Derek Lee Deep Learning Fall 2020

Professor Curro Quiz #6

1. Explain MAML carefully… (think gradients)

MAML is designed for rapid adaptation, allowing a model to quickly learn. MAML can be combined with any gradient-based training model and any differentiable objective. In addition, it does not introduce any learned parameters. It is merely a weight initialization. The goal is to find model parameters that are “sensitive to changes in the task, such that small changes… will produce large improvements on the loss function”. This is accomplished by using SGD, where the gradient update is obtained by multiplying the gradient of the gradient by β, the meta step size. This computation requires an additional backwards pass to compute Hessian vector products. The meta-optimization is performed on the parameters θ, while the objective is computed using the updated parameters θ’, obtained using SGD. This meta-optimization is significant. Unlike other learning algorithms, where θ becomes θ’, in MAML, θ is updated using the gradient of the loss function, which is computed using θ’.

The authors show an example for regression of a sine wave. The model is initially trained on a specific sine wave, and later fine-tuned on a different sine wave. The MAML model was able to quickly adapt compared to a model without MAML. The authors suggest MAML optimizes parameters such that the parameters lie in a region that is “amenable to fast adaptation and is sensitive to loss functions”.

The authors note that computing second derivatives is computationally expensive and tested a first-order approximation of MAML. It performed almost the same as a second-order MAML, which suggests that “most of the improvement in MAML comes from the gradients of the objective at the post-update parameter values, rather than the second order updates from differentiating through the gradient update.” The authors explained this by noting past work has shown that ReLUs are “locally almost linear”, which means second derivatives are close to zero.

2. What are the main benefits and weaknesses of Neural ODEs?

Defining a model using ODE solvers is memory efficient. Gradients can be computed without backpropagating through the ODE solver, which allows for training with “constant memory cost as a function of depth”. It can also be used for adaptive computation. Modern ODE solvers monitor the approximation error and adapt their evaluation strategy to achieve the desired accuracy. The computational cost can therefore scale with the problem complexity. In addition, the change of variables formula becomes easier to compute, which can be used to “construct a new class of invertible density models that avoids the single-unit bottleneck of normalizing flows”. Finally, it can “naturally incorporate data which arrives at arbitrary times” because of its continuously-defined dynamics.

The authors propose using the adjoint sensitivity method to perform backpropagation through the ODE solver. Instead of differentiating through the operations of the forward pass, which has a high memory cost and introduces additional error, the authors use a separate ODE backwards in time. The separate ODE is used to compute the adjoint, which is equal to the gradient of the loss at each instant. This is linear in the size of the problem, controls numerical error, and has low memory error. Surprisingly, “the number of evaluations in the backward pass is roughly half of the forward pass.” Thus, the adjoint sensitivity is both more memory efficient and computationally efficient than directly backpropagating, which would require backprop through each function evaluation in the forward pass.

Neural ODEs have some limitations. For example, using mini-batches requires concatenating the states of each batch element, creating a D by K ODE. It is possible that “controlling error on all batch elements together might require evaluating the combined system K times more often than if each system was solved individually”, but the authors did not observe a substantial change in the number of evaluations in practice. Additionally, reconstructing forward trajectories during the adjoint sensitivity method “can introduce extra numerical error if the reconstructed trajectory diverges from the original.” The authors state that the problem can be solved with checkpointing and did not find this to be a practical problem.

3. Does magnet loss require any extra label information per example compared to softmax cross entropy?

Magnet loss is proposed as an alternative to metric learning approaches. Metric learning approaches must “define a relationship between similarity and distance… similarity has been canonically defined a-priori by integrating available supervised knowledge.” In practice, examples from the same class are “tightly clustered together, far from examples of other classes”. This strategy “collapses intra-class variation and does not embrace shared structure between different classes.”

Magnet loss uses a soft k-nearest cluster metric. Instead of previous approaches, which define similarity based on distances in the original input space, magnet loss defines similarity based on distances in representations. It purses local separation, rather than global separation, which allows class distributions to be interleaved. The distance metric learning algorithm can then penalize the local overlap. This does not assume unimodality, nor does it assume “unreasonable prior target neighbourhood assignments”. Instead of manipulating individual examples, the objective will jointly manipulate entire clusters. Not only is this more computationally efficient, but it is also more consistent, which does not hinder the convergence rate.

Magnet loss does not require any extra label information. In fact, compared to other algorithms, it uses less. Other algorithms use “additional prior information, such as similarity ranking and hierarchical class taxonomy”. However, in practice, class labels are the “only available supervision”. Magnet loss uses a novel approach, not additional data.