Data Mining & Machine Learning

CS37300
Purdue University

October 30, 2017

Kaggle competition: added extra credits

Welcome to the **extra-credit competition** of CS37300. Your goal is to predict whether or not the lender will or will not payoff their loan.

Instructions and Policy: Each student should write up their own solutions independently. You need to indicate the names of the people you discussed a problem with; ideally you should discuss with no more than four other people.

Winning entries will be asked to submit their approach in a PDF via Blackboard. **Winners are required to submit their Python code [any excessive copying from online resources will make the entry ineligible for extra credits].** Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.

To participate you **MUST** choose a screen name from the following list:
https://www.cs.purdue.edu/homes/ribeirob/courses/Fall2017/data/star_wars_characters.txt

Extra credit assignment is as follows:

- **ALL PARTICIPANTS WITH PRIVATE LEADERBOARD SCORES > 0.6 WILL GET 3% EXTRA CREDIT**
- +5% extra credit to the top 1%
- +4% extra credit to the 2%-10%
- +3% extra credit to the 11%-20%
- +2% extra credit to the 21%-30%
- +1% extra credit to the 41%-50%
- 0% extra to bottom <50%

Kaggle competition update

Public Leaderboard | **Private Leaderboard**

This leaderboard is calculated with approximately 30% of the test data.
The final results will be based on the other 70%, so the final standings may be different.

Raw Data | Refresh

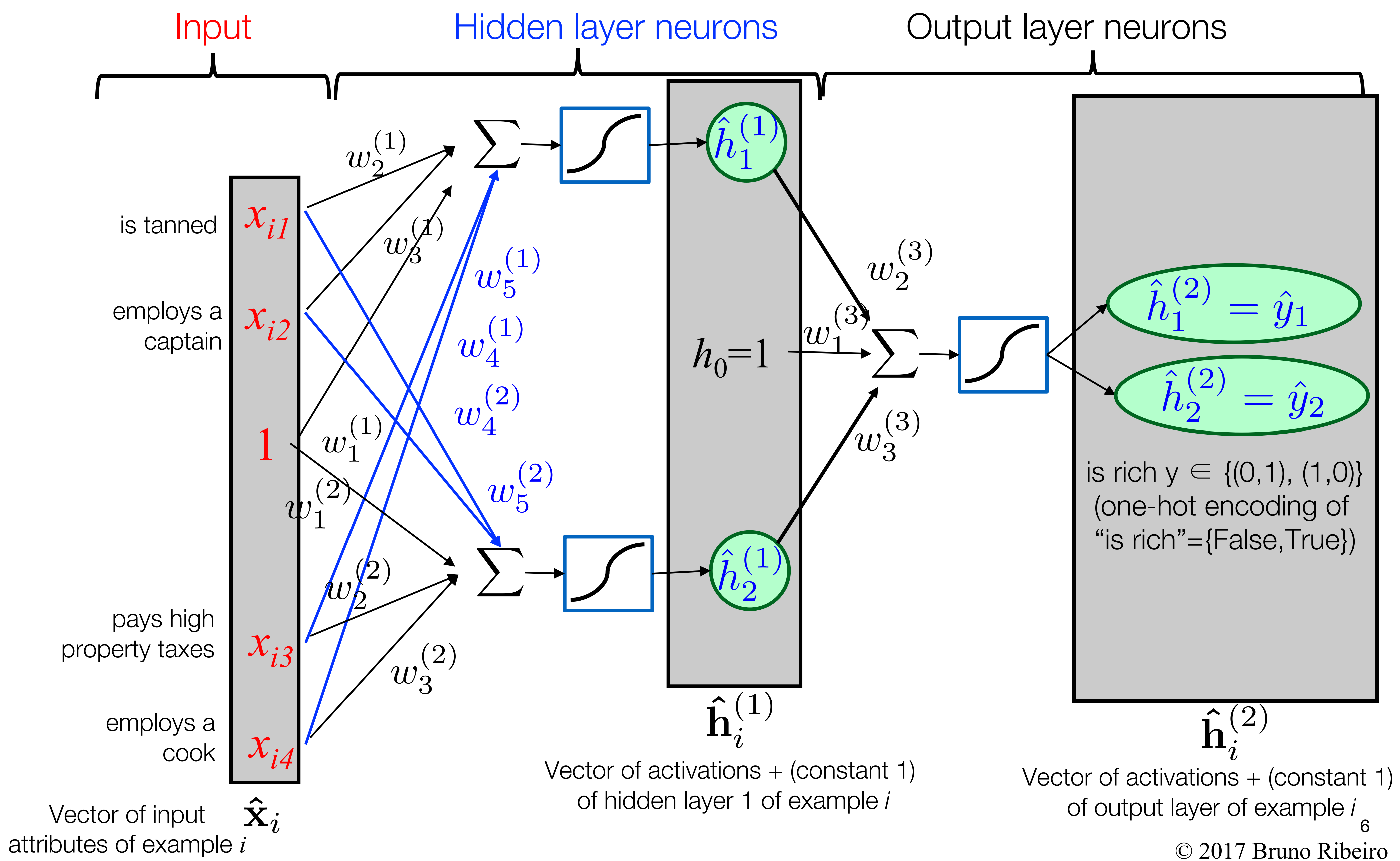
#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1	▲3	**Revan**			0.83407	12	1d
2	▼1	**Luke Skywalker**			0.83286	8	16d
3	▼1	**Cad Bane**			0.81991	18	6d
4	▼1	**Yoda**			0.80979	13	2d
5	—	**Ki-Adi-Mundi**			0.76811	2	7d
6	—	**Kyp Durron**			0.73613	4	20d
7	—	**Shaak Ti**			0.70133	2	19d
8	—	**Bossk**			0.67826	2	24d
9	—	**Admiral Thrawn**			0.65520	2	7d
10	—	**General Grievous**			0.58599	3	19d
11	—	**Boba Fett**			0.52407	12	7d
12	—	**Count Dooku**			0.51071	1	16d
13	—	**Darth Maul**			0.49129	2	24d
		Bank_Sample_Submission.csv			0.49008		
14	—	**Zuckuss**			0.49008	1	9d

Model Search (Practical Deep Learning)

Outline

- Review: Feedfoward networks
    - Input examples, their hidden values, and output
- Review: Backpropagation with Forward and Backward Passes
    - Emphasis on what happens to each training example
- Stochastic Gradient Descent

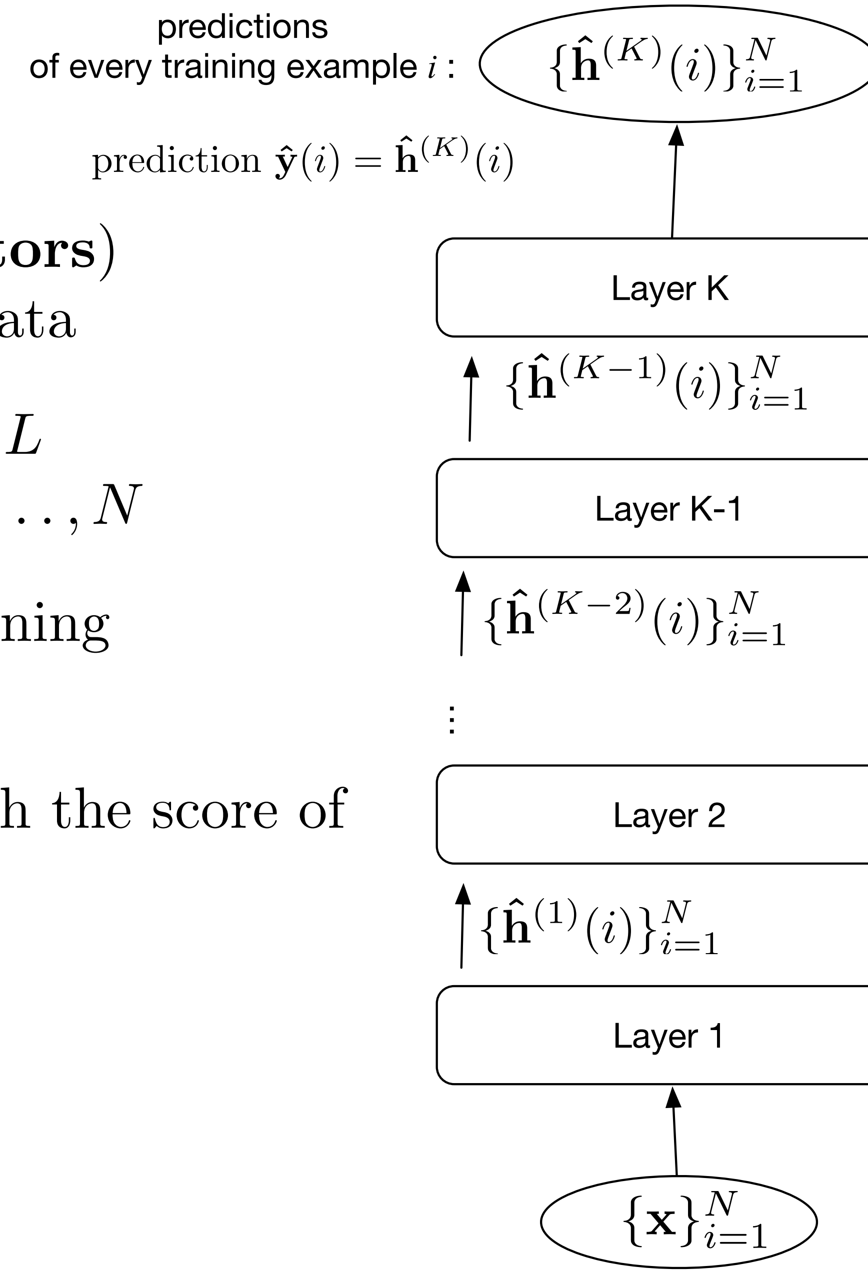
Feedforward Neural Network Example (is person rich?)



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 :

$$\{L(\hat{\mathbf{h}}^{(K)}(i))\}_{i=1}^{N}$$

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

$$\left\{\frac{\partial L(\hat{\mathbf{h}}^{(K)})}{\partial \mathbf{h}^{(K)}(i)}\right\}_{i=1}^{N}$$

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

Layer K

$$\{\hat{\mathbf{h}}^{(K-1)}(i)\}_{i=1}^{N}$$

$$\left\{\frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}\right\}_{i=1}^{N} = \left\{\frac{\partial \hat{\mathbf{h}}^{(K)}(\mathbf{h}^{(K-1)}(i))}{\partial \mathbf{h}^{(K-1)}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K)}}\right\}_{i=1}^{N}$$

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

Layer K-1

$$\{\hat{\mathbf{h}}^{(K-2)}(i)\}_{i=1}^{N}$$

$$\left\{\frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-2)}}\right\}_{i=1}^{N} = \left\{\frac{\partial \mathbf{h}^{(K-1)}(\mathbf{h}^{(K-2)}(i))}{\partial \mathbf{h}^{(K-2)}} \frac{\partial L(\hat{\mathbf{h}}^{(K)}(i))}{\partial \mathbf{h}^{(K-1)}}\right\}_{i=1}^{N}$$

$\vdots$ $\vdots$

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

Layer 2

$$\{\hat{\mathbf{h}}^{(1)}(i)\}_{i=1}^{N}$$

$$\left\{\frac{\partial L(\hat{\mathbf{h}}^{(K)})}{\partial \mathbf{h}^{(1)}}\right\}_{i=1}^{N} = \left\{\frac{\partial \mathbf{h}^{(2)}(\mathbf{h}^{(1)})}{\partial \mathbf{h}^{(1)}} \frac{\partial L(\hat{\mathbf{h}}^{(K)})}{\partial \mathbf{h}^{(2)}}\right\}_{i=1}^{N}$$

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

Layer 1

$$\{\mathbf{x}\}_{i=1}^{N}$$

Approximate Model Search (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 $\{\mathbf{x}\}_{i=1}^{n}$ where $n < N$.
- At every gradient update we randomly choose another set of *n* training examples
  - In practice, we do sampling **without** replacement;
    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)

Model Search for Deep Neural Network

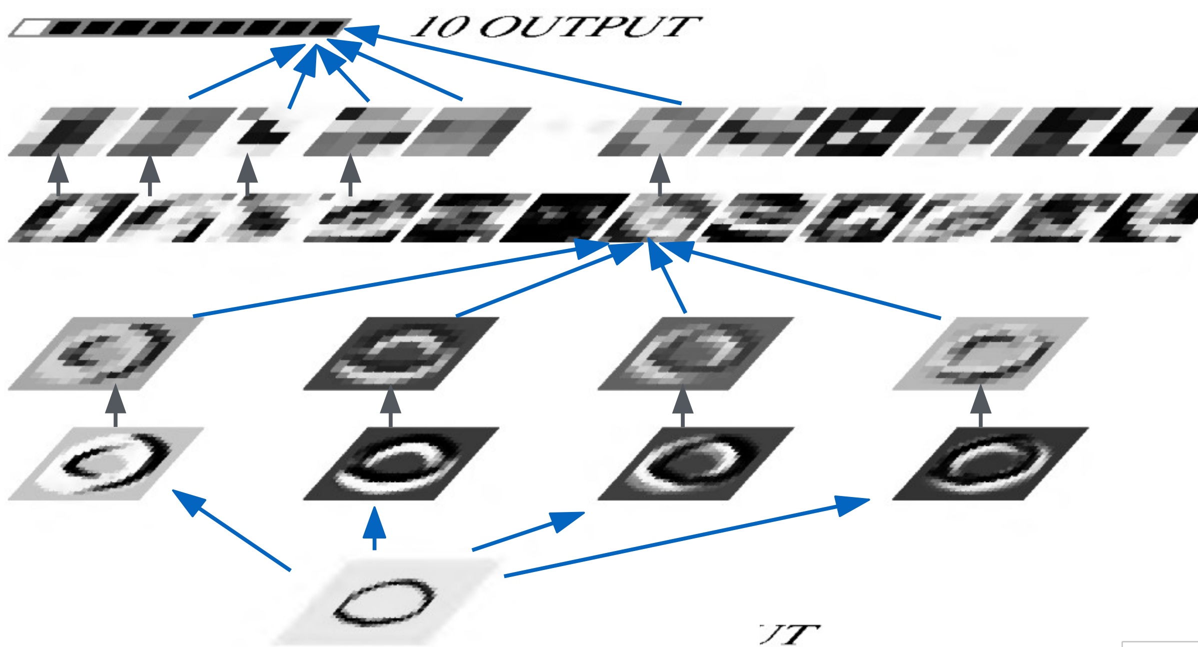
Q: Is it better to search for the best model (highest likelihood score) using all the training data?

A: Depends (Zhang et al. 2017)

- Deep neural network scores are nonconvex, many local minima
- 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 infinitesimal gradient ε increments (learning rate)

(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



Model searching in digit classification task using convolutional neural networks

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… but we have some hypotheses