Нейронные сети

Лабораторные работы

Единый сборник описаний лабораторных работ

Содержание

Вводная часть

Данный файл не является самостоятельной частью практической составляющей курса «Нейронные сети». Актуальная навигация по материалам курса ведется через сайт дисциплины: https://gitverse.ru/gurgutan/nn-site. Текст сборника «Нейронные сети. Лабораторные работы» не является самостоятельным учебным материалом. Он предназначен только для того, чтобы в одном файле собрать описания работ для удобной навигации по ним и создания комплекта документов по учебной дисциплине.

01-lab-01. Равномерная аппроксимация на компакте

Цель

Научиться воспроизводить вычислительный пример аппроксимации непрерывной функции на компакте и наблюдать, как максимальная ошибка меняется при увеличении степени аппроксимирующего семейства.

Материал лекции

Лабораторная относится к лекции 1 «Основные теоремы нейроинформатики» и опирается на понятия:

1. компакт;
2. непрерывная функция;
3. равномерная аппроксимация;
4. максимальная ошибка на сетке;
5. переход от теоремы существования к вычислительному наблюдению.

В лабораторной не требуется строить нейронную сеть или обучать модель.

Состав шаблона

Файлы лабораторной находятся в текущем каталоге:

1. **lab.py** - основной блокнот в виде **.py**-файла с ячейками **# %%** ;
2. **results/** - каталог, который создается автоматически после запуска.

Команда запуска блокнота

Откройте lab.py в редакторе, который поддерживает запуск ячеек **# %%**, и выполните ячейки сверху вниз.

Для проверки всего файла как обычного скрипта можно выполнить:

```
uv run python lab.py
```

Структура ячеек блокнота

1. **Постановка опыта** - объясняет цель и показывает каталог результатов.
2. **Параметры опыта** - задает функцию, степени многочленов и размер сетки.
3. **Целевые функции** - показывает доступные непрерывные функции на **[0, 1]**.
4. **Подготовительное вычисление** - вычисляет значение многочлена Бернштейна в одной точке.
5. **Таблица ошибок на сетке** - выводит максимальные ошибки для выбранных степеней.
6. **Задание** - предлагает изменить параметры опыта.
7. **Проверка выбранного варианта** - пересчитывает таблицу для измененных параметров.

8. **Визуализация выбранного варианта** - строит график выбранной функции и лучшей аппроксимации.
9. **Финальный артефакт** - сохраняет файлы результата.

Параметры, которые нужно изменить

В ячейке **Задание** разрешается изменить:

1. **SELECTED_TARGET_NAME** - имя функции из словаря шаблона;
2. **SELECTED_DEGREES** - список степеней аппроксимирующего многочлена;
3. **SELECTED_GRID_SIZE** - число точек сетки для оценки максимальной ошибки.

Шаги воспроизведения

1. Откройте **./lab.py**.
2. Выполните подготовительные ячейки сверху вниз без изменений.
3. Посмотрите таблицу ошибок и график аппроксимации.
4. В ячейке **Задание** измените функцию, степени или размер сетки.
5. Выполните ячейки **Проверка выбранного варианта** и **Финальный артефакт**.
6. Сравните лучшую ошибку базового и измененного вариантов.

Что нужно сдать

1. **approximation_errors.csv**
2. **approximation_summary.json**
3. **bernstein_approximation.png**

Формат результата

Файл **approximation_errors.csv** должен содержать столбцы:

```
target,degree,grid_size,max_abs_error
```

Файл **approximation_summary.json** должен содержать поля:

```
{
  "target": "sin_pi",
  "best_degree": 0,
  "best_max_abs_error": 0.0,
  "grid_size": 0
}
```

Критерии проверки

1. Оба файла результата существуют.
2. В CSV есть строка для каждой степени из **DEGREES**.
3. Значение **best_degree** в JSON соответствует строке с минимальной ошибкой в CSV.
4. Значения ошибок неотрицательны.
5. PNG-файл с графиком существует и открывается как изображение.

02-lab-01. Формальный нейрон и функции активации

Цель

Научиться наблюдать графики стандартного формального нейрона **y=f(wx+b)** для разных функций активации и визуализировать соединение двух нейронов через бинарную матрицу как ориентированный граф.

Материал лекции

Лабораторная относится к лекции 2 «Формальное описание искусственных нейронных сетей» и опирается на понятия:

1. стандартный формальный нейрон;
2. входной вектор;
3. вектор весов;
4. смещение;
5. функция активации;
6. матрица связи между выходами и входами.
7. граф соединений нейронов.

В лабораторной не требуется обучать веса или строить многослойную сеть.

Состав шаблона

Файлы лабораторной находятся в текущем каталоге:

1. **lab.py** - основной блокнот в виде **.py**-файла с ячейками **# %%** ;
2. **results/** - каталог, который создается автоматически после запуска.

Команда запуска блокнота

Откройте lab.py в редакторе, который поддерживает запуск ячеек **# %%**, и выполните ячейки сверху вниз.

Для проверки всего файла как обычного скрипта можно выполнить:

```
uv run python lab.py
```

Структура ячеек блокнота

1. **Постановка опыта** - связывает формальный нейрон и бинарную матрицу связей.
2. **Параметры нейрона** - задает вес, смещение и интервал входов.
3. **Функции активации** - вводит ReLU, sigmoid, tanh и Gaussian.
4. **Таблица значений стандартного нейрона** - вычисляет **z=wx+b** и **y=f(z)** на сетке.
5. **График функций активации нейрона** - строит четыре кривые **y=f(wx+b)**.
6. **Бинарная матрица соединения двух нейронов** - задает два исходных и два следующих нейрона.

7. **Значения, переданные через матрицу** - вычисляет входы следующего слоя.
8. **Визуализация графа соединения** - строит ориентированный граф через **networkx**.
9. **Задание** - предлагает изменить вес, смещение или бинарную матрицу.
10. **Проверка выбранного варианта** - пересчитывает кривые и переданные значения.
11. **Визуализация выбранного варианта** - строит графики и graph-схему для выбранных параметров.
12. **Финальный артефакт** - сохраняет файлы результата.

Параметры, которые нужно изменить

В ячейке **Задание** разрешается изменить:

1. **SELECTED_WEIGHT** - вес стандартного формального нейрона;
2. **SELECTED_BIAS** - смещение стандартного формального нейрона;
3. **SELECTED_CONNECTION_MATRIX** - бинарную матрицу связей;
4. **SELECTED_SOURCE_OUTPUTS** - вектор исходных выходов.

Функции активации ReLU, sigmoid, tanh и Gaussian уже заданы в шаблоне.

Шаги воспроизведения

1. Откройте **./lab.py**.
2. Выполните подготовительные ячейки сверху вниз без изменений.
3. Посмотрите график четырех функций активации стандартного нейрона.
4. Посмотрите graph-визуализацию бинарной матрицы соединения двух нейронов.
5. В ячейке **Задание** измените вес, смещение или матрицу связей.
6. Выполните ячейки **Проверка выбранного варианта**, **Визуализация выбранного варианта** и **Финальный артефакт**.
7. Сравните, как изменились кривые активаций, дуги графа и численные входы следующего слоя.

Что нужно сдать

1. **activation_curves.csv**
2. **connection_result.csv**
3. **lab_summary.json**
4. **selected_activation_curves.png**
5. **selected_binary_connection_graph.png**

Формат результата

Файл **activation_curves.csv** должен содержать столбцы:

```
activation,x,z,y
```

Файл **connection_result.csv** должен содержать столбцы:

```
target_neuron,value
```

Файл **lab_summary.json** должен содержать вес, смещение, список функций активации, размер матрицы связи и число дуг графа.

Критерии проверки

1. Все файлы результата существуют.
2. В CSV **activation_curves.csv** есть строки для ReLU, sigmoid, tanh и Gaussian.
3. Значения **x**, **z** и **y** являются числами.
4. Число строк в **connection_result.csv** равно числу следующих нейронов.
5. PNG-файл **selected_activation_curves.png** показывает четыре функции активации.
6. PNG-файл **selected_binary_connection_graph.png** показывает граф соединения двух нейронов через бинарную матрицу.

03-lab-01. Итерационные вычисления на графе

Цель

Научиться строить граф вычислений по матрице смежности, приписывать дугам веса, приписывать вершинам функции нейронов и выполнять несколько итераций вычислений на графе.

Материал лекции

Лабораторная относится к лекции 3 «Вычисления на графе» и опирается на понятия:

1. ориентированный граф;
2. матрица смежности;
3. веса дуг;
4. один шаг вычислений **x = y A**;
5. рациональная сигмоида;
6. итерационная схема **y^((n)) = f(y^((n-1)) A)**.

Состав шаблона

Файлы лабораторной находятся в текущем каталоге:

1. **lab.py** - основной блокнот в виде **.py**-файла с ячейками **# %%** ;
2. **results/** - каталог, который создается автоматически после запуска.

Команда запуска блокнота

Откройте lab.py в редакторе, который поддерживает запуск ячеек **# %%**, и выполните ячейки сверху вниз.

Для проверки всего файла как обычного скрипта можно выполнить:

```
uv run python lab.py
```

Структура ячеек блокнота

1. **Постановка опыта** - объясняет вычисления на графе.
2. **Матрица смежности вершин** - задает ориентированные дуги.
3. **Веса дуг** - приписывает дугам численные веса.
4. **Функции нейронов в вершинах** - задает функцию каждой вершины.
5. **Проверка согласованности матриц** - проверяет, что веса заданы только для существующих дуг.
6. **Визуализация графа с весами и функциями** - строит graph-схему через **networkx**.
7. **Один шаг вычислений на графе** - вычисляет входные суммы и новые значения.
8. **Несколько итераций вычислений** - строит траекторию значений вершин.
9. **График траекторий вершин** - показывает динамику по шагам.
10. **Задание** - предлагает изменить веса, функции вершин или начальные значения.

11. **Проверка выбранного варианта** - пересчитывает выбранную конфигурацию.
12. **Визуализация выбранного варианта** - строит выбранный граф и траектории.
13. **Финальный артефакт** - сохраняет файлы результата.

Параметры, которые нужно изменить

В ячейке **Задание** разрешается изменить:

1. **SELECTED_WEIGHT_MATRIX** - веса существующих дуг;
2. **SELECTED_VERTEX_FUNCTIONS** - функции нейронов в вершинах;
3. **SELECTED_INITIAL_VALUES** - начальные значения вершин;
4. **SELECTED_STEP_COUNT** - число итераций.

Размерность начального вектора должна совпадать с числом вершин. Ненулевые веса можно задавать только там, где в матрице смежности есть дуга.

Шаги воспроизведения

1. Откройте **./lab.py**.
2. Выполните подготовительные ячейки сверху вниз без изменений.
3. Посмотрите матрицу смежности, матрицу весов и функции в вершинах.
4. Посмотрите graph-визуализацию с подписями весов дуг и функций вершин.
5. Сравните один шаг вычислений с таблицей нескольких итераций.
6. В ячейке **Задание** измените веса, функции вершин или начальные значения.
7. Выполните ячейки **Проверка выбранного варианта**, **Визуализация выбранного варианта** и **Финальный артефакт**.

Что нужно сдать

1. **graph_iterations.csv**
2. **graph_summary.json**
3. **selected_weighted_computation_graph.png**
4. **selected_graph_iteration_trajectories.png**

Формат результата

Файл **graph_iterations.csv** должен содержать столбцы:

```
step,y1,y2,y3,y4
```

Число столбцов **y1**, **y2**, … зависит от числа вершин графа.

Файл **graph_summary.json** должен содержать число вершин, число шагов и последний вектор значений.

Критерии проверки

1. В CSV есть строка для шага **0** и для каждого выполненного шага.
2. Число координат в каждой строке совпадает с числом вершин.

3. В JSON поле **step_count** совпадает с параметром **STEP_COUNT**.
4. Последний вектор в JSON совпадает с последней строкой CSV.
5. PNG-файл графа показывает функции вершин и веса дуг.
6. PNG-файл траекторий показывает значения всех вершин по итерациям.

04-lab-01. Функции эмпирического риска

Цель

Научиться вычислять основные функции эмпирического риска без работы с нейронными сетями: MAE и MSE для регрессии, кросс-энтропию для бинарной и многоклассовой классификации, KL-дивергенцию для двух нормальных распределений.

Материал лекции

Лабораторная относится к лекции 4 «Обучение как задача многомерной оптимизации» и опирается на понятия:

1. обучающая выборка;
2. функция потерь на одном примере;
3. эмпирический риск;
4. средняя абсолютная ошибка;
5. среднеквадратичная ошибка;
6. бинарная кросс-энтропия;
7. многоклассовая кросс-энтропия;
8. дивергенция Кульбака-Лейблера.

В лабораторной не рассматриваются нейронные сети, параметры сети или шаги оптимизации.

Состав шаблона

Файлы лабораторной находятся в текущем каталоге:

1. **lab.py** - основной блокнот в виде **.py**-файла с ячейками **# %%** ;
2. **results/** - каталог, который создается автоматически после запуска.

Команда запуска блокнота

Откройте lab.py в редакторе, который поддерживает запуск ячеек **# %%**, и выполните ячейки сверху вниз.

Для проверки всего файла как обычного скрипта можно выполнить:

```
uv run python lab.py
```

Структура ячеек блокнота

1. **Постановка опыта** - задает четыре группы функций риска.
2. **Обучающая выборка для регрессии** - строит выборку для функции **a x + b + c sin(d x)**.
3. **MAE и MSE** - вычисляет среднюю абсолютную и среднеквадратичную ошибки.
4. **График регрессии и ошибок** - показывает обучающую выборку, прогноз, поточечные ошибки, MAE и MSE.
5. **CE для бинарной классификации** - вычисляет бинарную кросс-энтропию.

6. **CE для многоклассовой классификации** - вычисляет многоклассовую кросс-энтропию.
7. **KL-дивергенция двух распределений Гаусса** - вычисляет **D_KL(P || Q)**.
8. **График распределений Гаусса** - показывает плотности, поточечный вклад в KL и значение **D_KL(P || Q)**.
9. **Задание** - предлагает изменить прогнозные коэффициенты, вероятности или параметры распределений.
10. **Проверка выбранного варианта** - пересчитывает все функции риска.
11. **Визуализация выбранного варианта** - строит выбранные графики регрессии, MAE/MSE, распределений и KL.
12. **Финальный артефакт** - сохраняет файлы результата.

Параметры, которые нужно изменить

В ячейке **Задание** разрешается изменить:

1. **SELECTED_PREDICTED_COEFFICIENTS** - коэффициенты прогнозной функции;
2. **SELECTED_BINARY_CASES** - вероятности для бинарной классификации;
3. **SELECTED_MULTICLASS_CASES** - вероятности для многоклассовой классификации;
4. **SELECTED_GAUSSIAN_P** и **SELECTED_GAUSSIAN_Q** - параметры двух нормальных распределений.

Вероятности для кросс-энтропии должны быть строго положительными; в многоклассовом случае вероятности каждого объекта должны суммироваться в **1**.

Шаги воспроизведения

1. Откройте **./lab.py**.
2. Выполните подготовительные ячейки сверху вниз без изменений.
3. Сравните MAE и MSE для обучающей выборки.
4. Сравните CE для бинарной и многоклассовой классификации.
5. Посмотрите, как смещение и дисперсия нормальных распределений связаны с KL-дивергенцией.
6. В ячейке **Задание** измените коэффициенты, вероятности или параметры распределений.
7. Выполните ячейки **Проверка выбранного варианта**, **Визуализация выбранного варианта** и **Финальный артефакт**.

Что нужно сдать

1. **empirical_risks.csv**
2. **regression_sample.csv**
3. **risk_summary.json**
4. **selected_regression_mae_mse.png**
5. **selected_gaussian_kl.png**

Формат результата

Файл **empirical_risks.csv** должен содержать столбцы:

```
case_id,loss_type,value
```

Файл **regression_sample.csv** должен содержать столбцы:

```
x,y_true,y_pred
```

Файл **risk_summary.json** должен содержать значения MAE, MSE, бинарной CE, многоклассовой CE и KL-дивергенции.

Критерии проверки

1. В CSV **empirical_risks.csv** есть строки для MAE, MSE, бинарной CE, многоклассовой CE и KL.
2. Все значения функций риска неотрицательны.
3. В **regression_sample.csv** есть значения истинной функции и прогноза.
4. В **risk_summary.json** есть параметры двух нормальных распределений.
5. PNG-файл регрессии показывает обучающую выборку, прогноз, поточечные ошибки, MAE и MSE.
6. PNG-файл распределений показывает две нормальные плотности, поточечный вклад в KL и значение **D_KL(P || Q)**.

05-lab-01. Ручное обучение простой нейросети

Цель

Реализовать минимальную нейросеть из линейного слоя и слоя активации, явно вычислить MSE, градиент MSE, одну итерацию обучения и цикл обучения с постоянным **learning rate**.

Материал лекции

Лабораторная относится к лекции 5 «Общая схема градиентного обучения» и опирается на понятия:

1. прямой проход;
2. линейный слой;
3. слой активации;
4. функция ошибки MSE;
5. градиент функции ошибки;
6. направление антиградиента;
7. коррекция весов;
8. цикл обучения с постоянным шагом.

Состав шаблона

Файлы лабораторной находятся в текущем каталоге:

1. **lab.py** - основной блокнот в виде **.py**-файла с ячейками **# %%** ;
2. **results/** - каталог, который создается автоматически после запуска.

Команда запуска блокнота

Откройте lab.py в редакторе, который поддерживает запуск ячеек **# %%**, и выполните ячейки сверху вниз.

Для проверки всего файла как обычного скрипта можно выполнить:

```
uv run python lab.py
```

Структура ячеек блокнота

1. **Постановка опыта** - задает идею сети **LinearLayer -> TanhActivation**.
2. **Обучающая выборка** - строит одномерную регрессионную выборку.
3. **Линейный слой** - реализует **z = xW^T + b** и градиенты параметров.
4. **Слой активации** - реализует **tanh(z)** и производную активации.
5. **Простая нейросеть** - объединяет линейный слой и слой активации.
6. **MSE и градиент MSE** - вычисляет ошибку и градиент по прогнозам.
7. **Одна итерация обучения** - выполняет **forward -> MSE -> градиент -> коррекция весов**.
8. **Задание** - предлагает изменить начальные параметры, learning rate или число итераций.
9. **Цикл обучения** - выполняет несколько итераций с постоянным **learning rate**.

10. **Визуализация обучения** - показывает MSE по итерациям и итоговый прогноз сети.
11. **Финальный артефакт** - сохраняет файлы результата.

Параметры, которые нужно изменить

В ячейке **Задание** разрешается изменить:

1. **SELECTED_INITIAL_WEIGHTS** - начальную матрицу весов линейного слоя;
2. **SELECTED_INITIAL_BIASES** - начальные смещения;
3. **SELECTED_LEARNING_RATE** - постоянный шаг обучения;
4. **SELECTED_ITERATION_COUNT** - число итераций цикла обучения.

Слишком большой **learning rate** может увеличить MSE или привести к колебаниям траектории.

Шаги воспроизведения

1. Откройте **./lab.py**.
2. Выполните подготовительные ячейки сверху вниз без изменений.
3. Сравните MSE до и после одной итерации обучения.
4. В ячейке **Задание** измените **SELECTED_LEARNING_RATE** или начальные параметры.
5. Выполните ячейки **Цикл обучения**, **Визуализация обучения** и **Финальный артефакт**.
6. Сравните итоговое MSE и форму прогноза сети.

Что нужно сдать

1. **training_history.csv**
2. **training_sample.csv**
3. **final_model.json**
4. **training_mse_and_fit.png**

Формат результата

Файл **training_history.csv** должен содержать столбцы:

```
iteration,mse,weight,bias
```

Файл **training_sample.csv** должен содержать столбцы:

```
x,target
```

Файл **final_model.json** должен содержать архитектуру сети, начальные параметры, итоговые параметры, **learning rate**, число итераций, начальное и итоговое MSE.

Критерии проверки

1. В **training_history.csv** есть строка **iteration** = **0** и строка последней итерации.
2. Значения MSE неотрицательны.
3. В **final_model.json** указаны слои **LinearLayer** и **TanhActivation**.

4. Итоговые веса и смещения отличаются от начальных при ненулевом **learning rate**.
5. PNG-файл показывает траекторию MSE и итоговый прогноз сети.

06-lab-01. Автоматическое дифференцирование через Value

Цель

Реализовать простой механизм автоматического дифференцирования через класс **Value**, расширить его новой операцией и функцией, построить двухслойную сеть через **Value** и получить градиенты MSE для параметров каждого слоя.

Материал лекции

Лабораторная относится к лекции 6 «Автоматическое дифференцирование» и опирается на понятия:

1. вычислительный граф;
2. локальная производная операции;
3. цепное правило;
4. обратный топологический обход графа;
5. автоматическое дифференцирование в обратном режиме;
6. прямой проход нейросети;
7. MSE как узел вычислительного графа;
8. градиенты параметров слоя.

Исходный ориентир

Класс **Value** строится в учебном стиле скалярного автоматического дифференцирования: операция создает новый **Value**, сохраняет родителей и локальные производные, а **backward()** распространяет градиенты от итогового скаляра к листьям графа.

Состав шаблона

Файлы лабораторной находятся в текущем каталоге:

1. **lab.py** - основной блокнот в виде **.py**-файла с ячейками **# %%** ;
2. **results/** - каталог, который создается автоматически после запуска.

Команда запуска блокнота

Откройте lab.py в редакторе, который поддерживает запуск ячеек **# %%**, и выполните ячейки сверху вниз.

Для проверки всего файла как обычного скрипта можно выполнить:

```
uv run python lab.py
```

Структура ячеек блокнота

1. **Постановка опыта** - задает роль **Value** и обратного прохода.
2. **Базовый класс Value** - реализует скалярный autodiff-движок.
3. **Проверка цепного правила** - вычисляет градиенты простого выражения.

4. **Модификация Value** - добавляет **square()** и **sigmoid()** с локальными производными.
5. **Скалярная обучающая точка** - задает один вход **x** и целевое значение **y**.
6. **Скалярная двухслойная сеть** - строит выражение **w2 * sigmoid(w1 * x + b1) + b2**.
7. **MSE как узел графа** - вычисляет MSE через объекты **Value**.
8. **Обратный проход и градиенты слоев** - получает градиенты параметров скрытого и выходного слоев.
9. **Узлы вычислительного графа** - показывает компактную таблицу узлов графа.
10. **Граф Value через networkx** - строит граф зависимостей от MSE к параметрам, входу **x** и целевому значению **y**.
11. **Задание** - предлагает изменить скалярные параметры, learning rate или входное значение.
12. **Демонстрация шага по антиградиенту** - показывает потенциальную коррекцию параметров.
13. **Визуализация градиентов** - строит график градиентов и антиградиентов по слоям.
14. **Финальный артефакт** - сохраняет файлы результата.

Параметры, которые нужно изменить

В блокноте разрешается изменить:

1. начальные скалярные параметры в **ScalarTwoLayerNetwork**;
2. **X_VALUE** и **TARGET_VALUE**;
3. **SELECTED_LEARNING_RATE** для демонстрации шага;
4. **SELECTED_EXTRA_X** для дополнительного прямого прохода с видимым прогнозом.

После изменений нужно заново выполнить ячейки от создания сети до финального артефакта.

Шаги воспроизведения

1. Откройте **./lab.py**.
2. Выполните ячейки с **Value** и проверьте градиенты простого выражения.
3. Выполните ячейку **Модификация Value** и убедитесь, что добавленные **square()** и **sigmoid()** дают градиенты.
4. Выполните прямой проход скалярной двухслойной сети до MSE.
5. Выполните **loss.backward()** и сравните градиенты первого и второго слоев.
6. Измените параметры в ячейке **Задание** и пересчитайте результат.

Что нужно сдать

1. **layer_gradients.csv**
2. **autodiff_graph_nodes.csv**

3. **forward_mse_summary.json**
4. **layer_gradients.png**
5. **value_graph_from_mse.png**

Формат результата

Файл **layer_gradients.csv** должен содержать столбцы:

```
layer,parameter,value,gradient,antigradient
```

Файл **autodiff_graph_nodes.csv** должен содержать столбцы:

```
label,op,value,gradient,children
```

Файл **forward_mse_summary.json** должен содержать источник **microbert.py**, список добавленных членов **Value**, скалярное выражение сети, значение MSE, число параметров и имя PNG-файла с графом **Value**.

Критерии проверки

1. В **forward_mse_summary.json** указаны **square** и **sigmoid**.
2. В **layer_gradients.csv** есть параметры слоев **layer1** и **layer2**.
3. В **layer_gradients.csv** есть ненулевые градиенты для параметров обоих слоев.
4. В **autodiff_graph_nodes.csv** присутствуют операции **sigmoid** и **square**.
5. PNG-файл показывает градиенты и антиградиенты параметров по слоям.
6. PNG-файл **value_graph_from_mse.png** показывает граф **Value** от MSE к параметрам, входу **x** и целевому значению **y**.

07-lab-01. Траектории практических градиентных методов

Цель

Сравнить несколько практических методов градиентного обучения на одной квадратичной функции и увидеть, как момент, AdaGrad, RMSprop и Adam меняют траекторию параметра.

Материал лекции

Лабораторная относится к лекции 7 «Практические методы градиентного обучения» и опирается на понятия: градиентный шаг, метод тяжелого шарика, AdaGrad, RMSprop, Adam и методы второго порядка как ориентир для сравнения.

Состав шаблона

1. **lab.py** - запускаемый блокнот с ячейками **# %%**;
2. **results/** - каталог результатов после запуска.

Что нужно сделать

1. Запустите базовый вариант.
2. Измените **SELECTED_START**, **SELECTED_STEPS** или **SELECTED_LR**.
3. Сравните финальные значения функции потерь у разных методов.
4. Проверьте, как меняется график траекторий.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **optimizer_trajectories.csv**
2. **optimizer_summary.json**
3. **optimizer_trajectories.png**

Критерии проверки

1. CSV содержит столбцы **method**, **step**, **w**, **loss**, **gradient**.
2. JSON содержит лучший метод по финальному значению функции.
3. PNG показывает траектории функции потерь по шагам.

08-lab-01. Настройка обучения

Цель

Исследовать влияние расписания шага обучения, инициализации и размера мини-батча на простую задачу линейной регрессии.

Материал лекции

Лабораторная относится к лекции 8 «Настройка обучения и практический выбор оптимизатора» и опирается на выбор шага, планы изменения шага, разогрев, Xavier/He-инициализацию, мини-батчи и критерии останова.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_BATCH_SIZE**, **SELECTED_SCHEDULE** или **SELECTED_INIT_SCALE**.
3. Сравните кривые обучения и финальную ошибку.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **training_setup_history.csv**
2. **training_setup_summary.json**
3. **training_setup_loss.png**

Критерии проверки

1. CSV содержит историю **epoch**, **schedule**, **learning_rate**, **loss**.
2. JSON фиксирует выбранные параметры опыта.
3. PNG показывает кривую функции потерь.

09-lab-01. Стандартные блоки нейросетей

Цель

Собрать малую сеть из стандартных блоков, проследить размерности тензоров и сравнить статистики активаций.

Материал лекции

Лабораторная относится к лекции 9 «Стандартные вычислительные блоки нейросетей» и использует линейные слои, свертки, pooling, функции активации и нормализацию.

Что нужно сделать

1. Запустите базовый вариант.
2. Измените **SELECTED_ACTIVATION** или число каналов.
3. Сравните размерности и статистики активаций.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **block_shapes.csv**
2. **block_summary.json**
3. **activation_statistics.png**

Критерии проверки

1. CSV содержит имя блока, форму выхода и число параметров.
2. JSON содержит выбранную активацию и общее число параметров.
3. PNG показывает распределение активаций по блокам.

10-lab-01. Регуляризация и переобучение

Цель

Сравнить обучение полиномиальной регрессии без регуляризации, с L2-штрафом и с dropout-шумом в признаках.

Материал лекции

Лабораторная относится к лекции 10 «Методы регуляризации» и использует L2-регуляризацию, случайное выключение признаков и сравнение обучающей и проверочной ошибки.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_WEIGHT_DECAY** или **SELECTED_DROPOUT**.
3. Сравните разрыв между обучающей и проверочной ошибкой.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **regularization_metrics.csv**
2. **regularization_summary.json**
3. **regularization_curves.png**

Критерии проверки

1. CSV содержит train/test MSE для нескольких режимов.
2. JSON содержит лучший режим по test MSE.
3. PNG показывает сравнение ошибок.

11-lab-01. Обучение без учителя

Цель

Построить простую модель скрытых переменных для двумерных данных, сравнить реконструкцию и качество кластеризации в латентном пространстве.

Материал лекции

Лабораторная относится к лекции 11 «Обучение без учителя» и использует скрытые переменные, сжатие данных, генеративную реконструкцию и метрики качества.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_LATENT_DIM** или число эпох.
3. Сравните реконструкционную ошибку и разделимость скрытых кодов.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **unsupervised_latent_codes.csv**
2. **unsupervised_summary.json**
3. **latent_space.png**

Критерии проверки

1. CSV содержит координаты объектов и скрытый код.
2. JSON содержит реконструкционную ошибку.
3. PNG показывает скрытое пространство.

12-lab-01. Сверточные сети для изображений

Цель

Построить малую сверточную обработку изображения, сравнить карты признаков и увидеть влияние padding и stride.

Материал лекции

Лабораторная относится к лекции 12 «Сверточные сети для обработки изображений» и использует двумерную свертку, дополнение границ, шаг, pooling и карту признаков.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_KERNEL** или **SELECTED_STRIDE**.
3. Сравните форму выхода и карту признаков.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **convolution_features.csv**
2. **convolution_summary.json**
3. **feature_map.png**

Критерии проверки

1. CSV содержит координаты и значения карты признаков.
2. JSON содержит форму выхода и параметры свертки.
3. PNG показывает карту признаков.

13-lab-01. Рекуррентные сети

Цель

Развернуть простую рекуррентную сеть во времени и увидеть, как коэффициент рекуррентной связи влияет на скрытые состояния и градиентный множитель.

Материал лекции

Лабораторная относится к лекции 13 «Рекуррентные сети» и использует скрытое состояние, развертку во времени, BPTT и затухание или рост градиента.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_RECURRENT_WEIGHT**.
3. Сравните последовательность скрытых состояний и градиентный множитель.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **rnn_unroll.csv**
2. **rnn_summary.json**
3. **hidden_states.png**

Критерии проверки

1. CSV содержит **time**, **input**, **hidden**, **gradient_multiplier**.
2. JSON содержит финальное состояние и общий множитель градиента.
3. PNG показывает скрытое состояние по времени.

14-lab-01. Трансформеры и самовнимание

Цель

Вычислить матрицу самовнимания для короткой последовательности, применить причинную маску и визуализировать веса внимания.

Материал лекции

Лабораторная относится к лекции 14 «Трансформеры» и использует запросы, ключи, значения, softmax-внимание, многоголовое внимание как обобщение и причинную маску.

Что нужно сделать

1. Запустите базовый вариант.
2. Измените **SELECTED_TEMPERATURE** или включение маски.
3. Сравните веса внимания и выходные представления.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **attention_weights.csv**
2. **attention_summary.json**
3. **attention_heatmap.png**

Критерии проверки

1. CSV содержит веса внимания для каждой пары токенов.
2. JSON содержит параметр маски и форму выхода.
3. PNG показывает тепловую карту внимания.

15-lab-01. Q-обучение

Цель

Выполнить несколько итераций Q-обучения в малой среде и визуализировать изменение функции ценности действий.

Материал лекции

Лабораторная относится к лекции 15 «Обучение с подкреплением» и использует марковский процесс принятия решений, награду, дисконтирование, уравнение Беллмана и Q-обучение.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_GAMMA**, **SELECTED_ALPHA** или число эпизодов.
3. Сравните финальную таблицу Q-значений.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **q_learning_trace.csv**
2. **q_learning_summary.json**
3. **q_values.png**

Критерии проверки

1. CSV содержит трассу обновлений Q-таблицы.
2. JSON содержит лучшую стратегию для состояний.
3. PNG показывает Q-значения.

16-lab-01. Прямой процесс диффузии

Цель

Промоделировать прямой процесс зашумления, вычислить долю сигнала и шума и визуализировать траекторию **z_t**.

Материал лекции

Лабораторная относится к лекции 16 «Диффузионные модели» и использует прямой процесс зашумления, **alpha_t**, **bar(alpha)_t**, предсказание шума и MSE по шуму.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_STEPS** или диапазон **beta**.
3. Сравните траекторию сигнала и шума.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **diffusion_forward.csv**
2. **diffusion_summary.json**
3. **diffusion_trajectory.png**

Критерии проверки

1. CSV содержит **t**, **beta**, **alpha_bar**, **signal**, **noise**.
2. JSON содержит финальное **alpha_bar**.
3. PNG показывает изменение сигнала и шума.

17-lab-01. Байесовское предсказание

Цель

Сравнить несколько выборок параметров линейной модели и разложить неопределенность предсказания на среднее и дисперсию.

Материал лекции

Лабораторная относится к лекции 17 «Байесовы нейронные сети» и использует априорное распределение, MAP-связь с регуляризацией, апостериорное предсказательное распределение и эпистемическую неопределенность.

Что нужно сделать

1. Запустите базовый опыт.
2. Измените **SELECTED_PRIOR_STD** или сетку входов.
3. Сравните среднее предсказание и доверительную полосу.

Команда запуска

```
uv run python lab.py
```

Что нужно сдать

1. **bayesian_predictions.csv**
2. **bayesian_summary.json**
3. **predictive_uncertainty.png**

Критерии проверки

1. CSV содержит среднее и дисперсию предсказаний.
2. JSON содержит prior и среднюю эпистемическую дисперсию.
3. PNG показывает среднее и полосу неопределенности.