Bayesian Neural Networks

Spring 2024

Instructor: Shandian Zhe
zhe@cs.utah.edu
School of Computing

THE UNIVERSITY OF UTAH

Outline

- Neural networks and Back-propagation
- Stochastic optimization
- Bayesian neural networks
- Bayes by Backprop and reparameterization trick
- Auto-encoding variational Bayes

Outline

- Neural networks and Back-propagation
- Stochastic optimization
- Bayesian neural networks
- Bayes by Backprop and reparameterization trick
- Auto-encoding variational Bayes

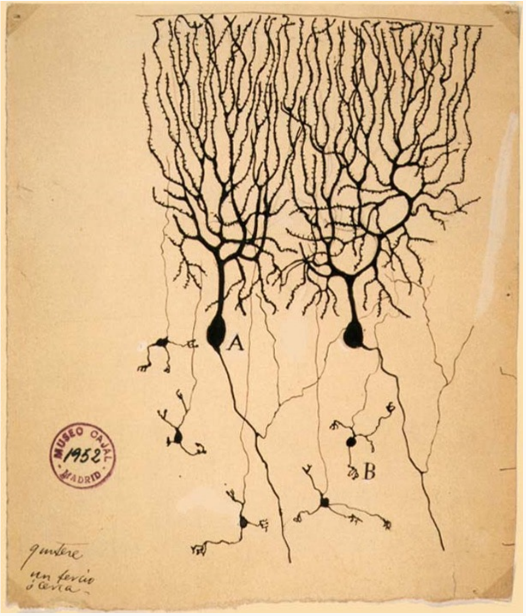
Very brief history

neural networks – very old topic

- 1943: McCullough and Pitts showed how linear threshold units can compute logical functions
- 1949: Hebb suggested a learning rule that has some physiological plausibility
- 1950s: Rosenblatt, the Peceptron algorithm for a single threshold neuron
- 1969: Minsky and Papert studied the neuron from a geometrical perspective
- 1980s: Convolutional neural networks (Fukushima, LeCun), the backpropagation algorithm (various)
- 2003-today: More compute, more data, deeper networks

See also: http://people.idsia.ch/~juergen/deep-learning-overview.html

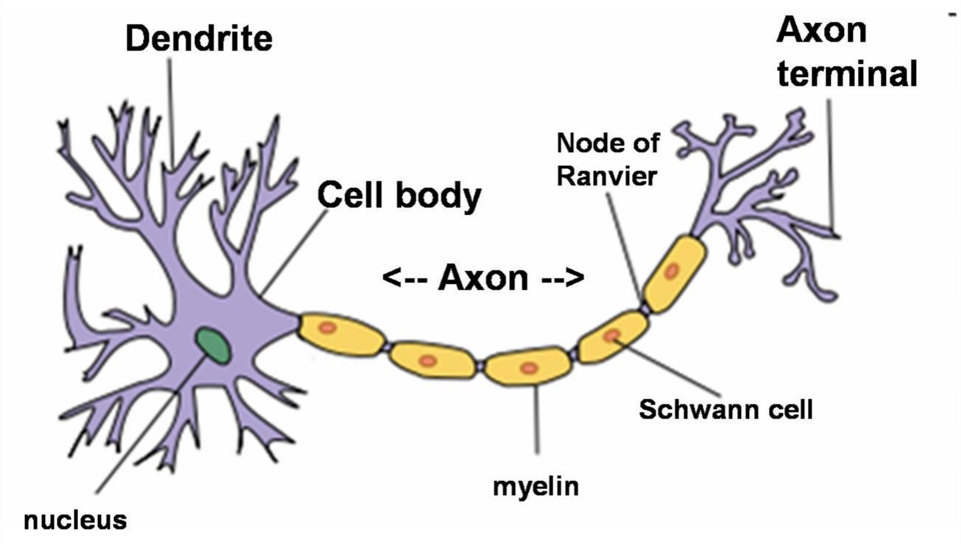
Biological neurons



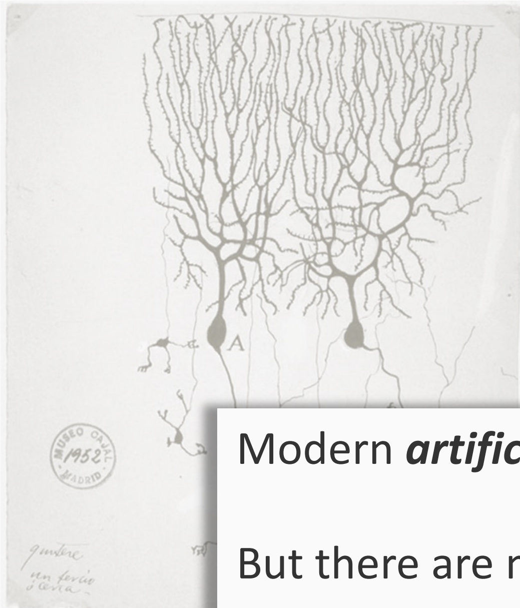
The first drawing of a brain cells by Santiago Ramón y Cajal in 1899

Neurons: core components of brain and the nervous system consisting of

1. Dendrites that collect information from other neurons
2. An axon that generates outgoing spikes



Biological neurons



The first d
cells by Sa
Cajal in 18

Neurons: core components of brain and the nervous system consisting of

1. Dendrites that collect information from other neurons
2. An axon that generates outgoing spikes

Modern ***artificial*** neurons are “inspired” by biological neurons

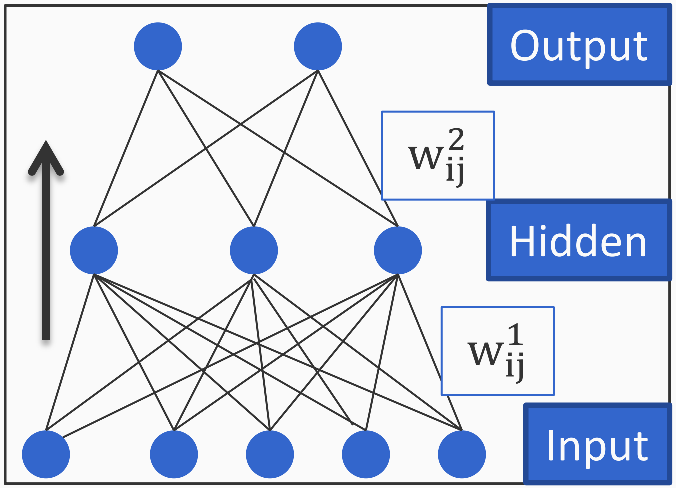
But there are many, many fundamental differences

Don’t take the similarity seriously (as also claims in the news about the “emergence” of intelligent behavior)

An artificial neural network

A function that converts inputs to outputs defined by a directed acyclic graph

- Nodes organized in layers, correspond to neurons
- Edges carry output of one neuron to another, associated with weights



- To define a neural network, we need to specify:
  - The structure of the graph
    - How many nodes, the connectivity
  - The activation function on each node
  - The edge weights

Called the *architecture* of the network

Typically predefined, part of the design of the classifier

Learned from data

Activation functions

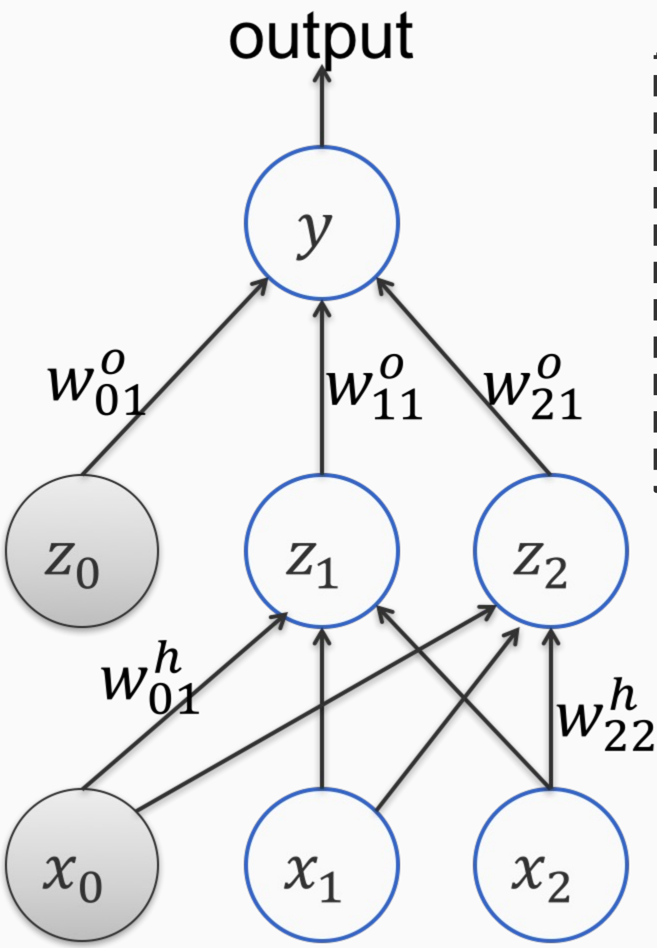
Also called transfer functions

$$output = activation(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Name of the neuron	**Activation function:** $activation(z)$
Linear unit	z
Threshold/sign unit	$\mathrm{sgn}(z)$
Sigmoid unit	$\frac{1}{1 + \exp(-z)}$
Rectified linear unit (ReLU)	$\max(0, z)$
Tanh unit	$\tanh(z)$

Many more activation functions exist (sinusoid, sinc, Gaussian, polynomial…)

An example network represented by scalars



Given an input **x**, how is the output predicted

output $y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

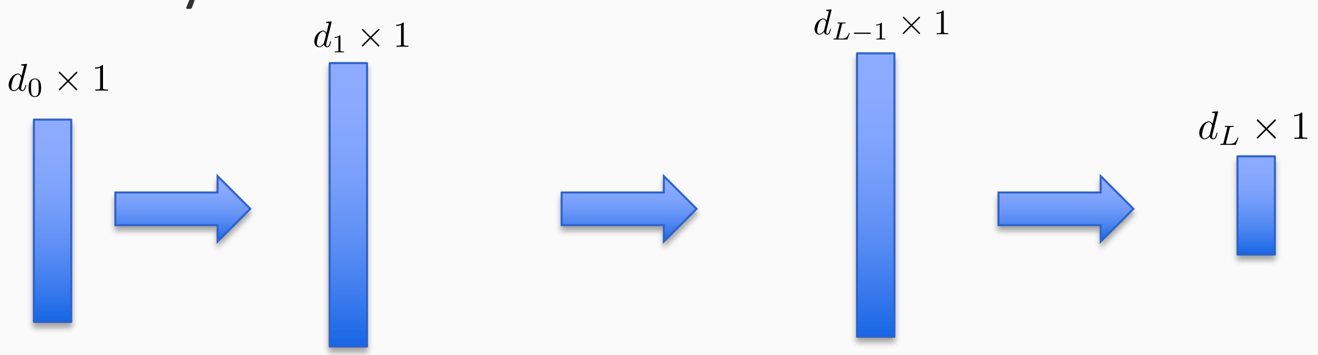
Suppose the true label for this example is a number y^*

We can write the *square loss* for this example as:

$$L = \frac{1}{2}(y - y^*)^2$$

Neural networks – A Succinct Representation

An L-layer NN



$$\mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \ldots \mathbf{x}_{L-1} \rightarrow \mathbf{x}_L$$

Arbitrary element-wise activation function

$$\mathbf{x}_j = \sigma(\mathbf{W}_j \mathbf{x}_{j-1}) \, (1 \leq j \leq L-1)$$ middle layer

$$\mathbf{f}_{\text{out}} = \mathbf{x}_L = \mathbf{W}_L \mathbf{x}_{L-1}$$ output layer

$$\mathbf{W}_j : d_j \times d_{j-1}$$

d_0 : input dim.

d_L : output dim.

Neural networks – A succinct representation

$$\mathbf{x}_0 \rightarrow \mathbf{x}_1 \rightarrow \ldots \mathbf{x}_{L-1} \rightarrow \mathbf{x}_L$$

$$\mathbf{x}_j = \sigma\big(\mathbf{W}_j \mathbf{x}_{j-1}\big) \, (1 \le j \le L-1)$$ Middle layer

$$\mathbf{f}_{\text{out}} = \mathbf{x}_L = \mathbf{W}_L \mathbf{x}_{L-1}$$ output layer

We can also recursively write

$$\mathbf{f}_{\mathcal{W}}(\mathbf{x}_0) = \mathbf{f}_{\text{out}} = \mathbf{W}_L \sigma(\mathbf{W}_{L-1} \sigma(\ldots \sigma(\mathbf{W}_1 \mathbf{x}_0)))$$

$$\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$$

Forward-pass

- To compute the output, you need to start from the bottom level and sequentially pass each layer

$$\mathbf{x}_0 \to \mathbf{x}_1 \to \ldots \mathbf{x}_{L-1} \to \mathbf{x}_L$$

This is called forward pass

Back-Propagation: Application of Chain Rule

In general, training NN is to minimize a loss function $\mathcal{L}(\mathcal{W}, \mathcal{D})$ where $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

For example, square loss:

$$\mathcal{L}(\mathcal{W}, \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} [y^{(n)} - f_{\mathcal{W}}(\mathbf{x}^{(n)})]^2$$

Back-Propagation: Application of Chain Rule

In general, training NN is to minimize a loss function $\mathcal{L}(\mathcal{W}, \mathcal{D})$ where $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$

e.g., $\mathcal{L}(\mathcal{W}, \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}[y^{(n)} - f_{\mathcal{W}}(\mathbf{x}^{(n)})]^2$

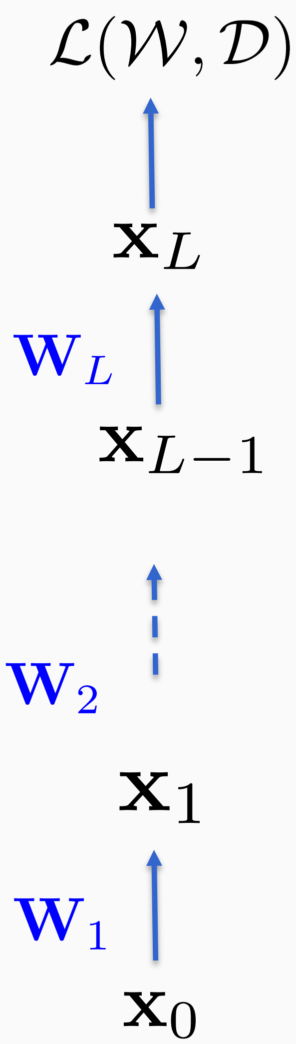
$$\mathbf{f}_{\mathcal{W}}(\mathbf{x_0}) = \mathbf{x}_L$$

$$\mathbf{x}_0 \xrightarrow{\mathbf{W}_1} \mathbf{x}_1 \xrightarrow{\mathbf{W}_2} \ldots \mathbf{x}_{L-1} \xrightarrow{\mathbf{W}_L} \mathbf{x}_L$$

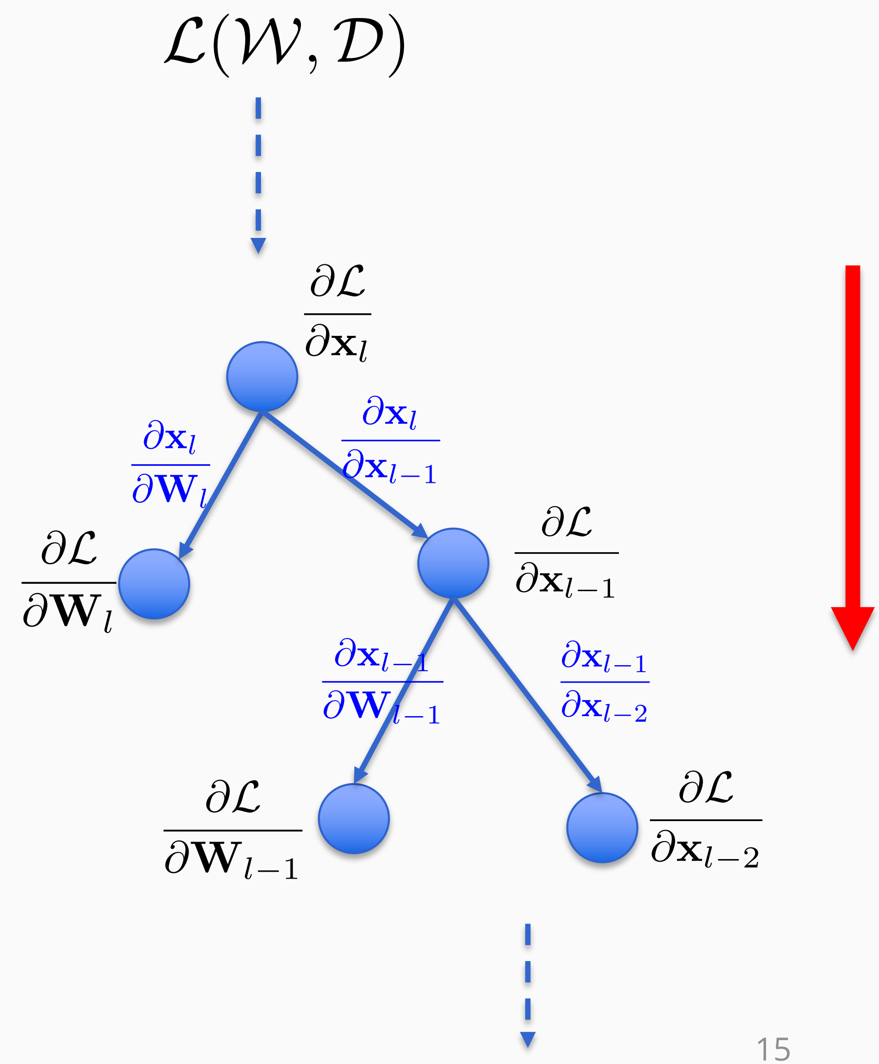
How to efficiently compute gradient?

Do it in backward!

Back-Propagation: Application of Chain Rule



from the root



Back-Propagation

- We will not discuss the detail because
  - It is trivial and mechanical
  - Nowadays, you never need to implement BP by yourself. TensorFlow, PyTorch, … will do this automatically for you

Outline

- Neural networks and Back-propagation
- Stochastic optimization
- Bayesian neural networks
- Bayes by Backprop and reparameterization trick
- Auto-encoding variational Bayes
- General adversarial networks

Stochastic optimization

- Suppose we aim to optimize an objective function that can be viewed as an expectation

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p(u)}[g(\boldsymbol{\theta}, u)]$$

- Then we can compute a stochastic gradient for stochastic optimization

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = \nabla\mathbb{E}_{p(u)}[g(\boldsymbol{\theta}, u)] = \mathbb{E}_{p(u)}[\nabla g(\boldsymbol{\theta}, u)]$$

under certainty conditions

Stochastic optimization

- Suppose we aim to optimize an objective function that can be viewed as an expectation

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{p(u)}[g(\boldsymbol{\theta}, u)]$$

- Then we can compute a stochastic gradient for stochastic optimization

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = \nabla\mathbb{E}_{p(u)}[g(\boldsymbol{\theta}, u)] = \mathbb{E}_{p(u)}[\boxed{\nabla g(\boldsymbol{\theta}, u)}]$$

under certainty conditions

Stochastic optimization: General Recipe

- 1. Initialize $\boldsymbol{\theta}$ randomly (or 0)
- 2. For t = 1.. T
  - Sample *u* from *p(u)*
  - Calculate stochastic gradient $\nabla g(\boldsymbol{\theta}, u)$
  - Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}$ - $\gamma_t \nabla g(\boldsymbol{\theta}, u)$
- 3. Return $\boldsymbol{\theta}$

γ_t: learning rate, many tweaks possible

Convergence and learning rates

With enough iterations, it will converge almost surely (i.e., with probability one)

Provided the step sizes are "*square summable, but not summable*"

- Step sizes γ_t are positive
- Sum of squares of step sizes over t = 1 to ∞ is not infinite
- Sum of step sizes over t = 1 to ∞ is infinity

- Some examples: $\gamma_t = \frac{\gamma_0}{1+\frac{\gamma_0 t}{C}}$ or $\gamma_t = \frac{\gamma_0}{1+t}$

There are numerous ways to determine to per-element learning rate

- Learning rate is critical to convergence rate
- There are many works that develop learning rate schedules
- The main-stream is momentum-based approaches
- Most popular approaches include ADAM, Adagrad, Adadelta, etc.
- There are well developed libraries, and you do not need to implement them by yourself.

Why stochastic optimization is so important?

- It is the foundation of modern NN training

$$\mathcal{L}(\mathcal{W}, \mathcal{D}) = \sum_{n=1}^{N} \mathcal{L}(\mathcal{W}, \mathbf{x}_n, y_n)$$

- If we partition the training data into mini-batches $\{B_1, B_2, \ldots\}$ and each with size *B (e.g., 100)*

$$\mathcal{L}(\mathcal{W}, \mathcal{D}) = \sum_{u=1}^{N/B} \frac{B}{N} \sum_{n \in \mathcal{B}_u} \frac{N}{B} \mathcal{L}(\mathcal{W}, \mathbf{x}_n, y_n)$$

$$= \mathbb{E}_{p(u)} \big[\frac{N}{B} \sum_{n \in \mathcal{B}_u} \mathcal{L}(\mathcal{W}, \mathbf{x}_n, y_n) \big]$$

Distribution: $p(u = j) = \frac{B}{N}$

stochastic gradient: $\sum_{n \in \mathcal{B}_u} \nabla \mathcal{L}(\mathcal{W}, \mathbf{x}_n, y_n)$

For each update we only need to access a small mini-batch. So it largely reduces the cost

Outline

- Neural networks and Back-propagation
- Stochastic optimization
- Bayesian neural networks
- Bayes by Backprop and reparameterization trick
- Auto-encoding variational Bayes

Bayesian neural networks

- Bayesian version of NNs
- We place prior over the weights
- We use different distributions to sample the observed output

$$\mathbf{f}_{\mathcal{W}}(\mathbf{x_0})$$

$$\parallel$$

$$\mathbf{x}_0 \xrightarrow{\mathbf{W}_1} \mathbf{x}_1 \xrightarrow{\mathbf{W}_2} \ldots \mathbf{x}_{L-1} \xrightarrow{\mathbf{W}_L} \mathbf{x}_L$$

$$\mathbf{f}_{\mathcal{W}}(\mathbf{x}_0) = \mathbf{f}_{\text{out}} = \mathbf{W}_L \sigma(\mathbf{W}_{L-1} \sigma(\ldots \sigma(\mathbf{W}_1 \mathbf{x}_0)))$$

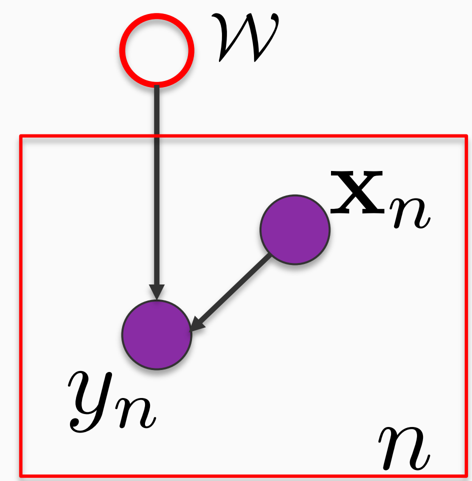
Bayesian neural networks

$$\mathcal{W} = \{\mathbf{W}_1, \dots, \mathbf{W}_L\}$$

- Joint probability

$$\begin{array}{c} y_n \\ \uparrow \\ \mathbf{f}_{\mathcal{W}}(\mathbf{x_n}) \\ \parallel \end{array}$$

$$\mathbf{x}_{n0} \xrightarrow{\mathbf{W}_1} \mathbf{x}_{n1} \xrightarrow{\mathbf{W}_2} \dots \rightarrow \mathbf{x}_{n,L-1} \xrightarrow{\mathbf{W}_L} \mathbf{x}_{nL}$$



$$p(\mathcal{W}, \mathcal{D}) = p(\mathcal{W}) \prod_{n=1}^{N} p\big(y_n | f_{\mathcal{W}}(\mathbf{x}_n)\big)$$

Bayesian neural networks

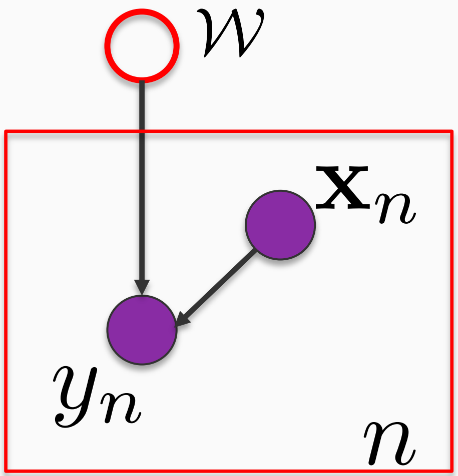
$$\mathcal{W} = \{\mathbf{W}_1, \dots, \mathbf{W}_L\}$$

- Joint probability

$$y_n$$

$$\mathbf{f}_{\mathcal{W}}(\mathbf{x_n})$$

$$\mathbf{x}_{n0} \xrightarrow{\mathbf{W}_1} \mathbf{x}_{n1} \xrightarrow{\mathbf{W}_2} \dots \to \mathbf{x}_{n,L-1} \xrightarrow{\mathbf{W}_L} \mathbf{x}_{nL}$$



$$p(\mathcal{W}, \mathcal{D}) = p(\mathcal{W}) \prod_{n=1}^{N} p\big(y_n | f_{\mathcal{W}}(\mathbf{x}_n)\big)$$

Example of weight priors

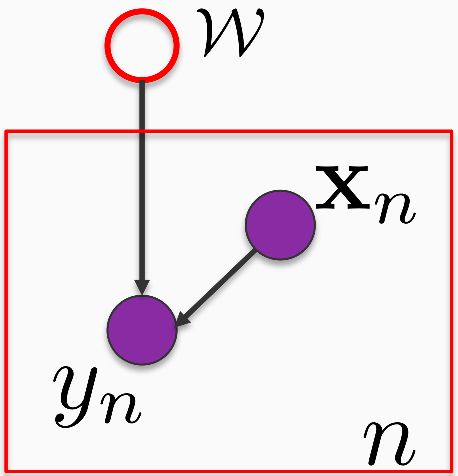
Individual Gaussian $\quad p(\mathcal{W}) = \prod_{w \in \mathcal{W}} \mathcal{N}(w|0, 1)$

Spike and slab: $\quad p(\mathcal{W}) = \prod_{w \in \mathcal{W}} \pi \mathcal{N}(w|0, \sigma_1^2) + (1 - \pi)\mathcal{N}(w|0, \sigma_2^2)$ Encourage sparsity

e.g., $\pi = 0.5, \sigma_1^2 = 1, \sigma_2^2 = 1e-3$

Bayesian neural networks

$$\mathcal{W} = \{\mathbf{W}_1, \ldots, \mathbf{W}_L\}$$



- Joint probability

$$y_n \uparrow \mathbf{f}_{\mathcal{W}}(\mathbf{x_n}) \parallel$$

$$\mathbf{x}_{n0} \xrightarrow{\mathbf{W}_1} \mathbf{x}_{n1} \xrightarrow{\mathbf{W}_2} \ldots \rightarrow \mathbf{x}_{n,L-1} \xrightarrow{\mathbf{W}_L} \mathbf{x}_{nL}$$

$$p(\mathcal{W}, \mathcal{D}) = p(\mathcal{W}) \prod_{n=1}^{N} p\big(y_n | f_{\mathcal{W}}(\mathbf{x}_n)\big)$$

Example of likelihood

Gaussian: $p\big(y_n | f_{\mathcal{W}}(\mathbf{x}_n)\big) = \mathcal{N}(y_n | f_{\mathcal{W}}(\mathbf{x}_n), \sigma^2)$

Bernoulli: $p\big(y_n | f_{\mathcal{W}}(\mathbf{x}_n)\big) = \mathrm{Bern}\big(y_n | 1/\big(1 + \exp\big(-f_{\mathcal{W}}(\mathbf{x}_n)\big)\big)\big)$

Categorical: $p\big(\mathbf{y}_n | \mathbf{f}_{\mathcal{W}}(\mathbf{x}_n)\big) = \prod_k \Big(\frac{\exp([\mathbf{f}_{\mathcal{W}}(\mathbf{x}_n)]_k)}{\sum_j \exp([\mathbf{f}_{\mathcal{W}}(\mathbf{x}_n)]_j)}\Big)^{\mathbb{1}(y_{nk}=1)}$ softmax

Inference Goal of BNNs

- Estimate the posterior distribution of NN weights

$$p(\mathcal{W}|\mathcal{D})$$

- Estimate the predictive distribution

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p\big(y^*|f_{\mathcal{W}}(\mathbf{x}^*)\big)p(\mathcal{W}|\mathcal{D})\mathrm{d}\mathcal{W}$$

Outline

- Neural networks and Back-propagation
- Stochastic optimization
- Bayesian neural networks
- Bayes by Backprop and reparameterization trick
- Auto-encoding variational Bayes

Bayes by Back Propagation

- The golden-standard for BNN inference is HMC. However, it is often too slow to be practical.
- We want to use variational inference, how?

Bayes by Back Propagation

- We want to use variational inference, how?

Introduce variational posterior and construct variational evidence lower bound!

Estimate a free parameter

We choose fully factorized Gaussian

$$q(\mathcal{W}) = \prod_i q(w_i) = \prod_i \mathcal{N}\big(w_i | \mu_i, \log(1 + \exp(\rho_i))\big)$$

$$\log(p(\mathcal{D})) \geq \mathcal{L}(\boldsymbol{\theta}) = \int q(\mathcal{W}) \log \frac{p(\mathcal{W})p(\mathcal{D}|\mathcal{W})}{q(\mathcal{W})} \mathrm{d}\mathcal{W} \qquad \boldsymbol{\theta} = \{(\mu_i, \rho_i)\}$$

$$= \sum_i \mathbb{E}_{q(w_i)}[\log p(w_i)] + \sum_{n=1}^{N} \mathbb{E}_{q(\mathcal{W})}[\log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n))] + \sum_i H\big(q(w_i)\big)$$

Bayes by Back Propagation

$$q(\mathcal{W}) = \prod_i q(w_i) = \prod_i \mathcal{N}\big(w_i | \mu_i, \log(1+\exp(\rho_i))\big)$$

$$\log(p(\mathcal{D})) \geq \mathcal{L}(\boldsymbol{\theta}) = \int q(\mathcal{W}) \log \frac{p(\mathcal{W})p(\mathcal{D}|\mathcal{W})}{q(\mathcal{W})} \mathrm{d}\mathcal{W}$$

$$= \sum_i \mathbb{E}_{q(w_i)}[\log p(w_i)] + \underbrace{\sum_{n=1}^{N} \mathbb{E}_{q(\mathcal{W})}[\log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n))]}_{} + \sum_i H\big(q(w_i)\big)$$

Analytical for Gaussian prior

Totally intractable, Why?

Gaussian entropy

$\log\big(\log(1+\exp(\rho_i))2\pi e\big)$

How to maximize $\mathcal{L}(\boldsymbol{\theta})$?

Bayes by Back Propagation

- Stochastic optimization
- The key question: How to compute the stochastic gradient for each

$$\mathbb{E}_{q(\mathcal{W})}[\log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))]$$

Can we use current parameters to sample $\mathcal{W}$, plugging into log and calculate the gradient?

$$\widehat{\mathcal{W}} \sim q(\mathcal{W}|\boldsymbol{\theta}) \qquad \boldsymbol{\theta} = \{(\mu_i, \rho_i)\}$$

$$\Downarrow$$

$$\nabla \log p(y_n|f_{\widehat{\mathcal{W}}}(\mathbf{x}_n))$$

Totally wrong!

Bayes by Back Propagation

- The reason is the distribution contains unknown parameters, and so the expectation and derivative are not interchangeable!

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n))] \neq \mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n))]$$

$$\parallel$$

$$\mathbf{0}$$

Why?

$$\nabla_{\boldsymbol{\theta}} \int q(\mathcal{W}|\boldsymbol{\theta}) \log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n)) \mathrm{d}\mathcal{W}$$

Bayes by Back Propagation

- The reason is the distribution contains unknown parameters, and so the expectation and derivative are not interchangeable!

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n))] \neq \mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}} \log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n))]$$

$$\nabla_{\boldsymbol{\theta}} \int q(\mathcal{W}|\boldsymbol{\theta}) \log p(y_n | f_{\mathcal{W}}(\mathbf{x}_n)) \mathrm{d}\mathcal{W}$$

$$\mathbf{0}$$

Why?

Because the log likelihood itself does not include variational parameters!

Reparameterization trick

- The solution is to get rid of the unknown parameters in the distribution under which we compute the expectation. How?

$$q(\mathcal{W}) = \prod_i q(w_i) = \prod_i \mathcal{N}\big(w_i | \mu_i, \log(1 + \exp(\rho_i))\big)$$

$$w_i = \mu_i + \epsilon_i \sqrt{\log(1 + \exp(\rho_i))} \qquad \epsilon_i \sim \mathcal{N}(0, 1)$$

$$\Downarrow$$

$$\mathrm{vec}(\mathcal{W}) = \boldsymbol{\mu} + \mathrm{diag}\big(\sqrt{\log(1 + \exp(\boldsymbol{\rho}))}\big) \cdot \boldsymbol{\epsilon} \quad \Rightarrow \quad \mathcal{W} = T(\boldsymbol{\theta}, \boldsymbol{\epsilon}), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Reparameterized Gaussian sample

Reparameterization trick

$$\mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$$

$$\Downarrow$$

$$\int q(\mathcal{W}|\boldsymbol{\theta}) \log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))\mathrm{d}\mathcal{W} = \int p(\boldsymbol{\epsilon}) \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))\mathrm{d}\boldsymbol{\epsilon}$$

$$\Downarrow$$

$$\nabla_{\boldsymbol{\theta}} \int q(\mathcal{W}|\boldsymbol{\theta}) \log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))\mathrm{d}\mathcal{W} = \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{\epsilon}) \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))\mathrm{d}\boldsymbol{\epsilon}$$

$$\|$$

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))]$$

$$= \int \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\epsilon}) \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))\mathrm{d}\boldsymbol{\epsilon}$$

$$= \int p(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\theta}} \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))\mathrm{d}\boldsymbol{\epsilon}$$

$$= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\theta}} \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$$

Reparameterization trick

$$\mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))] = \mathbb{E}_{p(\boldsymbol{\epsilon})}[\log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$$

$$\Downarrow$$

$$\int q(\mathcal{W}|\boldsymbol{\theta}) \log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n)) \mathrm{d}\mathcal{W} = \int p(\boldsymbol{\epsilon}) \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n)) \mathrm{d}\boldsymbol{\epsilon}$$

$$\Downarrow$$

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \int q(\mathcal{W}|\boldsymbol{\theta}) \log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n)) \mathrm{d}\mathcal{W} &= \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{\epsilon}) \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n)) \mathrm{d}\boldsymbol{\epsilon} \\
&= \int \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\epsilon}) \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n)) \mathrm{d}\boldsymbol{\epsilon} \\
&= \int p(\boldsymbol{\epsilon}) \nabla_{\boldsymbol{\theta}} \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n)) \mathrm{d}\boldsymbol{\epsilon} \\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\boxed{\nabla_{\boldsymbol{\theta}} \log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))}]
\end{aligned}$$

$$\| \quad \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q(\mathcal{W}|\boldsymbol{\theta})}[\log p(y_n|f_{\mathcal{W}}(\mathbf{x}_n))]$$

Stochastic gradient ascent!

Look back at ELBO

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_i \mathbb{E}_{q(w_i)}[\log p(w_i)] + \sum_i H\big(q(w_i)\big)$$

$$+ \sum_{u=1}^{N/B} \frac{B}{N} \sum_{n \in \mathcal{B}_u} \frac{N}{B} \mathbb{E}_{p(\boldsymbol{\epsilon})}[\log p(y_n | f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$$

$$\mathbb{E}_{p(u)} \mathbb{E}_{p(\boldsymbol{\epsilon})} \sum_{n \in \mathcal{B}_u} \frac{N}{B} [\log p(y_n | f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$$

Constant distribution

$$\alpha(\boldsymbol{\theta}) = \sum_i \mathbb{E}_{q(w_i)}[\log p(w_i)] + \sum_i H\big(q(w_i)\big)$$

Bayes by Back Propagation

- 1. Initialize $\boldsymbol{\theta}$ randomly
- 2. For t = 1.. T
  - Sample *u* from *p(u)*, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - Calculate stochastic gradient $\nabla_{\boldsymbol{\theta}}[\alpha(\boldsymbol{\theta})] + \frac{N}{B}\sum_{n \in \mathcal{B}_u} \nabla_{\boldsymbol{\theta}}[\log p(y_n | f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$
  - Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma_t \cdot \left(\nabla_{\boldsymbol{\theta}}[\alpha(\boldsymbol{\theta})] + \frac{N}{B}\sum_{n \in \mathcal{B}_u} \nabla_{\boldsymbol{\theta}}[\log p(y_n | f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]\right)$
- 3. Return $q(\mathcal{W}|\boldsymbol{\theta}) = \prod_i \mathcal{N}\big(w_i | \mu_i, \log(1 + \exp(\rho_i))\big)$

$$\alpha(\boldsymbol{\theta}) = \sum_i \mathbb{E}_{q(w_i)}[\log p(w_i)] + \sum_i H\big(q(w_i)\big)$$

Bayes by Back Propagation

- 1. Initialize $\boldsymbol{\theta}$ randomly
- 2. For t = 1.. T
  - Sample *u* from *p(u)*, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - Calculate stochastic gradient $\nabla_{\boldsymbol{\theta}}[\alpha(\boldsymbol{\theta})] + \frac{N}{B}\sum_{n\in\mathcal{B}_u} \nabla_{\boldsymbol{\theta}}[\log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]$
  - Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma_t \cdot \left(\nabla_{\boldsymbol{\theta}}[\alpha(\boldsymbol{\theta})] + \frac{N}{B}\sum_{n\in\mathcal{B}_u} \nabla_{\boldsymbol{\theta}}[\log p(y_n|f_{T(\boldsymbol{\theta},\boldsymbol{\epsilon})}(\mathbf{x}_n))]\right)$

output of the NN, so it needs BP!

- 3. Return $q(\mathcal{W}|\boldsymbol{\theta}) = \prod_i \mathcal{N}\big(w_i|\mu_i, \log(1+\exp(\rho_i))\big)$

Predictive distribution

$$p(y^*|\mathbf{x}^*,\mathcal{D}) = \int p\big(y^*|f_{\mathcal{W}}(\mathbf{x}^*)\big)p(\mathcal{W}|\mathcal{D})\mathrm{d}\mathcal{W}$$
$$\approx \int p\big(y^*|f_{\mathcal{W}}(\mathbf{x}^*)\big)q(\mathcal{W}|\boldsymbol{\theta})\mathrm{d}\mathcal{W}$$

Still intractable, but we can use Monte-Carlo approximation

$$\approx \frac{1}{M}\sum_{j=1}^{m} p\big(y^*|f_{\mathcal{W}_j}(\mathbf{x}^*)\big) \qquad \mathcal{W}_j \sim q(\mathcal{W}|\boldsymbol{\theta})$$

We can also generate samples of y^* to obtain an empirical (or histogram) distribution

Performance

Table 1. Classification Error Rates on MNIST. $\star$ indicates result used an ensemble of 5 networks.

Method	**# Units/Layer**	**# Weights**	**Test Error**
SGD, no regularisation (Simard et al., 2003)	800	1.3m	1.6%
SGD, dropout (Hinton et al., 2012)			$\approx$ 1.3%
SGD, dropconnect (Wan et al., 2013)	800	1.3m	**1.2%***
SGD	400	500k	1.83%
	800	1.3m	1.84%
	1200	2.4m	1.88%
SGD, dropout	400	500k	1.51%
	800	1.3m	1.33%
	1200	2.4m	1.36%
Bayes by Backprop, Gaussian	400	500k	1.82%
	800	1.3m	1.99%
	1200	2.4m	2.04%
Bayes by Backprop, Scale mixture	400	500k	1.36%
	800	1.3m	1.34%
	1200	2.4m	**1.32%**

Performance

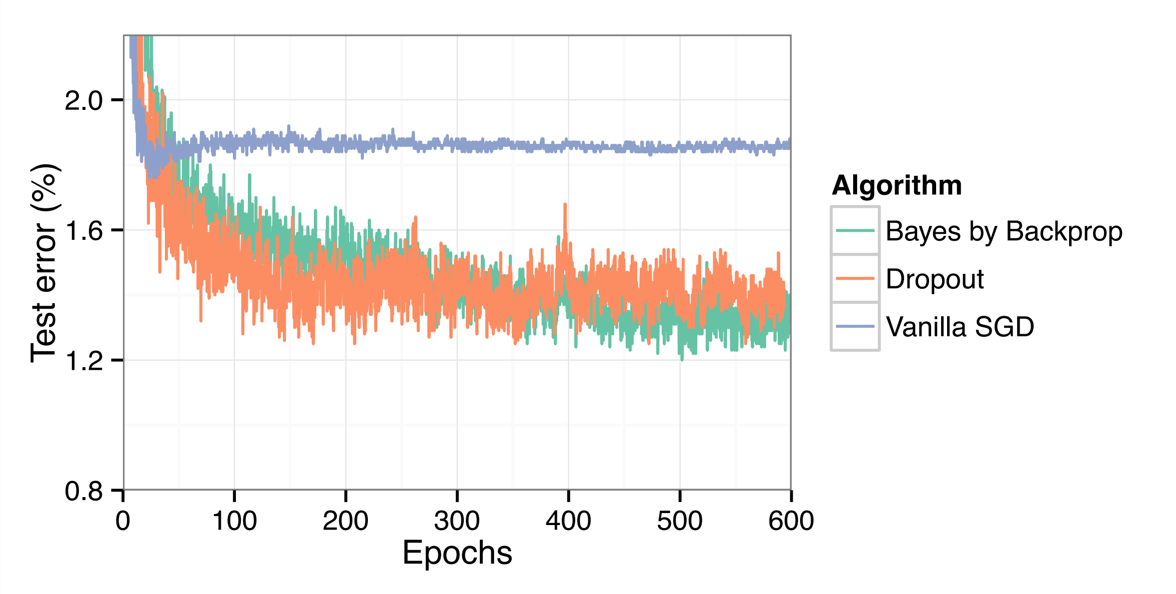


Figure 2. Test error on MNIST as training progresses.

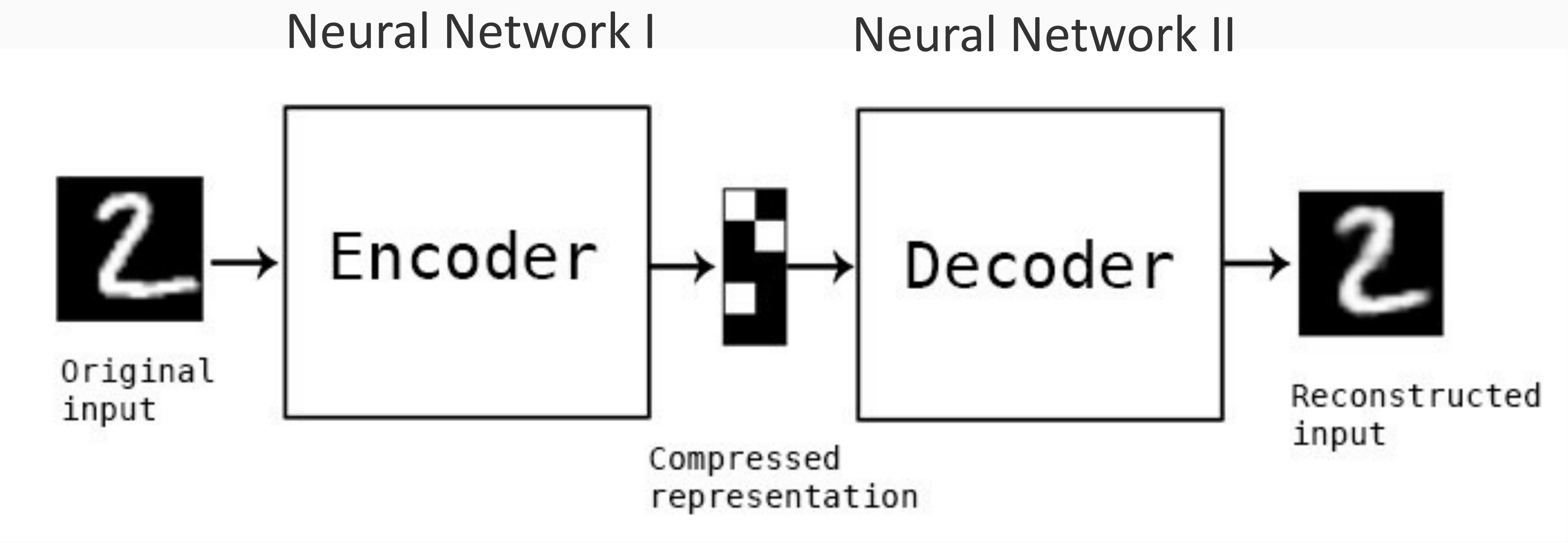
BBB: Summary

- State of the art NN inference, very popular
- The same scalability to SGD, but it can estimate posteriors!
- Core idea : variational inference + reparameterization trick
- This is also the foundation of nearly all the modern Bayesian NN training.

Outline

- Neural networks and Back-propagation
- Stochastic optimization
- Bayesian neural networks
- Bayes by Backprop and reparameterization trick
- Auto-encoding variational Bayes

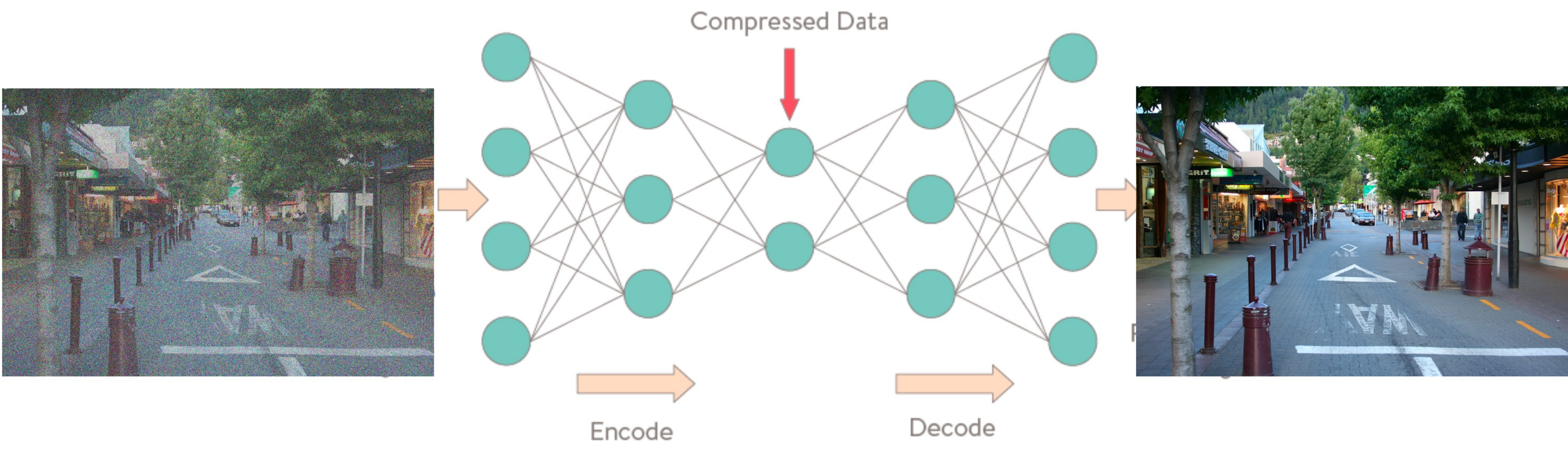
Auto-Encoder: Dimension Reduction



[Provided by Will Badr]

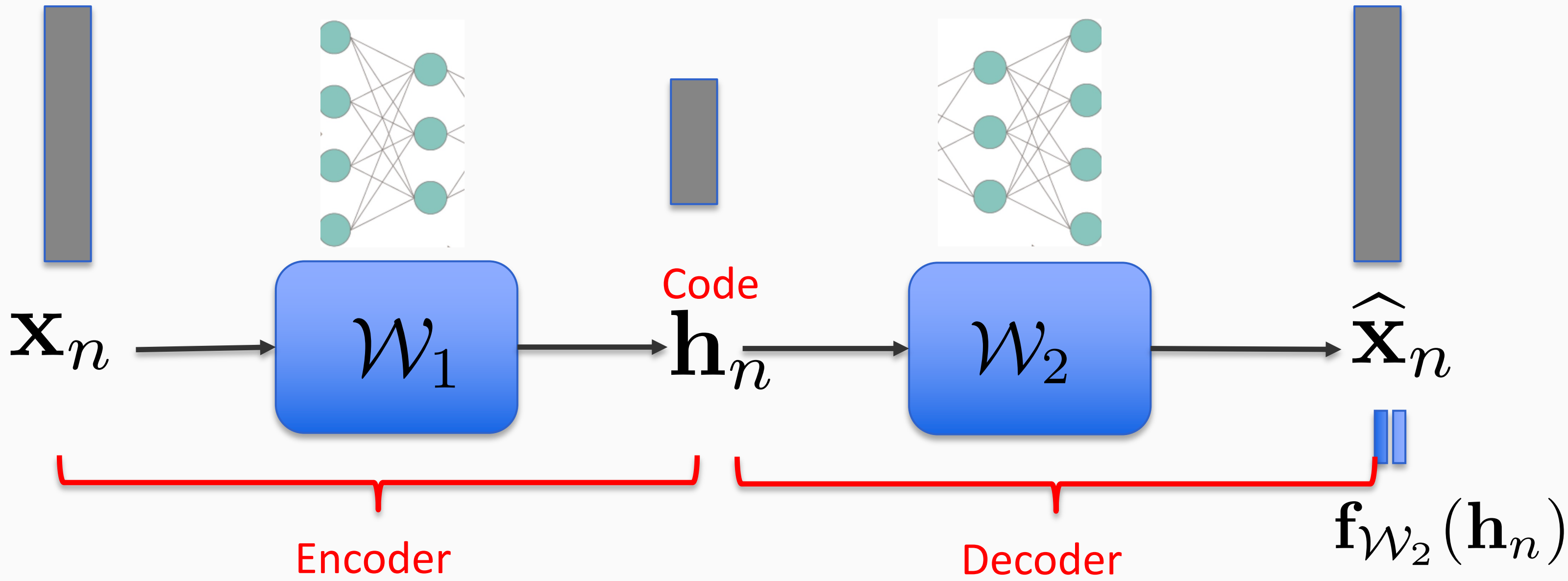
Auto-Encoder

Dimension reduction is very important: compression, denoise, …



[Provided by Will Badr]

Vanilla Auto-Encoder

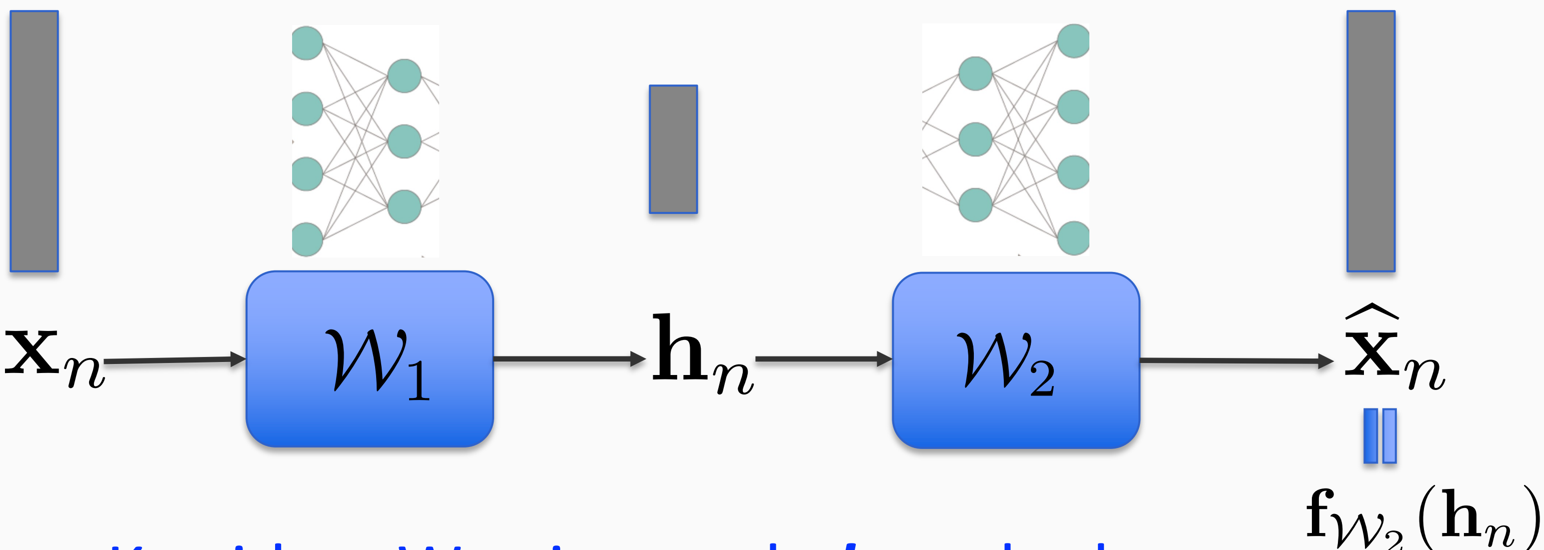


Given data $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$

Loss: $$\sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_{\mathcal{W}_1}(\mathbf{x}_n))\|^2$$

Variational Auto-Encoder

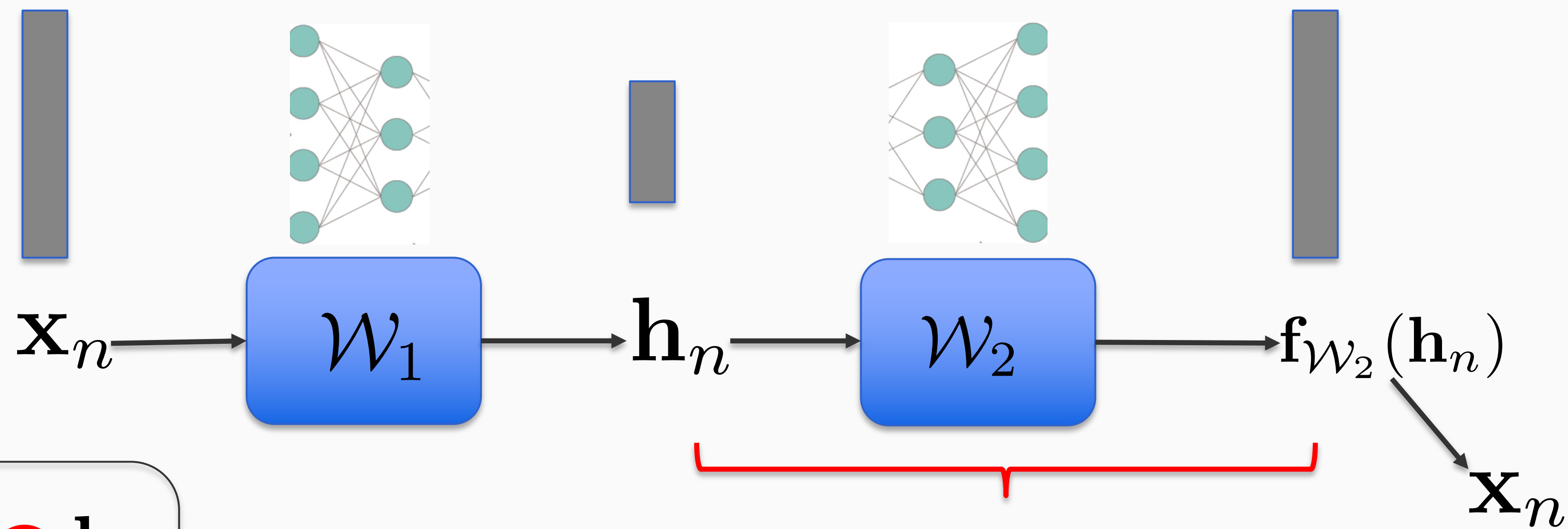
Data: $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$

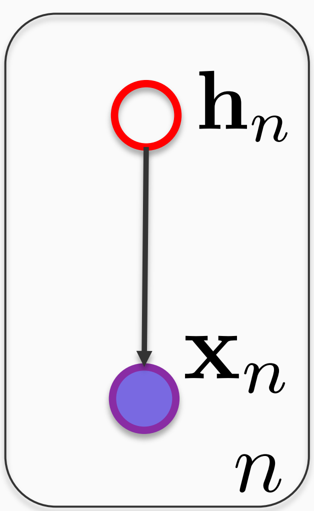


Key idea: We view code ***h*** as the latent random variables. We want to estimate the posterior distribution of ***h***; However, the NN weights are considered as hyper-parameters rather than RVs.

Variational Auto-Encoder

Data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



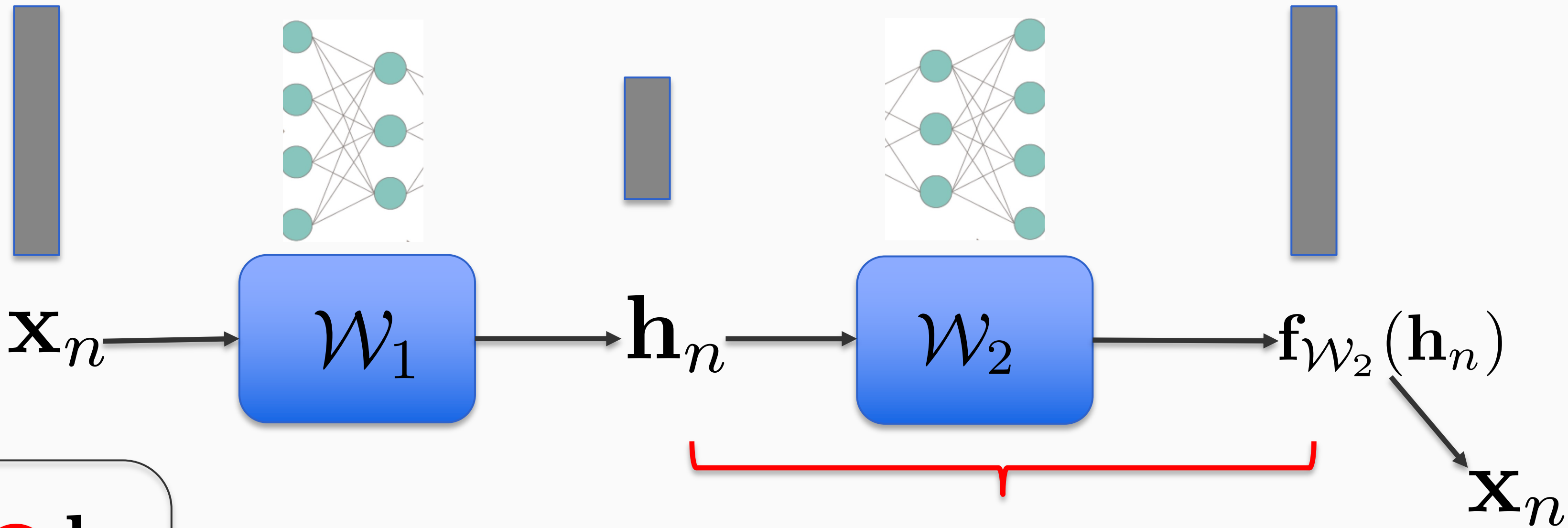


The model is only the decoder part

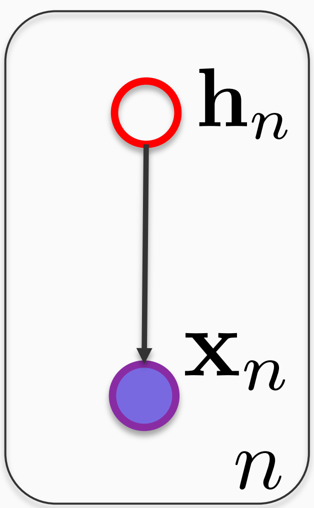
$$p(\mathbf{H}, \mathcal{D}) = \prod_n p(\mathbf{h}_n) p(\mathbf{x}_n | \mathbf{h}_n)$$

Variational Auto-Encoder

Data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



Probabilistic Decoder

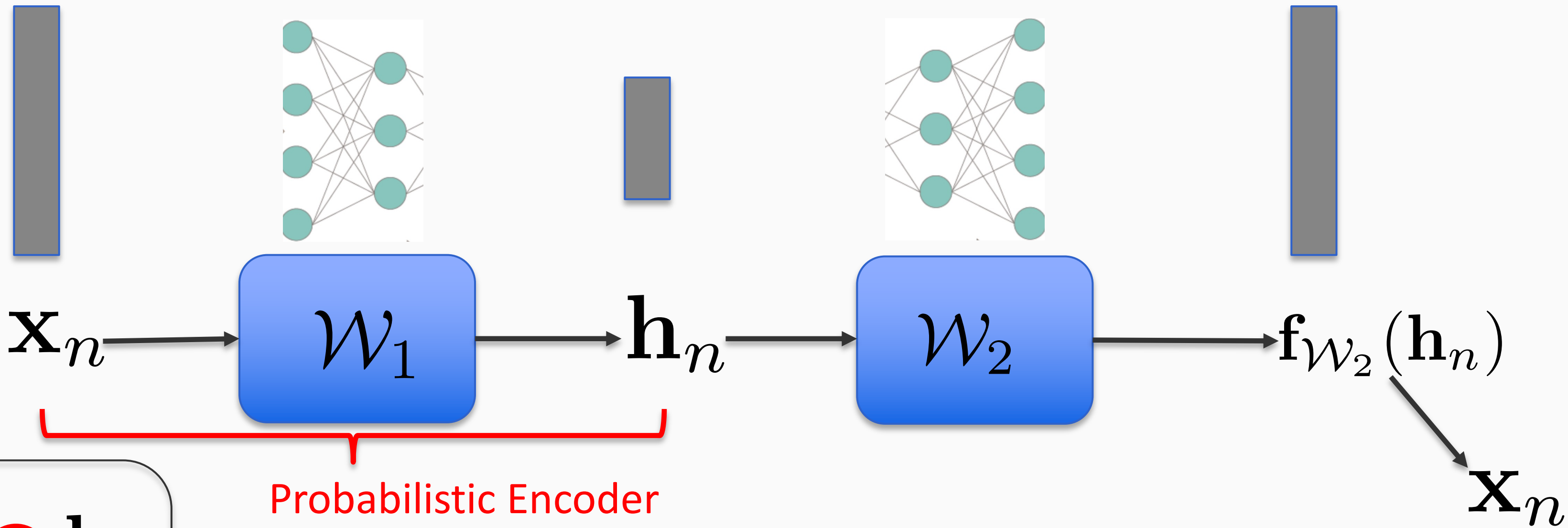


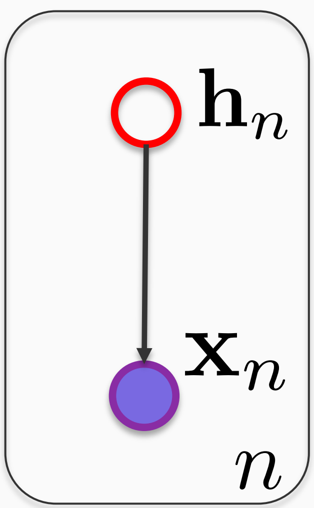
$$p(\mathbf{H}, \mathcal{D}) = \prod_n p(\mathbf{h}_n) p(\mathbf{x}_n | \mathbf{h}_n)$$

$$\mathcal{N}(\mathbf{h}_n | \mathbf{0}, \mathbf{I}) \qquad p\big(\mathbf{x}_n | \mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)\big)$$

Variational Auto-Encoder

Data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$



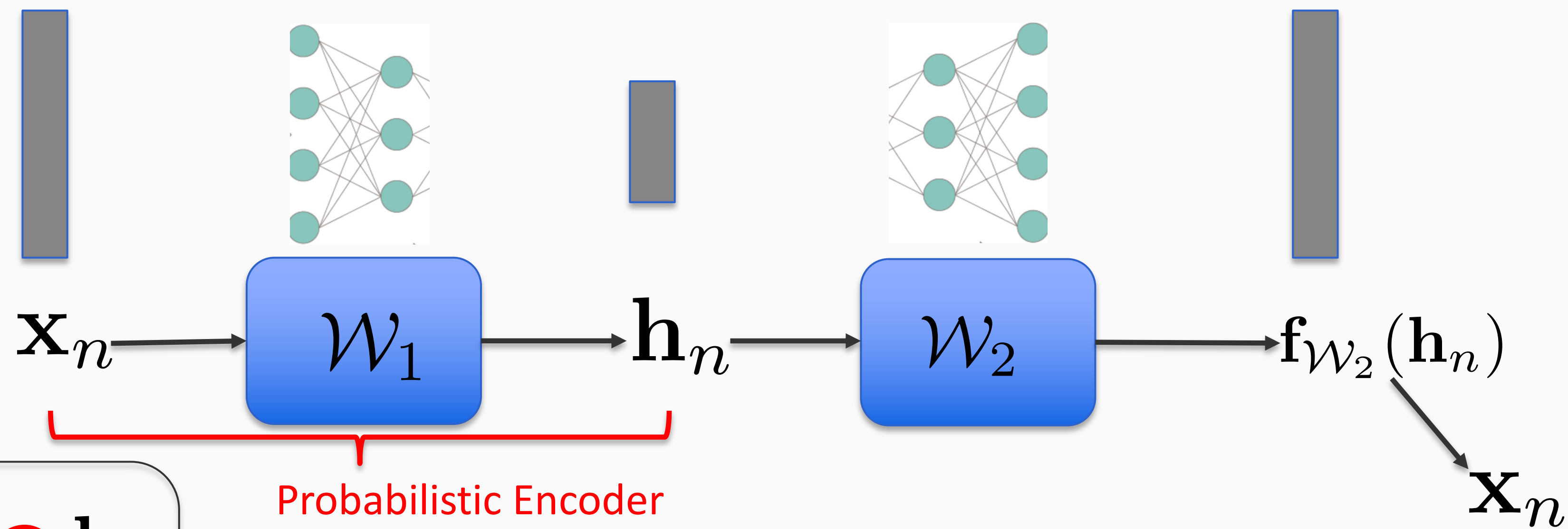


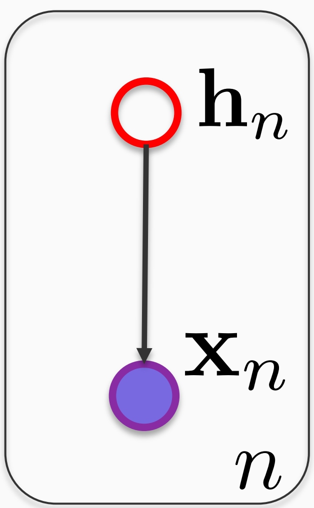
Encoder is defined as the variational posterior distribution of ***h***

$$q(\mathbf{H}) = \prod_n q_{\mathcal{W}_1}(\mathbf{h}_n | \mathbf{x}_n)$$

Variational Auto-Encoder

Data: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$





Encoder is defined as the variational posterior distribution of ***h*$_n$**

$$q(\mathbf{H}) = \prod_n q_{\mathcal{W}_1}(\mathbf{h}_n | \mathbf{x}_n)$$

We use NN output to parameterize the variational posterior, namely, the encoder!

Variational Auto-Encoder: Inference

- Maximize the variational ELBO

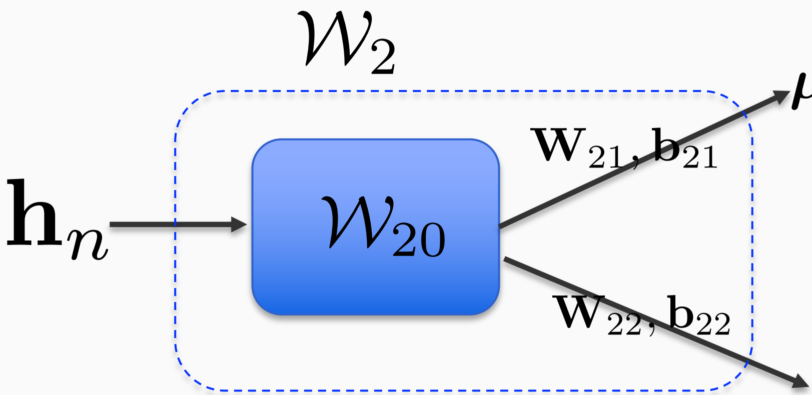
$$
\begin{aligned}
\mathcal{L} &= \int q(\mathbf{H}) \log \frac{p(\mathbf{H})p(\mathbf{H}, \mathcal{D})}{q(\mathbf{H})} \mathrm{d}\mathbf{H} \\
&= \sum_{n=1}^{N} \int q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n) \log \frac{p(\mathbf{h}_n)p\big(\mathbf{x}_n|\mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)\big)}{q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n)} \mathrm{d}\mathbf{h}_n \\
&= \sum_{n=1}^{N} \mathbb{E}_{q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n)} \Big[\log \frac{p(\mathbf{h}_n)p\big(\mathbf{x}_n|\mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)\big)}{q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n)} \Big]
\end{aligned}
$$

ELBO is obviously intractable, why?

Use reparameterization trick + stochastic optimization (on mini-batches)!

Concrete example

- Likelihood for continuous output

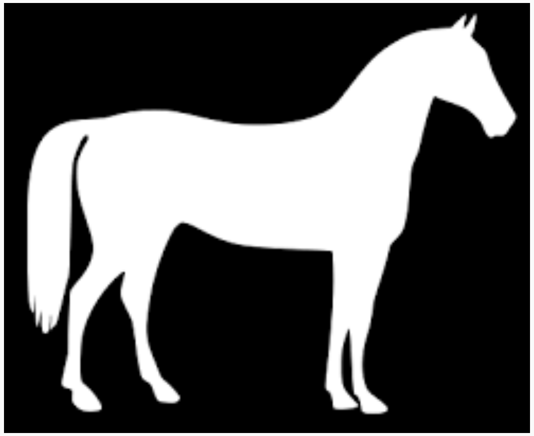


$$\left.\begin{aligned} \boldsymbol{\mu}_n &= \mathbf{W}_{21} \cdot \underbrace{\mathbf{g}_{\mathcal{W}_{20}}(\mathbf{h}_n)}_{\text{The 2}^{\text{nd}}\text{ last layer}} + \mathbf{b}_{21} \\ \boldsymbol{\rho}_n &= \mathbf{W}_{22} \cdot \mathbf{g}_{\mathcal{W}_{20}}(\mathbf{h}_n) + \mathbf{b}_{22} \end{aligned}\right\} \mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n) \rightarrow \mathbf{x}_n$$

$$p(\mathbf{x}_n|\mathbf{h}_n) = p\big(\mathbf{x}_n|\mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)\big) = \boxed{\mathcal{N}\big(\mathbf{x}_n|\boldsymbol{\mu}_n, \mathrm{diag}(\exp(\boldsymbol{\rho}_n))\big)}$$

Gaussian with diagonal covariance

Concrete example



- Likelihood for binary output

$$\mathbf{h}_n \longrightarrow \boxed{\mathcal{W}_2} \longrightarrow \mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n) \longrightarrow \mathbf{x}_n$$

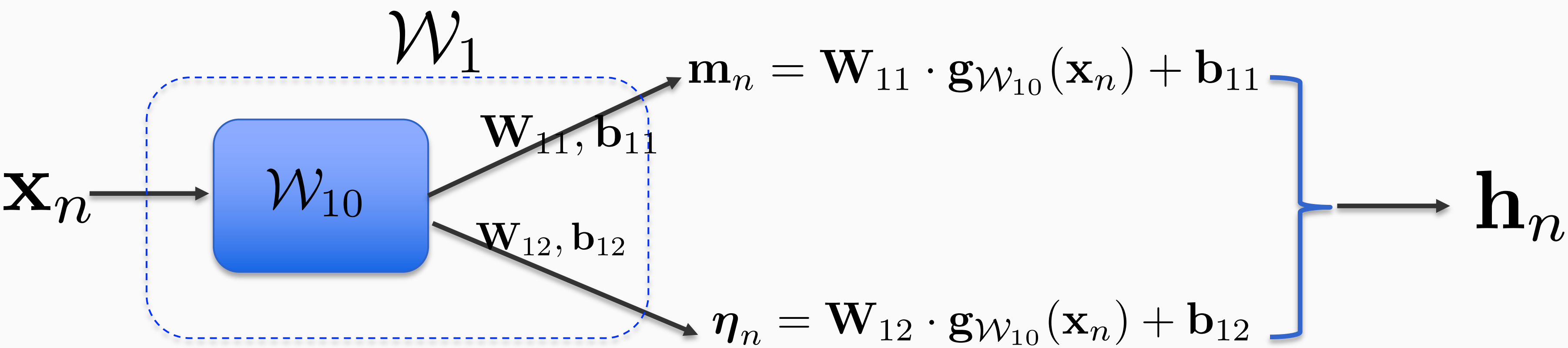
$$p(\mathbf{x}_n|\mathbf{h}_n) = p\big(\mathbf{x}_n|\mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)\big) = \prod_j \boxed{\mathrm{Bern}\big([\mathbf{x}_n]_j|\alpha([\mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)]_j)\big)}$$

Bernoulli likelihood over each element

$$\alpha(t) = 1/(1+\exp(-t))$$

Concrete example

- Gaussian encoder (most commonly used)



$$q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n) = \mathcal{N}\big(\mathbf{h}_n|\mathbf{m}_n, \mathrm{diag}(\exp(\boldsymbol{\eta}_n))\big)$$

$$\mathcal{L} = \sum_{n=1}^{N} \mathbb{E}_{q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n)}\left[\log \frac{p(\mathbf{h}_n)p\big(\mathbf{x}_n|\mathbf{f}_{\mathcal{W}_2}(\mathbf{h}_n)\big)}{q_{\mathcal{W}_1}(\mathbf{h}_n|\mathbf{x}_n)}\right]$$

Very easy to use reparameterization trick!

VAE: summary

- Convert auto-encoder estimation into a probabilistic inference problem
- Trivial application of VI
- State-of-the-art
- Very popluar

What you need to know

- What are Bayesian NNs?
- What are the key idea of BP and stochastic optimization?
- How to conduct variational inference for BNNs?
- What is the reparameterization trick?
- The key idea of Bayes by Backprop, and variational auto-encoder
- You should be able to implement them (with TensorFlow or pyTorch) now!