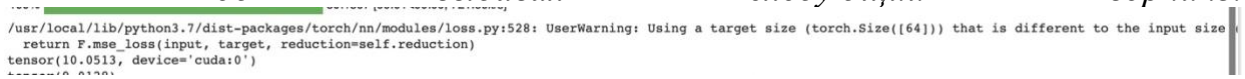
Отчет о ДЗ 1. Гогарев Игорь – ИАД-3

Мое выполнение ДЗ глобально разделилось на две части: до нахождения ошибки реализации работы нейронной сети, в результате которой качество резко увеличилось, и после.

Ошибка была в разных размерностях тензоров, с которыми работала модель – код выдавал следующий ворнинг:



До исправления данной ошибки лосс не опускался ниже отметки 10.2. Поэтому помимо тестирования разных архитектур и гиперпараметров, я предпринимал разные попытки «улучшить» данные. Я опишу эти попытки ниже, но, из-за ошибки, качество практически никогда не улучшалось, поэтому сложно оценить реальную полезность каждой из попыток.

Исследовательский анализ данных:

- Так как в Задании 0 требовалось обучить Rigde регрессию, я посмотрел на веса признаков после обучения, взял их по модулю и отсеял 20 самых неважных. Я попробовал также обучить Lasso на гиперпараметрах по умолчанию, но эта модель просто обнулила все веса.
- Далее я посмотрел на корреляцию соответствующих признаков и увидел, что несколько признаков (около 5) имеют взаимную корреляцию более, чем 0.8. В связи с этим я попробовал использовать метод главных компонент, сокращая число признаков до 80.
- Было понятно, что надо масштабировать признаки, я попробовал две реализации: StandardScaler и Normalizer. Обе давали примерно одинаковый результат.
- Далее у меня возникла идея отмасштабировать и таргет, но так как мне нужна была обратная функция для корректного вычисления лосса, я воспользовался алгоритмом, который выполняет StandardScaler: из каждого числа вычесть среднее по набору и затем разделить на стандартное отклонение. Это единственная попытка, которая даже при наличии ошибки в коде дала улучшение – именно после данного преобразования модель показывала лосс 10.2.
- Я также посмотрел на зависимость таргета от многих признаков (с помощью графиков – plt.scatter) и стало ясно, что зависимость была совсем не похожая на линейную, поэтому я пытался различными преобразованиями (np.log, np.exp, 1/x, x**(a)) повысить эту линейную зависимость, однако это не дало серьезного улучшения.

После обнаружения ошибки

Я пробовал много разных архитектур и гиперпараметров, однако в трех экспериментах я опишу три основные стратегии, которые у меня были:

- Нейронная сеть с малым числом слоев и большим количеством нейронов в каждом из них
- Нейронная сеть с большим количеством слоев и маленьким количеством нейронов в каждом из слоев
- «золотая середина» - немного слоев и не очень много (в пределах 200) нейронов в каждом слое - именно она и дала лучшее качество

Эксперимент 1:

Архитектура сети была следующая:

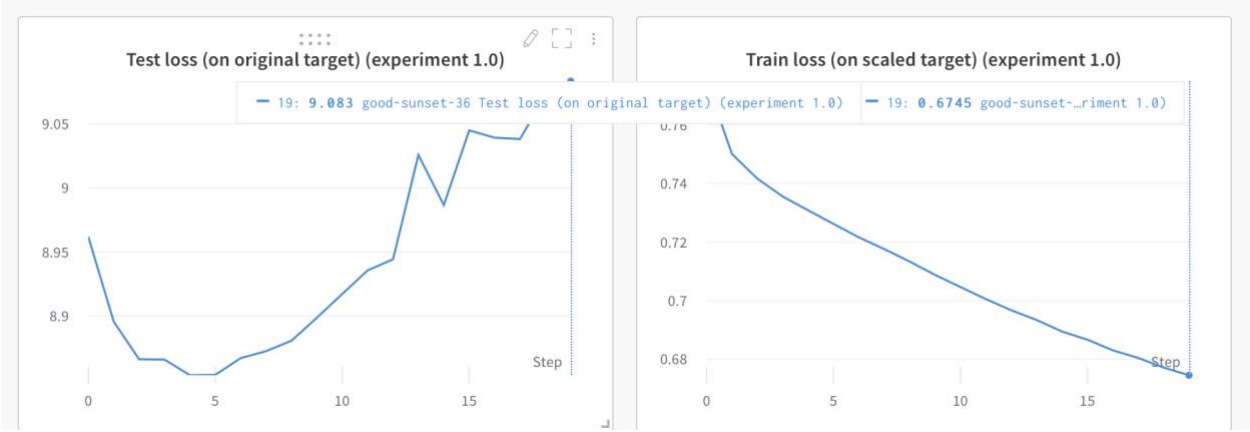
```
class CustomModel(nn.Module):
  def __init__(self, p_dropout):
    super().__init__()
    self.linear_1 = nn.Linear(in_features=90, out_features=1024)
    self.linear_2 = nn.Linear(in_features=1024, out_features=128)
    self.linear_3 = nn.Linear(in_features=128, out_features=1)

    self.relu = nn.ReLU()
    self.dropout = nn.Dropout(p=p_dropout)



  def forward(self, x):
    x = self.relu(self.linear_1(x))
    x = self.dropout(x)
    x = self.relu(self.linear_2(x))

    x = self.linear_3(x)
    return x
```

И вот такие гиперпараметры:

```
set_random_seed(42)
model = CustomModel(0.1)
optimizer = torch.optim.Adam(model.parameters(), lr=(1e-3))
criterion = RMSELoss()
```

Такая модель начала почти сразу переобучаться, поэтому я сразу отошел от идеи больших слоев.

Так как на трейне мне надо было контролировать только динамику изменения лосса, поэтому я не масштабировал обратно показатель. С тестом все в порядке – перед подсчетом метрики я отмасштабировал обратно.



Эксперимент 2:

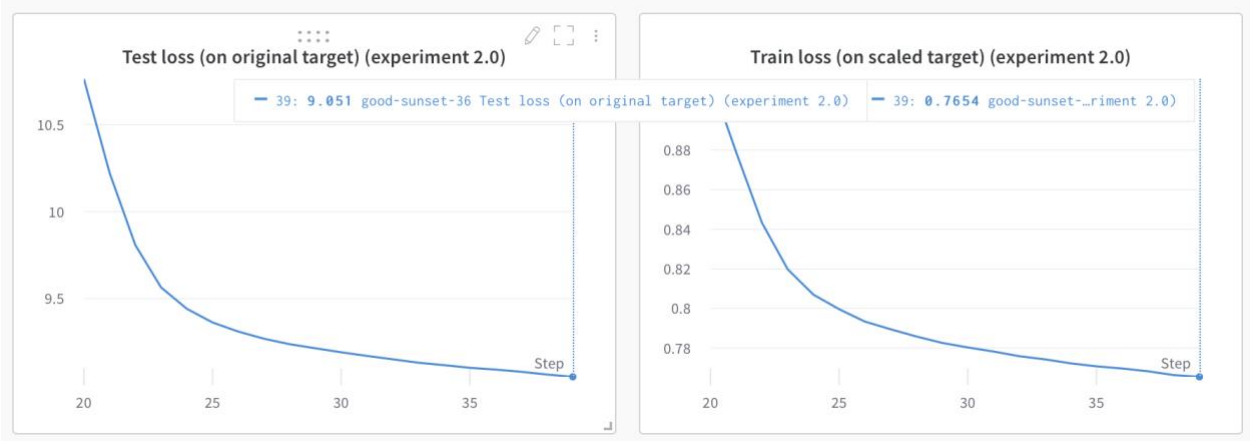
Архитектура:

```
class CustomModel(nn.Module):
  def __init__(self, p_dropout):
    super().__init__()
    self.linear_1 = nn.Linear(in_features=90, out_features=80)
    self.linear_2 = nn.Linear(in_features=80, out_features=70)
    self.linear_3 = nn.Linear(in_features=70, out_features=60)
    self.linear_4 = nn.Linear(in_features=60, out_features=50)
    self.linear_5 = nn.Linear(in_features=50, out_features=40)
    self.linear_6 = nn.Linear(in_features=40, out_features=40)
    self.linear_7 = nn.Linear(in_features=40, out_features=1)

    self.relu = nn.ReLU()
    self.dropout = nn.Dropout(p=p_dropout)


  def forward(self, x):
    x = self.relu(self.linear_1(x))
    x = self.dropout(x)
    x = self.relu(self.linear_2(x))
    x = self.dropout(x)
    x = self.relu(self.linear_3(x))
    x = self.dropout(x)
    x = self.relu(self.linear_4(x))
    x = self.dropout(x)
    x = self.relu(self.linear_5(x))
    x = self.dropout(x)
    x = self.relu(self.linear_6(x))

    x = self.linear_7(x)
    return x
```

Гиперпараметры:

```
set_random_seed(42)
model = CustomModel(0.1)
optimizer = torch.optim.Adam(model.parameters(), lr=(1e-5))
criterion = RMSELoss()
```

Данная сеть обучалась очень медленно по двум причинам: из-за большого количества слоев градиент затухал ближе к первым слоям, а также шаг был небольшим. Метрики получились следующие:



Эксперимент 3:

С архитектурой я определился практически сразу: стало понятно, что нейронов должно быть не больше 200 в каждом слое. Помимо размеров сети, я попробовал также реализовать следующую идею: не уменьшать сильно и часто число нейронов в каждом последующем слое. Таким образом сеть была следующая:

```
class CustomModel(nn.Module):
  def __init__(self, p_dropout):
    super().__init__()
    self.linear_1 = nn.Linear(in_features=90, out_features=150)
    self.linear_2 = nn.Linear(in_features=150, out_features=100)
    self.linear_3 = nn.Linear(in_features=100, out_features=100)
    self.linear_4 = nn.Linear(in_features=100, out_features=60)
    self.linear_5 = nn.Linear(in_features=60, out_features=1)

    self.relu = nn.ReLU()
    self.dropout = nn.Dropout(p=p_dropout)
```

```
  def forward(self, x):
    x = self.relu(self.linear_1(x))
    x = self.dropout(x)
    x = self.relu(self.linear_2(x))
    x = self.dropout(x)
    x = self.relu(self.linear_3(x))
    x = self.dropout(x)
    x = self.relu(self.linear_4(x))

    x = self.linear_5(x)
    return x
```

Сначала я поставил небольшой параметр для дропаута – 0.1. В качестве оптимайзера я использовал Adam с шагом 1e-4 – я пробовал периодически для одинаковых данных запускать с SGD + Momentum, но он был нестабилен, а также требовал более тщательного подбора гиперпараметров – так как я часто менял архитектуру сети и другие параметры, я решил остановиться на Adam.

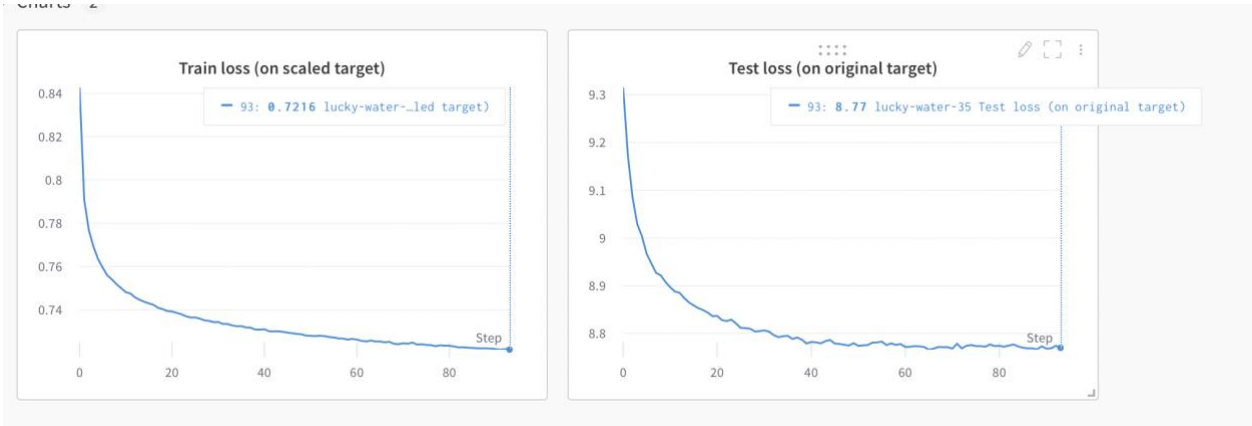
Я также пробовал разные нелинейный функции, в том числе PRELU и Tanh, но вторая видимо слишком сильно уменьшала значения нейронов, потому что обучение становилось медленнее, а первая не давала видимого улучшения.

Модель достигала лосса около 8.00 и дальше уходила вверх. Так как это показатель явного переобучения, я начал по чуть-чуть увеличивать значения

параметра дропаута. Параллельно я решил немного уменьшать шаг в Adame – я подумал, что при большем значении параметра дропаута модель становится более непостоянной и неустойчивой, а значит, каждому из шагов стоит меньше «доверять».
Итоговые параметры такие:

```
set_random_seed(42)
model = CustomModel(0.46)
optimizer = torch.optim.Adam(model.parameters(), lr=0.8 * (1e-4))
criterion = RMSELoss()
```

Я поставил 94 эпохи, на 70й качество было 8.768, далее оно по чуть-чуть колебалось, к последней эпохе оно стало 8.7698 (точное число показано в ноутбуке, на графике wandb оно округлено до 8.77).



Заключение

Несмотря на то, что большая часть моей работы была сделана в тот момент, когда в коде была ошибка, не позволяющая значению лосса быть ниже 10.2, а значит многие эксперименты оказались в каком-то смысле бесполезными, в ходе выполнения домашнего задания я вынес несколько полезных для себя идей:

- Масштабируя данные, стоит сразу попробовать отмасштабировать и таргет;
- Дропаут стоит увеличивать понемногу, начиная с небольшого значения – так можно лучше контролировать момент, когда наступает переобучение;
- Параллельно с увеличением параметра дропаута можно немного уменьшать шаг – так модель будет сходиться стабильнее;
- Стоит больше внимания обращать на возникающие ворнинги, а также тщательно проверять код перед обучением :)