Методы регуляризации нейронных сетей. Часть 2.

Weight Decay

Добавление к основной целевой функции

$$L_{reg} = \frac{\lambda}{2} \sum_i |w_i|^q,$$

$$q = 2 \text{ или } q = 1$$

Пример того как отключать Weight Decay для выбранных обучаемых параметров

Weight Decay в PyTorch

```
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3,
weight_decay=1e-4)

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3,
weight_decay=1e-4)
```

CLASS `torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0, nesterov=False)` [SOURCE]

Implements stochastic gradient descent (optionally with momentum).

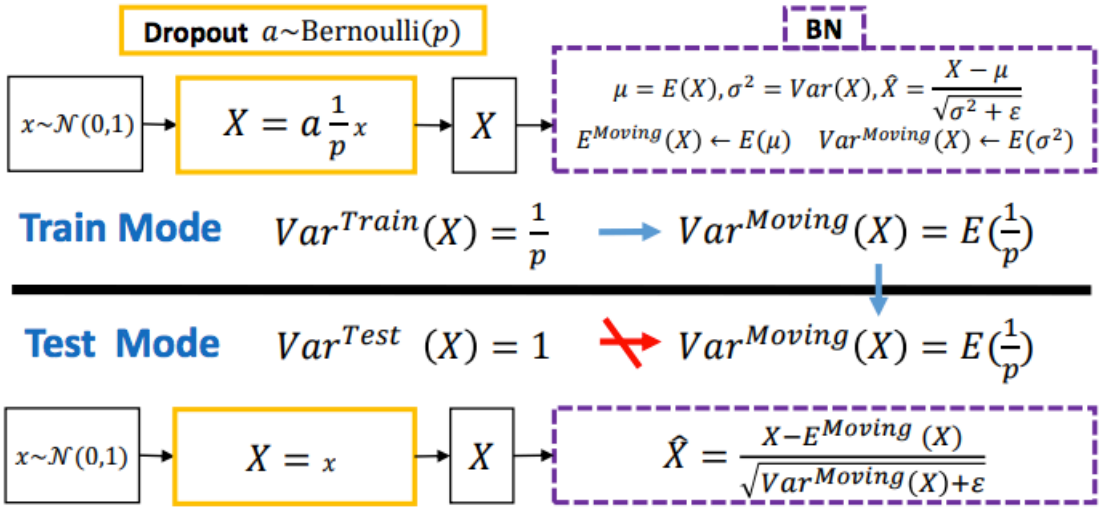
Nesterov momentum is based on the formula from On the importance of initialization and momentum in deep learning.

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – learning rate
- **momentum** (*float*, *optional*) – momentum factor (default: 0)
- **weight_decay** (*float*, *optional*) – weight decay (L2 penalty) (default: 0)
- **dampening** (*float*, *optional*) – dampening for momentum (default: 0)
- **nesterov** (*bool*, *optional*) – enables Nesterov momentum (default: False)

Dropout

Проблема (https://arxiv.org/pdf/1801.05134.pdf)



Dropout

Решение

1. Dropout после BatchNorm
2. Uout: добавление случайной величины с равномерным распределением к выходам

Dropout - Uout

$$X = x_i + x_i r_i$$
$$r_i \sim \mathcal{U}(-\beta, \beta)$$
$$0 <= \beta <= 1$$

Dropout - Uout

distributed random variable with zero mean and standard deviation equal to the activation of the unit, i.e., $x_i + x_i r$ and $r \sim \mathcal{N}(0, 1)$, we modify r as a uniform distribution that lies in $[-\beta, \beta]$, where $0 \leq \beta \leq 1$. Therefore, each hidden activation would be $X = x_i + x_i r_i$ and $r_i \sim \mathcal{U}(-\beta, \beta)$. We name this form of Dropout as "Uout" for simplicity. With the mutually independent distribution between x_i and r_i being hold, we apply the form $X = x_i + x_i r_i$, $r_i \sim \mathcal{U}(-\beta, \beta)$ in train stage and $X = x_i$ in test mode. Similarly, in the simplified case of $c = 0$, we can deduce the variance shift again as follows:

$$\frac{Var^{Test}(X)}{Var^{Train}(X)} = \frac{Var(x_i)}{Var(x_i + x_i r_i)} = \frac{v}{E((x_i + x_i r_i)^2)} = \frac{v}{E(x_i^2) + 2E(x_i^2)E(r_i) + E(x_i^2)E(r_i^2)} = \frac{3}{3 + \beta^2}. \tag{15}$$

Giving β as 0.1, the new variance shift rate would be $\frac{300}{301} \approx 0.9966777$ which is much closer to 1.0 than the previous

Label Smoothing

Используется и для борьбы с неправильной аннотацией.

$$E = -\sum_i \sum_k t_k^i log(y_k(x_i))$$

One-hot encoding: $t_i = (0, 0, 1, 0), t_i^k = 1, k = 2$ <- единица там, где правильный класс для данного наблюдения.
Меняем One-hot encoding вектор:

$$t_i = (\frac{\epsilon}{\text{C} - 1}, \frac{\epsilon}{\text{C} - 1}, 1 - \epsilon, \frac{\epsilon}{\text{C} - 1}),$$

где C -- кол-во классов

Эвристики проверки нейронной сети

Начните с данных

Посмотрите, что

1. Они нормализованы
2. Категориальным переменным соответствует верное число класса, которыми они кодируются
3. Лейблы корректны для каждого наблюдения
4. Если есть предобработка данных, либо какая-то специфичная функция формирования батча, проконтролируйте, что лейблы по-прежнему правильные для каждого наблюдения
5. Выводите ошибки по каждому батчу, например, если это картинка, то каким картинкам присваиваются неверные классы.
6. Продебажьте полный pipeline, что на каждом шаге получается с данными: считывание, предобработка, подача в сеть.

Зафиксируйте **random seed** (везде!)

```
torch.manual seed(0)
np.random.seed(0)
```

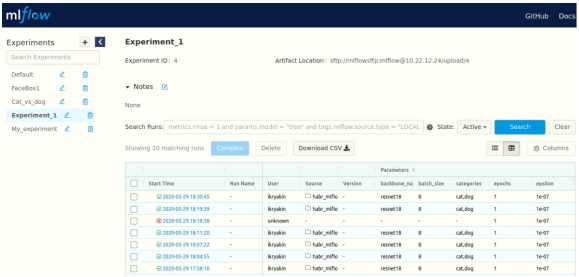
Воспроизводимость в PyTorch: отдельная статья из официальной документации насчет того, как получать максимально идентичные результаты от запуска к запуску

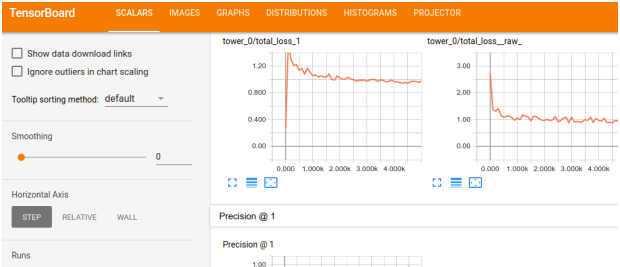
Начинайте с **простого**

Начинайте с простой предобработки данных.
Только после получения логичных результатов начинайте усложнение.

Логируйте все, что можно

Используйте TensorBoard, MLflow.





Risk
modelling.research

log(1 / кол-во классов)

Проверьте, что в самом начале loss выдает значение
log(1 / кол-во классов)
для корректного класса.
Если не так, то скорее всего надо исправить инициализацию весов, в основном, править надо самый первый или последний слой.

Инициализация весов

Проверьте, что именно стоит по умолчанию.
Опираясь на распределение данных, возможно, стоит поправить среднее или стандартное отклонение.
Также не забывайте про баесы, не всегда нужно инициализировать их нулями.

Посмотрите, что обученная модель лучше **случайно** проинициализированной модели.
Либо лучше модели, которая на вход принимает одни нули.

Заоверфитьте на одном батче

Проверьте, что значение целевой функции сильно близко к нулю, а точность под 100 %

Don’t be a hero!

Выбирайте такую архитектуру, которая уже дала хороший результат.
Берите то, что пишут в статьях, но не сложное.
Усложняйте модель постепенно.

Используйте советы, которые дают другие.
Берите к сведению любые эвристики.
Например, начинайте с Adam и ReLU.

Нет понятия маленького или большого Learning Rate.

Есть маленький и большой Learning Rate для Вашей (!) задачи.
Обращайте на его начальное значение.
Вначале делайте его константным, только после получения логичных результатов вводите политику его изменения.

Не переусердствуйте с **Dropout, BatchNorm и Weight Decay**.

Просто дайте нейронной сети **время**.
Не всегда сеть может обучиться быстро.