Давайте подробно разберем процесс “смешивания текущего входа с предыдущим состоянием” в блоке RWKV в контексте покерной игры. Я объясню это на конкретном примере.

Пошаговое объяснение процесса смешивания в RWKV_Block

В архитектуре RWKV смешивание текущего входа с предыдущим состоянием происходит с помощью специального механизма, который позволяет модели сохранять информацию о прошлых ходах, но при этом обновлять её с учетом нового входа.

Ключевые уравнения в блоке RWKV

В коде эти операции выглядят так:

```
        k = self.key(xt * self.time_mix_k + self.state * (1 - self.time_mix_k))
        v = self.value(xt * self.time_mix_v + self.state * (1 - self.time_mix_v))
        r = torch.sigmoid(self.receptance(xt * self.time_mix_r + self.state * (1 - self.ti

        # Обновление состояния
        self.state = xt + self.state * torch.exp(-torch.exp(self.time_decay))

        # Вычисление выхода
        out = r * self.output(v)
```

Пример в контексте покера

Представим конкретную ситуацию в покере:

Предыдущие действия: - Игрок в позиции UTG сделал рейз размером 3BB - Вы на позиции Button со средней парой и решаете сделать колл

Текущий ход: - Теперь действие на вас снова после флопа (появились три общие карты) - Ваш оппонент сделал продолжительную ставку размером 5BB - Вы должны решить: фолд, колл или рейз

Шаг 1: Подготовка входных данных

Входными данными в этой ситуации будут векторы признаков, включающие: - Ваша текущая рука (средняя пара) - Карты на столе (флоп) - Размер ставки оппонента (5BB) - Размер банка (около 7.5BB) - Ваша позиция (Button) - Позиция оппонента (UTG) - Размер стека (допустим, 100BB) - Предыдущие действия (оппонент сделал рейз до флопа, вы сделали колл)

Шаг 2: Смешивание через механизм “key-value”

Модель имеет “состояние”, которое содержит информацию обо всех предыдущих действиях: - Вектор состояния кодирует, что оппонент агрессивно разыгрывал руку - Также содержит информацию о том, что вы играли пассивно (только колл)

Когда происходит новый ход, модель выполняет следующие операции:

1. **Формирование ключа (k)**:

   ```
   k = self.key(xt * self.time_mix_k + self.state * (1 - self.time_mix_k))
   ```

   В этом уравнении:

- `xt` - вектор текущего хода (новая ставка оппонента 5BB на флопе)
- `self.state` - предыдущее состояние (информация о предфлопе)
- `time_mix_k` - вес, определяющий баланс между новой и старой информацией

Например, если агрессивная игра оппонента особенно важна, `k` будет выделять этот аспект.

2. **Формирование значения (v)**:

```
v = self.value(xt * self.time_mix_v + self.state * (1 - self.time_mix_v))
```

Здесь формируется “значение” - вектор, содержащий информацию о том, как следует реагировать. Например, знание о том, что с вашей средней парой против агрессивной ставки обычно лучше не продолжать.

3. **Вычисление рецептивности (r)**:

```
r = torch.sigmoid(self.receptance(xt * self.time_mix_r + self.state * (1 - self.time_mix_r)))
```

“Рецептивность” определяет, насколько модель должна учитывать новую информацию. Если ставка оппонента в 5BB типична для его агрессивного стиля, рецептивность может быть низкой, потому что это согласуется с предыдущим поведением. Если ставка неожиданно мала, рецептивность может быть высокой, указывая на изменение в поведении оппонента.

“Рецептивность и реактивность — два условия, которые необходимы для квалификации познавательного процесса или механизма как обладающего адекватной чувствительностью к причинам. Рецептивность здесь означает распознаваемость причин, а реактивность — способность реагировать на них” “РЕЦЕПТИВНОСТЬ — восприимчивость, а также и само состояние восприятия; способность получить представление благодаря воздействию на нас предмета.”

Шаг 3: Обновление состояния

```
self.state = xt + self.state * torch.exp(-torch.exp(self.time_decay))
```

Здесь происходит обновление “памяти” модели: - Новая информация (`xt`) добавляется напрямую - Старое состояние (`self.state`) умножается на коэффициент затухания - `time_decay` определяет, насколько быстро “забывается” старая информация

В нашем примере, если оппонент ранее показывал агрессию, но сейчас сделал странно маленькую ставку, `time_decay` может дать больший вес новой информации, снижая влияние предыдущих наблюдений.

Шаг 4: Формирование выхода

```
out = r * self.output(v)
```

Выход формируется как произведение рецептивности и трансформированного значения: - Если рецептивность высока (новая информация важна), выход будет ближе к трансформированному новому значению - Если рецептивность низка (старая информация важнее), выход будет менее подвержен влиянию нового входа

В нашем примере, если модель обнаружила, что оппонент обычно делает агрессивные ставки с сильными руками, а на флопе с потенциально опасными картами делает типичную ставку, выход может склонять модель к фолду со средней парой.

Практическое значение в покере

В контексте покера этот механизм позволяет модели:

1. **Отслеживать тенденции**: Если оппонент часто блефует, модель “запоминает” это через состояние и может рекомендовать колл чаще
2. **Адаптироваться к изменениям**: Если игрок, обычно играющий тайтово, внезапно начинает играть агрессивно, механизм смешивания позволяет гибко обновить представление о нем
3. **Обрабатывать последовательности**: Каждое действие в раздаче (префлоп → флоп → тёрн → ривер) влияет на последующие решения; смешивание позволяет сохранять эту информацию
4. **Учитывать контекст**: Одно и то же действие (например, ставка в полбанка) может иметь разное значение в зависимости от предыдущих ходов и текущей ситуации

По сути, смешивание входа и состояния имитирует то, как опытный игрок в покер использует как общие покерные знания, так и специфичные наблюдения о конкретной игре и оппонентах, чтобы принимать более информированные решения.