитектуру. Первая сеть получает на вход последовательность $\mathbf{x_t} \in \mathbb{R}^{n_T \times d}$ длины n_T, которая является многомерным временным рядом фичей, далее сеть вычисляет фиксированное векторное представление этого ряда $\mathbf{c} = f(\mathbf{x_t}, \mathbf{\Theta}_f), \mathbf{c} \in \mathbb{R}^{d'}$. Это представление обычно называется вектором контекста и в модели является последним скрытым состоянием сети-энкодера или его функцией. Далее сеть-декодер учится выдавать последовательность $\mathbf{\hat{y}_t} \in \mathbb{R}^{n_O}$ длины n_O на основе полученного вектора контекста. Seq2seq является моделью на основе двух рекуррентных сетей, которые обучаются минимизировать функцию ошибки:

$$\mathcal{L}(\mathbf{\Theta}) = \sum_{t=0}^{n_O - 1} (y[t] - \hat{y}[t])^2 + \Omega(\mathbf{\Theta}), \quad \mathbf{\Theta} = [\mathbf{\Theta}_f, \mathbf{\Theta}_g]$$

$$\hat{y}[t] = g(y[t-1], \mathbf{h}[t-1], \mathbf{c}; \mathbf{\Theta})$$

где $\hat{y}[t]$ - предсказание целевого значения для момента времени t, $y[t]$ - реальное целевое значения в момент времени t, $\mathbf{h}[t-1]$ - последнее скрытое состояние декодера в момент времени t-1, $\mathbf{c}$ – вектор контекста, полученный от энкодера из входной последовательности $\mathbf{x_t}$, $\Omega(\mathbf{\Theta})$ - регуляризация параметров модели (так как решение написано на фреймворке *pytorch*, то регуляризация параметров модели не задается явно в функции ошибки, а вшита в оптимизатор и регулируется параметром *weight_decay*). Причина выбора MSE описана в п4. Тренировочная схема для этой модели называется *teacher forcing* [2]. Во время обучения мы используем ground-truth значение *y[t-1]*, чтобы получить значения *h[t]* и $\hat{y}[t]$. Вовремя инференса мы не знаем ground-truth значения, поэтому используем полученные предсказания вместо них и получаем следующую формулу для предсказаний:

Примечание: разница в параметрах декодера при обучении и инференсе рождает дополнительные ошибки. Одним из путей решения является *professor forcing* схема обучения [3], использующая

$$\hat{y}[t] = g(\hat{y}[t-1], \mathbf{h}[t-1], \mathbf{c}; \mathbf{\Theta}).$$

генеративную сеть (GAN), но в данном решении она не будет использоваться.

Seq2seq модель не так нова и уже появились открытые репозитории с имплементированными на фреймворке *pytorch* seq2seq моделями. За основу решения была взята модель из статьи [4], главные идеи которой уходят в решение [5] соревнования на Kaggle. В рамках AIJ-contest были проведены эксперименты для определения оптимальных гиперпараметров модели, обучения, функции ошибки, оптимизатора, параметров входной последовательности. Архитектура модели в решении представлена на схеме 2.

Имплементация позволяет использовать time dependent и категориальные признаки. Таким образом сеть учится на данных сразу всех целевых постов, закодированных с помощью one-hot encoding.

Резюмируя, опишем плюсы данной архитектуры:

- Можно подавать данные любых размерностей, без необходимости переписывать код. Достаточно указать размеры входной матрицы при создании модели.
- Можно выбрать любую длину предсказания, указав нужную при создании модели, без необходимости переписывать какой-либо код.
- Гибкость: сеть легко принимает любые данные, не требует подгонки под конкретный пост.
- Скорость: во время тестов на Google Colab с видеокартой Tesla T4 предсказание ~365 значений с шагом в 10 дней (01-01 -> 01-11 ->…) для каждого из 7-ми целевых постов в сумме заняло 376 миллисекунд (Wall time).

3. Оптимизатор

При решении было протестировано 3 оптимизатора:

- COCOBOptimizer [6]
- AdamW [7]
- ASGD

Наилучшую скорость и сходимость показал AdamW. Именно он и используется в итоговом решении. Модель использует три отдельных оптимизатора: для энкодера, декодера и модели в целом. Такое решение позволяет обеспечить лучшую сходимость.

4. Функция ошибки

В решении используется стандартная MSE, поскольку нам важно, чтобы модель хорошо предсказывала пиковые значения. Так же для сравнения был взят SmoothL1Loss (*torch.nn.SmoothL1Loss()*), который

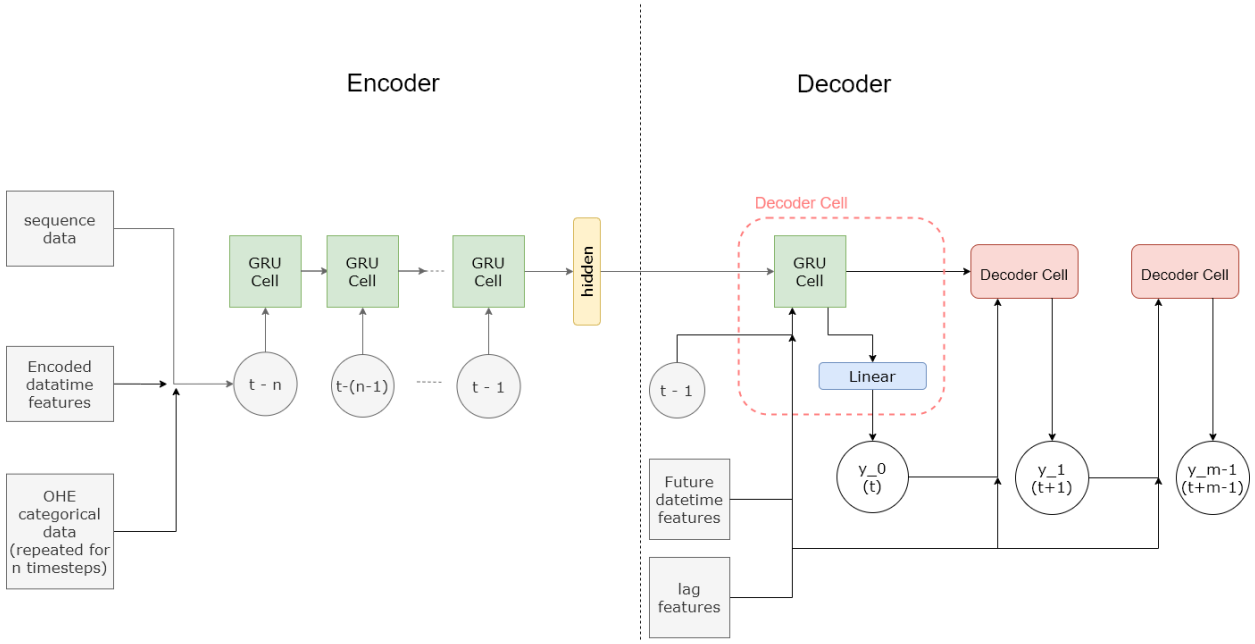


Схема 2

позволяет использовать L1 норму, пока абсолютная поэлементная ошибка выше *beta* и L2 норму в противном случае. Результаты оказались почти одинаковыми, но из-за меньшей параметризации был выбран MSE.

5. Scheduler

Следуя архитектуре статьи [4], в решении использован OneCycleLR [9] (*torch.optim.lr_scheduler.OneCycleLR()*) шедулер отдельно для энкодера и декодера, а так же найден оптимальный *learning rate* для всей модели с помощью *torch-lr-finder.LRFinder()*.

6. Подготовка данных

Предоставленные данные имели достаточно много пропусков. Для корректного обучения и валидации пришлось восстанавливать отдельно каждый ряд. Ниже представлен список примененных методов восстановления рядов, в том порядке, в котором они применялись по мере необходимости:

- Восстановление подстановкой значения соседнего дня, при условии большой корреляции данных соседних дней.
- Восстановление крупных пропусков данными за аналогичный период с соседних гидрологических или метеорологических постов.
- Восстановление подстановкой значения за аналогичную дату соседнего года, с учетом наличия линейной годовой корреляции.
- Использование знаний о физической природе значений ряда.
- Линейная интерполяция, с учетом отсутствия крупных разрывов в данных (заметных при ручном анализе).

Тем не менее не все возможные фичи были использованы. По причине слишком маленького набора некоторых данных.

После восстановления были также добавлены фичи косинусной и синусной меры времени (день и месяц) и направления ветра, подчеркивающие для модели цикличность данных. Подробнее об этом можно прочитать в статье [8]. В случае с датой это помогает обозначить сезонность некоторых данных (например, температуры).

После удаления сильнокоррелированных признаков получаем матрицу корреляции, изображенную на схеме 3 (порог равен 0.3).

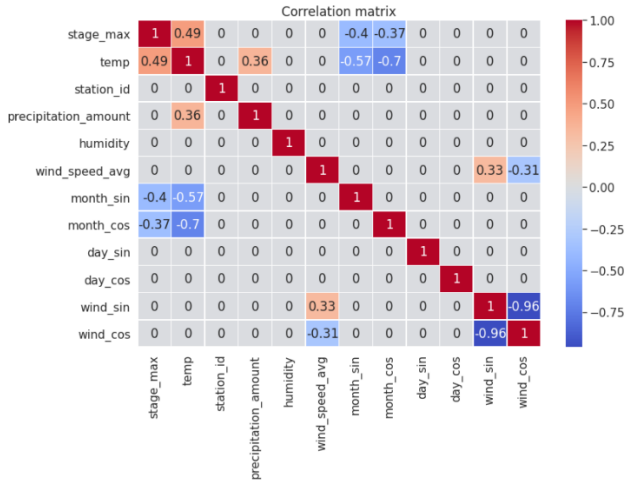


Схема 3

Видно, что все признаки являются информативными и не конфликтуют между собой.

Таким образом итоговая матрица фичей имеет размер *17 x 84219*, включая one-hot encoding колонки 7-ми целевых постов. Более подробное объяснение предобработки данных вы найдете в .ipynb файле построения модели, где можно посмотреть на пропуски и результаты применения методов их заполнения.

Поскольку нейронная сеть требует на вход отнормированные данные, то каждый ряд отдельно приводим к нулевому среднему и единичной дисперсии.

После этого мы создаем датафрейм с последовательностями $\mathbf{x_t} \in \mathbb{R}^{n_T \times d}$, также сопоставляя им векторы целевых значений.

7. Обучение

Во-многом процесс обучения был описан в п2.2. Сеть училась на Google Colab.

Для наглядности модель обучалась на периоде с 1984 года по 2012 включительно, а данные за 2013 год (как один из самых сложных по части предсказания) были взяты в качестве валидационных. Помимо функции ошибки для оценки качества сети использовалась NSE метрика (Nash-Sutcliffe efficiency) (считается как средняя по всем батчам валидационной выборки, что не позволяет считать ее достаточно релевантной, но позволяет видеть примерный вектор обучения).

Было проведено несколько испытаний с разными параметрами сети: 60, 80, 100, 365, 730 дней – длина входной матрицы x_t; 1, 2, 3 или 4 слоя (GRU) в энкодере; включенный и выключенный bidirectional режим.

Финальной, самой лучшей конфигурацией стала следующая:

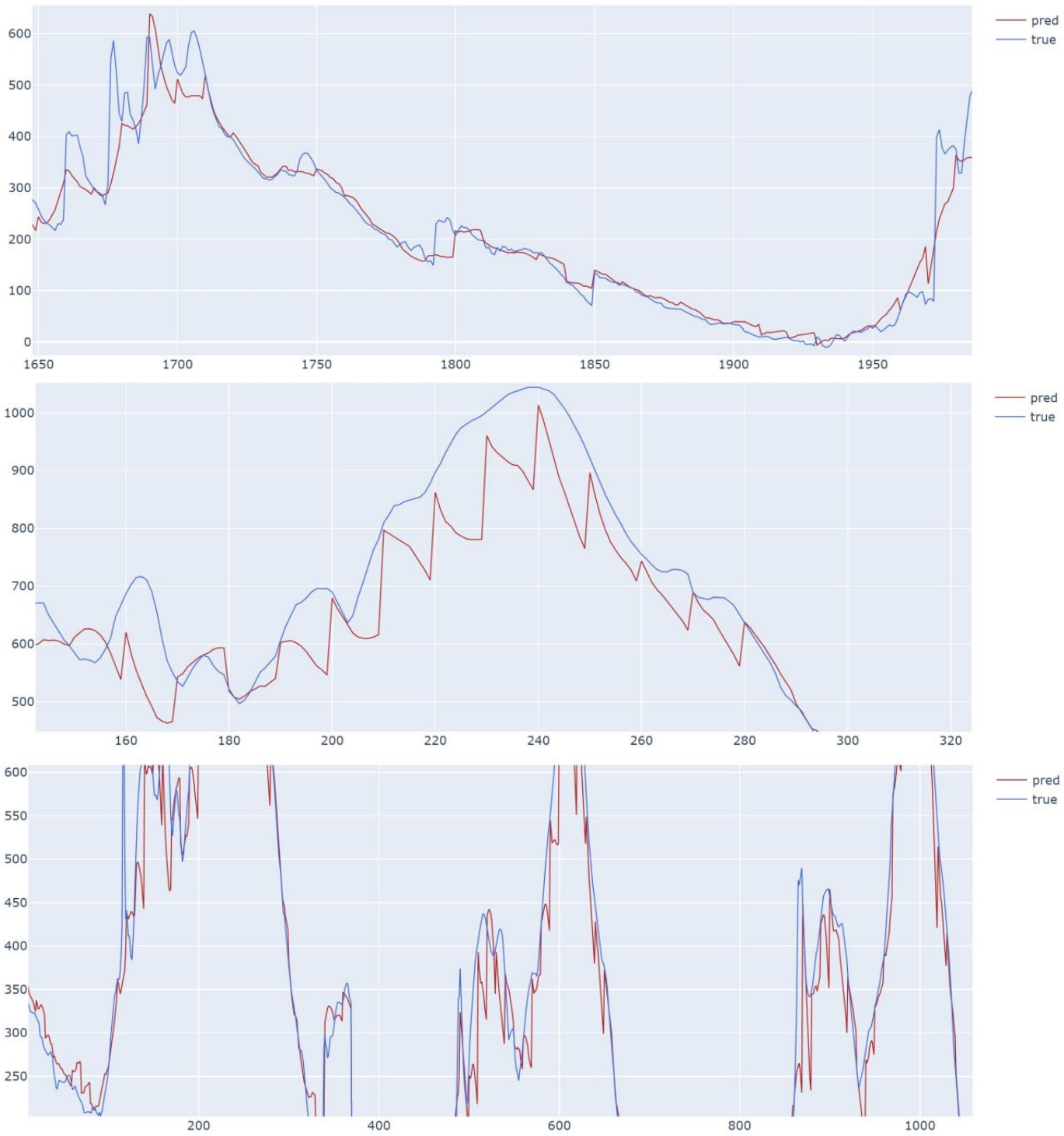
- 80 дней – длина входной матрицы
- 3 GRU в энкодере (выход предыдущего является входом следующего)
- Отключенный bidirectional режим

- Размер батча 256

Сеть обучалась 60 эпох, с сохранением весов за последние 25 эпох. Лучший результат был получен на 22 эпохе. MSE на тесте было около 0.04-0.06, на валидации 0.20. NSE среднее по батчам валидации равно 0.757.

8. Предсказания

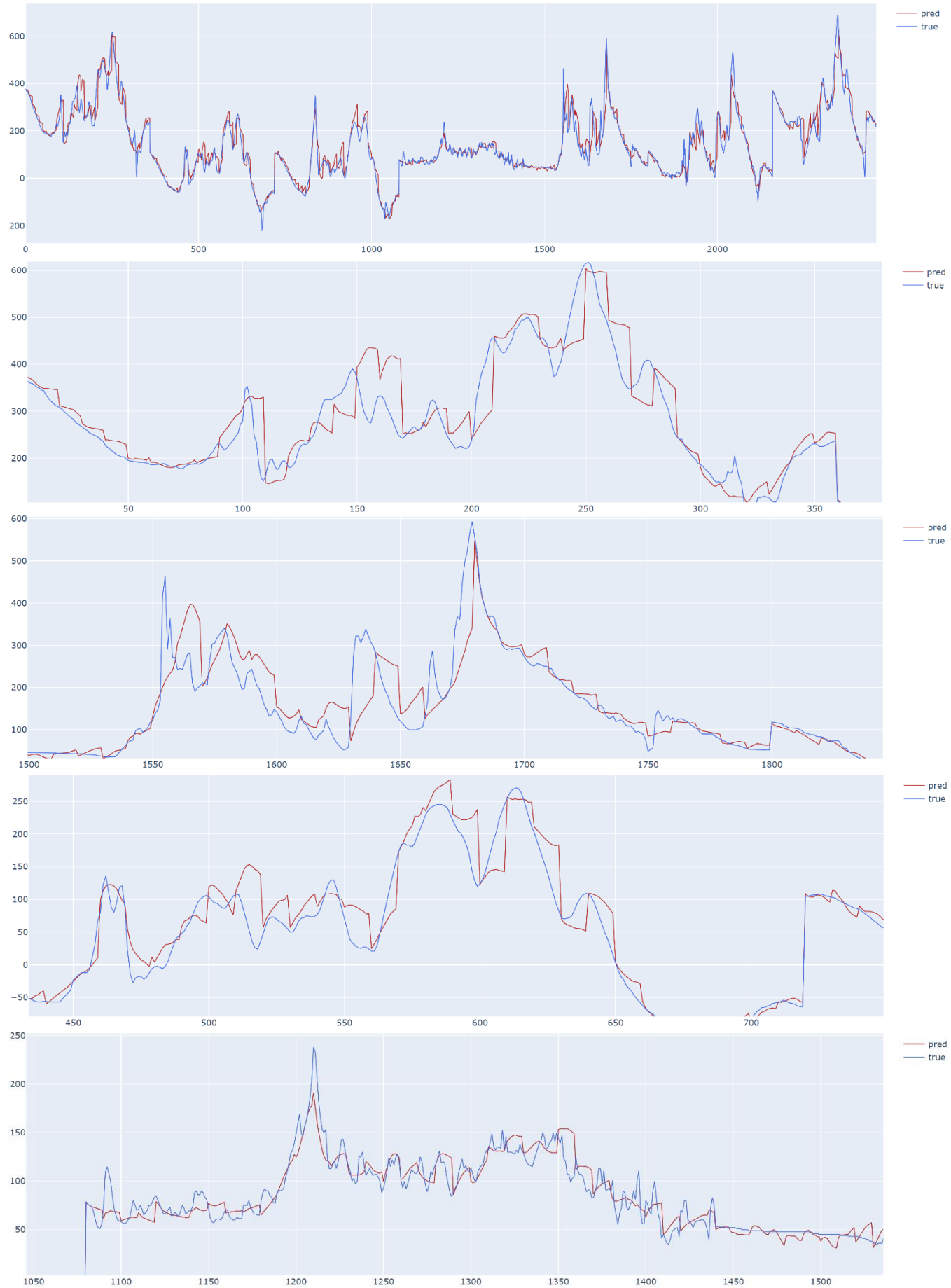
Ниже представлены подробные графики предсказаний на валидации (общий и увеличенные некоторые его части). Построены **на последовательно соединенных данных предсказаний каждого поста.** По оси X отложены дни (365 дней * 7 постов ~ 2500 значений). По оси Y отложены значения максимального уровня воды (см).



Как видно по графикам, модель демонстрирует хорошую производительность в момент скачков уровней воды, хорошо предсказывает значения до 400-450 см, а также дает неплохую статистику по спаду уровня воды даже с пиковых значений в 600-800 см.

К минусам модели можно отнести высокую дисперсию на пиковых значениях (800+ см). В пояснениях к решению [5] автор поднимал эту проблему, советуя применить ансамблирование моделей с разными сидами, а также некоторые другие методы, которые чаще всего хорошо подходят только для Kaggle соревнований. В решении нынешней задачи они применены не были.

Также ниже представлены аналогичные графики для более “спокойного” 2017 года:



По графикам видно, что модель неплохо предсказывает значения разной величины, также ушла большая дисперсия в предсказаниях. Стоит заметить, что в данном случае сеть обучалась дополнительно на 4-х годах, что также позволило улучшить результаты.

9. Итог

Суммируя, модель является гибкой, легко масштабируется, не требует создания специализированных сложных фичей на основе имеющихся данных, имеет хорошую скорость работы, может легко дообучаться с помощью добавления новых фичей и данных за новый период времени.

10. Ссылки

[0] https://www.researchgate.net/publication/334624519_Deep_Learning_for_Time_Series_Forecasting_The_Electric_Load_Case

[1] https://www.researchgate.net/publication/262877889_Learning_Phrase_Representations_using_RNN_Encoder-Decoder_for_Statistical_Machine_Translation

[2] https://www.researchgate.net/publication/2775093_A_Learning_Algorithm_for_Continually_Running_Fully_Recurrent_Neural_Networks

[3] https://www.researchgate.net/publication/309551292_Professor_Forcing_A_New_Algorithm_for_Training_Recurrent_Networks

[4] https://towardsdatascience.com/encoder-decoder-model-for-multistep-time-series-forecasting-using-pytorch-5d54c6af6e60

[5] https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/43795

[6] https://arxiv.org/abs/1705.07795

[7] https://arxiv.org/abs/1711.05101

[8] https://ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/

[9] https://arxiv.org/abs/1708.07120