IntroML - Lecture Notes Week 7

Ruben Schenk, ruben.schenk@inf.ethz.ch

July 15, 2022

0.1 Stochastic Gradient Descent for ANNs

The **stochastic gradient descent** approach for ANNs looks as follows:

Algorithm: We want to solve

$$W^* = \arg\min_W \sum_{i=1}^n l(W;\, x_i,\, y_i)$$

1. We initialize the weight W
2. For $t = 1,\, 2, \ldots$:
   - Pick data point $(x,\, y) \in D$ uniformly at random
   - Take step in *negative gradient* direction

$$W \leftarrow W - \eta \Delta_W l(W;\, x,\, y)$$

Typically we use *minibatches* to reduce variance/exploit parallelization.

But how do we optimize over weights? We want to do **empirical risk minimization,** i.e. jointly optimize over all weights for all layers to minimize loss over the training data. This is in general a *non-convex* optimization problem! Nevertheless, we can still try to find a local optimum.

Remark: There are **weight-space symmetries.** Multiple distinct weights can compute the same predictions. In other words, multiple local minima can be equivalent in terms of input-output mapping.

Computing the gradient is done through backpropagation in matrix form:

Algorithm:

1. For the output layer
   - Compute "error": $\delta^{(L)} = \Delta_f l$
   - Gradient: $\Delta_{W^{(L)}} l = \delta^{(L)} v^{(L-1)T}$
2. For each hidden layer $l = L-1 : -1 : 1$
   - Compute "error": $\delta^{(l)} = \phi'(z^{(l)}) \odot (W^{(l+1)T} \delta^{(l+1)})$
   - Gradient: $\Delta_{W^{(l)}} l = \delta^{(l)} v^{(l-1)T}$

0.2 Weight Initialization in Neural Networks

Onr problem we might encounter are **vanishing** and **exploding gradients.** Remember the gradient computation in backpropagation was defined as:

$$\Delta_{W^{(i)}} l = \delta^{(i)} v^{(i-1)T} \text{ where } \delta^{(i)} = \phi'(z^{(i)}) \odot (W^{(i+1)T} \delta^{(i+1)}) \text{ and } v^{(i)} = \phi(W^{(i)} v^{(i-1)})$$

The potential issue is exploding ($||\Delta_{W^{(i)}} l|| \to \infty$) or vanishing ($||\Delta_{W^{(i)}} l|| \to 0$) gradients can cause optimization to fail. Why can this happen? Potential reasons are $||\delta^{(i)}||$ going to 0 or ∞ (or analog. for

$||v^{(i)}||$).

- Using certain activation functions (e.g. ReLU) can help avoid $||\delta^{(i)}|| \to 0$.
- The error signal $\delta^{(i)}$ is scaled by $v^{(i)}$. This can help to reduce the vanishing/exploding gradient problem by keeping the magnitude of $v^{(i)}$ constant across the layers.

The general goal when initializing weights is to keep the variance of weights approximately constant across layers to avoid vanishing and exploding gradients and network activations. Usually, **random initialization** works well, e.g.

- Glorot (tanh): $w_{i,\,j} \sim \mathcal{N}(\frac{1}{n_{in}})$ or $w_{i,\,j} \sim \mathcal{N}(\frac{2}{n_{in}+n_{out}})$
- He (ReLU): $w_{i,\,j} \sim \mathcal{N}(\frac{2}{n_{in}})$

We need to ensure that at initialization, unit activations are approximately standardized.

0.3 Learning Rates

To implement the SGD update rule ($W \leftarrow W - \eta_t \Delta_W l(W;\, x,\, y)$), we need to choose the learning rate η_t. In practice, we often use a decaying learning rate schedule, e.g. piecewise constant.

We may want to monitor the ratio of weight change (gradient) to weight magnitude:

- If it's too small, we increase the learning rate
- If it's too large, we decrease the learning rate

Learning with momentum is a common extension to training with (stochastic) gradient descent. It can help to escape the local minima. The idea is as follows: We move not only into the direction of the gradient, but also in direction of the last weigh update. The updates then are:

- $d \leftarrow m \cdot d + \eta_t \Delta_W l(W;\, x,\, y)$
- $W \leftarrow W - d$

In some cases, learning with momentum cad *prevent oscillation.*

0.4 Recularization in Neural Networks

Neural networks have many parameters, so there's a potential danger of overfitting. Countermeasures are:

- *Regularization (weight decay):* Add penatly term to keep the weights small

$$W^* = \arg\min_W \sum_{i=1}^{n} l(W;\, x_i,\, y_i) + \lambda ||W||_2^2$$

- *Early stopping:* Don't run SDG until convergence
- *"Dropout"*

The idea behind **early stopping** is as follows: In general, we might not want to run the training until the weights convergence since this potentially leads to overfitting. One posssibility is:

1. Monitor the prediction perfomance on a validation set
2. Stop the training once the valdiation error stops to decrease