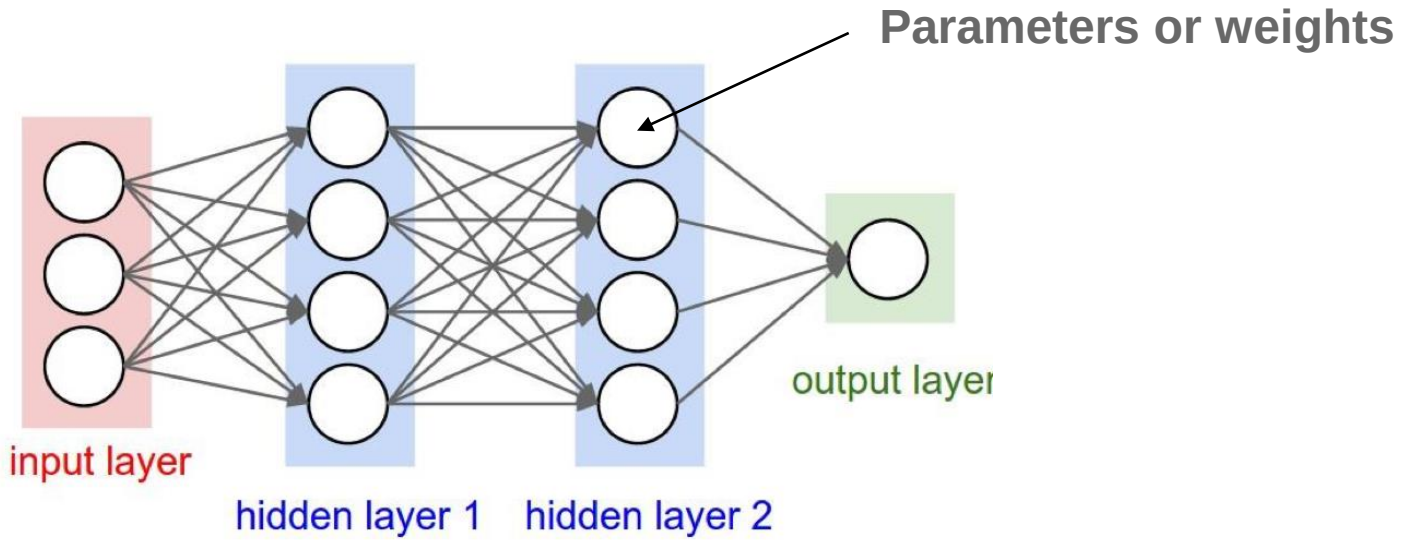
Bayesian Neural Networks

Machine learning for computer vision (IN2357)

Motivation: Neural Networks Revisited

- Neural Networks are **parameterized** functions:



- Main idea is to **learn a composition** of functions or layers.
- Typically, trained with a **maximum likelihood principle** (point estimates) by optimizing some loss function using stochastic gradient descent.
- The result is a set of **most likely parameters** that explains the data.

Motivation: Challenges in Deep Learning

Neural Networks with many layers (deep learning) are useful but:

- **Can you train neural networks with better generalization?**
- **Can you efficiently learn by leveraging prior experiences?**
- **Can you design architectures without trial and error only?**
- **Can you reduce computational complexity?**
- **Can you trust it? Or, represent well-calibrated uncertainty?**

Bayesian Neural Networks brings Bayesian reasoning to deep learning and might be useful for reducing overfitting, uncertainty estimates, etc.

What is Bayesian Neural Networks?

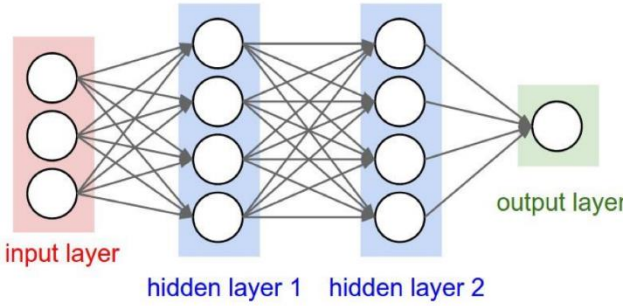
- **Bayesian Reasoning?**

Probability distribution as your belief rather than frequency.

Dealing with uncertainty about your model (e.g. model parameters).

- **When applied to neural networks, parameters have distributions.**

What is Bayesian Neural Networks?



Given that …

- **Data:** $\mathcal{D} = \{(x^n, y^n)\}_{n=1}^N = (X, y)$
- **Parameters** w **are weights of neural networks.**

Then, instead of trying to find a set of most likely parameters:

- **Define prior:** $p(w)$
- **Bayesian learning of posterior:** $p(w|\mathcal{D}) \propto p(y|X, w)p(w)$
- **Prediction:** $p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$

Tables of Contents

Aim: Understanding the basics related to Bayesian Neural Networks

1. **Point Estimates Vs Model Uncertainty**
2. **On Priors of Bayesian Neural Networks.**
3. **On Posteriors of Bayesian Neural Networks.**
4. **On Predictions with Bayesian Neural Networks.**
5. **Bayesian Deep Learning.**

Disclaimer: very active area of research – we cover only the basics.

Tables of Contents

Aim: Understanding the basics related to Bayesian Neural Networks

Point Estimates

Many learning algorithms such as Deep NNs or Support Vector Machines only determine one optimal (or most likely) set of model parameters.

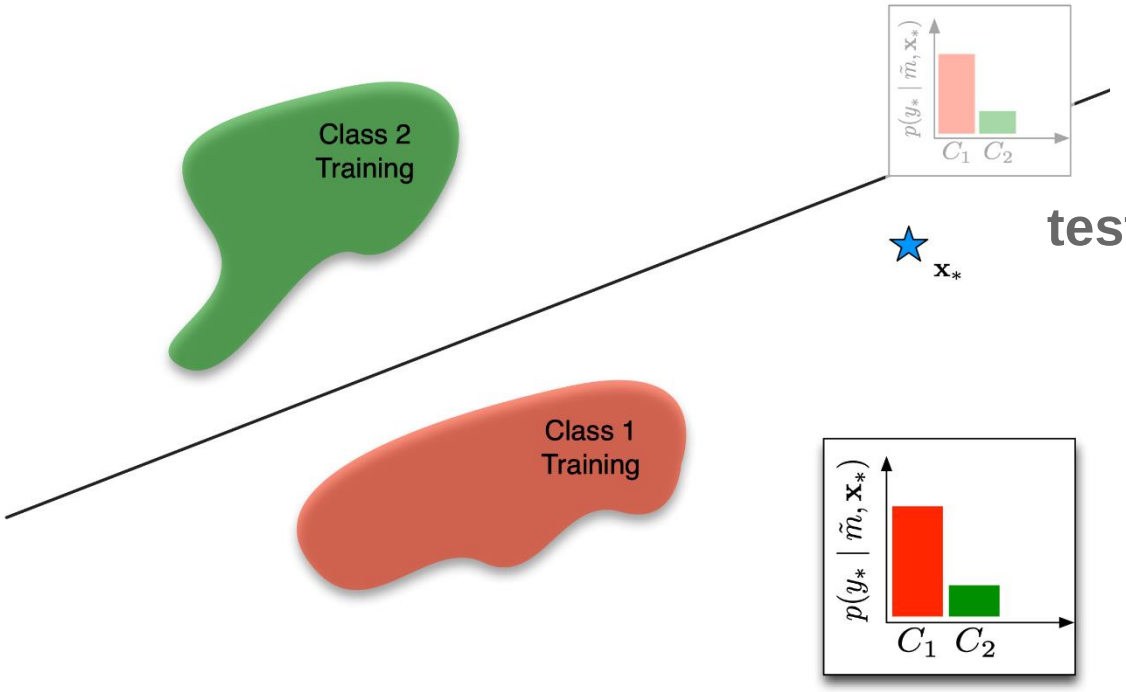
E.g.: $$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$$ **(MLE)**

or: $$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$$ **(MAP)**

Point Estimates: Example

Example with a linear classifier (e.g. a SVM):

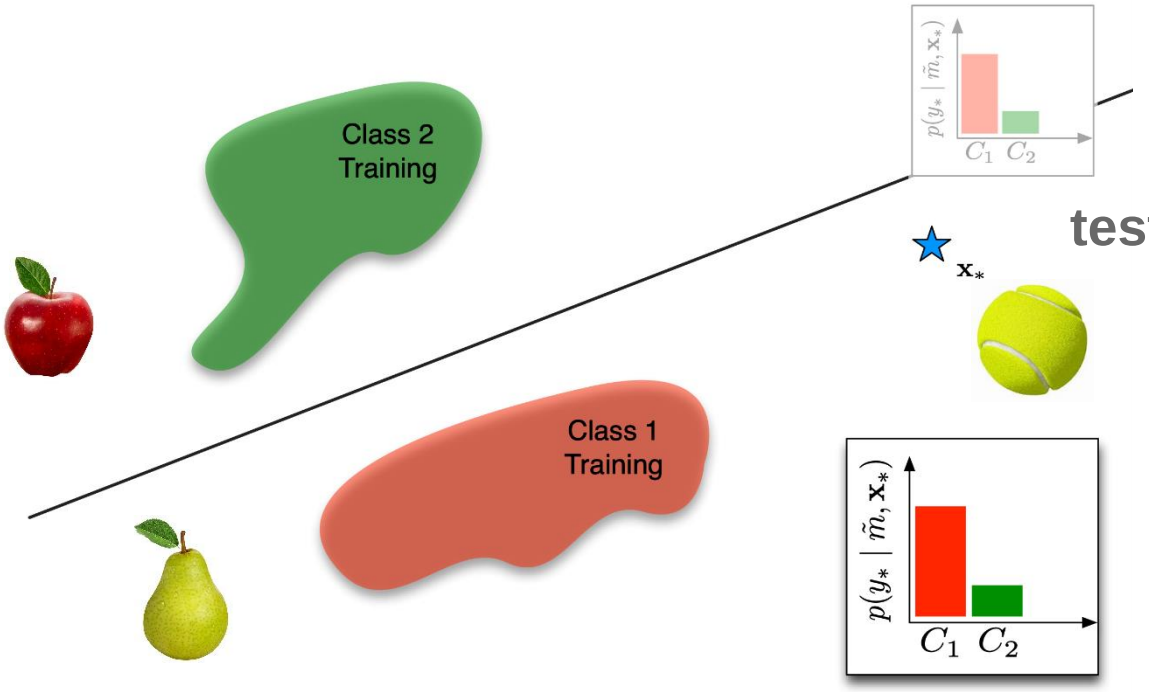
- **Binary classification**
- **Aim: predict the label of a sample that is far from the training data set.**



Point Estimates: Example

Example with a linear classifier (e.g. a SVM):

- **Binary classification**
- **Aim: predict the label of a sample that is far from the training data set.**



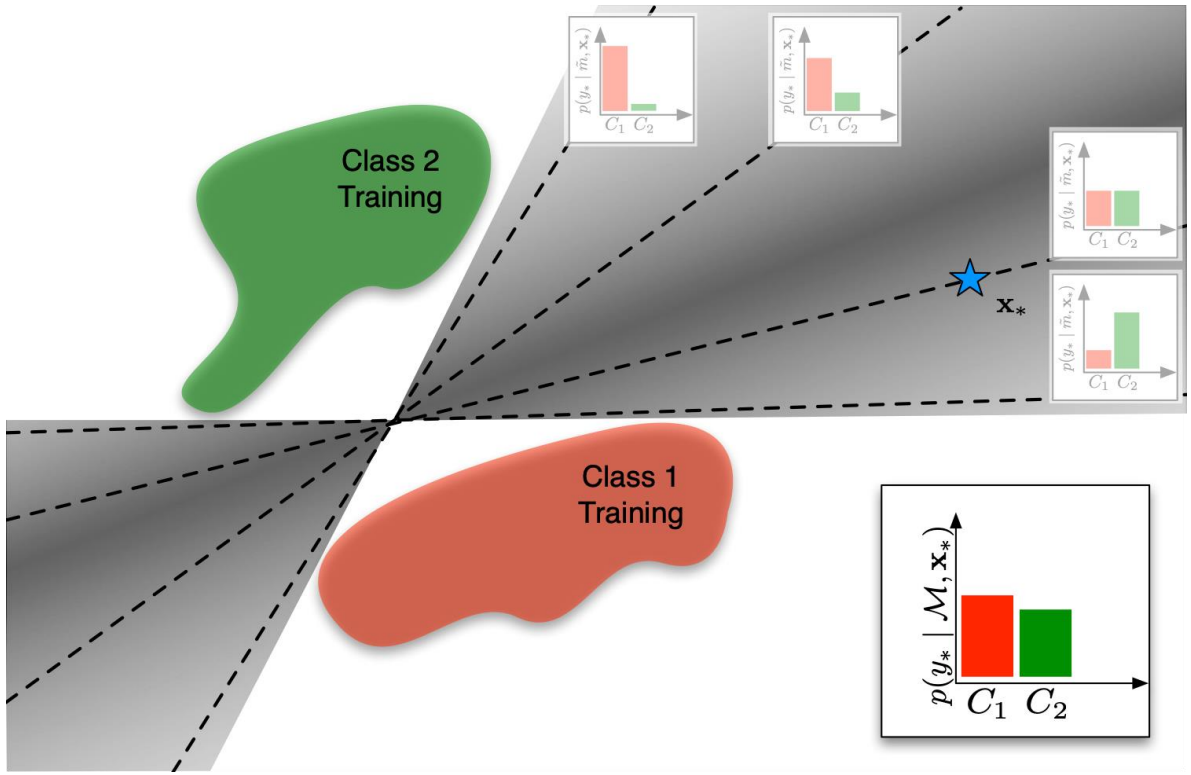
test sample

predictive distribution

Point Estimates Vs Model Uncertainty

The right answer is to say “I do not know.“

predictive distributio



Prediction with a weighted average

Point Estimates Vs Model Uncertainty

Many learning algorithms such as Deep NNs or Support Vector Machines only determine one optimal (or most likely) set of model parameters.

E.g.:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) \qquad \textbf{(MLE)}$$

or:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}, \mathbf{y}) \qquad \textbf{(MAP)}$$

Instead, Bayesian methods consider a distribution over model parameters and take the average:

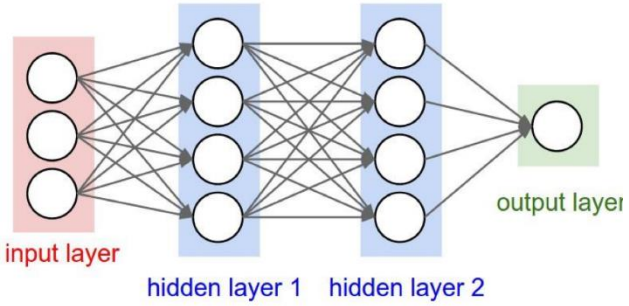
$$p(\mathbf{y}^* \mid X, \mathbf{y}, \mathbf{x}^*) = \int p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) p(\mathbf{w} \mid X, \mathbf{y}) d\mathbf{w}$$

This motivates to take model uncertainty into account, instead of MAP.

Tables of Contents

Aim: Understanding the basics related to Bayesian Neural Networks

Recap on Bayesian Neural Networks



Given that …

- **Data:** $\mathcal{D} = \{(x^n, y^n)\}_{n=1}^N = (X, y)$
- **Parameters w are weights of neural networks.**

Then, Instead of trying to find a most likely parameters:

- **Define prior:** $p(w)$
- **Bayesian learning of posterior:** $p(w|\mathcal{D}) \propto p(y|X, w)p(w)$
- **Prediction:** $p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$

On Priors: Why important?

- **PAC (Probably Approximately Correct) Bayes**:

$$R(Q) \leq \hat{R}(Q) + \sqrt{\frac{KL(Q||P) + \ln\left(\frac{m+1}{\delta}\right)}{2m}}$$

- **Q** and **P** can be posterior and prior distributions.
- $R(Q)$ a true risk and $\hat{R}(Q)$ an empirical risk for sample size m.
- $KL(Q||P)$ defines KL divergence (similarity) between two distributions.
- For any $\delta > 0$, PAC Bayes shows a bound to true risk.

On Priors: Why important?

- **PAC (Probably Approximately Correct) Bayes**:

$$R(Q) \leq \widehat{R}(Q) + \sqrt{\frac{KL(Q||P) + \ln\left(\frac{m+1}{\delta}\right)}{2m}}$$

- **Mostly frequentist paradigm, but can be viewed as Bayesian too.**
- **Set P to be the prior distribution, and Q to be the posterior distribution.**
- **Generalization of a learner (how close you are to a true risk), depends on how well you fit the data $\widehat{R}(Q)$ and how you are constrained to $KL(Q||P)$.**

On Priors: some thoughts

- Specifying the prior distribution is required for Bayesian methods.
- If you have enough data, likelihood may dominate the posterior and so, a reasonable prior could work in practice.
- If you do not have enough data, mis-specifying the prior can lead to mis-specified posterior. But, if the prior is good, we might get better generalizable models.
- For Deep Neural Networks, specifying a good prior is unintuitive and largely a not solved problem.
- Still, Bayesian's averaging way of predictions can be beneficial over merely using point estimates.

On Priors: A Good Default?

- So, a standard procedure is to use a normal prior:

$$p(w) \sim \mathcal{N}(0, \sigma I)$$

Think of it as ranges

- In the context of deep learning, this implies:

$$w^* = \text{argmax}\, p(w|X, Y)$$

MAP estimates

$$\mathbf{w}^* = \text{argmax}\, \mathbf{p}(\mathbf{Y}|\mathbf{X}, \mathbf{w})\mathbf{p}(\mathbf{w})$$

$$w^* = \text{argmin}\, \hat{R}(w) + \text{reg}(w)$$

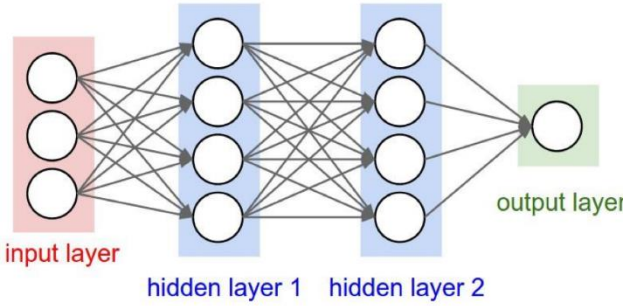
Log likelihood

- Equivalent to a weighted l2 regularization on neural network weights when training a point estimate.

Tables of Contents

Aim: Understanding the basics related to Bayesian Neural Networks

Recap on Bayesian Neural Networks



Given that …

- **Data:** $\mathcal{D} = \{(x^n, y^n)\}_{n=1}^N = (X, y)$
- **Parameters w are weights of neural networks.**

Then, Instead of trying to find a most likely parameters:

- **Define prior:** $p(w)$
- **Bayesian learning of posterior:** $p(w|\mathcal{D}) \propto p(y|X, w)p(w)$
- **Prediction:** $p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$

On Posteriors: Bayes Rule Revisited

$$p(hypothesis|data) = \frac{p(data|hypothesis)p(hypothesis)}{\sum p(data|hypothesis)p(hypothesis)}$$

- Bayes rule tells us how to do inference about **hypothesis** (uncertain quantities) from **data** (measured quantities).
- Learning and predictions can be seen as forms of inference.

On Posteriors: The problem of integrals.

- **Posterior Distribution:**

$$p(\mathbf{w}|\mathbf{X},\mathbf{Y}) = \frac{p(Y|X,w)p(w)}{p(Y|X)}$$

- **Evidence Term:**

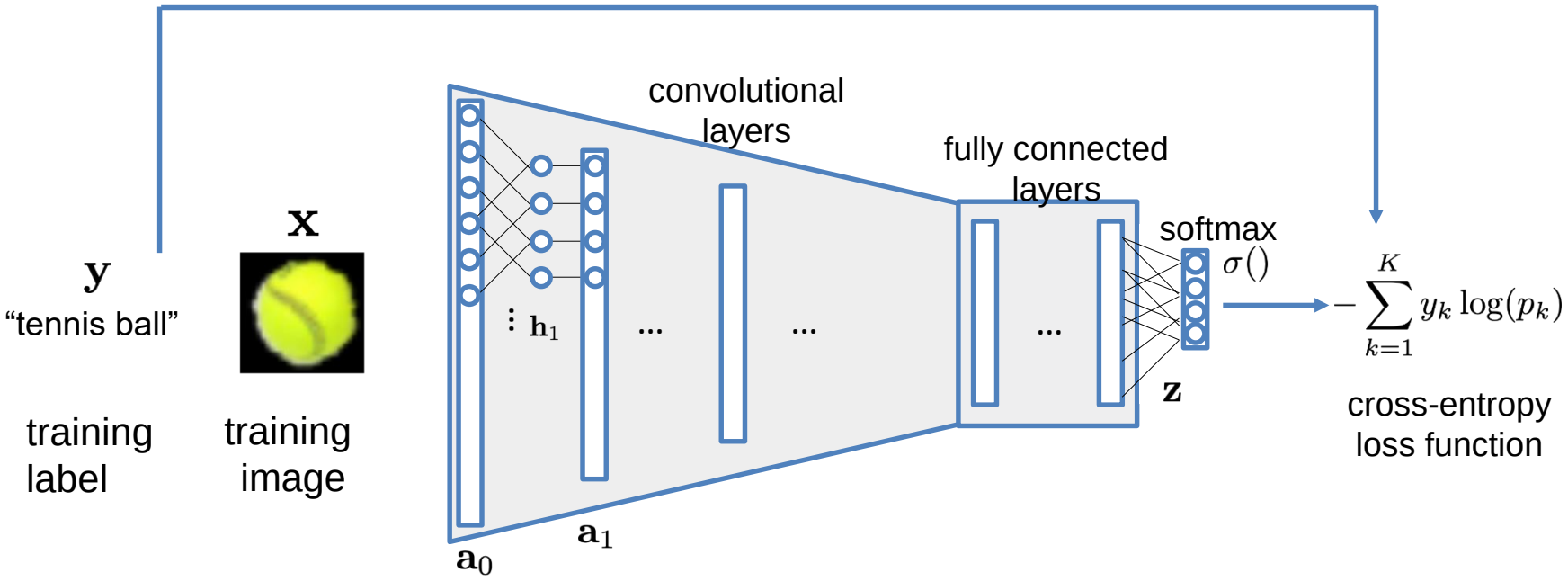
$$p(Y|X) = \int p(Y|X,w)p(w)dw$$

Our hypothesis is on parameters!

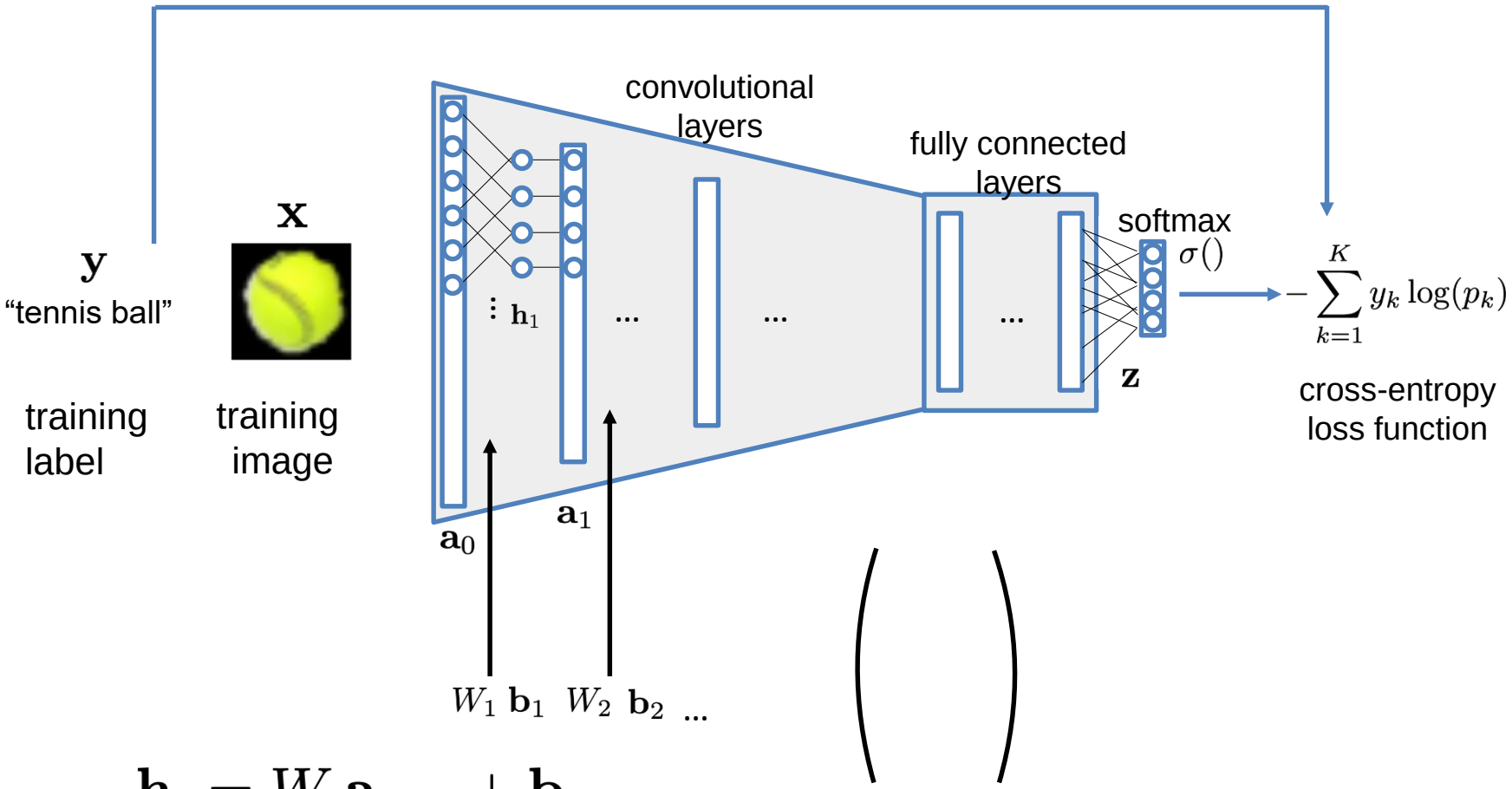
$p(\mathbf{w}\|\mathbf{X},\mathbf{Y})$	Posterior
$p(Y\|X,w)$	Likelihood
$p(w)$	Prior
$p(Y\|X)$	Evidence

- No conjugate priors – no analytical form for the posterior.
- Solving the integral over large quantities – intractable.
- Need Approximate Bayesian Inference techniques.

On Posteriors: An Example



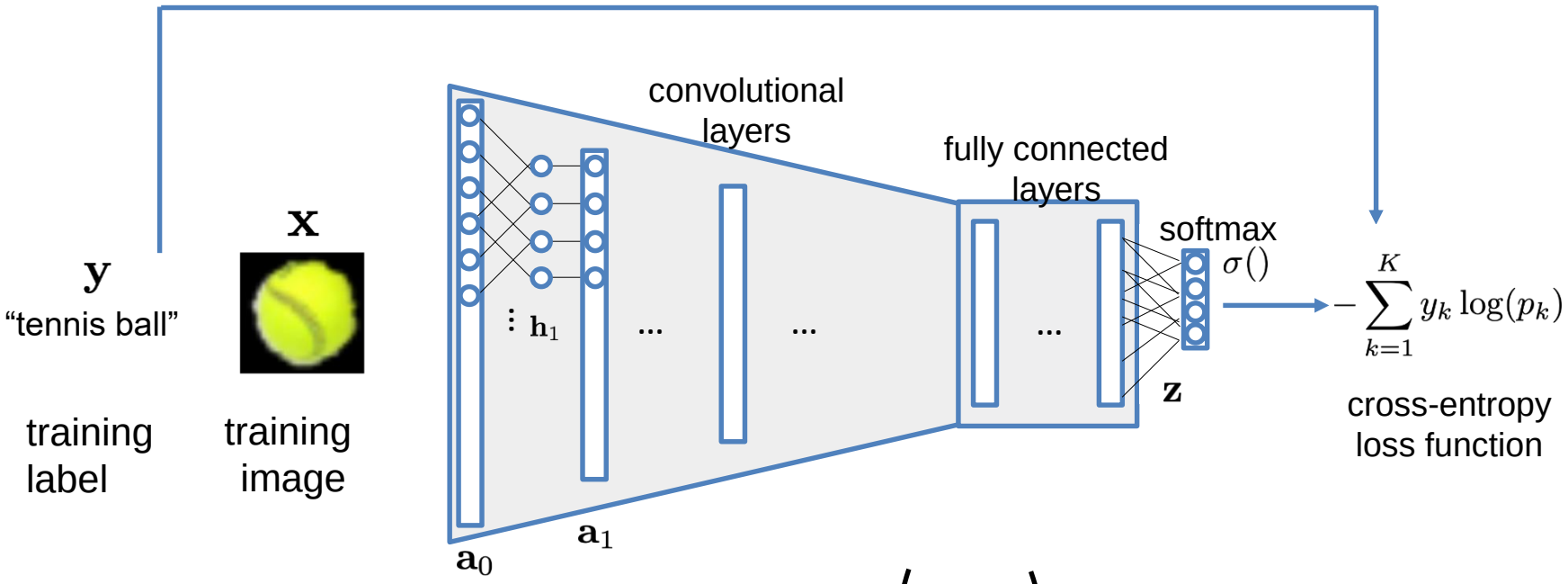
On Posteriors: An Example



$$\mathbf{h}_i = W_i \mathbf{a}_{i-1} + \mathbf{b}_i$$

$$\mathbf{a}_i = \phi(\mathbf{h}_i)$$

On Posteriors: An Example

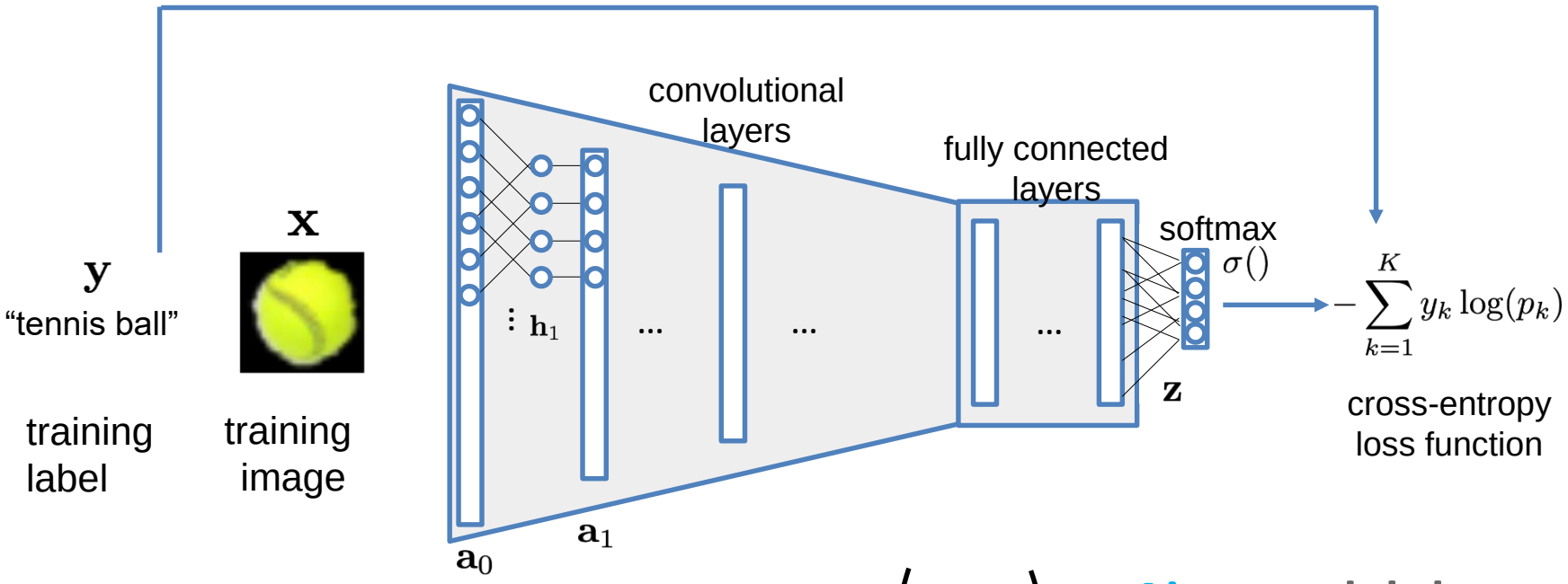


$$\mathbf{w} = \begin{pmatrix} W_1 \; \mathbf{b}_1 \\ W_2 \; \mathbf{b}_2 \\ \dots \end{pmatrix}$$

$$\mathbf{h}_i = W_i \mathbf{a}_{i-1} + \mathbf{b}_i$$

$$\mathbf{a}_i = \phi(\mathbf{h}_i)$$

On Posteriors: An Example



$$\mathbf{w} = \begin{pmatrix} W_1 \ \mathbf{b}_1 \\ W_2 \ \mathbf{b}_2 \\ \dots \end{pmatrix}$$

Aim: model the parameter posterior

$$p(\mathbf{w} \mid X, \mathbf{y})$$

$$\mathbf{h}_i = W_i \mathbf{a}_{i-1} + \mathbf{b}_i$$

$$\mathbf{a}_i = \phi(\mathbf{h}_i)$$

But: This can not be done efficiently in closed form!

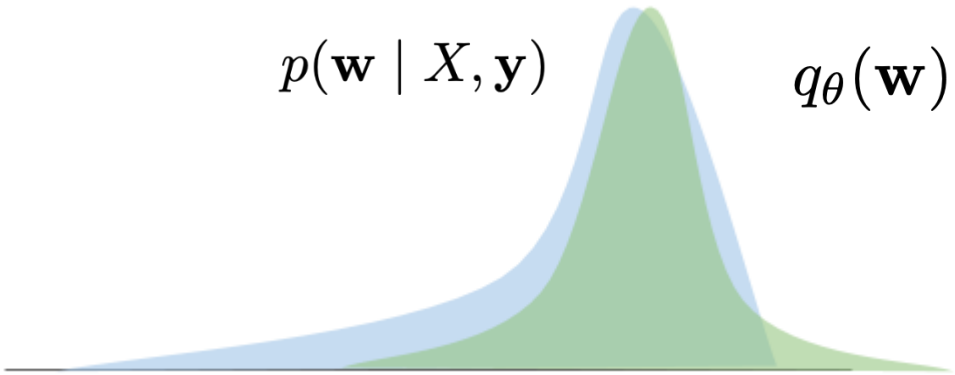
On Posteriors: How to approximate it?

In the literature, there are different principles:

1. Variational inference (next lectures)

Find a tractable distribution q that approximates p, e.g. in terms of the KL-divergence – through optimization:

$$q_\theta(\mathbf{w}) \approx p(\mathbf{w} \mid X, \mathbf{y})$$



On Posteriors: How to approximate it?

In the literature, there are different principles:

2. Sampling Methods (next lectures)

Generate samples from the posterior distribution:

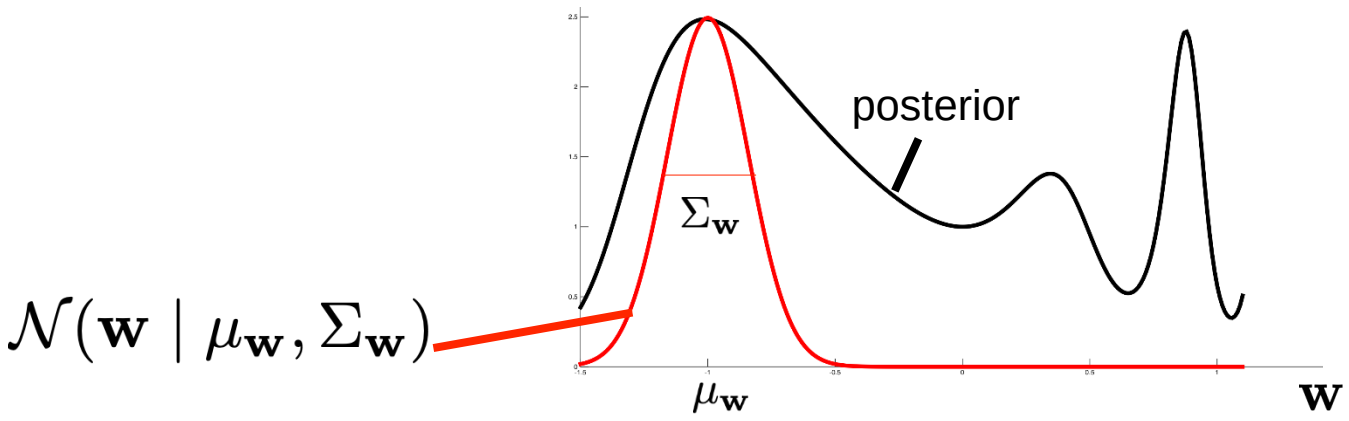
$$q_1, \ldots, q_M \sim p(\mathbf{w} \mid X, \mathbf{y})$$

On Posteriors: How to approximate it?

In the literature, there are different principles:

3. Laplace Approximation

Approximate the posterior at its mode with a Normal distribution.



On Posteriors: Laplace Approximation

Step 1: Train a neural network with point estimates.

E.g.: $$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$$ **(MLE)**

or: $$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$$ **(MAP)**

Recall that you can solve this optimization problem using stochastic gradient descents or some variants of it.

On Posteriors: Laplace Approximation

Step 2: approximate the log-posterior using the second-order Taylor expansion:

$$\log \mathbf{p}(\mathbf{w}|\mathbf{X},\mathbf{y}) \approx \log \mathbf{p}(\mathbf{w}^*|\mathbf{X},\mathbf{y}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^{\mathbf{T}}\mathbf{H}(\mathbf{w}-\mathbf{w}^*)$$

where H is the Hessian matrix:

$$[H_0]_{ij} = -\frac{\partial^2}{\partial w_i \partial w_j} \log p(y \mid \mathbf{x}, \mathbf{w})$$

Note: we are omitting the prior term for simplicity in derivation.

On Posteriors: Laplace Approximation

Step 3: Taking the exponential on both sides:

$$\log p(\mathbf{w} \mid X, \mathbf{y}) \approx \log p(\mathbf{w}^* \mid X, \mathbf{y}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T H (\mathbf{w} - \mathbf{w}^*)$$

$$p(\mathbf{w} \mid X, \mathbf{y}) \approx p(\mathbf{w}^* \mid X, \mathbf{y}) \exp\left(\frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T H (\mathbf{w} - \mathbf{w}^*)\right)$$

$$= p(\mathbf{w}^* \mid X, \mathbf{y}) \exp\left(-\frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \Sigma^{-1} (\mathbf{w} - \mathbf{w}^*)\right)$$

On Posteriors: Laplace Approximation

Step 3: Taking the exponential on both sides:

$$\log p(\mathbf{w} \mid X, \mathbf{y}) \approx \log p(\mathbf{w}^* \mid X, \mathbf{y}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T H (\mathbf{w} - \mathbf{w}^*)$$

$$p(\mathbf{w} \mid X, \mathbf{y}) \approx p(\mathbf{w}^* \mid X, \mathbf{y}) \exp\left(\frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T H (\mathbf{w} - \mathbf{w}^*)\right)$$

$$= p(\mathbf{w}^* \mid X, \mathbf{y}) \exp\left(-\frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \Sigma^{-1} (\mathbf{w} - \mathbf{w}^*)\right)$$

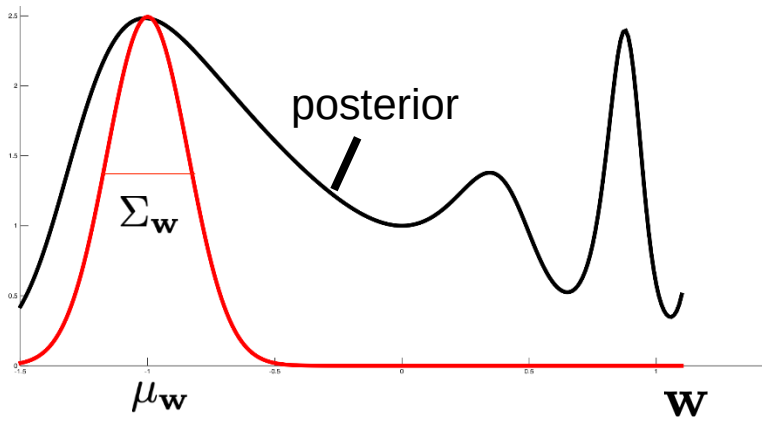
This is an unnormalised Gaussian. and the normaliser is

$$\boldsymbol{Z = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}}}$$

On Posteriors: Summary

- **Posterior Distribution:**

$$p(\mathbf{w}|\mathbf{X},\mathbf{Y}) = \frac{p(Y|X,w)p(w)}{p(Y|X)} \approx \mathcal{N}\left(w^*, H^{-1}\right)$$



- **Evidence Term:**

$$p(Y|X) = \int p(Y|X,w)p(w)dw$$

- **Computing the Hessian matrix involves taking 2nd order derivative.**
- **However, the Hessian matrix for deep neural networks is too large!**

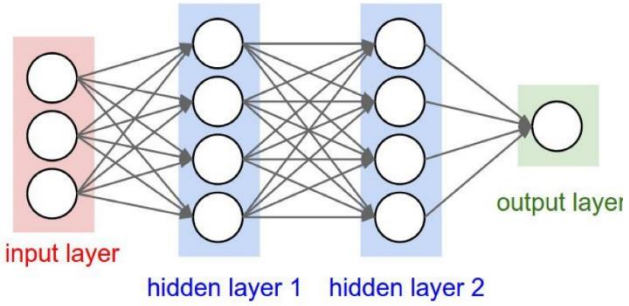
E.g. if you have 1 million parameters, the Hessian matrix is a matrix of size 1 million by 1 million. Then, how to compute such large matrix?

- **A short-cut is to approximate the Hessian matrix, using Fisher Information or Gauss Newton from 2nd order optimization for networks.**

Tables of Contents

Aim: Understanding the basics related to Bayesian Neural Networks

Recap on Bayesian Neural Networks



Given that …

- **Data:** $\mathcal{D} = \{(x^n, y^n)\}_{n=1}^N = (X, y)$
- **Parameters w are weights of neural networks.**

Then, Instead of trying to find a most likely parameters:

- **Define prior:** $p(w)$
- **Bayesian learning of posterior:** $p(w|\mathcal{D}) \propto p(y|X, w)p(w)$
- **Prediction:** $p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$

On Predictions: Bayesian Vs Frequentist

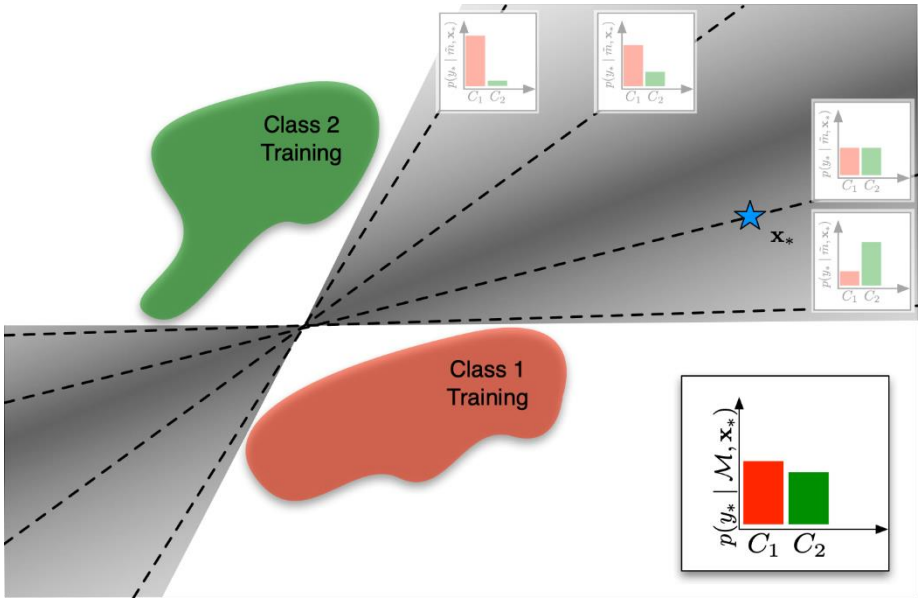
- Two different paradigms of statistics (similar to EMAC or VIM).

Frequentist statistics	Bayesian statistics
Probabilities are frequency of an outcome. Objective and relative frequency	Probabilities are your own beliefs about randomness. Subjective and degree of beliefs.
Data is random, not your parameters (or no model uncertainty exists). Hence, you have confidence intervals and point estimates.	Data is fixed, but your parameters are random variables (parameters or model cannot be perfect, and their uncertainty is expressed with probability theory).
E.g. ensembles such as random forest or Ada boost.	E.g. Gaussian processes, Bayesian Neural Networks.

On Predictions: Bayesian Predictions

- So far, we specified a **prior distribution**, and have also an approximation to the **posterior distribution**.
- In Bayesian methods, the predictions are also **not a point estimate**, but should be **a probability distribution**, e.g. GPs.
- A **predictive distribution** should account for **uncertainty** about $\mathbf{w}$.
- **Marginalize** the uncertain $\mathbf{w}$ quantity to predict!

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$



On Predictions: The problem of intractable integrals

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$

- **No analytical form** for such integrals.
- Numerical methods? E.g. Runge-Kutta? **Too high dimensional.**
- **Approximate Solutions** can be obtained:

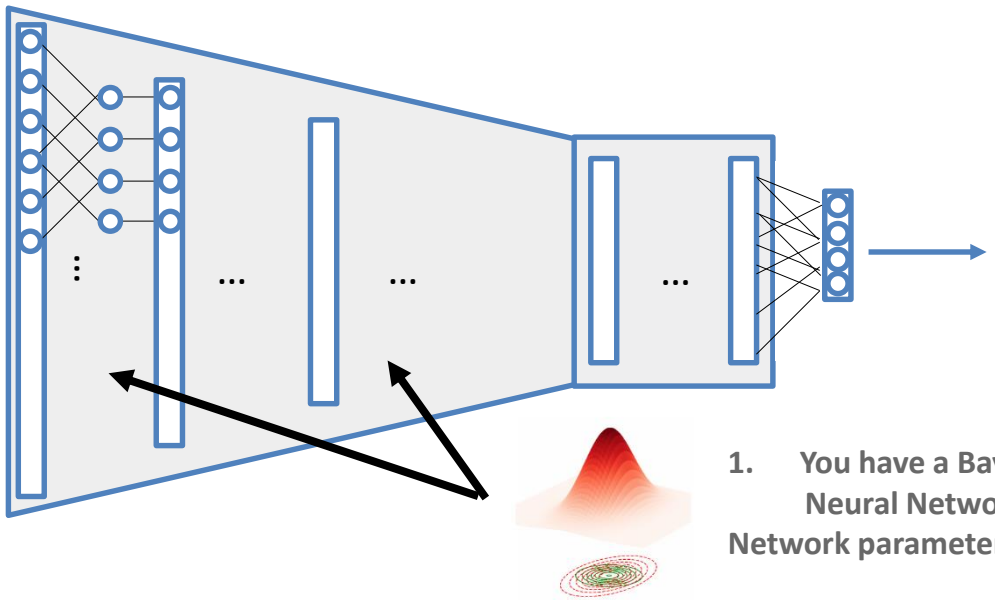
- Monte-carlo integration.
- Linear approximation.

On Predictions: Monte-carlo Integration

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$

$$\rightarrow \quad y^* \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, w_t) \qquad w_t \sim p(w|\mathcal{D})$$

- Monte-carlo approximates integrals in large dimensions via sampling.



1. You have a Bayesian Neural Network.

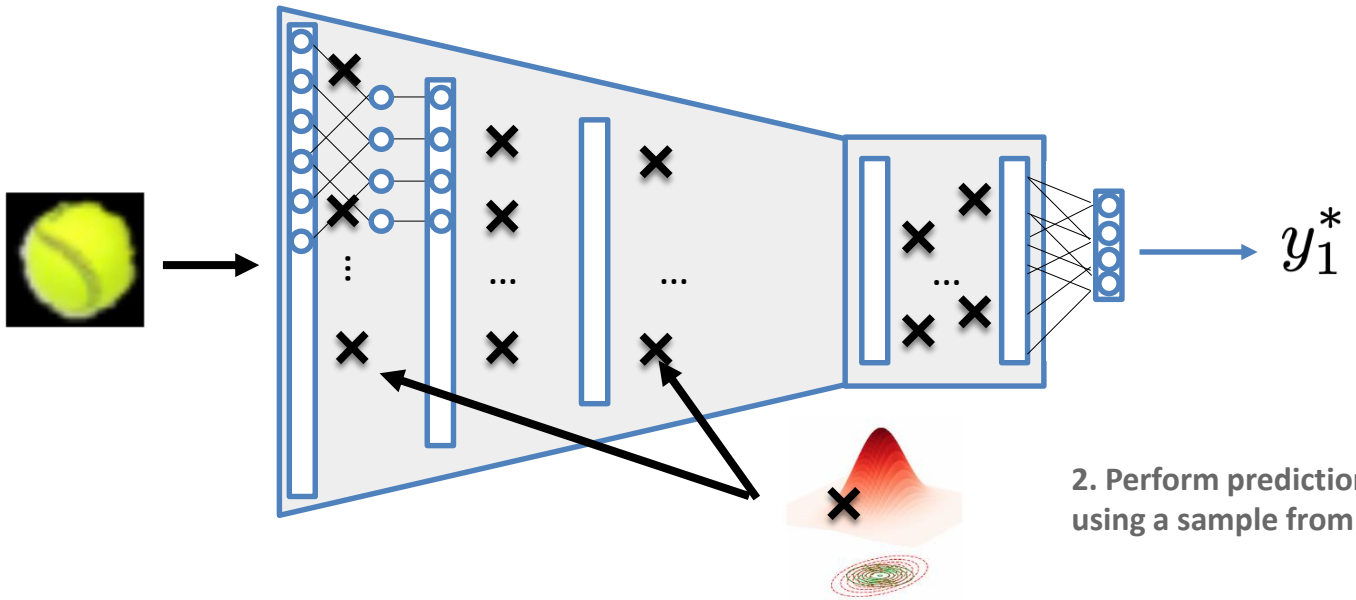
Network parameters are distributions!

On Predictions: Monte-carlo Integration

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$

$$\rightarrow \quad y^* \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, w_t) \qquad w_t \sim p(w|\mathcal{D})$$

- Monte-carlo approximates integrals in large dimensions via sampling.



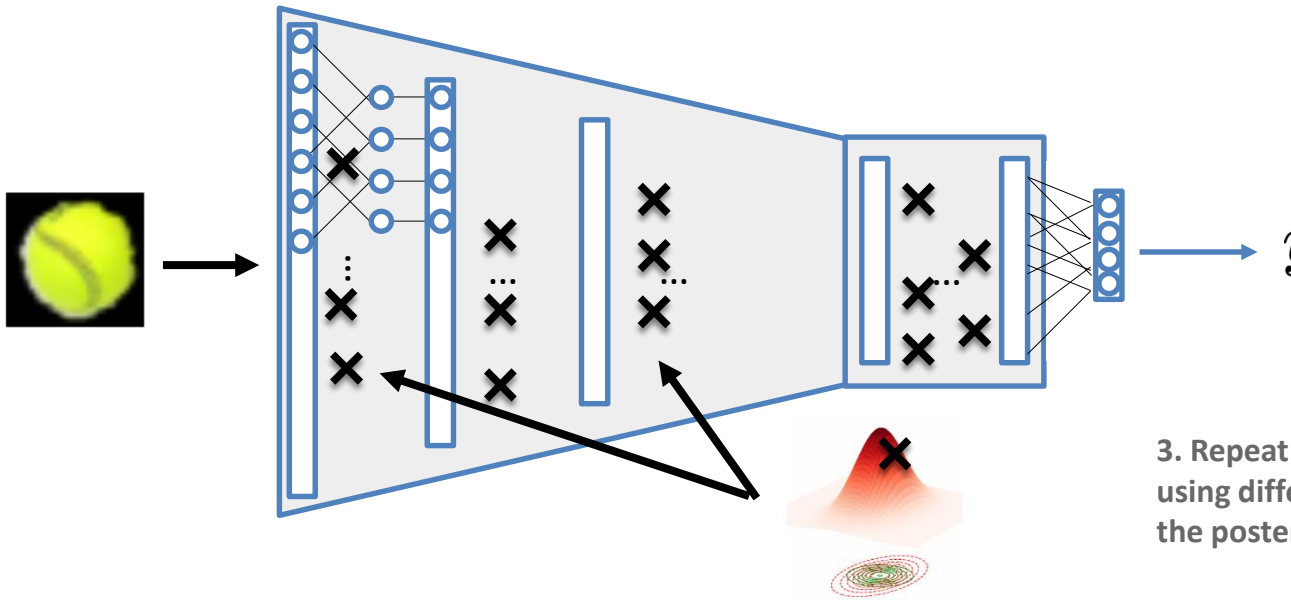
2. Perform prediction with a test sample using a sample from distribution.

On Predictions: Monte-carlo Integration

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$

$$\rightarrow \quad y^* \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, w_t) \qquad w_t \sim p(w|\mathcal{D})$$

- Monte-carlo approximates integrals in large dimensions via sampling.



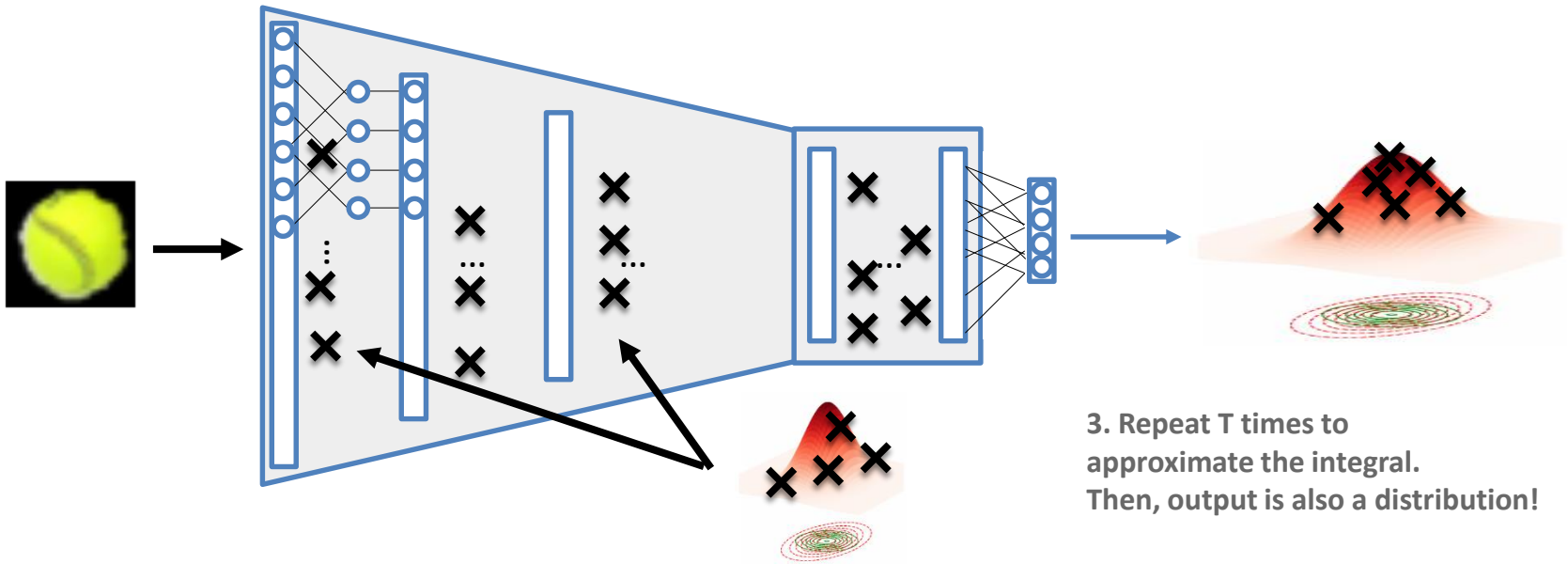
3. Repeat this with differently using different samples from the posterior distribution.

On Predictions: Monte-carlo Integration

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$

$$\rightarrow \quad y^* \approx \frac{1}{T}\sum_{t=1}^{T} p(y^*|x^*, w_t) \qquad w_t \sim p(w|\mathcal{D})$$

- Monte-carlo approximates integrals in large dimensions via sampling.

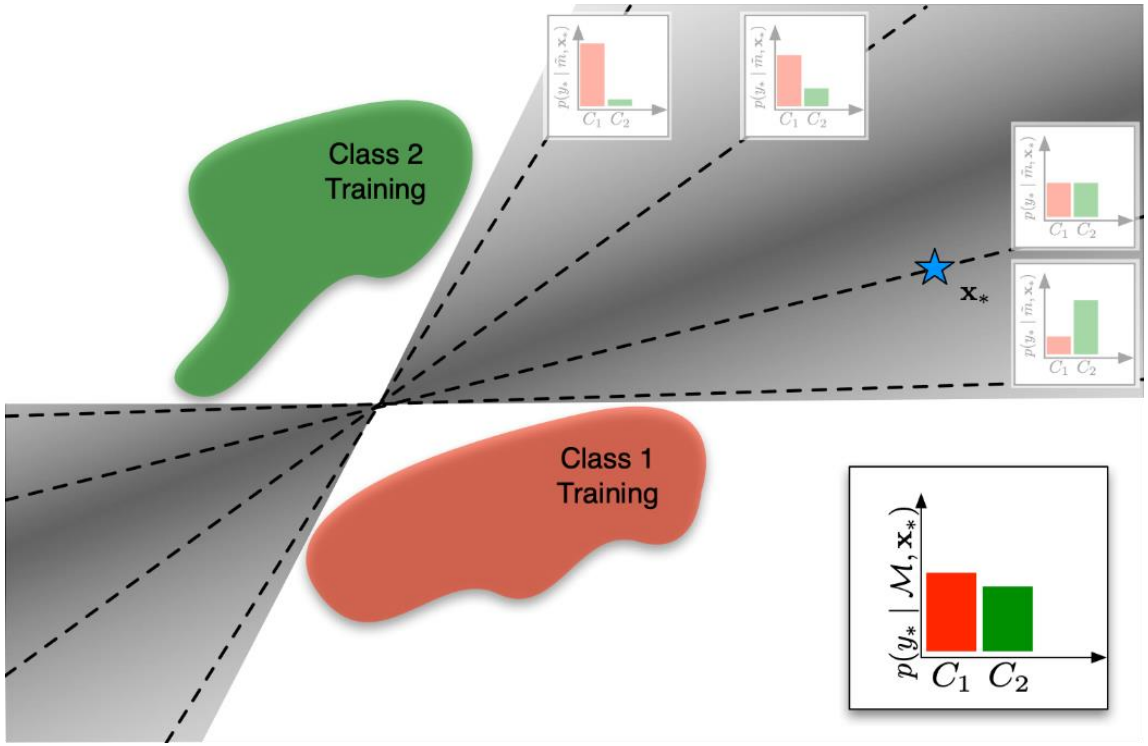


3. Repeat T times to approximate the integral. Then, output is also a distribution!

On Predictions: Monte-carlo integration

$$p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$$

- **Bias-variance** view.
- **Uncertainty estimates** can be obtained!
- **Convergence rate** is bounded to
- **But**, expensive!
  E.g. 100 times slower
  if 100 samples are used.



On Predictions: A Linear Approximation

- Aim is again to evaluate the integral: $p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$
- Consider that you have the followings:

1. The prior distributions specified as a normal: $p(w) \sim \mathcal{N}(0, \sigma I)$

2. Trained Neural Networks for MAP estimates: $w^* = \mathbf{argmax}\, p(w|X, Y)$

3. Obtained the posterior distributions with Laplace Approximation:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{Y}) = \frac{p(Y|X, w)p(w)}{p(Y|X)} \approx \mathcal{N}\left(w^*, H^{-1}\right)$$

On Predictions: A Linear Approximation

- Aim is again to evaluate the integral: $\boldsymbol{p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw}$

Step 1: Take Taylor expansion on neural network $\boldsymbol{f(x, w)}$:

$$\boldsymbol{f(x, w) \simeq f(x, w^*) + g^T(w - w^*) + \varepsilon}$$

where we define $\boldsymbol{g = \frac{\partial f(x, w)}{\partial w}\Big|_{w=w^*}}$ **the Jacobian of neural networks.**

On Predictions: A Linear Approximation

- Aim is again to evaluate the integral: $\boldsymbol{p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw}$

Step 1: Take Taylor expansion on neural network.

Step 2: Recall our random variables are Gaussian distributions:

$$\boldsymbol{f(x, w) \simeq f(x, w^*) + g^T(w - w^*) + \varepsilon}$$

$$\rightarrow \quad \boldsymbol{p(y|x, w) \simeq \mathcal{N}\left(y\middle|f(x, w^*) + g^T(w - w^*), \sigma\right)}$$

Assume output is also a Gaussian, then mean is a linear function of w.

On Predictions: A Linear Approximation

- Aim is again to evaluate the integral: $\boldsymbol{p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw}$

Step 1: Take Taylor expansion on neural network.

Step 2: Recall our random variables are Gaussian distributions.

Step 3: Further using the rules of Gaussian distributions:

$$\boldsymbol{f(x, w) \simeq f(x, w^*) + g^T(w - w^*) + \varepsilon}$$

$$\rightarrow \quad \boldsymbol{p(y|x, w) \simeq \mathcal{N}\left(y \middle| f(x, w^*) + g^T(w - w^*), \sigma\right)}$$

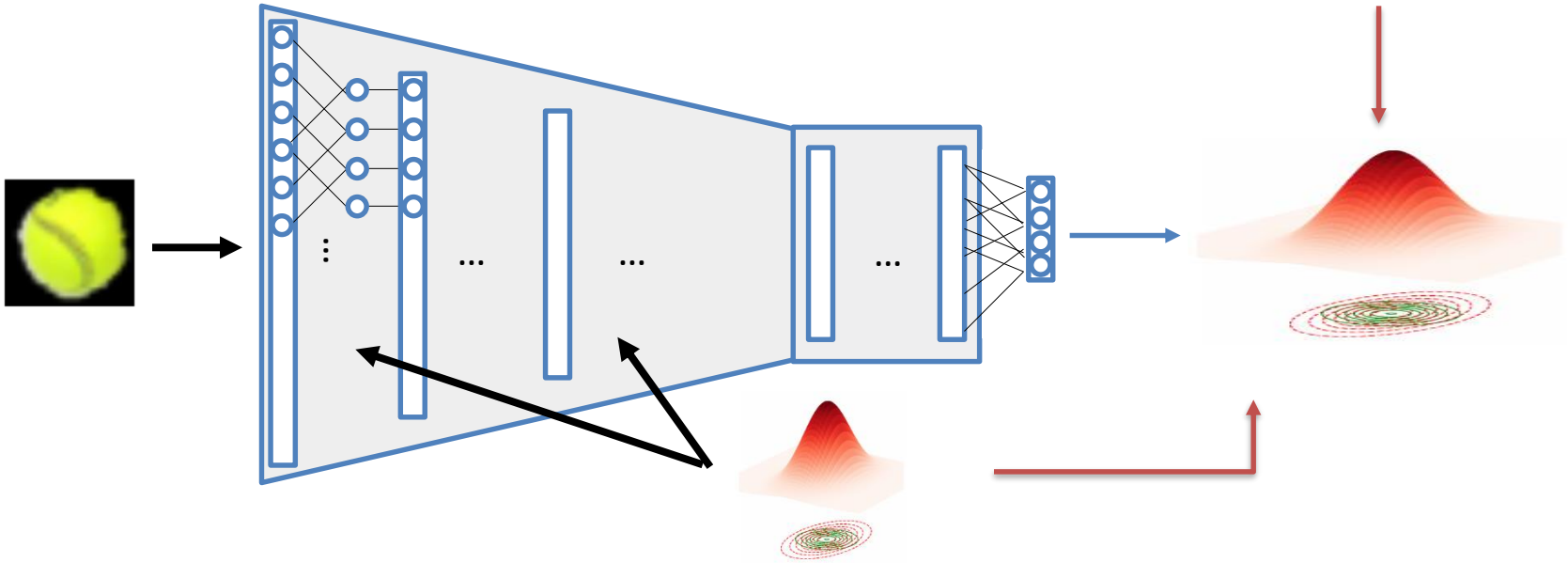
$$\rightarrow \quad \boldsymbol{p(y^*|x^*, \mathcal{D}) \simeq \mathcal{N}(y|f(x, w^*), \Sigma)}$$

Marginal and conditional Gaussians

$$\boldsymbol{\Sigma = \sigma + g(x)^T H g(x)}$$

On Predictions: A Linear Approximation

- Aim is again to evaluate the integral: $p(y^*|\mathcal{D}, x^*) = \int p(y^*|x^*, w)p(w|\mathcal{D})dw$
- Linear Approximation: $p(y^*|x^*, \mathcal{D}) \simeq \mathcal{N}(y|f(x, w^*), \Sigma)$ with $\Sigma = \sigma + g(x)^T H g(x)$



- Do not need multiple samples but a good approximation?

Tables of Contents

Aim: Understanding the basics related to Bayesian Neural Networks

Bayesian Deep Learning

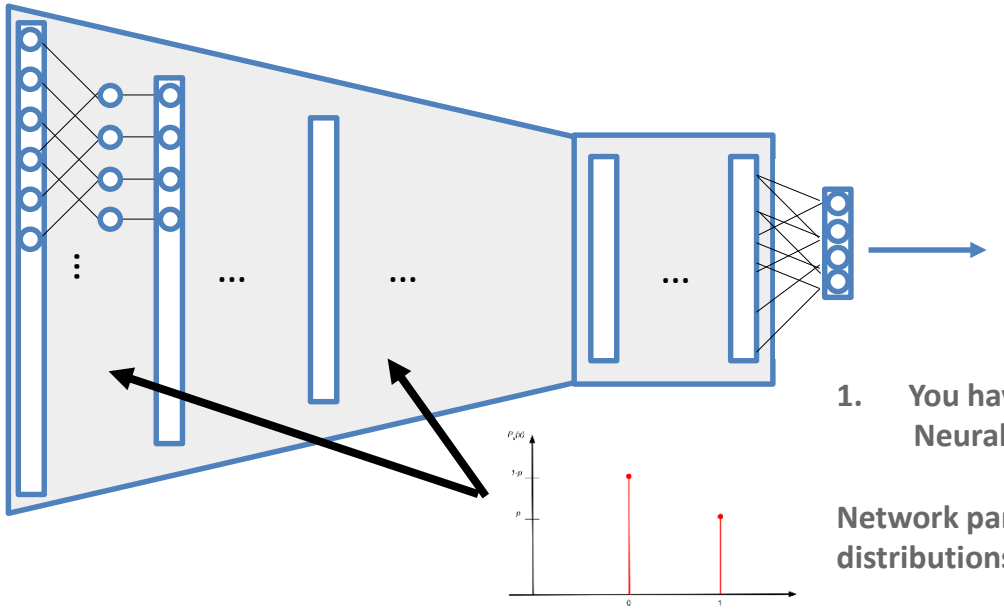
- So far, we have learned about researched in 1990s already.
- But, with deep learning, very popular active area of research.

- Many open questions: How to specify prior? An accurate and efficient way of computing posterior probabilities? Cheap but accurate ways of estimating predictive uncertainty?
- There are some frameworks that might be useful!

Bayesian Deep Learning

- **Dropout as variational inference (MC-dropout):**

- **Main idea: one can prove that training a neural network with dropout is equivalent to performing variational inference (hence result is a Bayesian Neural Networks).**



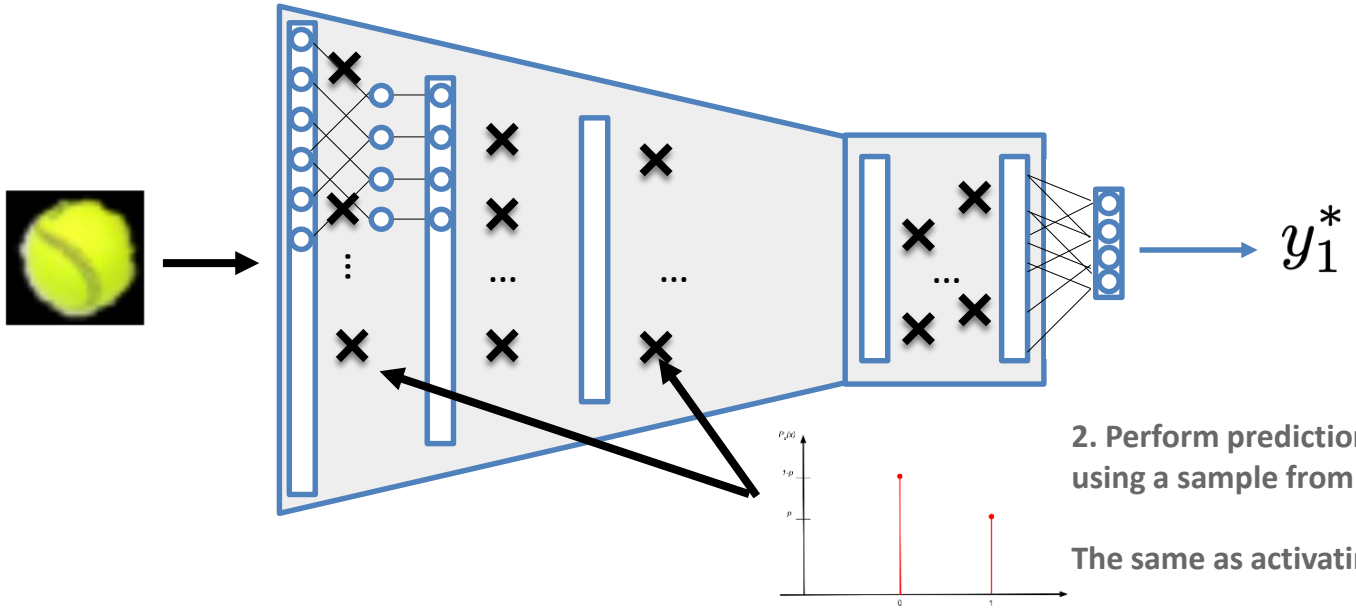
1. You have a Bayesian Neural Network.

Network parameters are Bernoulli distributions since dropout is!

Bayesian Deep Learning

- **Dropout as variational inference (MC-dropout):**

- **Main idea: one can prove that training a neural network with drop out is equivalent to performing variational inference (hence result is a Bayesian Neural Networks).**



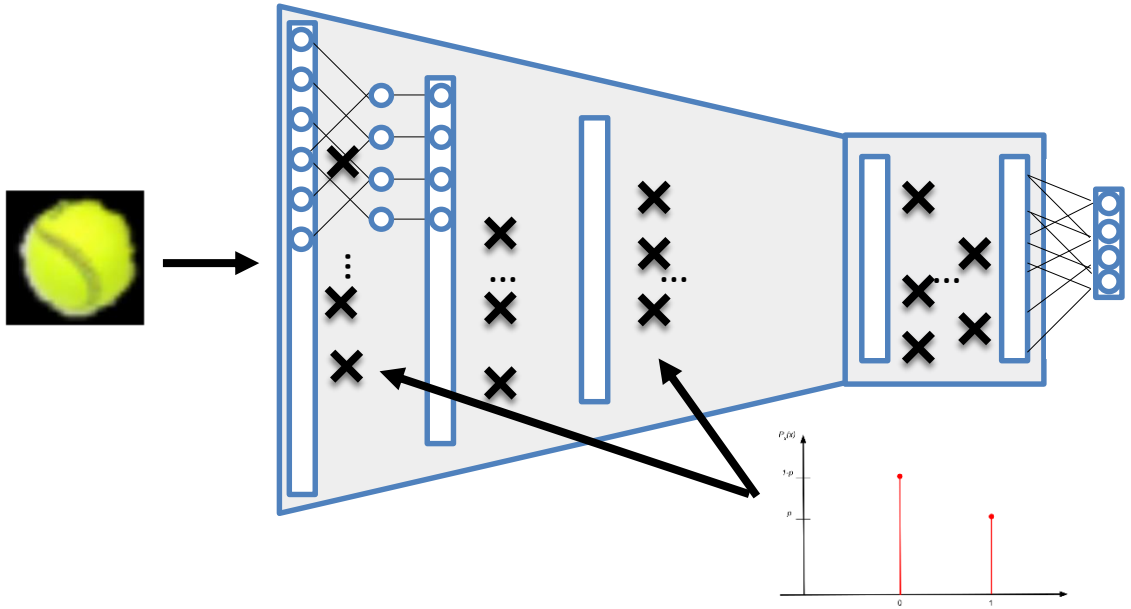
2. Perform prediction with a test sample using a sample from distribution.

The same as activating dropout at testing.

Bayesian Deep Learning

- **Dropout as variational inference (MC-dropout):**

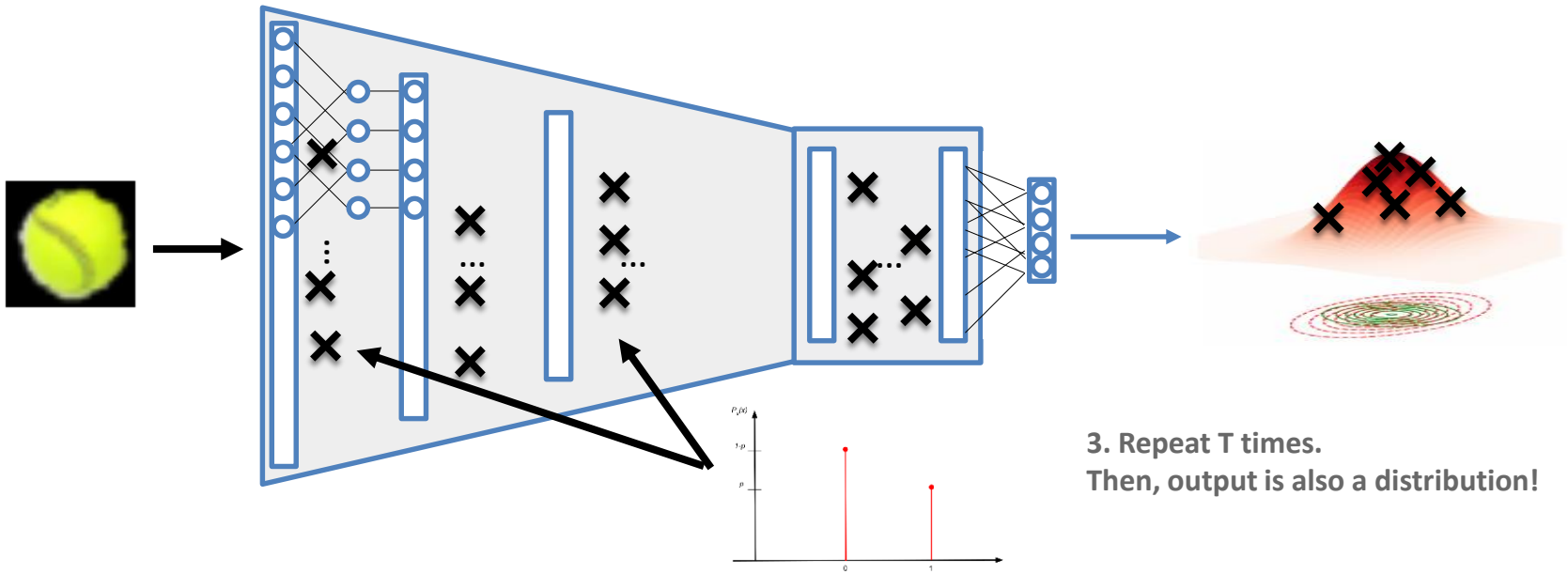
- **Main idea: one can prove that training a neural network with drop out is equivalent to performing variational inference (hence result is a Bayesian Neural Networks).**



3. Repeat this with differently using different samples from activated dropout layers.

Bayesian Deep Learning

- Dropout as variational inference (MC-dropout):
- **Main idea:** one can prove that training a neural network with drop out is equivalent to performing variational inference (hence result is a Bayesian Neural Networks).



3. Repeat T times.
Then, output is also a distribution!

On Predictions: Monte-carlo Integration

- Dropout as variational inference (MC-dropout):
- **Main idea:** one can prove that training a neural network with drop out is equivalent to performing variational inference (hence result is a Bayesian Neural Networks).
- Very easy to use. You only enable dropout layers on during testing.
- Assumptions about the model, e.g. use of dropout layers, specific loss.

Bayesian Deep Learning

- **Scalable Laplace Approximation**
- **Main idea: use scalable approximation of the Hessian.**

$$-\Sigma_{\mathbf{w}} \approx H^{-1}$$

The Hessian for the entire data set is approximately the expectation of the single H_0

$$H \approx E_{\mathbf{y}}[H_0]$$

Bayesian Deep Learning

- **Scalable Laplace Approximation**
- **Main idea: use scalable approximation of the Hessian.**

$$-\Sigma_{\mathbf{w}} \approx H^{-1} \approx F^{-1}$$

Fisher information matrix

"The Fisher information quantifies the amount of information about the parameter $\mathbf{w}$"

$$F(\mathbf{w}) := E_{\mathbf{y}}\left[\left(\frac{\partial}{\partial \mathbf{w}} \log p(\mathbf{y} \mid X, \mathbf{w})\right)^2\right] = E_{\mathbf{y}}[H_0]$$

Bayesian Deep Learning

- **Scalable Laplace Approximation**
- **Main idea: use scalable approximation of the Hessian.**

$$-\Sigma_{\mathbf{w}} \approx H^{-1} \approx F^{-1} \approx \begin{bmatrix} F_1 & & & \\ & F_2 & & \\ & & \ddots & \\ & & & F_l \end{bmatrix}^{-1}$$

Fisher information matrix

$$F = \mathbb{E}[\delta\boldsymbol{\theta}\delta\boldsymbol{\theta}^T]$$

where

$$\delta\boldsymbol{\theta} = \nabla_{\mathbf{W}} \log p(\mathbf{y} \mid \mathbf{x}, \mathbf{W})$$

Assume uncorrelated layers

Bayesian Deep Learning

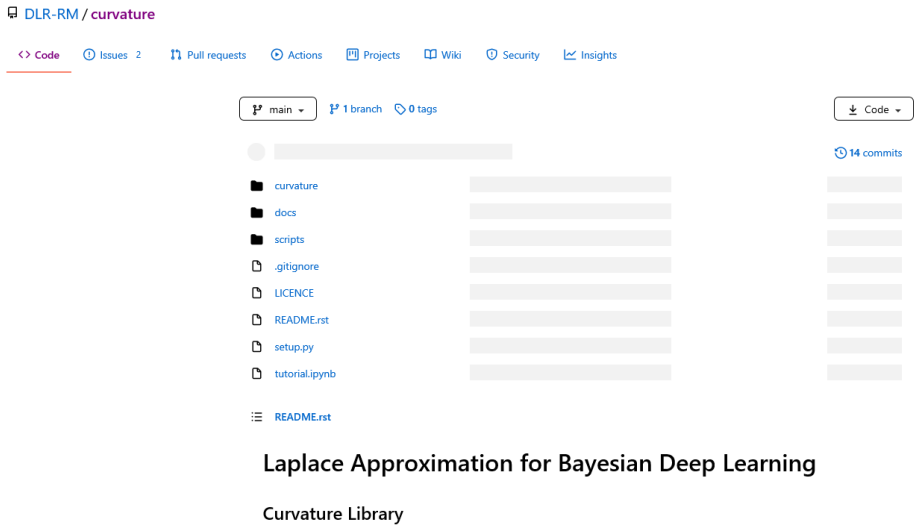
- Scalable Laplace Approximation
- **Main idea:** use scalable approximation of the Hessian.

$$-\Sigma_{\mathbf{w}} \approx H^{-1} \approx F^{-1} \approx \begin{pmatrix} F_1 & & & \\ & F_2 & & \\ & & \ddots & \\ & & & F_l \end{pmatrix}^{-1}$$

$$F_i = \mathbb{E}\left[\mathbf{a}_{i-1}\mathbf{a}_{i-1}^T \otimes \mathbf{g}_i\mathbf{g}_i^T\right] \quad \text{where} \quad \mathbf{g}_i = \frac{\partial}{\partial \mathbf{w}} h_i$$

Bayesian Deep Learning

- Scalable Laplace Approximation
- **Main idea:** use scalable approximation of the Hessian.
- Easy to use as it directly works with already trained neural networks.

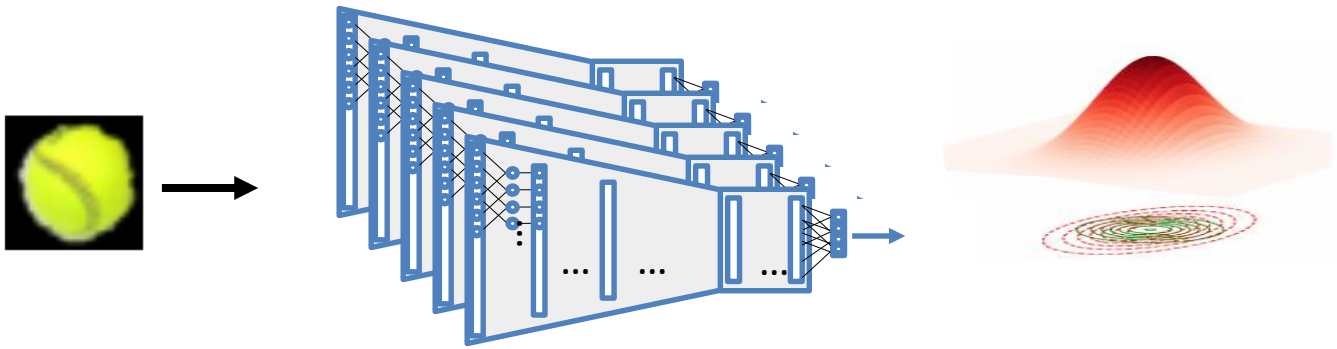


- Predictions with both monte-carlo method and linear approximation.
- Scales to large data-set and only assumes Hessian exists.
- But, are we making good approximations?

Bayesian Deep Learning

- **Deep Ensembles (a Frequentist approach):**

- **Main idea: train the same neural networks with different initializations and use them as ensembles.**

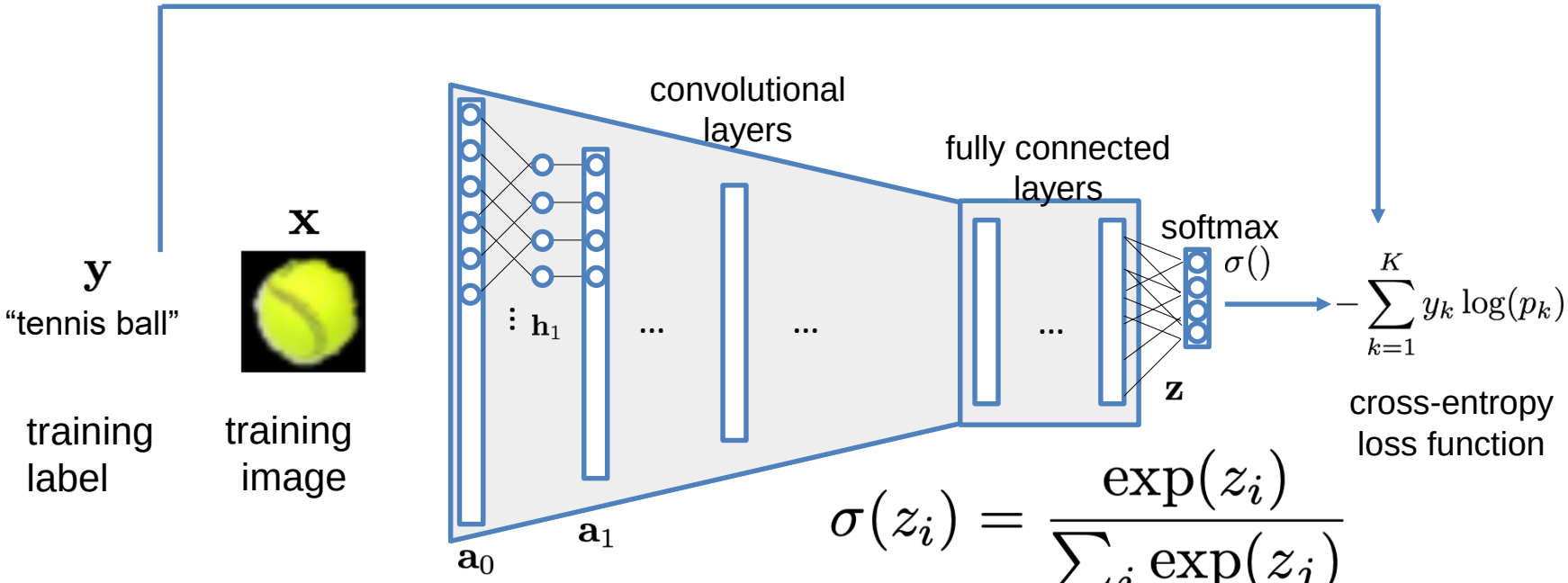


Bayesian Deep Learning

- Deep Ensembles (a Frequentist approach):
- **Main idea:** train the same neural networks with different initializations and use them as ensembles.
- Easy to use; using the same code for training many times.
- High performing often in practice.
- Memory intensive? Can you load them all on a GPU?
- Model selection? Other downstream tasks of Bayesian methods?

Bayesian Deep Learning

- **Temperature Scaling (a Frequentist approach)**



Bayesian Deep Learning

- **Temperature Scaling (a Frequentist approach)**

Intuitively, the confidence estimate of a prediction should correspond to the true probability of being correct

$$f(\mathbf{x}^*) = (y^*, c^*)$$

classifier — test data — estimated confidence

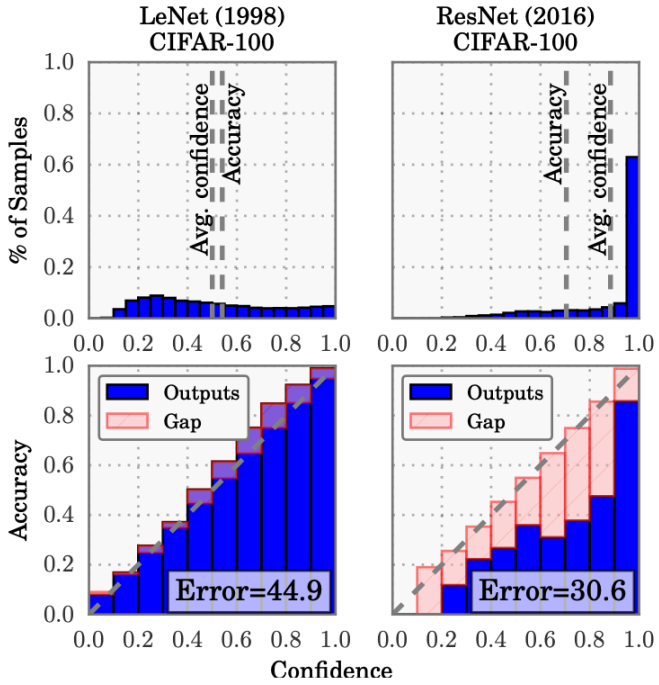
Bayesian Deep Learning

- **Temperature Scaling (a Frequentist approach)**

Intuitively, the confidence estimate of a prediction should correspond to the true probability of being correct

$$f(\mathbf{x}^*) = (y^*, c^*)$$

classifier — test data — estimated confidence

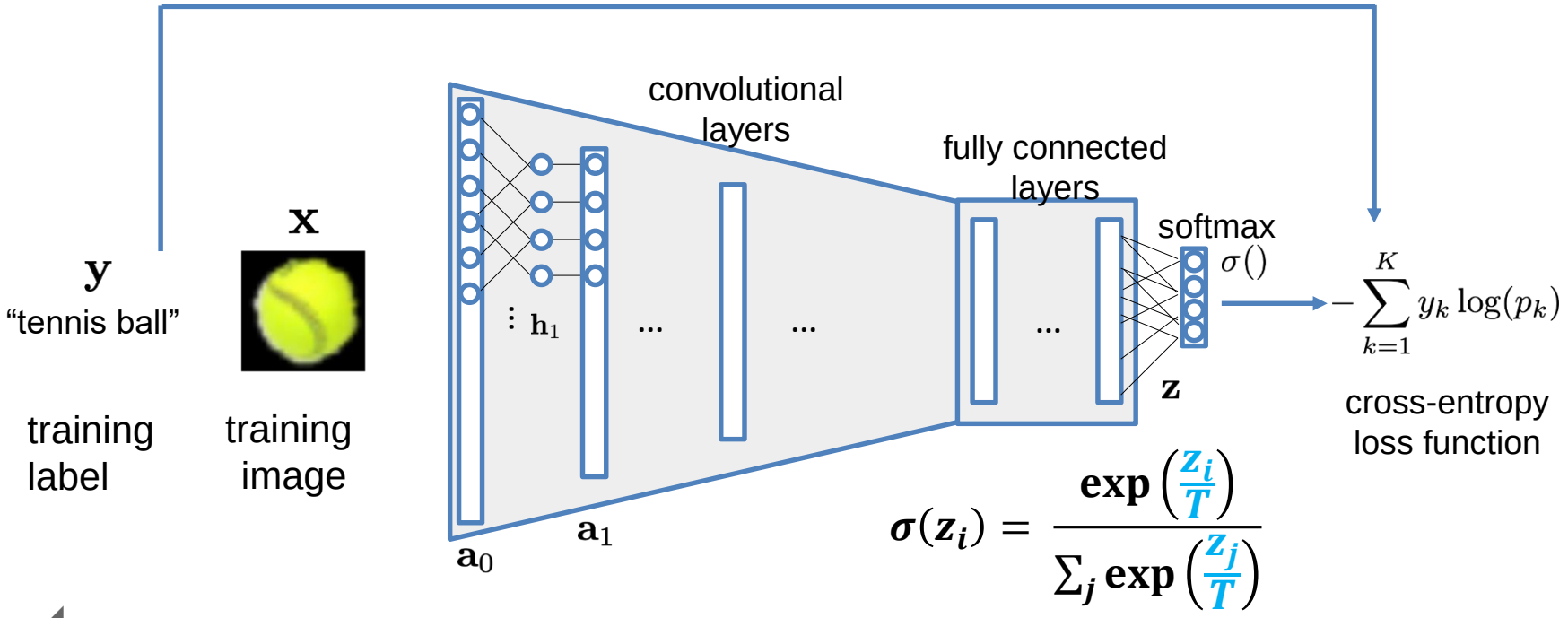


Thus, we want $p(y^* = y \mid c^* = c) = c \qquad \forall c \in [0, 1]$

$$\mathbb{E}[\mid p(y^* = y \mid c^* = c) - c \mid]$$

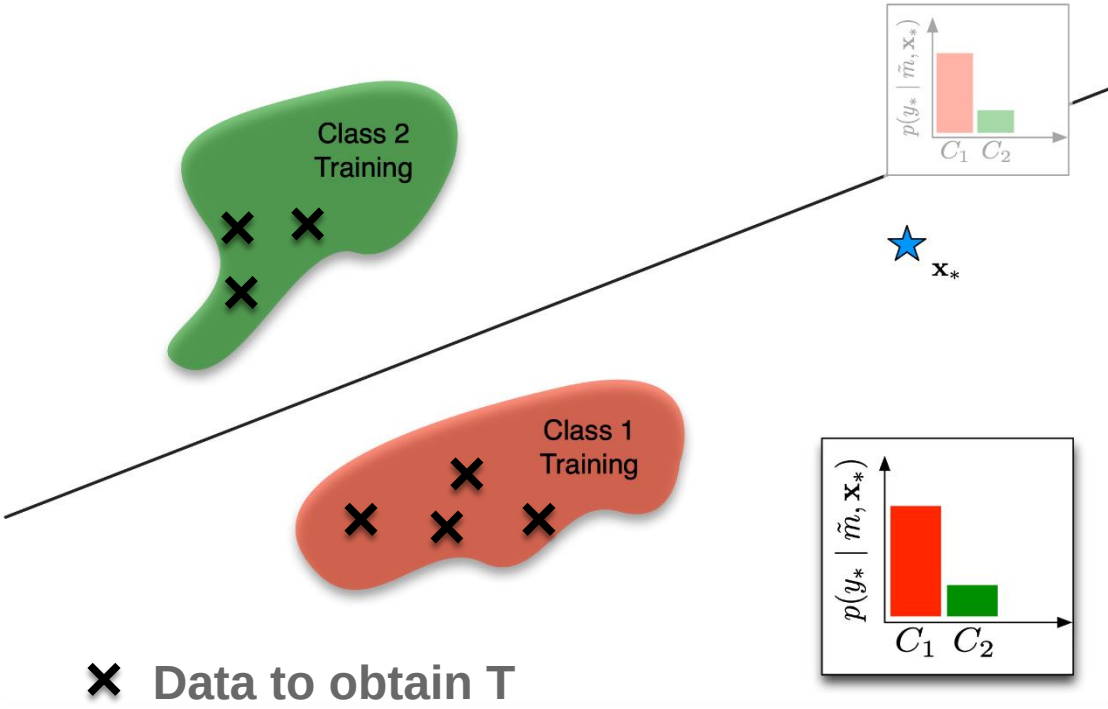
Bayesian Deep Learning

- **Temperature Scaling**
- **Main idea:** **Find a scaling factor T (using a set of training samples), that ensures calibration.**



Bayesian Deep Learning

- Temperature Scaling
- **Main idea:** Find a scaling factor T (using a set of training samples), that ensures calibration.
- Very simple conceptually and easy to use.
- Still point estimate Vs model uncertainty?
- What about regression?
- Is it really mis-calibrated?



Summary

Main Take Aways:

- **Bayesian Neural Networks combines Bayesian reasoning with Neural Networks in order to reduce overfitting, uncertainty estimates, etc.**
- **The idea is to specify a prior distribution over the parameters, update to the posterior given data, and use it to make an average prediction.**

Bayesian Neural Networks:

- **On prior: a normal with zero mean and a variance as default.**
- **On posterior: Laplace Approximation.**
- **On predictions: Monte-carlo integration and a linear approximation.**

Bayesian Deep Learning:

- **Several modern techniques for uncertainty estimation.**