Mathematics of Image and Data Analysis
Math 5467

Lecture 23: Classification with Neural Networks

Instructor: Jeff Calder
Email: jcalder@umn.edu

http://www-users.math.umn.edu/~jwcalder/5467S21

Announcements

- HW4 is online, due April 30.
- Project 3 online, due May 9.
- Please fill out *Student Rating of Teaching (SRT)* online as soon as possible, and before **May 3**.
  - You should have received an email from Office of Measurement Services with a link.
  - You can also find a link on our Canvas website.

Last time

- Intro to Neural Networks

Today

- Classification with neural networks.

Neural networks

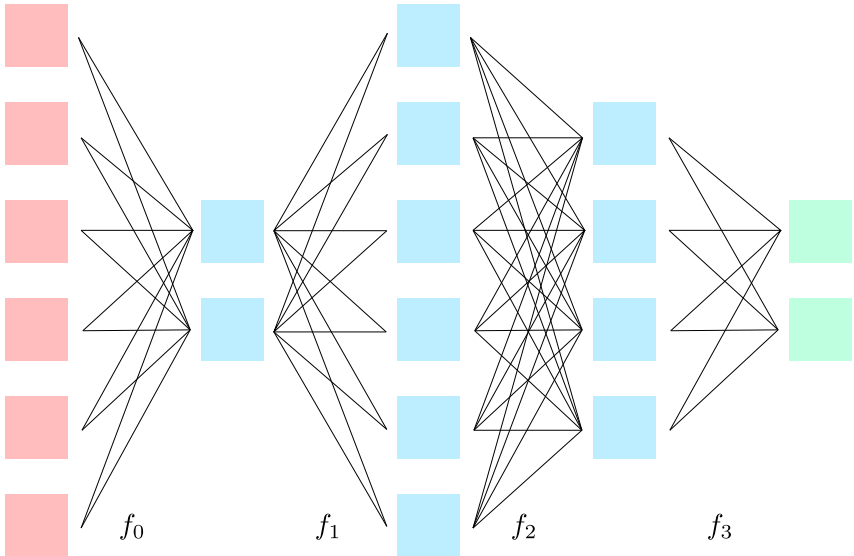


Figure 1: An example of a fully connected neural network with three hidden layers. The blue nodes are the hidden layers, the red is the input, and the green is the output. The hidden layers have width $n_1 = 2$, $n_2 = 6$, and $n_3 = 4$ and the number of input variables is $n_0 = 6$.

Neural network

In more compact notation, we can write a fully connected neural network with L layers recursively as

$$f_k = \sigma_k(W_k f_{k-1} + b_k), \quad k = 1, \ldots, L, \tag{1}$$

where

- $f_0 \in \mathbb{R}^{n_0}$ is the input to the network,
- $f_k \in \mathbb{R}^{n_k}$ for $k = 1, \ldots, L-1$ are the values of the network at the hidden layers,
- f_L is the output of the neural network,
- n_k is the number of hidden nodes in the k^{th} layer,
- The weights $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$ and biases $b_k \in \mathbb{R}^{n_k}$ are the learnable parameters in the neural network.

Loss

The output of the neural network $f_L \in \mathbb{R}^{n_L}$ is typically fed into a loss function

$$\mathcal{L} : \mathbb{R}^{n_L} \to \mathbb{R},$$

which measures the performance of the network for the given learning task.

Typically the loss has the form

$$\mathcal{L}(W_1, b_1, \ldots, W_L, b_L) = \sum_{i=1}^{m} \ell(f_L(x_i), y_i), \tag{2}$$

where (x_i, y_i) for $i = 1, \ldots, m$ are the training data. Here, we write $f_L(x)$ to denote the value of the output of the network f_L given the input is $f_0 = x$.

Neural networks are trained by minimizing the loss function $\mathcal{L}$ with gradient descent.

Back Propagation

For notational simplicity, we will write

$$z_k = W_k f_{k-1} + b_k, \tag{3}$$

so that $f_k = \sigma_k(z_k)$. Let $\frac{\partial \mathcal{L}}{\partial z_k} \in \mathbb{R}^{n_k}$ denote the gradient of $\mathcal{L}$ with respect to z_k. We also let D_k be the diagonal $n_k \times n_k$ matrix with diagonal entries given by the vector $\sigma_k'(z_k)$. That is

$$D_k = \operatorname{diag}(\sigma_k'(z_k)).$$

Theorem 1 (Back propagation)**.** *For $k = 2, \ldots, L$ we have*

$$\frac{\partial \mathcal{L}}{\partial z_{k-1}} = D_{k-1} W_k^T \frac{\partial \mathcal{L}}{\partial z_k}, \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial z_k} f_{k-1}^T, \quad \textit{and} \quad \frac{\partial \mathcal{L}}{\partial b_k} = \frac{\partial \mathcal{L}}{\partial z_k}. \tag{5}$$

ResNet

The traditional neural network architecture

$$f_k = \sigma_k(W_k f_{k-1} + b_k), \quad k = 1, \ldots, L,$$

often yields worse performance for deeper networks compared to shallower ones.

$$\frac{\partial L}{\partial z_1} = D_1 W_2^T D_2 W_2^T D_3 W_3^T \cdots D_L W_L^T \frac{\partial L}{\partial z_L}$$

Vanishing Grad.
Gradient Blowup.

The *Residual Neural Network (ResNet)* architecture is a recent development in deep learning that solves this problem by changing the architecture to

$$f_k = f_{k-1} + W_{k,1}\sigma_k(W_{k,2} f_{k-1} + b_k), \quad k = 1, \ldots, L. \tag{6}$$

The idea is to have each layer learn the *residual* $f_k - f_{k-1}$, which allows the network to easily skip layers, by setting $f_k = f_{k-1}$. Thus, a deeper network with ResNet architecture should not perform worse than a shallower network.

Classification with neural networks

Recall that our label vectors are given as one hot vectors $e_1, \ldots, e_k$ in $\mathbb{R}^k$ (i.e., the standard basis vectors), where e_i represents the i^{th} class.

For a k-class classification problem, the output of the neural network $f_L(x)$ has k components, so $f_L(x) \in \mathbb{R}^k$, and the classification of x is taken to be the largest component of $f_L(x)$.

Let $z_1, \ldots, z_m \in \mathbb{R}^k$ denote the output vectors of the neural network aplied to m training points $x_1, x_2, \ldots, x_m$, so

$$z_i = f_L(x_i).$$

These outputs are fed through the *soft-max* function to conver them into probability vectors $p_1, \ldots, p_m \in \mathbb{R}^k$ given by

$$p_i(j) := \frac{e^{z_i(j)}}{\sum_{q=1}^{k} e^{z_i(q)}}.$$

Loss

The loss used for classification is normally the negative log likelihood loss. Letting $y_1, \ldots, y_m \in \mathbb{R}^k$ denote the one hot vectors representing the classes of the training data, the negative log likelihood loss is

$$\mathcal{L}(f_L) = -\sum_{i=1}^{n} y_i^T \log(p_i). \tag{7}$$

m = size of training dataset.

We claim that

$$\mathcal{L}(f_L) = -\sum_{i=1}^{n} f_L(x_i)^T y_i + \sum_{i=1}^{n} \log\left(\sum_{j=1}^{k} e^{f_L(x_i)^T e_j}\right).$$

Proof:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i(j)}}$$

z_i ← vector

$z_i(j)$ ← scalar

$\|\cdot\|$ = over vector

$$\log(p_i) = \log(e^{z_i}) - \log\left(\sum_{j=1}^{k} e^{z_i(j)}\right)\|$$

$$= f_L(x_i) - \log\left(\sum_{j=1}^{k} e^{f_L(x_i)^T e_j}\right)\mathbb{1}$$

$$L(f_L) = -\sum_{i=1}^{m} y_i^T \log(p_i)$$

$$= -\sum_{i=1}^{m} y_i^T f_L(x_i) + \sum_{i=1}^{m} \overbrace{(y_i^T \mathbb{1})}^{=1} \log\left(\sum_{j=1}^{k} e^{f_L(x_i)^T e_j}\right)$$

Toy data

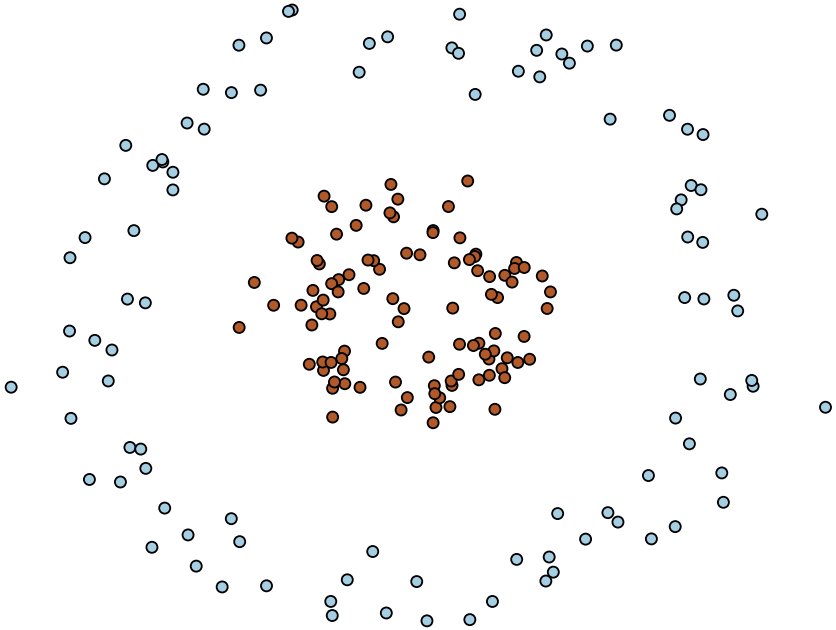
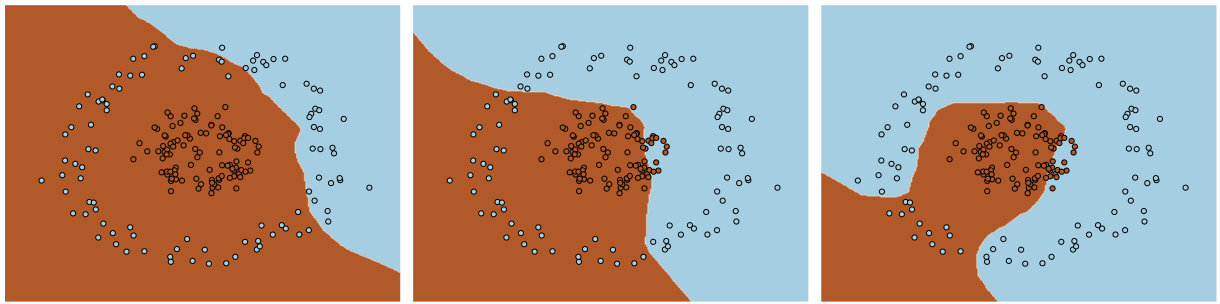


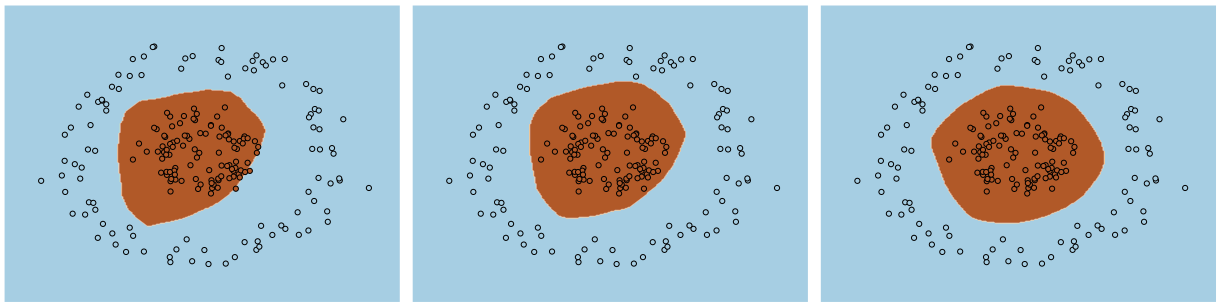
Figure 2: Synthetic data on rings consisting of two classes.

Training

2-Layer 100-hidden units



(a) Iteration 50 (b) Iteration 150 (c) Iteration 250



(d) Iteration 350 (e) Iteration 500 (f) Iteration 3000

MNIST

We consider classification of MNIST digits with 60000 training images and 10000 testing images.

- 2-Layer network with 10 hidden nodes: 93% testing accuracy
- 2-Layer network with 32 hidden nodes: 97% testing accuracy
- Even with good testing accuracy, there is overfitting:
  - 1-pixel shift of testing data: 87% accuracy
  - 2-pixel shift of testing data: 62% accuracy

MNIST

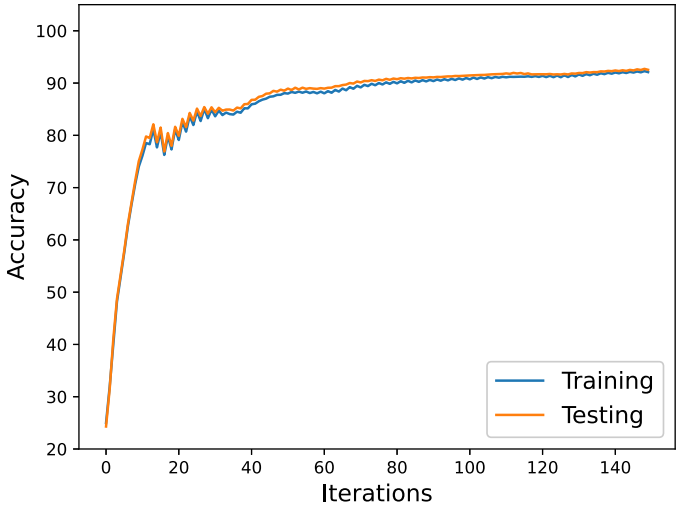
We consider classification of MNIST digits with 60000 training images and 10000 testing images.

- 2-Layer network with 10 hidden nodes: 93% testing accuracy
- 2-Layer network with 32 hidden nodes: 97% testing accuracy
- Even with good testing accuracy, there is overfitting:
  - 1-pixel shift of testing data: 87% accuracy
  - 2-pixel shift of testing data: 62% accuracy

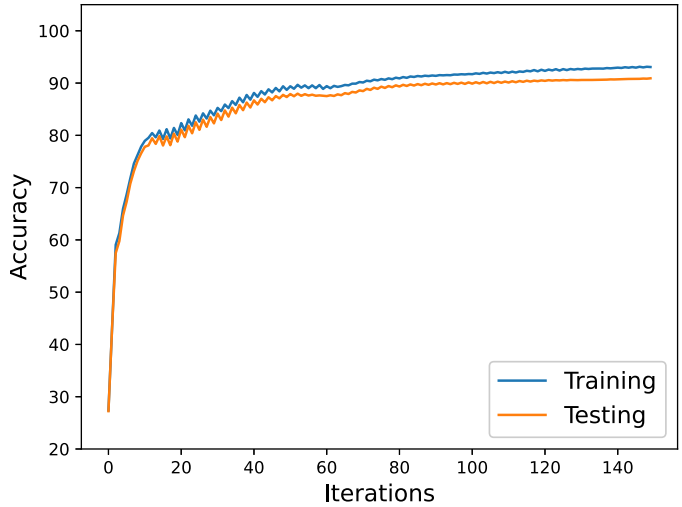
Figure 3: The 10 hidden nodes for MNIST classification.

Overfitting with less tranining data

32 hidden nodes.



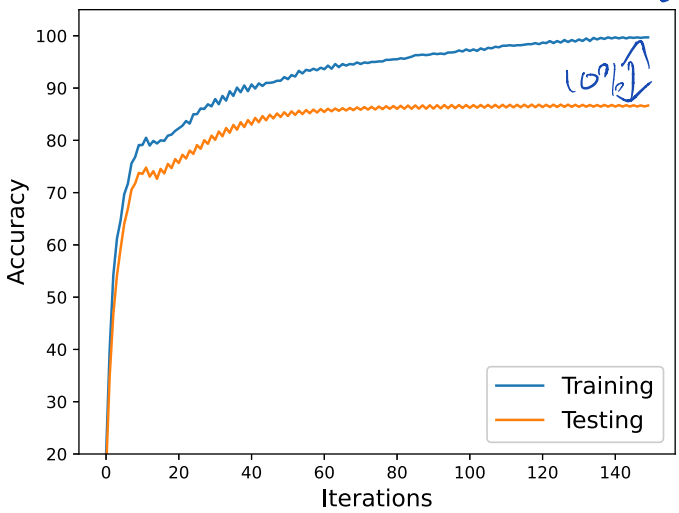
(a) 60000 Training images



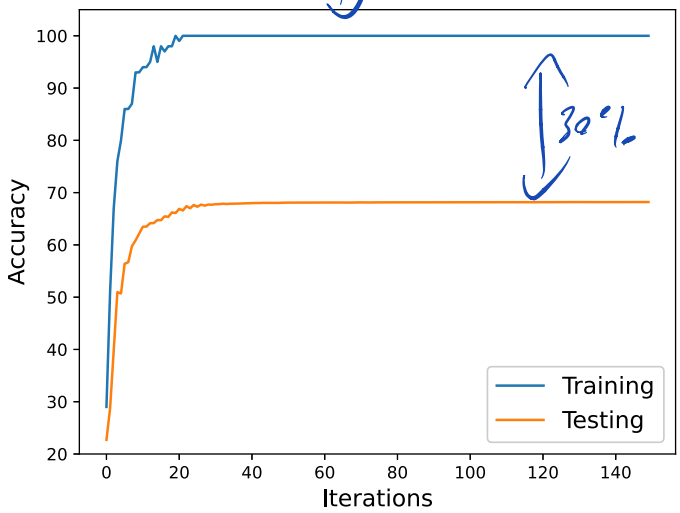
(b) 10000 Training images

Data Augmentation

Overfitting with less tranining data



(c) 1000 Training images



(d) 100 Training images

Classification with Torch (.ipynb)