Weight Uncertainty in Neural Networks

Nova Search Reading Group

by: Simão Gonçalves
date: 2 July 2020

Motivation

- Neural Networks lack a confidence metric on their predictions
  - Critical applications need this: self driving, fraud detection, credit scoring...

Some background first

- Marginal probability:

$$\forall x \in \mathrm{x},\ P(\mathrm{x} = x) = \sum_{y} P(\mathrm{x} = x, \mathrm{y} = y)$$

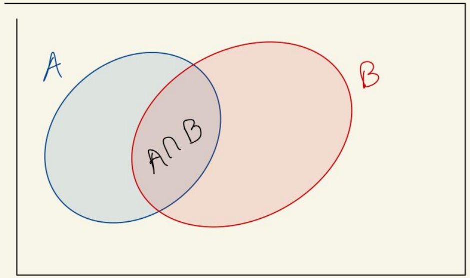
Joint distribution: $P(H, L)$

H \ L	Red	Yellow	Green	Marginal probability P(*H*)
Not Hit	0.198	0.09	0.14	0.428
Hit	0.002	0.01	0.56	0.572
Total	0.2	0.1	0.7	1

Some background first

- Conditional probability:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$



What portion of B contains A?

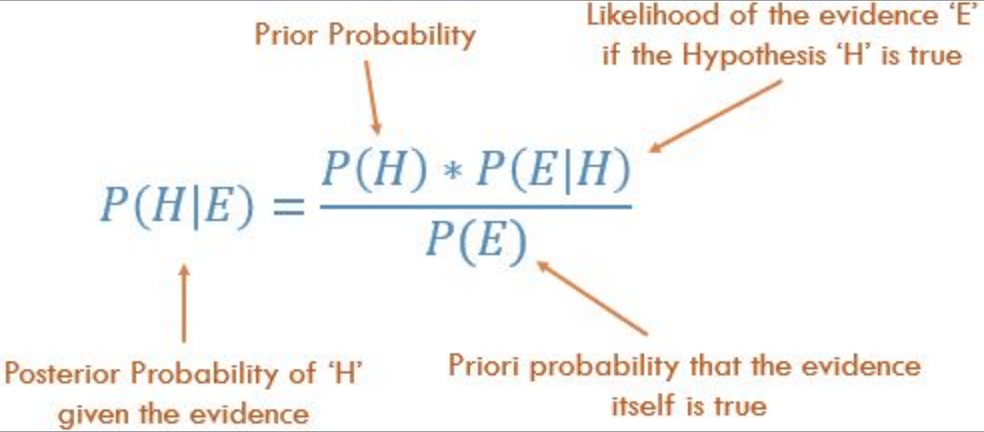
Some background first

- Bayes theorem:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

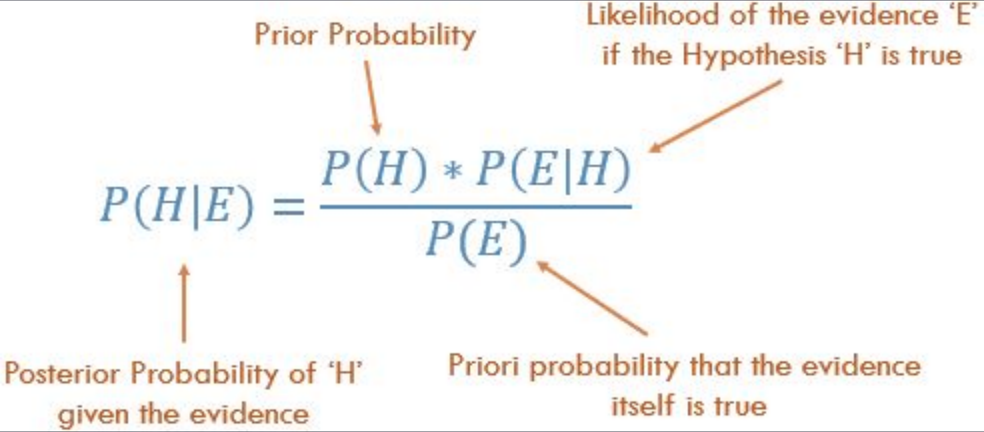
Some background first

- Bayes theorem:



Some background first

- Bayes theorem:

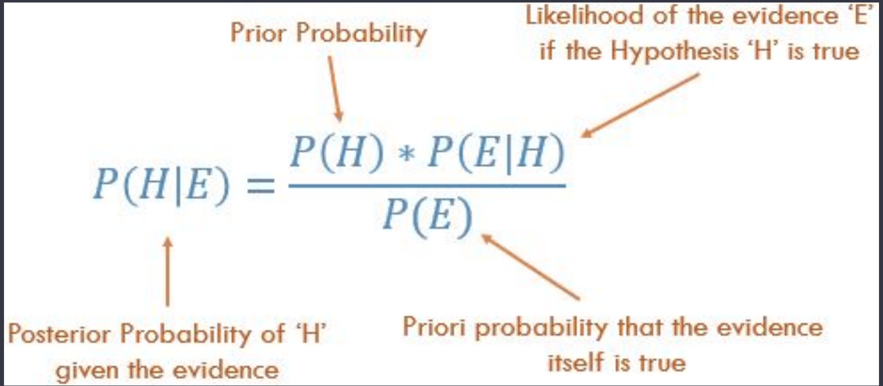


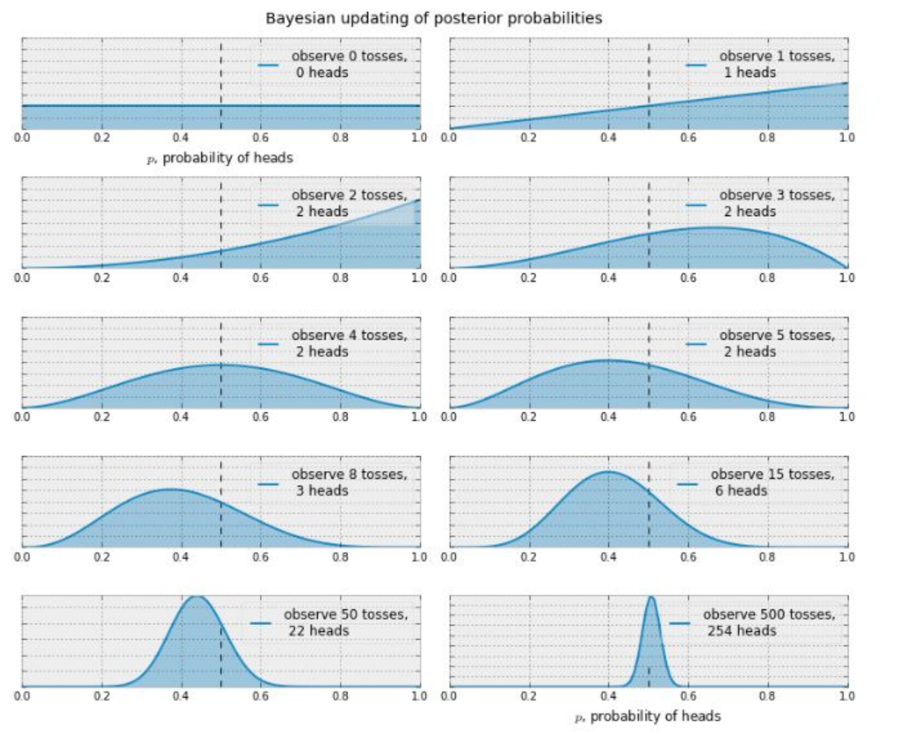
Example with coin tosses

Some background first

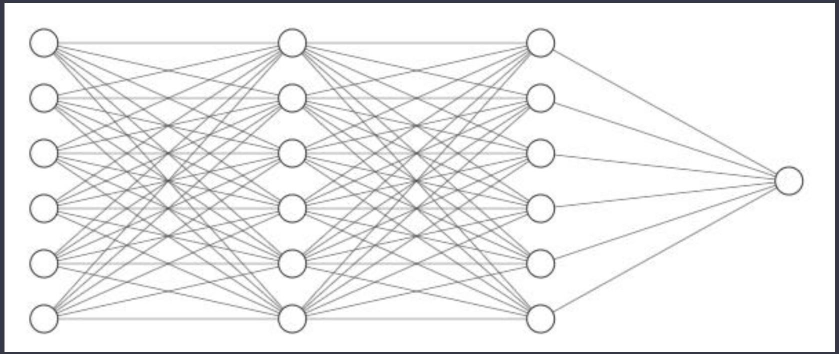
Example with coin tosses

- Bayes theorem:





Probabilistic View on Neural Networks



$$p_z(y|x)$$

Probabilistic View on Neural Networks

$$z^{MLE} = \underset{z}{arg\,max}\ \log P(D|z)$$
$$= \underset{z}{arg\,max} \sum_i \log P(y_i|x_i, z)$$

Probabilistic View on Neural Networks

Bayesian setup

$$\{x^{(i)}, y^{(i)}\}_{i=1:M} \longrightarrow p(y|x))$$

Posterior

$$p(z|D) = \frac{p(D|z)p(z)}{p(D)}$$

Predictions

$$
\begin{aligned}
p(\hat{y}|\hat{x}, \mathcal{D}) &= \int_Z p(\hat{y}|\hat{x}, z)p(z|\mathcal{D})dz \\
&= E_{p(z|\mathcal{D})}\left[P(\hat{y}|\hat{x}, z)\right]
\end{aligned}
$$

- Ensemble of predictions

Bad news: it is intractable.

Variational Learning

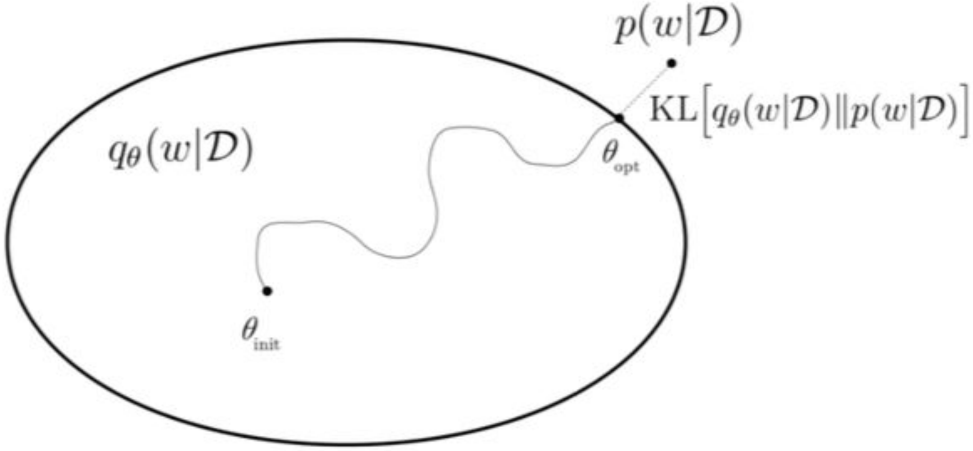


Fig. 1: Intuition of variational learning.

Source:https://medium.com/neuralspace/probabilistic-deep-learning-bayes-by-backprop-c4a3de0d9743

Variational Learning

How do we find q?

1. Minimize KL divergence between the two distributions.

$$\theta_{\text{opt}} = \underset{\theta \in \Theta}{argmin}\ \mathbf{KL}\big(q_\theta(z) || p(z|x)\big)$$

Variational Learning

$$\mathrm{KL}\Big(q_\theta(z)||p(z|x)\Big) = \int q_\theta(z) log \frac{q_\theta(z)}{p(z|x)} dz$$
$$= E_{q_\theta(z)}\left[log \frac{q_\theta(z)}{p(z|x)}\right]$$

Variational Learning

$$\mathrm{KL}\Big(q_\theta(z)||p(z|x)\Big) = \int q_\theta(z) log \frac{q_\theta(z)}{p(z|x)} dz$$
$$= E_{q_\theta(z)}\Big[log \frac{q_\theta(z)}{p(z|x)}\Big]$$

• • •

$$= -\Big(E_{q_\theta(z)}\big[\log p(z,x)\big] - E_{q_\theta(z)}\big[\log q_\theta(z)\big]\Big)$$
$$+ \log p(x)$$

Variational Learning

Evidence Lower Bound (ELBO): $$E_{q_\theta(z)}\big[\log p(z,x)\big] - E_{q_\theta(z)}\big[\log q_\theta(z)\big]$$

$$= E_{q_\theta(x|z)}\big[p(x|z)\big] + \boldsymbol{KL}\big[p(x)||q(z)\big]$$

Variational Learning

Evidence Lower Bound (ELBO): $$E_{q_\theta(z)}\left[\log p(z,x)\right] - E_{q_\theta(z)}\left[\log q_\theta(z)\right]$$

$$= E_{q_\theta(x|z)}\left[p(x|z)\right] + \boldsymbol{KL}\left[p(x)||q(z)\right]$$

Maximizing ELBO minimizes the KL divergence!

Cost Function

As we saw earlier:

$$\begin{aligned}\theta_{\text{opt}} &= \underset{\theta}{argmin}\ \mathrm{KL}\big(q_\theta(z)||p(z|\mathcal{D})\big)\\ &= \underset{\theta}{argmin}\int q_\theta(z)\log\frac{q_\theta(z)}{p(z)p(\mathcal{D}|z)}dz\\ &= \underset{\theta}{argmin}\ \mathrm{KL}\big[q_\theta(z)||p(z)\big] - E_{q_\theta(z)}[\log p(\mathcal{D}|z)]\end{aligned}$$

Cost function

$$F(\mathcal{D},\theta) = \mathrm{KL}\big[q_\theta(z)||p(z)\big] - E_{q_\theta(z)}[\log p(\mathcal{D}|z)]$$

Gradient Learning

$$\frac{\partial}{\partial z} F(D, \theta)$$

→

We can't compute gradients of expectations.

Gradient Learning

Authors' contribution: Bayes by Backprop

Reparameterize with a deterministic function:

$$\epsilon \sim q(\epsilon)$$

$$w = t(\theta, \epsilon) \text{ - (deterministic)}$$

Under certain conditions:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{q(\mathbf{w}|\theta)}[f(\mathbf{w}, \theta)] = \mathbb{E}_{q(\epsilon)} \left[\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial \theta} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \theta} \right]$$

Gradient Learning

Authors' contribution: Bayes by Backprop

Let's say our weights follow a normal distribution:

$$\mathbf{w} \sim N(\mu, \sigma^2)$$

$$\epsilon \sim N(0, I)$$

$$f(\epsilon) = \mathbf{w} = \mu + \sigma \cdot \epsilon$$

Gradient Learning

Authors’ contribution: Bayes by Backprop

$$\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon.$$

Let’s say our weights follow a normal distribution:

$$\mathbf{w} \sim N(\mu, \sigma^2)$$

$$\epsilon \sim N(0, I)$$

$$f(\epsilon) = \mathbf{W} = \mu + \sigma \cdot \epsilon$$

$$\sigma = \log(1 + \exp(\rho))$$

Gradient Learning

Authors' contribution: Bayes by Backprop

Let's say our weights follow a normal distribution:

$$\mathbf{w} \sim N(\mu, \sigma^2)$$
$$\epsilon \sim N(0, I)$$
$$f(\epsilon) = \mathbf{W} = \mu + \sigma \cdot \epsilon$$

$$\sigma = \log(1 + \exp(\rho))$$

$$\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon.$$

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}.$$

$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}$$

Gradient Learning

Authors' contribution: Bayes by Backprop

Let's say our weights follow a normal distribution:

$$\mathbf{w} \sim N(\mu, \sigma^2)$$
$$\epsilon \sim N(0, I)$$
$$f(\epsilon) = \mathbf{w} = \mu + \sigma \cdot \epsilon$$

$$\sigma = \log(1 + \exp(\rho))$$

$$\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon.$$

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}.$$

$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}$$

$$\mu \leftarrow \mu - \alpha \Delta_\mu$$
$$\rho \leftarrow \rho - \alpha \Delta_\rho$$

Going back to the cost function

$$F(\mathcal{D}, \theta) = \mathbf{KL}\left[q_\theta(z) || p(z)\right] - E_{q_\theta(z)}[\log p(\mathcal{D}|z)]$$

$$= E[\log q_\theta(z^{(i)}|D)] - E[\log p(z^{(i)})] - E[\log p(D|z^{(i)})]$$

We approximate it by taking samples:

$$F(\mathcal{D}, \theta) \approx f(z, \theta) = \sum_{i=1}^{n} \log q_\theta(z^{(i)}) - \log p(z^{(i)}) - p(\mathcal{D}|z^{(i)})$$

How do we build this?

```python
class Linear_BBB(nn.Module):
    """
        Layer of our BNN.
    """
    def __init__(self, input_features, output_features, prior_var=1.):
        """
            Initialization of our layer : our prior is a normal distribution
            centered in 0 and of variance 20.
        """
        # initialize layers
        super().__init__()
        # set input and output dimensions
        self.input_features = input_features
        self.output_features = output_features

        # initialize mu and rho parameters for the weights of the layer
        self.w_mu = nn.Parameter(torch.zeros(output_features, input_features))
        self.w_rho = nn.Parameter(torch.zeros(output_features, input_features))

        #initialize mu and rho parameters for the layer's bias
        self.b_mu =  nn.Parameter(torch.zeros(output_features))
        self.b_rho = nn.Parameter(torch.zeros(output_features))

        #initialize weight samples (these will be calculated whenever the layer makes a prediction)
        self.w = None
        self.b = None

        # initialize prior distribution for all of the weights and biases
        self.prior = torch.distributions.Normal(0,prior_var)
```

How do we build this?

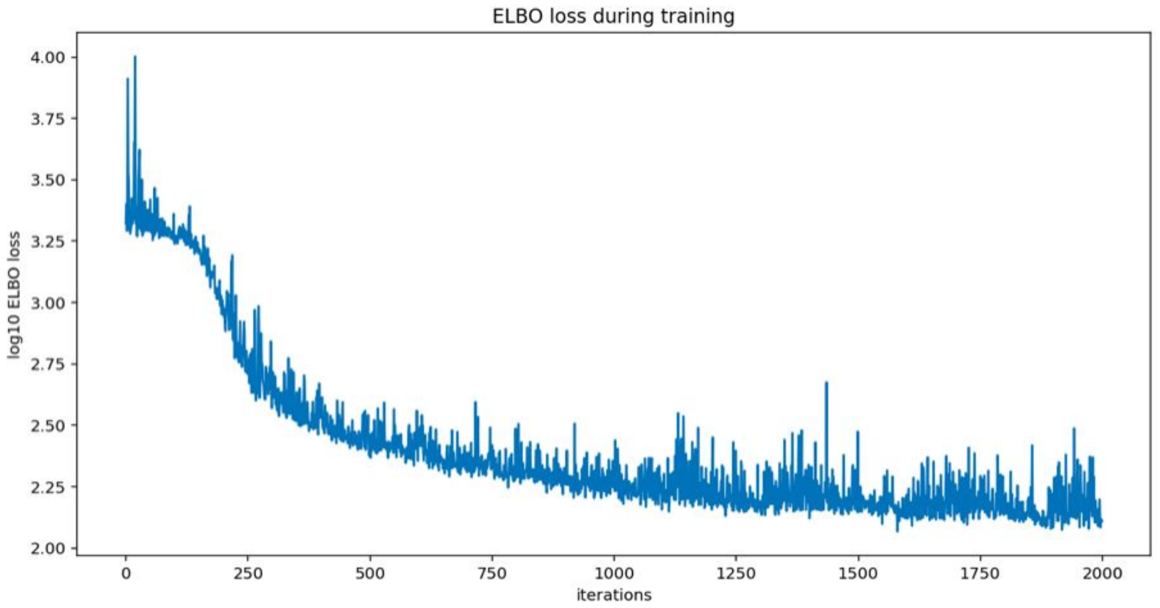
```python
def forward(self, input):
    """
      Optimization process
    """
    # sample weights
    w_epsilon = Normal(0,1).sample(self.w_mu.shape)
    self.w = self.w_mu + torch.log(1+torch.exp(self.w_rho)) * w_epsilon

    # sample bias
    b_epsilon = Normal(0,1).sample(self.b_mu.shape)
    self.b = self.b_mu + torch.log(1+torch.exp(self.b_rho)) * b_epsilon

    # record log prior by evaluating log pdf of prior at sampled weight and bias
    w_log_prior = self.prior.log_prob(self.w)
    b_log_prior = self.prior.log_prob(self.b)
    self.log_prior = torch.sum(w_log_prior) + torch.sum(b_log_prior)

    # record log variational posterior by evaluating log pdf of normal distribution
    # defined by parameters with respect at the sampled values
    self.w_post = Normal(self.w_mu.data, torch.log(1+torch.exp(self.w_rho)))
    self.b_post = Normal(self.b_mu.data, torch.log(1+torch.exp(self.b_rho)))
    self.log_post = self.w_post.log_prob(self.w).sum() + self.b_post.log_prob(self.b).sum()

    return F.linear(input, self.w, self.b)
```

How do we build this?

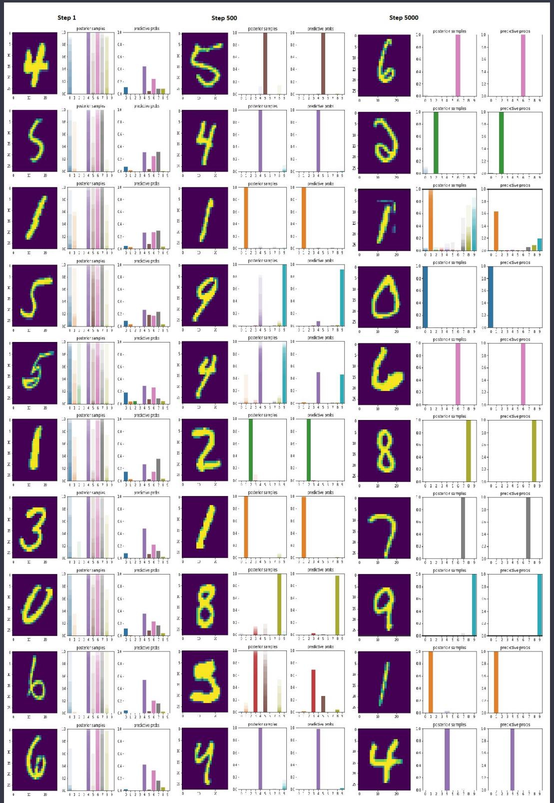
```
def sample_elbo(self, input, target, samples):
    # we calculate the negative elbo, which will be our loss function
    #initialize tensors
    outputs = torch.zeros(samples, target.shape[0])
    log_priors = torch.zeros(samples)
    log_posts = torch.zeros(samples)
    log_likes = torch.zeros(samples)
    # make predictions and calculate prior, posterior, and likelihood for a given number of samples
    for i in range(samples):
        outputs[i] = self(input).reshape(-1) # make predictions
        log_priors[i] = self.log_prior() # get log prior
        log_posts[i] = self.log_post() # get log variational posterior
        log_likes[i] = Normal(outputs[i], self.noise_tol).log_prob(target.reshape(-1)).sum() # calculate the log likelihood
    # calculate monte carlo estimate of prior posterior and likelihood
    log_prior = log_priors.mean()
    log_post = log_posts.mean()
    log_like = log_likes.mean()
    # calculate the negative elbo (which is our loss function)
    loss = log_post - log_prior - log_like
    return loss
```

How do we build this?

```
plt.figure(figsize=(12,6))
plt.plot(range(len(loss_history)), np.log10(loss_history))
plt.title('ELBO loss during training')
plt.xlabel('iterations')
plt.ylabel('log10 ELBO loss')
plt.show()
```



Bayesian deep learning with CNNs - MNIST



Bayesian deep learning with CNNs - MNIST

- What happens if we input an image out of distribution of the training set?

Bayesian deep learning with CNNs - MNIST

- What happens if we input an image out of distribution of the training set?