Notes on 'What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision' by Kendall et al., 2017.

or Statisticians hate her! She found a quick, easy way to approximate posteriors.

These notes are a bit of an overview of this week's paper, but my main focus is to explain some key concepts underlying uncertainty quantification in neural networks and to present the ideas outlined in the paper in a more general context. Part 1 is a basic, simplified theoretical background, part 2 is the overall results of a few, uh, yet-to-be successful experiments I ran this week.

I've made a few simplifications in the notation. Bias is not mentioned as a parameter, but in most cases it should be assumed that weights refer to both weights and bias. The paper is about CNNs, but this is about neural networks in general since the method is easily adaptable to different types neural networks with minor modifications. The paper also covers regression and classification tasks, but these notes only cover regression. Section 3.3 of the paper outlines the small differences between the two.

It's very possible that I've let some mistakes or misconceptions slip in - corrections and feedback welcome.

Part 1

Types of Uncertainty

Aleatoric

Aleatoric uncertainty is just a measurement of noise in the data. It can be caused by measurement error, mislabelling, or any other changes in the outcome y that can't be explained by input x. It is sometimes referred to as 'irreducible uncertainty' - no matter how much more data is thrown at it, the uncertainty stays the same. At least until data collection methods are improved.

In the paper's framework, aleatoric uncertainty is contained into the model likelihood. Typically the likelihood is $y_i \sim \mathcal{N}(f(x_i), \sigma^2)$, where the mean, $f(x_i)$, is the network's random output and σ^2 is a

fixed noise parameter. The mean is random because of stochastic regularization (more on this in the SR section).

With fixed variance, we assume that the variance stays the same across the input space. If necessary, aleatoric uncertainty can be modelled as varying according to different x values (a.k.a. heteroskedasticity). An example of this would be in modelling salary as a function of age. It makes sense to assume less variation in teens' salaries than for middle aged people.

The mean is already set as a random quantity dependent on the data, so the same can be done with the variance (yolo). So $y \sim \mathcal{N}(f(x_i), g(x_i))$, with the variance from the random network output as $g(x_i)$.

Epistemic

This type of uncertainty can be split into two more categories: uncertainty about the model structure and uncertainty about the parameters (eg. weights, bias). The paper focuses on the latter, but the part on model intrapolation and extrapolation addresses structural uncertainty.

A distribution over the weights quantifies how much the weights vary given the data at hand. The goal is to get a set of competing plausible weights that are more or less likely to have generated the output. Epistemic uncertainty can be completely explained with more data: uncertainty $\to 0$ as $N \to \infty$. Because of that, it is also called 'reducible uncertainty'. The further test data is from the training data, the larger the epistemic uncertainty.

The distribution of weights given the data is intractable for any moderately complex model, so it needs to be approximated. Some fancy Monte Carlo methods can be fast enough for this purpose (i.e. Langevin algorithms, though I'm sure there are others I'm not aware of). But with the regularization trick, variational inference should be the simplest, quickest way to approximate the posterior on the model parameters.

Predictive

Ultimately, the uncertainty of interest is the uncertainty about the predictions y_i^*. The paper shows how to measure both types of uncertainties about y_i^* at once in a neural network. Distributions about the noise and about the weights can also be modelled separately, but the focus here will be on the combination of the two.

Recall from Bayesian school that posterior $\propto$ likelihood·prior. The posterior on the weights, $p(\omega|x)$ is what is to be approximated with a simpler distribution $q(\omega)$, where ω are the random weights of the neural network. This is the part that quantifies (epistemic) model uncertainty.

The predictive distribution over y^* given new input x^* is

$$p(y^*|x^*, X, Y) = \int f(y^*|x^*, \omega)p(\omega|X, Y)d\omega$$

,

where $p(\omega|X, Y)$ is the posterior distribution on the weights and $f(y^*|x^*, \omega)$ is the likelihood with new data. Uncertainty about the data (aleatoric) is contained in the likelihood, while epistemic uncertainty is in the posterior over the weights. Nice and compartmentalized.

Stochastic Regularization as Variational Approximation

Using variational methods directly on the weight posterior of a deep network is not exactly practical, but we can take advantage of stochastic regularization (SR) in a way that is mathematically equivalent to variational approximations, under certain conditions. This means that only minor changes to an existing network are needed to effectively sample from the predictive posterior distribution in a reasonable amount of time. Training time should be about the same, but testing takes about T times longer, with T forward passes with dropout to run during testing. Each pass can be viewed as an MC sample from the predictive posterior distribution (hence the name MC dropout). This may just be the opportunity you've been waiting for to refine your parallelization skills. It's not clear what a reasonable number of passes needed for convergence is. In most applications I've seen, the number ranges between 10 and 100. It seems low to me, but it might be sufficient.

In non-Bayesian neural networks, SR is usually thought of as the injection of random noise into a layer input to prevent overfitting. Since we are interested in uncertainty about the parameters, noise about the feature space can be mapped onto the parameter space with a bit of algebra on the deterministic weight matrix, the layer input and a random binary vector.

This is the basic setup. The random weight matrices are just deterministic weight matrices times a diagonal matrix with random binary numbers on the diagonal (in the case of dropout). The loss function is now a random quantity as well. Time to optimize.

Relationship to Variational Methods

Variational Inference - Basic Concepts

Instead of trying to approximate the posterior directly, a simpler distribution is picked and the distance between the approximating distribution and the true posterior is minimized in terms of KL-divergence. When choosing an approximating distribution, there's a trade-off between simplicity and precision. You'll be surprised to learn that most people pick a normal distribution.

The KL-divergence (a.k.a. relative entropy) is the expected loss of information with the approximation relative to the true distribution. With log base 2, it is the expected number of information in bits we are expected to lose for a single datapoint:

$$KL(q(\omega)||p(\omega|x)) = \mathbf{E}_q[log(q(\omega)) - log(p(\omega|x))]$$

The idea is to minimize the distance between some approximating distribution $q(\omega)$ and a complex unknown distribution. Not very useful. But it turns out that minimizing the KL-divergence is equivalent to maximizing the evidence lower bound.

$$ELBO(q_\omega) = \mathbf{E}[log(p(x|\omega))] - KL(q(\omega)||p(\omega))$$

In words: we want to maximize the expected log-likelihood minus the distance between the approxima-

tion distribution and the prior on the weights. The ELBO is the objective function to optimize.

The loss function with the SR approximation is similar, but still different. The general form of the function when using dropout is:

$$\mathcal{L}_{DO} = -\mathbf{E}[log(p(x|\omega))] + \lambda_1||M_1||^2 + ... + \lambda_L||M_L||^2$$

Where M_i is a non-random weight matrix at layer i and λ_i is weight decay. In the paper, $\lambda = \frac{(1-p)\sigma}{2N}$, with p the probability of dropout and $\sigma = 1$.

We can make sure the optimization of both loss functions leads to the same results, up to a proportional constant. For that, we need to talk about everyone's favourite topic: priors.

Priors and the KL Condition

For VI and SR to be equivalent, the prior over the weights $p(\omega)$ must be defined such that

$$\frac{\partial}{\partial M} KL(q_M(\omega)||p(\omega)) = \frac{\partial}{\partial M}(\lambda_1||M_1||^2 + ... + \lambda_L||M_L||^2)N\sigma^{-1}$$

If this is true, the derivatives of the loss functions used for optimization are:

$$\frac{\partial}{\partial M}\hat{\mathcal{L}}_{SR}(M) = \frac{\sigma}{N}\frac{\partial}{\partial M}\hat{\mathcal{L}}_{VI}(M)$$

$$\frac{\partial}{\partial M}\hat{\mathcal{L}}_{SR}(M) \propto \frac{\partial}{\partial M}\hat{\mathcal{L}}_{VI}(M)$$

assuming constant variance.

The choice of priors over ω is clearly limited. In the paper, the approximation distribution $q(\omega)$ is a mixture of two normal distributions with small variance, one with mean 0 and one with mean $f(x)$. A prior on the weights that fulfills the KL-condition is a simple matrix normal distribution with mean 0 and identity covariance. The covariance matrix can be scaled by a fixed value if you really need to feel

like you've made a decision (really, it does have some incidence on weight magnitude). In the paper, that parameter was fixed at 1.

As a sidenote, it is well documented that the optimized approximation function will underestimate variance, unless the true distribution is in the same parametric family. It follows that this is an issue with SR approximation as well. In both cases, predictive variance will be underestimated, with the objective function putting a strong penalty on approximating distribution $q(\omega)$ for placing mass where the true posterior has no mass, but a small penalty for not placing mass where it should.

1 Part 2

Overall Results

The paper is short and brushing over a lot of details, so I tried to fill in the gaps with Yarin Gal's PhD thesis (Gal, 2016). I contacted Alex Kendall to get some of their code, but it can't be shared yet since this is a preprint. The problem with using the thesis is that since there's *a lot* more information, it's not clear what to include and what to leave out. The thesis addresses some issues that might help fix my code. A recurrent fix is to optimize more parameters by grid search: the probability of dropout and weight decay simultaneously, and the scaling parameter on the weight prior. According to the author's tests, the setting of these three parameters have a lot of influence on uncertainty quantification and distribution approximation. The paper fixed all these parameters, but since their optimum is model dependent, it could help to try some additional optimization scheme. An apparently simpler, faster solution for optimizing dropout probability is presented in the paper Concrete Dropout (Gal et al., 2017). I'll see how that goes, I'm still fiddling around it.

In the end, I got some results that I can't explain, a couple loss= $-\infty$, numerical issues, and lack of memory on the GPU I'm using.

I ran the tests on various CNN architectures, with dropout $p = 0.2$ after each convolutional layer and I used one dataset for regression and one dataset for classification:

1. By-pixel depth regression with the Make3D dataset. I had to shrink the images by a lot: from 2272×1704 to 168×168 pixels. I used a 240 images training set and a 60 images test set, with minibatches of size 20 for each epoch. The output I'm interested in is depth uncertainty heatmaps. Obviously an image of a blank wall has much more depth uncertainty than an image of a crowd, and I want to capture this information along with my predictions. I ran a 6 layer vanilla CNN, and a simplified version of the CNN from Eigen and Fergus (2015).

   The vanilla CNN uncertainty was almost entirely aleatoric. This could be because the images were too small, but I think it should have picked up more epistemic uncertainty. The loss also converged too fast and I couldn't get it to behave.

   I tried the Eigen model since it is specifically designed for depth regression, but it would not have finished running on time.

   I figured CIFAR-10 would be simpler than by-pixel regression.

2. CIFAR-10: assuming there's no need for a description.

   I started out with a simple 6 layer CNN that I've used before on CIFAR-10. I ran into numerical issues and six digit loss.

   Since the paper uses Densenet in their experiments, I also ran the smallest Densenet tested in Huang et al. (2016), with 40 layers, 3 dense layers, and 12 added feature maps per layer. As I expected from previous experiments with Densenet, it would be too long to run.

I'll be bringing my opinions with me to the Meetup. I'm curious to see what other people think about the test results in the paper. How to reconcile the results on uncertainty quality (figure 5) and extrapolation (table 5). The method to measure heteroskedasticity also seems pretty ad-hoc to me.

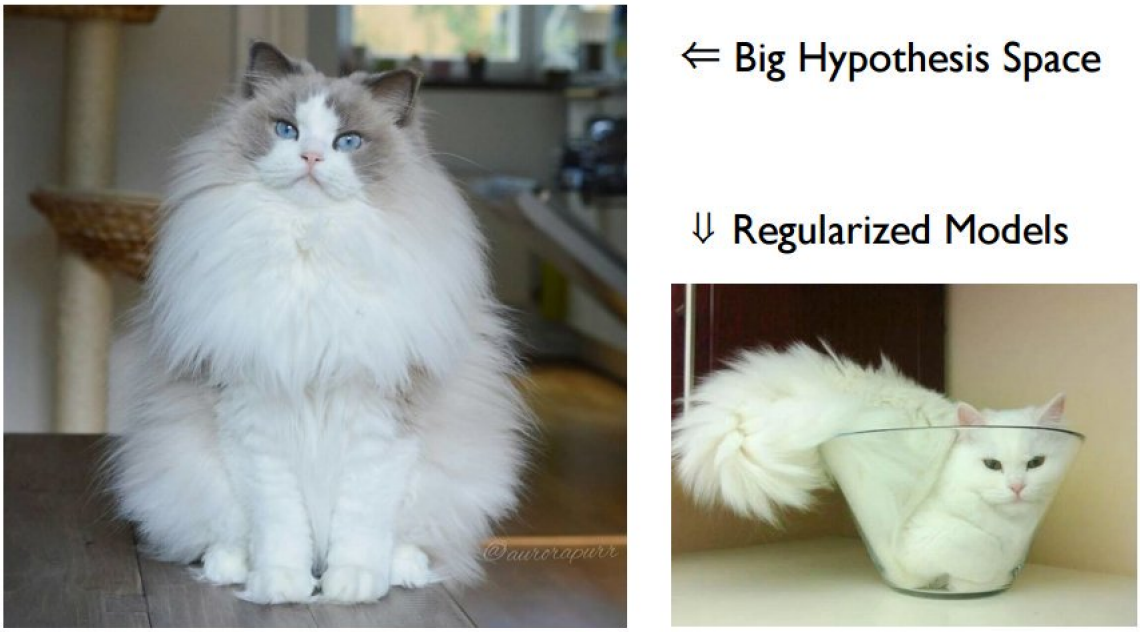

Figure 1: Turns out that regularization is more than just cats and bowls.

References

Eigen, D. and Fergus, R. (2015), "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2650–2658.

Gal, Y. (2016), "Uncertainty in Deep Learning," Ph.D. thesis, University of Cambridge.

Gal, Y., Hron, J., and Kendall, A. (2017), "Concrete Dropout," in *arXiv:1705.07832*.

Huang, G., Liu, Z., and Weinberger, K. Q. (2016), "Densely Connected Convolutional Networks," *CoRR*, abs/1608.06993.