Московский государственный университет имени М.В. Ломоносова

Задание 1

Автоматическое дифференцирование для автокодировщика

Факультет: Вычислительной математики и кибернетики
Кафедра: Системного программирования
Группа: 628
Студент: Колосков Андрей Анатольевич
Предмет: Глубинное обучение

8 октября 2017

Формулировка задания

1. Для модели автокодировщика реализовать процедуру вычисления функции потерь $F(\Theta)$ с помощью прохода вперёд, а также её градиента с помощью прохода назад. Проверить корректность вычислений с помощью разностной аппроксимации;

2. Выписать формулы для прохода назад для оператора $Rp\{\cdot\}$ производной функции-аргумента по заданному направлению p, полученные формулы вставить в отчёт. Реализовать процедуру вычисления произведения гессиана функции $F(\Theta)$ на произвольный вектор p. Проверить корректность вычислений с помощью разностной аппроксимации;

3. Реализовать процедуру вычисления произведения гаусс-ньютоновской аппроксимации гессиана функции $F(\Theta)$ на произвольный вектор p. Для реализованного кода проверить корректность вычислений;

4. Реализовать набор стохастических методов оптимизации: 1) стохастический градиентный спуск (SGD), 2) SGD+momentum, 3) RMSprop и 4) ADAM;

5. Провести исследование с реализованной моделью автокодировщика и методами стохастической оптимизации. Рассмотреть набор данных MNIST1, а также несколько архитектур автокодировщика, в которых на среднем слое находится два нейрона и используется линейная функция потерь. Для различных методов оптимизации построить графики сходимости значения функции потерь на валидационной выборке в зависимости от числа эпох. Прокомментировать использованный способ настройки длины шага и размера мини-батча. Какой из методов оптимизации показывает наилучшие результаты? Визуализировать значения выходов со среднего слоя (там, где два нейрона) для начальных значений параметров автокодировщика и для параметров, найденных в результате оптимизации;

6. Составить отчёт в формате PDF о результатах всех проведённых исследований;

Реализация

Реализация всех функций прохода вперед и назад для слоя находятся в файле $layers.py$, приложенном к данному отчету. Реализация функций работы автокодировщика в целом находится в файле $autoencoder.py$. Вычисление функций активации и численного контроля градиентов находятся в файлах $activations.py$ и $check\_grad\_functions.py$ соответственно.

Проверка корректности вычисления прохода вперед—назад с помощью разностной аппроксимации осуществляется в $check\_forward\_backward.py$, запуск:

$python3\ check\_forward\_backward.py$

Получаем результат порядка $1.0e-8$ – $1.0e-11$, что сопоставимо с $\varepsilon = \sqrt{\varepsilon_{mach}}$, и соответствует заданию.

Формулы для прохода назад оператора $Rp\{\cdot\}$ получены путем применения оператора $Rp\{\cdot\}$ к каждой строчке стандартного алгоритма прохода назад, представленного в задании:

$$\nabla_{z^L} R_p\{L\} = R_p\{z^L\}$$
$$\nabla_{u^L} R_p\{L\} = \nabla_{z^L} R_p\{L\} \odot g'(u^L) + \nabla_{z^L} L \odot g''(u^L) \odot R_p\{u^L\}$$
$$\nabla_{\bar{W}^L} R_p\{L\} = (\nabla_{u^L} R_p\{L\})(\bar{z}^{L-1})^T + (\nabla_{u^L} L)(R_p\{\bar{z}^{L-1}\})^T$$
$$l = L-1, L-2, ..., 1:$$
$$\nabla_{z^l} R_p\{L\} = (\bar{p}^{l+1})^T(\nabla_{u^L} L) + (W^{l+1})^T(\nabla_{u^L} R_p\{L\})$$
$$\nabla_{u^l} R_p\{L\} = \nabla_{z^l} R_p\{L\} \odot g'(u^l) + \nabla_{z^l} L \odot g''(u^l) \odot R_p\{u^l\}$$
$$\nabla_{\bar{W}^l} R_p\{L\} = (\nabla_{u^l} R_p\{L\})(\bar{z}^{l-1})^T + (\nabla_{u^l} L)(R_p\{\bar{z}^{l-1}\})^T$$

Проверка корректности вычисления произведения вычисления произведения гессиана функции $F(\Theta)$ на произвольный вектор p с помощью разностной аппроксимации осуществляется в $check\_rp\_forward\_backward.py$, запуск:

$python3\ check\_rp\_forward\_backward.py$

Получаем результат порядка $1.0e-9$ – $1.0e-13$, что сопоставимо с $\varepsilon = \sqrt{\varepsilon_{mach}}$, и соответствует заданию.

Проверка корректности вычисления произведения гаусс-ньютоновской аппроксимации гессиана функции $F(\Theta)$ на произвольный вектор p с помощью разностной аппроксимации осуществляется в $check\_gauss\_newton.py$, запуск:

$python3\ check\_gauss\_newton.py$

Получаем результат порядка $1.0e-8$ – $1.0e-10$, что сопоставимо с $\varepsilon = \sqrt{\varepsilon_{mach}}$, и соответствует заданию.

Стохастический градиентный спуск (SGD), SGD+momentum, RMSprop и ADAM реализованы в файле $autoencoder.py$. Их использование в файле $use\_autoencoder.py$, запуск:

```
python3 use_autoencoder.py -p <path to MNIST folder>
        -t <type of gradient function> -tr <train data size>
        -ts <test data size> -e <number of epochs> -ms <minibatch size>
        -mm <momentum> -d <print to display>
```

В результате тестирования на наборе данных MNIST нескольких моделей автокодировщика со средним слоем 2 нейрона и линейной функцией потерь, наилучшую сходимость удалось достигнуть при небольшом числе слоев и резких изменениях в количестве нейронов между слоями: 784, 250, 50, 2, 50, 250, 784.

Длина шага определяется согласно рекомендациям: практикум на ЭВМ (317)/2012-2013/Autoencoder. Зависит от номера эпохи:

$$s_0 = 1.0\ s_k = \frac{s_0}{k^{\frac{1}{4}}}$$

Это представляется логичным, так как с ростом количества эпох, величина функции потерь снижается и градиентный спуск должен осуществляться более мелкими шагами. Размер мини-батча выбирал исходя из того, чтобы туда гарантированно попадала каждая цифра несколько раз, это должно уменьшить вероятность переобучения. Соответственно, он должен быть не менее 100, но не более 1000, так как в противном случае происходит незначительное замедление вычислительного процесса из-за операций с большими матрицами.

Кроме того, немного модифицировал стандартную батч нормализацию. Дело в том, что если преобразовывать входные данные к единичной дисперсии, то значения некоторых точек будут больше 1.0, но так как в сети используется сигмоид, то на выходе все значения будут не больше 1.0. Это увеличит размер ошибки. Поэтому все значения входной выборки делю на максимальное значение плюс 1.0. В данном примере это 256. Таким образом гарантированно получаю значения меньше 1.0 и относительно нормальную сходимость.

На рисунке 1 представлен график сходимости значения линейной функции потерь (вертикальная ось) на валидационной выборке в зависимости от числа эпох (горизонтальная ось) для стохастического градиентного спуска -

sgd. Размер тренировочной выборки 1000, тестовой 100, размер мини-батча 100.

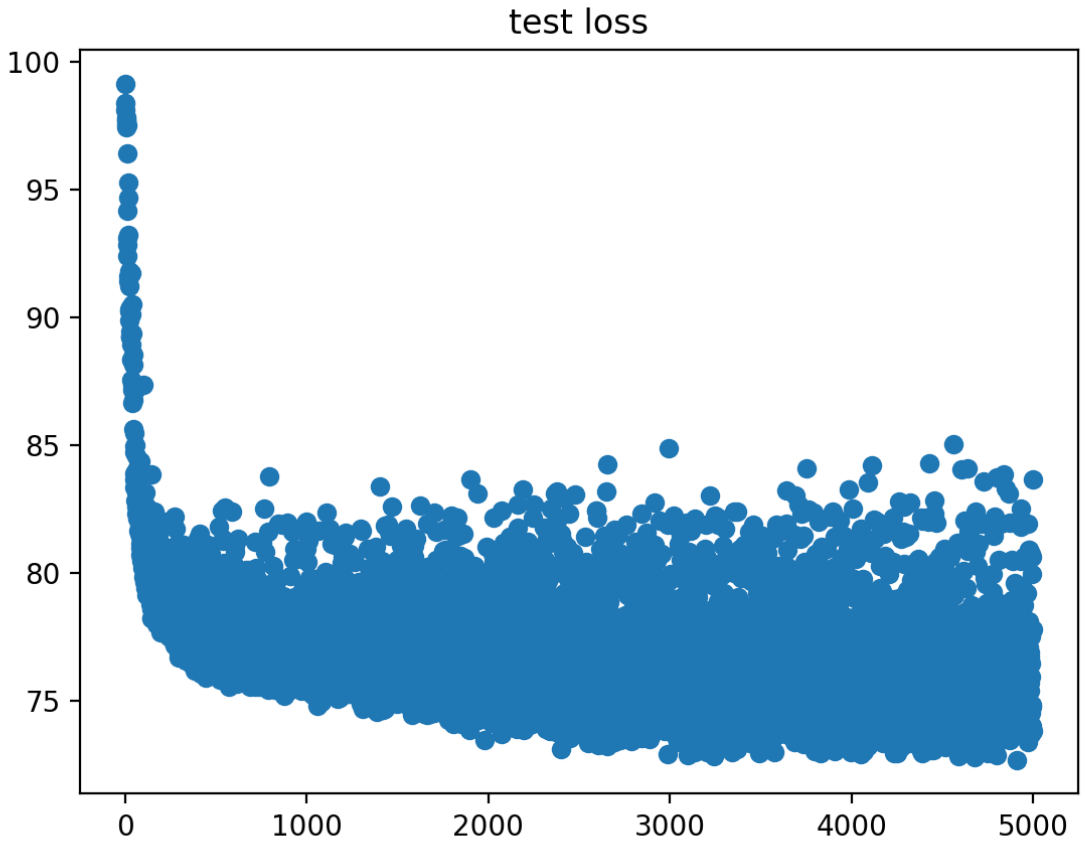

Рис. 1: Функция потерь для SGD

Вначале происходит резкое снижение величины функции потерь, а затем ее постепенное плавное снижение, но при этом растет дисперсия. Оптимальное значение функции потерь достигается примерно на 3000 эпохе. Достаточно высокие значения обусловлены тем, что в центральном слое расположено всего 2 нейрона. Эксперименты с большим числом нейронов показывали значительное улучшение данного показателя.

При этом первоначальная картинка в большинстве случаев восстанавливается до приемлемого качества. На рисунках 2 и 3 с левой стороны показаны примеры исходной картинки из тестовой выборки, а справа этой же картинки после прохода через автокодировщик, обученный на 3000 эпохах. Конечно попадаются и плохие варианты, когда происходит сильное искажение, но их значительно меньше.

На рисунке 4 представлен аналогичный предыдущему график сходимости значения линейной функции потерь для sgd + momentum. Здесь наблю-

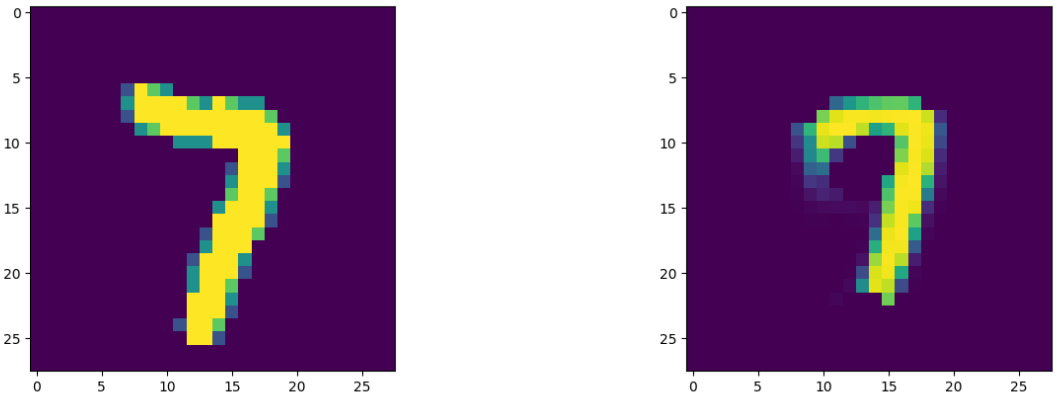

Рис. 2: Пример картинки до и после преобразования

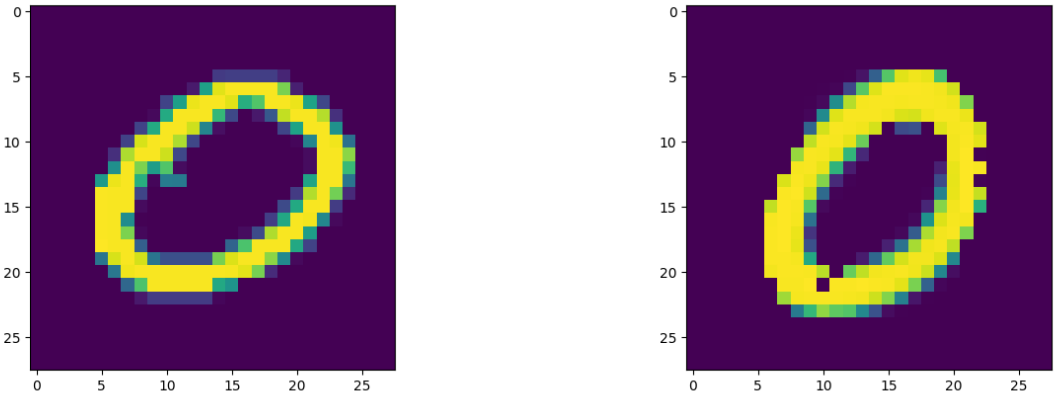

Рис. 3: Пример картинки до и после преобразования

дается некоторое улучшение относительно простого sgd, немного снизилась величина функции потерь, и уменьшилась дисперсия.

На рисунке 5 представлен аналогичный предыдущему график сходимости значения линейной функции потерь для RMSprop. Здесь наблюдается еще большее улучшение относительно sgd и sgd+momentum, снизилась величина функции потерь, и значительно уменьшилась дисперсия.

На рисунке 6 представлен аналогичный предыдущем график сходимости значения линейной функции потерь для ADAM. Здесь улучшений не наблюдается, все примерно на уровне предыдущего метода.

На рисунке 7 представлена визуализация среднего слоя для начальных значений автокодировщика, а на рисунке 8 после оптимизации.

По итогам данной работы можно сделать вывод, что модель автокодировщика со средним слоем в 2 нейрона достигает приемлемого качества, но для улучшения показателей все–таки желательно большее количество нейронов.

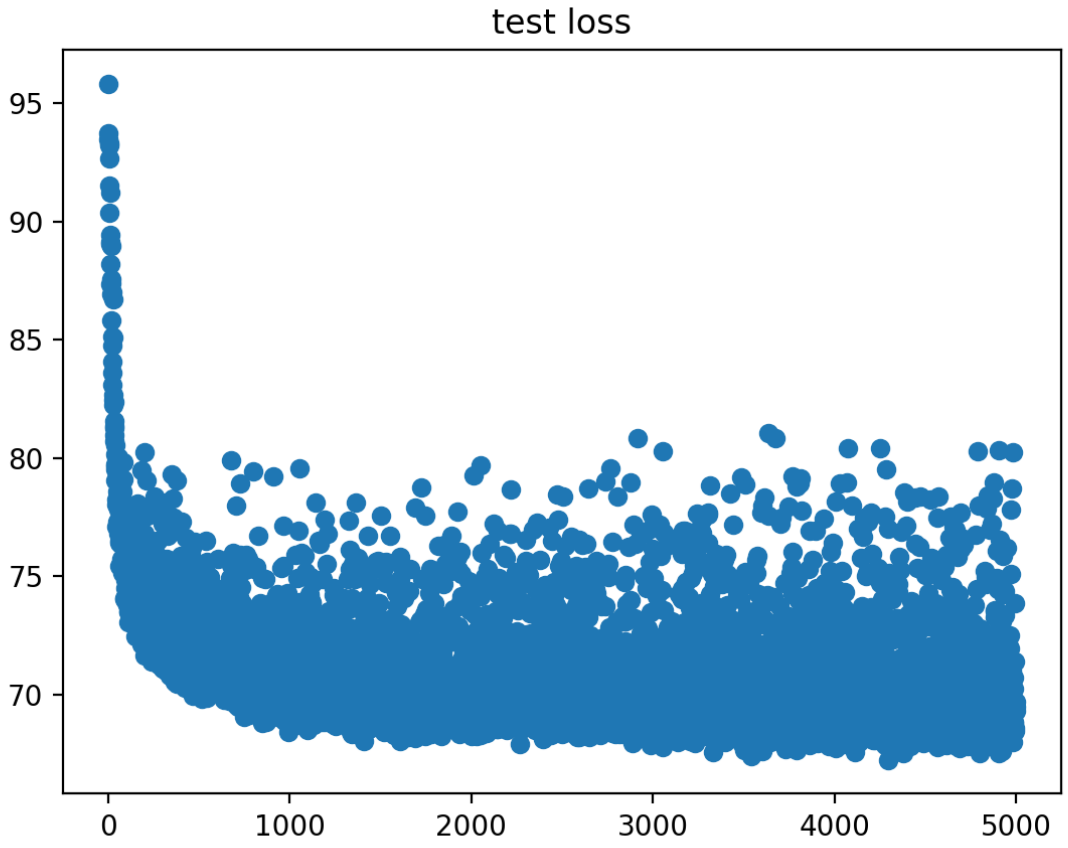

Рис. 4: Функция потерь для SGD+momentum

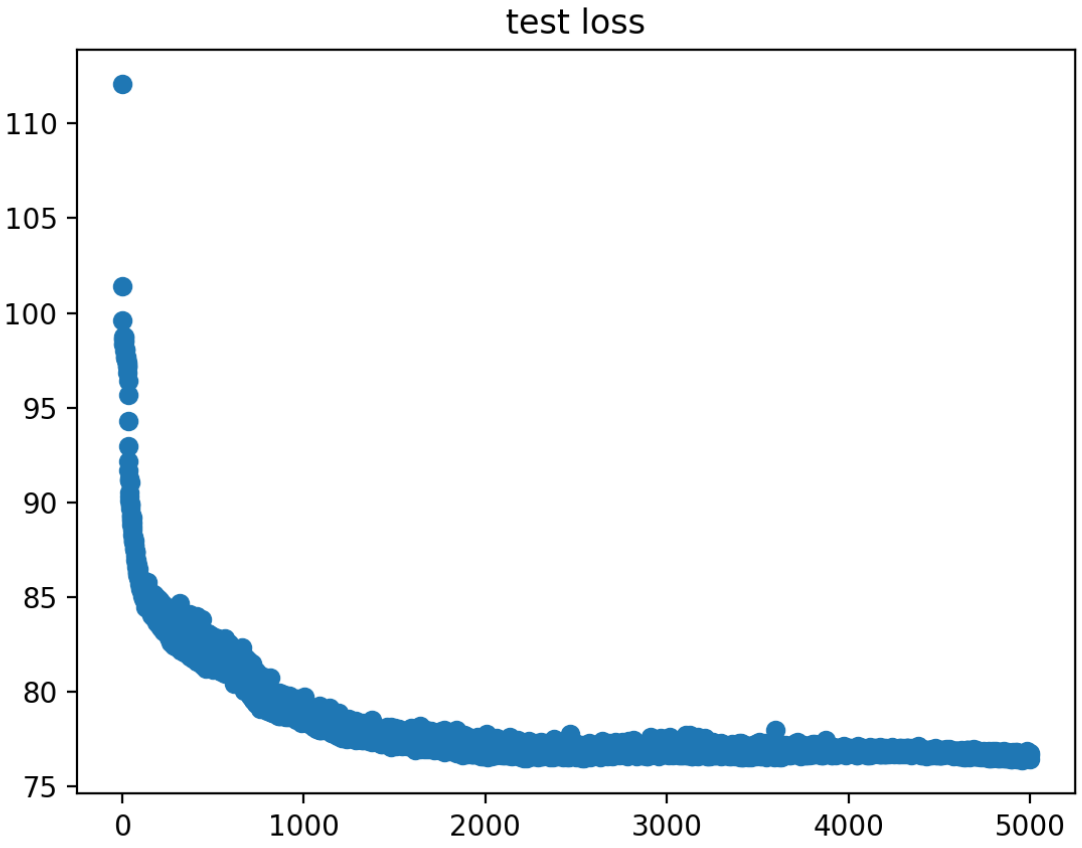

Рис. 5: Функция потерь для RMSprop

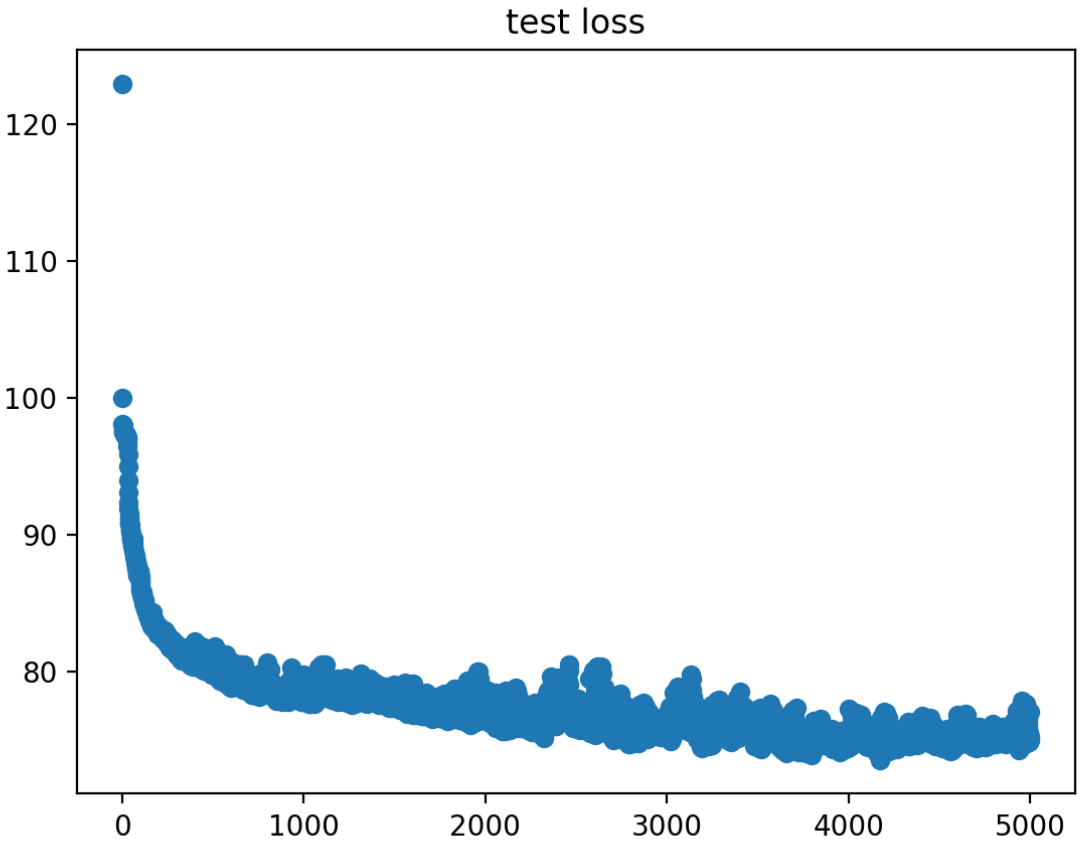

Рис. 6: Функция потерь для ADAM

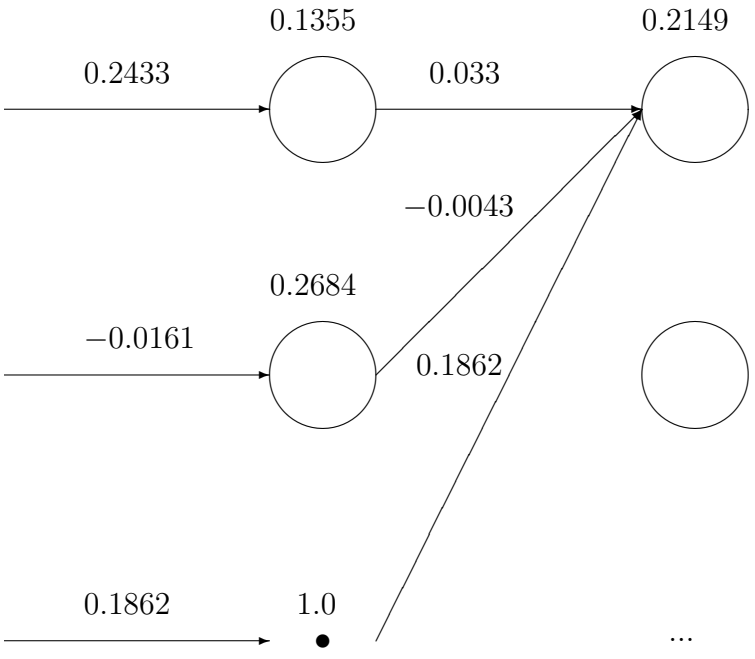

Рис. 7: Визуализация среднего слоя для начальных значений

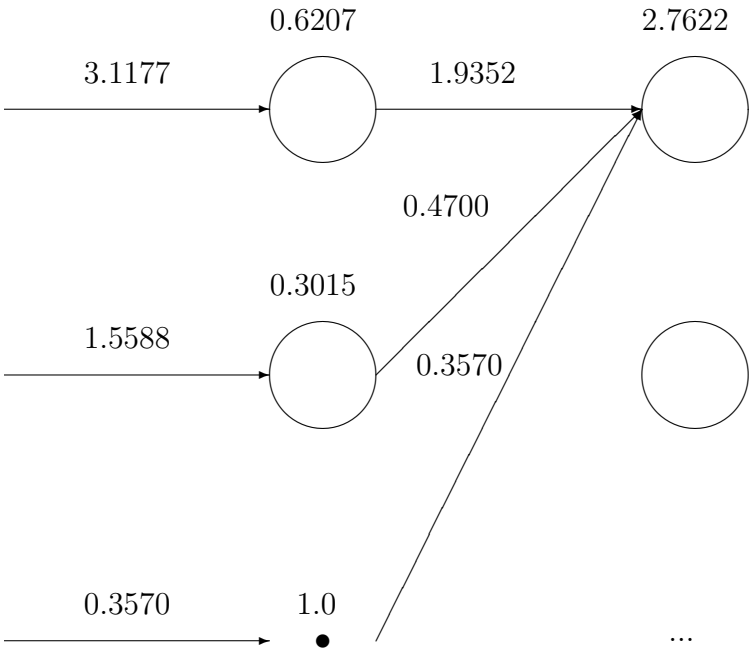

Рис. 8: Визуализация среднего слоя после оптимизации