Week 1, Lecture 2 - Multi-task learning

In a traditional supervised ML paradigm, you have a labeled dataset, $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})_k\}$, where the goal is to minimize some loss function ($\mathcal{L}$) given $\mathcal{D}$, by finding relevant parameters, θ (i.e., the **task**):

$$\min_{\theta} \mathcal{L}(\theta, \mathcal{D}).$$

Multi-task setup

The setup is the same as above, but instead of optimizing one objective, we want to optimize several (under the same model!) at once.

$$\text{Solve multiple tasks } \mathcal{T}_1, \cdots, \mathcal{T}_T \text{ at once.}$$

Example (credit):

instead of doing this (as usual):

we do this:

Three components:

- Model
- Loss (objective)
- Optimization

Model

We ask, should we share parameters across tasks? Or should the learnable parameters in the architecture be entirely separate for each task? The latter may look something like this (credit: CS 330 slides):

and the former:

Loss

To obtain the loss, we perform a simple summation over each of the individual task's loss:

$$\min_{\theta} \sum_{i=1}^{T} \mathcal{L}_i \left(\theta, \mathcal{D}_i\right)$$

Of course, some tasks might be important than others so we can performed a weighted summation:

$$\min_{\theta} \sum_{i=1}^{T} w_i \mathcal{L}_i \left(\theta, \mathcal{D}_i\right)$$

Note that we can update the weights during training (as a learnable parameter) or manually determime the weights of each loss beforehand (a hyperparameter).

Optimization

Optimization is straightforward and is nearly the same as in the supervised setting. The CS 330 slides defines the following sequence:

1. Sample mini-batch of tasks $\mathcal{B} \sim \{\mathcal{T}_i\}$
2. Sample mini-batch datapoints for each task $\mathcal{D}_i^b \sim \mathcal{D}_i$
3. Compute loss on the mini-batch: $\hat{\mathcal{L}}(\theta, \mathcal{B}) = \sum_{\mathcal{T}_k \in \mathcal{B}} \mathcal{L}_k \left(\theta, \mathcal{D}_k^b\right)$
4. Backpropagate loss to compute gradient $\nabla_\theta \hat{\mathcal{L}}$
5. Apply gradient with your favorite neural net optimizer

Note: one place where multi-task loss & optimization differ from the traditional supervised learning paradigm is that we can weight the tasks differently. That is, we can weight the losses differently corresponding to different tasks. This is a key hyperparameter to pay attention to in the multi-task setting. Explicity, we can assign more weight (zero-sum total) to a task that we (human) deem as more important. However, we can also realize that, during training, one task might outweigh the others and cause bad performance for the other tasks. As such, it may be a good idea to weight the "weaker" tasks stronger than the "stronger" ones. And by stronger, I mean tasks that, if evenly weighted for all tasks, would dominate the optimization procedure (the model params would be biased towards this task).[1]

[1] Question: can we treat the task weights as actual parameters (not hyperparameters) that we update via the optimization process (either explicitly or implicitly)?

PyTorch Example

Coming soon... see this blog post for an example.

Applications

- Tesla uses a multi-task "hydranet" model for its inference in self-driving cars. The vision-only system takes in a tuple of 8 images from the car's cameras, passes the concatenated feature vector through a shared backbone encoder, and separates each task via a decoder head. For example, Tesla uses one network for things like depth estimation, lane segmentation, etc. See here.

Confusions/quesions

- Does the input feature vector need to be the same for each task? (I believe the answer is no)

References

- CS 330
- Tutorial blog