Reliable and Interpretable Artificial Intelligence

Dr. Dana Drachsler-Cohen
ETH Zurich

Fall 2017

So far: Synthesis

First, studied algorithms that accurately solve a problem

- e.g., given specification or by learning an intent from examples
- accuracy is guaranteed, obtaining a specification is not trivial

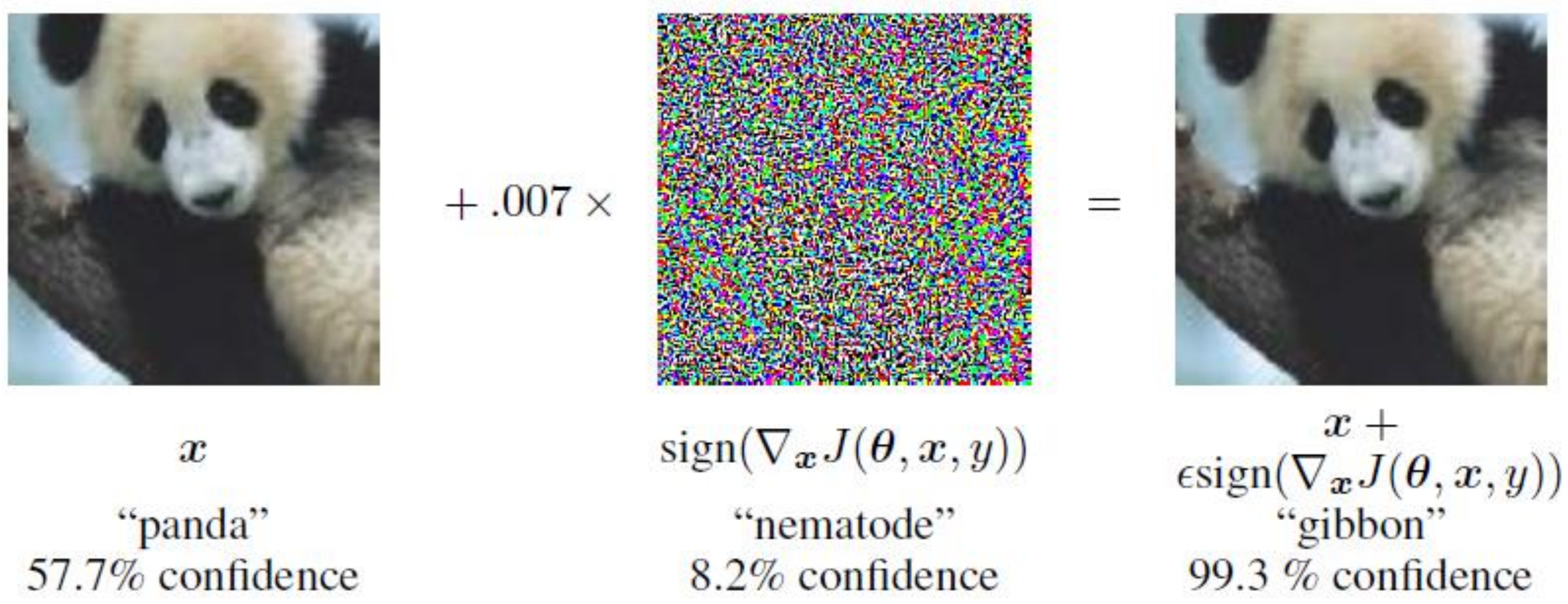
Then, studied synthesis algorithms that use statistical information

- e.g., Slang, Neural Turing Machine
- models with high accuracy, require labeled data
- Does not impose burden on the user

We will learn that these models are easily fooled, i.e., they are not robust

Part II: Robustness of Deep Learning

Examples of (lack of) Robustness



taken from: Explaining and Harnessing Adversarial Examples. Goodfellow, Shlens, Szegedy, ICLR ‘15

Today: Background on Deep Learning

(the background needed to understand robustness)

Technical Material (Today)

Perceptron

(Empirical) Loss functions

Training with gradient descent

Deep learning models

Back propagation

Motivation: Classify Handwritten Digits

Dataset: MNIST

Images consist of 28 x 28 pixels

(hence, 784 pixels)

The space of possible images is: $[0,1]^{784}$ (not countable)

A black pixel has value 0, a white pixel has value 1

Goal: a model that classifies images to digits: $f: [0,1]^{784} \rightarrow \{0, \dots, 9\}$

Classification through Machine Learning

Take a data-driven approach and learn a model (function) f from data

$f: I \to C$ approximates the optimal function $f^*: I \to C$

A model is an architecture with real-valued weights and biases

The architecture defines the space of expressible models

Example of a Model f: A Perceptron

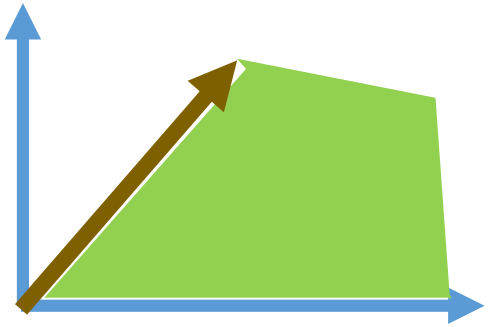
A classifier parametrized by $w_0, \ldots, w_{n-1}$ and b

$\bar{x} = (x_0, \ldots, x_{n-1})$

$$f(x) = \begin{cases} 1 & \Sigma w_i x_i + b > 0 \\ 0 & otherwise \end{cases}$$

$0/1$

Linearly separates the space



Evaluating Models: Loss Function

Goal: model and optimal classifier are equal:

$$\forall i. f^*(i) = f(i)$$

Induces a loss function which measures how good a classifier is:

$$\Sigma_{i \in I}[f^*(i) \neq f(i)]$$

where $[\cdot]$ is the Iverson brackets (1 if condition is true, 0 otherwise)

The smaller the loss, the better the classifier

Goal: find weights and biases of the model f which minimize loss

Empirical Loss

Practical challenge: not all labels of input data I are given

Approach: estimate the loss function on some of the labelled inputs D

Given the labelled data D, compute the empirical loss: $\Sigma_{i \in D}[f^*(i) \neq f(i)]$

Operationally:

To avoid overfitting to D, split D into training D_{Tr} and test D_{Te} sets

Learn model by minimizing the loss on D_{Tr}, estimate loss on D_{Te}

Tuning Model's Parameters

The optimal solution of $\Sigma_{i \in D_{Tr}} [f^*(i) \neq f(i)]$ is a **global minima**

If $\Sigma_{i \in D_{Tr}} [f^*(i) \neq f(i)]$ were differentiable, we could compute the minima by checking when the derivative is zero

Define (a different) loss function that is differentiable and find a point that nullifies its derivative

Example of Differentiable Loss Function

The mean squared error (MSE):

$$\mathrm{MSE} = \Sigma_{i \in D_{Tr}} \left(f^*(i) - f(i) \right)^2$$

If the model consists of a single weight: $f(i) = w \cdot i$, then:

$$MSE = \Sigma_{i \in D_{Tr}} (f^*(i) - w \cdot i)^2$$

The minimum nullifies the derivative:

$$\Sigma_{i \in D_{Tr}} 2(f^*(i) - w \cdot i) \cdot (-i) = 0$$

Can compute the best model (i.e., w) analytically

Gradients

For $f(w_0, \ldots,\ w_{n-1}, b)$ with (>1 parameter), derivative of the

$\mathrm{MSE} = \Sigma_{i \in D_{Tr}} \left(f^*(i) - f(i)\right)^2$ is generalized to gradient

A gradient is a vector defined by partial derivatives of the variables

$$\nabla MSE = \left(\frac{\partial MSE}{\partial w_0}, \ldots, \frac{\partial MSE}{\partial w_{n-1}}, \frac{\partial MSE}{\partial b}\right)$$

Minimum nullifies ∇MSE in all dimensions → Hard to compute analytically

Example: Mean Squared Error Gradients

Example: $f(i_1, i_2) = w_1 \cdot i_1 + w_2 \cdot i_2 + b$

$$\mathrm{MSE} = \Sigma_{(i_1,i_2)\in D_{Tr}} \left(f^*(i_1, i_2) - f(i_1, i_2)\right)^2$$

$$\nabla MSE = \Big(\overbrace{\Sigma_{(i_1,i_2)\in D_{Tr}} 2\left(f^*(i_1, i_2) - (w_1 \cdot i_1 + w_2 \cdot i_2 + b)\right) \cdot (-\, i_1)}^{\frac{\partial MSE}{\partial w_1}},$$

$$\underbrace{\Sigma_{(i_1,i_2)\in D_{Tr}} 2(f^*(i_1, i_2) - (w_1 \cdot i_1 + w_2 \cdot i_2 + b)) \cdot (-\, i_2)}_{\frac{\partial MSE}{\partial w_2}},$$

$$\underbrace{\Sigma_{(i_1,i_2)\in D_{Tr}} 2(f^*(i_1, i_2) - (w_1 \cdot i_1 + w_2 \cdot i_2 + b)) \cdot (-\, 1)}_{\frac{\partial MSE}{\partial b}}\Big)$$

Finding Minima via Gradient Descent

1. **Initialize** randomly with certain values for weights/bias a_0 , $\mathrm{set} \quad \mathrm{n} = 0$

2. Compute the **gradient** of the MSE at a_n

3. The **next point** is the one maximizing the decrease in MSE

   $a_{n+1} = a_n - \gamma \nabla MSE(a_n)$ (γ is called the learning rate)

   n = n + 1

4. If **loss is small enough**, complete; otherwise, repeat 2

Recap

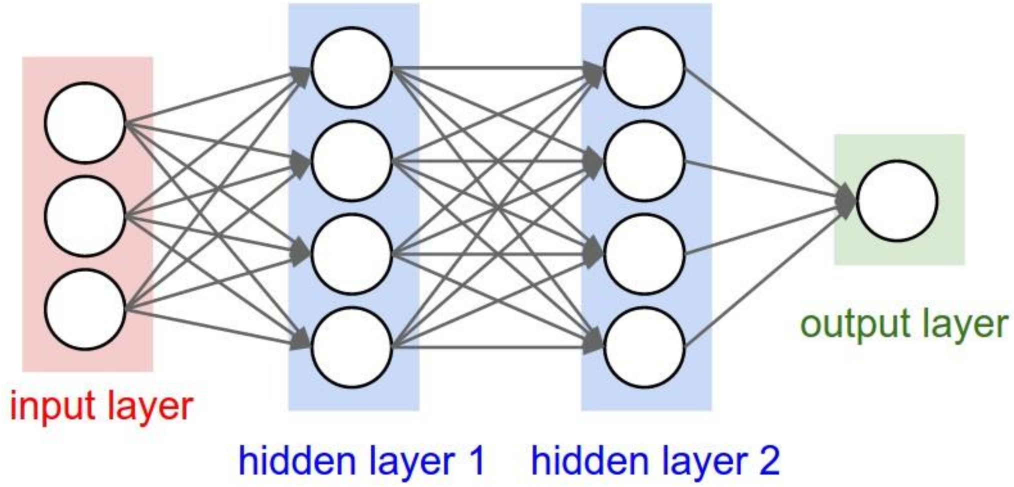
Models approximate a function (that is hard to find algorithmically)

A model is an architecture (e.g., perceptron) and a set of parameters

Parameters are tuned to have lowest loss via gradient descent

(A very simplified introduction, omits many important details: regularization, weight initialization, validation set,…)

Deep Learning



Perceptrons are too simplistic models

Deep models **combine** multiple simple models (e.g., perceptron)

A deep model is a directed graph of **neurons**

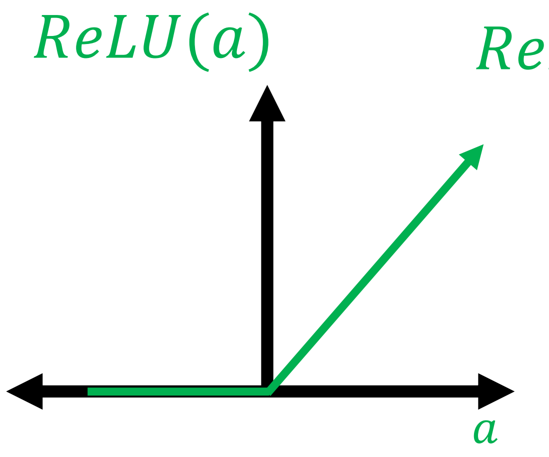
Neurons are organized in **layers**

A neuron is a simple model followed by an **activation function**

Activation Functions

Determine whether to propagate the output of the neuron

Examples: tangent, sigmoid, ReLU

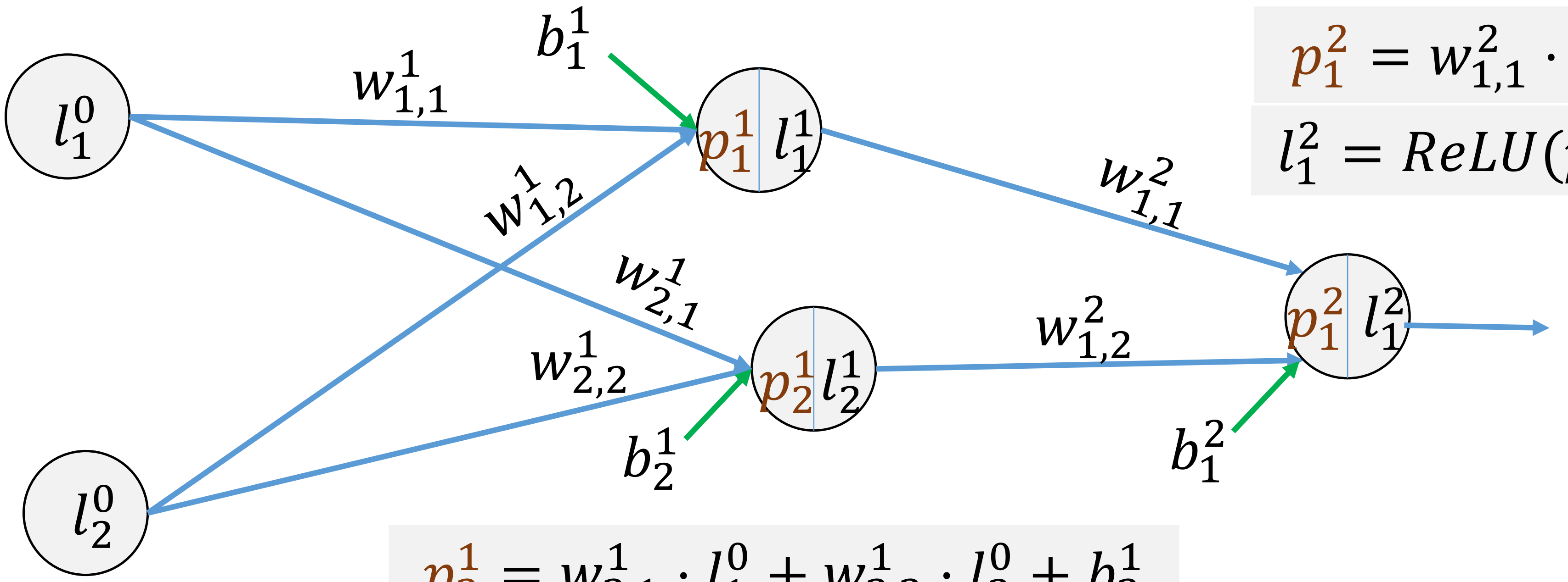


$$ReLU(a) = \max(0, a)$$

Neurons and Layers: Notation

$$p_1^1 = w_{1,1}^1 \cdot l_1^0 + w_{1,2}^1 \cdot l_2^0 + b_1^1$$

$$l_1^1 = ReLU(p_1^1)$$



$$p_1^2 = w_{1,1}^2 \cdot l_1^1 + w_{1,2}^2 \cdot l_2^1 + b_1^2$$

$$l_1^2 = ReLU(p_1^2)$$

$$p_2^1 = w_{2,1}^1 \cdot l_1^0 + w_{2,2}^1 \cdot l_2^0 + b_2^1$$

$$l_2^1 = ReLU(p_2^1)$$

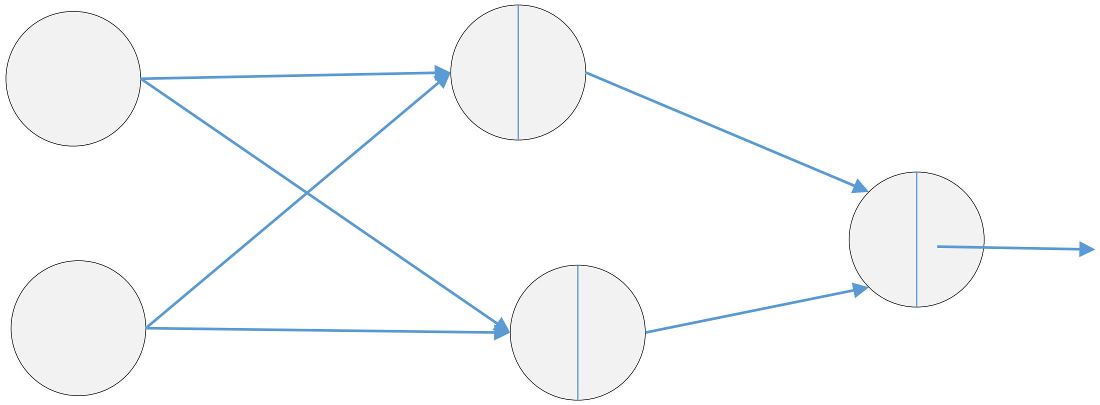
Layer 0: Input Layer

Layer 1

Layer 2

Feed Forward Neural Network (FF NN)

Neurons are connected to all neurons in the **next layer**



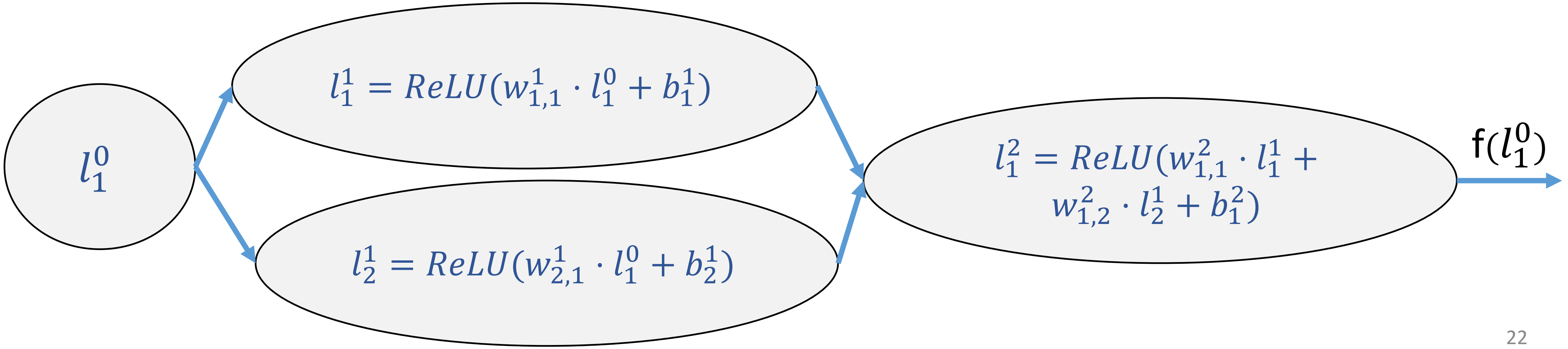
In theory:

Any continuous function over the real numbers can be approximated with an FF NN “ as much as we want”

Computing Outputs of Deep Models

To compute output, feed forward the input through the network

$$f(l_1^0) = ReLU\left(w_{1,1}^2 \cdot l_1^1 + w_{1,2}^2 \cdot l_2^1 + b_1^2\right) =$$
$$ReLU\left(w_{1,1}^2 \cdot ReLU(w_{1,1}^1 \cdot l_1^0 + b_{1,1}\right) + w_{1,2}^2 \cdot ReLU(w_{2,1}^1 \cdot l_1^0 + b_2^1) + b_1^2$$



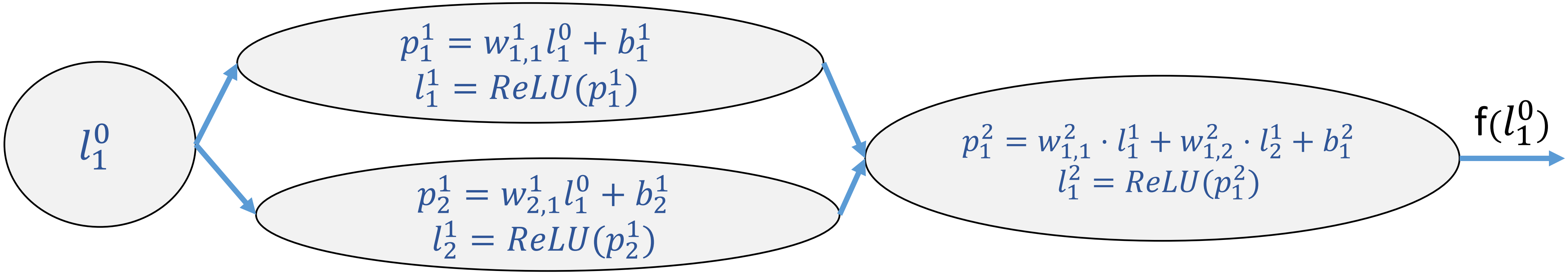
Gradient Descent in Deep Models

Deep model can be tuned by gradient descent

Need to compute the partial derivatives of all weights and biases

Hard to compute

Gradients in Deep Models: Difficulty

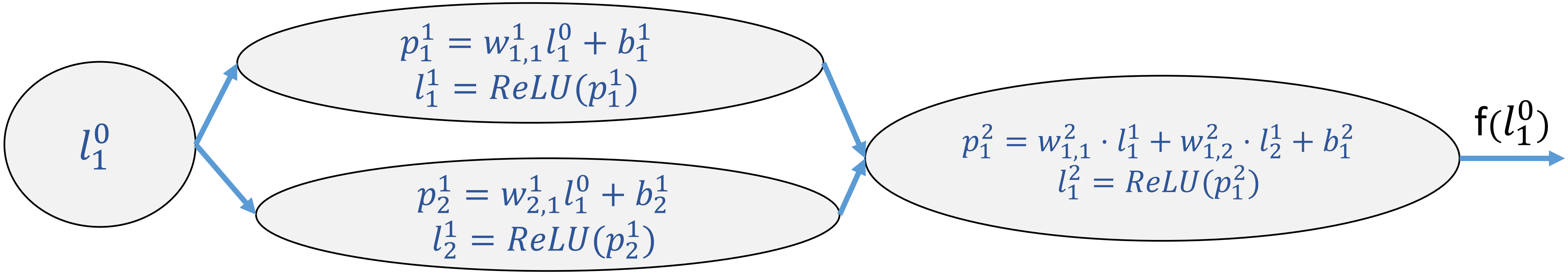


$$MSE(l_1^0) = \Sigma_{l_1^0 \in D_{Tr}} \left(f^*(l_1^0) - l_1^2\right)^2$$

$$\frac{\partial MSE}{\partial w_{1,1}^2} = \frac{\partial MSE}{\partial l_1^2} \cdot \frac{\partial l_1^2}{\partial p_1^2} \cdot \frac{\partial p_1^2}{\partial w_{1,1}^2}$$

$$\frac{\partial MSE}{\partial w_{1,1}^1} = \frac{\partial MSE}{\partial l_1^2} \cdot \frac{\partial l_1^2}{\partial p_1^2} \cdot \frac{\partial p_1^2}{\partial l_1^1} \cdot \frac{\partial l_1^1}{\partial p_1^1} \cdot \frac{\partial p_1^1}{\partial w_{1,1}^1}$$

Gradients in Deep Models: Difficulty



$$MSE(l_1^0) = \Sigma_{l_1^0 \in D_{Tr}} \left(f^*(l_1^0) - l_1^2\right)^2$$

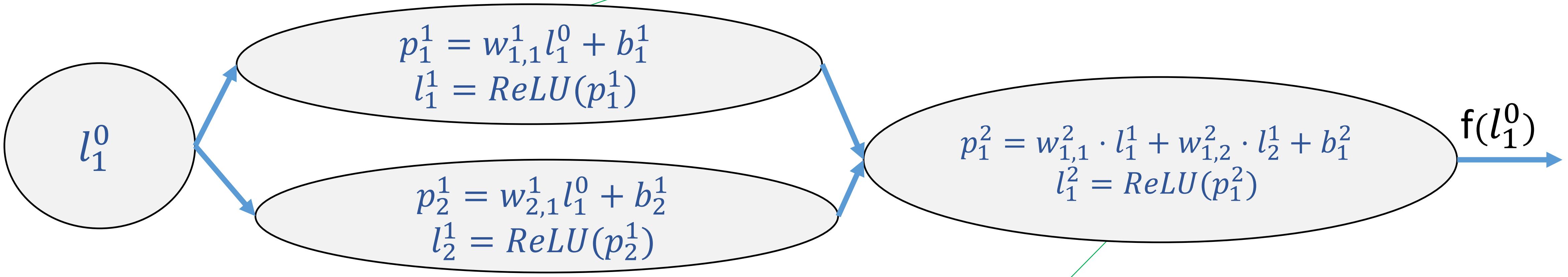
$$\frac{\partial MSE}{\partial w_{1,1}^2} = \frac{\partial MSE}{\partial l_1^2} \cdot \frac{\partial l_1^2}{\partial p_1^2} \cdot \frac{\partial p_1^2}{\partial w_{1,1}^2}$$

$$\frac{\partial MSE}{\partial w_{1,1}^1} = \frac{\partial MSE}{\partial l_1^2} \cdot \frac{\partial l_1^2}{\partial p_1^2} \cdot \frac{\partial p_1^2}{\partial l_1^1} \cdot \frac{\partial l_1^1}{\partial p_1^1} \cdot \frac{\partial p_1^1}{\partial w_{1,1}^1}$$

Backpropagation of the Loss Function

(Back)propagates the common parts of the derivatives

$$\frac{\partial MSE}{\partial l_1^2} \cdot \frac{\partial l_1^2}{\partial p_1^2} \cdot \frac{\partial p_1^2}{\partial l_1^1} \cdot \frac{\partial l_1^1}{\partial p_1^1} \cdot \frac{\partial p_1^1}{\partial w_{1,1}^1}$$



$$\frac{\partial MSE}{\partial w_{1,1}^2} = \frac{\partial MSE}{\partial l_1^2} \cdot \frac{\partial l_1^2}{\partial p_1^2} \cdot \frac{\partial p_1^2}{\partial w_{1,1}^2}$$

Overall

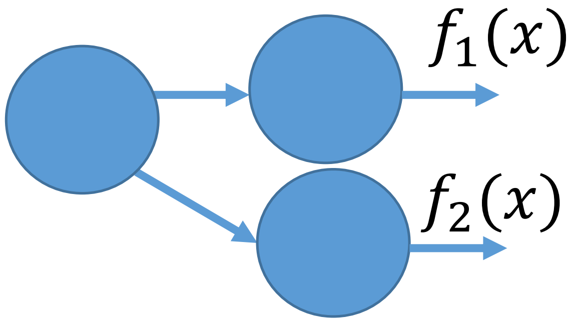
Defined learning models and deep models

Learned to train models through loss function and gradient descent

Learned to use backpropagation to train deep networks

Next Time: robustness of deep learning

Multiclass Classification

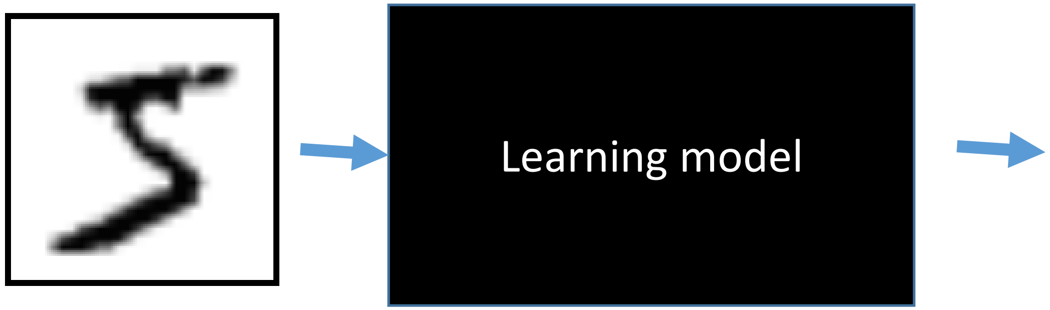


Output layer has a neuron for each class

Outputs normalized to a probability distribution with softmax:

$$P(c_i) = \frac{e^{f_i(x)}}{\sum_j e^{f_j(x)}}$$

For example, a model classifying to digits: $f: [0,1]^{784} \rightarrow [0,1]^{10}$



0: 0	5:0.7
1: 0	6:0
2: 0.1	7:0
3: 0.2	8:0
4: 0	9:0

Matrix Representation of FF NN

A perceptron computes: $l_i^{n+1} = \overline{w}_i^{n+1^T} \cdot \overline{l}^n + b_i^{n+1}$

A layer computes $\overline{l}^{n+1} = W^{n+1} \cdot \overline{l}^n + \overline{b}^{n+1}$ for:

$$W^{n+1} = \begin{pmatrix} w_{1,1}^{n+1} & w_{1,2}^{n+1} \\ w_{2,1}^{n+1} & w_{2,2}^{n+1} \end{pmatrix} \quad \overline{b}^{n+1} = \begin{pmatrix} b_1^{n+1} \\ b_2^{n+1} \end{pmatrix}$$