НЕЙРОННЫЕ СЕТИ, ПРЕДЛИНЗЫ И ТРОЙНЫЕ МОДУЛИ ТАМБАРЫ

Bartosz Milewski

Перевод:
Геннадий Чернышев
(https://henrychern.wordpress.com/)

1 Введение

Нейронные сети являются примером компонуемых систем, поэтому неудивительно, что их можно моделировать с помощью теории категорий, которая является высшим средством композиционности. Более того, категорные идеи, лежащие в основе нейронных сетей, могут быть немедленно реализованы и протестированы на языке программирования. Сначала, я приведу реализацию параметрических линз на Haskell, обобщу их до пред-линз и представлю их профункторное представление. Используя представление профунктора, я построю работающий многослойный перцептрон.

Далее я представлю бикатегорию **PreLens** пред-линз и бикатегорию тройных профункторов Тамбары и покажу, как они связаны с пред-линзами.

2 Реализация на Haskell

Каждый компонент нейронной сети можно рассматривать как систему, преобразующую входные данные в выходные, действие которой зависит от некоторых параметров. На языке нейронных сетей это называется *прямым проходом*. Такой компонент принимает набор параметров

`p`, объединяет их с входными данными `s` и выдает выходные данные `a`. Его можно описать следующей функцией, реализованной на Haskell:

```
fwd :: (p, s) -> a
```

Но настоящая сила нейронных сетей заключается в их способности обучаться, учитывая ошибки. Если нам не нравится результат работы сети, можно подтолкнуть ее к лучшему решению. Если мы хотим изменить результат на некоторую величину `da`, какое изменение `dp` в параметрах следует внести? *Обратный проход* распределяет ответственность за предполагаемую ошибку прямо пропорционально влиянию каждого параметра на результат.

Поскольку нейронные сети состоят из слоев нейронов, каждый из которых имеет свои собственные множества параметров, можно также задать вопрос: какое изменение `ds` во входных данных этого слоя (которые являются выходными данными предыдущего слоя) мы должны внести, чтобы улучшить результат? Затем мы могли бы передать эту информацию обратно на предыдущий уровень и позволить ему настроить свои собственные параметры. Обратный проход можно описать другой функцией на Haskell:

```
bwd :: (p, s, da) -> (dp, ds)
```

Комбинация приведенных двух функций образует *параметрическую линзу*:

```
data PLens a da p dp s ds =
     PLens { fwd :: (p, s)     -> a
           , bwd :: (p, s, da) -> (dp, ds) }
```

В этом представлении не сразу понятно, как компоновать параметрические линзы, поэтому я собираюсь привести и другие представления, которые могут быть более удобными в некоторых приложениях.

2.1. Экзистенциальная параметрическая линза. Заметим, что обратный проход повторно использует аргументы `(p, s)` прямого прохода. Хотя некоторая информация из прямого прохода необходима для обратного прохода, не всегда ясно, что она необходима вся. Для прямого

прохода имеет смысл создать некий пакет услуг, который будет доступен при обратном проходе. В простейшем случае этот пакет будет просто парой `(p, s)`. Но, с точки зрения пользователя линзы, точный тип этого пакета является внутренней деталью реализации, поэтому можно, с таким же успехом, скрыть его как экзистенциальный тип `m`. Таким образом, приходим к более симметричному представлению:

```
data ExLens a da p dp s ds =
  forall m . ExLens ((p, s)  -> (m a)
                     ((m, da) -> (dp, ds))
```

Тип `m` часто называют *остатком* линзы.

Эти экзистенциальные линзы могут быть скомпонованы последовательно. Результат композиции параметризуется произведением (кортежем) исходных параметров. Мы увидим это более явно далее.

Но поскольку произведение типов ассоциативно только с точностью до изоморфизма, композиция параметрических линз ассоциативна, также, лишь с точностью до изоморфизма.

Также имеется тождественная линза:

```
identityLens :: ExLens a da () () a da
identityLens  = ExLens id id
```

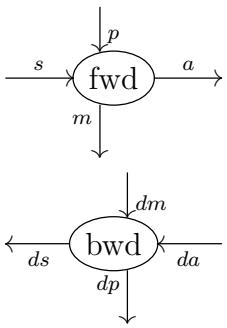
но, опять же, законы категорного тождества выполняются только с точностью до изоморфизма. Вот почему параметрические линзы нельзя интерпретировать как hom-множества линз в традиционной категории. Вместо этого, они являются частью бикатегории, возникающей из конструкции **Para**.

2.2. Пред-линзы. Отметим, что по-прежнему существует асимметрия в трактовке параметров и остатков. Параметры накапливаются (объединяются) в процессе композиции, а остатки прослеживаются (категорно, экзистенциальный тип описывается ко-концом, который представляет собой обобщенный след). Нет причин, по которым не следует накапливать остатки в процессе композиции и откладывать фиксацию следа до самого конца.

Таким образом, приходим к полностью симметричному определению пред-линзы:

```
data PreLens a da m dm p dp s ds =
     PreLens ((p, s)   -> (m, a))
             ((dm, da) -> (dp, ds))
```

Теперь имеются два отдельных типа: `m`, описывающий остаток, и `dm`, описывающий изменение остатка.



Если все, что требуется в конце, — это проследить остатки, мы идентифицируем два типа.

Заметим, что роль параметров и остатков при прямом и обратном проходах меняется на противоположную. Прямой проход с учетом параметров и входных данных выдает выходные данные плюс остаток. Обратный проход отвечает на вопрос: как следует подправить параметры и входные данные `(dp, ds)`, если требуется, чтобы остатки и выходные данные изменились на значения `(dm, da)`. В нейронных сетях это будет рассчитываться с использованием градиентного спуска.

Композиция пред-линз аккумулирует, как параметры, так и остатки, в кортежи:

```
preCompose ::
   PreLens a' da' m dm p dp s ds ->
   PreLens a da n dn q dq a' da' ->
   PreLens a da (m, n) (dm, dn) (q, p) (dq, dp) s ds
preCompose (PreLens f1 g1) (PreLens f2 g2) = PreLens f3 g3
  where
    f3 = unAssoc . second f2 . assoc . first sym .
         unAssoc . second f1 . assoc
```

```
    g3 = unAssoc . second g1 . assoc . first sym .
         unAssoc . second g2 . assoc
```

Мы используем ассоциаторы и симметризаторы для перестановки различных кортежей. Обратите внимание на разделение проходов вперед и назад. В частности, обратный проход скомпонованное линзы зависит только от обратных проходов составных линз.

Также имеется тождественная пред-линза:

```
idPreLens :: PreLens a da () () () () a da
idPreLens  = PreLens id id
```

Таким образом, пред-линзы образуют бикатегорию, объединяющую конструкции **Para** и **coPara** в одну.

В этой категории также существует моноидальная структура, вызванная параллельной композицией. При параллельной композиции мы объединяем соответствующие входные и выходные данные, а также параметры и остатки, как в прямом, так и в обратном проходах.

Экзистенциальную линзу можно получить из пред-линзы, в любой момент прослеживая остатки:

```
data ExLens a da p dp s ds =
  forall m. ExLens (PreLens a da m m p dp s ds)
```

Однако, заметим, что трассировку можно выполнить *после* того, как будут сформированы все (последовательные и параллельные) композиции. В частности, можно было бы выделить один конвейер для выполнения прямых проходов, сбора параметров и остатков, а затем отправить эти данные в другой конвейер, который выполняет обратные проходы. Данные производятся и потребляются в порядке LIFO.

2.3. Пред-нейрон. В качестве примера, реализуем базовый строительный блок нейронных сетей — нейрон. В дальнейшем, нам понадобятся следующие синонимы типов:

```
type D = Double
type V = [D]
```

Нейрон можно разложить на три мини-слоя. Первый слой — это линейное преобразование, которое вычисляет скалярное произведение входного вектора и вектора параметров:

$$a = \sum_{i=1}^{n} p_i \times s_i \,.$$

Он также создает остаток, который, в данном случае, состоит из кортежа входных данных и параметров, `(V, V)`:

```
fw       :: (V, V) -> ((V, V), D)
fw (p, s) = ((s, p), sumN n $ zipWith (*) p s)
```

Обратный проход имеет общую сигнатуру:

```
bw :: ((dm, da) -> (dp, ds))
```

Поскольку, в конечном итоге, потребуется отслеживать остатки, будем использовать тот же тип для `dm`, что и для `m`. И поскольку мы собираемся выполнять арифметические действия над параметрами, повторно используем тип `p` для дельта `dp`. Таким образом, сигнатура обратного прохода есть:

```
bw :: ((V, V), D) -> (V, V)
```

На обратном проходе, закодируем градиентный спуск. Направление и наклон самого крутого градиента определяются частными производными:

$$\frac{\partial a}{\partial p_i} = s_i \,, \qquad \frac{\partial a}{\partial s_i} = p_i \,.$$

Умножаем их на желаемое изменение выходного сигнала `da`:

```
dp = fmap (da *) s
ds = fmap (da *) p
```

Получившаяся линза — это:

```
linearL :: Int -> PreLens D D (V, V) (V, V) V V V V
linearL n = PreLens fw bw
  where
    fw       :: (V, V) -> ((V, V), D)
    fw (p, s) = ((s, p), sumN n $ zipWith (*) p s)
    bw             :: ((V, V), D) -> (V, V)
    bw ((s, p), da) = (fmap (da *) s
                      ,fmap (da *) p)
```

За линейным преобразованием следует смещение, которое использует одно число в качестве параметра и не создает остатка:

```
biasL :: PreLens D D () () D D D D
biasL  = PreLens fw bw
  where
    fw        :: (D, D) -> ((), D)
    fw (p, s)  = ((), p + s)
    -- da/dp = 1, da/ds = 1
    bw        :: ((), D) -> (D, D)
    bw (_, da) = (da, da)
```

Наконец, реализуем слой нелинейной активации, используя функцию `tanh`:

```
activL :: PreLens D D D D () () D D
activL  = PreLens fw bw
  where
    fw (_, s)  = (s, tanh s)
    -- da/ds = 1 + (tanh s)^2
    bw (s, da) = ((), da * (1 - (tanh s)^2))
```

Нейрон с m входами представляет собой композицию трех компонентов по модулю некоторых моноидальных перестановок:

```
neuronL :: Int ->
    PreLens D D ((V, V), D) ((V, V), D) Para Para V V
```

```
neuronL mIn = PreLens f' b'
  where
    PreLens f b =
      preCompose (preCompose (linearL mIn) biasL) activL
    f'                 :: (Para, V) -> (((V, V), D), D)
    f' (Para bi wt, s) = let (((vv, ()), d), a) =
        f (((), (bi, wt)), s)
                           in ((vv, d), a)
    b'                 :: (((V, V), D), D) -> (Para, V)
    b' ((vv, d), da)    = let ((), (d', w')), ds) =
        b (((vv, ()), d), da)
                           in (Para d' w', ds)
```

Параметры нейрона удобно упаковать в одну структуру данных:

```
data Para = Para { bias   :: D
                 , weight :: V }
mkPara (b, v) = Para b v
unPara p      = (bias p, weight p)
```

Используя параллельную композицию, можно создавать целые слои нейронов, а затем использовать последовательную композицию для моделирования многослойных нейронных сетей. Функция потерь, которая сравнивает фактический результат с ожидаемым результатом, также может быть реализована в виде линзы. Эту конструкцию мы будем использовать позже, используя профункторное представление.

2.4. Модули Тамбары. Как правило, вся оптика, имеющая экзистенциальное представление, имеет также и своего рода профункторное представление. Преимущество профункторных представлений состоит в том, что они являются функциями и компонуются с использованием композиции функций.

Линзы, в частности, имеют представление с использованием специальной категории профункторов, называемых модулями Тамбары. Базовый модуль Тамбары представляет собой профунктор `p`, оснащенный семейством преобразований. Его можно реализовать как класс Haskell:

```
class Profunctor p => Tambara p where
  alpha :: forall a da m. p a da -> p (m, a) (m, da)
```

Соответствующая линза тогда представляется следующей профункторно-полиморфной функцией:

```
type Lens a da s ds = forall p.
                      Tambara p => p a da -> p s ds
```

Аналогичное представление можно построить и для пред-линз. Однако пред-линза имеет дополнительную зависимость от параметров и остатков, поэтому аналог модуля Тамбары тоже должен параметризоваться ими. Поэтому, нужен более сложный конструктор типа `t`, принимающий шесть аргументов:

```
t m dm p dp s ds
```

Предполагается, что это профунктор в трех парах аргументов, `s ds`, `p dp` и `dm m`. Профункциональность в первых двух парах реализована в виде функций `diampS` и `dimapP`. Инвертированный порядок в `dm m` означает, что `t` ковариантен в `m` и контравариантен в `dm`, как видно из необычной сигнатуры типа `dimapM`:

```
dimapM :: (m -> m') -> (dm' -> dm) ->
          t m dm p dp s ds ->
          t m' dm' p dp s ds
```

Чтобы обобщить модули Тамбары, сначала заметим, что пред-линза теперь имеет два независимых остатка, `m` и `dm`, и оба должны трансформироваться отдельно. Кроме того, композиция пред-линз накапливает (посредством кортежа), как остатки, так и параметры, поэтому имеет смысл использовать дополнительные аргументы типа для `TriProFunctor` в качестве аккумуляторов. Таким образом, обобщенный модуль Тамбары имеет два метода: один для накопления остатков, а другой, для накопления параметров:

```
class TriProFunctor t => Trimbara t where
  alpha :: t m dm p dp s ds ->
           t (m1, m) (dm1, dm) p dp (m1, s) (dm1, ds)
  beta  :: t m dm p dp (p1, s) (dp1, ds) ->
           t m dm (p, p1) (dp, dp1) s ds
```

Эти обобщенные модули Тамбары удовлетворяют некоторым условиям когерентности.

Можно также определить естественные преобразования, совместимые с новыми структурами, так что модули `Tambara` образуют категорию.

Возникает вопрос: может ли этому определению удовлетворять реальный нетривиальный `TriProFunctor` ? К счастью, оказывается, что пред-линза сама по себе является примером модуля `Tambara`. Реализация `alpha` для `PreLens` имеет вид:

```
alpha (PreLens fw bw)  = PreLens fw' bw'
  where
    fw' (p, (n, s))    = let (m, a) = fw (p, s)
                         in ((n, m), a)
    bw' ((dn, dm), da) = let (dp, ds) = bw (dm, da)
                         in  (dp, (dn, ds))
```

а реализация `beta`:

```
beta (PreLens fw bw)= PreLens fw' bw'
  where
    fw' ((p, r), s) = let (m, a) = fw (p, (r, s))
                      in  (m, a)
    bw' (dm, da)    = let (dp, (dr, ds)) = bw (dm, da)
                      in ((dp, dr), ds)
```

Этот результат станет важным в следующем пункте.

2.5. TriLens. Поскольку модули `Trimbara` образуют категорию, можно определить тип полиморфной функции (категорный конец) над модулями `Trimbara` Это дает (три-)профунторное представление для пред-линзы:

```
type TriLens a da m dm p dp s ds =
    forall t. Trimbara t => forall p1 dp1 m1 dm1.
      t m1 dm1 p1 dp1 a da ->
      t (m, m1) (dm, dm1) (p1, p) (dp1, dp) s ds
```

Действительно, имея пред-линзу, можно сформировать необходимое отображение модулей `Trimbara`, просто подняв две функции (прямой и обратный проходы) и поместив их между двумя структурными отображениями Тамбары:

```
toTamb                  :: PreLens a da m dm p dp s ds ->
                           TriLens a da m dm p dp s ds
toTamb (PreLens fw bw) = beta . dimapS fw bw . alpha
```

И наоборот, учитывая отображение между модулями `Trimbara`, можно построить пред-линзу, применив ее к единичной пред-линзе (по модулю некоторой перестановки кортежей с использованием моноидальных законов правой/левой единицы):

```
fromTamb  :: TriLens a da m dm p dp s ds ->
             PreLens a da m dm p dp s ds
fromTamb f = dimapM runit unRunit $
             dimapP unLunit lunit $
             f idPreLens
```

Основное преимущество профункторного представления состоит в том, что теперь можно скомпоновать две линзы, используя простую композицию функций; снова, по модулю некоторых ассоциаторов:

```
triCompose   ::
    TriLens b db m dm p dp s ds ->
    TriLens a da n dn q dq b db ->
    TriLens a da (m, n) (dm, dn) (q, p) (dq, dp) s ds
triCompose f g = dimapP unAssoc assoc .
                 dimapM unAssoc assoc .
                 f . g
```

Параллельная композиция `TriLenses` также относительно проста, хотя и требует большого объема программного кода (см. реализацию на gitHub).

2.6. Обучение нейронной сети. В качестве доказательства концепции был реализован и обучен простой трехслойный перцептрон.

Отправной точкой является преобразование отдельных компонентов нейрона из их пред-линзового представления в профункторное представление с использованием `toTamb`. Например:

```
linearT  :: Int -> TriLens D D (V, V) (V, V) V V V V
linearT n = toTamb (linearL n)
```

Получим профункторное представление нейрона, скомпоновав три его компонента:

```
neuronT     :: Int ->
  TriLens D D ((V, V), D) ((V, V), D) Para Para V V
neuronT mIn =
  dimapP (second (unLunit . unPara))
         (second (mkPara  . lunit)) .
  triCompose (dimapM (first runit) (first unRunit) .
  triCompose (linearT mIn) biasT) activT
```

При параллельной композиции три-линз, можно построить слой нейронов произвольной ширины.

```
layer         :: Int -> Int ->
  TriLens V V [((V, V), D)] [((V, V), D)] [Para] [Para] V V
layer mIn nOut =
  dimapP (second unRunit) (second runit)  .
  dimapM (first lunit)    (first unLunit) .
  triCompose (branch nOut) (vecLens nOut (neuronT mIn))
```

В результате, снова получается три-линза, и такие три-линзы можно скомпоновать последовательно, чтобы создать многослойный перцептрон.

```
makeMlp :: Int -> [Int] ->
  TriLens V V                                    -- output
           [[((V, V), D)]] [[((V, V), D)]]       -- residues
           [[Para]] [[Para]]                     -- parameters
           V V                                   -- input
```

Здесь, первое целое число указывает количество входов каждого нейрона в первом слое. Список `[Int]` указывает количество нейронов в последовательных слоях (что также является количеством входов каждого нейрона в следующем слое).

Обучение нейронной сети обычно осуществляется путем подачи ей пакета входных данных вместе с пакетом ожидаемых результатов. Это можно просто сделать, расположив несколько перцептронов параллельно и накопив параметры для всей партии.

```
batchN :: (VSpace dp)  =>   Int ->
    TriLens a da m dm p dp s ds ->
    TriLens [a] [da] [m] [dm] p dp [s] [ds]
```

Чтобы сделать накопление возможным, параметры должны образовывать векторное пространство, отсюда и ограничение `VSpace dp`.

Затем, все это ограничивается линзой потерь квадратичного расстояния, которая параметрируется основными истинностными значениями:

```
lossL :: PreLens D D ([V], [V]) ([V], [V]) [V] [V] [V] [V]
lossL  = PreLens fw bw
  where
    fw (gTruth, s) =
      ((gTruth, s), sqDist (concat s) (concat gTruth))
    bw ((gTruth, s), da) = (fmap (fmap negate) delta', delta')
      where
        delta' = fmap (fmap (da *)) (zipWith minus s gTruth)
```

3 Категорное представление

3.1. Конструкция Para. Большой интерес к категорным основам глубокого обучения возник не сегодня. Основная идея заключается в параметрической категории, в которой морфизмы параметризуются объектами из моноидальной категории. $\mathcal{P}$:

$$a \xrightarrow[f_p]{} b$$

Здесь, p — объект в $\mathcal{P}$.

Когда два таких морфизма скомпонованы, результат параметризуется тензорным произведением параметров.

$$a \xrightarrow[f_p]{} b \xrightarrow[g_q]{} c \qquad a \xrightarrow{h_{q \otimes p}} c$$

Тождественный морфизм параметризуется моноидальной единицей I.

Если моноидальная категория $\mathcal{P}$ не является строгой, то законы параметрической композиции и тождественности также не являются строгими. Они выполняются с точностью до ассоциаторов и объединителей $\mathcal{P}$. Категория с нестрогими композицией и законами тождества называется бикатегорией. 2-клетки в параметрической бикатегории называются репараметризациями.

Особый интерес представляют параметризованные бикатегории, построенные на основе актегорий. Актегория $\mathcal{C}$ — это категория, в которой определяется действие моноидальной категории $\mathcal{P}$:

$$\bullet : \mathcal{P} \times \mathcal{C} \to C$$

удовлетворяющая некоторым очевидным условиям связности (для единицы и композиции):

$$I \bullet c \cong c$$
$$p \bullet (q \bullet c) \cong (p \otimes q) \bullet c$$

Существуют две основные конструкции параметрической категории поверх актегории, обозначаемые **Para** и **coPara**. Первая строит параметрические морфизмы от a к b, согласно $f_p = p \bullet a \to b$, а вторая, согласно $g_p = a \to p \bullet b$.

3.2. Параметрическая оптика. Конструкцию **Para** можно распространить на оптику, где мы имеем дело с парами объектов из базовой категории (или категорий в случае смешанной оптики). Параметризованная оптика определяется следующим кодом:

$$O\,\langle a, da\rangle\,\langle p, dp\rangle\,\langle s, ds\rangle = \int^{m} \mathcal{C}(p \bullet s, m \bullet a) \times C(m \bullet da, dp \bullet ds)$$

де остатки m являются объектами некоторой моноидальной категории $\mathcal{M}$, а параметры $\langle p, dp\rangle$ происходят из другой моноидальной категории $\mathcal{P}$.

На Haskell, это именно экзистенциальная линза:

```
data ExLens a da p dp s ds =
  forall m . ExLens ((p, s)  -> (m, a))
                    ((m, da) -> (dp, ds))
```

Однако, существует более общая бикатегория пред-оптики, лежащая в основе экзистенциальной оптики. В ней, и параметры, и остатки, рассматриваются симметрично.

3.3. Бикатегория PreLens. Пред-оптика разрывает петлю обратной связи, в которой остатки прямого прохода передаются в обратный проход. Получаем следующую формулу:

$$O\,\langle a, da\rangle\,\langle m, dm\rangle\,\langle p, dp\rangle\,\langle s, ds\rangle = \mathcal{C}(p \bullet s, m \bullet a) \times C(dm \bullet da, dp \bullet ds)$$

Мы интерпретируем это как hom-множество от пары объектов $\langle s, ds\rangle$, из $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$, к паре объектов $\langle a, da\rangle$, также из $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$, параметризованное парой $\langle m, dm\rangle$ из $\mathcal{M} \times \mathcal{M}^{\mathsf{op}}$ и парой $\langle p, dp\rangle$ из $\mathcal{P}^{\mathsf{op}} \times \mathcal{P}$.

Чтобы упростить обозначения, будем использовать полужирный шрифт $\mathbf{C}$, для категории $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$, и полужирные буквы для пар объектов и (переплетенных) пар морфизмов. Например, $\mathbf{f} : \mathbf{a} \to \mathbf{b}$ является членом hom-множества $\mathbf{C}(\boldsymbol{a}, \mathbf{b})$, представленного парой $\langle f : a' \to a,\ g : b \to b'\rangle$.

Аналогично, будем использовать нотацию $\mathbf{m} \bullet \mathbf{a}$ для обозначения моноидального действия $\mathcal{M} \times \mathcal{M}^{\mathsf{op}}$ на $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$:

$$\langle m, dm\rangle \bullet \langle a, da\rangle = \langle m \bullet a, dm \bullet da\rangle$$

и аналогичного действия $\mathcal{P}^{\mathsf{op}} \times \mathcal{P}$.

В этих обозначениях пред-оптику можно просто записать как:

$$O\mathbf{a}\,\mathbf{m}\,\mathbf{p}\,\mathbf{s} = \mathbf{C}(\mathbf{m} \bullet \mathbf{a}, \mathbf{p} \bullet \mathbf{b})$$

а каждый конкретный морфизм как тройку:

$$(\mathbf{m}, \mathbf{p}, \mathbf{f} : \mathbf{m} \bullet \mathbf{a} \to \mathbf{p} \bullet \mathbf{b})$$

Пред-оптика образует hom-множества в бикатегории **PreLens**. Композиция представляет собой отображение:

$$\mathbf{C}(m \bullet b, p \bullet c) \times \mathbf{C}(n \bullet a, q \bullet b) \to \mathbf{C}((m \otimes n) \bullet a, (q \otimes p) \bullet c)$$

Действительно, поскольку оба моноидальных действия функториальны, можно поднять первый морфизм посредством $(q \bullet -)$, а второй посредством $(m \bullet -)$:

$$\mathbf{C}(m \bullet b, p \bullet c) \times \mathbf{C}(n \bullet a, q \bullet b) \xrightarrow{(q\bullet)\times(m\bullet)} \mathbf{C}(q \bullet m \bullet b, q \bullet p \bullet c) \times \mathbf{C}(m \bullet n \bullet a, m \bullet q \bullet b)$$

Можно скомпоновать эти hom-множества из $\mathbf{C}$, если два моноидальных действия коммутативны, то есть, если имеет место:

$$q \bullet m \bullet b \to m \bullet q \bullet b$$

для всех $\mathbf{q}$, $\mathbf{m}$ и $\mathbf{b}$. Тождественный морфизм есть тройка:

$$(\mathbf{1}, \mathbf{1}, \mathbf{id})$$

параметризованная единичными объектами в моноидальных категориях $\mathbf{M}$ и $\mathbf{P}$. Законы ассоциативности и тождественности выполняются по модулю ассоциаторов и объединителей.

Если базовая категория $\mathcal{C}$ моноидальна, то бикатегория **PreOp** также моноидальна, с очевидной поточечно-параллельной композицией пред-оптики.

3.4. Тройные модули Тамбары. Тройной модуль Тамбары является функтором:

$$T : \mathbf{M}^{\mathsf{op}} \times \mathbf{P} \times \mathbf{C} \to \mathbf{Set}$$

оснащенным двумя семействами естественных преобразований:

$$\alpha : T\,\mathbf{m}\,\mathbf{p}\,\mathbf{a} \to T\,(\mathbf{n} \otimes \mathbf{m})\,\mathbf{p}\,(\mathbf{n} \bullet \mathbf{a})$$
$$\beta : T\,\mathbf{m}\,\mathbf{p}\,(\mathbf{r} \bullet \mathbf{a}) \to T\,\mathbf{m}\,(\mathbf{p} \otimes \mathbf{r})\,\mathbf{a}$$

и некоторыми условиями согласованности. Например, два пути от $T\,\mathbf{m}\,\mathbf{p}\,(\mathbf{r} \bullet \mathbf{a})$ к $T\,(\mathbf{n} \otimes \mathbf{m})\,(\mathbf{p} \otimes \mathbf{r})\,(\mathbf{n} \bullet \mathbf{a})$ должны приводить к одному и тому же результату.

Можно также определить естественные преобразования между такими функторами, сохраняющими две структуры, и определить бикатегорию тройных модулей Тамбары, **TriTamb**.

В частном случае, если мы выберем категорию $\mathcal{P}$, в качестве тривиальной одно-объектной моноидальной категории, то получим версию (двойных) модулей Тамбары. Если затем мы образуем ко-конец $P\,\langle a, b\rangle = \int^m T\,\langle m, m\rangle\,\langle a, b\rangle$, то получим обычные модули Тамбары.

Сами пред-оптики являются примером тройного представления Тамбары. Действительно, для любого фиксированного $\mathbf{a}$, можно определить отображение α от тройки

$$(\mathbf{m}, \mathbf{p}, \mathbf{f} : \mathbf{m} \bullet \mathbf{a} \to \mathbf{p} \bullet \mathbf{b})$$

к тройке

$$(\mathbf{n} \otimes \mathbf{m}, \mathbf{p}, \mathbf{f}' : (\mathbf{n} \otimes \mathbf{m}) \bullet \mathbf{a} \to \mathbf{p} \bullet (\mathbf{n} \bullet \mathbf{b}))\,,$$

с помощью поднятия $\mathbf{f}$ посредством $(n \bullet -)$, и перестановки действий, используя их коммутативность. Аналогично, для β, отобразим

$$(\mathbf{m}, \mathbf{p}, \mathbf{f} : \mathbf{m} \bullet \mathbf{a} \to \mathbf{p} \bullet (\mathbf{r} \bullet \mathbf{b}))$$

к:

$$(\mathbf{m}, (\mathbf{p} \otimes \mathbf{r}), \mathbf{f}' : \mathbf{m} \bullet \mathbf{a} \to (\mathbf{p} \otimes \mathbf{r}) \bullet \mathbf{b})$$

3.5. Представление Тамбары. Основной результат состоит в том, что морфизмы в **PreOp** могут быть выражены с использованием тройных модулей Тамбары. Оптика

$$(\mathbf{m}, \mathbf{p}, \mathbf{f} : \mathbf{m} \bullet \mathbf{a} \to \mathbf{p} \bullet \mathbf{b})$$

эквивалентна тройному концу

$$\int_{\mathbf{r}:\mathbf{P}} \int_{\mathbf{n}:\mathbf{M}} \int_{T:\mathbf{TriTamb}} \mathbf{Set}(T\,\mathbf{n}\,\mathbf{r}\,\mathbf{a}, T\,(\mathbf{m}\otimes\mathbf{n})\,(\mathbf{r}\otimes\mathbf{p})\,\mathbf{b})\,.$$

Действительно, поскольку пред-оптика, сама по себе, является тройным модулем Тамбары, можно применить полиморфное отображение модулей Тамбары к тождественной оптике $(\mathbf{1}, \mathbf{1}, \mathbf{id})$ и получить произвольную пред-оптику.

И наоборот, для оптики

$$(\mathbf{m}, \mathbf{p}, \mathbf{f} : \mathbf{m} \bullet \mathbf{a} \to \mathbf{p} \bullet \mathbf{b})$$

можно построить полиморфное отображение тройных модулей Тамбары:

$$T\,\mathbf{n}\,\mathbf{r}\,\mathbf{a} \xrightarrow{\alpha} T\,(\mathbf{m}\otimes\mathbf{n})\,\mathbf{r}\,(\mathbf{m}\bullet\mathbf{a}) \xrightarrow{T\,\mathbf{f}} T\,(\mathbf{m}\otimes\mathbf{n})\,\mathbf{r}\,(\mathbf{p}\bullet\mathbf{b}) \xrightarrow{\beta} T\,(\mathbf{m}\otimes\mathbf{n})\,(\mathbf{r}\otimes\mathbf{p})\,\mathbf{b}\,.$$

4 Список литературы

- Brendan Fong, Michael Johnson, Lenses and Learners.
- Brendan Fong, David Spivak, Rémy Tuyéras, Backprop as Functor: A compositional perspective on supervised learning, 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) 2019, pp. 1-13, 2019.
- G.S.H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, Fabio Zanasi, Categorical Foundations of Gradient-Based Learning.
- Bruno Gavranović, Compositional Deep Learning.
- Bruno Gavranović, Fundamental Components of Deep Learning, PhD Thesis. 2024.