stochastic_estimation

April 24, 2018

연구자: 정회희 (2018/04/23

해당 코드는 *Understanding Black-box Predictions via Influence Functions* 논문의 Stochastic estimate을 구현한 코드이다.

이 코드는 논문 원작자가 제공한 코드에서 몇군데 필요없는 부분을 조금 수정한 것이다.

이 코드는 논문에서 언급한 알고리즘과 조금 다른데, 이 글을 통해서 어느 부분이 달라졌는지, 원작자는 왜 다르게 했는지, 유효한 modification인지를 파악할 것이다.

```
In [2]:     def get_inverse_hvp_lissa(self, v,
                                      batch_size=None,
                                      scale=10, damping=0.0, num_samples=1,
        recursion_depth=10000):
                """
                This uses mini-batching; uncomment code for the single sample case.
                """
                inverse_hvp = None
                print_iter = recursion_depth / 10

                for i in range(num_samples):
                    # samples = np.random.choice(self.num_train_examples, size=recursion_depth)

                    cur_estimate = v

                    for j in range(recursion_depth):

                        # feed_dict = fill_feed_dict_with_one_ex(
                        #   data_set,
                        #   images_placeholder,
                        #   labels_placeholder,
                        #   samples[j])
                        feed_dict = self.fill_feed_dict_with_batch(self.data_sets.train,
        batch_size=batch_size)

                        feed_dict = self.update_feed_dict_with_v_placeholder(feed_dict,
        cur_estimate)
                        hessian_vector_val = self.sess.run(self.hessian_vector,
```

```
        feed_dict=feed_dict)
                    # gradient is summed among minibatch (since def loss <- total loss, not
        loss vector)
                    # v, cur_estimate, hessian_vector_val are vectors i.e. R^|v| (no
        minibatch dimension)
                    # cf) for feed_dict list concatenation doesn't matter
                    # since we do batch sampling, each point of iteration is changed at
        every recursion
                    cur_estimate = [a + (1-damping) * b - c/scale for (a,b,c) in zip(v,
        cur_estimate, hessian_vector_val)]

                    # Update: v + (I - Hessian_at_x) * cur_estimate
                    if (j % print_iter == 0) or (j == recursion_depth - 1):
                        print("Recursion at depth %s: norm is %.8lf" % (j,
        np.linalg.norm(np.concatenate(cur_estimate))))

                if inverse_hvp is None:
                    # element wise division (divide values by scale among batch_size)
                    inverse_hvp = [b/scale for b in cur_estimate]
                else:
                    # summation among trials (num_samples)
                    # still, inverse_hvp has batch_size X |v| dimension
                    inverse_hvp = [a + b/scale for (a, b) in zip(inverse_hvp, cur_estimate)]
            inverse_hvp = [a/num_samples for a in inverse_hvp]
            return inverse_hvp
```

0.1 Theoretical Backgrounds

Taylor series expansion을 통해서

$$H_j^{-1} \triangleq \sum_{i=0}^{j} (I - H)^i$$

Hessian matrix의 inverse를 approximate할 수 있다. 이 식을 새로 정리하면 다음과 같은 점화식을 얻을 수 있다.

$$H_j^{-1} = I + (I - H)H_{j-1}^{-1}$$

우리는 이 점화식을 반복하여 진행하며 Hessian의 inverse를 추정할 것이다. 이 점화식을 반복하기 위해서는 H를 계산해야한다. 이 때 H는 $\frac{1}{n}\sum_{i=0}^{n} \nabla^2 L(z_i, \hat{\theta})$ 값으로 이 값 정확하게 구하려면 지나치게 많은 계산량을 필요로 한다. 때문에 이를 unbiased estimator인 $\nabla^2 L(z_i, \hat{\theta})$로 대처하여 iteration을 진행할 것이라는게 논문의 내용이다. 이 때 unbiased estimator of Hessian을 $\tilde{H}$라고 하자. 실제 구현 코드에서는 unbiased estimator $\tilde{H}$를 좀 더 잘 (낮은 variance로 추정하기) 구하기 위해서 몇 가지 방법을 제시하고 있다. 나는 이 방법을 임의로 minibatching, scaling이라고 명명했다.

Minibatching 하나의 sample만을 가지고 unbiased estimator를 사용하는 대신, 여러 sampling에 대해서 평균취하는 방법이다. 평균을 취했기 때문에 mean값에는 변화가 없고, variance는 줄어들 것으로 기대할 수 있다. 이 때 sample 수는 m개이고 당연히 $m << n$이다. e.g., Gaussian random variable이라고 가정할 경우 mean은 변함없고, variance는 $\frac{1}{m^2}$로 줄어들 것이다.

$$
\begin{aligned}
\tilde{H} &= \nabla^2 L(z_i, \hat{\theta}) \\
&\rightarrow \frac{1}{m}\sum_{i=0}^{m} \nabla^2 L(z\_i, \hat{\theta})
\end{aligned}
$$

Scaling H를 추정하는 대신 $H' \triangleq \frac{H}{Scale}$을 추정하는 것이다. 상수배로 나눠었기 때문에 variance가 줄어들 것이라고 예측할 수 있다. e.g., Gaussian random variable이라고 가정할 경우 variance가 $\frac{1}{Scale^2}$ 만큼 줄어들 것이다. 이는 minibatching과 다르게 하나의 step에서 실질적인 계산량은 변함없이 정확도를 향상시킬 수 있다. 하지만 수렴을 위해서 더 많은 recursion step (j)을 요구한다. 때문에 scaling factor는 learning rate와도 연관지어 생각할 수 있다.

$$
\begin{aligned}
{H'}_j^{-1} &= I + (I - H'){H'}_{j-1}^{-1} \\
&= I + (I - \frac{H}{Scale}){H'}_{j-1}^{-1}
\end{aligned}
$$

그리고 이 값은 ${H'}_j^{-1} = \left(\frac{H_j}{Scale}\right)^{-1}$를 추정한 것이기 때문에 H_j^{-1}을 얻기위해선 iteration을 통해 얻은 최종값을 Scale값으로 나눠야한다.

cf) Talor expansion이기 때문에 initialization 값은 변함이 없다, i.e., ${H'}_0^{-1} = I$.

Damping 마지막으로 code에는 damping term이 있다. 이 term은 앞서 Influence function을 얻을 때 strictly convex와 twice differentiable이라는 두 조건 중 strictly convex 조건을 더 강력하게 맞추기위해서 생겨난 regularization term이다. 특히 deep neural network 같은 경우에는 비용함수가 strictly convex라는 보장이 전혀 없기 때문에 꼭 필요한 term이다.

정확하게 말하자면 positive eigenvalue를 얻기위해 임의로 identity matrix를 더해주는 것으로 수식으로 표현하자면 다음과 같다.

$$
\bar{H} = \tilde{H} + \lambda I
$$

이 $\bar{H}$를 점화식에 넣게 되면 다음과 같은 식을 얻을 수 있다.

$$\begin{aligned} H_j^{-1} &= I + (I - \bar{H})H_{j-1}^{-1} \\ &= I + \left((1-\lambda)I - \tilde{H}\right) H_{j-1}^{-1} \end{aligned}$$

0.2 Implementations

이 방법을 통해서 찾고자하는 최종 값은 θ와 차원이 같은 임의의 vector v와 Inverse of Hessian matrix의 곱, $H^{-1}v$ 값이다. 따라서 위 점화식의 양 변에 v를 곱하고 추정할 vector를 s_j로 새로 정의하면, i.e., $s_j \triangleq H_j^{-1}v$, 다음과 같은 반복식을 얻을 수 있다.

$$s_j = v + s_{j-1} + \bar{H}s_{j-1}$$

Strictly convex 조건을 만족하기위해서 damping term을 풀면 다음과 같은 식을 얻을 수 있다.

$$s_j = v + (1-\gamma)s_{j-1} + \tilde{H}s_{j-1}$$

위에서 설명한 minibatching과 scaling을 포함하여 점화식을 얻으면 다음과 같다.

$$s'_j = v + (1-\gamma)s'_{j-1} + \frac{\frac{1}{m}\sum_{i=0}^{m}\nabla^2 L(z\_i, \hat{\theta})}{Scale}s'_{j-1} \text{ where } s'_j = Scale \cdot s_j$$

실제 구현된 코드의 input과 theoretical background의 hyperparameter들과 관련하여 설명하자면

1. $\gamma \leftarrow$ damping : logistic regression 같이 strictly convex한 loss를 사용할 경우 0.0으로 두면 된다. 논문에서는 $\lambda = 0.01$로 사용해서 CNN network를 재 조정했다.
2. $Scale \leftarrow$ scale : 앞서 설명했듯 H에 대해서 iteration을 반복할 때 H를 scale로 나눠서 진행하는 것. vector s의 변화량이 줄어들기 때문에 variance가 줄어들 수 있겠지만 더 많은 recursion step이 필요할 것이다. recursion step을 거쳐서 얻은 후 다시 scale로 나누는 것을 잊지 말 것.
3. $m \leftarrow$ batch_size : 여러 sample을 사용하여 평균취하는 것으로 unbiased estimator를 얻은 것. 클 수록 좋겠지만 rtp * batch_size만큼 계산량이 느는 것이라 주의해야함. 실제 구현 단계에서는 total loss 자체가 minibatch에 대해서 reduce sum한 것이기 때문에 feed_dict만 minibatch wise로 하면 된다.

$O(rtp)$

r: num_samples

t: recursion_depth

p: dimension of θ

0.3 Furtherworks

시간이 남는다면 위 3개 hyperparameter의 효과를 바꿔가면 찍어볼 것.