Data Mining & Machine Learning

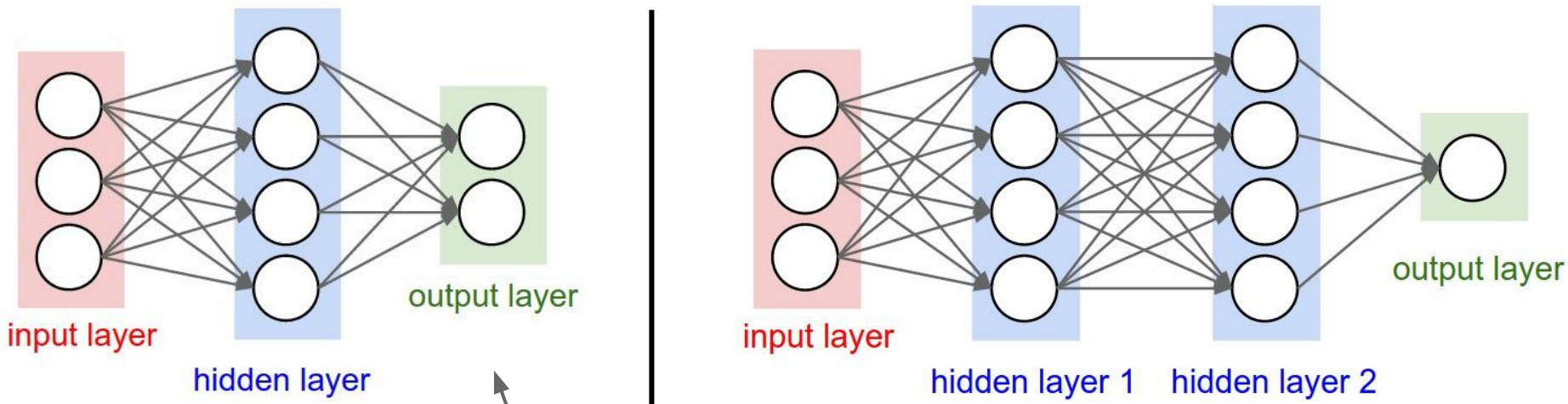
CS57300
Purdue University

March 1, 2018

Recap of Last Class (Model Search)

Forward and Backward passes

Feedforward Neural Networks



"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

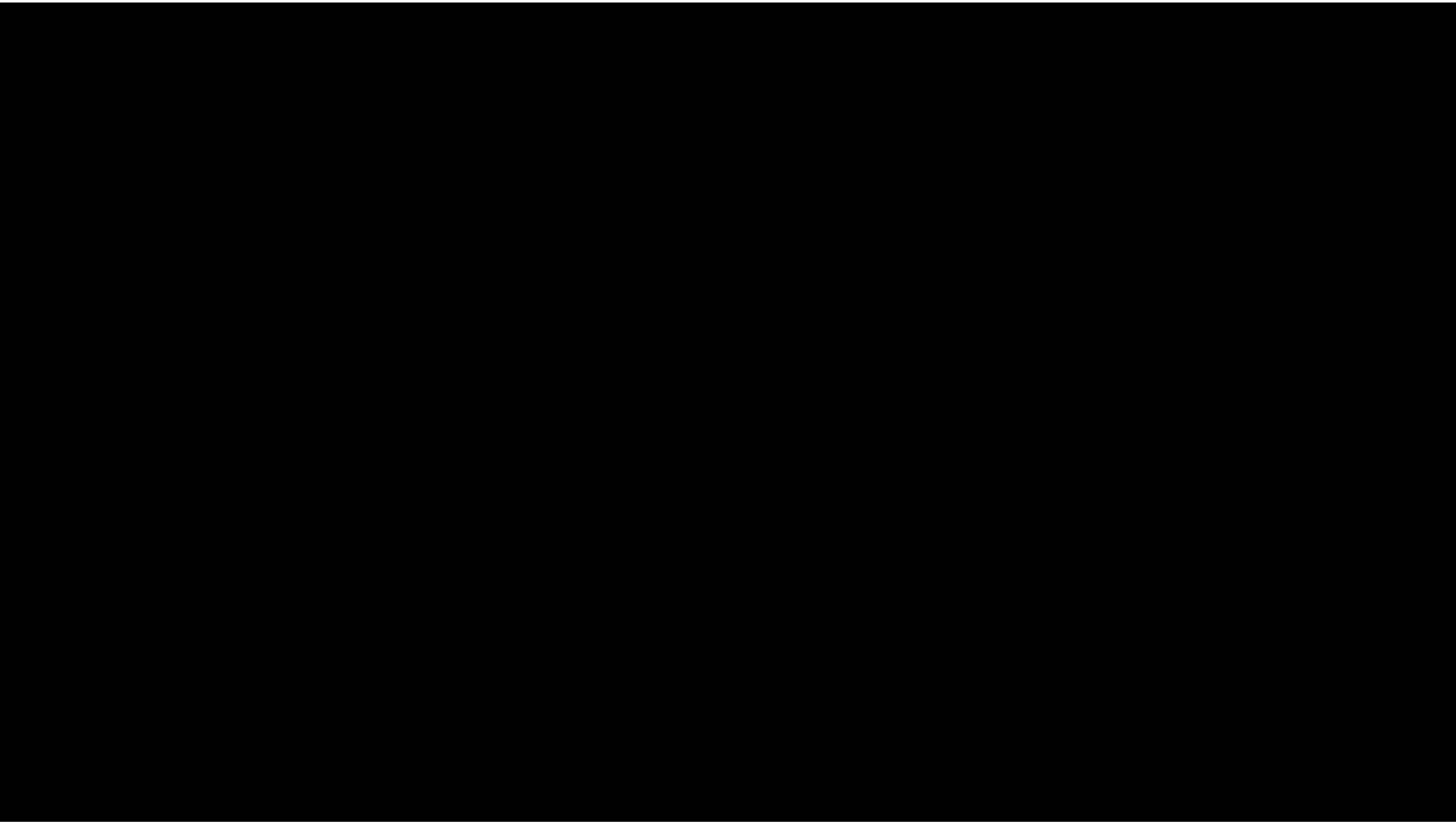
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Ack: Fei-Fei Li & Andrej Karpathy & Justin Johnson

- Layers do not need to be fully connected
- Size of layer (number of units) is another *hyper-parameter**

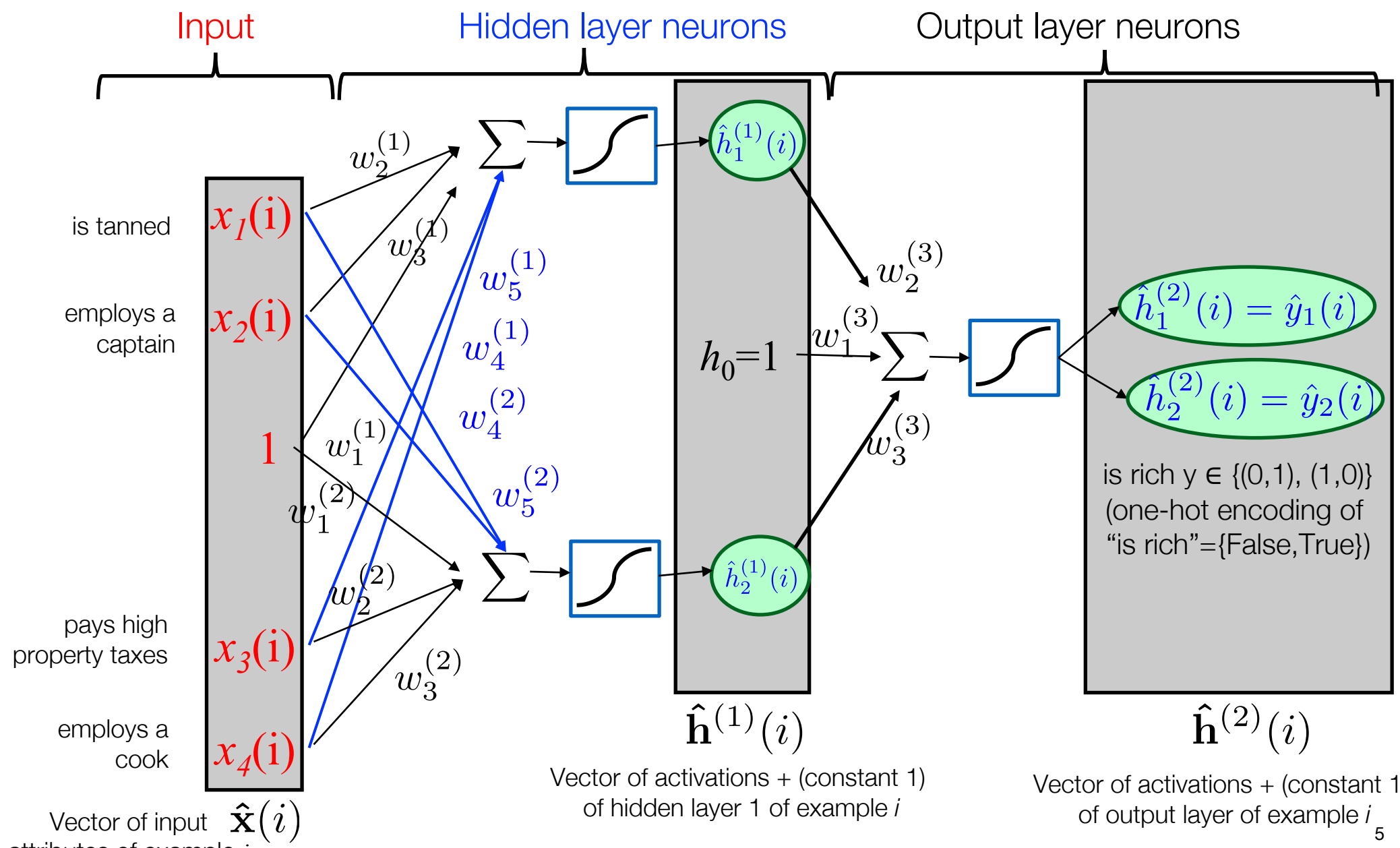
  ** hyper-parameter is a parameter that is fixed, not learned*

How Feedforward Networks Work



https://www.youtube.com/watch?v=aircAruvnKk

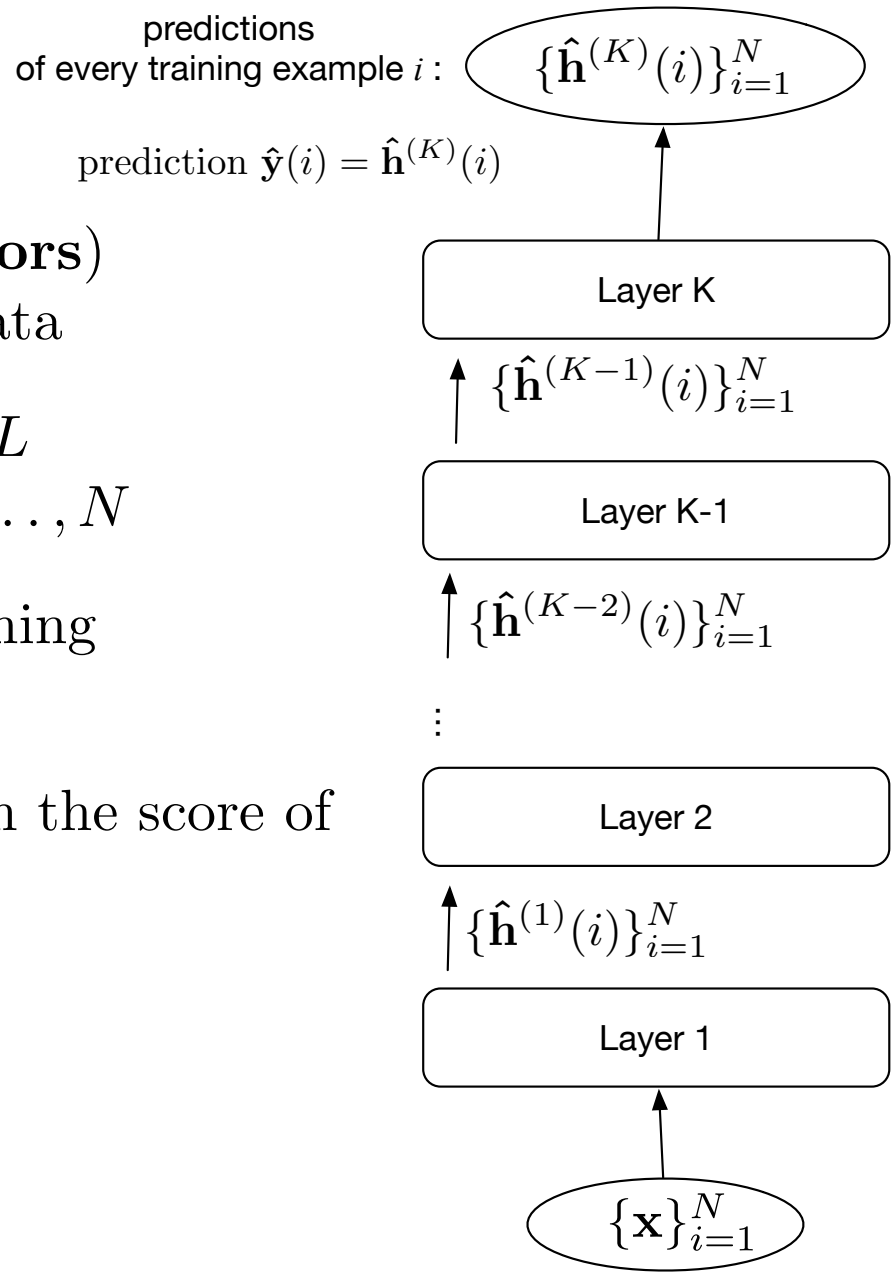
Feedforward Neural Network Example (is person rich?)



General Prediction Procedure (Forward Pass)

Variables:

- $\{\mathbf{x}\}_{i=1}^{N}$ are the inputs (attribute **vectors**) of example $i = 1, \ldots, N$ of training data
- $\hat{\mathbf{h}}^{(L)}(i)$ is the **vector** of hidden layer L neuron activations of example $i = 1, \ldots, N$
- The final (softmax) prediction of training example i is the **vector** $\hat{\mathbf{h}}^{(K)}(i)$
- $\mathbf{L} = \{L(\hat{\mathbf{h}}^{(K)}(i))\}_{i=1}^{N}$ is a **matrix** with the score of all output neurons of all training examples $i = 1, \ldots, N$
- Row $L(\hat{\mathbf{h}}^{(K)}(i))$ of $\mathbf{L}$ is the score of the i-th training example.



Forward + Backward Updates (following the training data)

All training examples!

loss function computes prediction error of every training example i :

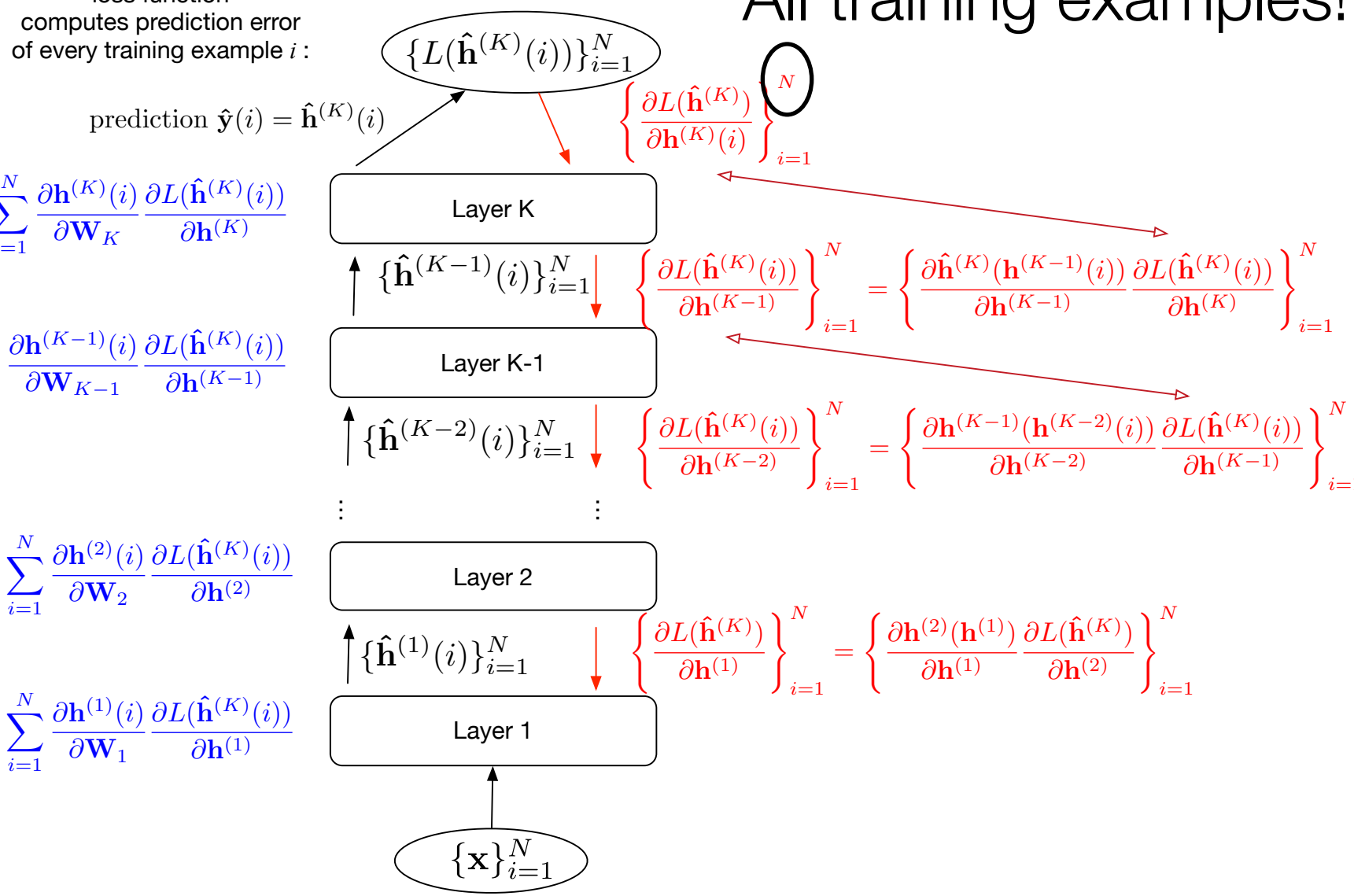
prediction $\hat{\mathbf{y}}(i) = \hat{\mathbf{h}}^{(K)}(i)$

$$\mathbf{W}_K = \mathbf{W}_K + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K)}(i)}{\partial \mathbf{W}_K} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}$$

$$\mathbf{W}_{K-1} = \mathbf{W}_{K-1} + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K-1)}(i)}{\partial \mathbf{W}_{K-1}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(2)}(i)}{\partial \mathbf{W}_2} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(2)}}$$

$$\mathbf{W}_1 = \mathbf{W}_1 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(1)}(i)}{\partial \mathbf{W}_1} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(1)}}$$



Stochastic Gradient Descent

- Rather than using all training examples in the gradient descent, we will use just a subset of the data at each time
  - At every gradient descent step we will just use a subset of the examples
- At every gradient update we randomly choose another set of *n* training examples
  - In practice, we do sampling **without** replacement; $\{\mathbf{x}\}_{i=1}^{n}$ where $n < N$.
    If training data is exhausted, restart sampling
- The "new" training data $\{\mathbf{x}\}_{i=1}^{n}$ is known as a mini-batch
  - The of training via gradient descent with mini-batches is called *mini-batch stochastic gradient ascent*
    (or mini-batch stochastic gradient descent if we are trying to minimize the score)

Rationalizing Mini-Batch Sizes

- Gradient computation: gradient is averaged over all training examples

$$\mathbf{W}_K = \mathbf{W}_K + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K)}(i)}{\partial \mathbf{W}_K} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}$$

$$\mathbf{W}_{K-1} = \mathbf{W}_{K-1} + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(K-1)}(i)}{\partial \mathbf{W}_{K-1}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}$$

$$\mathbf{W}_2 = \mathbf{W}_2 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(2)}(i)}{\partial \mathbf{W}_2} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(2)}}$$

$$\mathbf{W}_1 = \mathbf{W}_1 + \epsilon \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathbf{h}^{(1)}(i)}{\partial \mathbf{W}_1} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(1)}}$$

- **Larger mini-batches** produce more accurate gradients
  - Would larger batches be better for training?
  - “Optimization benefits from more data because we can compute better gradients”?
- **Smaller learning rates ε** provide more accurate gradient descent approximation, better models?

Model Search for Deep Neural Network

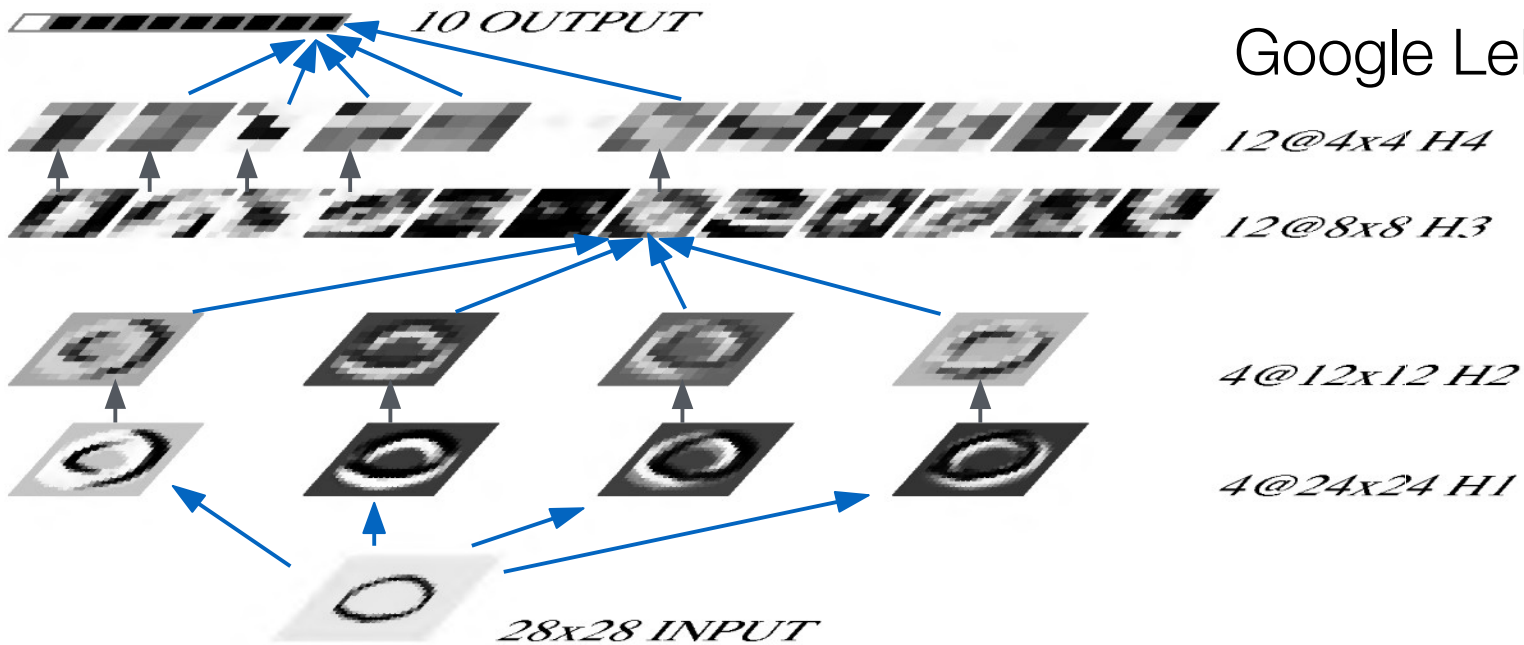
Q: Is it better to search for the best model (highest likelihood score) using all the training data?

A: Depends (Zhang et al. 2017)

- Deep neural network scores are nonconvex, many local maxima
- Pros of using all training data: Searching using all the training data, we will surely find a model that better fits the training data
- Cons: Using all training examples often works ***terribly*** in practice
  - Model found by gradient descent performs **poorly** even on the **training data** itself (due to local minima)
  - Small mini-batches are often better than larger batches…
  - …but not too small…
  - …and depends on the learning rate (infinitesimal gradient ε steps)

(Zhang et al. 2017) Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O., 2016. Understanding deep learning requires rethinking generalization. ICLR 2017

Weird Learning Characteristics of Deep Neural Networks



Google LeNet

In this example:

- Increasing mini-batch sizes reduces model accuracy on the **test data** (model generalizes less)
- But increasing learning rate improves things (i.e., making a worse approximation of gradient ascent, improves things?!?)
- We do not yet know why… many different hypotheses (as of 2018)