Лабораторная работа 3: реализация алгоритма обратного распространения ошибки, обучение много

1. Цель работы

Целью данной лабораторной работы является получение практических навыков реализации и обучения полносвязной нейронной сети с нуля на языке Python с использованием объектно-ориентированного подхода. Вам нужно реализовать алгоритм обратного распространения ошибки (backpropagation) с применением градиентного спуска на мини-батчах (mini-batch gradient descent) для обучения сети на задаче бинарной классификации.

2. Задачи

1. Изучить теоретические основы алгоритма обратного распространения ошибки и градиентного спуска на мини-батчах.
2. Реализовать класс `NeuralNetwork`, инкапсулирующий структуру и методы полносвязной нейронной сети.
3. Реализовать основные функции активации (Sigmoid, ReLU, Tanh) и их производные.
4. Реализовать метод прямого распространения сигнала (`_forward`) через слои сети.
5. Реализовать метод вычисления функции потерь (`_compute_cost`), используя перекрестную энтропию.
6. Реализовать основной метод обратного распространения ошибки (`_backward`) для вычисления градиентов по весам и смещениям.
7. Реализовать метод обновления параметров сети (`_update_parameters`) на основе вычисленных градиентов.
8. Реализовать метод обучения (`fit`), организующий процесс обучения по эпохам с использованием мини-батчей.
9. Реализовать метод предсказания (`predict`) для получения выходных данных сети.
10. Протестировать реализованную нейронную сеть на синтетическом наборе данных `make_moons` и визуализировать результаты.

3. Теоретические сведения

Перед выполнением работы рекомендуется ознакомиться с предоставленными теоретическими материалами, содержащими подробный вывод формул.

Ключевые концепции:

Полносвязная нейронная сеть (Feedforward Neural Network): сеть, в которой нейроны организованы в слои, и каждый нейрон слоя связан со всеми нейронами предыдущего слоя.

Прямое распространение (Forward Propagation): процесс вычисления выхода сети путем последовательной передачи входных данных через слои с применением весов, смещений и функций активации.

$$Z^L = W^L A^{L-1} + b^L \; A^L = h_L(Z^L)$$

Функция потерь (Loss Function): мера расхождения между предсказаниями сети и истинными значениями. В данной работе используется бинарная перекрестная энтропия:

$$E = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(a^{L,(i)}) + (1 - y^{(i)}) \log(1 - a^{L,(i)})]$$

Обратное распространение ошибки (Backpropagation): алгоритм вычисления градиентов функции потерь по всем параметрам сети (весам W и смещениям b) путем применения цепного правила дифференцирования, двигаясь от выходного слоя к входному.

Выходной слой (L): $\frac{\partial E}{\partial W^L}$, $\frac{\partial E}{\partial b^L}$

Скрытые слои (l < L): $\frac{\partial E}{\partial W^l}$, $\frac{\partial E}{\partial b^l}$

Градиентный спуск (Gradient Descent): итерационный алгоритм оптимизации, который корректирует параметры сети в направлении, противоположном градиенту функции потерь, для минимизации потерь.

© Dmitry Vlasov 2025. All rights reserved.
Licensed under CC BY-NC-ND 4.0 – https://creativecommons.org/licenses/by-nc-nd/4.0/

$W^l = W^l - \alpha \frac{\partial E}{\partial W^l}$ $b^l = b^l - \alpha \frac{\partial E}{\partial b^l}$

Мини-батчи (Mini-batches): обучение сети не на всем наборе данных сразу и не по одному примеру за раз, а на небольших случайных подмножествах (мини-батчах) данных. Это обеспечивает компромисс между точностью оценки градиента и скоростью обучения.

4. Порядок выполнения работы

1. **Подготовка:**

   Получите файл-шаблон Jupyter Notebook `template_lab.ipynb`. Убедитесь, что у вас установлены необходимые библиотеки Python: `numpy`, `matplotlib`, `scikit-learn`. Ознакомьтесь со структурой шаблона и комментариями к коду.

2. **Реализация класса `NeuralNetwork`:**

   Откройте `template_lab.ipynb`. Последовательно реализуйте недостающие части кода в методах класса `NeuralNetwork`, следуя комментариям и TODO-маркерам (`# --- НАЧАЛО ВАШЕГО КОДА ---`, `# --- КОНЕЦ ВАШЕГО КОДА ---`, `# ЗАМЕНИТЬ None`).

   - *`_forward(self, X)`:* реализуйте цикл прямого распространения, вычисляя Z^l и A^l для каждого слоя и сохраняя их в `cache`.
   - *`_compute_cost(self, A_last, Y)`:* реализуйте вычисление бинарной перекрестной энтропии.
   - *`_backward(self, A_last, Y, cache)`:* реализуйте алгоритм обратного распространения. Начните с вычисления градиентов для выходного слоя L, а затем в цикле вычислите градиенты для скрытых слоев от $L-1$ до 1. Используйте значения из `cache` и производные функций активации.
   - *`_update_parameters(self, grads)`:* реализуйте простое правило обновления параметров с использованием градиентов из `grads` и скорости обучения `self.learning_rate`.
   - *`fit(self, X_train, Y_train, epochs, batch_size, ...)`:* реализуйте цикл обучения по эпохам. В каждой эпохе: Перемешайте обучающие данные (`X_train`, `Y_train`). Разбейте данные на мини-батчи. Для каждого мини-батча выполните прямое распространение, вычисление потерь, обратное распространение и обновление параметров. Вычислите среднюю стоимость за эпоху.
   - *`predict(self, X)`:* реализуйте метод предсказания. Выполните прямое распространение и преобразуйте выходные активации (вероятности) в бинарные предсказания (0 или 1) с использованием порога 0.5.

3. **Тестирование и анализ:**

Запустите ячейки ноутбука, отвечающие за генерацию данных (`make_moons`), определение архитектуры сети и запуск обучения (`nn.fit(...)`).

Убедитесь, что обучение проходит без ошибок, и значение функции потерь уменьшается со временем (отображается в выводе ячейки и на графике).

Проанализируйте график функции потерь. Он должен показывать сходимость алгоритма.

Запустите ячейку для визуализации границы решений. Убедитесь, что сеть научилась разделять два класса данных.

Запустите ячейку для вычисления точности (accuracy) на обучающем наборе данных. Оцените качество работы сети.

7. **Эксперименты (опционально):**

   Попробуйте изменить архитектуру сети (количество слоев, количество нейронов в слоях).

   Попробуйте использовать другие функции активации (например, `tanh` вместо `relu`). Измените гиперпараметры обучения: скорость обучения (`learning_rate`), размер мини-батча (`batch_size`), количество эпох (`epochs`). Проанализируйте, как эти изменения влияют на скорость сходимости и итоговое качество модели. Попробуйте использовать другие наборы данных из `sklearn.datasets` (например, `make_circles`, `make_blobs`).

© Dmitry Vlasov 2025. All rights reserved.
Licensed under CC BY-NC-ND 4.0 – https://creativecommons.org/licenses/by-nc-nd/4.0/

5. Оформление результатов

Представьте отчет о лабораторной работе, включающий:

1. Титульный лист.
2. Цель работы и задачи.
3. Краткие теоретические сведения (основные формулы и концепции).
4. Описание реализованного класса `NeuralNetwork` и его методов.
5. Результаты тестирования: График функции потерь в процессе обучения. Визуализация границы решений для набора данных `make_moons`. Значение точности (accuracy) на обучающей выборке.
6. Анализ результатов и выводы по работе.
7. (Опционально) Результаты дополнительных экспериментов и их анализ.
8. Приложение: Полностью рабочий код Jupyter Notebook (`.ipynb` файл).

6. Контрольные вопросы

1. Объясните шаги алгоритма прямого распространения.
2. Запишите формулу бинарной перекрестной энтропии и объясните ее смысл.
3. Объясните основную идею алгоритма обратного распространения ошибки. Какую роль играет цепное правило?
4. Как вычисляются градиенты dW^l и db^l для скрытого слоя l?
5. В чем разница между пакетным, стохастическим и мини-батчевым градиентным спуском? Каковы их преимущества и недостатки?
6. Зачем нужно перемешивать данные перед каждой эпохой при использовании мини-батчей?
7. Какую роль играет скорость обучения (alpha)? Что произойдет, если она будет слишком большой или слишком маленькой?
8. Объясните инициализацию весов (например, Xavier/Glorot, He). Почему нельзя инициализировать все веса нулями?
9. Какие проблемы могут возникнуть при обучении глубоких сетей (например, исчезающие/взрывающиеся градиенты)?

© Dmitry Vlasov 2025. All rights reserved.
Licensed under CC BY-NC-ND 4.0 – https://creativecommons.org/licenses/by-nc-nd/4.0/