



Computer
Science

CSC696H: Advanced Topics in Probabilistic Graphical Models

Bayesian Deep Learning

Prof. Jason Pacheco

Outline

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

Outline

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

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



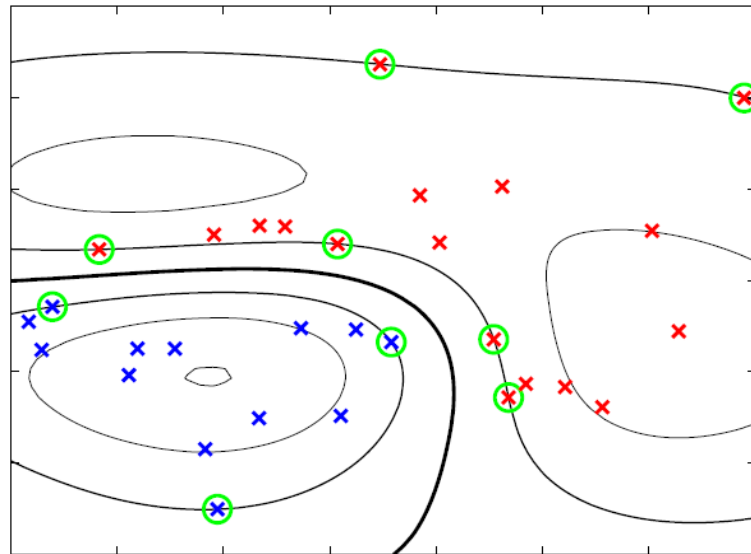
$$y = \sigma(w^T \phi(x))$$

...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

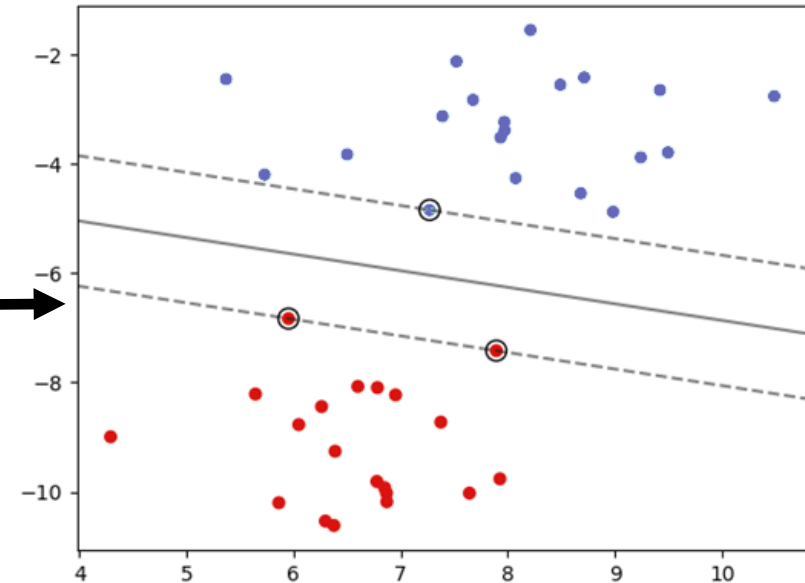


Ignore the circled points...I reused these from the SVM slides

Neural Net
 $\phi(x)$



Warped Space



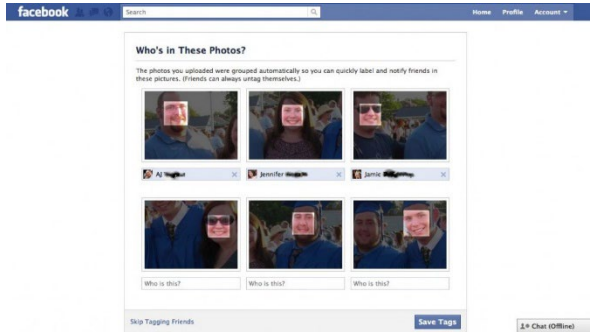
...this is essentially what standard neural networks do...

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...



FB Auto Tagging

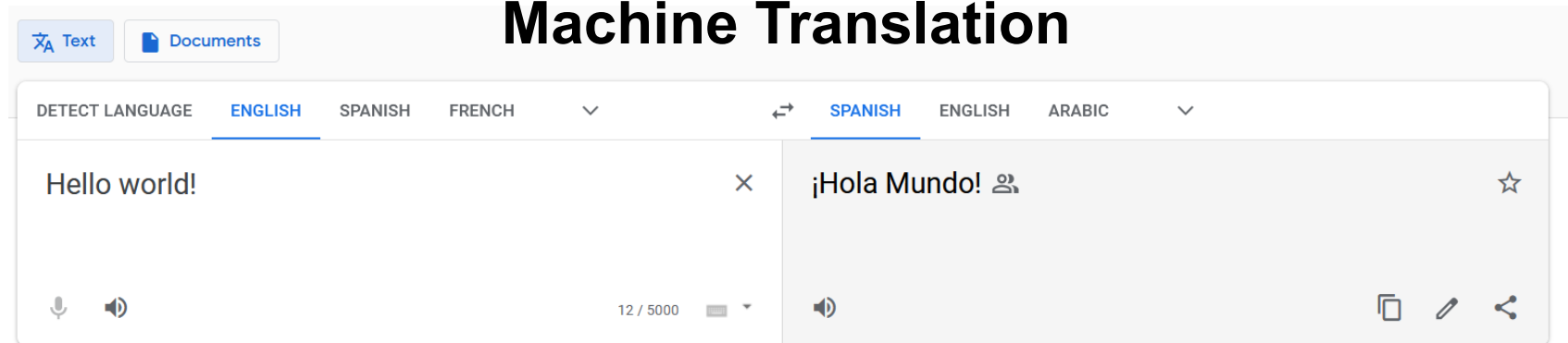


Self-Driving Cars



Creepy Robots

Machine Translation

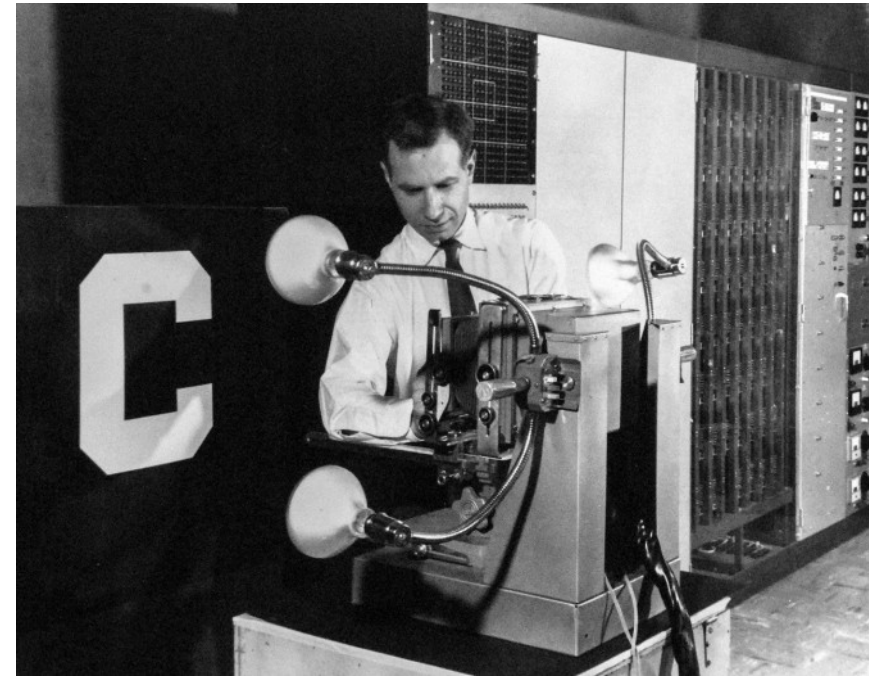


[Send feedback](#)

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

FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

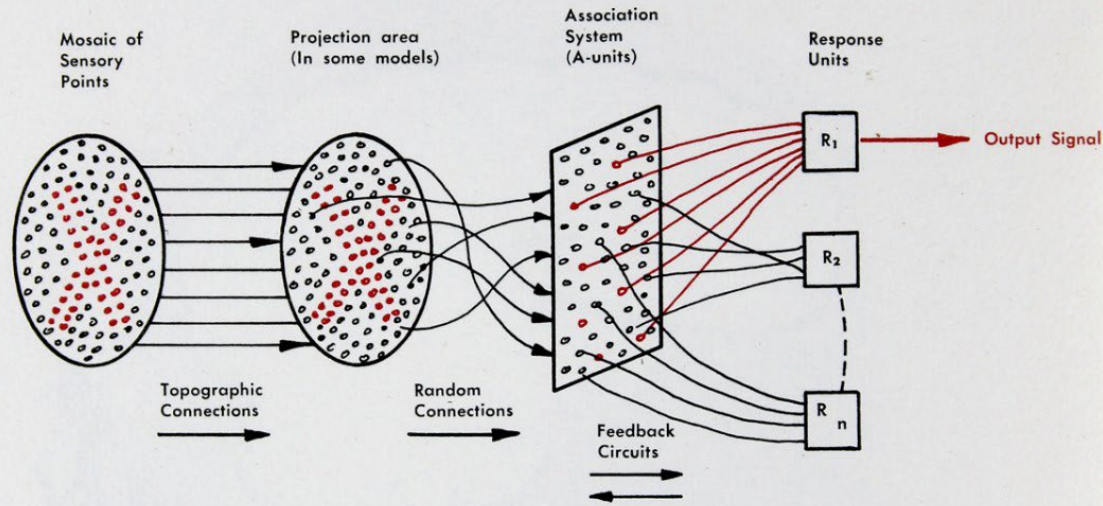
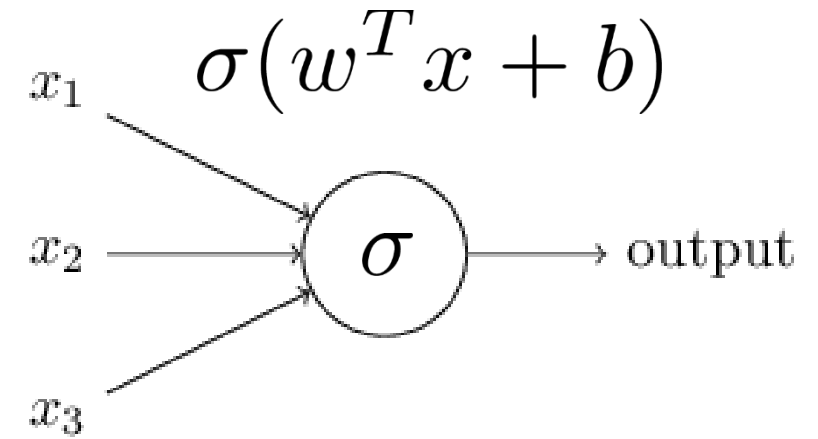


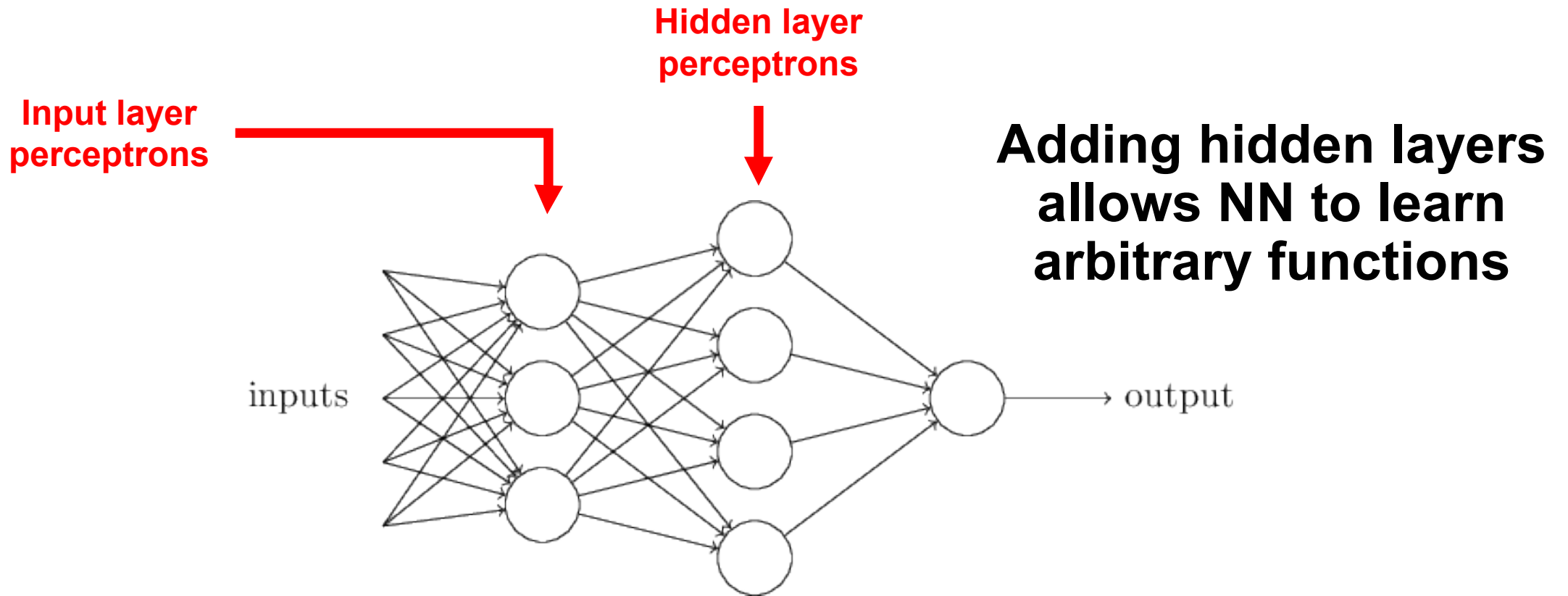
FIG. 2 — Organization of a perceptron.

Perceptron



- 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
- **The perceptron is just logistic regression in disguise**

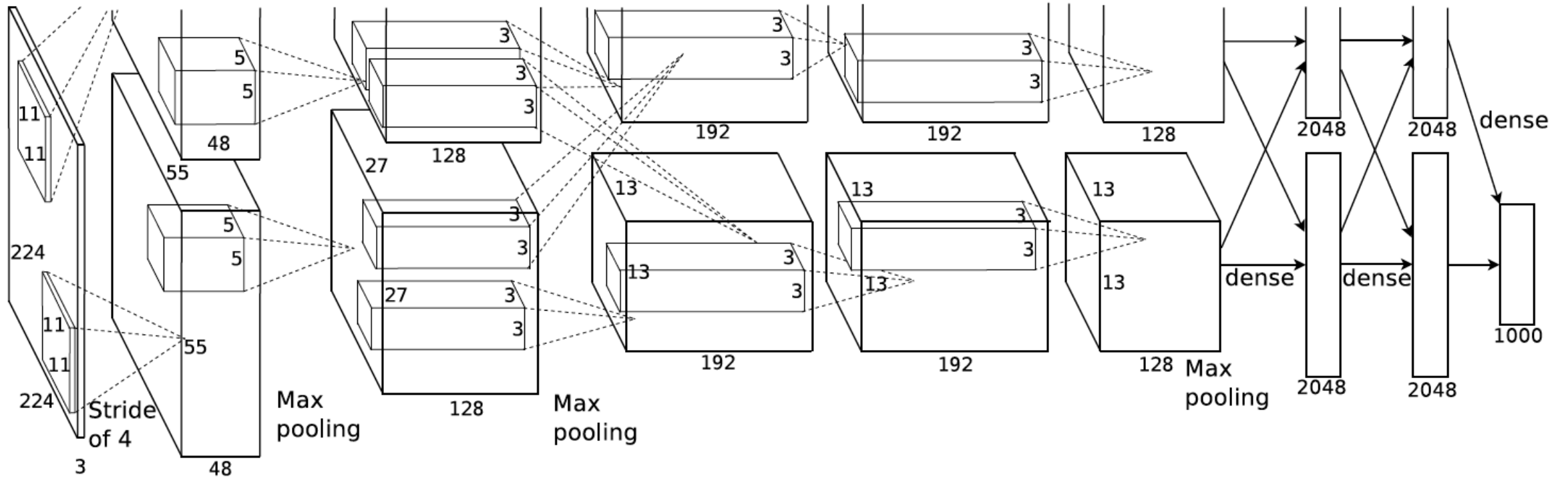
Multilayer Perceptron



This is the quintessential *Neural Network...*
...also called *Feed Forward Neural Net* or *Artificial Neural Net*

“Deep” Neural Networks

Modern *Deep Neural networks* add many hidden layers



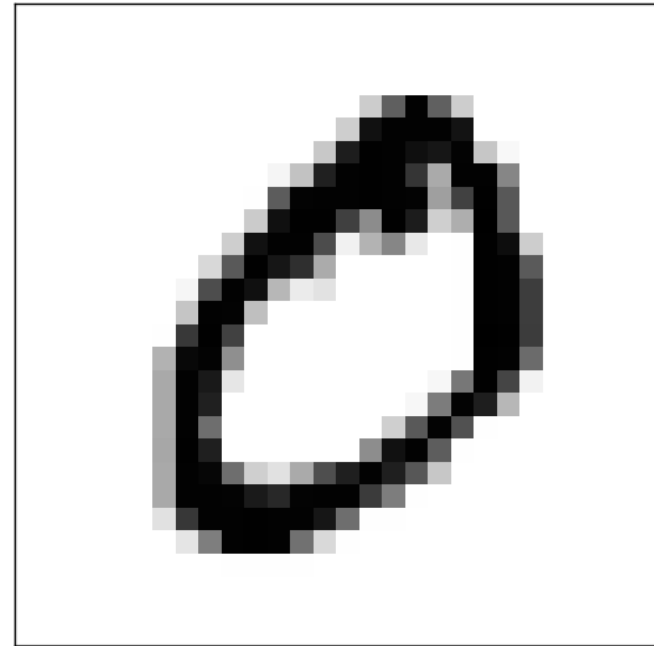
...and have many millions of parameters to learn

Handwritten Digit Classification

Classifying handwritten digits is the “Hello World” of NNs



Each character is centered
in a $28 \times 28 = 784$ pixel
grayscale image

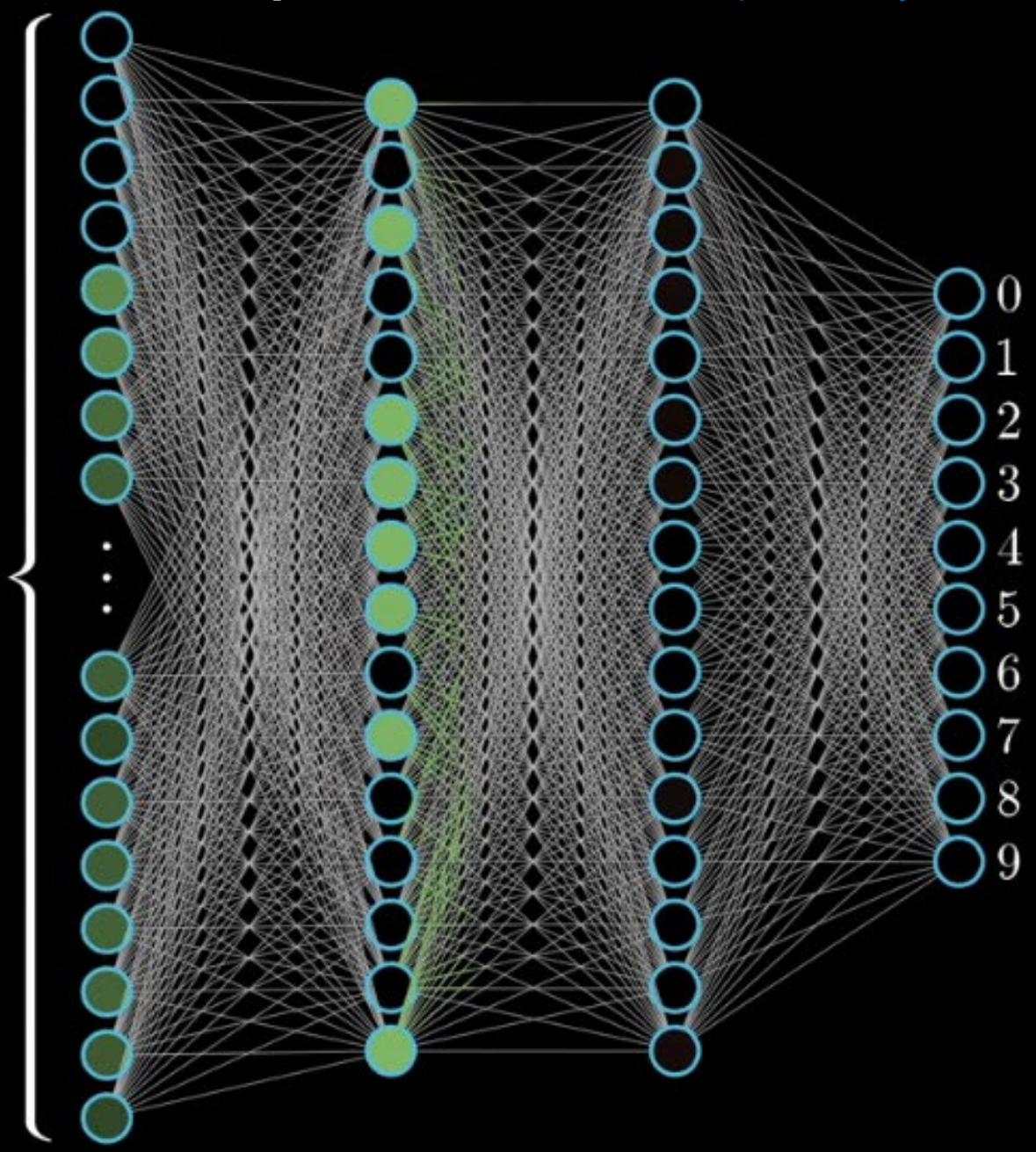


Modified National Institute of
Standards and Technology
(MNIST) database contains 60k
training and 10k test images

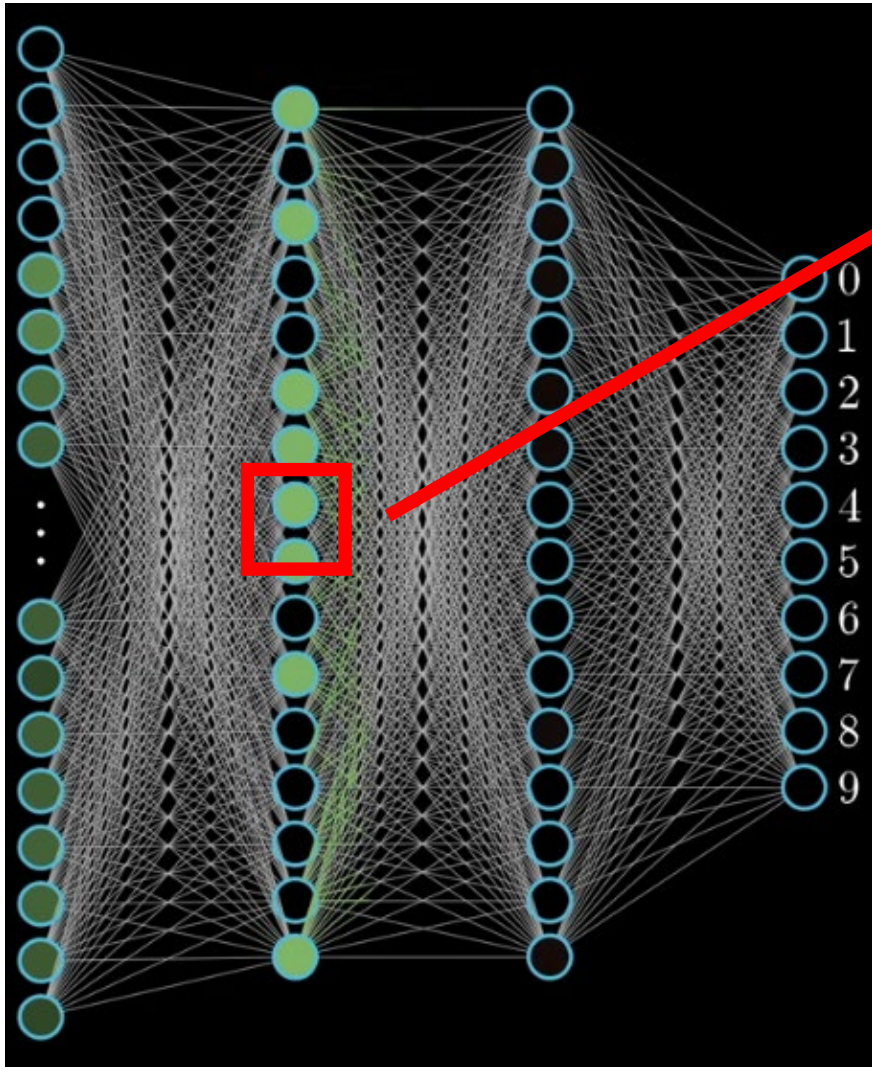


784

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



Feedforward Procedure



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

$$w_1x_1 + w_2x_2 + \dots + w_nx_n$$

Then applies a *nonlinear function* to the result

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

Often, we also introduce a constant *bias* parameter

Nonlinear Activation functions

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

$$\sigma(w_1x_1 + w_2x_2 + \dots + w_nx_n + b) = \sigma(w^T x + b)$$

An early choice was the *logistic function*,

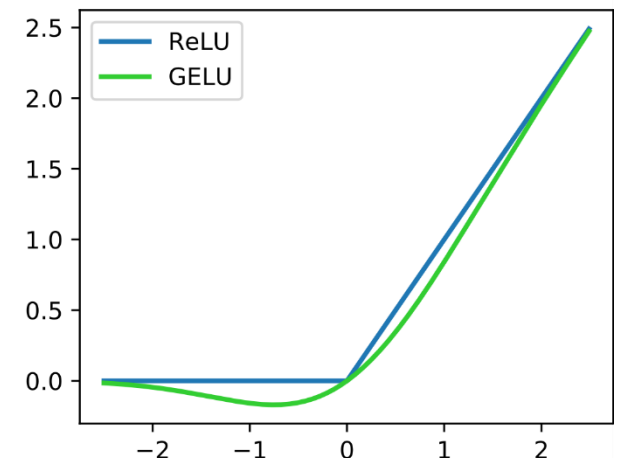
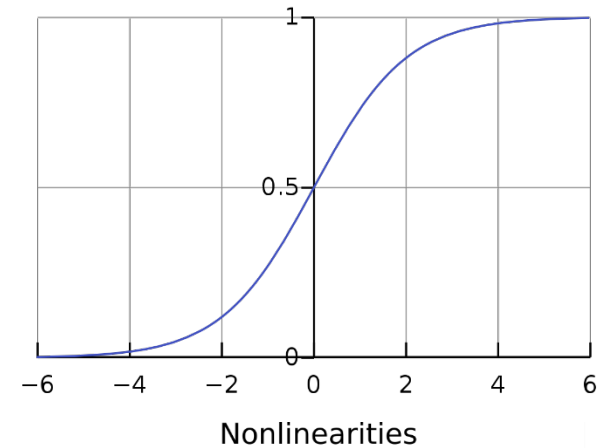
$$\sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

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

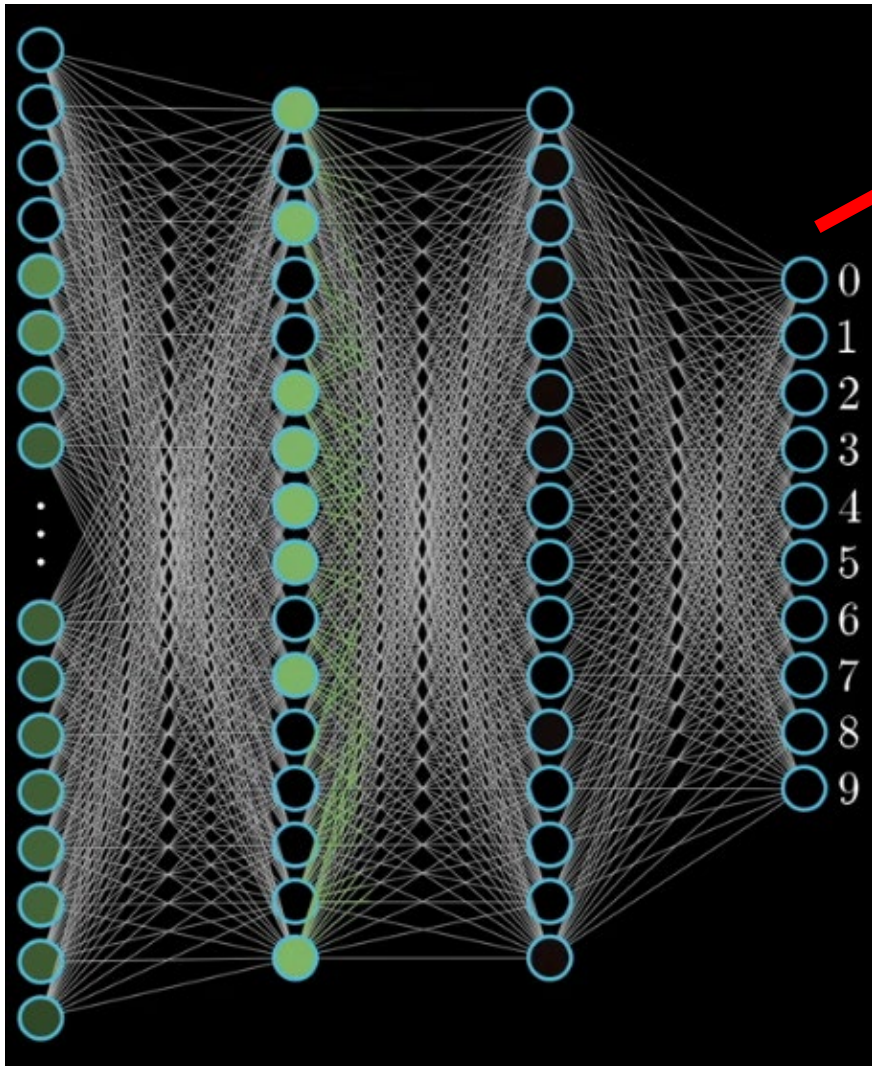
$$\sigma(w^T x + b) = \max(0, w^T x + b)$$

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

$$v = w^T x + b \quad \sigma(v) = v\Phi(v) \quad \leftarrow \text{Gaussian CDF}$$



Multilayer Perceptron



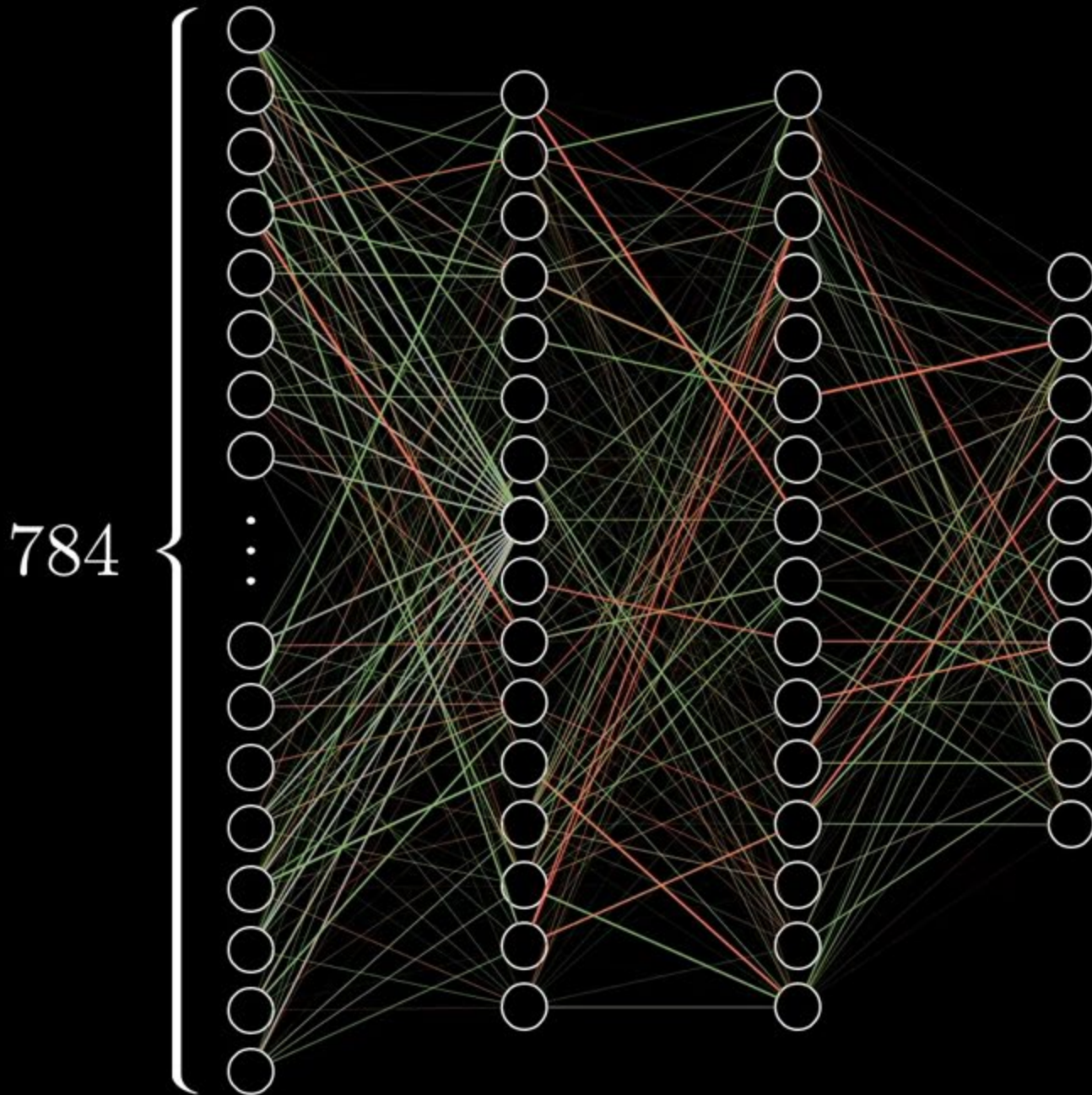
Final layer is typically a linear model...for classification this is a Logistic Regression

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

Vector of activations from previous layer

Recall that for multiclass logistic regression with K classes,

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



$$784 \times 16 + 16 \times 16 + 16 \times 10$$

weights

$$16 + 16 + 10$$

biases

13,002

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

Training Multilayer Perceptron

$$X^{\text{Train}} = \begin{matrix} \begin{matrix} 0 & 4 & 1 & 9 & 2 & 1 & 3 & 1 & 4 & 3 \\ 5 & 3 & 6 & 1 & 7 & 2 & 8 & 6 & 9 & 4 \\ 0 & 9 & 1 & 1 & 2 & 4 & 3 & 2 & 7 & 3 \\ 8 & 6 & 9 & 0 & 5 & 6 & 0 & 7 & 6 & 1 \\ 8 & 7 & 9 & 3 & 9 & 8 & 5 & 9 & 3 & 3 \\ 0 & 7 & 4 & 9 & 8 & 0 & 9 & 4 & 1 & 4 \\ 4 & 6 & 0 & 4 & 5 & 6 & 1 & 0 & 0 & 1 \\ 7 & 1 & 6 & 3 & 0 & 2 & 1 & 1 & 7 & 9 \\ 0 & 2 & 6 & 7 & 8 & 3 & 9 & 0 & 4 & 6 \\ 7 & 4 & 6 & 8 & 0 & 7 & 8 & 3 & 1 & 5 \end{matrix} \end{matrix}$$

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



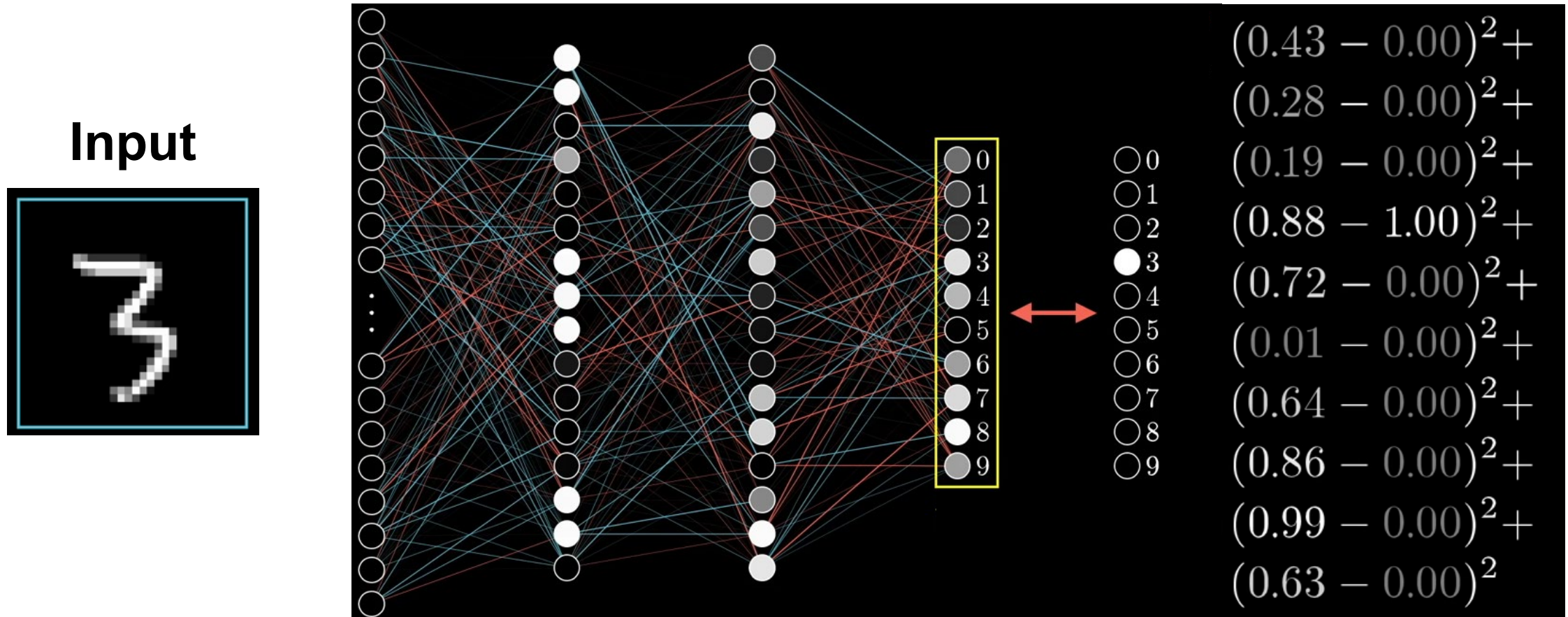
For each training example, predict label and adjust weights...



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


Training Multilayer Perceptron

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



Training Multilayer Perceptron

Our cost function for i^{th} input is error in terms of weights / biases...

$$\text{Cost}_i(w_1, \dots, w_n, b_1, \dots, b_n)$$


**13,002 Parameters
in this network**

...minimize cost over all training data...

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

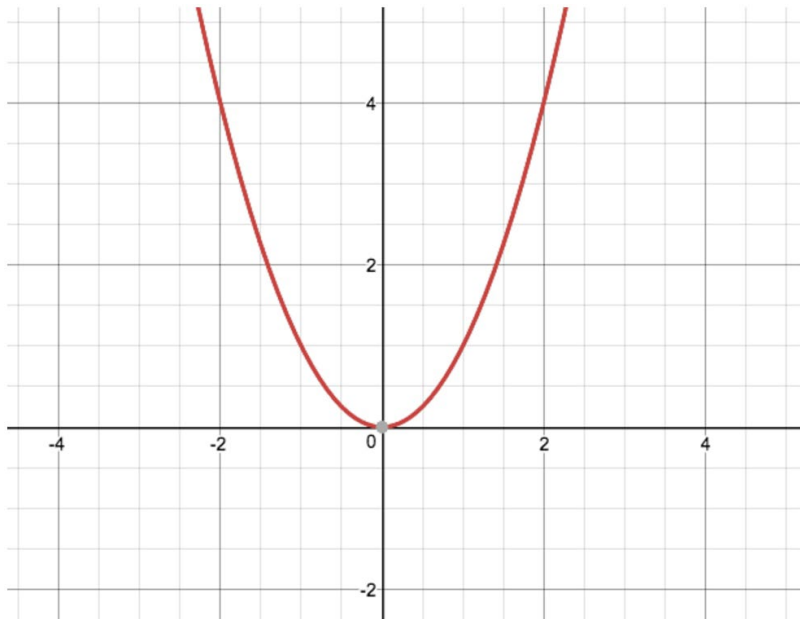
This is a super high-dimensional optimization (13,002 dimensions in this example)...how do we solve it?

Gradient descent!

Training Multilayer Perceptron

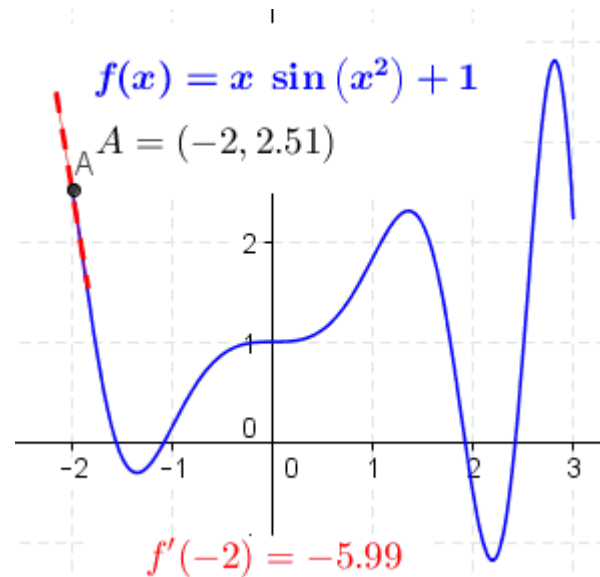
Need to find zero derivative (gradient) solution...

Convex Cost Function



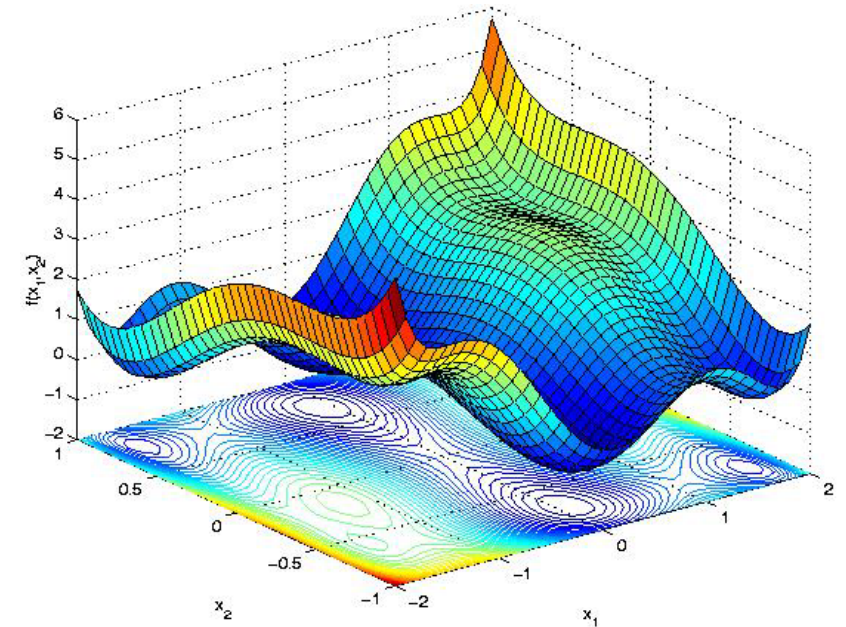
YAY!

Non-convex Cost Function



Boo!

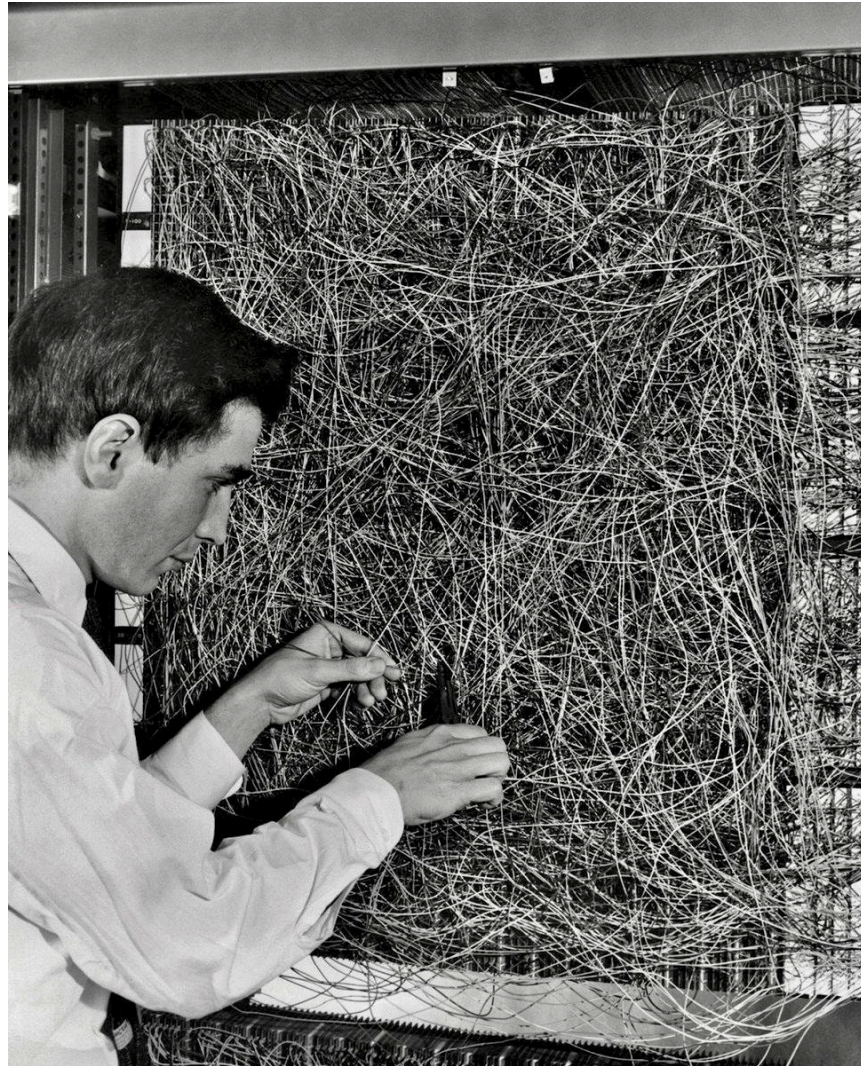
High-Dimensional Non-convex



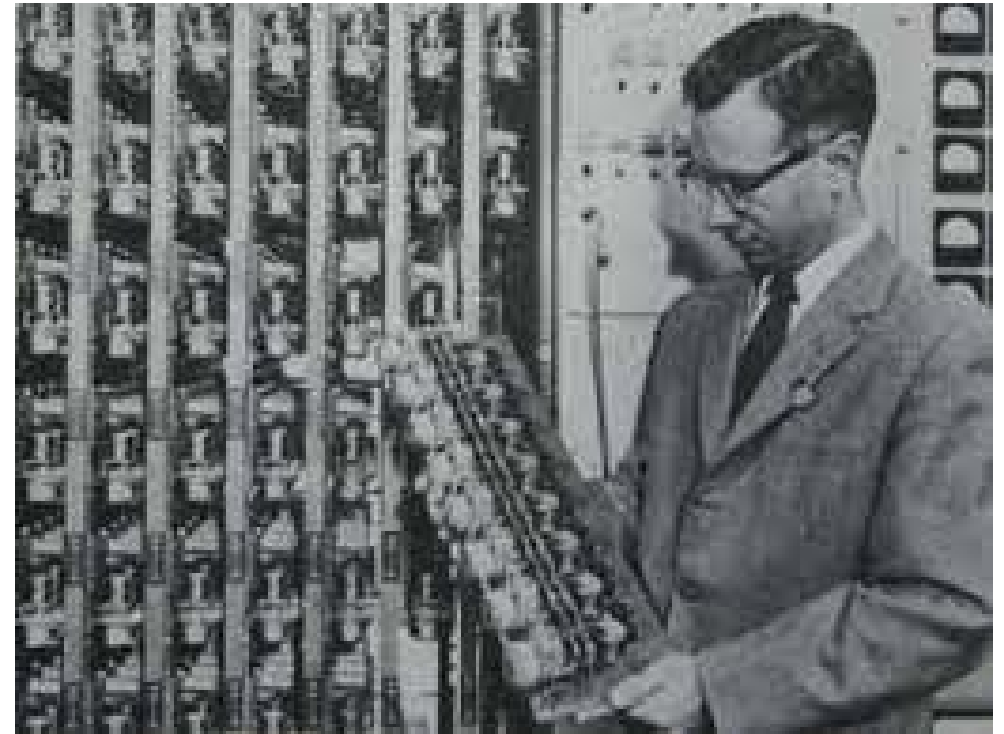
Super Boo!

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...

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

Dropped bias terms
for simplicity

Recall the **derivative chain rule**

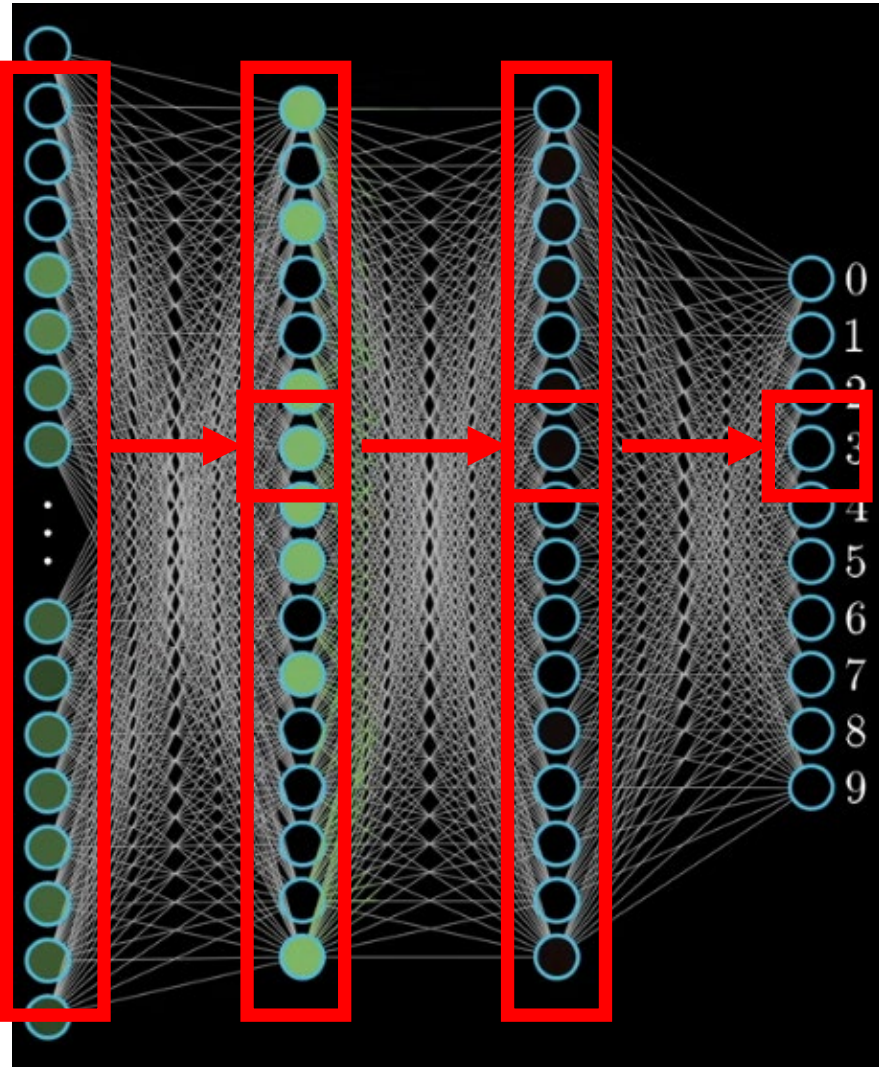
$$\frac{d}{dw} f(g(w)) = \underbrace{\frac{d}{dg(w)} f(g(w))}_{\text{Derivative of } f \text{ at its argument } g(w)} \left(\underbrace{\frac{d}{dw} g(w)}_{\text{Differentiate } g \text{ with respect to } w} \right)$$

Derivative of f at its argument $g(w)$
e.g. treat $g(w)$ as a variable

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...

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

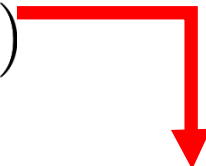
And so on...

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

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**
- Motivating Bayesian Deep Learning

Some Drawbacks of Standard Deep Learning

- Predictions can be “brittle” (i.e. very discontinuous w.r.t. input)
- Fail to generalize outside 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

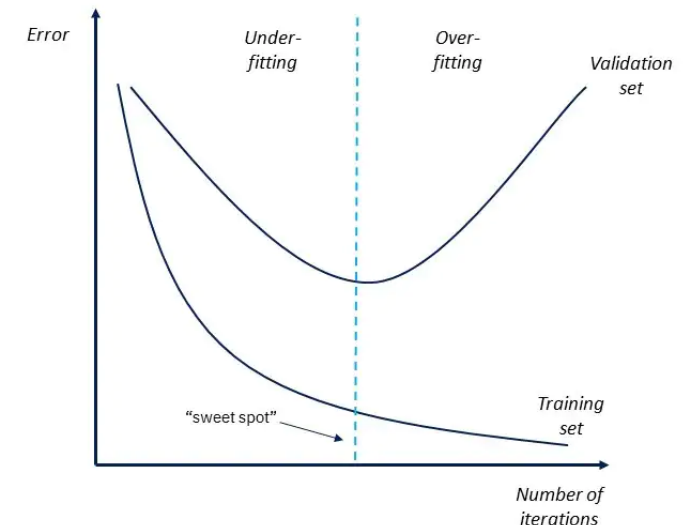
Regularization

With four parameters I can fit an elephant. With five I can make him wiggle his trunk. - John von Neumann

$$w = \arg \min_w \text{Cost}(w) + \alpha \cdot \text{Regularizer}(\text{Model})$$

Our example model has 13,002 parameters...that's a lot of elephants!
Regularization is critical to avoid overfitting...

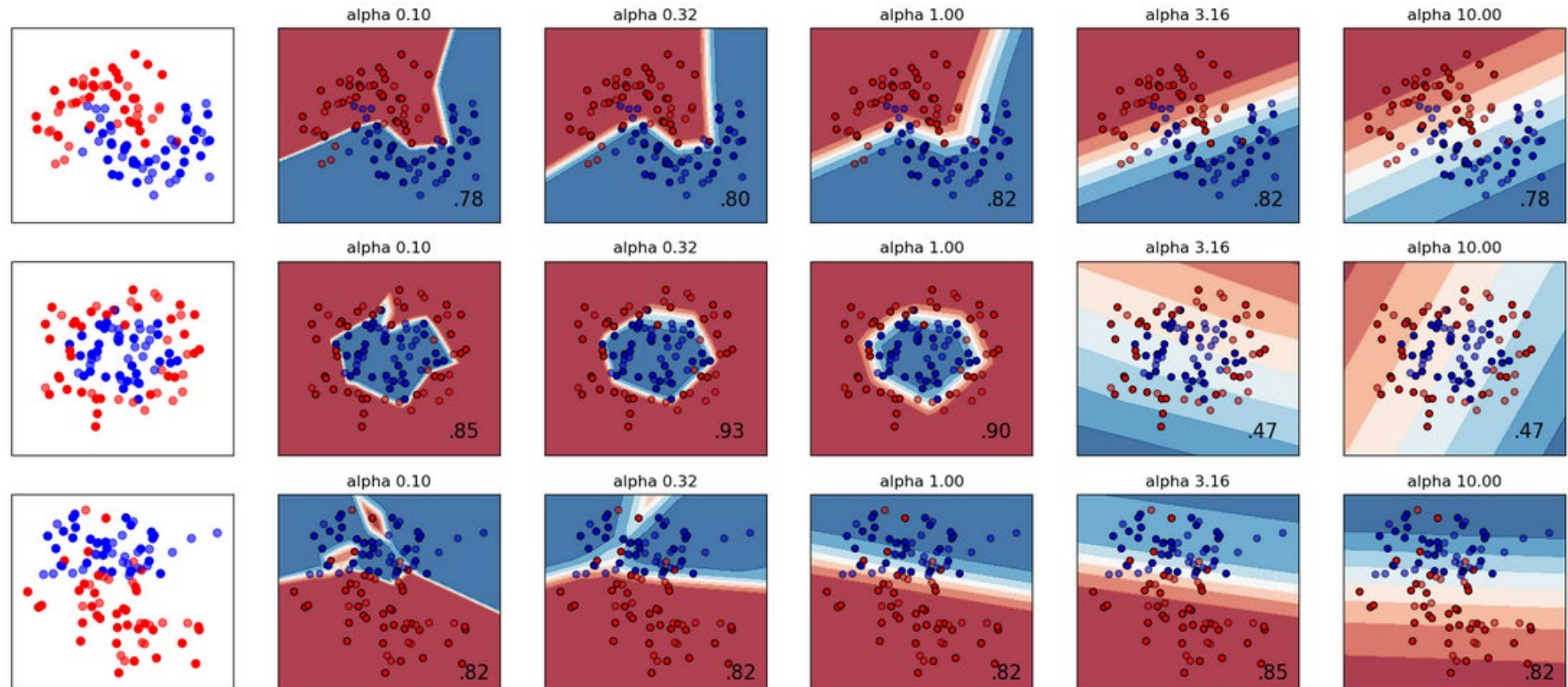
...numerous regularization schemes are used in training neural networks



Regularization : Weight Decay

In neural network speak, adding an L2 penalty is called *weight decay*

$$w = \arg \min_w \text{Cost}(w) + \frac{\alpha}{2} \|w\|^2$$



Learning sensitive to regularization strength and too expensive to do cross-validation

Regularization

- L1 regularization and L1+L2 (elastic net) regularization
- **Dropout** Each iteration randomly selects a small number of edges to temporarily exclude from the network (weights=0)
 - **Intuition** Avoids predictions that are overly sensitive to any small number of edges
- **Early stopping** Just as it sounds...stop the network before reaching a local minimum...dumb-but-effective

Brittleness : Discontinuities in Predictions

Nearly imperceptible changes to input change prediction



All images in right column predicted as “ostrich”

Safety Concerns



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

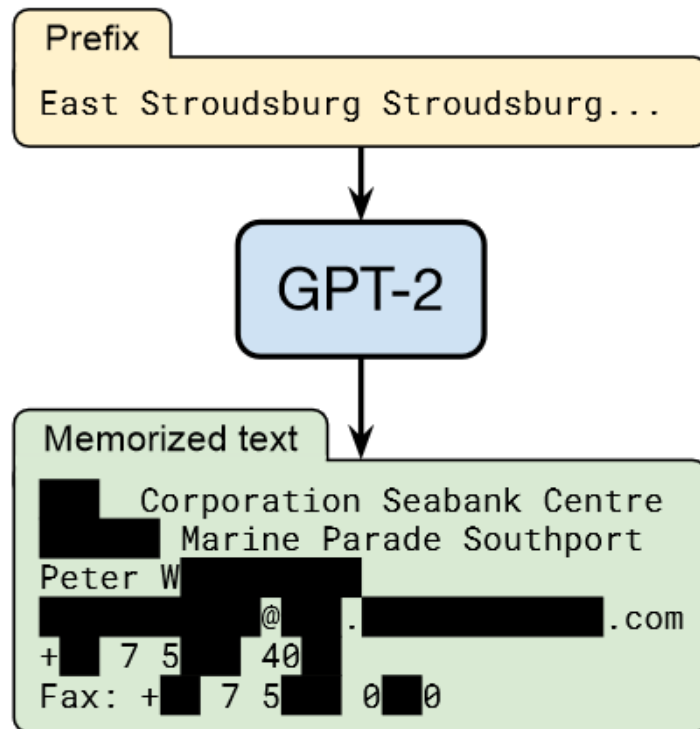
- Artistic graffiti
- Subtle graffiti
- Poster

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

Does not require knowledge of network internals

Privacy Concerns

Large DNNs capable of memorizing training data...



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)

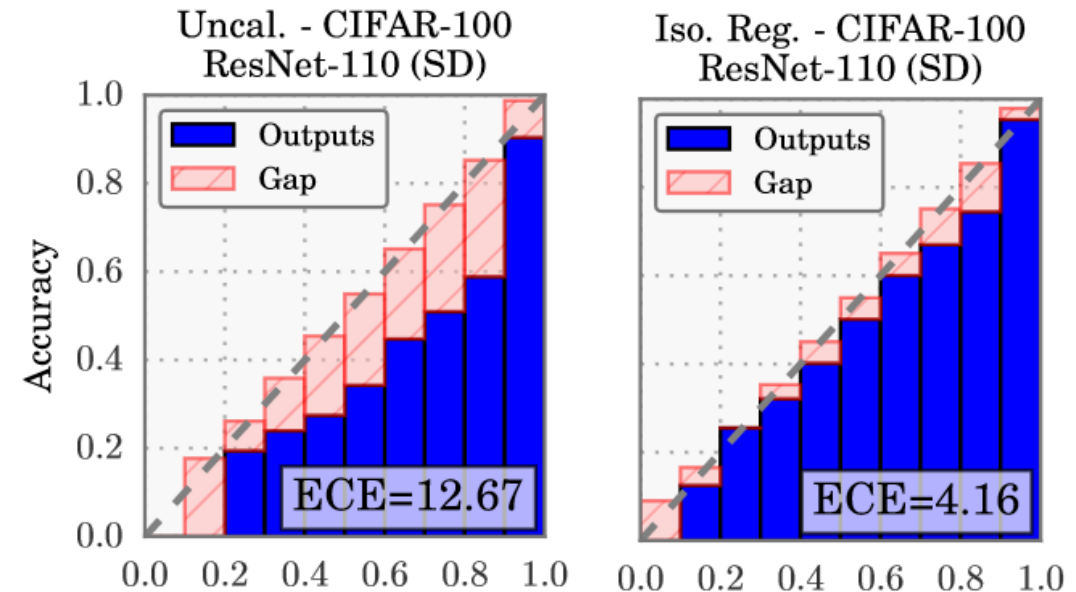
Figure 1: **Our extraction attack.** Given query access to a neural network language model, we extract an individual person's name, email address, phone number, fax number, and physical address. The example in this figure shows information that is all accurate so we redact it to protect privacy.

Outline

- Artificial Neural Network (ANN) : A Review
- Shortcomings of Standard Deep Learning
- **Motivating 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 (aleatoric)
- 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

Probabilistic Perspectives on Deep learning

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

$$\text{Prediction} \longrightarrow y = f_{\theta}(x) \longleftarrow \text{Input}$$

↑
Network Parameters: Weights,
architecture, activation funcs

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

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

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

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

ANN outputs

Bayesian Perspective on Deep Learning

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

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



Think of this
as a prior
over models

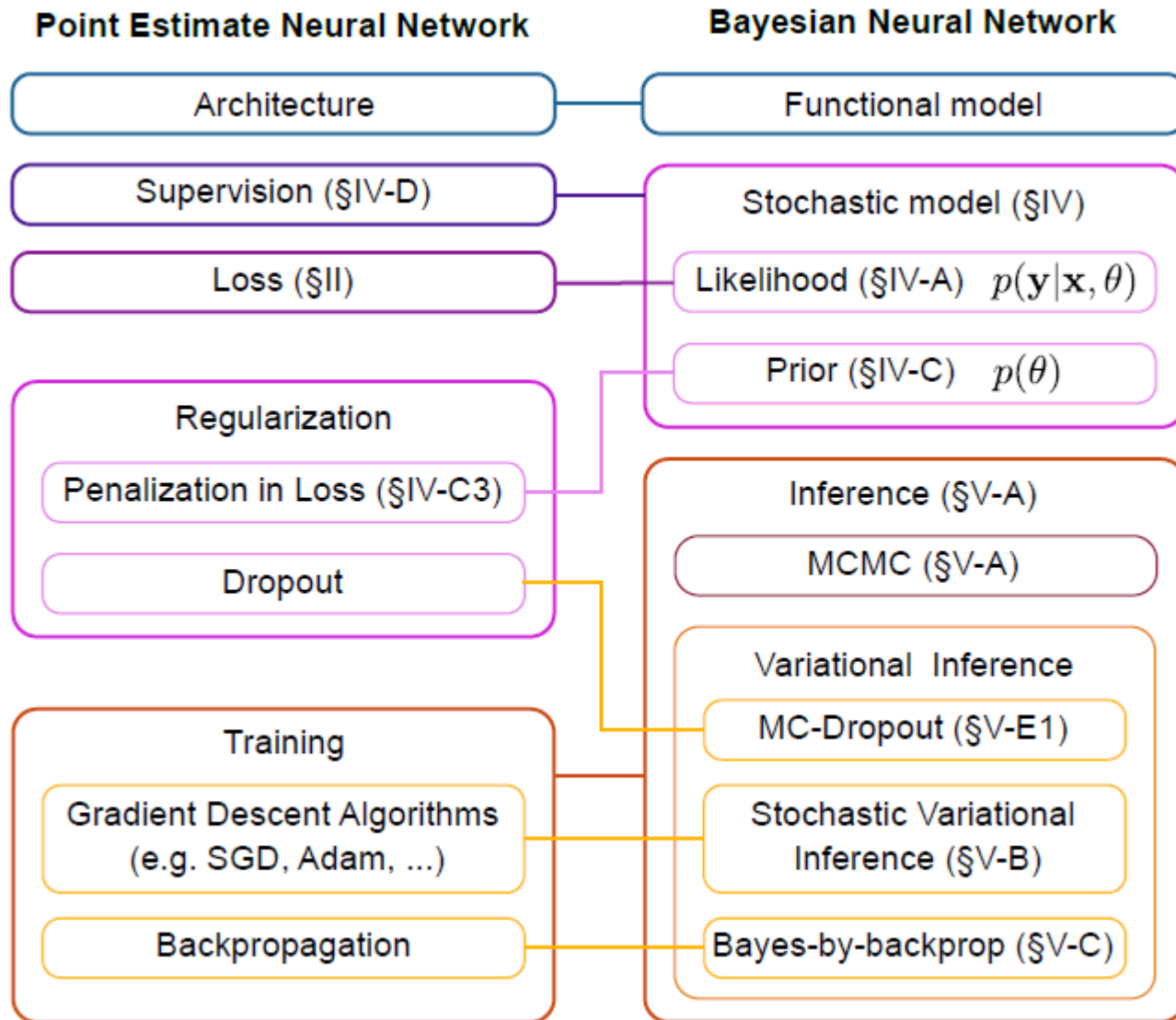
Benefits

- Can compute posterior over all networks $p(\theta | x)$
- Or marginalize over network parameters $p(y | x) = \int p(\theta, y | x) d\theta$
- 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

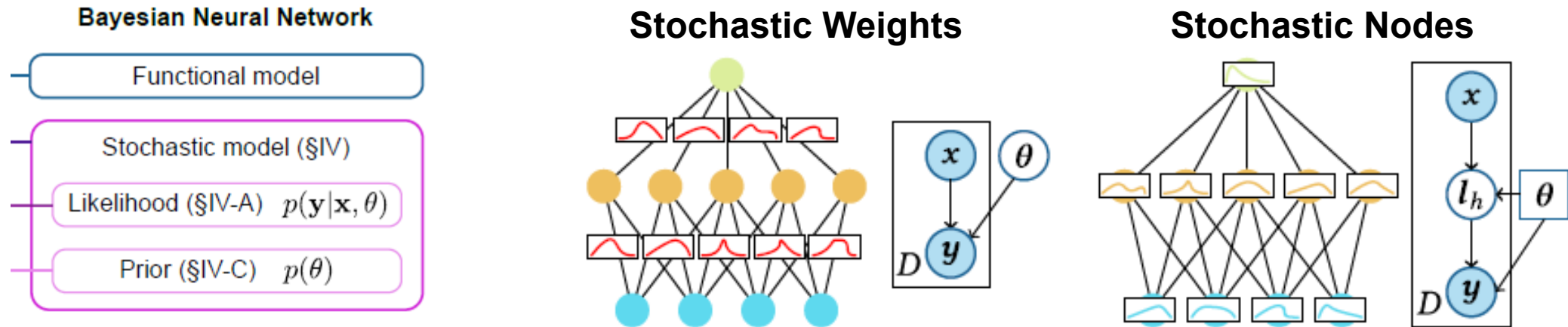


The learning process of Bayesian DL fundamentally differs from point estimate ANNs

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

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

Bayesian Neural Network



- 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
- Simple examples based on Gaussian or Categorical outputs:

$$\theta \sim p(\theta) = \mathcal{N}(\mu, \Sigma),$$
$$\mathbf{y} \sim p(\mathbf{y}|\mathbf{x}, \theta) = \mathcal{N}(\Phi_{\theta}(\mathbf{x}), \Sigma)$$

$$\theta \sim p(\theta) = \mathcal{N}(\mu, \Sigma),$$
$$\mathbf{y} \sim p(\mathbf{y}|\mathbf{x}, \theta) = \text{Cat}(\Phi_{\theta}(\mathbf{x}))$$

Bayesian Neural Network

Many different constructions, but all essentially a stochastic ANN

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

$$\begin{aligned}l_0 &= \mathbf{x}, \\l_i &= s_i(\mathbf{W}_i l_{i-1} + \mathbf{b}_i) \quad \forall i \in [1, n] \\y &= l_n.\end{aligned}$$

Two main types of BNNs

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

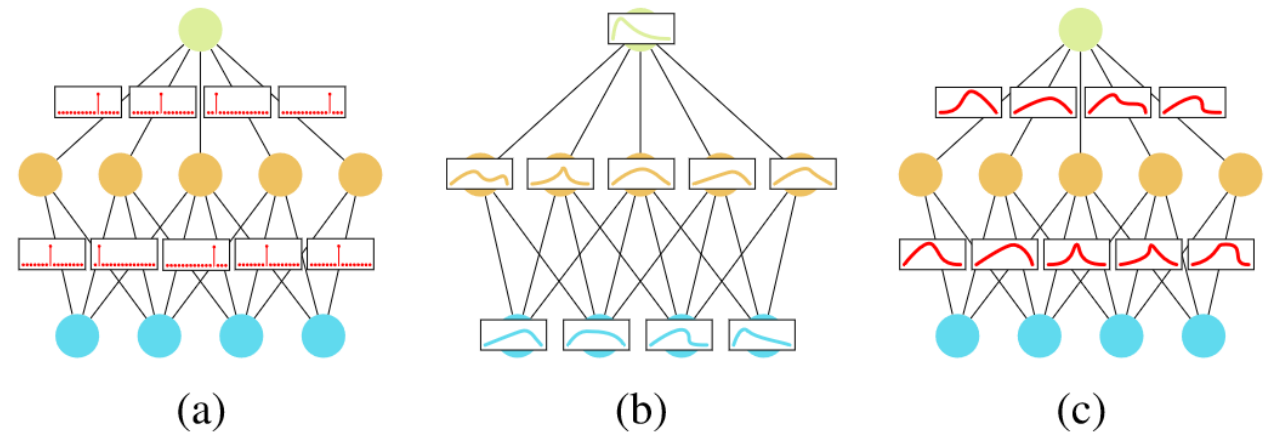


Fig. 3: (a) Point estimate neural network, (b) stochastic neural network with a probability distribution for the activations, and (c) stochastic neural network with a probability distribution over the weights.

Bayesian Neural Network

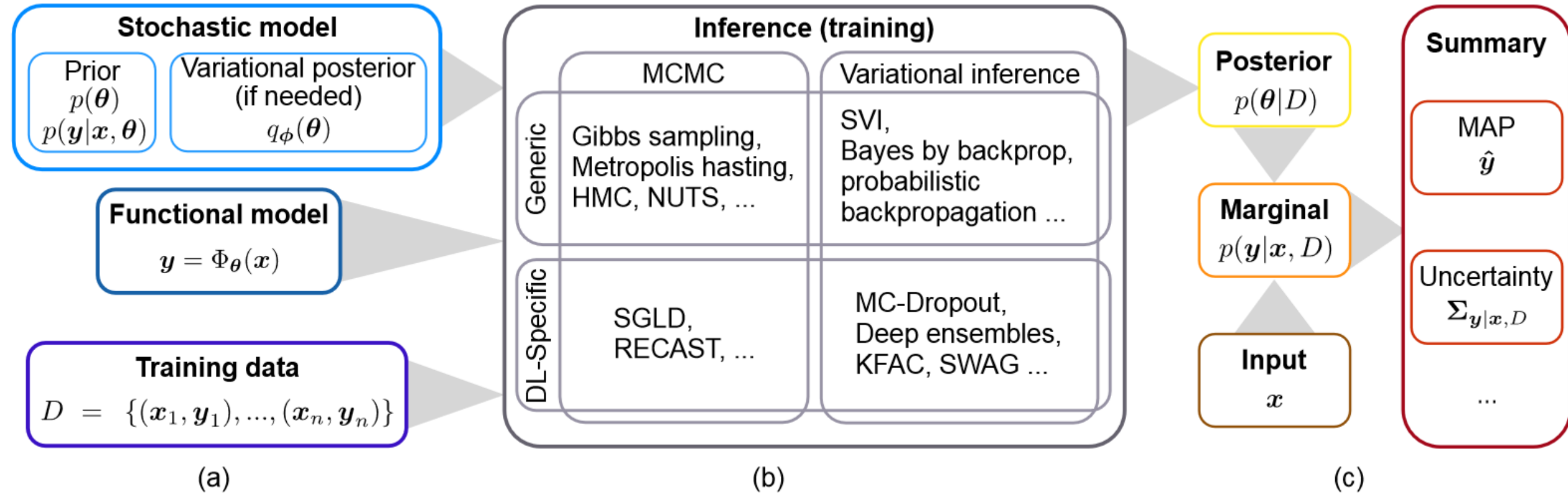


Fig. 2: Workflow to design (a), train (b) and use a BNN for predictions (c).

Inference in a BNN

Given training data $D=\{D_x,D_y\}$ compute posterior over network params,

$$p(\boldsymbol{\theta}|D) = \frac{p(D_y|D_x, \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} p(D_y|D_x, \boldsymbol{\theta}')p(\boldsymbol{\theta}')d\boldsymbol{\theta}'} \propto p(D_y|D_x, \boldsymbol{\theta})p(\boldsymbol{\theta}).$$

- 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 weve learned

Prediction in a BNN

When predicting we often marginalize over network parameters,

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

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

Given samples from posterior,

$$\theta_i \sim p(\theta|D);$$

Can sample predictions in feedforward process,

$$\mathbf{y}_i = \Phi_{\theta_i}(\mathbf{x});$$

Algorithm 1 Inference procedure for a BNN.

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

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

 Draw $\theta_i \sim p(\theta|D)$;

$\mathbf{y}_i = \Phi_{\theta_i}(\mathbf{x})$;

end for

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

Training Data

Training Labels

Prediction in a BNN

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

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

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

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

Sample covariance can be used to quantify predictive uncertainty,

$$\Sigma_{\mathbf{y}|\mathbf{x}, D} = \frac{1}{|\Theta|-1} \sum_{\boldsymbol{\theta}_i \in \Theta} (\Phi_{\boldsymbol{\theta}_i}(\mathbf{x}) - \hat{\mathbf{y}}) (\Phi_{\boldsymbol{\theta}_i}(\mathbf{x}) - \hat{\mathbf{y}})^\top.$$

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

Prediction in a BNN

One can also consider the empirical distribution over predictions,

$$\hat{p} = \frac{1}{|\Theta|} \sum_{\theta_i \in \Theta} \Phi_{\theta_i}(\mathbf{x}).$$

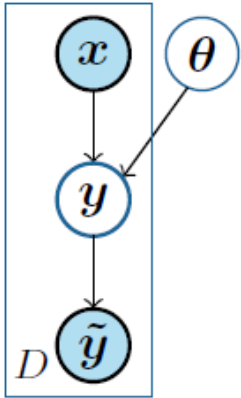
The maximum a posteriori (MAP) prediction is then,

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

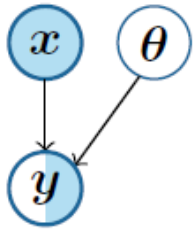
- Uncertainty given via the empirical entropy
- Straightforward for classification tasks
- Continuous (i.e. regression) predictions require density estimation

Generalizing Beyond Supervised Learning

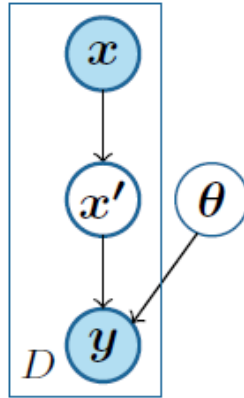
Bayesian DL can effectively use unlabeled data and uncertain labels...



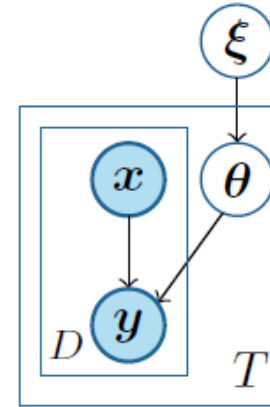
(a) Noisy labels



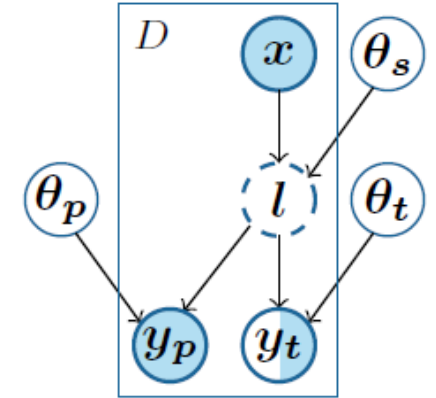
(b) Semi-supervised learning



(c) Data augmentation



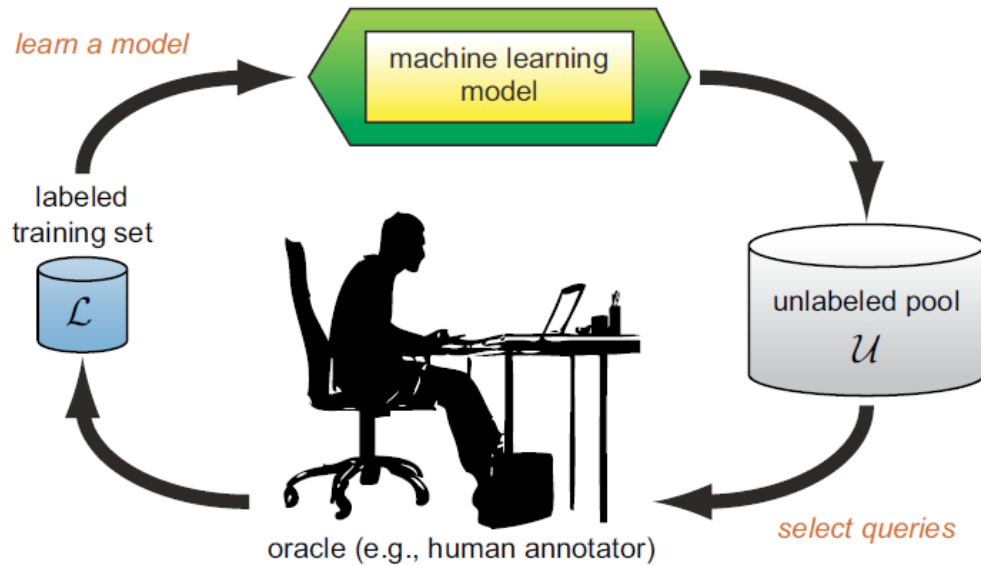
(d) Meta-learning



(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 $U \neq \emptyset$ and $\Sigma_{\mathbf{y}|\mathbf{x}_{max},D} < \text{threshold}$ and $C < \text{MaxC}$
do

Draw $\Theta = \{\theta_i \sim p(\theta|D) | i \in [0, N)\}$;

for $x \in U$ **do**

$$\Sigma_{\mathbf{y}|\mathbf{x},D} = \frac{1}{|\Theta|-1} \sum_{\theta_i \in \Theta} (\Phi_{\theta_i}(\mathbf{x}) - \hat{\mathbf{y}}) (\Phi_{\theta_i}(\mathbf{x}) - \hat{\mathbf{y}})^T;$$

if $\Sigma_{\mathbf{y}|\mathbf{x},D} > \Sigma_{\mathbf{y}|\mathbf{x}_{max},D}$ **then**

$\mathbf{x}_{max} = \mathbf{x}$;

end if

end for

$D_{\mathbf{x}} = D_{\mathbf{x}} \cup \{\mathbf{x}_{max}\}$;

$D_{\mathbf{y}} = D_{\mathbf{y}} \cup \{\text{Oracle}(\mathbf{x}_{max})\}$;

$U = U \setminus \{\mathbf{x}_{max}\}$;

$C = C + 1$;

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)

Variational Inference in a BNN

Recall variational inference minimizes Kullback-Leibler divergence,

$$D_{\text{KL}}(q_{\phi} \| P(H, D)) = \int_H q_{\phi}(H') \log \frac{q_{\phi}(H')}{P(H', D)} dH'$$

- Where H is hidden (latent) and D is observed data
- $q_{\phi}(H)$ is variational approximation of posterior $P(H | D)$
- Can be done in a BNN using *stochastic variational inference (SVI)*
- Required gradients of variational objective w.r.t. parameters ϕ
- **Problem** Straightforward backpropagation of network to compute gradients doesn't work with stochastic representation
- Need to adapt SVI approach for Bayesian DL

Gaussian Reparameterization

Suppose we want to sample a Gaussian RV,

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

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

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

Gaussians are closed under linear transformations so,

$$X = \mu + \sigma Z \sim \mathcal{N}(\mu, \sigma^2)$$

- Defines X as a *deterministic transformation* of Z
- Yields samples from the correct distribution
- Instance of general technique called the **reparameterization trick**

Bayes By Backprop

Algorithm 5 Bayes-by-backprop algorithm.

Variational Bound →

```
 $\phi = \phi_0;$   
for  $i = 0$  to  $N$  do  
  Draw  $\varepsilon \sim q(\varepsilon);$   
   $\theta = t(\varepsilon, \phi);$   
   $f(\theta, \phi) = \log(q_\phi(\theta)) - \log(p(D_y|D_x, \theta)p(\theta));$   
   $\Delta_\phi f = \text{backprop}_\phi(f);$   
   $\phi = \phi - \alpha \Delta_\phi f;$   
end for
```

Reparameterization Trick

- Define nonvariational noise parameter $\varepsilon \sim q(\varepsilon)$
- Define deterministic transformation $\theta = t(\varepsilon, \phi)$
- Transformed parameters are from variational distribution $\theta \sim q_\phi$
- Allows use of standard backpropagation

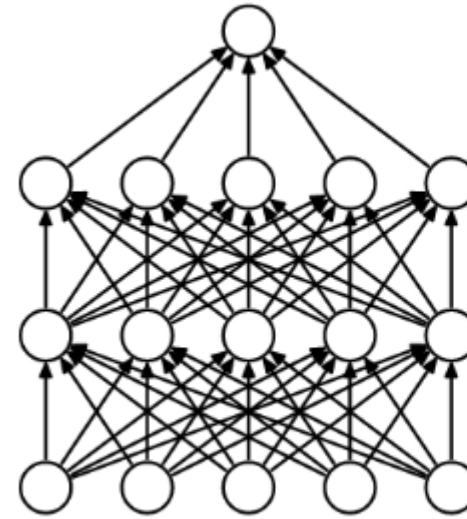
Scaling Up Inference in BNNs

- Learning in standard DNNs is expensive
- Inference in BNNs is even more costly
- Being only “approximately Bayesian” may be sufficient to achieve well-calibrated model (*Kristiadi et al. 2020*)
- Some simple approaches to be approximately Bayesian,
 - Monte Carlo Dropout
 - Bayes via Stochastic Gradient Descent (VI and MCMC methods)
 - Limit inference to last few layers
- These methods scale to larger instances and yield better uncertainty calibration than non-Bayesian point estimates

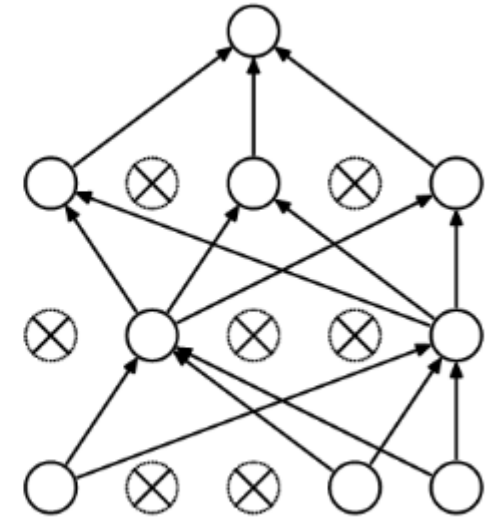
Monte Carlo Dropout

Dropout

- Typically used as regularizer in training
- Each grad update randomly remove edges
- 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,

$$z_{i,j} \sim \text{Bernoulli}(p_i),$$
$$W_i = M_i \cdot \text{diag}(z_i),$$

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	Can be combined
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	Can be combined
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	

Conclusions

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” perform decently