Computer Science

CSC535: Probabilistic Graphical Models

Bayesian Deep Learning

Prof. Jason Pacheco

Outline

- Artificial Neural Network (ANN) : A Review
- Shortcomings of Standard Deep Learning
- Bayesian Deep Learning

Outline

- Artificial Neural Network (ANN) : A Review
- Shortcomings of Standard Deep Learning
- Bayesian Deep Learning

Basis Functions

Basis functions transform linear models into nonlinear ones…

Linear Regression

$$y = w^T x$$

$$y = w^T \phi(x)$$

Classification
(Logistic Regression)

$$y = \sigma(w^T x)$$

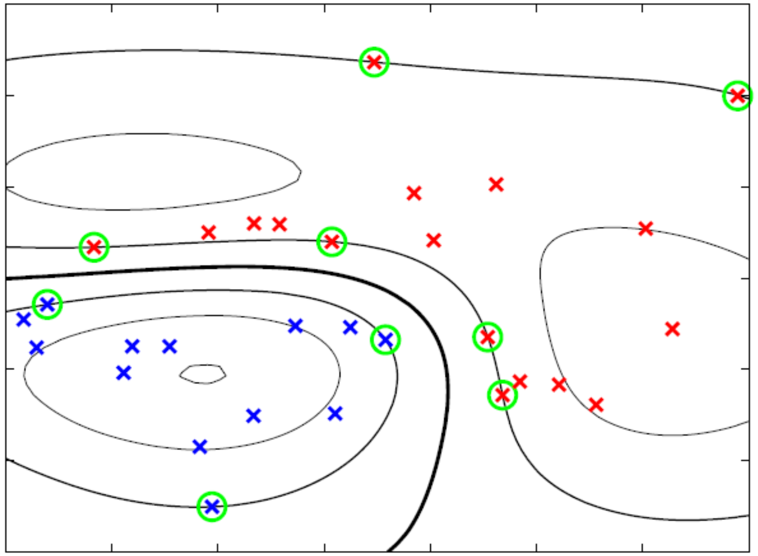
$$y = \sigma(w^T \phi(x))$$

…but it is often difficult to find a good basis transformation

Learning Basis Functions

What if we could learn a basis function so that a simple linear model performs well…

Data Space

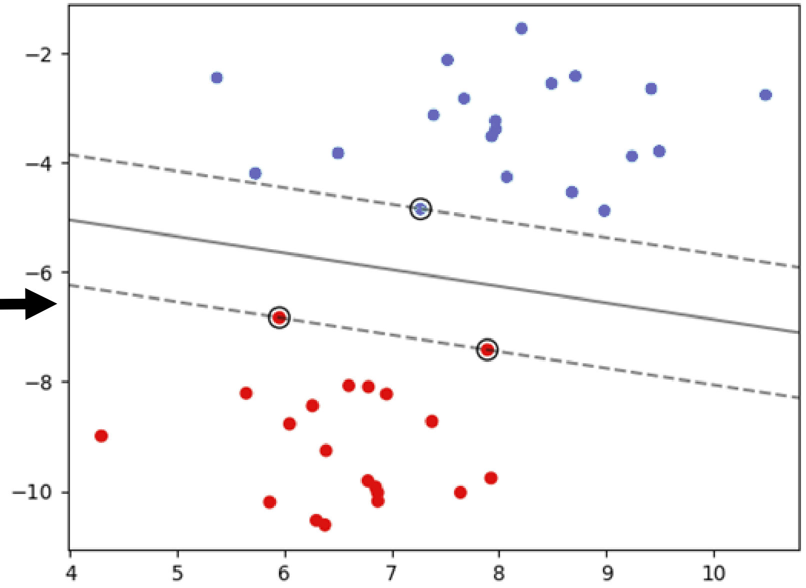


Ignore the circled points…I reused these from the SVM slides

Neural Net

$$\phi(x)$$

Warped Space



…this is essentially what standard neural networks do…

Neural Networks

- Flexible nonlinear transformations of data
- Resulting transformation is easily fit with a linear model
- Relatively efficient learning procedure scales to massive data
- Apply to many Machine Learning / Data Science problems
  - Regression
  - Classification
  - Dimensionality reduction
  - Function approximation
  - Many application-specific problems

Neural Networks

Forms of NNs are used all over the place nowadays…

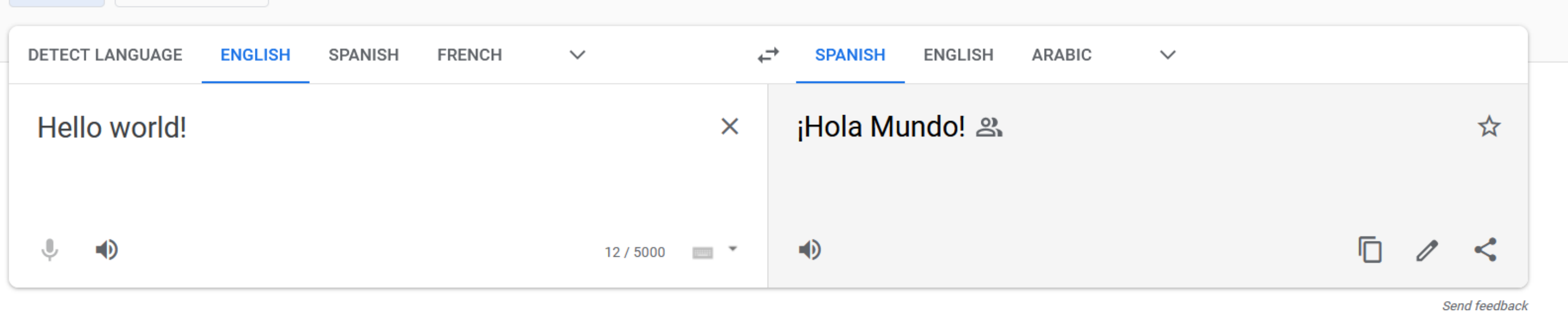
ChatGPT

Large Language Models

Self-Driving Cars

Creepy Robots

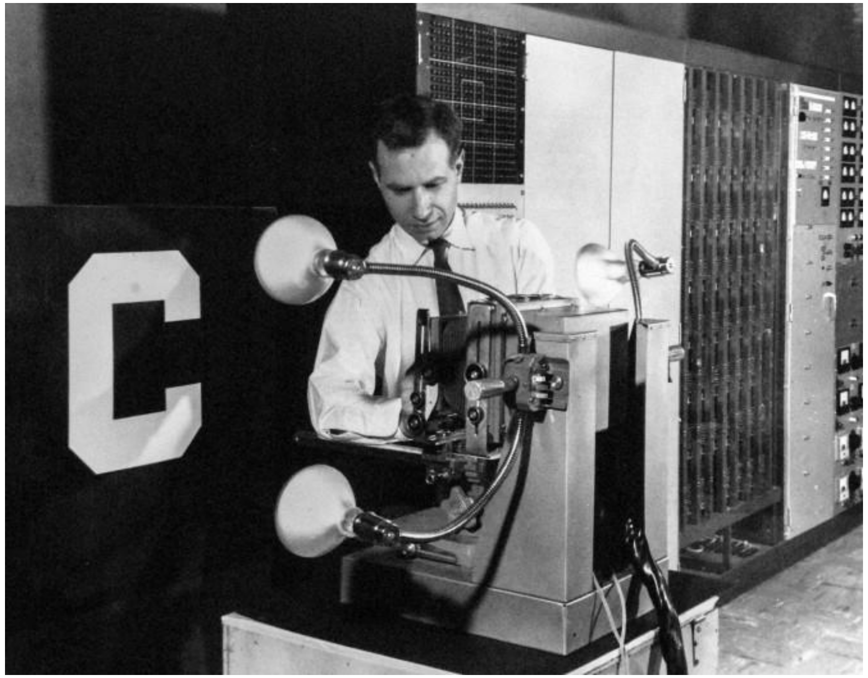
Machine Translation



Rosenblatt's Perceptron

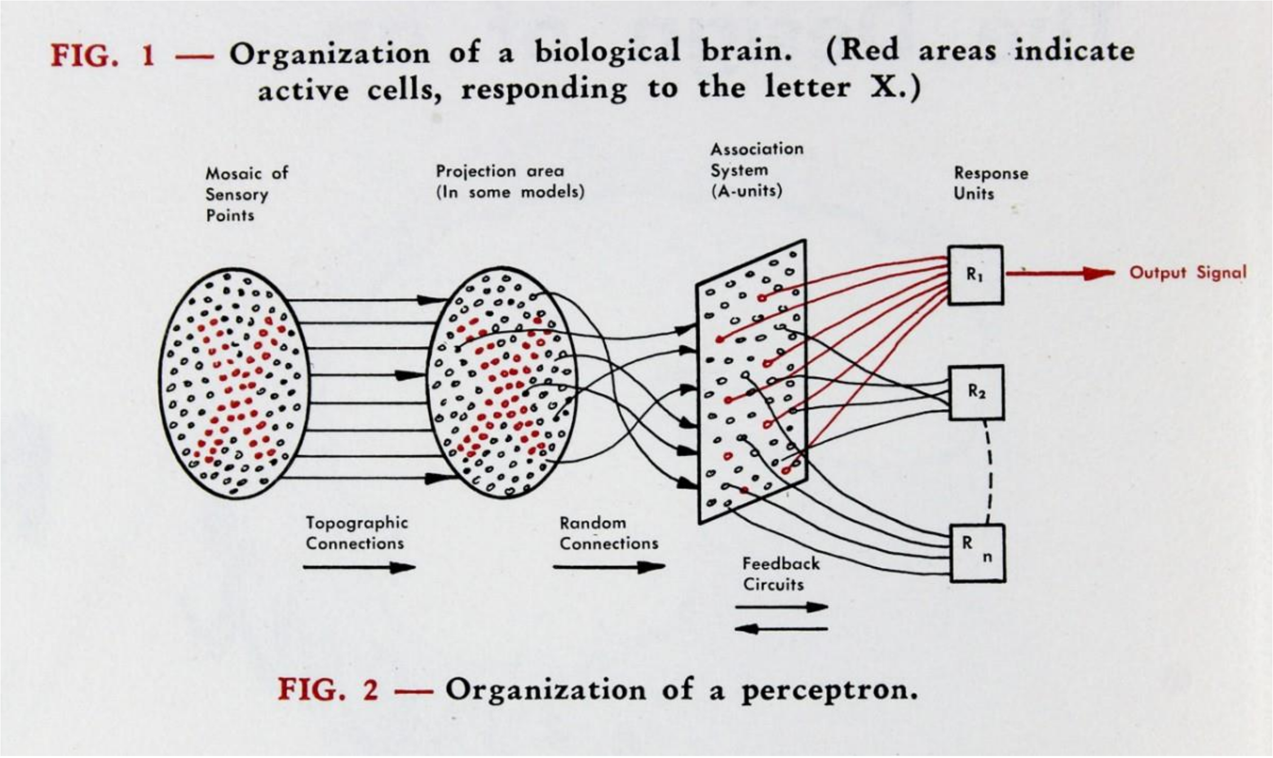
Despite recent attention, neural networks are fairly old

In 1957 Frank Rosenblatt constructed the first (single layer) neural network known as a "perceptron"



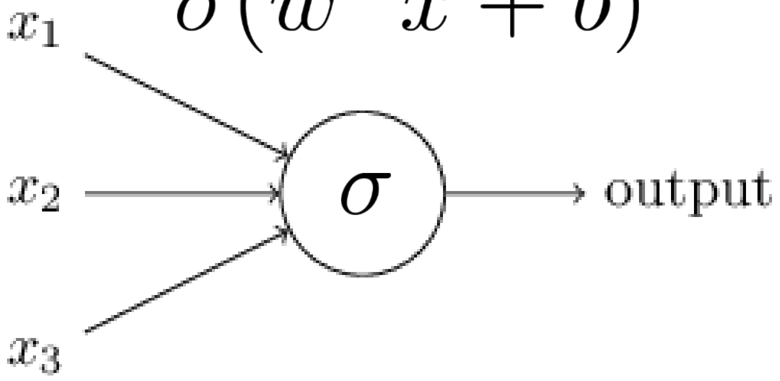
He demonstrated that it is capable of recognizing characters projected onto a 20x20 "pixel" array of photosensors

Rosenblatt's Perceptron



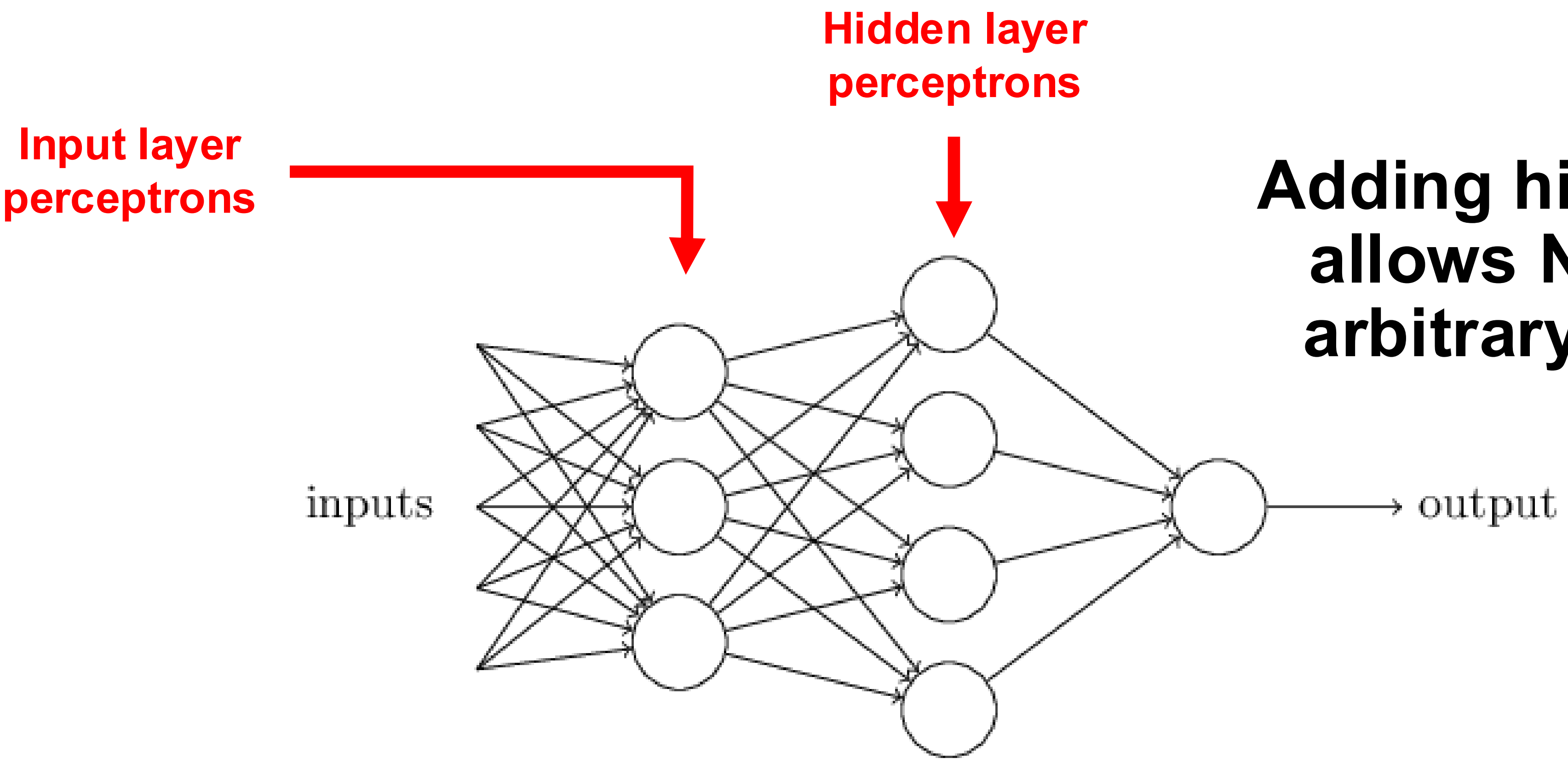
Perceptron

$$\sigma(w^T x + b)$$



- In Rosenblatt's perceptron, the inputs are tied directly to output
- "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanics" (1962)
- Criticized by Marvin Minsky in book "Perceptrons" since can only learn linearly-separable functions

Multilayer Perceptron



Adding hidden layers allows NN to learn arbitrary functions

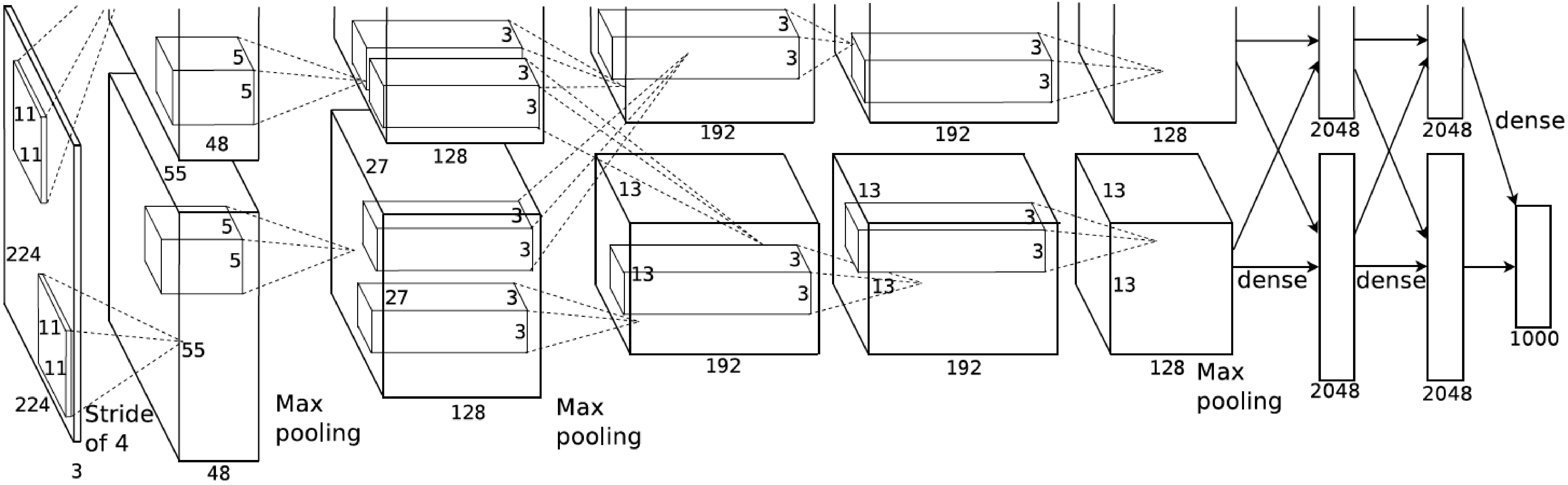
This is the quintessential *Neural Network…*

…also called *Feed Forward Neural Net* or *Artificial Neural Net*

[Source: http://neuralnetworksanddeeplearning.com]

“Deep” Neural Networks

Modern *Deep Neural networks* add many hidden layers



…and have many millions of parameters to learn

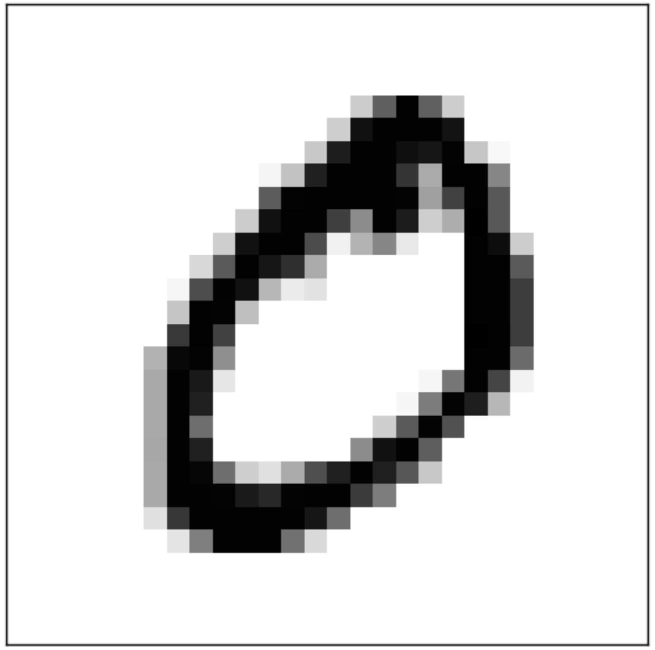
[Source: Krizhevsky et al. (NIPS 2012)]

Handwritten Digit Classification

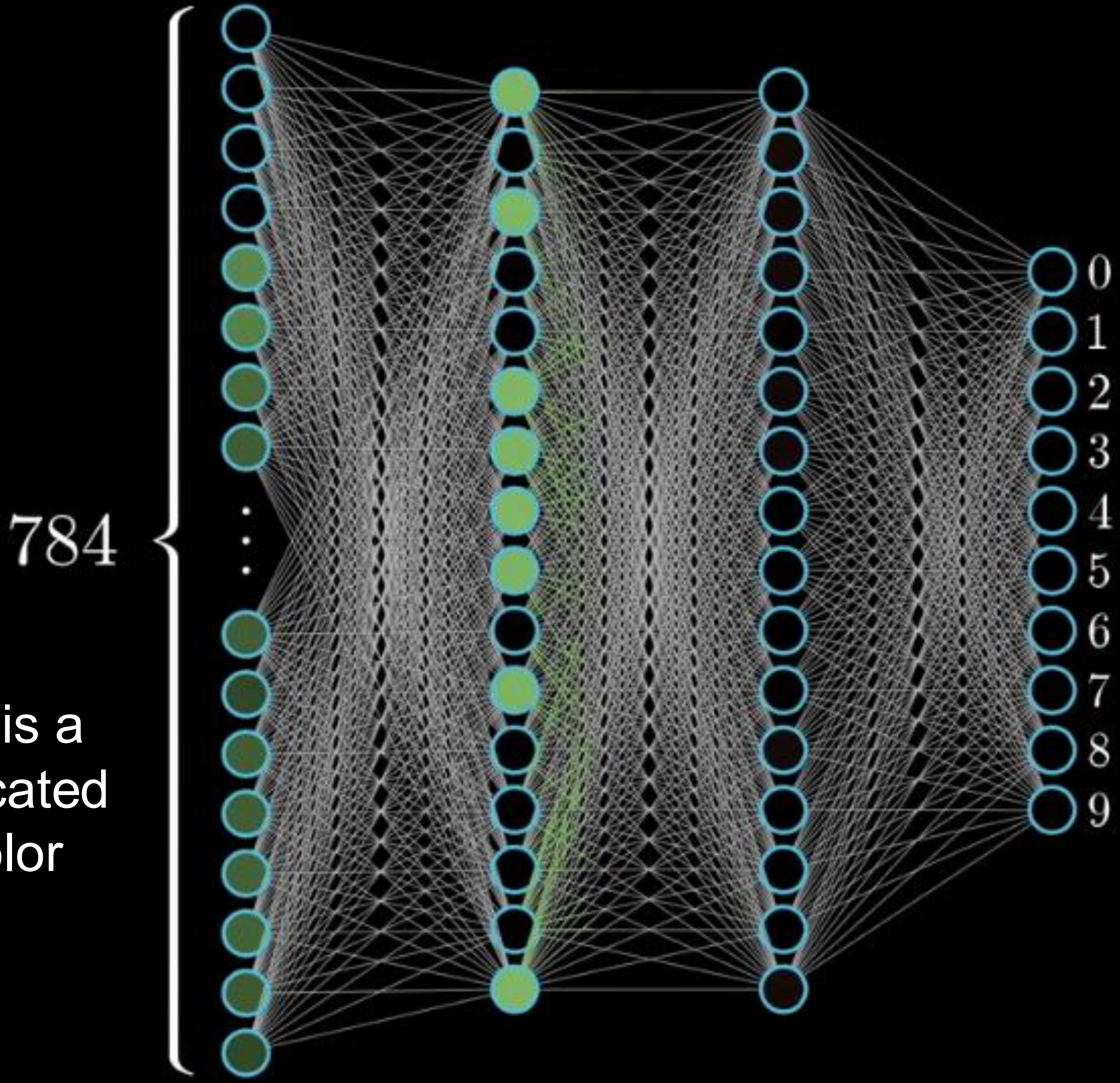
Classifying handwritten digits is the "Hello World" of NNs

Modified National Institute of Standards and Technology (MNIST) database contains 60k training and 10k test images

Each character is centered in a 28x28=784 pixel grayscale image

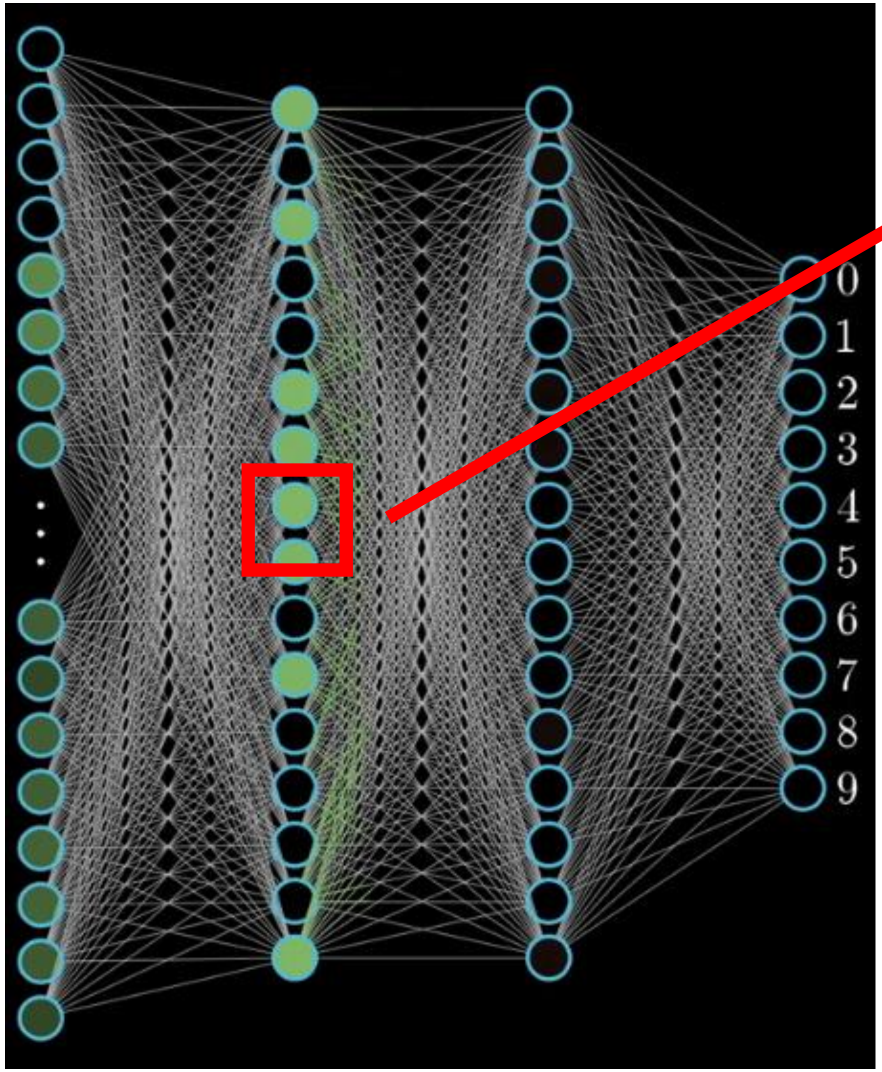


[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]



Each image pixel is a numer in [0,1] indicated by highlighted color

Feedforward Procedure



Each node computes a *weighted combination* of nodes at the previous layer…

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

Then applies a *nonlinear function* to the result

$$\sigma(w_1x_1 + w_2x_2 + \ldots + w_nx_n + b)$$

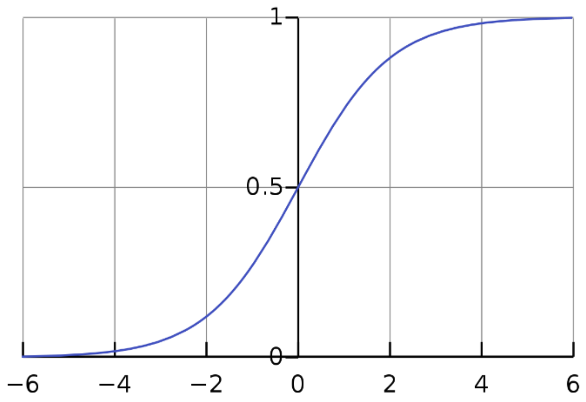
Often, we also introduce a constant *bias* parameter

Nonlinear Activation functions

We call this an *activation function* and typically write it in vector form,

$$\sigma(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b) = \sigma(w^T x + b)$$

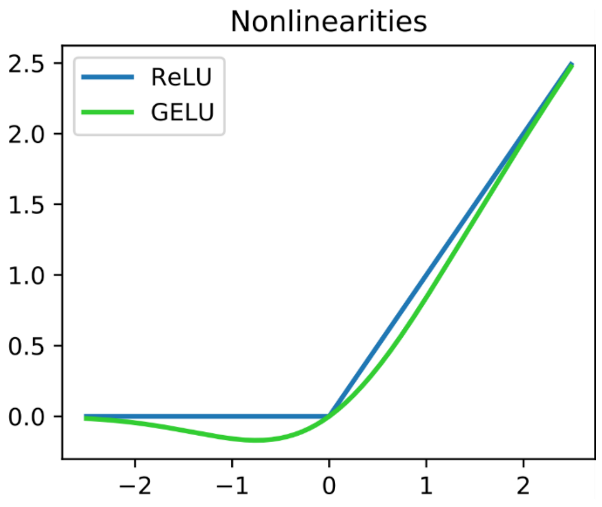
An early choice was the *logistic function*,



$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Later found to lead to slow learning and *ridge functions* like the *rectified linear unit (ReLU)*,

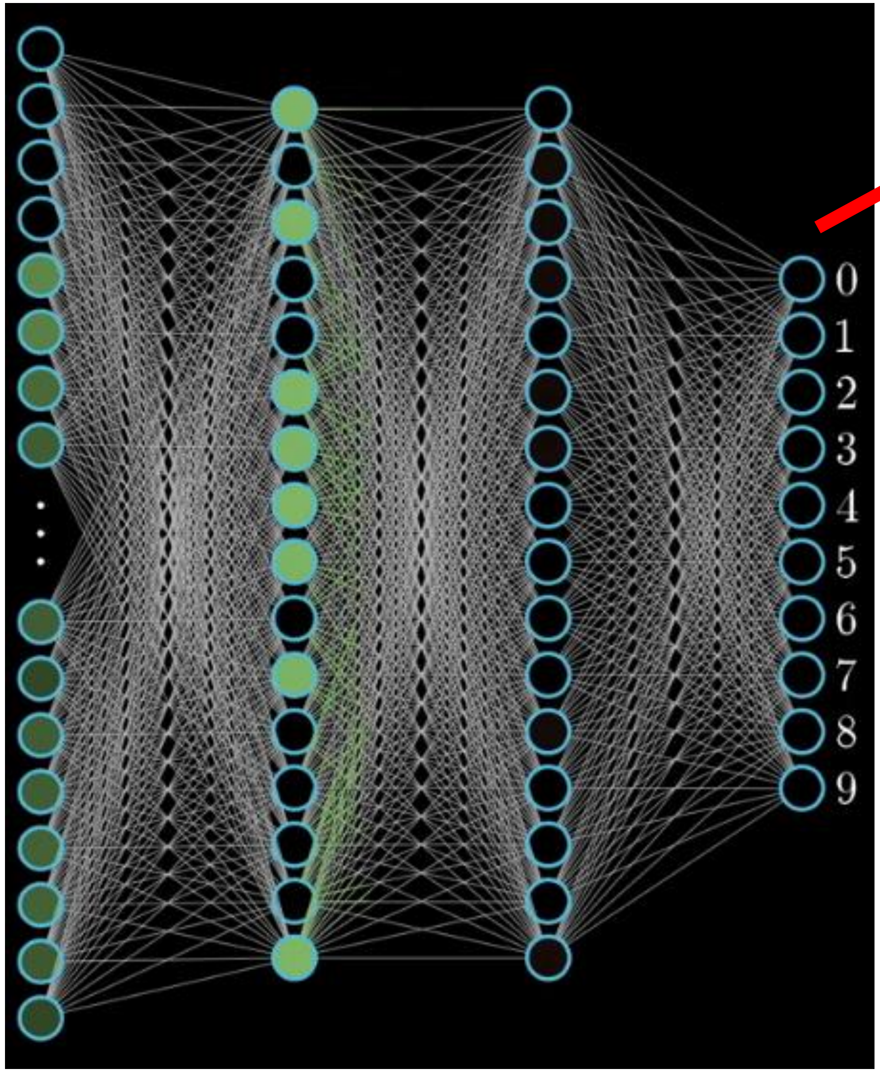
$$\sigma(w^T x + b) = \max(0, w^T x + b)$$



Or the smooth *Gaussian error linear unit (GeLU)*,

$v = w^T x + b$ $\qquad \sigma(v) = v\Phi(v)$ ← **Gaussian CDF**

Multilayer Perceptron



Final layer is typically a linear model…for classification this is a Logistic Regression

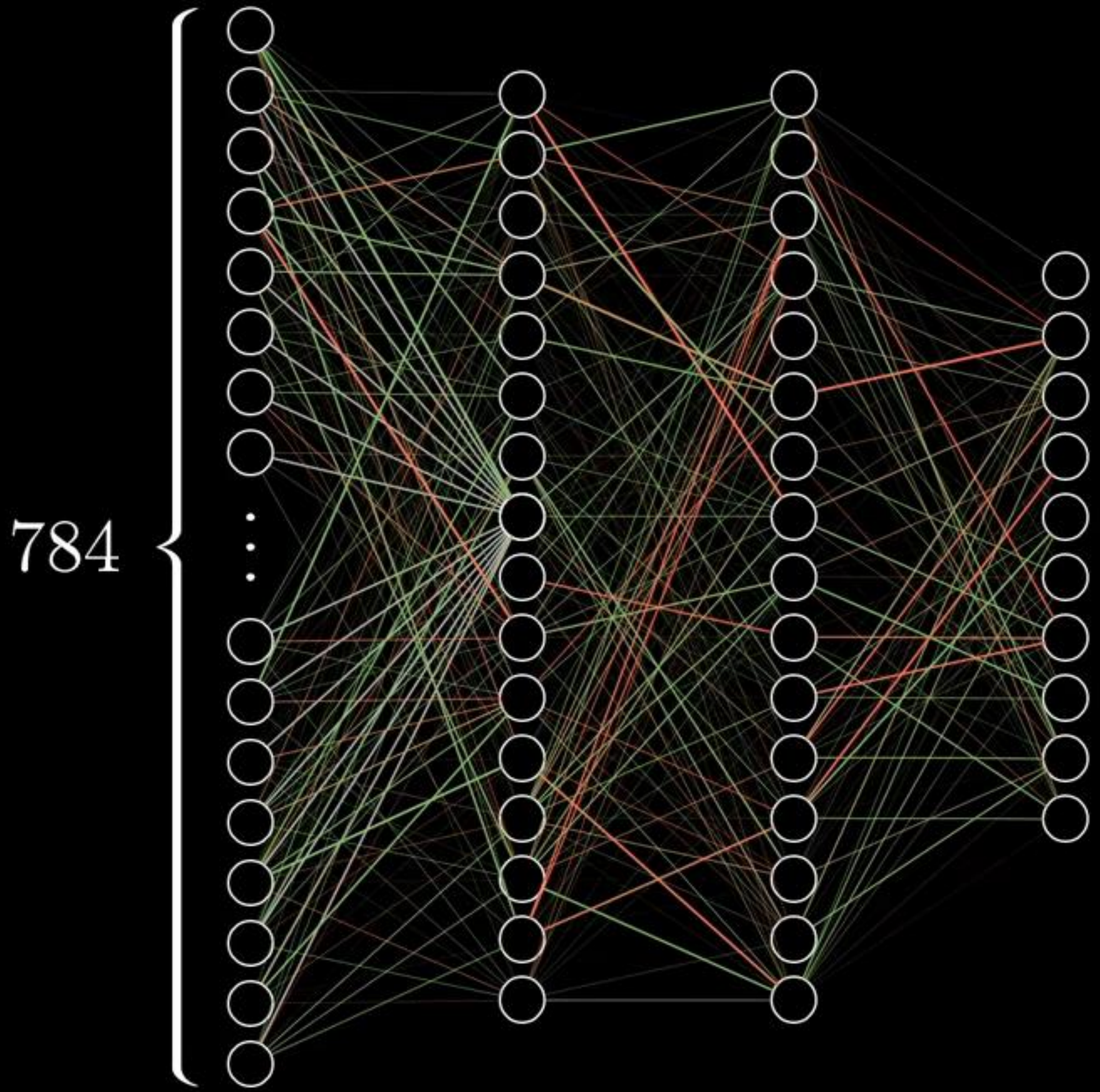
$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

Vector of activations from previous layer

Recall that for multiclass logistic regression with K classes,

$$p(\text{Class} = k \mid x) \propto \sigma(w_k^T x + b_k)$$

[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]



$$\boxed{\begin{array}{c} 784\times16+16\times16+16\times10 \\ \text{weights} \\ \\ 16\ +16+10 \\ \text{biases} \end{array}}$$

13,002

Each parameter has some impact on the output…need to tweak (learn) all parameters simultaneously to improve prediction accuracy

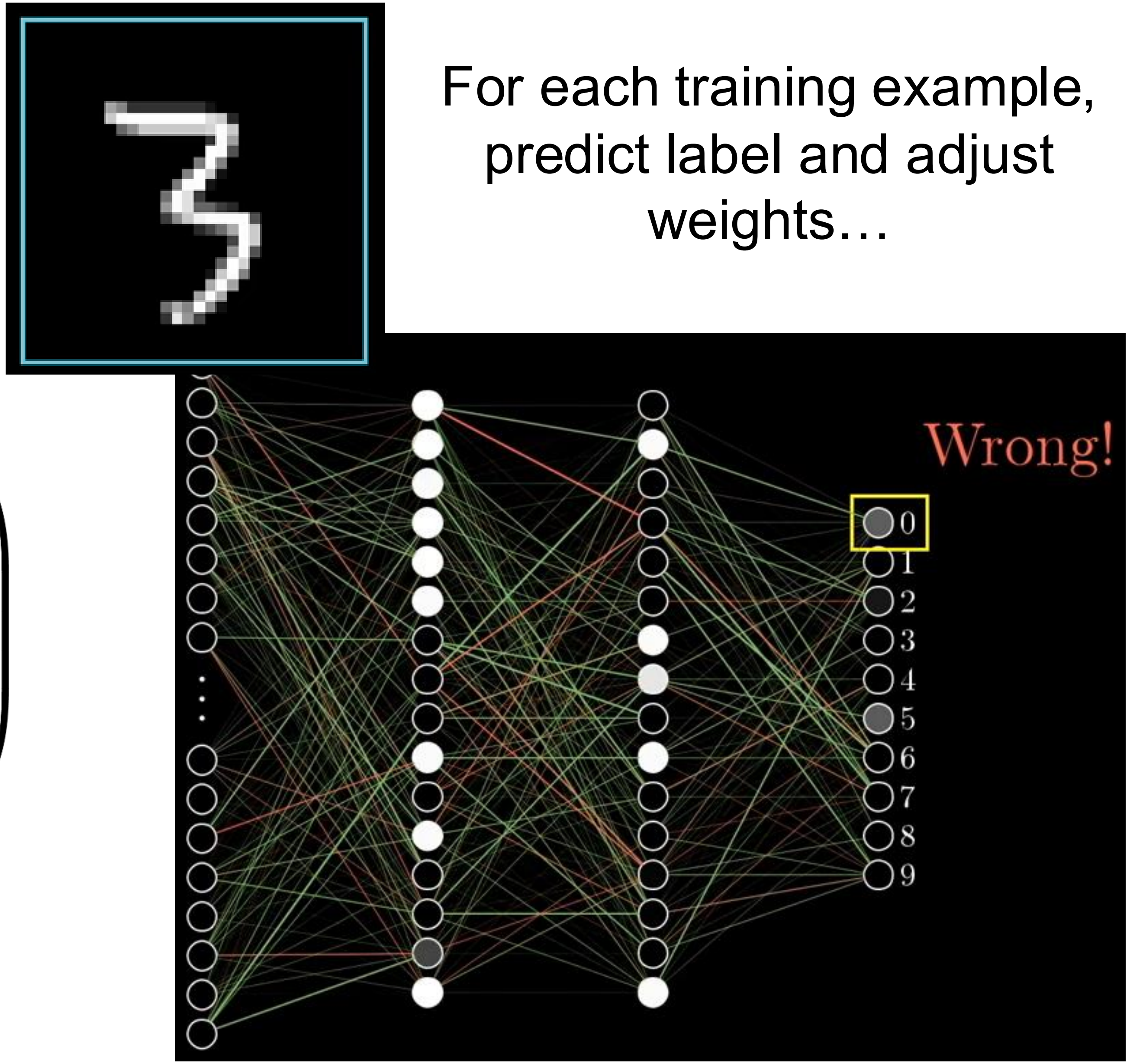
Training Multilayer Perceptron

$$X^{\text{Train}} =$$

$$Y^{\text{Train}} = \begin{pmatrix} 0 & 4 & 1 & \dots & 3 \\ 5 & 3 & 6 & \dots & 4 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 7 & 4 & 6 & \dots & 5 \end{pmatrix}$$

- How to score final layer output?
- How to adjust weights?

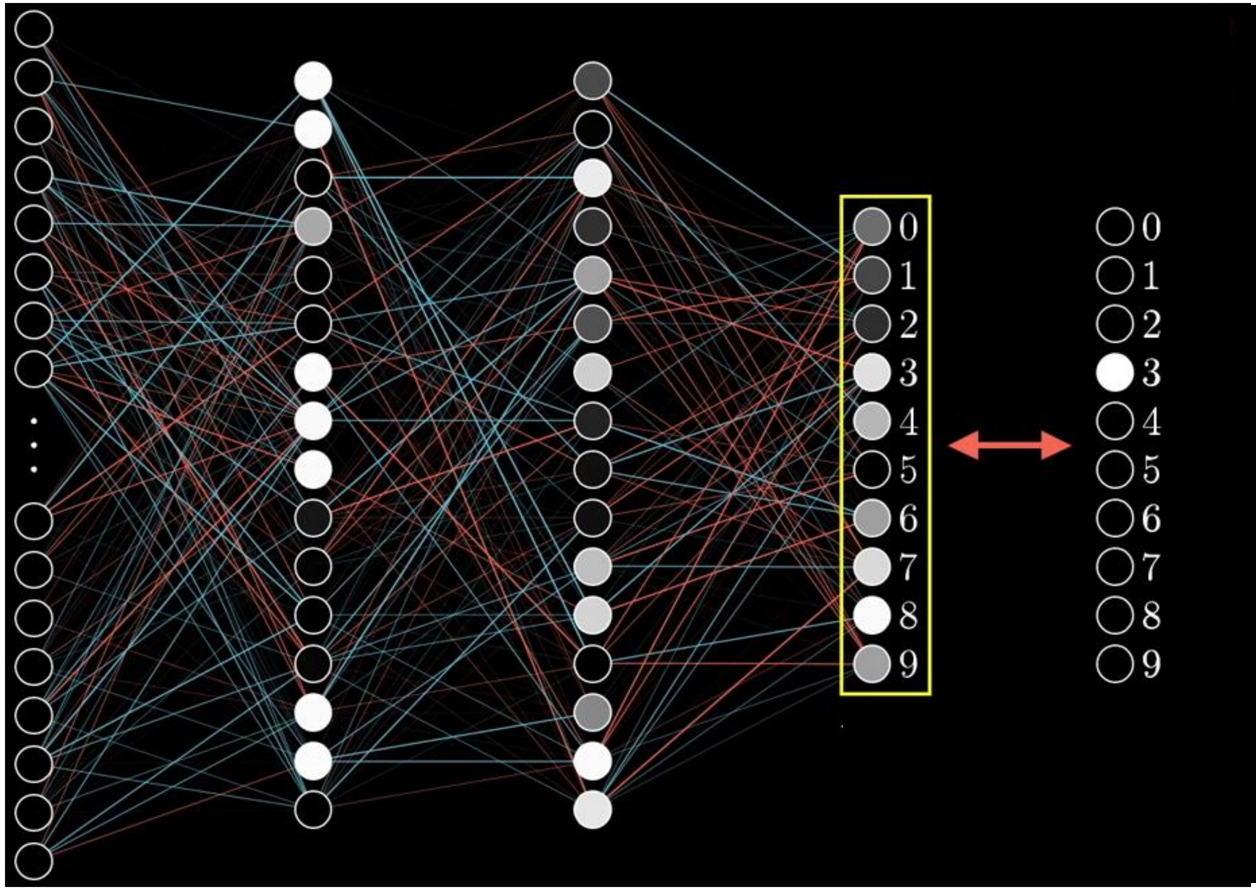
For each training example, predict label and adjust weights…



Training Multilayer Perceptron

Score based on difference between final layer and one-hot vector of true class…

Input



$$(0.43-0.00)^2+$$
$$(0.28-0.00)^2+$$
$$(0.19-0.00)^2+$$
$$(0.88-1.00)^2+$$
$$(0.72-0.00)^2+$$
$$(0.01-0.00)^2+$$
$$(0.64-0.00)^2+$$
$$(0.86-0.00)^2+$$
$$(0.99-0.00)^2+$$
$$(0.63-0.00)^2$$

[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]

Training Multilayer Perceptron

Our cost function for i[th] input is error in terms of weights / biases…

$$\underbrace{\text{Cost}_i(w_1, \ldots, w_n, b_1, \ldots, b_n)}_{\textbf{13,002 Parameters in this network}}$$

…minimize cost over all training data…

$$\min_{w,b} \mathcal{L}(w, b) = \sum_i \text{Cost}_i(w_1, \ldots, w_n, b_1, \ldots, b_n)$$

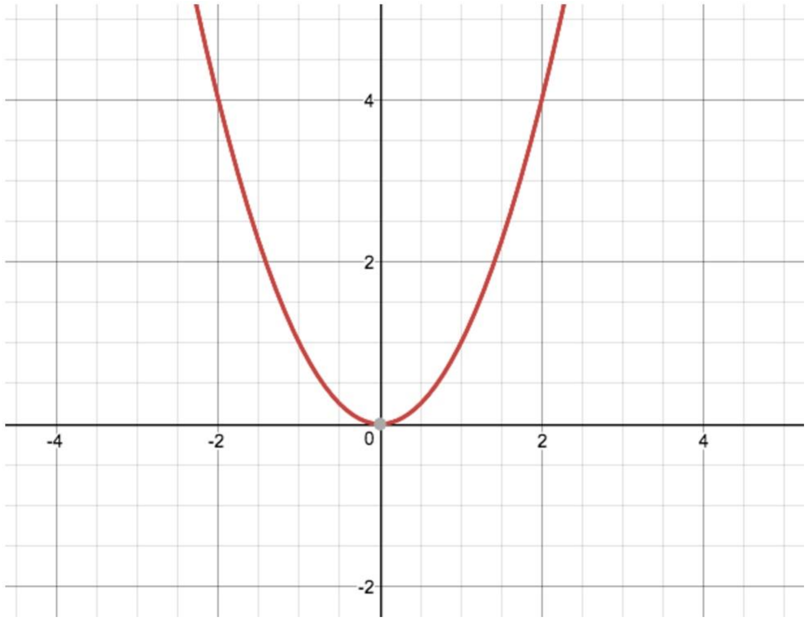
This is a super high-dimensional optimization (13,002 dimensions in this example)…how do we solve it?

Gradient descent!

Training Multilayer Perceptron

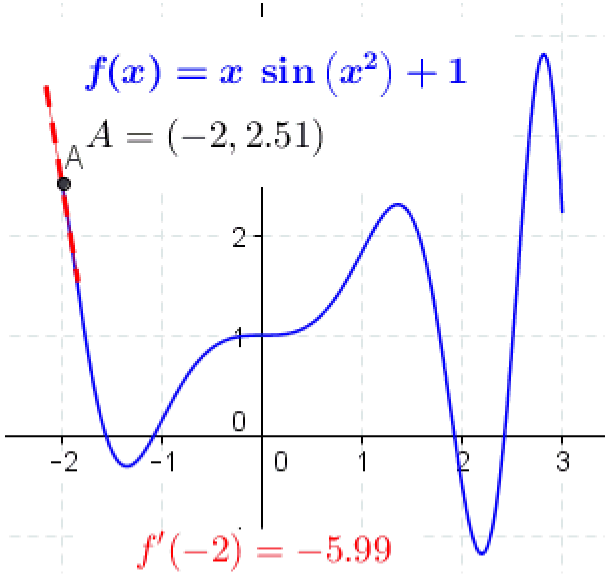
Need to find zero derivative (gradient) solution…

Convex Cost Function



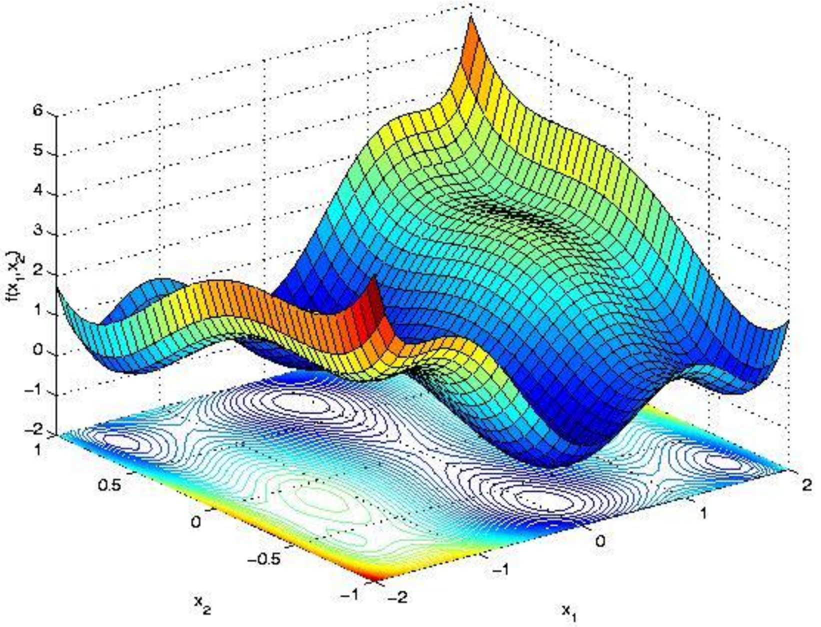
YAY!

Non-convex Cost Function



Boo!

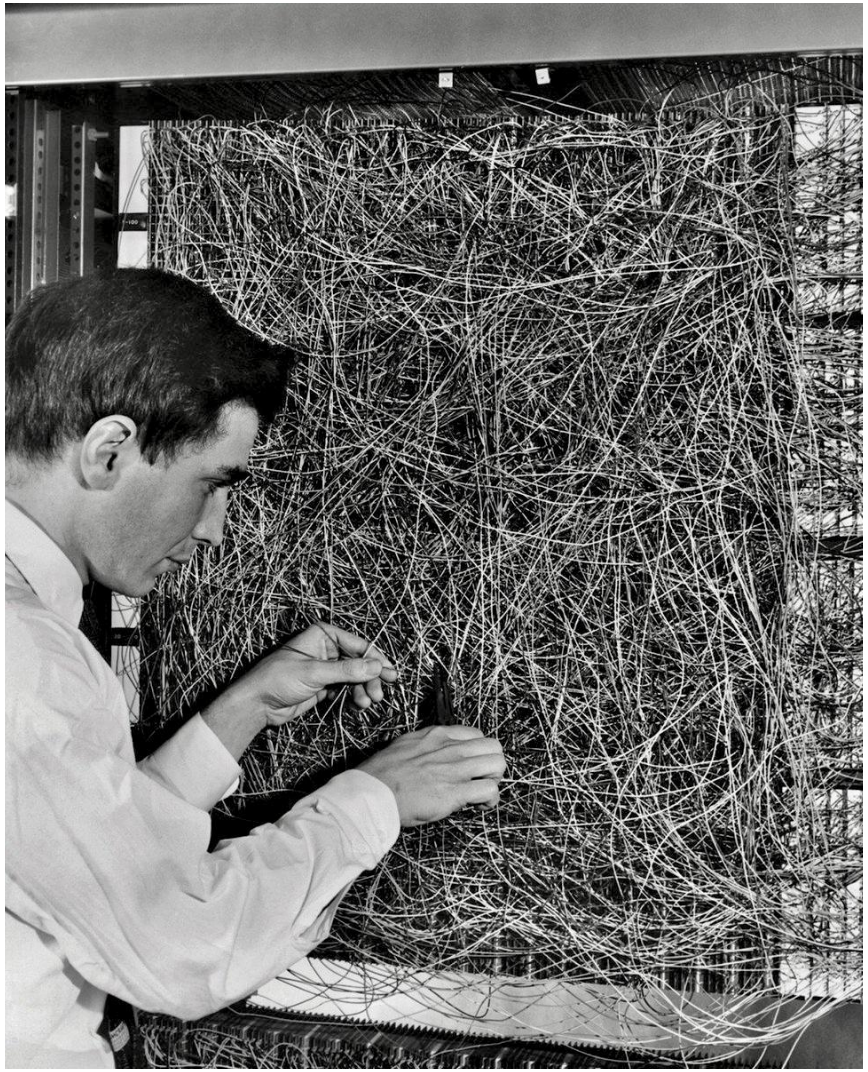
High-Dimensional Non-convex



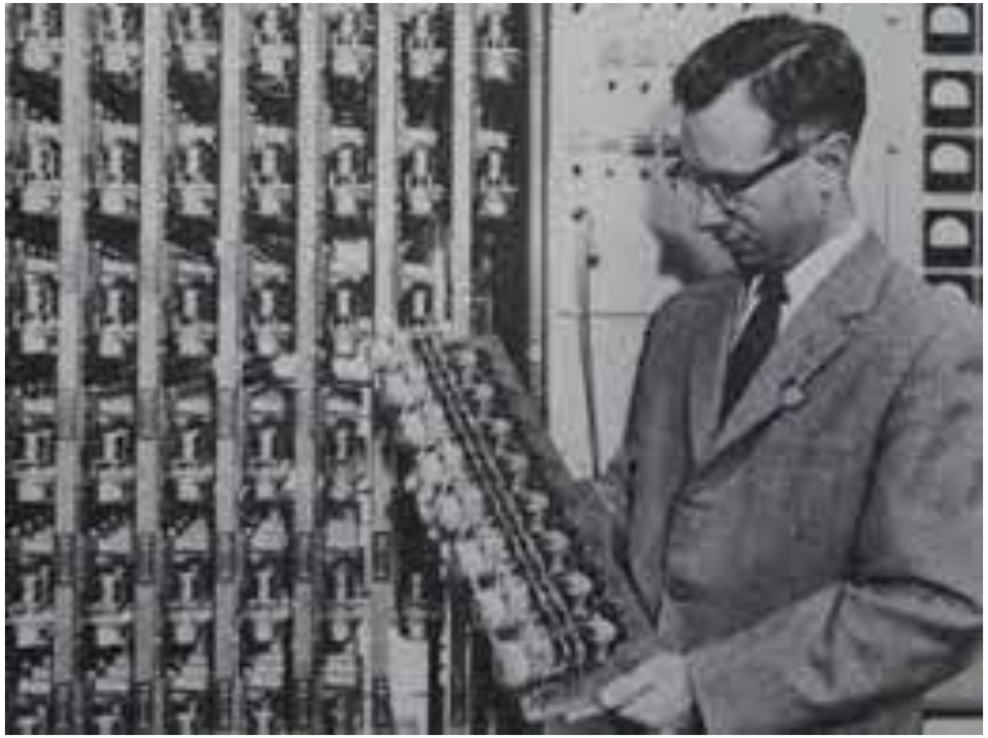
Super Boo!

Actually, the situation is much worse, since the cost is super (13,002) high dimensional…but we proceed as if…

Training the Multilayer Perceptron



Training the MLP is challenging…but it's much easier than how Rosenblatt did it



Example

Play with a small multilayer perceptron on a binary classification task…

https://playground.tensorflow.org/

Computing the Derivative

So we need to compute derivatives of a super complicated function…

$$\frac{d}{dw}\mathcal{L}(w) = \sum_i \frac{d}{dw}\mathrm{Cost}_i(w)$$

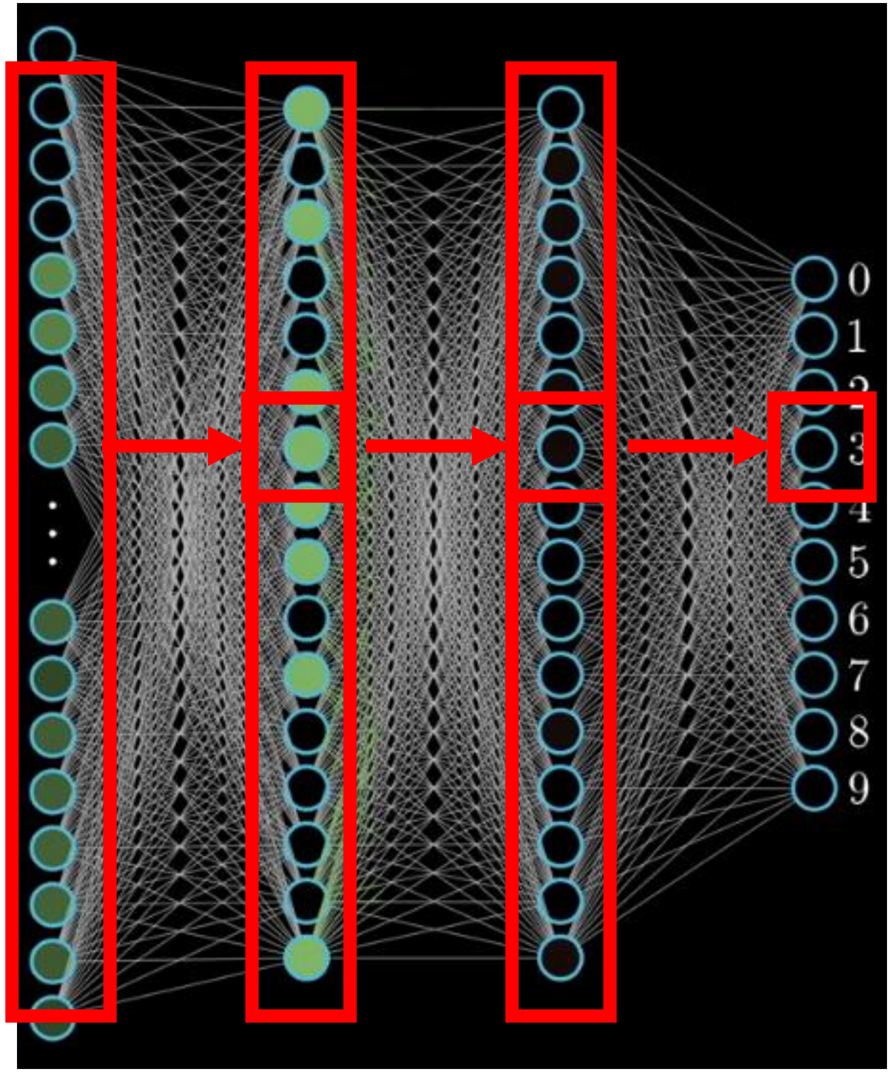
Dropped bias terms for simplicity

Recall the **derivative chain rule**

$$\frac{d}{dw}f(g(w)) = \underbrace{\frac{d}{dg(w)}f(g(w))}_{\text{Derivative of f at its argument g(w) e.g. treat g(w) as a variable}} \underbrace{\left(\frac{d}{dw}g(w)\right)}_{\text{Differentiate g with respect to w}}$$

Backpropagation

[Source : 3Blue1Brown : https://www.youtube.com/watch?v=aircAruvnKk]



Activation at final layer involves weighted combination of activations at previous layer…

$$\sigma(w^T x)$$

Which involves a weighted combination of the layer before it…

$$\sigma(w_n^T \sigma(w_{n-1}^T x))$$

And so on…

$$\sigma(w_n^T \sigma(w_{n-1}^T \sigma(w_{n-2}^T \sigma(\ldots))))$$

Backpropagation

Backpropagation is the procedure of repeatedly applying the derivative chain rule to compute the full derivative

Example

$$\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$$

$$\frac{d}{dz}\sigma(\sigma(z)) = \sigma(\sigma(z))(1 - \sigma(\sigma(z)))\frac{d}{dz}\sigma(z)$$

This is simply the derivative chain rule applied through the entire network, from the output to the input

Backpropagation

- Implementation-wise all we need is a function that computes the derivative of each nonlinear activation

- We can repeatedly call this function, starting at the end of the network and moving backwards

- In practice, neural network implementations use *auto differentiation* to compute the derivative on-the-fly

- Can do this efficiently on *graphical processing units (GPUs)* on extremely large training datasets

Universal Approximation Theorem

(Informally) For *any* function $f(x)$ there exists a multilayer perceptron that approximates $f(x)$ with arbitrary accuracy.

- Specific cases for arbitrary depth (number of hidden layers) and arbitrary width (number of nodes in a layer)
- Not a constructive proof (doesn’t guarantee you can learn parameters)
- Corollary : The multilayer perceptron is a *universal turing machine*
- Also means it can easily overfit training data (regularization is critical)

Outline

- Artificial Neural Network (ANN) : A Review
- Shortcomings of Standard Deep Learning
- Bayesian Deep Learning

Some Drawbacks of Standard Deep Learning

- Predictions can be “brittle” (i.e. very discontinuous w.r.t. input)
- Fail to generalize outsize training data (regularization important)
- Difficult to tune learning procedure
- Unable to accurately quantify uncertainty over predictions
- Lack privacy (memorize training data)
- Lack interpretability (models are “black box”)
- Pose safety issues in critical applications

How many r’s are in the word “strawberry”?

There are 2 r's in the word "strawberry".

Wrong

I apologize for the error in my previous response. You are correct to point out that I was mistaken. Let me correct that:

There is actually only 1 'r' in the word "strawberry".

Thank you for bringing this to my attention. I always aim for accuracy, and I appreciate the opportunity to provide the correct information.

Claude
3.5 Sonnet

Wrong

I apologize again for the confusion. You're right to point out that I'm still incorrect. Let me take a moment to carefully count the letters:

s-t-r-a-w-b-e-r-r-y

After a careful recount, I can now state with certainty:

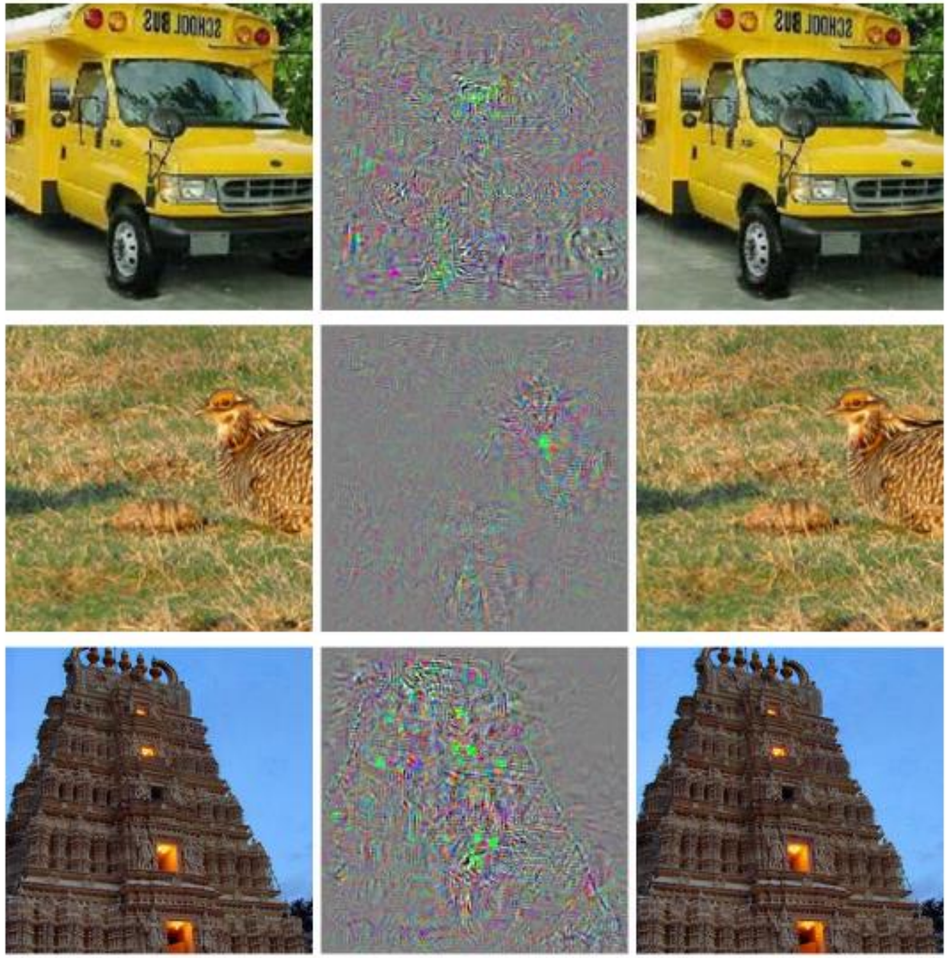
There are 2 'r's in the word "strawberry".

This is the correct answer. The first 'r' appears after the 't', and the second 'r' is the second-to-last letter of the word. Thank you for your patience as we arrived at the accurate answer. If you have any more questions or need clarification on anything else, please don't hesitate to ask.

Claude
3.5 Sonnet

Brittleness : Discontinuities in Predictions

Nearly imperceptible changes to input change prediction

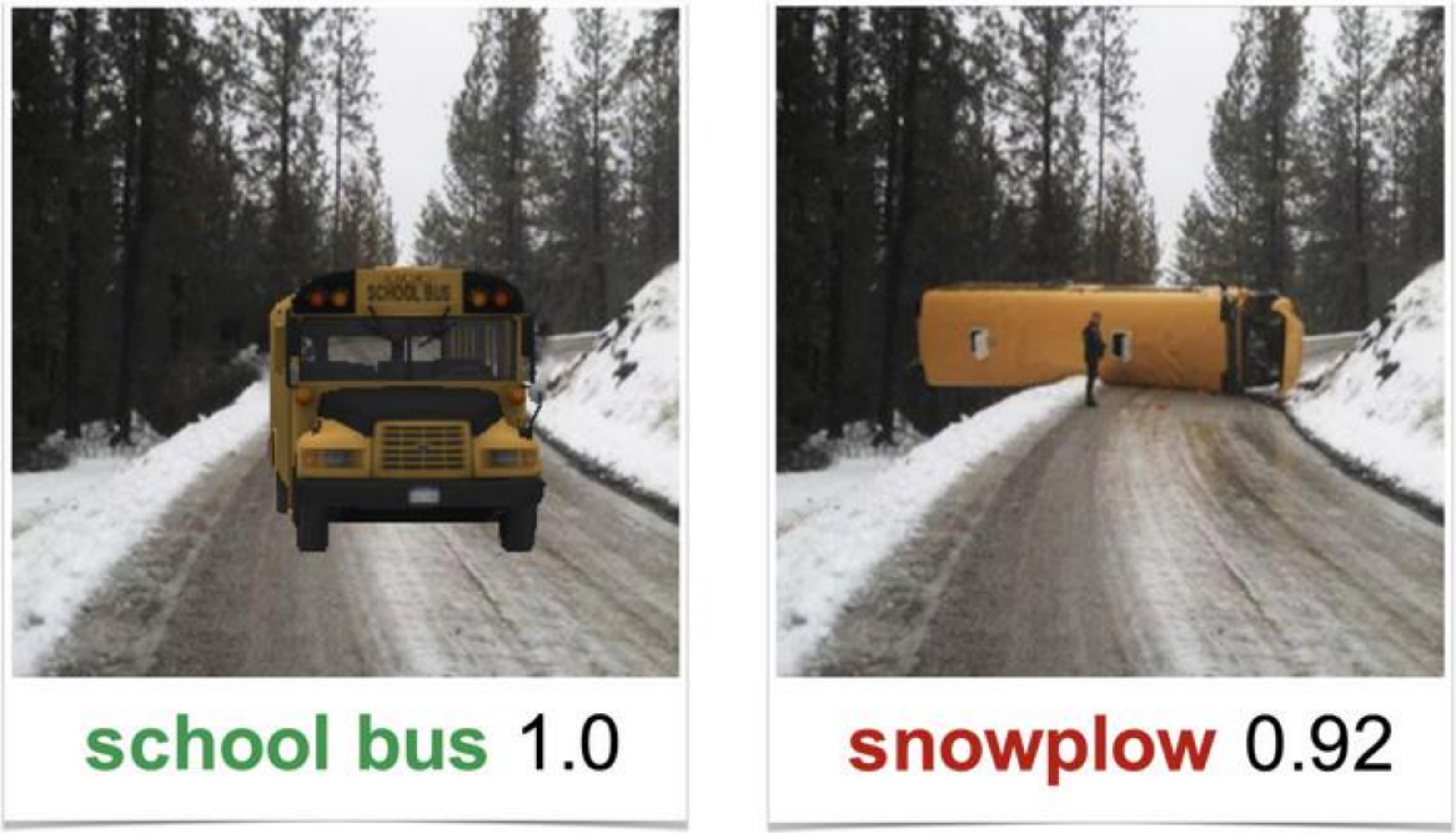


All images in right column predicted as "ostrich"

[Szegedy et al. "Intriguing properties of neural networks." ICLR 2014](#)

Deep Learning Has No Common Sense

They often fail to generalize to novel instances



[Alcorn et al. "Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects." CVPR 2019]

Safety Concerns

Variety of black-box *physical attacks* left-to-right*:*

- Artistic graffiti
- Subtle graffiti
- Poster

Can reliably cause ANN to misclassify as intended target (e.g. speed limit 45mph)

Does not require knowledge of network internals

[Evtimov et al. “Robust Physical-World Attacks on ML Models.” 2017]

Deep Learning is Opaque

It is often unclear <u>what</u> a DL system has learned

- This can be dangerous!
  - Biases and bugs may exist
  - Only discovered when it’s too late!
- Important real-world applications must account for this
  - Diagnosing medical patients
  - Self-driving cars
  - Safety-critical systems
  - Etc.

Privacy Concerns

Large DNNs capable of memorizing training data…

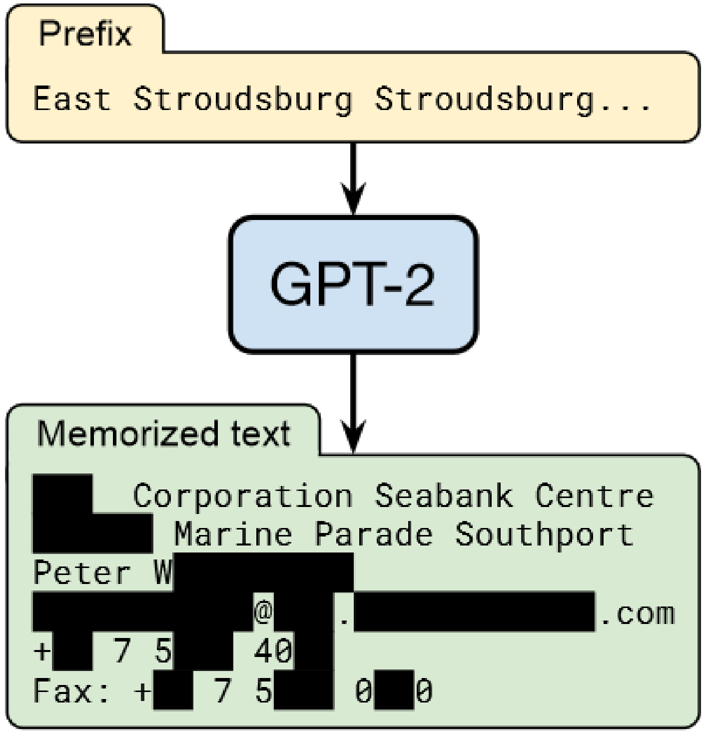


Figure 1: **Our extraction attack.** Given query access to a neural network language model, we extract an individual person's name, email address, phone number, fax number, and physical address. The example in this figure shows information that is all accurate so we redact it to protect privacy.

Carlini et al. demonstrate that training data can be recovered from GPT-2, a large language model…

…this can be done in a black-box manner (i.e. without knowledge of network internals)

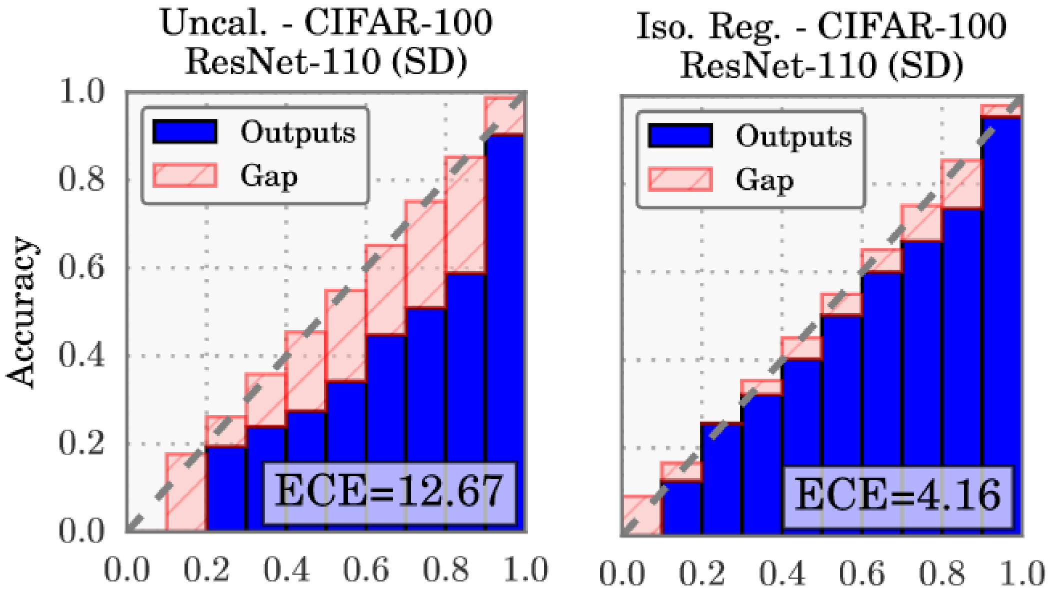
*** Carlini et al. "Extracting training data from large language models." USENIX 2021*

Outline

- Artificial Neural Network (ANN) : A Review
- Shortcomings of Standard Deep Learning
- Bayesian Deep Learning

Uncertainty Quantification

- Many of the shortcomings of DL can be addressed by quantifying uncertainty
- Uncertainty comes in a variety of forms:
  - Uncertainty that can be eliminated with more training data (epistemic)
  - Uncertainty that is inherent in the stochastic process (aleotoric)
- Preliminary work aims to calibrate uncertainty in the prediction layer (e.g. softmax) via “network uncertainty calibration”



(left) Before calibration (right) after calibration on CIFAR-100 image classification task

[Guo et al. “On Calibration of Modern Neural Networks.” NeurIPS. 2017](#)

Probabilistic Perspectives on Deep learning

DNNs typically provide a deterministic mapping of inputs-to-predictions:

Prediction $\rightarrow$ $$y = f_\theta(x)$$ $\leftarrow$ **Input**

Network Parameters: Weights, architecture, activation funcs

Can extend this to *discriminative* probability model relatively easily:

$$p(y \mid x, \theta)$$

- E.g. use 2nd-to-last softmax layer as PMF (bad idea)
- Use networks to parameterize parametric density

$$p(y \mid x, \theta) = \mathcal{N}(y \mid \mu_\theta(x), \Sigma_\theta(x))$$

ANN outputs

Bayesian Perspective on Deep Learning

Idea Treat parameters as random variables with prior $\theta \sim p(\theta)$ to define *generative model*:

$$p(\theta, y \mid x)$$

Think of this as a prior over models

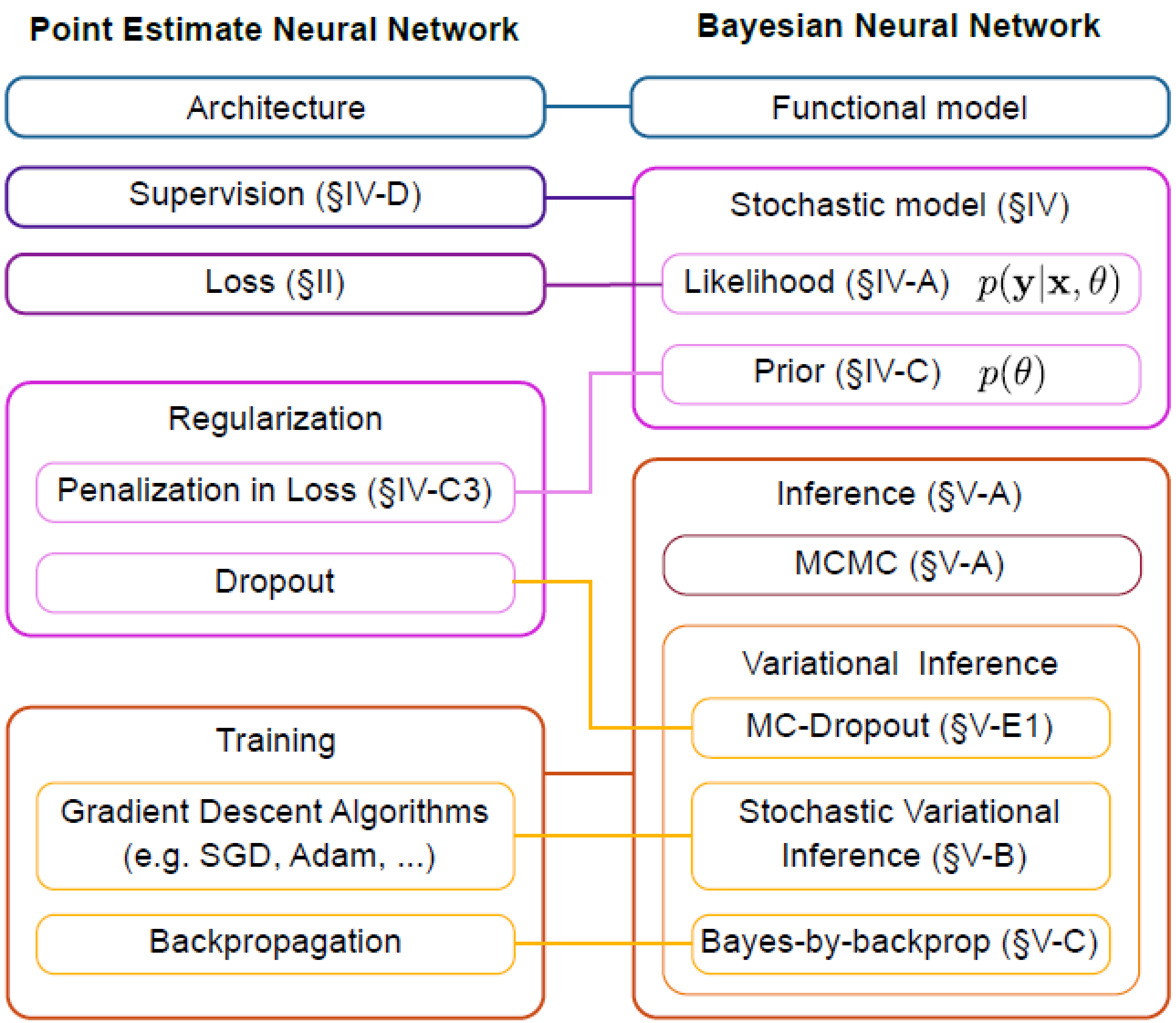
Benefits

- Can compute posterior over all networks $p(\theta \mid x)$
- Or marginalize over network parameters $p(y \mid x) = \int p(\theta, y \mid x)\, d\theta$
- Natural approach to quantify uncertainty over network and/or prediction
- Distinguish between *epistemic* and *aleotoric* uncertainty*
- There is *always* a prior…Bayesian methods just make it explicit

* Der Kiureghian and Ditlevsen. "Aleatory or epistemic? Does it matter?." *Structural safety* (2009)

* Kendall and Gal. "What uncertainties do we need in Bayesian deep learning for computer vision?." *NeurIPS*. (2017)

Point Estimate vs. Bayesian DL Correspondence



The learning process of Bayesian DL fundamentally differs from point estimate ANNs

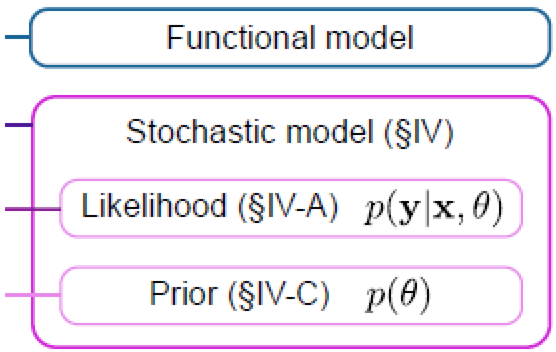
Instead of minimizing a loss function, Bayesian DL does inference via MCMC, Variational, etc.

Online prediction often requires inference (unless amortized inference is done)

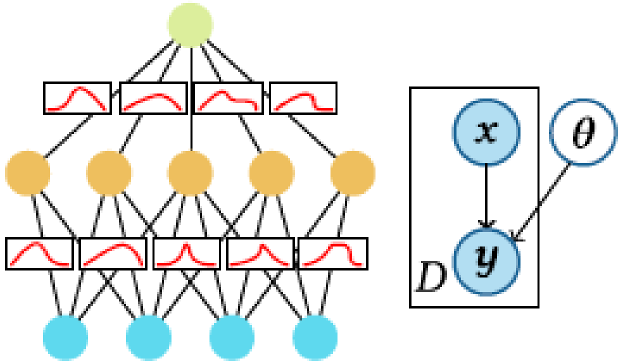
Source: Jospin et al. “Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users.” IEEE Comp. Intell. Mag. (2022)

Bayesian Neural Network

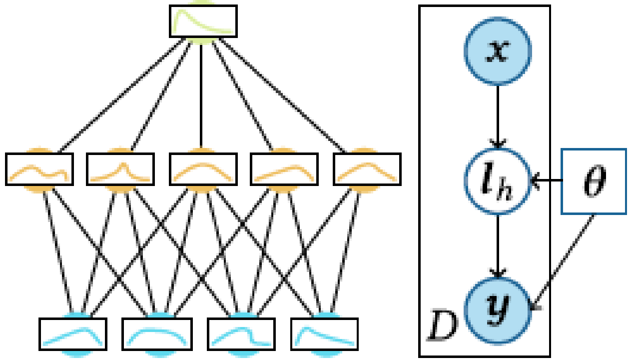
Bayesian Neural Network



Stochastic Weights



Stochastic Nodes



- Both standard ANN and BNN require functional model
- BNN additionally requires stochastic model (likelihoods, priors)
- Stochastic model depends on whether weights or nodes are random
- Either choice determines structure of the underlying PGM

Bayesian Neural Network

Many different constructions, but all essentially a stochastic ANN

An ANN construction with parameters $\theta = (W, b)$:

$$\begin{aligned} \boldsymbol{l}_0 &= \boldsymbol{x}, \\ \boldsymbol{l}_i &= s_i(\boldsymbol{W}_i \boldsymbol{l}_{i-1} \ + \ \boldsymbol{b}_i) \quad \forall i \in [1, n] \\ \boldsymbol{y} &= \boldsymbol{l}_n. \end{aligned}$$

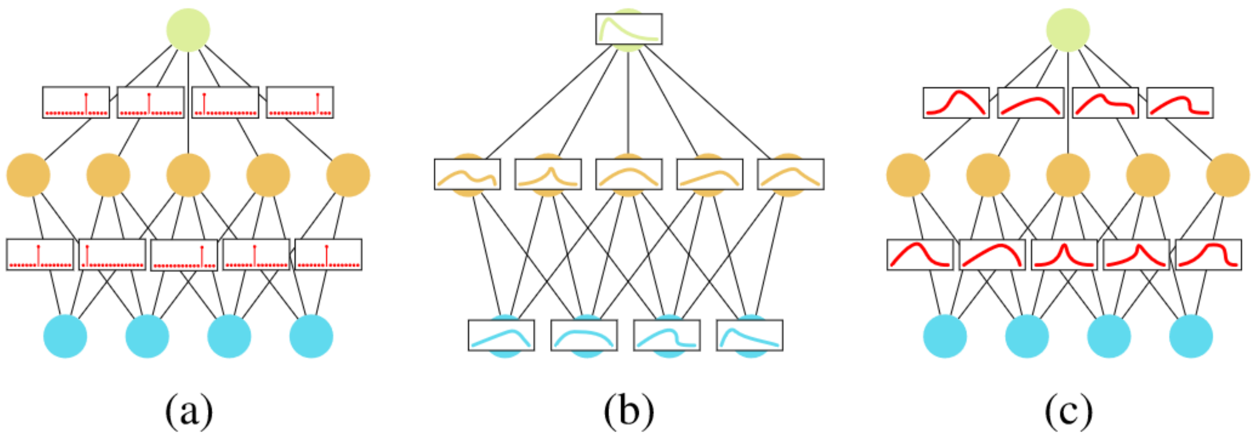


Fig. 3: (a) Point estimate neural network, (b) stochastic neural network with a probability distribution for the activations, and (c) stochastic neural network with a probability distribution over the weights.

Two main types of BNNs

- *Add stochastic activations at nodes*
- *Make parameters random (add priors)*

Source: Jospin et al. "Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users." IEEE Comp. Intell. Mag. (2022)

Bayesian Neural Network

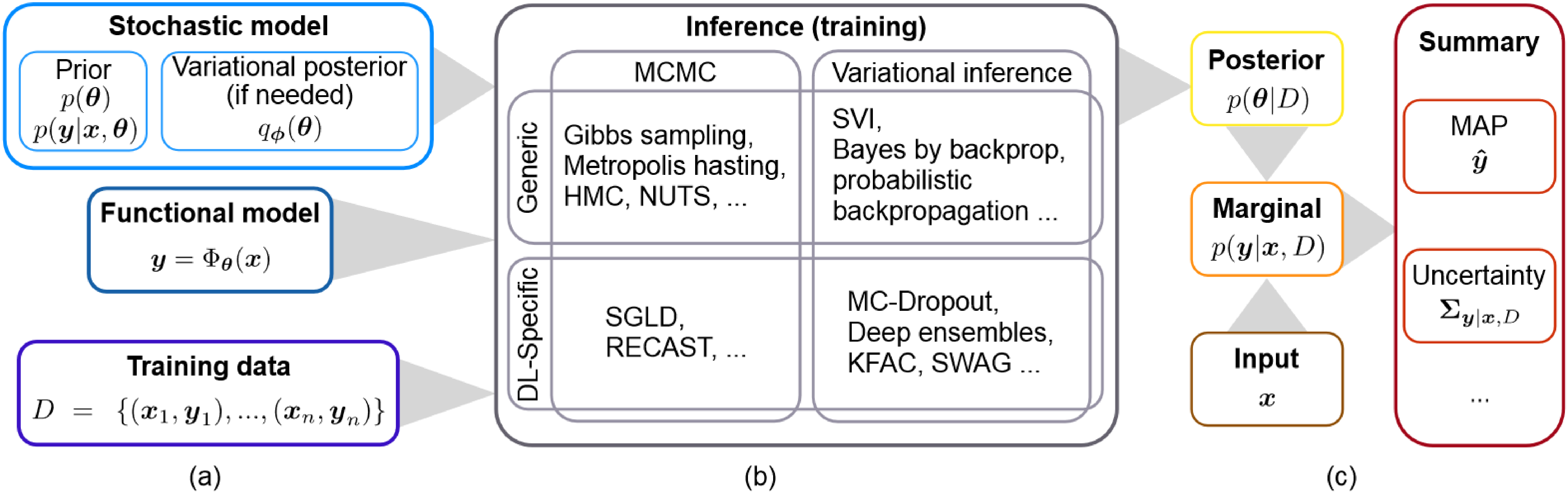


Fig. 2: Workflow to design (a), train (b) and use a BNN for predictions (c).

Source: Jospin et al. “Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users.” IEEE Comp. Intell. Mag. (2022)

Inference in a BNN

Given training data $D=\{Dx,Dy\}$ compute posterior over network params,

$$p(\boldsymbol{\theta}|D) = \frac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}},\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}},\boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'} \propto p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}},\boldsymbol{\theta})p(\boldsymbol{\theta}).$$

- Represents distribution over all possible networks based on training data
- In general restricted to a subclass, i.e. fixed architecture / activations
- Parameters are typically network weights
- Inference is intractable in general, need look at algorithms we've learned

Prediction in a BNN

When predicting we often marginalize over network parameters,

$$p(\boldsymbol{y}|\boldsymbol{x}, D) = \int_{\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta'})p(\boldsymbol{\theta'}|D)d\boldsymbol{\theta'}.$$

Marginal $p(y \mid x, D)$ characterizes predictive uncertainty of the network.

Given samples from posterior,

$$\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|D);$$

Can sample predictions in feedforward process,

$$\boldsymbol{y}_i = \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x});$$

Algorithm 1 Inference procedure for a BNN.

Define $p(\boldsymbol{\theta}|D) = \dfrac{p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_{\boldsymbol{y}}|D_{\boldsymbol{x}}, \boldsymbol{\theta'})p(\boldsymbol{\theta'})d\boldsymbol{\theta'}}$;

for $i = 0$ **to** N **do**

  Draw $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|D)$;

  $\boldsymbol{y}_i = \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x})$;

end for

return $Y = \{\boldsymbol{y}_i | i \in [0, N)\},\ \Theta = \{\boldsymbol{\theta}_i | i \in [0, N)\}$;

Training Data

Training Labels

Prediction in a BNN

Approach generates a set of predictions from an ensemble of networks,

$$Y = \{\boldsymbol{y}_i | i \in [0, N)\}, \;\; \Theta = \{\boldsymbol{\theta}_i | i \in [0, N)\};$$

Can use **model averaging** for a single prediction,

$$\hat{\boldsymbol{y}} = \frac{1}{|\Theta|} \sum_{\boldsymbol{\theta}_i \in \Theta} \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}).$$

Sample covariance can be used to quantify predictive uncertainty,

$$\boldsymbol{\Sigma}_{\boldsymbol{y}|\boldsymbol{x},D} = \frac{1}{|\Theta|-1} \sum_{\boldsymbol{\theta}_i \in \Theta} \left(\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right) \left(\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right)^{\intercal}.$$

Better uncertainty estimates are possible (e.g. predictive entropy)

Prediction in a BNN

One can also consider the empirical distribution over predictions,

$$\hat{\boldsymbol{p}} = \frac{1}{|\Theta|} \sum_{\boldsymbol{\theta}_i \in \Theta} \Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}).$$

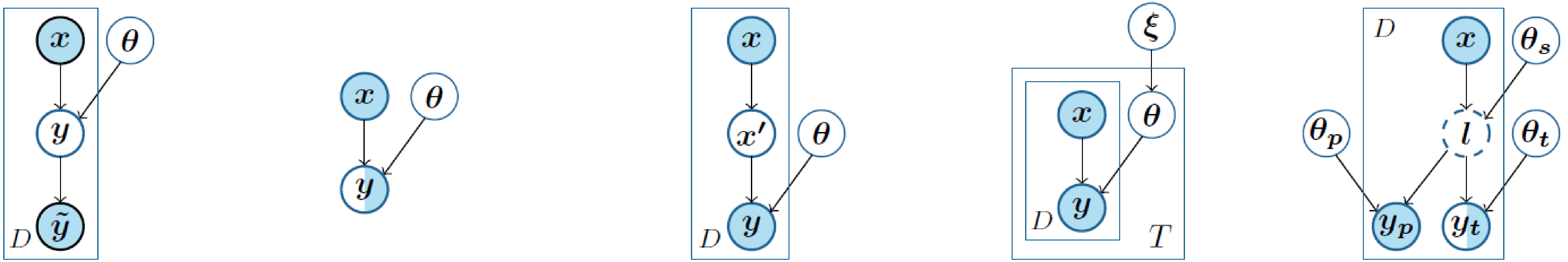
The maximum a posteriori (MAP) prediction is then,

$$\hat{\boldsymbol{y}} = \arg\max_i p_i \in \hat{\boldsymbol{p}}.$$

- Uncertainty given via the empirical entropy
- Straightforward for classification tasks
- Continuous (i.e. regression) predictions require density estimation

Generalizing Beyond Supervised Learning

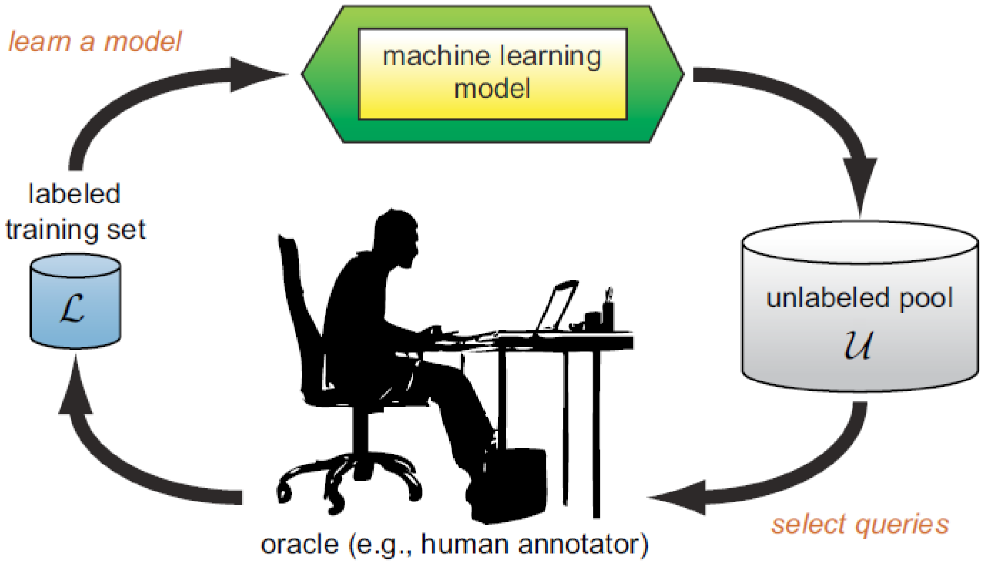
Bayesian DL can effectively use unlabeled data and uncertain labels…



(a) Noisy labels (b) Semi-supervised learning (c) Data augmentation (d) Meta-learning (e) Self-supervised learning

- **Noisy Labels** Annotations can be imprecise
- **Semi-Supervised** Use, both, labeled and unlabeled training data
- **Augmentation** Transformations of inputs that do not change label
- **Meta-Learning** Learn how to learn
- **Self-Supervised** Labels are directly obtained from inputs, but do not relate to the task…need to learn a proxy task

Active Learning in a BNN



Data annotation is expensive…

…uncertainty over prediction allows us to be smart about what data we need to label

Algorithm 2 Active learning loop with a BNN.

while $U \neq \varnothing$ and $\Sigma_{\boldsymbol{y}|\boldsymbol{x}_{max},D} <$ threshold and $C <$ MaxC **do**
  Draw $\Theta = \{\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}|D) | i \in [0, N)\}$;
  for $\boldsymbol{x} \in U$ **do**
    $\Sigma_{\boldsymbol{y}|\boldsymbol{x},D} = \frac{1}{|\Theta|-1} \sum_{\boldsymbol{\theta}_i \in \Theta} \left(\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right) \left(\Phi_{\boldsymbol{\theta}_i}(\boldsymbol{x}) - \hat{\boldsymbol{y}}\right)^{\mathsf{T}}$;
    if $\Sigma_{\boldsymbol{y}|\boldsymbol{x},D} > \Sigma_{\boldsymbol{y}|\boldsymbol{x}_{max},D}$ **then**
      $\boldsymbol{x}_{max} = \boldsymbol{x}$;
    end if
  end for
  $D_{\boldsymbol{x}} = D_{\boldsymbol{x}} \cup \{\boldsymbol{x}_{\max}\}$;
  $D_{\boldsymbol{y}} = D_{\boldsymbol{y}} \cup \{\text{Oracle}(\boldsymbol{x}_{\max})\}$;
  $U = U \setminus \{\boldsymbol{x}_{\max}\}$;
  $C = C + 1$;
end while

Source: Jospin et al. "Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users." IEEE Comp. Intell. Mag. (2022)

Source: Settles et al. "Active Learning Literature Survey." Univ. of Wisc. Madison TR. (2010)

DL vs. Bayesian DL

Standard Deep Learning

- Works great much of the time if we only care about predictive accuracy
- Point estimate-based learning can be brittle, yield poor uncertainty calibration

Bayesian Deep Learning

- Combines DL models with Bayesian concepts and inference
- Directly represents uncertainty over network and predictions
- More robust predictive models than point estimates
- Significantly increases computational burden
- Some simple “approximately Bayesian” methods perform decently

Bayesian Model Averaging (BMA)

We wish to compute the *posterior predictive distribution:*

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

Typically approximated via Monte Carlo integration:

$$p(y \mid x, \mathcal{D}) \approx \frac{1}{M} \sum_{m=1}^{M} p(y \mid x, w_m), \quad w_m \sim p(w \mid \mathcal{D})$$

Typically, two classes of methods: (1) MCMC (2) deterministic methods such as variational inference, Laplace approximation, etc.

Bayesian DL Inference

	Benefits	Limitations	Use cases	
MCMC (V.A)	Directly samples the posterior	Requires to store a very large number of samples	Small and average models	
Classic methods (HMC, NUTS)(§V-A)	State of the art samplers limit autocorrelation between samples	Do not scale well to large models	Small and critical models	**Can be combined**
SGLD and derivates (§V-E2a)	Provide a well behaved Markov Chain with minibatches	Focus on a single mode of the posterior	Models with larger datasets	
Warm restarts (§V-E2a)	Help a MCMC method explore different modes of the posterior	Requires a new burn-in sequence for each restart	Combined with a MCMC sampler	
Variational inference (V.B)	The variational distribution is easy to sample	Is an approximation	Large scale models	
Bayes by backprop (§V-C)	Fit any parametric distribution as posterior	Noisy gradient descent	Large scale models	**Can be combined**
Monte Carlo-Dropout (§V-E1)	Can transform a model using dropout into a BNN	Lack expressive power	Dropout based models	
Laplace approximation (§V-E2b)	By analyzing standard SGD get a BNN from a MAP	Focus on a single mode of the posterior	Unimodals large scale models	
Deep ensembles (§V-E2b)	Help focusing on different modes of the posterior	Cannot detect local uncertainty if used alone	Multimodals models and combined with other VI methods	

Source: Jospin et al. "Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users." IEEE Comp. Intell. Mag. (2022)

Deep Ensembles

- Train an ensemble of DNNs from random initializations to produce an *ensemble* of networks with weights:

$$\{w_m\}_{m=1}^{M} \sim \operatorname{argmin}_w \mathcal{L}(w)$$

- Do Bayesian model averaging (BMA) using Monte Carlo integration:

$$p(y \mid x, \mathcal{D}) \approx \frac{1}{M} \sum_m p(y \mid x, w_m)$$

- Finds many low-loss solutions in different basins of attraction
- Easy to implement, and ensemble can be trained in parallel

Lakshminarayanan et al. "Simple and scalable predictive uncertainty estimation using deep ensembles." *Advances in neural information processing systems* 30 (2017)

Posterior Predictive Distribution

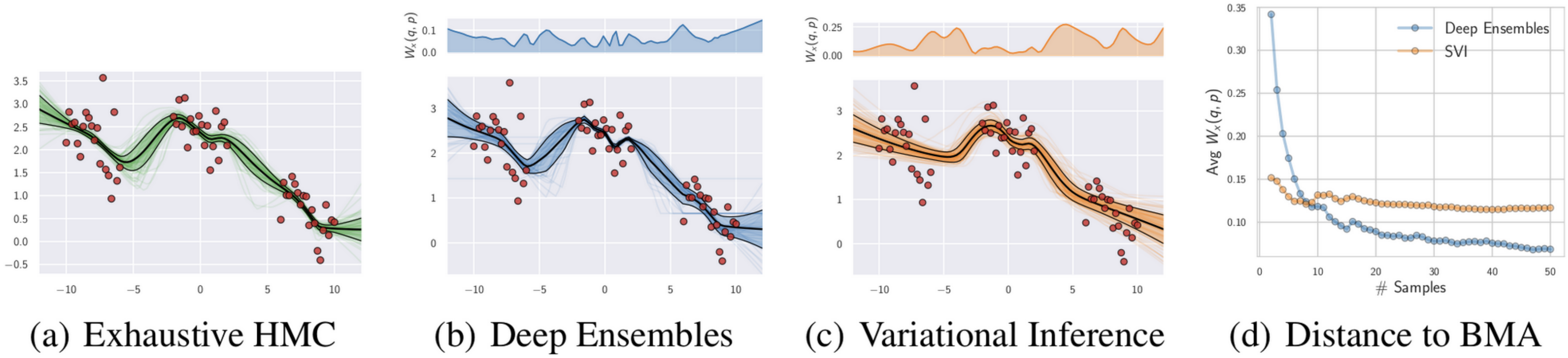


Figure 2. **(a)**: A close approximation of the true predictive distribution obtained by combining 10 HMC chains, each producing 500 samples. **(b)**: Deep ensembles predictive distribution using 50 independently trained networks. **(c)**: Predictive distribution for factorized variational inference (VI). **(d)**: Convergence of the predictive distributions for deep ensembles and variational inference as a function of the number of samples; we measure the average Wasserstein distance between the marginals in the range of input positions. The multi-basin deep ensembles approach provides a more faithful approximation of the Bayesian predictive distribution than the conventional single-basin VI approach, which is overconfident between data clusters. The top panels show the Wasserstein distance between the true predictive distribution and the deep ensemble and VI approximations, as a function of inputs x. For experimental details, see [1].

A.G. Wilson, P. Izmailov. *Bayesian Deep Learning and a Probabilistic Perspective of Generalization*. Advances in Neural Information Processing Systems, 2020

Proper Scoring Rules

- Scoring rules measure the quality of predictive uncertainty
- Assigns a numerical score to the predictive distribution,
- Expected scoring rule:

$$S(p_w, q) = \int q(x, y) S(p_w, (x, y)) \, dx \, dy$$

- A scoring rule is *proper* if: $S(p_w, q) \leq S(q, q)$, with equality only if p=q
- For example, the *Brier* score is given by:

$$-S(p_w, q) = \frac{1}{M} \sum_{m=1}^{M} \left(\delta_{k=y} - p(y = k \mid x, w)\right)^2$$

Adversarial Training

- Used as technique to smooth predictive distributions
- Inputs 'close' to original training examples that are classified incorrectly
- Goodfellow et al. (2015) proposed *fast gradient sign method:*

$$\ell(w, x, y) = -\log p(y \mid x, w), \qquad x' = x + \epsilon \cdot \mathrm{sign}(\nabla_x \ell(w, x, y))$$

- Creates a new training example x' along direction that network is likely to increase the loss
- Augment training set with (x', y) as additional training example
- Other methods exist that do not require access to training label y

Deep Ensembles

Algorithm 1 Pseudocode of the training procedure for our method

$\triangleright$ *Let each neural network parametrize a distribution over the outputs, i.e.* $p_\theta(y|\mathbf{x})$. *Use a proper scoring rule as the training criterion* $\ell(\theta, \mathbf{x}, y)$. *Recommended default values are* $M = 5$ *and* $\boldsymbol{\epsilon} = 1\%$ *of the input range of the corresponding dimension (e.g 2.55 if input range is [0,255]).*
Initialize $\theta_1, \theta_2, \ldots, \theta_M$ randomly
for $m = 1 : M$ **do** $\triangleright$ *train networks independently in parallel*
  Sample data point n_m randomly for each net $\triangleright$ *single* n_m *for clarity, minibatch in practice*
  Generate adversarial example using $\mathbf{x}'_{n_m} = \mathbf{x}_{n_m} + \boldsymbol{\epsilon}\, \mathrm{sign}\big(\nabla_{\mathbf{x}_{n_m}} \ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m})\big)$
  Minimize $\ell(\theta_m, \mathbf{x}_{n_m}, y_{n_m}) + \ell(\theta_m, \mathbf{x}'_{n_m}, y_{n_m})$ w.r.t. θ_m $\triangleright$ *adversarial training (optional)*

Regression on Toy Datasets

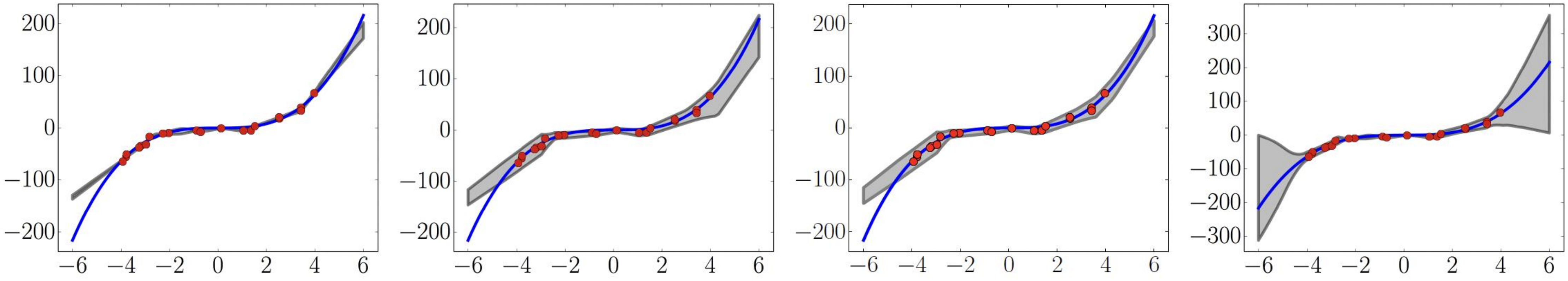


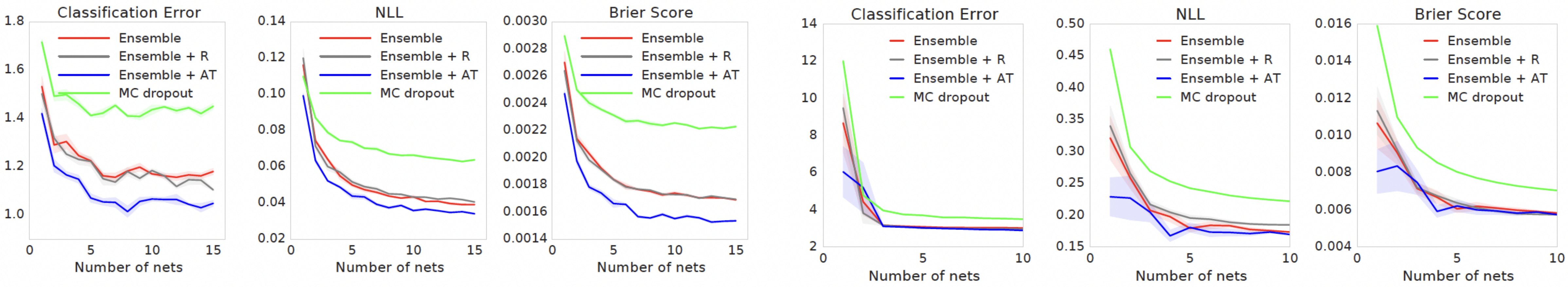
Figure 1: Results on a toy regression task: x-axis denotes x. On the y-axis, the blue line is the *ground truth* curve, the red dots are observed noisy training data points and the gray lines correspond to the predicted mean along with three standard deviations. Left most plot corresponds to empirical variance of 5 networks trained using MSE, second plot shows the effect of training using NLL using a single net, third plot shows the additional effect of adversarial training, and final plot shows the effect of using an ensemble of 5 networks respectively.

Regression on Real World Datasets

Datasets	RMSE PBP	RMSE MC-dropout	RMSE Deep Ensembles	NLL PBP	NLL MC-dropout	NLL Deep Ensembles
Boston housing	**3.01 ± 0.18**	**2.97 ± 0.85**	**3.28 ± 1.00**	**2.57 ± 0.09**	**2.46 ± 0.25**	**2.41 ± 0.25**
Concrete	**5.67 ± 0.09**	**5.23 ± 0.53**	**6.03 ± 0.58**	**3.16 ± 0.02**	**3.04 ± 0.09**	**3.06 ± 0.18**
Energy	**1.80 ± 0.05**	**1.66 ± 0.19**	**2.09 ± 0.29**	2.04 ± 0.02	1.99 ± 0.09	**1.38 ± 0.22**
Kin8nm	0.10 ± 0.00	0.10 ± 0.00	**0.09 ± 0.00**	-0.90 ± 0.01	-0.95 ± 0.03	**-1.20 ± 0.02**
Naval propulsion plant	0.01 ± 0.00	0.01 ± 0.00	**0.00 ± 0.00**	-3.73 ± 0.01	-3.80 ± 0.05	**-5.63 ± 0.05**
Power plant	**4.12 ± 0.03**	**4.02 ± 0.18**	**4.11 ± 0.17**	2.84 ± 0.01	**2.80 ± 0.05**	**2.79 ± 0.04**
Protein	4.73 ± 0.01	**4.36 ± 0.04**	4.71 ± 0.06	2.97 ± 0.00	2.89 ± 0.01	**2.83 ± 0.02**
Wine	**0.64 ± 0.01**	**0.62 ± 0.04**	**0.64 ± 0.04**	0.97 ± 0.01	**0.93 ± 0.06**	**0.94 ± 0.12**
Yacht	**1.02 ± 0.05**	**1.11 ± 0.38**	**1.58 ± 0.48**	1.63 ± 0.02	1.55 ± 0.12	**1.18 ± 0.21**
Year Prediction MSD	8.88 ± NA	**8.85 ± NA**	8.89 ± NA	3.60 ± NA	3.59 ± NA	**3.35 ± NA**

Table 1: Results on regression benchmark datasets comparing RMSE and NLL. See Table 2 for results on variants of our method.

Classification (MNIST & SVHN)

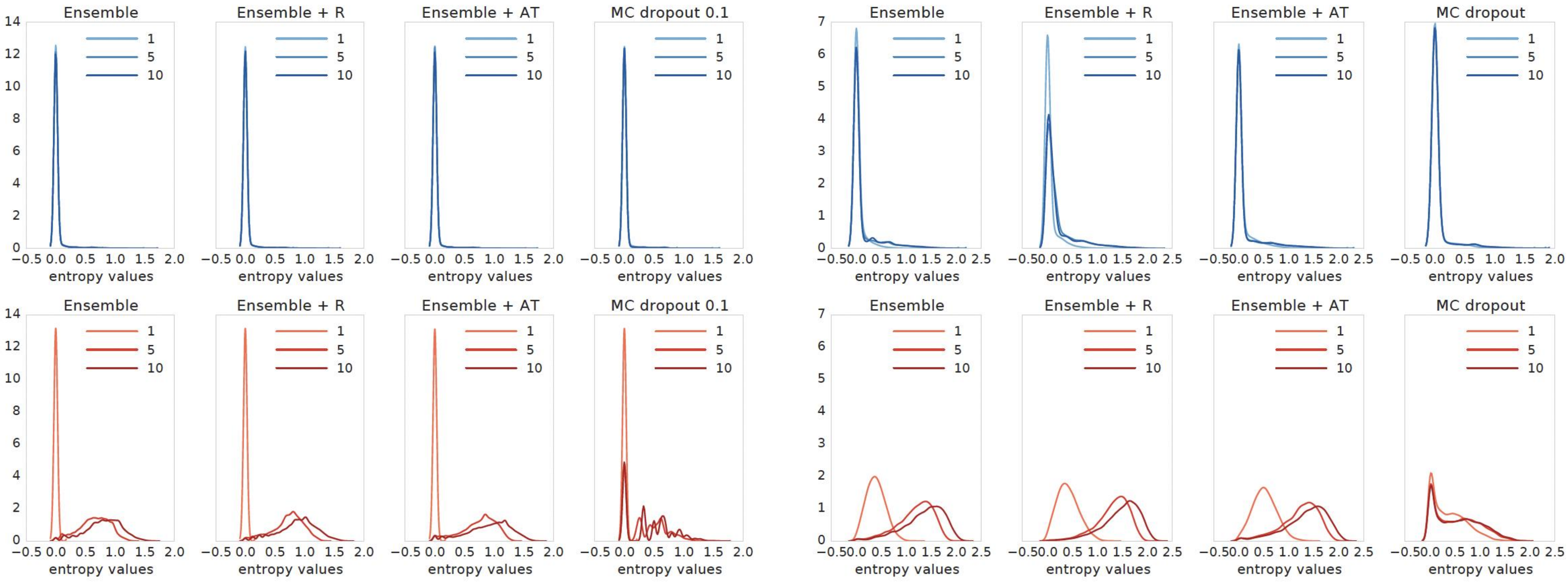


(a) MNIST dataset using 3-layer MLP

(b) SVHN using VGG-style convnet

Figure 2: Evaluating predictive uncertainty as a function of ensemble size M (number of networks in the ensemble or the number of MC-dropout samples): Ensemble variants significantly outperform MC-dropout performance with the corresponding M in terms of all 3 metrics. Adversarial training improves results for MNIST for all M and SVHN when $M = 1$, but the effect drops as M increases.

Classification (MNIST & SVHN)



(a) MNIST-NotMNIST

(b) SVHN-CIFAR10

Figure 3: : Histogram of the predictive entropy on test examples from known classes (top row) and unknown classes (bottom row), as we vary ensemble size M.

MNIST vs. NotMNIST

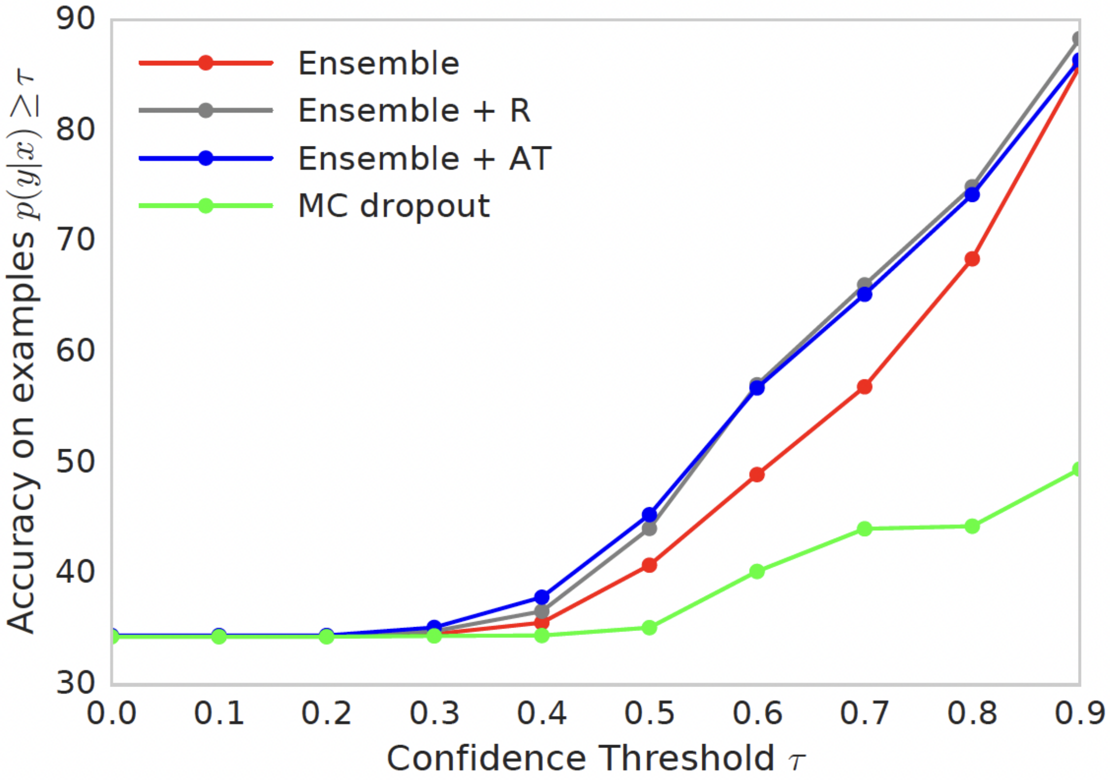


Figure 6: Accuracy vs Confidence curves: Networks trained on MNIST and tested on both MNIST test containing known classes and the NotMNIST dataset containing unseen classes. MC-dropout can produce overconfident wrong predictions, whereas deep ensembles are significantly more robust.

Classification on ImageNet

M	Top-1 error %	Top-5 error %	NLL	Brier Score $\times 10^{-3}$
1	22.166	6.129	0.959	0.317
2	20.462	5.274	0.867	0.294
3	19.709	4.955	0.836	0.286
4	19.334	4.723	0.818	0.282
5	19.104	4.637	0.809	0.280
6	18.986	4.532	0.803	0.278
7	18.860	4.485	0.797	0.277
8	18.771	4.430	0.794	0.276
9	18.728	4.373	0.791	0.276
10	18.675	4.364	0.789	0.275

Figure 4: Results on ImageNet: Deep Ensembles lead to lower classification error as well as better predictive uncertainty as evidenced by lower NLL and Brier score.

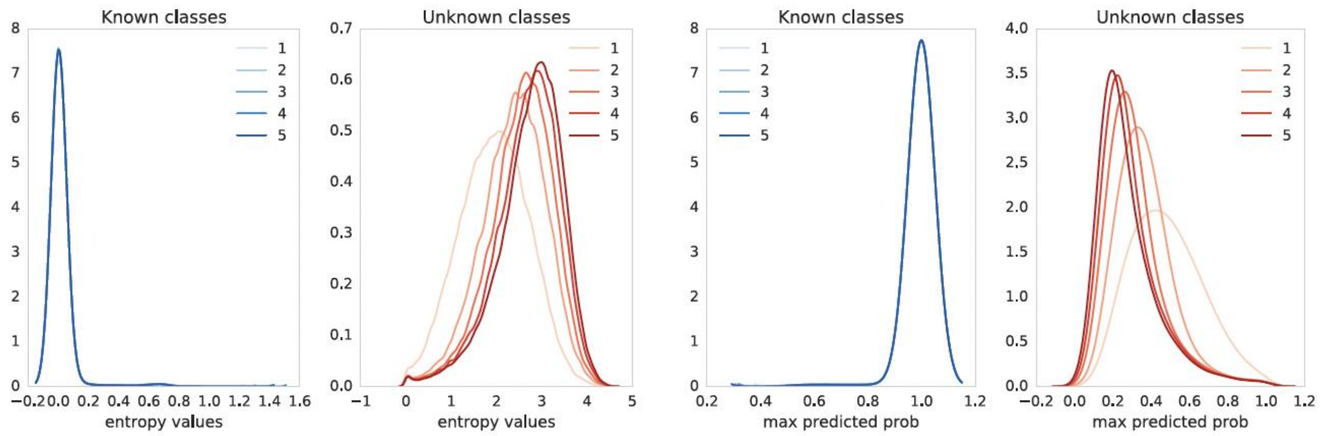


Figure 5: ImageNet trained only on dogs: Histogram of the predictive entropy (left) and maximum predicted probability (right) on test examples from known classes (dogs) and unknown classes (non-dogs), as we vary the ensemble size.

Bayesian DL Inference

	Benefits	Limitations	Use cases
MCMC (V.A)	Directly samples the posterior	Requires to store a very large number of samples	Small and average models
Classic methods (HMC, NUTS)(§V-A)	State of the art samplers limit autocorrelation between samples	Do not scale well to large models	Small and critical models
SGLD and derivates (§V-E2a)	Provide a well behaved Markov Chain with minibatches	Focus on a single mode of the posterior	Models with larger datasets
Warm restarts (§V-E2a)	Help a MCMC method explore different modes of the posterior	Requires a new burn-in sequence for each restart	Combined with a MCMC sampler
Variational inference (V.B)	The variational distribution is easy to sample	Is an approximation	Large scale models
Bayes by backprop (§V-C)	Fit any parametric distribution as posterior	Noisy gradient descent	Large scale models
Monte Carlo-Dropout (§V-E1)	Can transform a model using dropout into a BNN	Lack expressive power	Dropout based models
Laplace approximation (§V-E2b)	By analyzing standard SGD get a BNN from a MAP	Focus on a single mode of the posterior	Unimodals large scale models
Deep ensembles (§V-E2b)	Help focusing on different modes of the posterior	Cannot detect local uncertainty if used alone	Multimodals models and combined with other VI methods

Can be combined

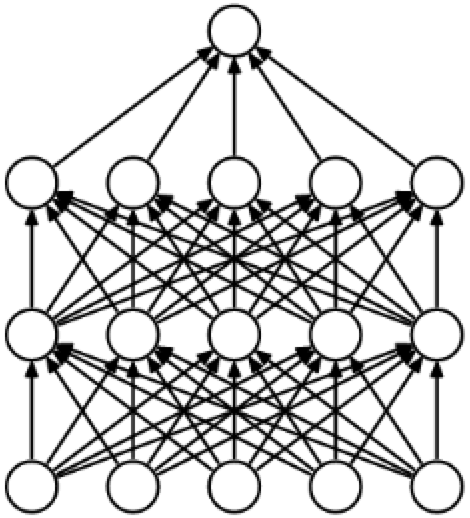
Can be combined

Source: Jospin et al. "Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users." IEEE Comp. Intell. Mag. (2022)

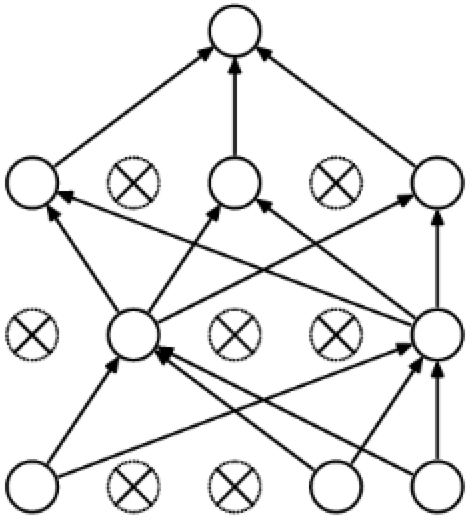
Monte Carlo Dropout

Dropout

- Typically used as regularizer in training
- Each grad update randomly remove nodes
- Ensures network not overly sensitive to small subset of edges



(a) Standard Neural Net



(b) After applying dropout.

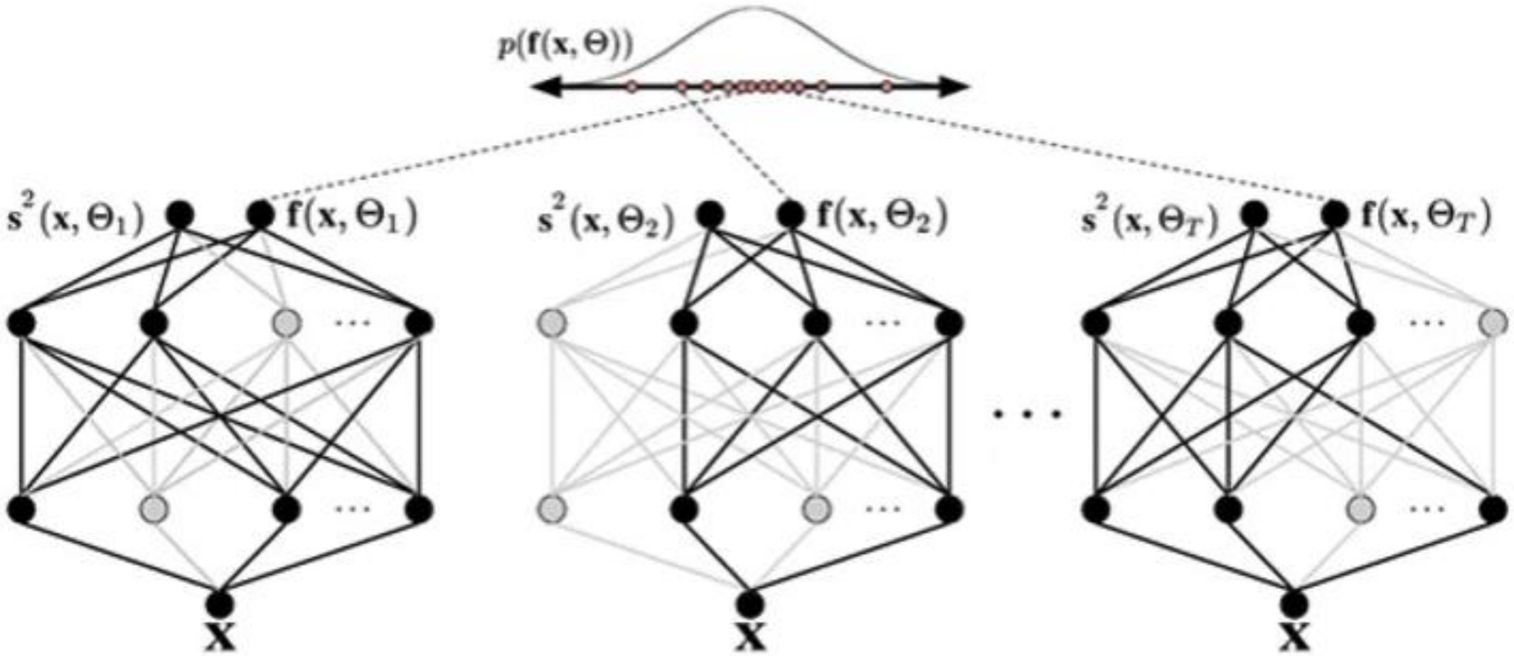
Monte Carlo Dropout

- Do dropout at prediction…generate ensemble of predictions by dropping a subset of edges for each
- Equivalent to VI with variational distribution for each weight as,

$$z_{i,j} \sim \text{Bernoulli}(p_i),$$
$$\boldsymbol{W}_i = \boldsymbol{M}_i \cdot \text{diag}(\boldsymbol{z}_i),$$

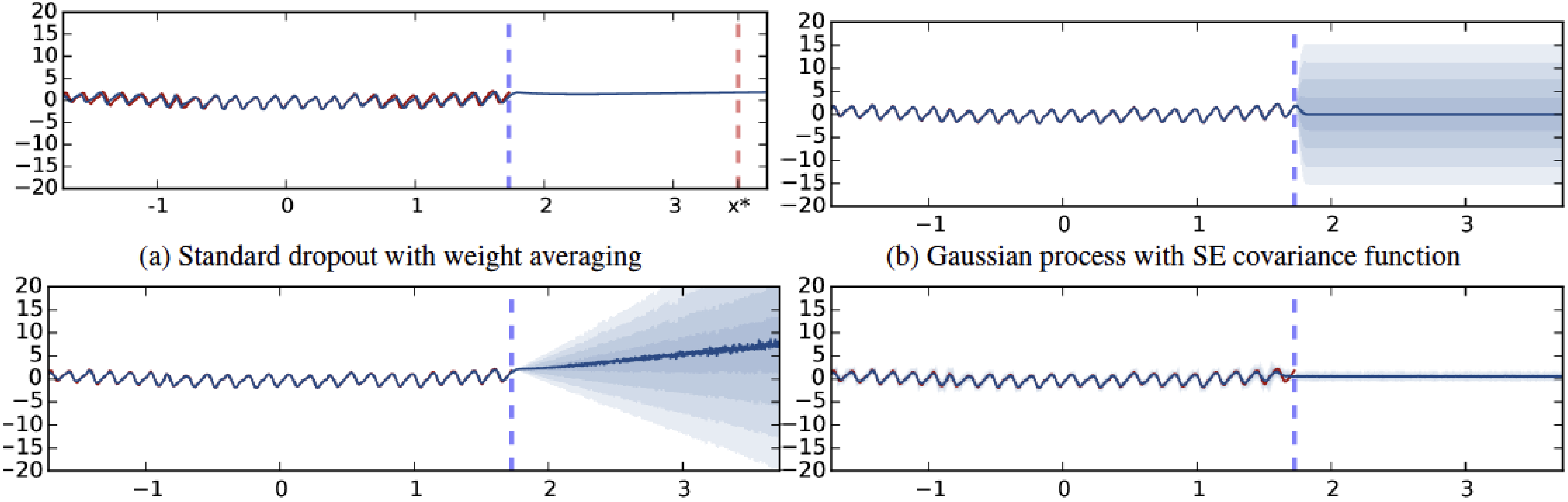
Monte Carlo Dropout

Distribution of outputs quantifies uncertainty



Source: https://medium.com/@ciaranbench/monte-carlo-dropout-a-practical-guide-4b4dc18014b5

Mauna Loa CO2 Concentrations



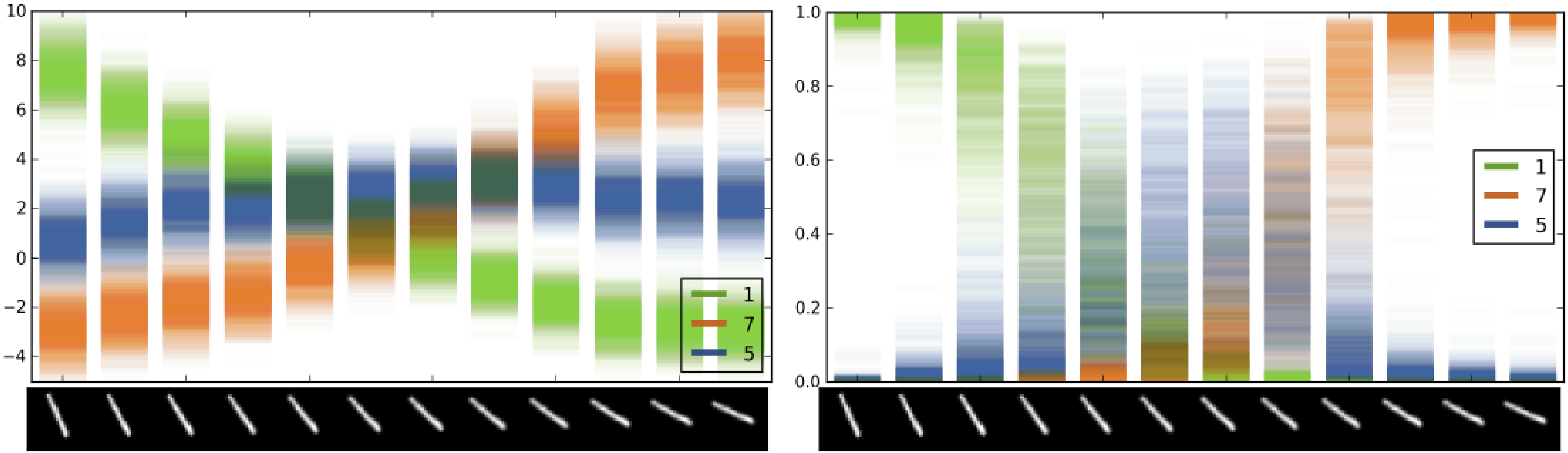
(a) Standard dropout with weight averaging

(b) Gaussian process with SE covariance function

(c) MC dropout with ReLU non-linearities

(d) MC dropout with TanH non-linearities

Gal and Ghahramani. ICML. 2016



(a) Softmax *input* scatter (b) Softmax *output* scatter

Figure 4. **A scatter of 100 forward passes of the softmax input and output for dropout LeNet.** On the X axis is a rotated image of the digit 1. The input is classified as digit 5 for images 6-7, even though model uncertainty is extremly large (best viewed in colour).

Bayesian DL Inference

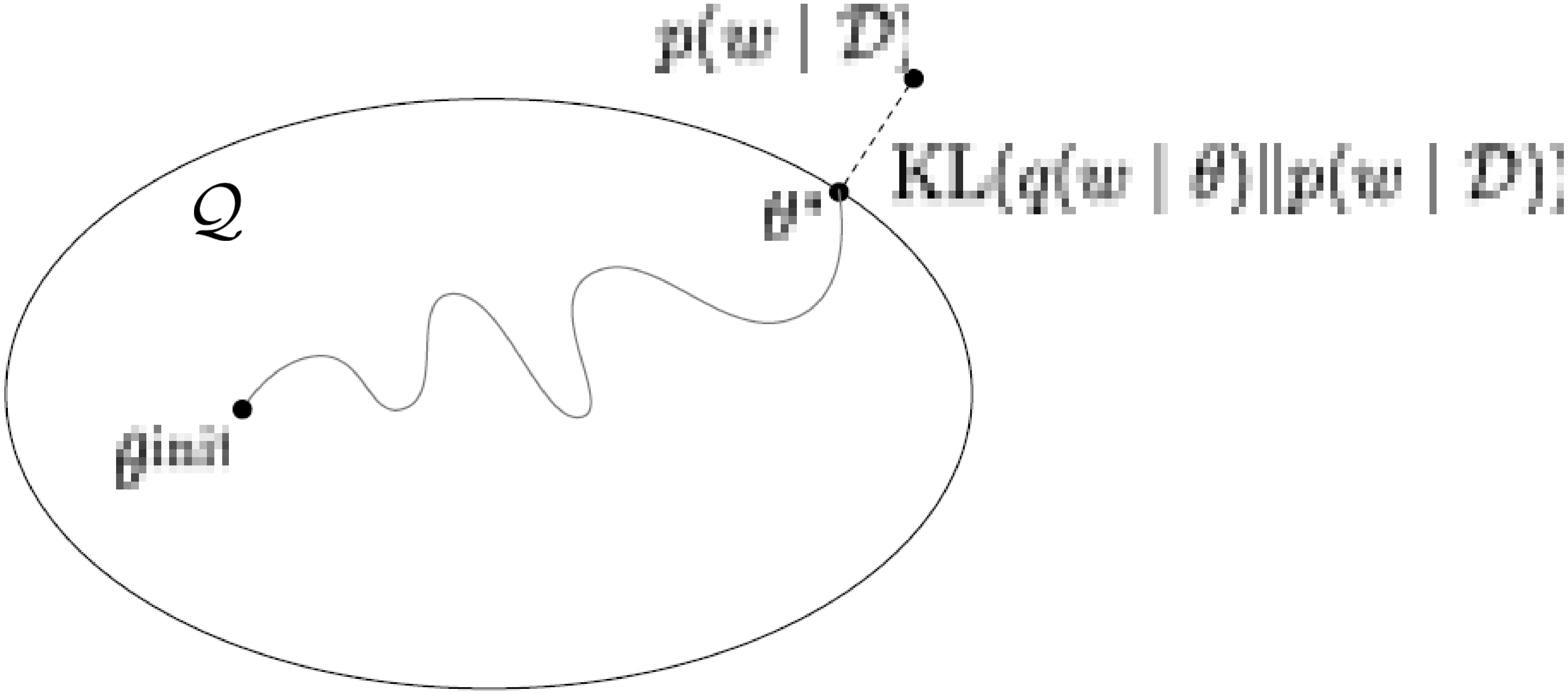
	Benefits	Limitations	Use cases
MCMC (V.A)	Directly samples the posterior	Requires to store a very large number of samples	Small and average models
Classic methods (HMC, NUTS)(§V-A)	State of the art samplers limit autocorrelation between samples	Do not scale well to large models	Small and critical models
SGLD and derivates (§V-E2a)	Provide a well behaved Markov Chain with minibatches	Focus on a single mode of the posterior	Models with larger datasets
Warm restarts (§V-E2a)	Help a MCMC method explore different modes of the posterior	Requires a new burn-in sequence for each restart	Combined with a MCMC sampler
Variational inference (V.B)	The variational distribution is easy to sample	Is an approximation	Large scale models
Bayes by backprop (§V-C)	Fit any parametric distribution as posterior	Noisy gradient descent	Large scale models
Monte Carlo-Dropout (§V-E1)	Can transform a model using dropout into a BNN	Lack expressive power	Dropout based models
Laplace approximation (§V-E2b)	By analyzing standard SGD get a BNN from a MAP	Focus on a single mode of the posterior	Unimodals large scale models
Deep ensembles (§V-E2b)	Help focusing on different modes of the posterior	Cannot detect local uncertainty if used alone	Multimodals models and combined with other VI methods

Can be combined

Can be combined

Source: Jospin et al. "Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users." IEEE Comp. Intell. Mag. (2022)

Variational Approximation



Minimize KL between $q(w \mid \theta)$ and posterior $p(w \mid \mathcal{D})$.

[Source: David Blei]

Variational Inference

Recall the Kullback-Leibler divergence given as,

$$\mathrm{KL}[q(x)\|P(x)] \equiv \int q(x) \log \frac{q(x)}{P(x)} dx$$

Our variational parameters are given by,

$$\begin{aligned}\theta^* &= \arg\min_\theta \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w}|\mathcal{D})} \mathrm{d}\mathbf{w} \\ &= \arg\min_\theta \int q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})} \mathrm{d}\mathbf{w}\end{aligned}$$

Variational Loss

So our loss function is given by,

$$\mathcal{F}(\mathcal{D},\theta) = \int q(\mathbf{w}|\theta)\log\frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})} - q(\mathbf{w}|\theta)\log P(\mathcal{D}|\mathbf{w})\mathrm{d}\mathbf{w}$$
$$= \mathrm{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]$$

Just differentiate the loss function and optimize, right?

$$\nabla_\theta\mathcal{F}(\mathcal{D},\theta) = \nabla_\theta\mathrm{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \nabla_\theta\mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]$$

Gradient-Based Optimization

No. We don't get a straightforward Monte Carlo estimator…

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(w|\theta)}[f(w,\theta)] = \int \frac{\partial}{\partial\theta} q(w \mid \theta) f(w,\theta)\, dw$$
$$= \int q'(w \mid \theta) f(w,\theta)\, dw + \int q(w \mid \theta) f'(w,\theta)\, dw$$

…first term is not an expected value!

Reparameterization Trick

Proposition 1. *Let ϵ be a random variable having a probability density given by $q(\epsilon)$ and let $\mathbf{w} = t(\theta, \epsilon)$ where $t(\theta, \epsilon)$ is a deterministic function. Suppose further that the marginal probability density of $\mathbf{w}$, $q(\mathbf{w}|\theta)$, is such that $q(\epsilon)d\epsilon = q(\mathbf{w}|\theta)d\mathbf{w}$. Then for a function f with derivatives in $\mathbf{w}$:*

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

Proof of Reparameterization Trick

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

$\square$

Variational Loss

So our loss function is given by,

$$\mathcal{F}(\mathcal{D}, \theta) = \mathrm{KL}[q(\mathbf{w}|\theta) \| P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})]$$

Given samples $\{w^{(i)}\}_{i=1}^{n} \sim q(w \mid \theta)$ approximate loss as,

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)}|\theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathcal{D}|\mathbf{w}^{(i)})$$

Use reparameterization trick to calculate gradient.

Gaussian Reparameterization

So we need a deterministic function s.t. $\mathbf{w} = t(\theta, \epsilon)$.

Suppose we want to sample a Gaussian RV,

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

But we only know how to sample a *standard* Gaussian RV,

$$\epsilon \sim \mathcal{N}(0, 1)$$

Gaussians are closed under linear transformations so,

$$\mathbf{w} = \underbrace{\mu + \sigma\epsilon}_{\mathbf{w} = t(\theta, \epsilon)} \sim \mathcal{N}(\mu, \sigma^2)$$

1. Sample $\epsilon \sim \mathcal{N}(0, I)$.
2. Let $\mathbf{w} = \mu + \log(1 + \exp(\rho)) \circ \epsilon$.
3. Let $\theta = (\mu, \rho)$.
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathbf{w})P(\mathcal{D}|\mathbf{w})$.

Done by backpropagtation

5. Calculate the gradient with respect to the mean

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu}. \tag{3}$$

6. Calculate the gradient with respect to the standard deviation parameter ρ

$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho}. \tag{4}$$

7. Update the variational parameters:

$$\mu \leftarrow \mu - \alpha \Delta_\mu \tag{5}$$

$$\rho \leftarrow \rho - \alpha \Delta_\rho. \tag{6}$$

Noisy Regression

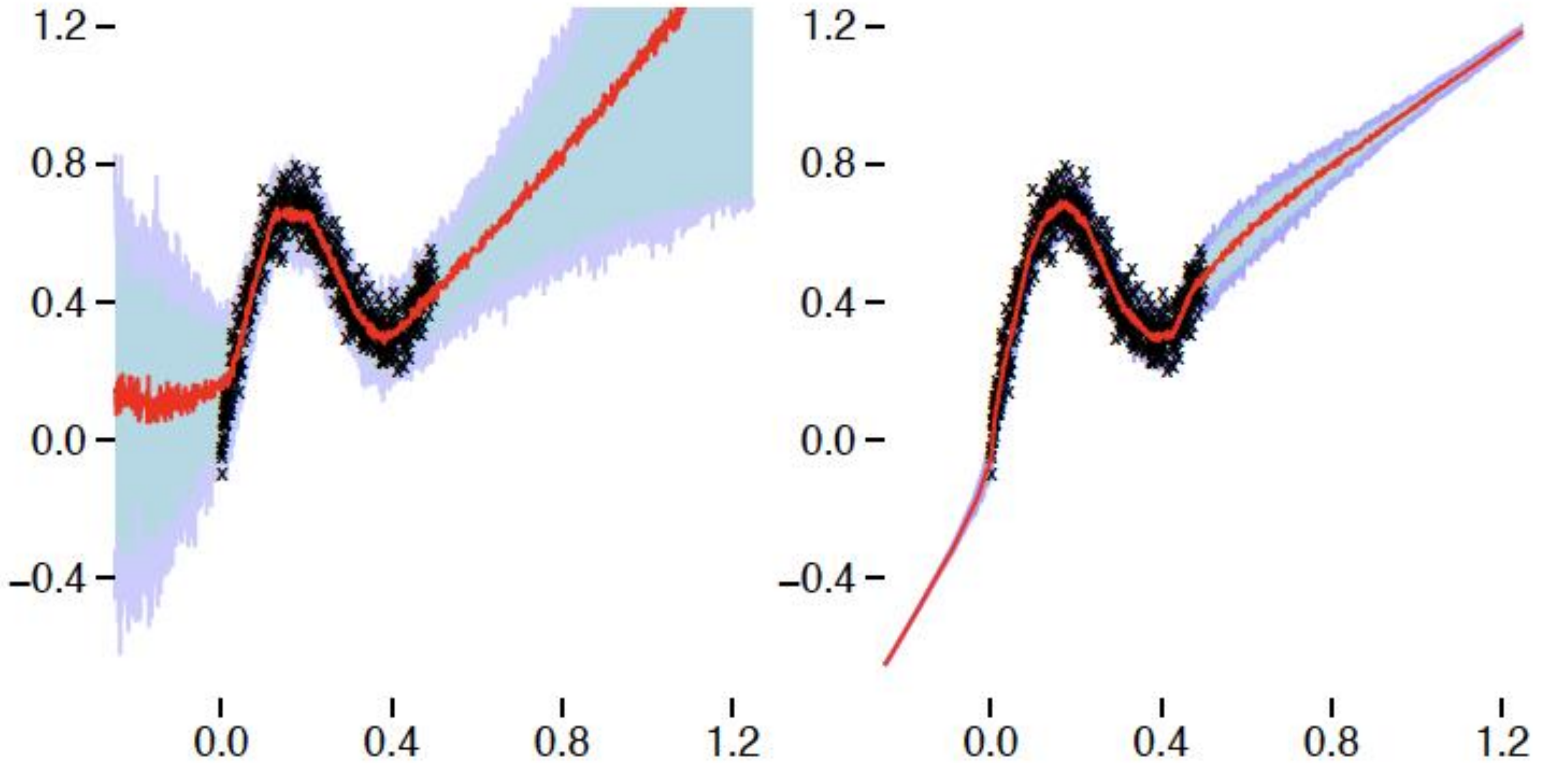
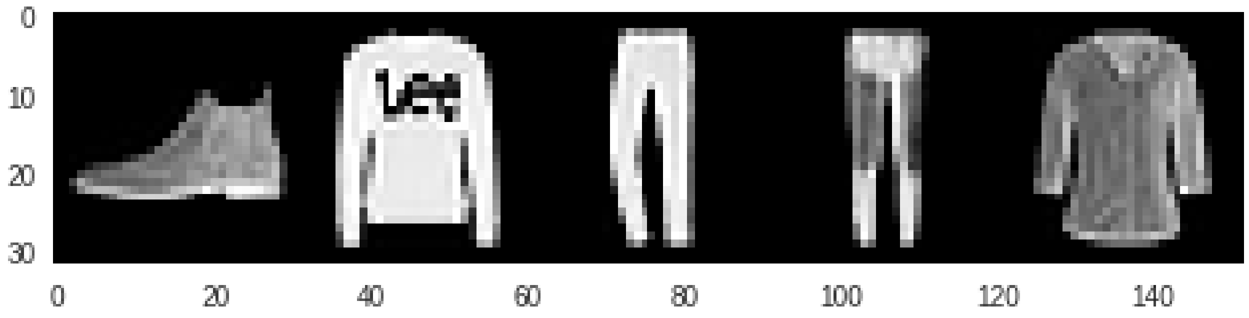


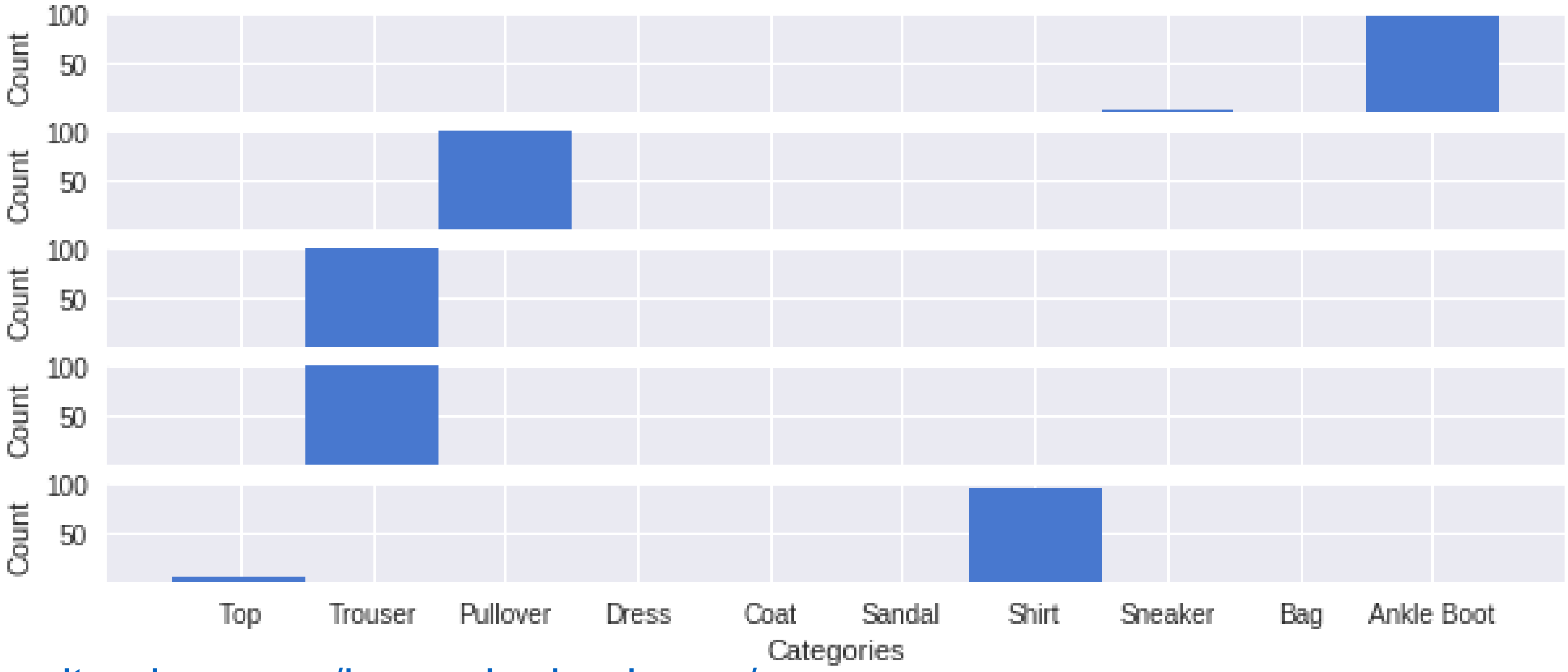
Figure 5. Regression of noisy data with interquatile ranges. Black crosses are training samples. Red lines are median predictions. Blue/purple region is interquartile range. Left: Bayes by Backprop neural network, Right: standard neural network.

Blundell et al., ICML. 2015

Fashion MNIST

70k images, 28x28, 10 classes of clothing objects

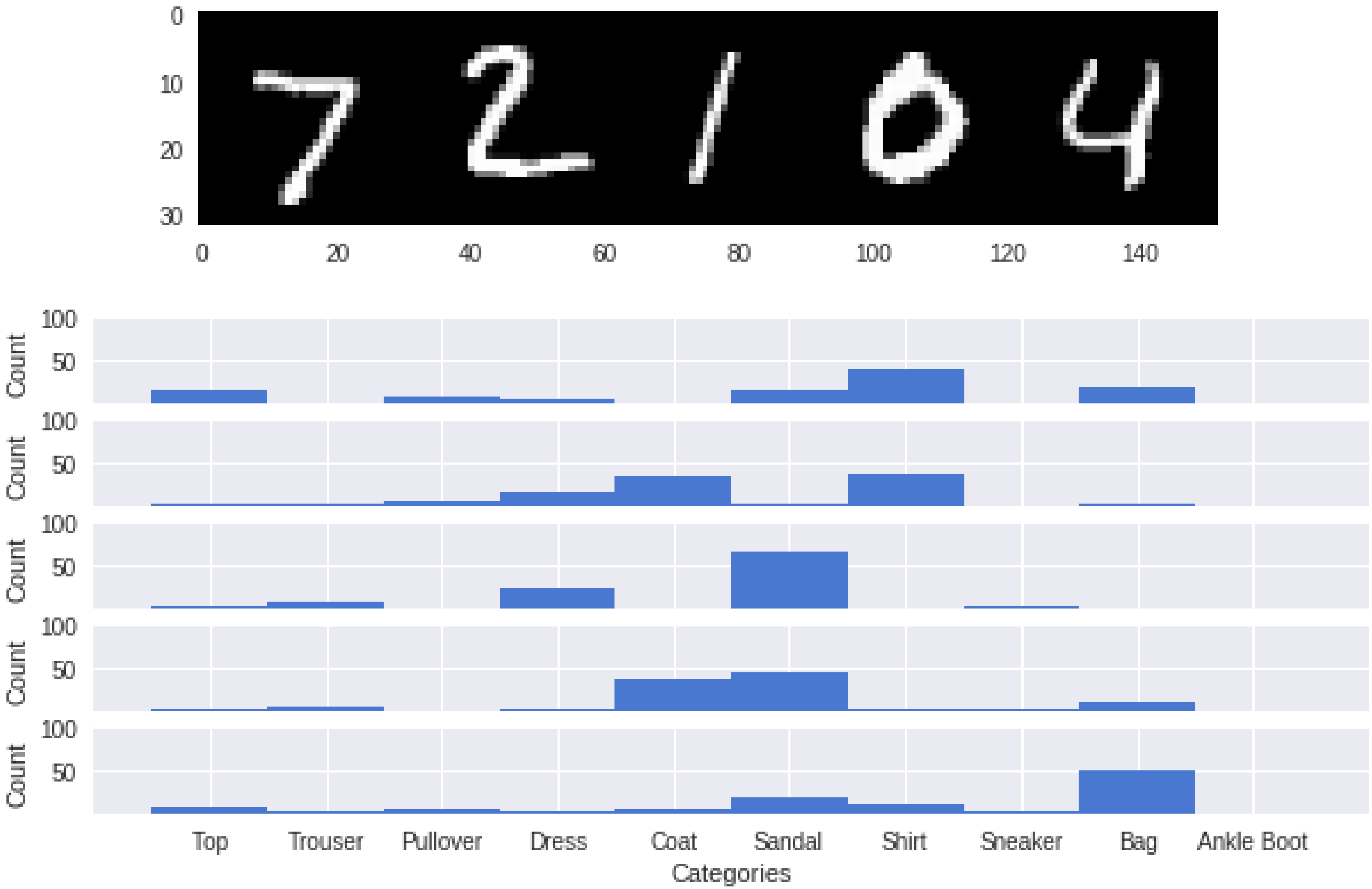




Source: https://www.nitarshan.com/bayes-by-backprop/

MNIST Out-of-Sample Prediction

Trained on FMNIST, tested on MNIST



Source: https://www.nitarshan.com/bayes-by-backprop/

In-Sample MNIST Prediction

Table 1. Classification Error Rates on MNIST. ⋆ 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)			≈ 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**%

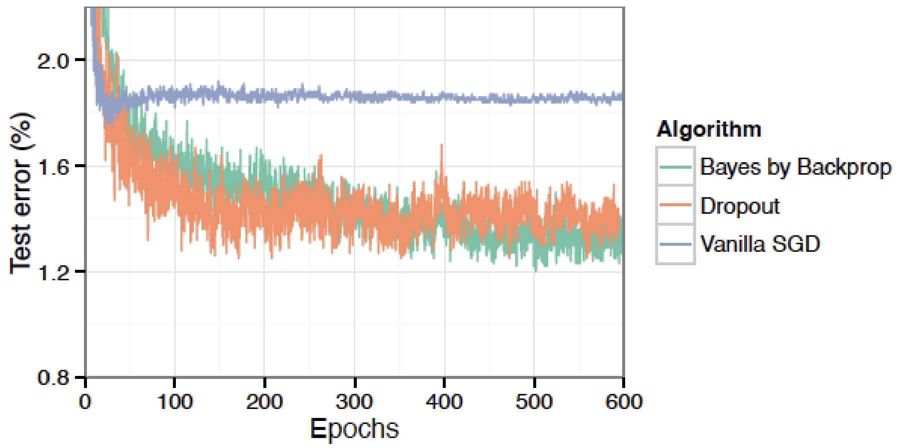


Figure 2. Test error on MNIST as training progresses.

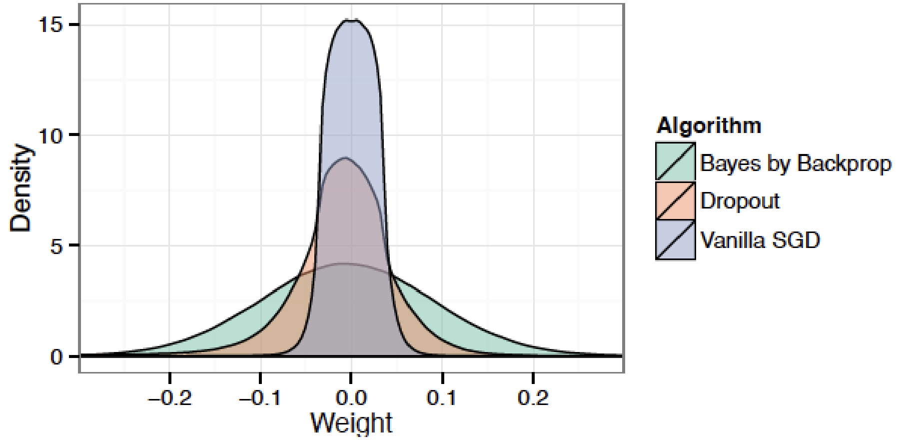


Figure 3. Histogram of the trained weights of the neural network, for Dropout, plain SGD, and samples from Bayes by Backprop.

Blundell et al., ICML. 2015

Weight Pruning (MNIST)

Remove weights by their signal-to-noise ratio…

Proportion removed	# Weights	Test Error
0%	2.4m	1.24%
50%	1.2m	1.24%
75%	600k	1.24%
95%	120k	1.29%
98%	48k	1.39%

…95% weights removed with minimal affect on accuracy.

Blundell et al., ICML. 2015