День 3: подробное знакомство с CNN

1 Baseline

Натренируйте базовую модель используя код из файла `deep_mnist.py`. Выведите и сохраните точность(`accuracy`) и лосс(`cross_entropy`) на train и test датасэтах в зависимости от количества итераций. Сохраните эти значения используя `numpy.save`, они нам потребуются для сравнения с другими методами.

2 Оптимизатор

Сейчас мы попробуем обучаться используя различные learning_rate и оптимизаторы.

1. Поварьируйте параметр learning_rate для стандартного оптимизатора AdamOptimizer[1], попробуйте значения 1e-6, 1e-4, 1e0, 1e3. Как меняется финальная точность.
2. Повторите эксперимент используя простейший GradientDescentOptimizer[2] и default значение learning_rate параметра. Точность должна упасть.
3. ** Для одной фиксированной переменной посчитайте вариацию градиентов для разных оптимизаторов с одинаковым learning_rate параметром. Таким образом мы проверим что Adam действительно использует скользящие средние предыдущих градиентов и вариация градиентов между шагами будет меньше.

Для справки используйте этот туториал `http://ruder.io/optimizing-gradient-descent/`.

3 Переобучение

Как известно нейронные сети склонны к переобучению, в особенности при большом количестве параметров сети. Самым простым способом предотвратить это является добавление регуляризации произвольного вида: в файле `deep_mnist.py` мы добиваемся регуляризации двумя способами - добавив dropout и сохраняя количество параметров сети небольшим.

1. Посчитайте количество параметров сети исходя лишь из архитектуры сети и её . Чтобы проверить свой ответ - сложите размеры всех переменных сети. Последние можно извлечь аналогичто строчке

   ```
   W_fc1_, b_fc1_ = sess.run([W_fc1, b_fc1])
   ```

[1] `https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer`
[2] `https://www.tensorflow.org/api_docs/python/tf/train/GradientDescentOptimizer`

из файла `deep_mnist.py`.

2. Выведите и сохраните точность(`accuracy`) и лосс(`cross_entropy`) на train и test датасэтах в зависимости от количества итераций.

3. Добейтесь переобучения сети любым из трёх способов:

   (a) Уберите дропаут

   (b) Добавьте больше kernels во всех конволюциях - и во всех нейронных сетях

   (c) Изменив архитектуру сети и добавив новых слоёв: проще всего добавить fully_connected слоёв. Добавление конволюционных слоёв не ведёт к переобучению.

   Верифицируйте переобучение повторив шаг 2.
   Замечание. Не всегда нужно ждать окончания тренировки - прерывайте тренировку как только увидим переобучение.

4. Теперь используйте аугментации для предотвращения переобучения. Возьмите модель содержащую больше параметров (или больше конволюционных блоков), которая раньше была склонна переобучению, и тренируйтесь на аугментированном датасэте. Используйте файл `simple_augmentation.py`.

5. Добавьте аугментации отражений: `np.fliplr, np.flipud` dd в файл `simple_augmentation.py`. Это должно снизить точность на тестовом датасэте.

6. Использовать сеть натренированную на MNIST для классификации USPS датасэта, который можно скачать по ссылке `https://github.com/Britefury/usps_dataset`. Для начала разберитесь чем отличаются MNIST и USPS и придумайте какую аугментацию стоит применить чтобы сделать их более похожими. Запустить

4 Pretrained VGG16

Используем pretrained сеть VGG16 для классификации изображений.

1. Классификация. Разобраться с кодом в файле `vgg16_clsf.py`. Запустить на паре примеров.

2. Сегментация. Разобраться с кодом в файле `vgg16_segm.py`. Запустить на паре примеров.

3. * Сегментация со скользящим окном.

4. ** Повторить эксперимент используя другую сеть [3]. Веса доступны по ссылке `https://github.com/tensorflow/models/tree/master/research/slim`.

[3] `https://github.com/tensorflow/models/blob/master/research/inception/inception/slim/README.md`