Modeling Uncertainty in Bayesian Neural Networks with Dropout:

The effect of weight prior and network architecture selection

Ellis Brown[1,2], Melanie Manko[1,2], Ethan Matlin[1]

[1] Columbia University
[2] BlackRock AI Labs

ellis.brown@columbia.edu
@ellisbrownii
ellisbrown.github.io
github.com/ellisbrown

COLUMBIA UNIVERSITY

BlackRock AI Labs

THE OSAGE NATION

Motivation:

- Neural Networks (NNs) are really good at predicting stuff. But what about when they're wrong?
- It is useful to know the uncertainty of a NN's prediction, rather than just a point estimate
  - E.g., a self-driving car could alert its driver that it is uncertain about conditions rather than confidently making a decision that could be dangerous
  - The Bayesian paradigm is natural framework for modelling uncertainty, and has been applied to NNs since the 1990s
  - "Posterior" distribution of *weights* given the data

$$P(\omega \mid X, Y) = \frac{P(Y \mid X, \omega) P(\omega)}{P(Y \mid X)}$$

- Bayesian NNs trained with Dropout are a *practical* way to model this uncertainty, and have theoretical foundations relating them to Variational Inference approximating Gaussian Process posteriors

Theoretical Background

Gaussian Process (GP)

- An infinite-dimensional stochastic process such that every finite collection of random variables generated by the process is normally distributed
- Distribution over all functions consistent with the data

Infinite NN ⇔ GP

- 1 hidden layer, infinite neurons
- Bounded nonlinearities (activation functions)
- Weights are i.i.d. with zero mean and finite variance

How Can We Approximate This Posterior?

- Monte Carlo techniques
- Variational Inference
- … neither scale with big networks or data

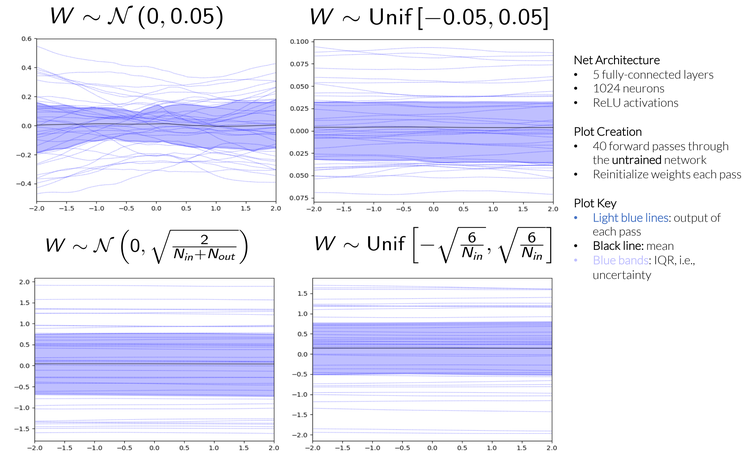
Dropout ⇔ Variational Inference (VI) in GP

- Dropout is a stochastic regularization technique that is widely adopted
  - Randomly switch on/off different neurons during training to prevent overfitting
- Take any network trained with dropout, and you can compute uncertainty information with little effort
  - Use dropout during prediction, drawsing a sample of point estimates with various dropout initializations

Neural Networks trained with **Dropout** can be modified to give **uncertainty estimates** which depend on the choice of **prior distribution over weights** and **network architecture.**

Snap a picture to learn more!

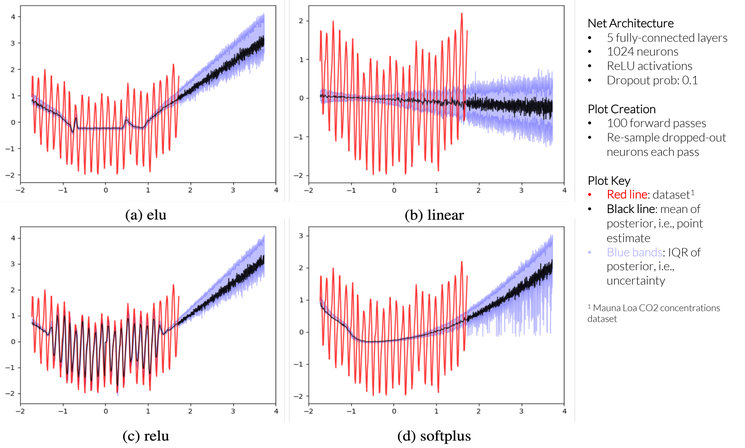
Certain priors introduce bias into the network. Scaling variance by layer reduces this bias.



Uncertainty estimates on [-2, 2] given by untrained networks with various weight priors.

Each plot shows the bias in our uncertainty measure that various choices of prior distributions for weight initialization introduce into the network before training on data. Since, we don't know where in the data-space we want our model to be more and less uncertain before seeing the data, we want the variance of our distribution to be relatively constant across values of **x**. The plots on the bottom row—which use the Xavier Normalization—satisfy this criterion the best. The basic idea of this procedure is to normalize variance per layer based on the number of inputs and outputs.

Activation Functions & Uncertainty Estimates



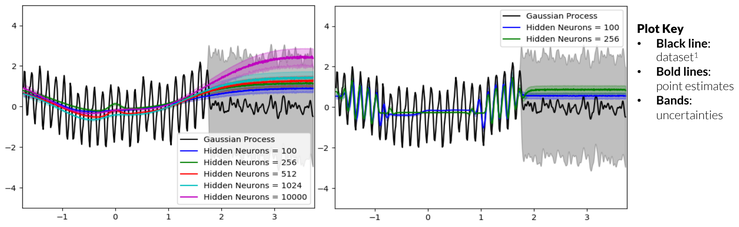
[1] Mauna Loa CO2 concentrations dataset

Uncertainty behavior on data far from what has been observed

Each plot shows the uncertainty estimate produced by various choices of activation function with identical network structure and training data. The linear (b), ReLU (c), and eLU (a) activation functions yield uncertainty bounds that fan out linearly. For these nonlinearities, this makes intuitive sense as linear is, of course, linear and ReLU and eLU contain linear components. Softplus yields uncertainty bounds that seem to fan out nonlinearly, with negative skewness . In all cases, the uncertainty bands fan out as they move farther from the training data which is exactly what we'd want and expect.

Note: In some experiments, the network was not trained long enough to converge. As our primary concern is the behavior of the uncertainty estimates as the model observes data far from the data it has already seen, we leave this to further investigation with more computational resources

Depth, Width, and Convergence to GP



(a) **NN** (Gaussian Nonlinearity, 1 Layer) **GP** (Squared Exponential Covariance)

(b) **NN** (Gaussian Nonlinearity, 5 Layers) **GP** (Squared Exponential Covariance)

Does the infinite width single layer network convergence to GP result hold for a finite network?

With any number of neurons wide that is feasible to train on a standard laptop, the single layer network doesn't approach a GP with squared exponential covariance. Depth on the other hand seems to help a lot. This result makes sense since there have been a number of papers recently covering the expressive power of depth (Telgarsky, 2015 & 2016; Yarotsky, 2017; Liang and Srikant, 2016; Eldan and Shamir, 2015).