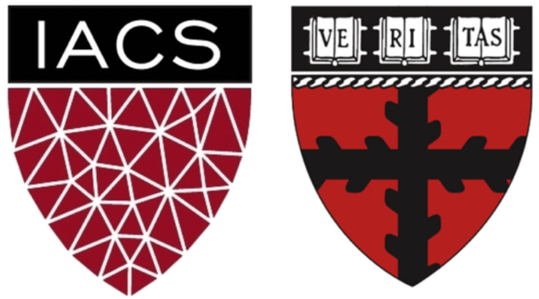
Lecture #17: Black-box Variational Inference

AM 207: Advanced Scientific Computing

Stochastic Methods for Data Analysis, Inference and Optimization

Fall, 2020


IACS
VE RI TAS

Outline

1. Review of Neural Network Models
2. Bayesian Neural Networks
3. Black-box Variational Inference

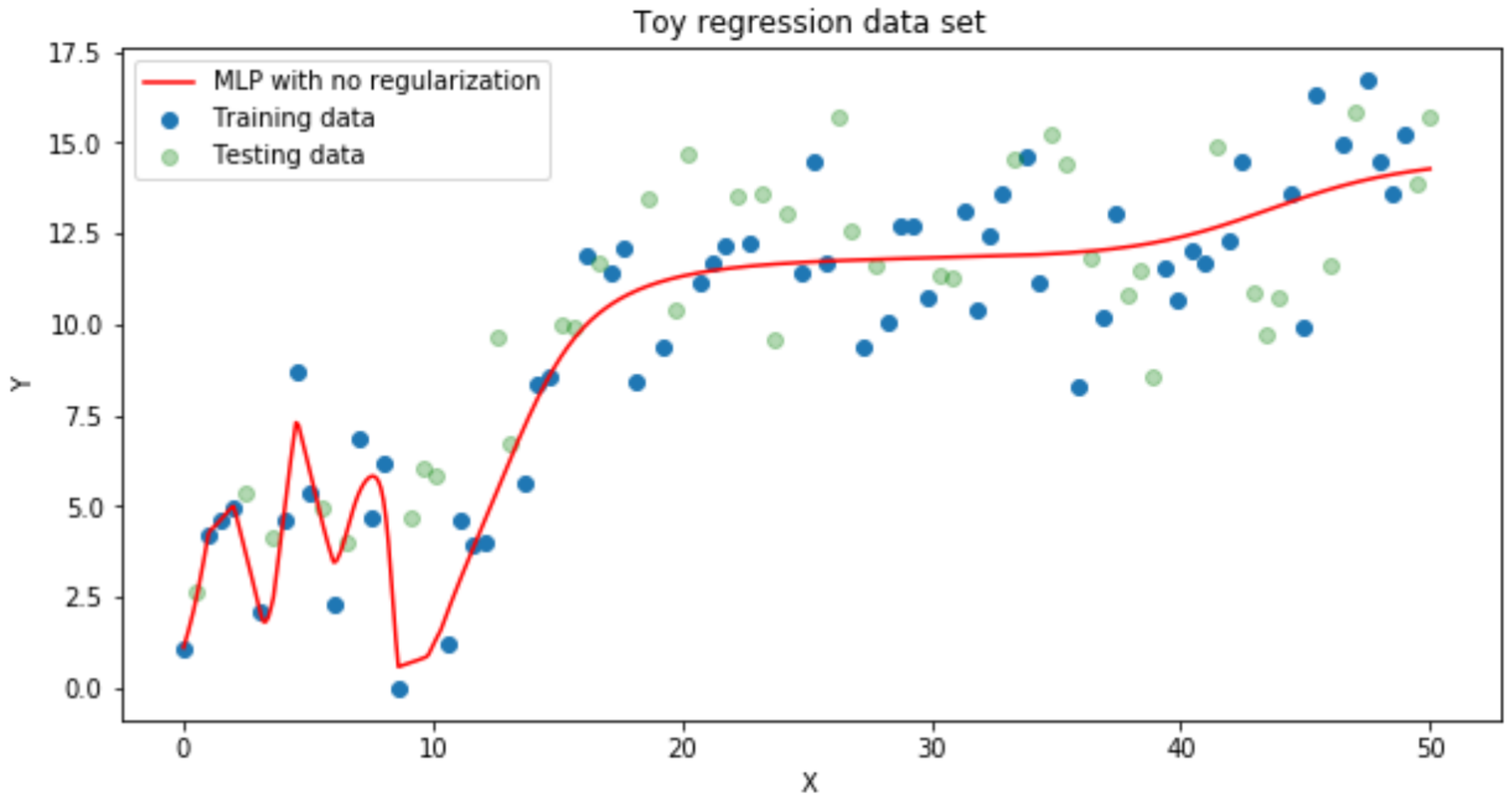
Review of Neural Network Models

How Would You Parameterize a Non-linear Trend?

In non-linear regression, we are interested in modeling observed outcome $Y^{(n)}$ as a non-linear function of observed covariates $\mathbf{X}^{(n)}$:

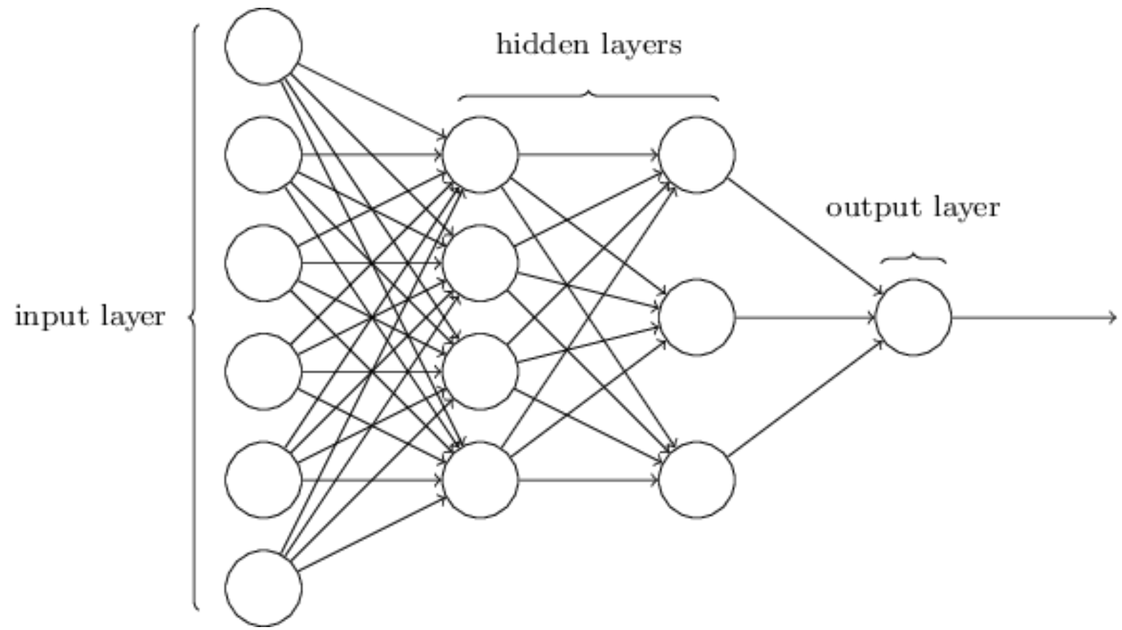
$$\mu = g_{\mathbf{w}}(\mathbf{X}^{(n)})$$
$$Y^{(n)} \sim \mathcal{N}(\mu, \sigma^2)$$

But it's not easy to think of a function class $g(x)$ can capture the trend in the data (e.g. polynomial or sinusoical)?



Representing Arbitrarily Complex Functions

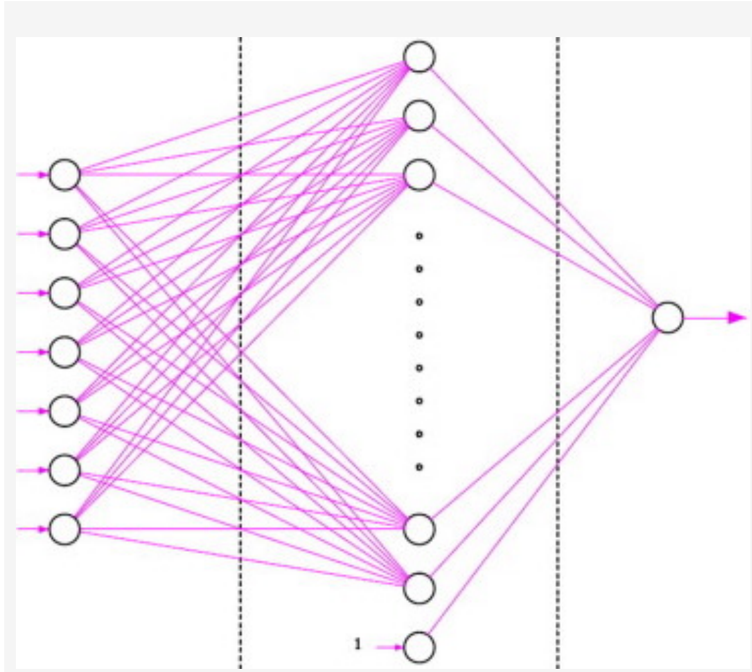
Motto: neural netoworks build up a complex function g by **composing** simple nonlinear functions. We represent neural networks as **layered directed graphs** where each node i in the l-th layer represents the function $f\left(\sum_j w_{ji}^{l-1} h_j^{l-1}\right), h_j^{l-1}$ being the hidden nodes from the $l-1$-th layer.

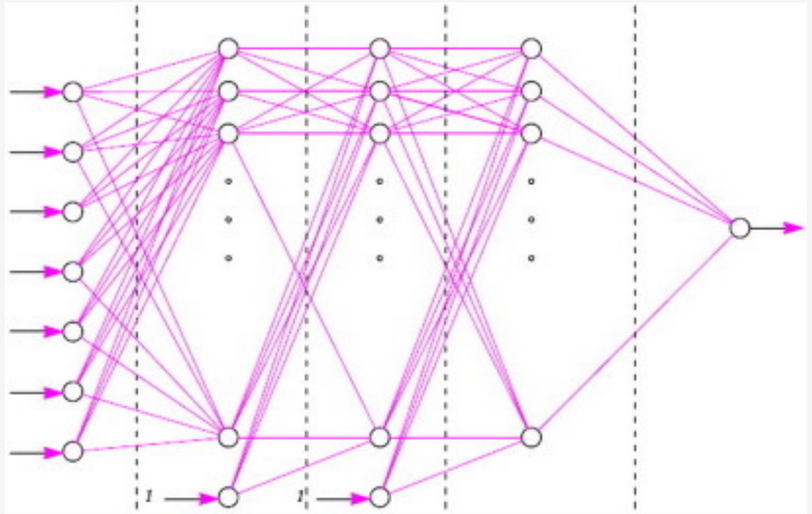


This is a ***neural network***. We denote the weights of the neural network collectively by $\mathbf{W}$. The non-linear function f is called the ***activation function***.

Design Choices: Depth or Width

Ideally, we want our architecture to be able to express a number of very complex functions, since we don't know what is appropriate for our data. So what architecture is more effective for expressing complex functions?





Neural Networks Regression

Model for Regression: $Y^{(n)} \sim \mathcal{N}(\mu, \sigma^2), \mu = g_{\mathbf{W}}(\mathbf{X}^{(n)})$, where $g_{\mathbf{W}}$ is a neural network with parameters $\mathbf{W}$.

Training Objective: find $\mathbf{W}$ to maximize the likelihood of our data. This is equivalent to minimizing the Mean Square Error,

$$\min_{\mathbf{W}} \text{MSE}(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^{N} (y_n - g_{\mathbf{W}}(x_n))^2$$

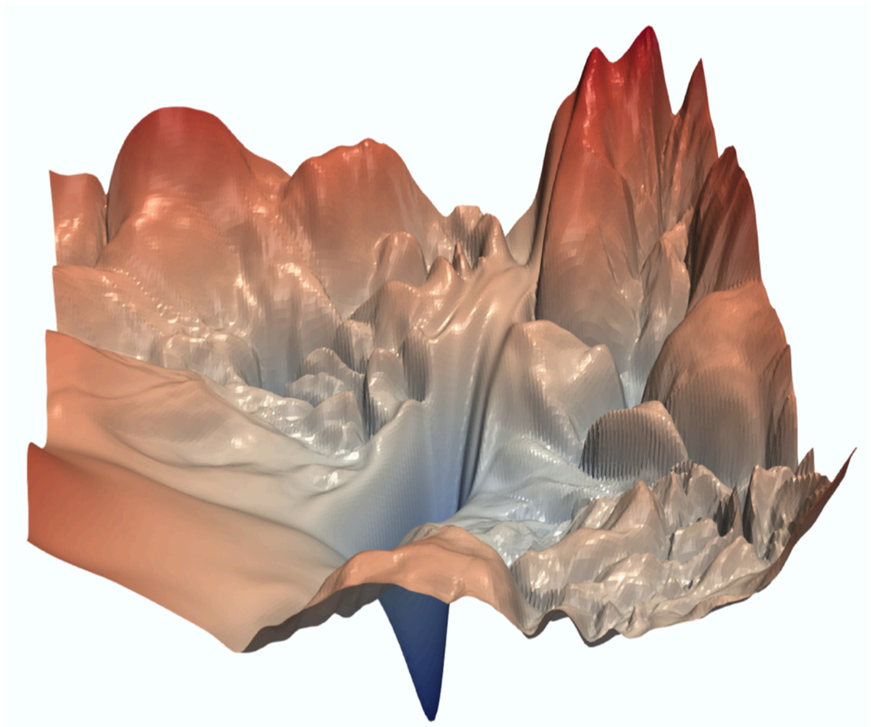
Optimizing the Training Objective: The main challenge of optimizing the objective is computing the gradient of a neural network. Luckily, the gradient can be computed in an algorithmic way using the chain rule, working from the output node **backwards** towards the input. For example, the derivative with respect to the hidden node h_i^l is:

$$\frac{\partial \text{MSE}}{\partial h_i^l} = \sum_j \underbrace{\frac{\partial \text{MSE}}{\partial h_j^{l+1}}}_{\text{derivative from layer } l} \underbrace{\frac{\partial h_j^{l+1}}{\partial h_i^l}}_{\text{derivative of } f\left(\sum_k w_k h_k\right)}$$

Using this backwards gradient computation, we can optimize a neural network with respect to the MSE using **gradient descent**.

The Maximum Likelihood Objective is Non-Convex for Neural Networks

Unfortunately, the likelihood and MSE functions for neural network regression models are not convex! This means that **just because your gradient is zero, it doens't mean you've optimized anything**.

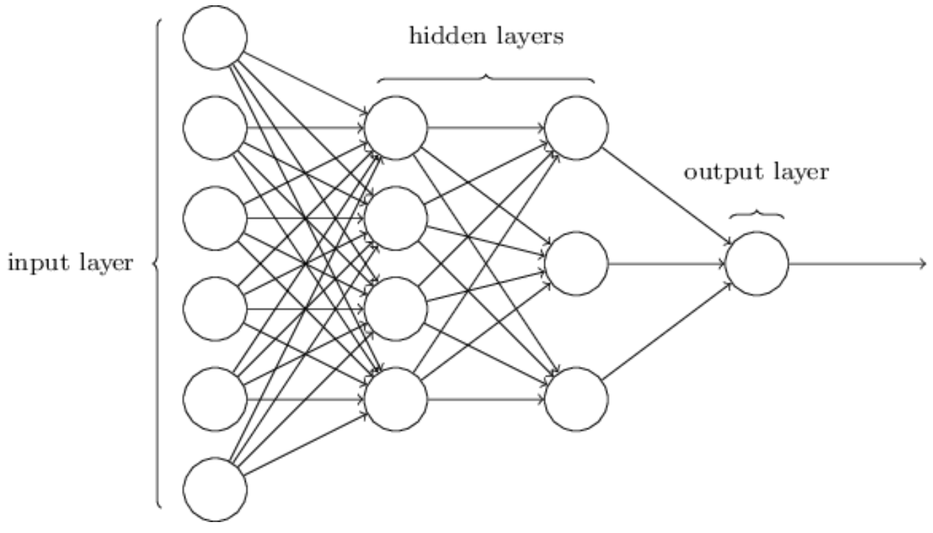


Neural Network Regression vs Linear Regression

Linear models are easy to interpret. Once we've found the MLE of the model parameters, we can formulate scientific hypotheses about the relationship between the outcome Y and the covariates $\mathbf{X}$:

$$\widehat{\text{income}} = 2 * \text{education (yr)} + 3.1 * \text{married} - 1.5 * \text{gaps in work history}$$

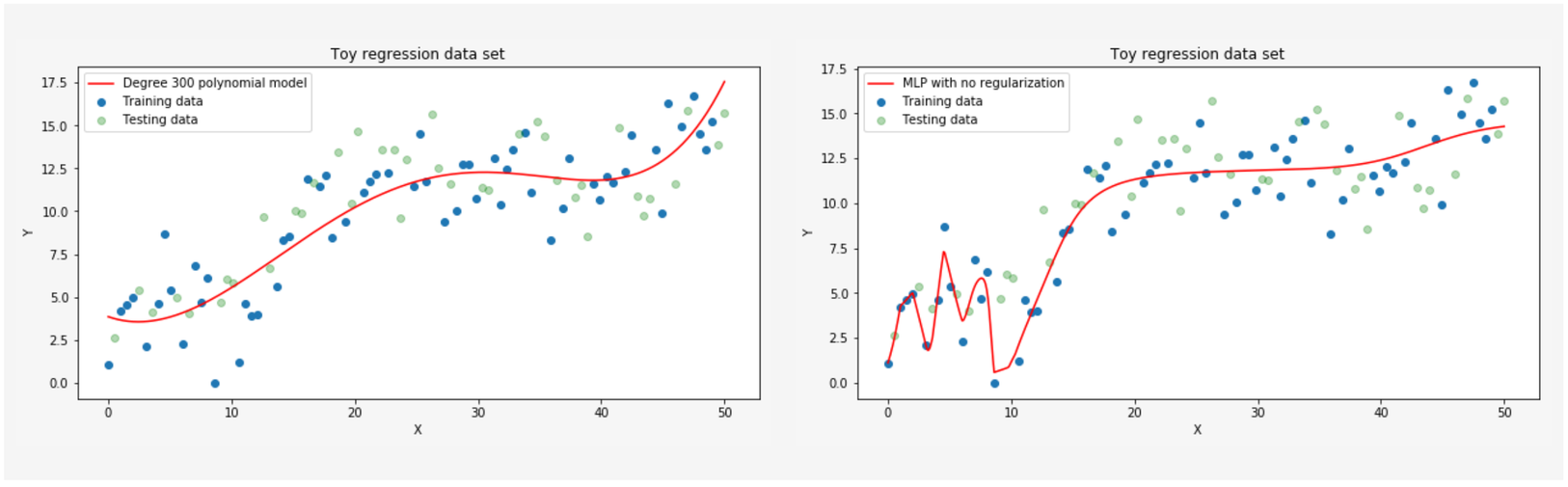
What do the weights of a neural network tell you about the relationship between the covariates and the outcome?



Generalization Error and Bias/Variance

Complex models have ***low bias*** -- they can model a wide range of functions, given enough samples.

But complex models like neural networks can use their 'extra' capacity to explain non-meaningful features of the training data that are unlikely to appear in the test data (i.e. noise). These models have ***high variance*** -- they are very sensitive to small changes in the data distribution, leading to drastic performance decrease from train to test settings.



Bayesian Neural Networks

Bayesian Polynomial Regression is Bayesian Linear Regression

A Bayesian polynomial regression model uses a polynomial of degree D to capture the relationship between $X \in \mathbb{R}$ and Y:

$$\begin{aligned}
\mathbf{W} &\sim \mathcal{N}(0, \sigma_W^2 \mathbf{I}) \\
\mu^{(n)} &= w_0 + w_1 X^{(n)} + w_2 (X^{(n)})^2 + \ldots + w_D (X^{(n)})^D \\
Y^{(n)} &\sim \mathcal{N}(\mu^{(n)}, \sigma_Y^2)
\end{aligned}$$

Rather than considering the polynomial as a non-linear function of X, we can see it as a linear function of the vector $\mathbf{X} = [1, X, X^2, \ldots, X^D] \in \mathbb{R}^{D+1}$:

$$\begin{aligned}
\mathbf{W} &\sim \mathcal{N}(0, \sigma_W^2 \mathbf{I}) \\
\mu^{(n)} &= \mathbf{W}^\top \mathbf{X}^{(n)} \\
Y^{(n)} &\sim \mathcal{N}(\mu^{(n)}, \sigma_Y^2)
\end{aligned}$$

This means that for a Bayesian polynomial regression model, the posterior $p(\mathbf{X}|\text{Data})$ is a multivariate Gaussian (just as in the case of Bayesian linear regression, see HW#0).

Bayesian Neural Networks

A **Bayesian neural network (BNN)** is a Bayesian model for regression that uses a neural network to capture the relationship between $\mathbf{X}$ and Y:

$$
\begin{aligned}
\mathbf{W} &\sim \mathcal{N}(0, \sigma_W^2 \mathbf{I}) \\
\mu^{(n)} &= g_{\mathbf{W}}(\mathbf{X}^{(n)}) \\
Y^{(n)} &\sim \mathcal{N}(\mu^{(n)}, \sigma_Y^2)
\end{aligned}
$$

Unfortunately, the posterior of a neural network is multimodal and very complex, posing a challenge for samplers. Furthermore, training data for BNNs are typically large, this makes gradient-based samplers like HMC extremely inefficient -- in every leap-frog iteration of HMC, the gradient $\frac{\partial U(q)}{\partial q}$ requires an evaluation over the entire training data set.

See papers:

1. NeuTra-lizing Bad Geometry in Hamiltonian Monte Carlo Using Neural Transport
2. Stochastic Gradient Hamiltonian Monte Carlo

Black-box Variational Inference

Review of Variational Inference

Goal: given a target posterior distribution $p(\psi|Y_1, \ldots, Y_N), \psi \in \mathbb{R}^I$ we want to find a distribution $q(\psi|\lambda^*)$ in a family of distributions $Q = \{q(\psi|\lambda)|\lambda \in \Lambda\}$ such that $q(\psi|\lambda^*)$ best approximates p.

Design Choices: we need to choose:

A. ***(Variational family)*** a family Q of candidate distributions for approximating p. The members of Q are called the ***variational distributions***.

Our Choice: we assume that the joint $q(\psi)$ factorizes completely over each dimension of ψ, i.e. $q(\psi) = \prod_{i=1}^{I} q(\psi_i|\lambda_i)$. This is called the ***mean field assumption***. What can go wrong with this design choice?

B. ***(Divergence measure)*** a divergence measure to quantify the difference between p and q.

Our Choice:

$$D_{\mathrm{KL}}(q(\psi|\lambda)\|p(\psi|Y_1, \ldots, Y_N)) = \mathbb{E}_{\psi\sim q(\psi|\lambda)}\left[\log\left(\frac{q(\psi|\lambda)}{p(\psi|Y_1, \ldots, Y_N)}\right)\right]$$

What can go wrong with this design choice?

Variational Inference as Optimization

Let's formalize variational inference for a target distribution $p(\psi)$: find a $q(\psi|\lambda^*)$ such that

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}}\ D_{\mathrm{KL}}(q(\psi|\lambda)\|p(\psi|Y_1, \dots, Y_N))) = \underset{\lambda}{\operatorname{argmin}}$$

$$\mathbb{E}_{\psi\sim q(\psi|\lambda)}\left[\log\left(\frac{q(\psi|\lambda)}{p(\psi|Y_1, \dots, Y_N))}\right)\right]$$

Recall that for EM, we had proved that minimizing the KL is equivalent to maximizing the ELBO (for which it is easier to compute the gradient). We will do the same here:

$$\begin{aligned}
\min_\lambda D_{\mathrm{KL}}(q(\psi|\lambda)\|p(\psi|Y_1, \dots, Y_N))) &\overset{\text{equiv}}{\equiv} \max_\lambda -D_{\mathrm{KL}}(q(\psi|\lambda)\|p(\psi|Y_1, \dots, Y_N))) \\
&= \max_\lambda -\mathbb{E}_{\psi\sim q(\psi|\lambda)}\left[\log\left(\frac{q(\psi|\lambda)}{p(\psi|Y_1, \dots, Y_N))}\right)\right] \\
&= \max_\lambda \underbrace{\mathbb{E}_{\psi\sim q(\psi|\lambda)}\left[\log\left(\frac{p(\psi, Y_1, \dots, Y_N))}{q(\psi|\lambda)}\right)\right]}_{ELBO(\lambda)} \\
&\quad - \log p(Y_1, \dots, Y_N).
\end{aligned}$$

Thus, the variational objective can be rephrased as maximizing the $ELBO$.

Variational Inference for Bayesian Neural Networks

Consider a Bayesian neural network:

$$\begin{aligned}\mathbf{W} &\sim \mathcal{N}(0, \sigma_W^2 \mathbf{I}) \\ Y^{(n)} &\sim \mathcal{N}(g_{\mathbf{W}}(\mathbf{X}^{(n)}), \sigma_Y^2)\end{aligned}$$

and a mean-field Gaussian variational family:

$$q(\mathbf{W}|\mu, \Sigma) = \mathcal{N}(\mathbf{W}; \mu, \Sigma)$$

where Σ is a diagonal matrix.

The ELBO is:

$$\begin{aligned}ELBO(\mu,\Sigma) &= \mathbb{E}_{\mathbf{W}\sim q(\mathbf{W}|\mu,\Sigma)}\left[\log\left(\frac{p(\mathbf{W})\prod_{n=1}^N p(Y^{(n)}|\mathbf{X}^{(n)}, \mathbf{W})}{q(\mathbf{W}|\mu, \Sigma)}\right)\right] \\ &= \mathbb{E}_{\mathbf{W}\sim q(\mathbf{W}|\mu,\Sigma)}\left[\log\left(\frac{\mathcal{N}(\mathbf{W}; 0, \sigma_W^2\mathbf{I})\prod_{n=1}^N \mathcal{N}(Y^{(n)}; g_{\mathbf{W}}(\mathbf{X}^{(n)}), \sigma_Y^2)}{\mathcal{N}(\mathbf{W}; \mu, \Sigma)}\right)\right].\end{aligned}$$

To find the optimal variational parameters such that $\mu^*, \Sigma^* = \text{argmax } ELBO(\mu,\Sigma)$, we need to compute the gradient of the ELBO:

$$\nabla_{\mu,\Sigma} \underbrace{\mathbb{E}_{\mathbf{W}\sim q(\mathbf{W}|\mu,\Sigma)}\left[\log\left(\frac{p(\mathbf{W})\prod_{n=1}^N p(Y^{(n)}|\mathbf{X}^{(n)}, \mathbf{W})}{q(\mathbf{W}|\mu, \Sigma)}\right)\right]}_{ELBO(\mu,\Sigma)}.$$

Black-Box Variational Inference

The ***Black-box Variational Inference (BBVI)*** algorithm for BNN's:

1. **Initialization:** pick an intial value $\mu^{(0)}$, $\Sigma^{(0)}$
2. **Gradient Ascent:** repeat:
   A. Approximate the gradient

$$\nabla_{\mu,\Sigma}\ ELBO(\mu,\Sigma) = \mathbb{E}_{\mathbf{W}\sim q(\mathbf{W}|\mu,\Sigma)}\left[\nabla_{\mu,\Sigma}\ q(\mathbf{W}|\mu,\Sigma) * \log\left(\frac{p(\mathbf{W})\prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)},\mathbf{W})}{q(\mathbf{W}|\mu,\Sigma)}\right)\right]$$

$$\approx \underbrace{\frac{1}{S}\sum_{s=1}^{S} \nabla_{\mu,\Sigma}\ \log q(\mathbf{W}^s|\mu,\Sigma) * \log\left(\frac{p(\mathbf{W}^s)\prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)},\mathbf{W}^s)}{q(\mathbf{W}^s|\mu,\Sigma)}\right)}_{\text{score function gradient}},$$

   where $\mathbf{W}^s \sim q(\mathbf{W}|\mu^{\text{current}}, \Sigma^{\text{current}})$.
   B. Update parameters $(\mu^{\text{current}}, \Sigma^{\text{current}}) \leftarrow (\mu^{\text{current}}, \Sigma^{\text{current}}) + \eta * \text{score function gradient}$

See Lecture Notes or the appendix of "Black-box Variational Inference" for the derivation of the expression for the gradient of the ELBO.

Variance of the Gradient Estimate

In Black-Box Variational Inference, we estimate the gradient using Monte Carlo estimation (i.e. the score function gradient approximation):

$$\nabla_{\mu,\Sigma}\ ELBO(\mathbf{W}) \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\mu,\Sigma}\ \log q(\mathbf{W}^s|\mu,\Sigma) * \log\left(\frac{p(\mathbf{W}^s)\prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)},\mathbf{W}^s)}{q(\mathbf{W}^s|\mu,\Sigma)}\right),$$

where $\mathbf{W}^s \sim q(\mathbf{W}|\mu^{\text{current}}, \Sigma^{\text{current}})$.

As in the case with every MC estimate, we worry about variance. As it turns out, the score function gradient approximation has very high variance. This leads to slow convergence for gradient descent. To mitigate this, we need to employ a number of **variance reduction methods**.

In the original paper "Black-Box Variational Inference", the authors implement control variates and Rao-Blackwellization.

Gradient of the ELBO with the Reparametrization Trick

An alternative to using the log-derivative trick to computing the gradient of the ELBO is to use the reparametrization trick.

We note that since $q(\mathbf{W}|\mu, \Sigma) = \mathcal{N}(\mathbf{W}; \mu, \Sigma)$, sampling $W \sim q(\mathbf{W}|\mu, \Sigma)$ is equivalent to sampling $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and then transforming the sample $\mathbf{W} = \epsilon^\top \Sigma^{1/2} + \mu$, where $\mathbf{I}$ and Σ have the same dimensions.

Thus, we can rewrite the ELBO:

$$
\begin{aligned}
\nabla_{\mu,\Sigma} \, ELBO(\mu,\Sigma) &= \nabla_{\mu,\Sigma} \mathbb{E}_{\mathbf{W}\sim\mathcal{N}(\mu,\Sigma)} \left[\log\left(\frac{p(\mathbf{W}) \prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)}, \mathbf{W})}{q(\mathbf{W}|\mu, \Sigma)} \right) \right] \\
&= \nabla_{\mu,\Sigma} \mathbb{E}_{\mathbf{W}\sim\mathcal{N}(\mu,\Sigma)} \left[\log\left(p(\mathbf{W}) \prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)}, \mathbf{W}) \right) \right] - \nabla_{\mu,\Sigma} \underbrace{\mathbb{E}_{\mathbf{W}\sim\mathcal{N}(\mu,\Sigma)} \left[\log\left(\mathcal{N}(\mathbf{W}; \mu, \Sigma) \right) \right]}_{\text{Guassian entropy: has closed form}} \\
&= \underbrace{\nabla_{\mu,\Sigma} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathbf{I})} \left[\log\left(p(\epsilon^\top \Sigma^{1/2} + \mu) \prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)}, \epsilon^\top \Sigma^{1/2} + \mu) \right) \right]}_{\text{reparametrization trick}} - \nabla_{\mu,\Sigma} \underbrace{\mathbb{E}_{\mathbf{W}\sim\mathcal{N}(\mu,\Sigma)} \left[\log \mathcal{N}(\mathbf{W}; \mu, \Sigma) \right]}_{\text{Guassian entropy: has closed form}} \\
&= \underbrace{\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\mathbf{I})} \left[\nabla_{\mu,\Sigma} \log\left(p(\epsilon^\top \Sigma^{1/2} + \mu) \prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)}, \epsilon^\top \Sigma^{1/2} + \mu) \right) \right]}_{\text{exchanging gradient and expectation}} - \nabla_{\mu,\Sigma} \underbrace{\mathbb{E}_{\mathbf{W}\sim\mathcal{N}(\mu,\Sigma)} \left[\log \mathcal{N}(\mathbf{W}; \mu, \Sigma) \right]}_{\text{Guassian entropy: has closed form}}
\end{aligned}
$$

Black-Box Variational Inference with the Reparametrization Trick

The ***Black-box Variational Inference (BBVI) with the reparametrization trick*** or ***Bayes By Backprop*** algorithm for BNN's:

1. **Initialization:** pick an intial value $\mu^{(0)}$, $\Sigma^{(0)}$
2. **Gradient Ascent:** repeat:
   A. Approximate the gradient

$$\nabla_{\mu,\Sigma}\ ELBO(\boldsymbol{\mu},\boldsymbol{\Sigma}) \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\mu,\Sigma} \log\left[p(\epsilon_s^\top \Sigma^{1/2} + \mu)\prod_{n=1}^{N} p(Y^{(n)}|\mathbf{X}^{(n)}, \epsilon_s^\top \Sigma^{1/2} + \mu)\right]$$

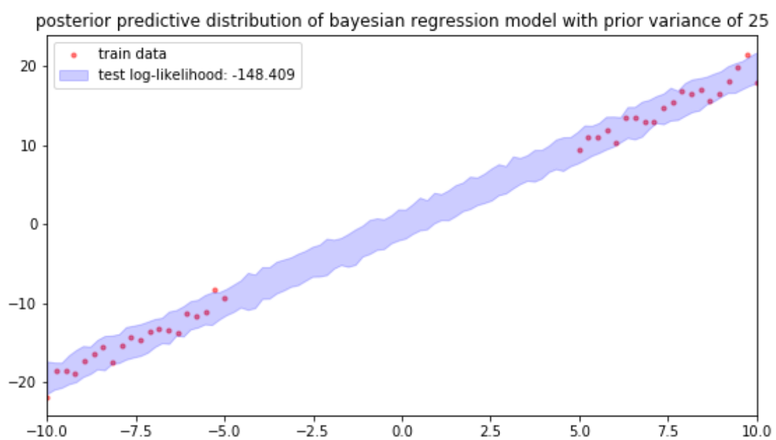
$$-\nabla_{\mu,\Sigma}\underbrace{\mathbb{E}_{\mathbf{W}\sim\mathcal{N}(\mu,\Sigma)}\left[\log \mathcal{N}(\mathbf{W};\mu,\Sigma)\right]}_{\text{Guassian entropy: has closed form}},$$

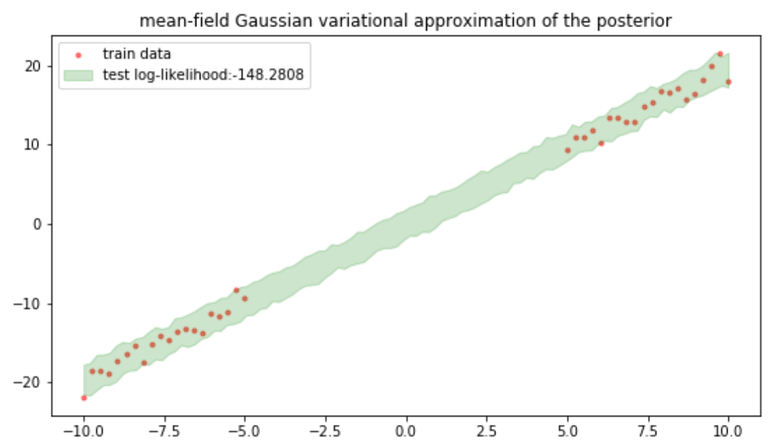
   where $\epsilon_s \sim \mathcal{N}(0, \mathbf{I})$.
   B. Update parameters $(\mu^{\text{current}}, \Sigma^{\text{current}}) \leftarrow (\mu^{\text{current}}, \Sigma^{\text{current}}) + \eta * \text{score function gradient}$

BBVI for Bayesian Linear Regression (Posterior Predictives)

In [5]:
```
posterior_sample_size = 100
##visualize the posterior predictive of a bayesian polynomial regression model
#prior variance
prior_var = 5**2
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
ax[0], joint_variance, joint_mean = bayesian_polynomial_regression(x_train, y_tr
ain, x_test, y_test, prior_var, y_var, ax[0], S=posterior_sample_size, poly_degr
ee=1)
ax[1], var_variance, var_mean = variational_polynomial_regression(Sigma_W, sigma
_y, x_test, y_test, x_train, y_train, forward, ax[1], posterior_sample_size=post
erior_sample_size, S=4000, max_iteration=2000, step_size=1e-1, verbose=False)
plt.show()
```

```
Optimizing variational parameters...
```





BBVI for Bayesian Linear Regression (Posteriors)

In [33]:
```
#plot the target density against variational densities
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
ax.contourf(x, y, z_p, levels=14, cmap='Reds', alpha=0.8)
ax.contour(x, y, z_p, levels=14, cmap='Reds', alpha=0.8)
ax.contour(x, y, z_q, levels=14, cmap='Blues', alpha=1.)
ax.set_title('Approximate posterior (blue) vs actual posterior (red)')
plt.show()
```