Bayesian Neural Networks

Motivation

Neural Nets are powerful

But we also want to know uncertainty in predictions

Perspective on the learnt parameter distributions

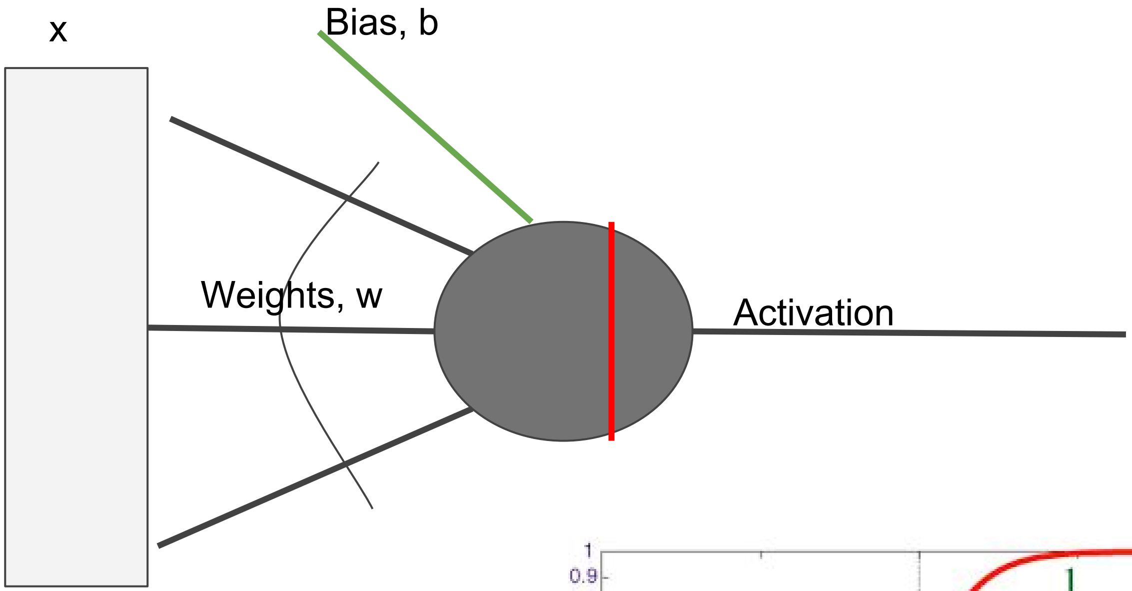
Use Neural Networks with Probabilistic modeling

Neural Nets as universal function approximators

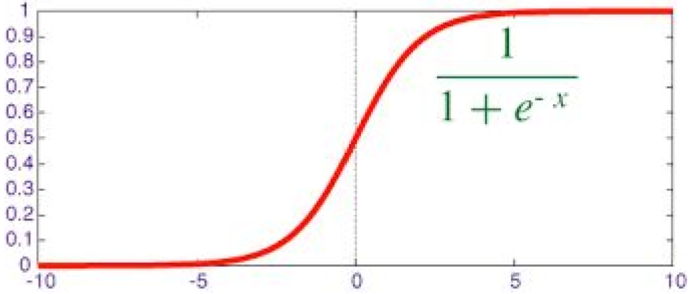
Defined modeling for known interactions, with principled results

Neural Nets at lightspeed

Artificial Neuron

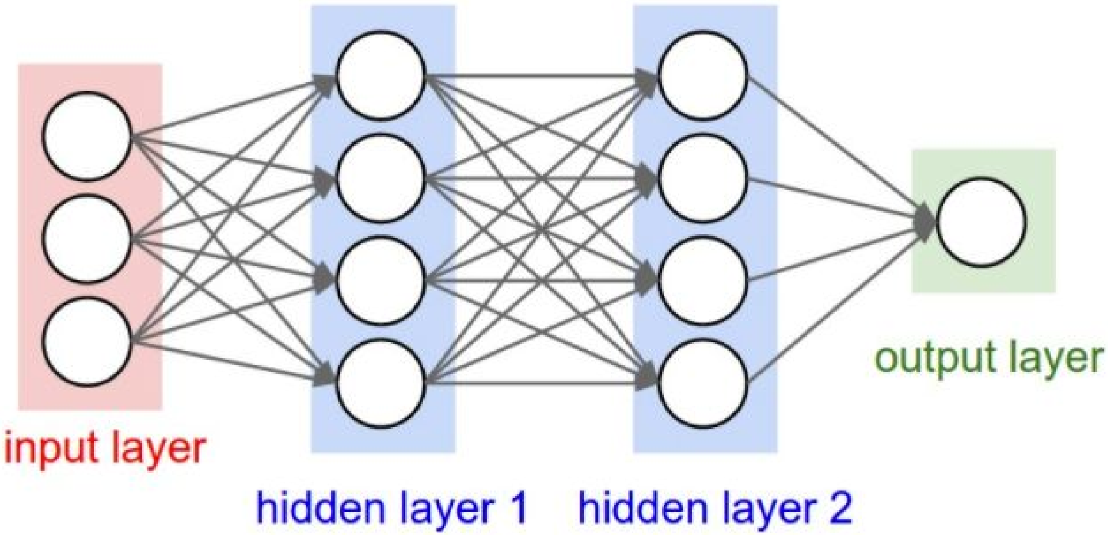


Linear, $z = wx + b$
Activation, $a = \sigma(z)$



Backpropagation:
$\partial\mathcal{L}/\partial w = (\partial\mathcal{L}/\partial z)\,(\partial z/\partial w)$

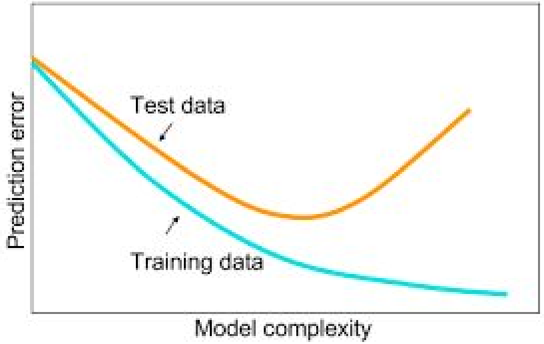
Neural Network at Light Speed (MLP)

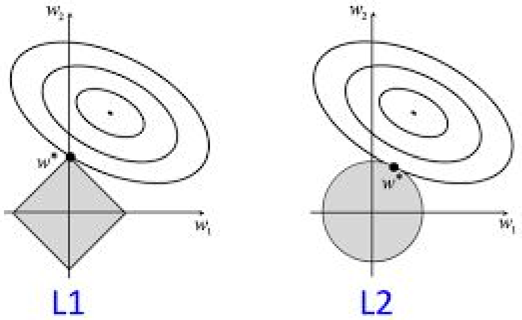


From CS231N Stanford

Neural Networks - Characteristics

$$\arg\max_{\theta} Pr(D;\theta)$$





Probabilistic Modeling - Bayes Rule

- Use of Product rule, Sum rule and Conditional probability gives us:

Posterior Probability

Likelihood of Observations

Prior Probability

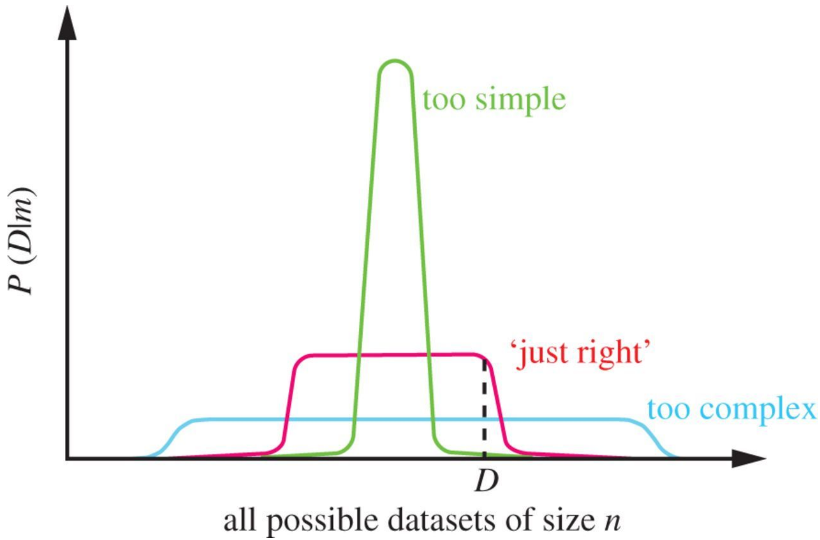
$$\Pr(\theta|y) = \frac{\Pr(y|\theta)\Pr(\theta)}{\Pr(y)}$$

Normalizing Constant

- **θ** is model parameter
- y is the data

Probabilistic Modeling

- Directly define parameterized statistical models
- Specify prior distributions on model parameters
- Specify likelihood of observed data given the parameters
- Learning the posterior parameters (inference) involve sampling or approximation techniques
    - Sampling using MCMC techniques: Metropolis, HMC, NUTS
    - Variational inference: Stochastic VI, Mean-Field, ADVI
- Posterior predictive checks for prediction



What makes inference possible?

Approximation

- Sampling
  - MCMC family of samplers

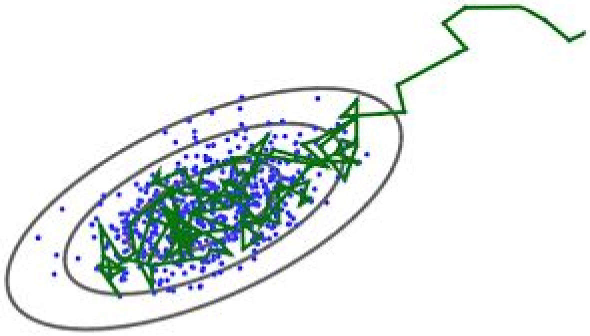
$$\alpha = \min\left(1, \frac{\pi(\theta_c)q(\theta_0|\theta_c)}{\pi(\theta_0)q(\theta_c|\theta_0)}\right)$$

- Variational Inference
  - Stochastic
  - Mean-Field
  - Automatic Differentiation

$$KL(q||p) = \mathbb{E}_q[\log \frac{q(\theta;\nu)}{p(\theta|x)}] :$$

Sampling

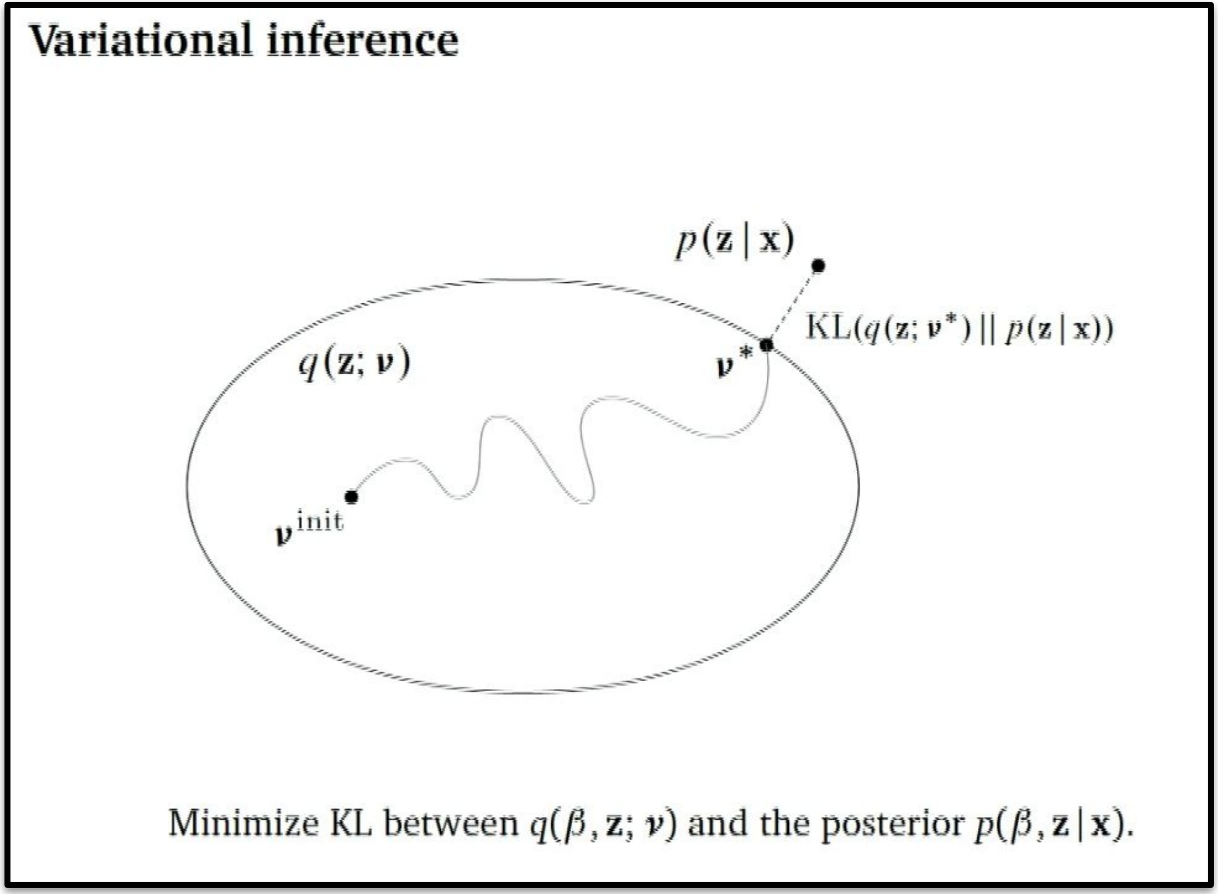
$$\alpha = \min\left(1, \frac{\pi(\theta_c)q(\theta_0|\theta_c)}{\pi(\theta_0)q(\theta_c|\theta_0)}\right)$$



- Detailed balance
- Ergodicity
- Irreducible

- Mixing time
- Burn in period
- Asymptotic convergence

- Gibbs Sampling
- Metropolis Hastings
- Hamiltonian Monte Carlo
- No-U-Turns Sampler



Source: D. Blei, Columbia University

Minimizing KL divergence (evidence lower bound maximization)

$$KL(q||p) = \mathbb{E}_q[\log \frac{q(\theta;\nu)}{p(\theta|x)}] :$$

$$= \mathbb{E}_q[\log \frac{q(\theta)p(x)}{p(\theta,x)}]$$

Rearranging terms, that simplifies to:

$$\overbrace{KL(q||p)}^{\text{KL divergence}} = -\overbrace{(\underbrace{\mathbb{E}_q[\log p(\theta,x)]}_{\text{exp. log joint}} - \underbrace{\mathbb{E}_q[\log q]}_{\text{entropy}})}^{\text{ELBO, } \mathscr{L}} + \overbrace{\log p(x)}^{\text{Constant}}$$

ADVI: Objective and Gradient - 1

ELBO: $\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log p(\boldsymbol{x}, \boldsymbol{\theta})\right] - \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log q(\boldsymbol{\theta}\,;\,\boldsymbol{\phi})\right].$

Transforming to a real coordinate space: $\zeta = T(\boldsymbol{\theta}).$

Transformed joint density: $p(\boldsymbol{x}, \zeta) = p\left(\boldsymbol{x}, T^{-1}(\zeta)\right)\left|\det J_{T^{-1}}(\zeta)\right|,$

Use a Gaussian distribution for variational approximation:

$$q(\zeta\,;\,\boldsymbol{\phi}) = \mathcal{N}\left(\zeta\,;\,\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$$

Non-unique Cholesky decomposition, $q(\zeta\,;\,\boldsymbol{\phi}) = \mathcal{N}\left(\zeta\,;\,\boldsymbol{\mu}, \boldsymbol{L}\boldsymbol{L}^\top\right)$

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{q(\zeta\,;\,\boldsymbol{\phi})}\left[\log p\left(\boldsymbol{x}, T^{-1}(\zeta)\right) + \log\left|\det J_{T^{-1}}(\zeta)\right|\right] + \mathbb{H}\left[q(\zeta\,;\,\boldsymbol{\phi})\right].$$

ADVI: Objective and Gradient - 2

Elliptical Standardization:
(Gaussian to Standard Gaussian)

$$\eta = S_{\phi}(\zeta) = L^{-1}(\zeta - \mu).$$

Objective becomes:

$$\phi^* = \arg\max_{\phi} \mathbb{E}_{\mathcal{N}(\eta\,;\,0,I)} \left[\log p\left(x, T^{-1}(S_{\phi}^{-1}(\eta))\right) + \log \left| \det J_{T^{-1}}\left(S_{\phi}^{-1}(\eta)\right) \right| \right] + \mathbb{H}[q(\zeta\,;\,\phi)].$$

Compute the gradients:

$$\boxed{\nabla_{\mu}\mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)}\left[\nabla_{\theta} \log p(x, \theta) \nabla_{\zeta} T^{-1}(\zeta) + \nabla_{\zeta} \log \left| \det J_{T^{-1}}(\zeta) \right| \right].}$$

$$\boxed{\nabla_{L}\mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)}\left[\left(\nabla_{\theta} \log p(x, \theta) \nabla_{\zeta} T^{-1}(\zeta) + \nabla_{\zeta} \log \left| \det J_{T^{-1}}(\zeta) \right| \right) \eta^{\top}\right] + (L^{-1})^{\top}.}$$

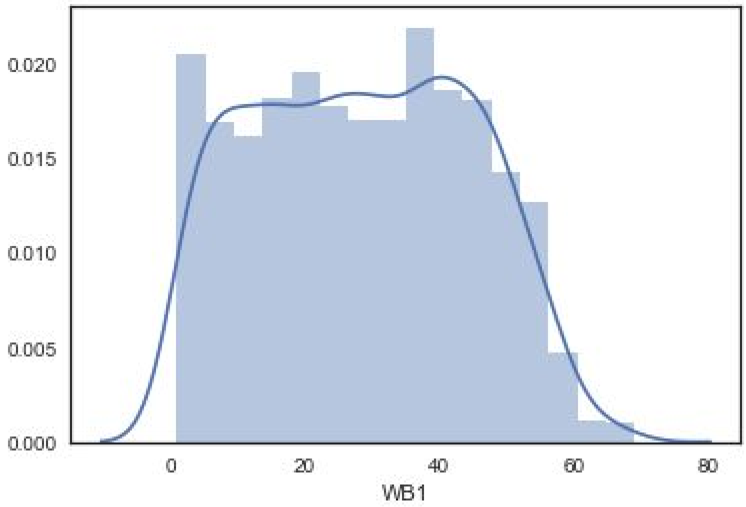
Experiments

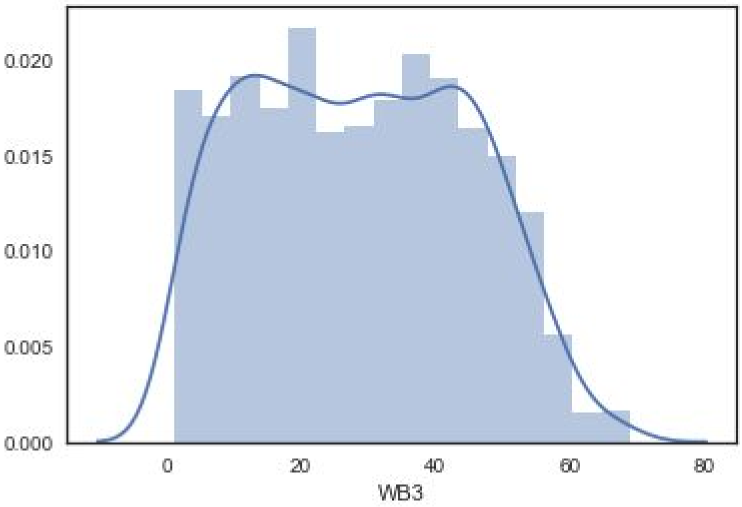
Dataset - powerball

Powerball / December 13, 2017

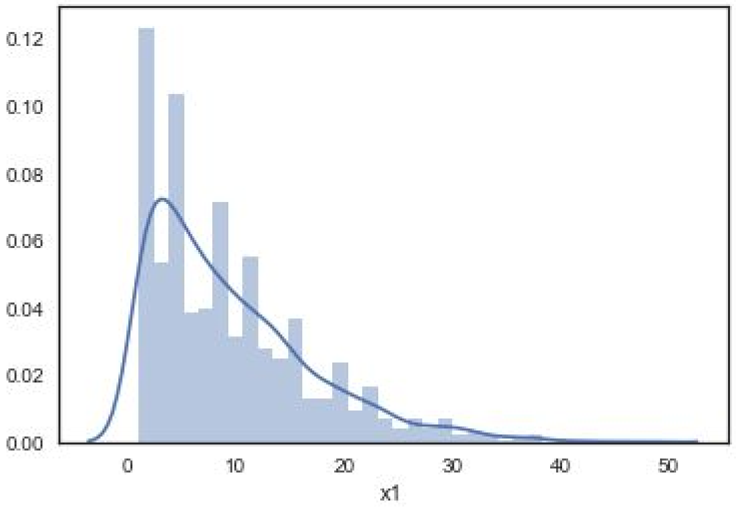
02 24 28 51 58 07

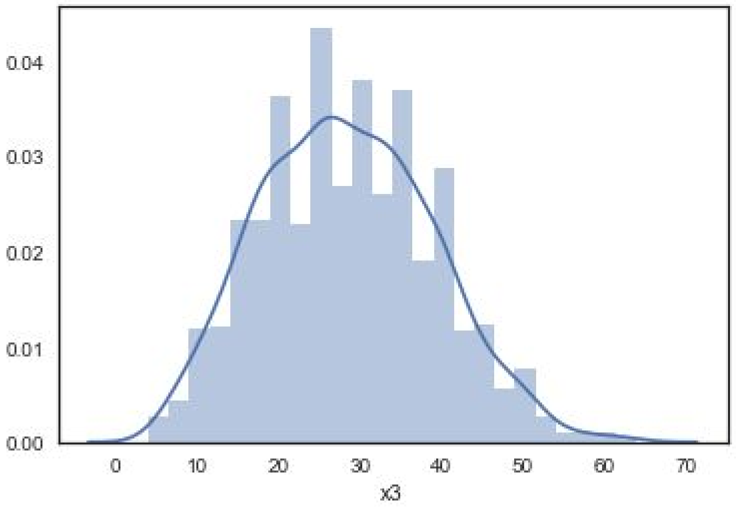
Powerball dataset



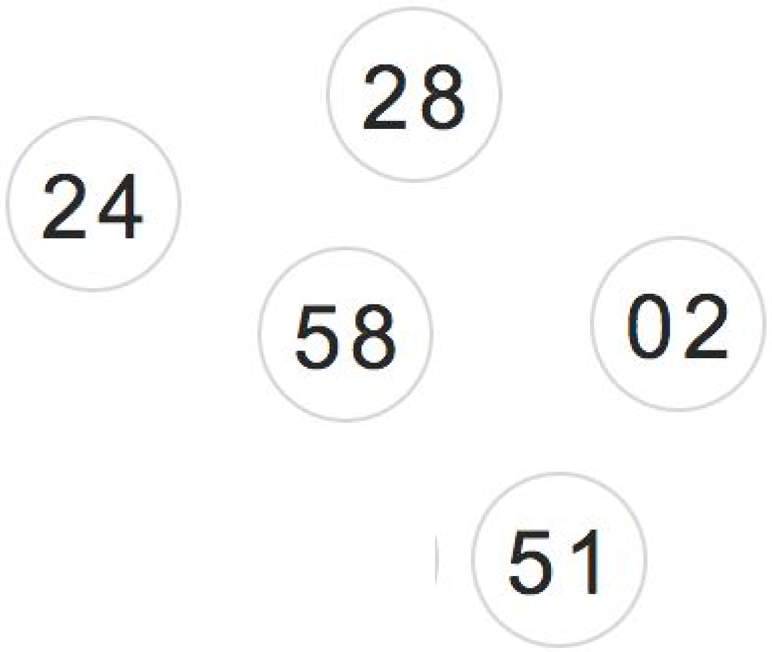


Pre-processed (creating bins)





Predict the bin and uncertainty



Model Building

```
ntrain = len(X_train)

#Construct the NN
def construct_nn(nn_in, nn_out):
    n_hidden = 5
    init_1 = np.random.randn(1, n_hidden).astype(theano.config.floatX)
    init_2 = np.random.randn(n_hidden, n_hidden).astype(theano.config.floatX)
    init_3 = np.random.randn(n_hidden, 1).astype(theano.config.floatX)

    with pm.Model() as bnn:
        weights_1 = pm.Normal('w_1', mu=0, sd=1, shape=(1, n_hidden), testval=init_1)
        weights_2 = pm.Normal('w_2', mu=0, sd=1, shape=(n_hidden, n_hidden), testval=init_2)
        weights_3 = pm.Normal('w_3', mu=0, sd=1, shape=(n_hidden, 1), testval=init_3)

        #Activations
        act_1 = pm.math.tanh(pm.math.dot(np.log(nn_in), weights_1))
        act_2 = pm.math.tanh(pm.math.dot(act_1, weights_2))
        act_3 = pm.math.sigmoid(pm.math.dot(act_2, weights_3))
        out = pm.Bernoulli('Category', p=act_3, observed=nn_out, total_size=ntrain)

    return bnn
```

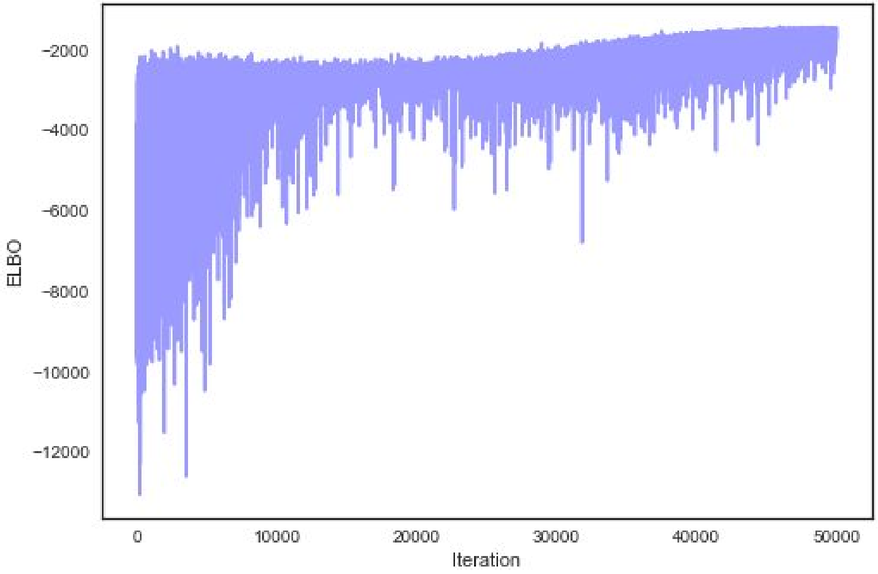
NN as a universal function approximator

Bernoulli Likelihood

Inference

ELBO maximization

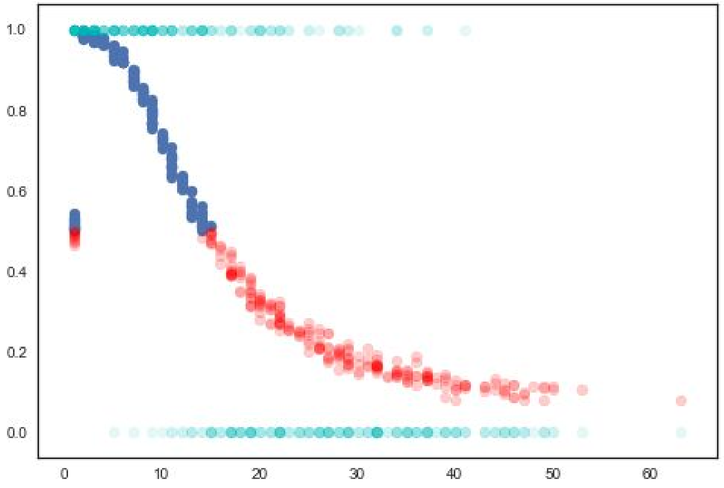
```python
with neural_network:
    inference = pm.ADVI()
    approx = pm.fit(n=30000, method=inference)
```

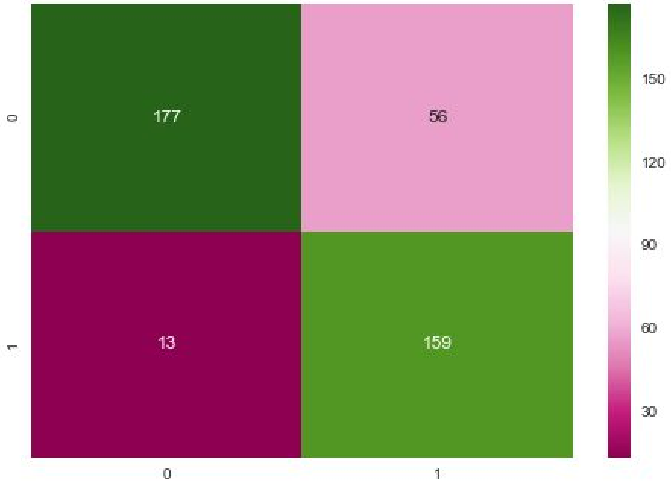


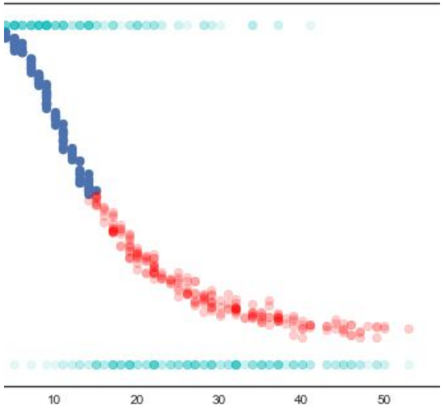
Predictions

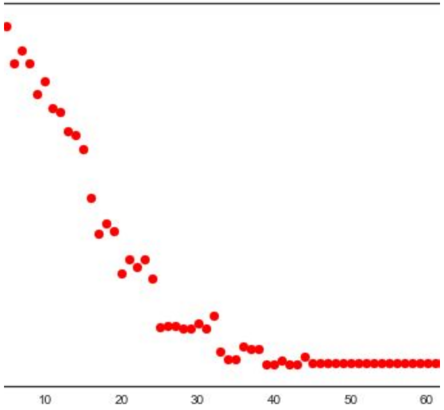
```python
nn_in.set_value(X_test)
nn_out.set_value(y_test)
with bnn:
    ppc = pm.sample_ppc(trace, 500)
    pred = ppc['Category'].mean(axis=0)
    proba = ppc['Category'].std(axis=0)
```

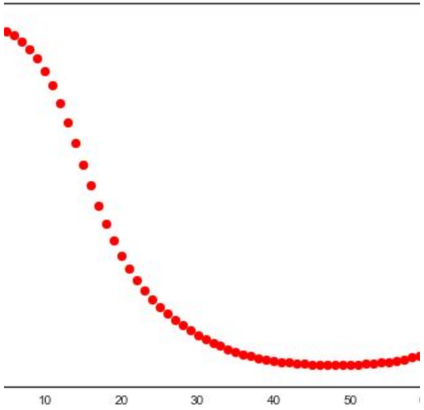
Predictions are sampled from posterior

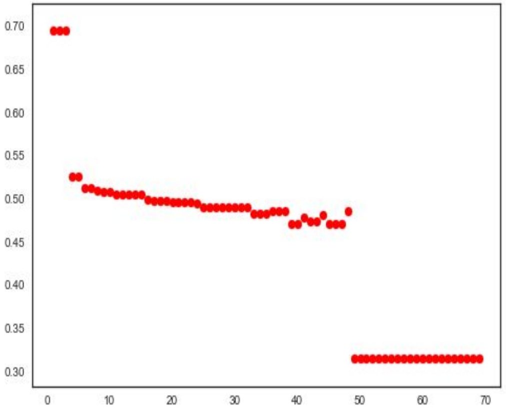












Comparison to other models

Random Forest Classifier

Gaussian Process Classifier

AdaBoost Decision Trees

Accuracy: 82.96

85.18

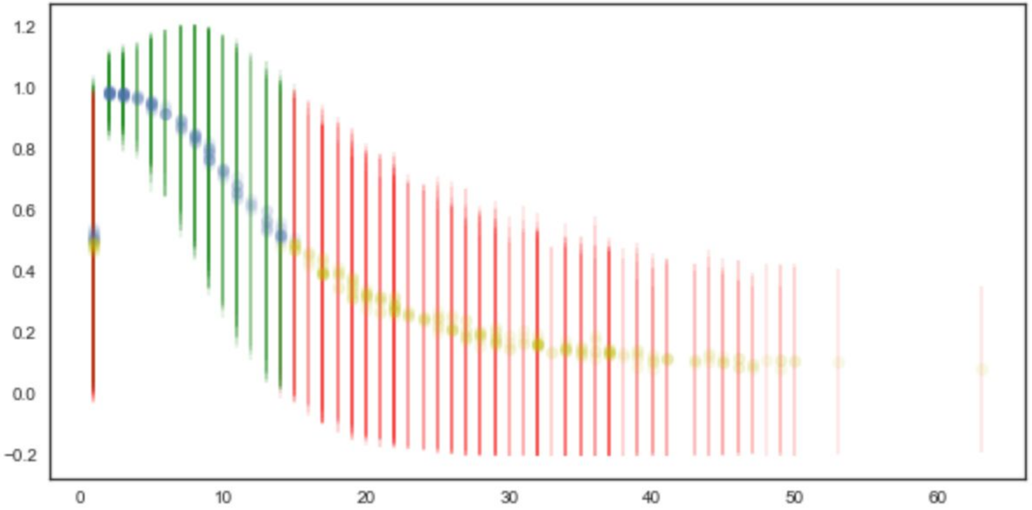
84.93

85.18

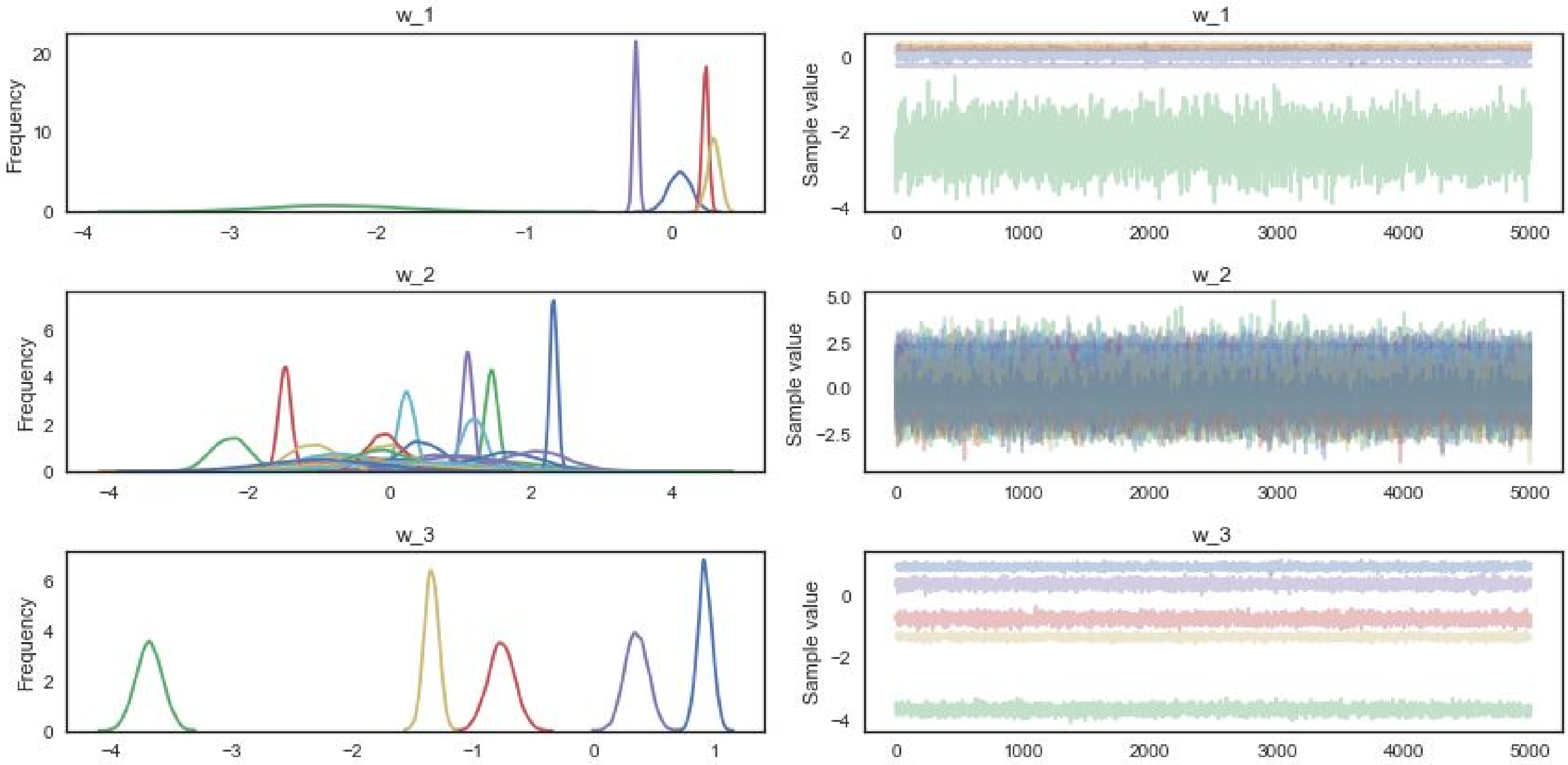
More Importantly, the uncertainty

```
fig, ax = plt.subplots(figsize=(10, 5))
ax.errorbar(X_test[pred > 0.5], pred[pred > 0.5], yerr = proba[pred > 0.5],
            fmt='o', ecolor='g', capthick=2, alpha=0.1)
ax.errorbar(X_test[pred <= 0.5], pred[pred <= 0.5], yerr = proba[pred <= 0.5],
            fmt='o', ecolor='r', color='y', capthick=2, alpha=0.1)
```

```
<Container object of 3 artists>
```



Posterior parameter distribution

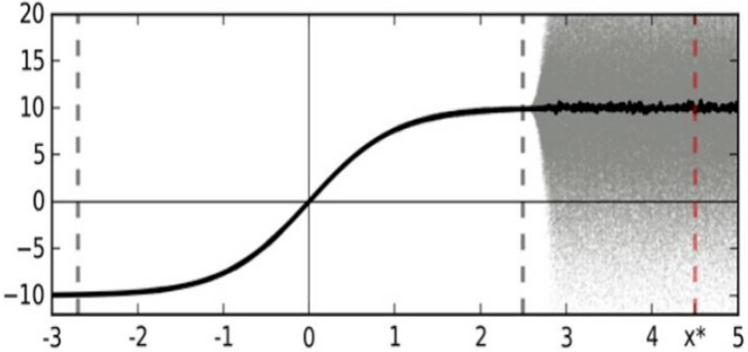


Active Areas of Work

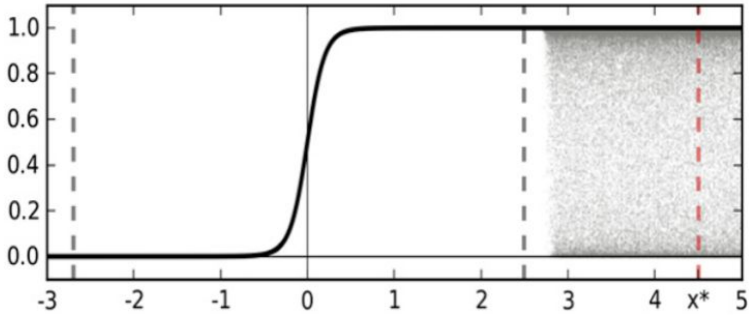
- Alternate Divergences Variational Inference
- Non-convexity of ELBO
- Better optimization
- Better approximations
- Variational Inference is less explored


Q
&
A

Input to Sigmoid



Sigmoid Prediction



Y. Gal:
http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html#uncertainty-sense

Sigmoid Layer

Retrieving a measure of unpredictability using complete distributional as opposed to MAP estimate

What makes inference possible?

$$\alpha = \min\left(1, \frac{\pi(\theta_c)q(\theta_1|\theta_c)}{\pi(\theta_0)q(\theta_c|\theta_0)}\right)$$

$$\overbrace{KL(q||p)}^{\text{KL divergence}} = -(\overbrace{\underbrace{\mathbb{E}_q[\log p(\theta, x)]}_{\text{cross-entropy}} - \underbrace{\mathbb{E}_q[\log q]}_{\text{entropy}}}^{\text{ELBO}, \mathscr{L}}) + \overbrace{\log p(x)}^{\text{Constant}}$$

$$\mathbb{E}(\mathbf{y}^*) \approx \frac{1}{T}\sum_{t=1}^{T} \widehat{\mathbf{y}}_t^*(\mathbf{x}^*)$$

$$\mathrm{Var}\big(\mathbf{y}^*\big) \approx \tau^{-1}\mathbf{I}_D + \frac{1}{T}\sum_{t=1}^{T} \widehat{\mathbf{y}}_t^*(\mathbf{x}^*)^T \widehat{\mathbf{y}}_t^*(\mathbf{x}^*) - \mathbb{E}(\mathbf{y}^*)^T \mathbb{E}(\mathbf{y}^*)$$

```
probs = []
for _ in xrange(T):
    probs += [model.output_probs(input_x)]
predictive_mean = numpy.mean(prob, axis=0)
predictive_variance = numpy.var(prob, axis=0)
tau = l**2 * (1 - model.p) / (2 * N * model.weight_decay)
predictive_variance += tau**-1
```

Python code to obtain predictive mean and uncertainty from dropout networks

Source:
http://www.cs.ox.ac.uk/people/yarin.gal/website/publications.html