ECE472 – Quiz 6

Jonathan Lam

October 14, 2020

1. *Explain MAML carefully... (think gradients)* (arXiv:1703.03400)

   Model-Agnostic Meta-Learning (MAML) is aimed towards making a network more sensitive to the tasks for which it is intended for use (e.g., regression, classification). This means that the network will be able to more quickly train with fewer examples (e.g., in the case of one- or few-shot training). As opposed to prior attempts at this, which usually involve adding more parameters or using a specialized architecture aimed at optimizing learning patterns ("meta-learning"), the authors of this paper surmise that a network can be pre-trained so that the initialized weights throughout the network generally tend to be more sensitive to the changes in the inputs (i.e., they will have a larger change in gradient for the kinds of tasks the network is intended for). They state, "the intuition behind this approach is that some internal representations are more transferrable than others." Therefore, MAML is essentially an initialization method, in that it does not depend on or affect the structure of the network (and can be applied to many different learning tasks and network structures, as the authors demonstrate).

   In particular MAML works by having a pre-training session that involves optimizing weights such that *the potential for gradient optimization upon training is maximized.* This is just another optimization (gradient descent) problem that embeds the original optimization (gradient descent) problem, and is all encoded in this formula:

   $$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta'_i}\right) = \min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}\left(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}\right)$$

   The expression on the left indicates the intent: we want to find the weights that would, when trained on a few tasks (producing θ', the weights after the stochastic gradient descent rule) picked from the tasks training dataset with a representative sample distribution (i.e., $\mathcal{T}_i \sim p(\mathcal{T})$), minimize the loss. In other words, we aim to choose the set of weights that cause the greatest decrease in loss after it an update rule (on a representative distribution of tasks), which can be thought of as increasing the sensitivity of the loss to updates. Note that this requires some knowledge of the distribution of tasks during pre-training, and involves calculating a second-order gradient (gradient of a gradient) (or at least approximating this with a first-order approximation).

2. *What are the main benefits and weaknesses of Neural ODEs?* (arXiv:1806.07366)

   Benefits :

   Memory efficiency From an intuitive perspective, there are no more layers like in the more traditional ResNet, but rather a differentiable state function. While we had to store much intermediate state for weights in order to propagate gradients in the more traditional structure (making memory usage linear w.r.t. number of layers), using the adjoint sensitivity model for reverse-mode autodiff is much more memory efficient (roughly constant, according to Table 1 in the report).

   ODE solvers can be treated as a black box. Not explicitly mentioned as a benefit, but this allows for existing ODE solvers to be somewhat plug-and-play, because the described implementation doesn't require any knowledge about the internal state of the ODE solver.

   Powerful existing ODE solvers The authors cite a long and successful history of ODE solvers. Some advanced ODE solvers use adaptive methods that scale really well to large networks. Next point is related.

   Adjustable precision/computing power Some of the ODE solvers allow for customizable levels of accuracy.

   "Scalable and invertible normalizing flows" I'm not too sure about what this means, because we did not cover the normalizing flow task, but neural ODEs are supposed to be beneficial for this type of problem.

   Arbitrary evaluation points This can be thought of as nicely accommodating for interpolated or extrapolated data points. This is nice, for example, if most of the data comes in discretized measurements (e.g., at specific points in time) but now you want to provide an inference on a data point that came at some in-between value. My (very layman's) understanding of this is that having the ODE solution be a differentiable (and thus smooth) function rather than being discretely parameterized like in a traditional ResNet allows for this smooth interpolation and extrapolation. The extrapolation proficiency of neural ODEs is demonstrated in Figure 8.

   Drawbacks :

   Non-uniqueness There is not always a unique solution to an ODE, so this is a potential problem. However, the authors note that finite weights (which can be encouraged by regularization) and traditional nonlinearities (e.g., activations) like tanh and ReLU don't pose this problem.

   Minibatching is inefficient Minibatching, which is a common theme in SGD, may be more inefficient. My intuition for this is that an ODE only allows a fixed-size input (and a single gradient update) at a time; the authors mention that a mini-batch can be made by concatenating the batch samples' states, but this may require more computation than if doing each sample separately.

3. *Does magnet loss require any extra label information per example compared to softmax cross entropy?* (arXiv:1511.05939)

   No.

   The authors mention at the beginning that traditional classification methods (i.e., not distance metric learning (DML)) tend to discard all but the label information by the end of the network (and thus these are not well-suited to classification on different classes). DML attempts to make the classification retain more information up until the last step (i.e., the inputs get encoded into some vector space, and then similarity is measured by some function of distance in that vector space), which offers some benefits, e.g., zero-shot learning and easier visualization of similarity.

   However, the only supervision given is still the labels (just like in softmax) – the algorithm provided generates a vector representation for DML, but this is all unsupervised. In other words, the labels provide guidance for the results to be clustered into "classes," but there is an unsupervised subclustering within those classes into smaller "clusters" using a K-means method and a loss function that penalizes cluster overlap. This allows for magnet loss to generate multi-dimensional vectors with some notion of similarity without supervised guidance.