Лабораторная работа №2.

Данные на входе сети: отзывы к сервису

Решаемая задача: анализ тональности

Тип сети: LMU-model

LMU - сеть типа RNN. Т.к. на семинаре была тема, где как раз рассматривался данный тип сетей, буду рассматривать различия м/у LMU и LSTM.

Архитектура:

- LSTM (Long Short-Term Memory):
  - включает механизм управления памятью с помощью трёх ворот: входных, забывания и выходных. Эти ворота позволяют регулировать поток информации, что помогает модели запоминать или забывать нужные данные на протяжении времени.
  - Содержит 2 основных состояния: скрытое состояние (h_t) и состояние ячейки (C_t), которые хранят долгосрочную память и помогают помнить информацию на нескольких временных шагов.

- LMU (Legendre Memory Unit):

  – в отличие от LSTM, LMU не использует ворота. Вместо этого он опирается на вектор памяти (m_t) и вектор состояния (h_t), которые вместе позволяют эффективно хранить информацию.

1) RNN берёт свой результат и процессует его же на вход на следующем шаге. Из-за этого на большом временном промежутке было сложно их обучать как раз из-за того, что много ресурсов уходило на создание "памяти".

2) Позже появилась LSTM, которая запоминала информацию до определённого момента, после чего новые данные начинали перезаписывать старые в зависимости от типа требуемых данных.

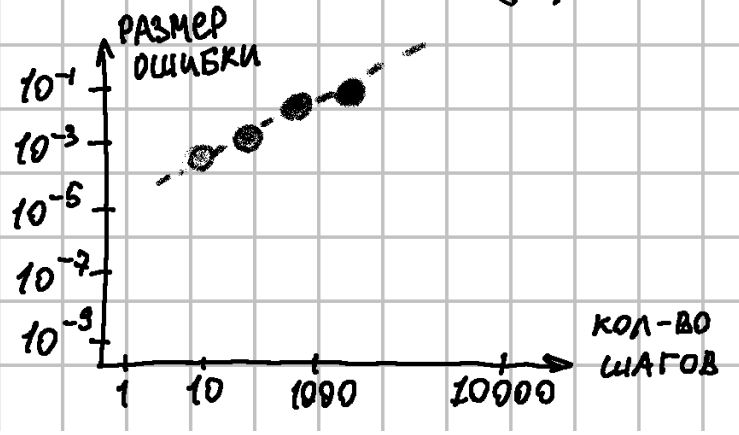
Этой сетью пользовались до 2010-2014 годов, после этого появились трансформеры, которые решили ряд проблем LSTM:

– сложность обучения на большом кол-ве данных, что критично, например, для NLP.

– данные в LSTM нужно обрабатывать последовательно

– LSTM-чёрный ящик, т.к. точные вычисления внутри сети полностью не известны.

График средне-квадратичной ошибки для LSTM:



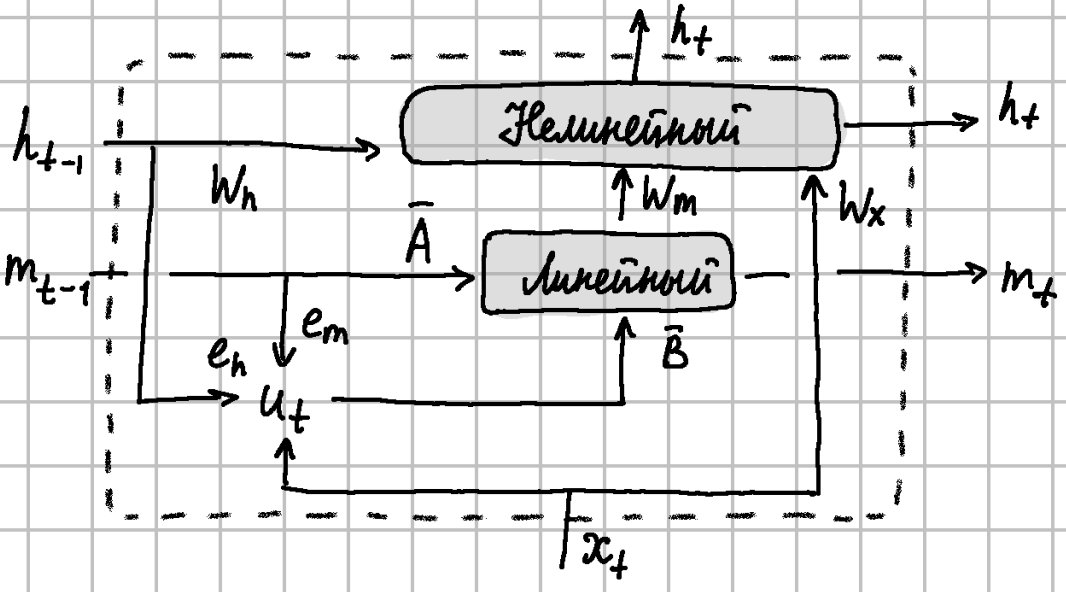
=> к моменту, когда кол-во шагов достигнет 1000, ошибка уже будет большой.

3) Трансформер обрабатывает все данные сразу, пропуская их через сеть прямого распространения, чтобы искать зависимость м/у разными (!)(!) данных. Т.к. данные обрабатываются параллельно => можно использовать GPU => скорость обработки ↑ в разы.

4) LMU - есть входной сигнал, который мы хотим в точности воспроизвести через некоторое время, например, 1 секунду.

Задача явл. бесконечно сложной, т.к. сигнал не ограничен =>

=> ученые интерпретировали задачу т.о., чтобы максимально приблизиться к оптимальному решению при произвольной задержке. Этим как раз и занимается блок Легандра.



• Основные компоненты блока Легандра:

1) Память m_t: Это вектор, который хранит информацию о предыдущих временных шагах. Он обновляется

на каждом шаге и представляет собой "окно" с проекцией прошлого на базис полинома Лежандра.

2) Состояние h_t: Это вектор состояния, который используется для передачи информации от одного временного шага к следующему. Оно также взаимодействует с памятью и обновляется каждый раз с помощью нелинейного преобразования.

3) Матрицы A и B: Эти матрицы служат для линейного обновления памяти. Они управляют тем, как текущий входной сигнал и предыдущая память комбинируется в новом состоянии памяти m_t.

4) Веса W_h, W_m, W_x: Эти веса регулируют связи м/у различными состояниями и входами, позволяя управлять тем, как информация из памяти и входных данных комбинируется и передаётся дальше.

• Механизм работы:

1) Линейное обновление памяти:
   - На каждом временном шаге вектор памяти m_t обновляется с помощью линейного преобразования. Сначала берётся предыдущая память m_{t-1} и к ней применяется матрица A, которая задаёт динамику памяти.
   - К результату добавляется вклад от текущего входного сигнала x_t, умноженного на матрицу B, чтобы новая информация интегрировалась в память. Это обновление позволяет эффективно "запоминать" данные из прошлых временных шагов.

2) Проекция на полиномы Лежандра:

- Полиномы Лежандра помогают представить длинные последовательности так, чтобы память могла "удерживать" информацию на больших временных отрезках. Проекция на полиномы позволяет памяти m_t хранить сжатую информацию о прошлом, делая блок Лежандра более устойчивым к эффекту затухающих градиентов.

3) Нелинейное обновление состояния:

- После того, как обновилась память m_t, вектор h_t вычисляется с учётом памяти и предыдущего состояния h_{t-1}. Этот процесс включает нелинейное преобразование, управляемое весами W_h и W_m, что позволяет моделировать сложные зависимости.
- В итоге, h_t содержит обработанную информацию из памяти и состояния, которая затем передаётся на следующий временной шаг.

- HIPPO: High-order Polinomial Projection Operator.

* Статья, написанная учёными из Стенфордского университета, в ней была улучшена работа LMU, разработана более гибкая и масштабируемая модель памяти HIPPO-LegS.

1) Основная идея HIPPO: HIPPO решает задачу представления последовательности данных с помощью проекций на полиномы. В отличие от классической LMU, HIPPO не фиксирует временное окно для сохранения данных, а позволяет динамически изменять его.

2) Улучшение по сравнению с LMU: В LMU, для хранения прошлых состояний используется фиксированное окно, что ограничивает возможности модели для задач с изменяющимися временными масштабами. HIPPO решает эту проблему с помощью метода "масштабированной памяти" (HIPPO-LegS), который распределяет вес для всей истории данных, адаптируясь к различным временным отрезкам. Это улучшает долговременную память и делает модель устойчивой к смене темпов поступления данных.

- обновление состояний требует меньше выч. ресурсов
- HIPPO-LegS уменьшает проблему "затухающих градиентов", что делает обучение более стабильным и позволяет работать с длинными п-тями.

⇒ Для решения задачи будет использоваться именно эта версия модели LMU.