CIS 7000-005: Debugging Data & Models Fall 2022

Scientific Discovery — October 20

Prof. Eric Wong

Explainability had a goal of explaining model decisions, or inspecting specific parts of a neural network model. Verification also has a well-defined goal: testing whether a property of a neural network holds or not. But what if we don't know a priori what exactly we should be looking or testing for? Finding or identifying patterns or biases of potential concern is the process of scientific discovery.

- How can we find correlations or biases that the model picked up on?
- How do we describe a correlation or bias in the first place?
- How do you identify whether such a bias is good or bad in the first place?

1 Influences

One way to find these kinds of patterns is to ask the following question: what were the most influential data points that shaped my model's decision boundary? For example, imagine a linear regression problem but with a few outliers. Outliers in linear regression have a large effect on the resulting classifier due to the square error, which severely penalized larger errors.

$$\min_\beta \sum_i (x_i^T \beta - y_i)^2$$

This can be mitigated to some degree by using, for example, the huber loss function instead of quadratic).

$$\ell(a) = \begin{cases} a^2 & \text{if } |a| \leq 1 \\ |a| & \text{otherwise} \end{cases}$$

But now let's say we don't know a priori that there were in fact outliers that drastically changed the model prediction. How could you discover this? One way is to search fo influential data points.

Deletion The basic idea here is to ask the following counterfactual: for a given datapoint x, what would have happened if I didn't have this datapoint? The difference in the resulting models is often described as the influence of that example. A simple way to measure this is the following:

1. Train a model f on a dataset D
2. Train a model f' on a dataset $D \setminus \{x\}$ for a particular example x
3. Evaluate the accuracy of f and f' and take the difference to get the influence of x.

While straightforward, this whole process has several challenges:

1. Retraining can be slow
2. Datasets are large, so deletion has little effect (possibly get around by grouping datapoints)
3. Differences in accuracy after deletion of one example can be drowned out by noise in the training process for deep networks (not a problem for models with unique solution)

1.1 Influence functions

One way to get around retraining is to use what is known as an influence function. This is a measure of how strongly parameters or predictions depend on a training instance, where instead of deleting, we instead perturb the instance weight by a small amount. This is akin to a soft deletion. Mathemtically, this is the following:

$$\hat{\theta}_{\epsilon,z} = \arg\min_{\theta\in\Theta}(1-\epsilon)\frac{1}{n}\sum_{i=1}^{n}L(z_i,\theta) + \epsilon L(z,\theta)$$

What we want now is the gradient of the loss of a prediction on a test example with respect to this upweighting ϵ:

$$\begin{aligned} I_{up,loss}(z, z_{test}) &= \left.\frac{dL(z_{test},\hat{\theta}_{\epsilon,z})}{d\epsilon}\right|_{\epsilon=0} \\ &= \nabla_\theta L(z_{test},\hat{\theta})^T \left.\frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\right|_{\epsilon=0} \\ &= -\nabla_\theta L(z_{test},\hat{\theta})^T H_\theta^{-1}\nabla_\theta L(z,\hat{\theta}) \end{aligned}$$

The first step follows by the chain rule, while the second step is a result from Cook & Weisberg (1982), which follows from taking a quadratic approximation as shown below. The first term is the Hessian, while the second term is the gradient.

The Hessian can be approximated as

$$H_\theta = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\hat{\theta}}^2 L(z_i,\hat{\theta})$$

How to calculate $H_\theta^{-1}\nabla_\theta L(z,\hat{\theta})$? We can use

1. Conjugate gradient: Solve $\min_t t^T H_\theta t - v^T t$, or solve the system $H_\theta t = v$. Requires only matrix multiplies with H_θ.
2. Stochastic estimation: $H_j^{-1}v = v + (I - \nabla_\theta^2 L(z_{s_j},\theta)H_j^{-1}v$ until stabilization.

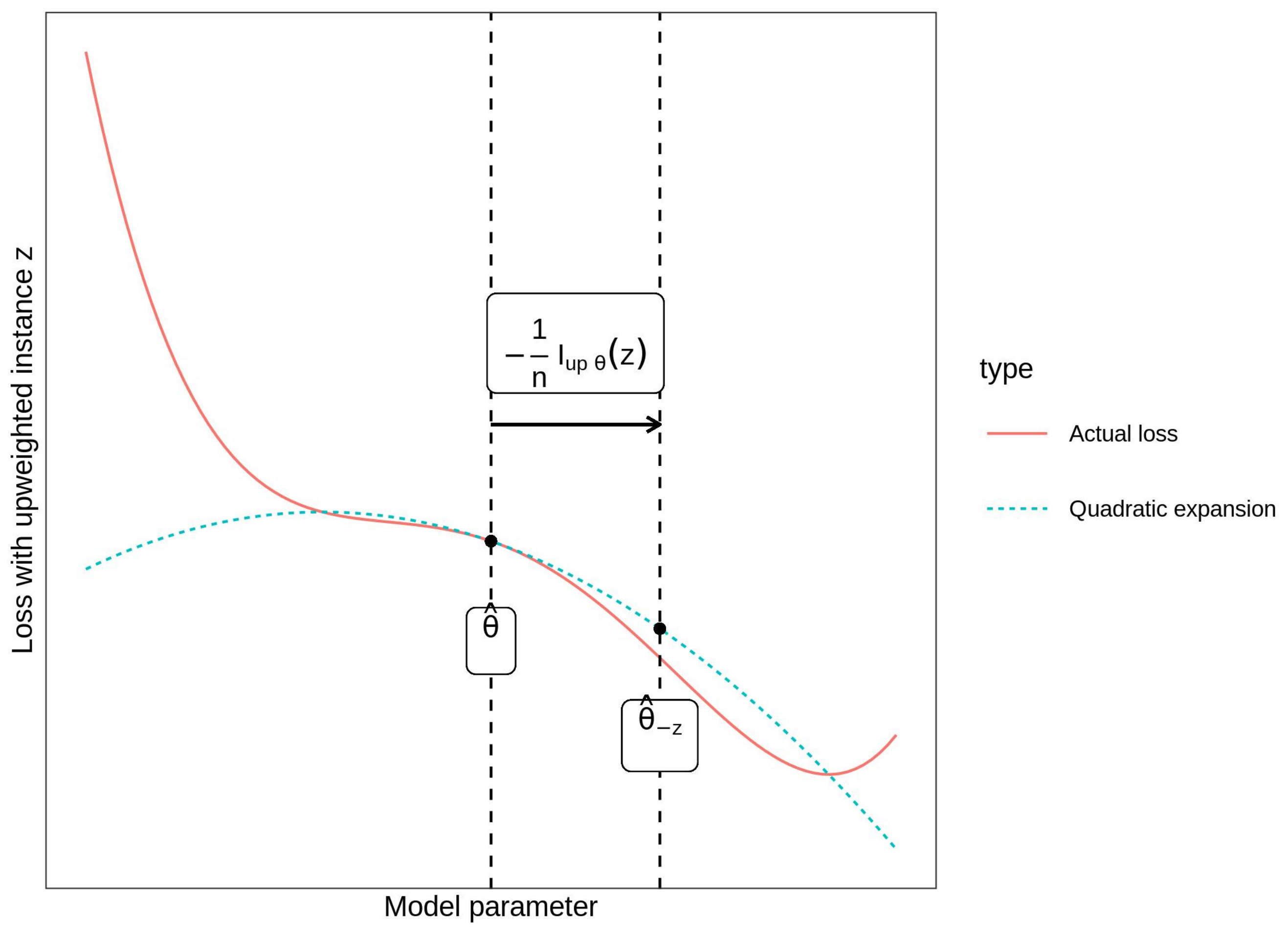

Figure 1: Quadratic expansion of the influence function. Figure from `https://christophm.github.io/interpretable-ml-book/influential.html`

1.2 Datamodels

If training time doesn't trouble you, then we can address the remaining issues with influences via data models. The idea here is that instead of deleting one example, we can instead delete subsets of examples and interpolate influences over many subsets.

1. Sample k subsets of the training data $S_k \subseteq D$ at some percentage size α.
2. Train k models f_{S_k} on a dataset S_k
3. Fit a linear model $g : \{0,1\}^{|D|} \to \mathbb{R}$ that predicts network accuracy from subsets using the dataset $\{1_{S_k}, Acc(f_{S_k})\}$.
4. Interpret linear weights as influences

By removing many datapoints at a time in the form of subsets, this amplifies the influence to a non-negligible amount. Furthermore, by averaging over many subsets we can reduce the variance of noise during training. However, you need to train many, many models. One way to reduce the number of subsets you need to train on is to remove datapoints in structured groups instead of individual points. For example, we can remove subsets of *classes* instead of random datapoints, effectively computing the influence of each class. The upside is that there are typically less classes than datapoints, but the downside is that you lose some fine-grained influence information.

1.3 Correlations and biases

The influence framework is focused on finding datapoints that have an effect on some output metric, such as performance. However, we aren't always interested in just performance—we would ideally like to make sure the performance of our model stems from real patterns as opposed to spurious correlations. How would you find spurious correlations (a form a bias) that the model has learned?

Let's start with an easier problem. Let's ignore the spurious factor, and instead just try to find correlations that the model learned. How would you do this? One way to find correlations is to look at examples that activate similar neurons. The intuition here is that examples with similar activations will capture similar patterns, and hence be correlated.

Let's simplify this further. Let's take one neuron at the last layer, and suppose that this neuron is used for two classes. Let's take a look at the examples from these classes that highly activate the neuron. Hopefully, these examples share some commonality that leads to the activation of that neuron. We can do this for an ImageNet classifier and find things like "baseball players" and "chainlink fences" to be correlated due to the chain pattern.

But there are too many neurons to look at, and every class uses every neuron. Instead, we can refit a sparse linear model to the final linear layer. This serves two purposes:

1. It reduces the number of neurons that need to be inspected
2. It reduces the number of classes that depend on each neuron
3. It amplifies the strongest signals used by the model

Spurious correlation How do we know if a correlation is spurious or not? Some correlations might be good (i.e. one might reasonably believe that suits should be correlated with grooms). We can set up the following MTurker experiment:

1. Show Mturker the exemplars from two classes
2. Is there a common pattern?
3. Is the pattern part of either class? If yes, then it is not spurious. If not, then it is spurious.

Biases. Sparsity in the last linear layer amplifies not only correlations, but also biases. Visualizations of the remaining "winning" features after sparsity represent the strongest signals of the model. Without sparsity, features with the largest weight can be redundant and all over the place.

2 References

Koh, Pang Wei, and Percy Liang. "Understanding black-box predictions via influence functions." International conference on machine learning. PMLR, 2017.

Ilyas, Andrew, et al. "Datamodels: Predicting predictions from training data." arXiv preprint arXiv:2202.00622 (2022).

Wong, Eric, Shibani Santurkar, and Aleksander Madry. "Leveraging sparse linear layers for debuggable deep networks." International Conference on Machine Learning. PMLR, 2021.