Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Обработка больших объемов информации

ОТЧЕТ

на лабораторной работе №1 “Реализация линейной рециркуляционной сети как модели автодекодера для задачи понижения размерности данных”

Студент
гр. 758601
Лимонтов А. С.

Проверил
Ивашенко В. П.

Цель

Ознакомиться, проанализировать и получить навыки реализации модели линейной рециркуляционной сети для задачи сжатия графической информации (или другой информации).

Исходные данные

В качестве изображения для проведения исследований была использована следующая фотография, полученная в открытом доступе:

При считывании она сжималась стандартными средствами библиотеки `Pillow` до необходимого размера (128x128), а также преобразовывалась к черно-белому режиму для уменьшения сложности обучения.

Помимо приведенной картинки, был использован другой набор изображений для сравнительного анализа работы на них. Они все были загружены из открытого доступа, приведем некоторые из них:

Класс Autoencoder

При инициализации класса автокодировщика можно выставлять следующие гиперпараметры:

- `input_layers`: количество нейронов на входном слое (равно размеру вектора, вычисляемого из нарезанного блока)
- `lr`: learning rate
- `dynamic_lr`: флаг, отвечающий за динамический пересчет learning rate после каждой итерации, либо использовать переданное значение
- `z`: коэффициент сжатия. Из него вычисляется количество нейронов на втором слое (`int(z * input_layers)`)
- `w_norm`: флаг, отвечающий за нормировку весов после каждой итерации

Алгоритм обучения

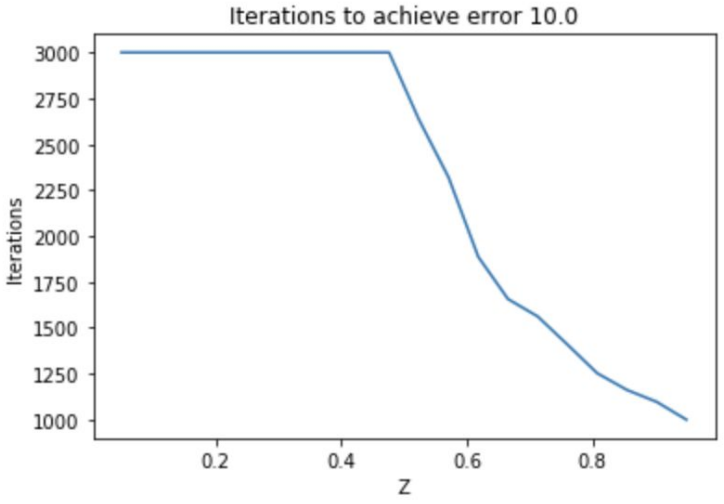
1. Инициализируем матрицы весов на первом и втором слое. Это можно сделать с помощью нормального распределения в пределах [-1..1], но здесь был использован метод `glorot_uniform`.
2. Исходную картинку разбиваем на `L` блоков заранее заданного размера `nxm`, блоки не пересекаются и следуют один за одним слева-направо и сверху-вниз.
3. Каждый блок преобразуется в вектор длины `nxm` и каждое значение пиксела приводится к интервалу [-1..1].

4. Полученный вектор “прогоняем” через оба слоя, выполняя перемножения с матрицами весов. В результате получаем вектор-результат и добавляем значение ошибки (Mean Squared Error) к общей ошибке.
5. На основе полученной суммарной ошибки на всей эпохе (всех блоках) пересчитываем веса, чтобы уменьшить ошибку.

Полученные результаты

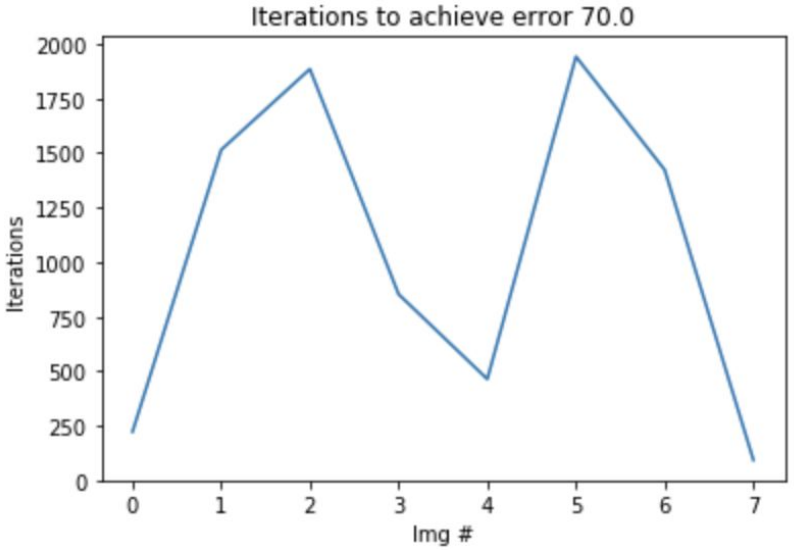
Приведем полученные графики в результате различных запусков:

1. Зависимость числа итераций обучения от коэффициента сжатия z (отметим, что больше 3000 итераций алгоритм не делал, поэтому изначально видно такое плато):



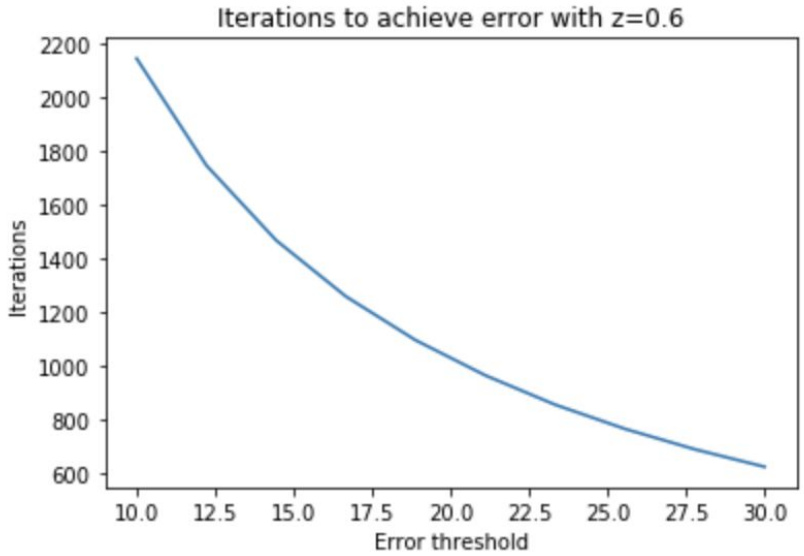
Видно, что чем больше z, тем меньше итераций необходимо выполнить, чтобы получить заданную заранее пороговую ошибку (в данном случае 10.0). Это и логично, поскольку коэффициент при большом z картинка сжимается меньше, а значит теряется меньше информации, поэтому ошибка будет меньше.

2. Зависимость числа итераций обучения для разных изображений:



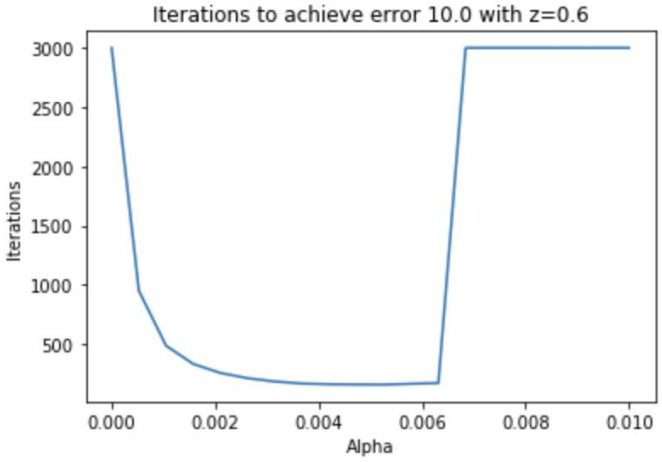
Видим, что в зависимости от содержания картинки (много однотонных областей, либо наоборот вся картинка "пестрит") алгоритм обучается неравномерно. Но в целом можно задать верхнюю планку на время обучения (в данном случае, где-то в районе 2000 эпох), для которой можно получить требуемую оценку в среднем для всех картинок.

3. Зависимость числа итераций от `e`:



Как и ожидалось, количество затраченных эпох до достижения пороговой ошибки гладко снижается.

4. Зависимость числа итераций от коэффициента обучения.Исходя из проведенных исследований, определим границы для него [1e−6..1e−2]:



Вывод по этому графику следующий: как и в других похожих алгоритмах обучения (например, в градиентном спуске) существуют две крайности задания learning rate (αα): при слишком низком значении алгоритм "недообучается", то есть показывает плохой результат еще во время обучения, при слишком же большом значении алгоритм "промахивается" мимо глобальных минимумов. Похожая ситуация проявляется и здесь: при слишком маленьком αα алгоритму необходимо очень много времени (эпох), чтобы получить хорошую ошибку (график очень резко опускается, если идти слева), затем переходит в зону, в которой αα оптимально для обучения (как видно, это достигается где-то при α=[0.004..0.006], затем начинается резкий скачок, когда αα лишь делает только хуже всему процессу и выходит из зоны минимума.

Выводы

В данной лабораторной работе мы научились производить сжатие изображений с помощью модели линейной рециркуляции сети. Достигнутые результаты можно описать так:

- Алгоритм выполняет меньшее число итераций с увеличением минимальной ошибки

- Алгоритм быстрее сходится при хорошо подобранном шаге обучения ($\alpha \approx 0.005$). Адаптивный шаг также работает неплохо, но все же при правильно подобранном α можно добиться лучших результатов/скорости
- Применение техник нормирования и адаптивного обучения сокращает число итераций не всегда помогает. Это, в целом, зависит и от начального задания матриц весов
- С уменьшением числа входов/выходов уменьшается время работы алгоритма