MATH 829: Introduction to Data Mining and Analysis
Neural networks II

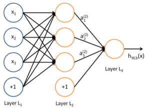
Dominique Guillot

Departments of Mathematical Sciences
University of Delaware

April 13, 2016

This lecture is based on the UFLDL tutorial (http://deeplearning.stanford.edu/)

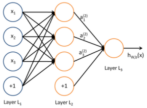
Recall



We have:

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b} = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}).$$

Recall (cont.)



Vector form:

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b} = a^{(3)} = f(z^{(3)}).$$

Training neural networks

Suppose we have

- A neural network with s_l neurons in layer l $(l = 1, \dots, n_l)$.
- Observations $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)}) \in \mathbb{R}^{s_1} \times \mathbb{R}^{s_{n_l}}$.

We would like to choose $W^{(l)}$ and $b^{(l)}$ in some optimal way for all l.

Let

$$J(W, b; x, y) := \frac{1}{2}\|h_{W,b}(x) - y\|_2^2 \qquad \text{(Squared error for one sample).}$$

Define

$$J(W, b) := \frac{1}{m}\sum_{i=1}^m J(W, b; x^{(i)}, y^{(i)}) + \frac{\lambda}{2}\sum_{l=1}^{n_l - 1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ji}^{(l)})^2.$$

(average squared error with Ridge penalty).

Note:

- The Ridge penalty prevents overfitting.
- We do not penalize the bias terms $b_i^{(l)}$.
- The loss function $J(W, b)$ is not convex.

Some remarks

- The loss function $J(W, b)$ is often used both for regression and classification.
- In classification problems, we choose the labels $y \in \{0, 1\}$ (if working with sigmoid) or $y \in \{-1, 1\}$ (if working with $\tanh$).
- For regression problems, we scale the output so that $y \in [0, 1]$ (if working with sigmoid) or $y \in [-1, 1]$ (if working with $\tanh$).
- We will use a gradient descent to minimize $J(W, b)$. Note that since the function is non-convex, we may only find a *local* minimum.
- We need an initial choice for $W_{ij}^{(l)}$ and $b_i^{(l)}$. If we initialize all the parameters to 0, then the parameters remain constant over the layers because of the symmetry of the problem.
- As a result, we initialize the parameters to a small constant at random (say, using $N(0, \epsilon^2)$ for $\epsilon = 0.01$).

Gradient descent and the backpropagation algorithm

We update the parameters using a gradient descent as follows:

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} \leftarrow b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b).$$

Here $\alpha > 0$ is a parameter (the *learning rate*).

Observe that:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(i)}, y^{(i)}) + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(i)}, y^{(i)}).$$

Therefore, it suffices to compute the derivatives of $J(W, b; x^{(i)}, y^{(i)})$.

Computing the derivatives using backpropagation

1. Compute the activations for all the layers.
2. For each output unit i in layer n_l (output), compute

$$\delta_i^{(n_l)} := \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|_2^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{n_l}).$$

3. For $l = n_l - 1, n_l - 2, \ldots, 2$
   For each node i in layer l, set

$$\delta_i^{(l)} := \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) \cdot f'(z_i^{(l)}).$$

4. Compute the desired partial derivatives:

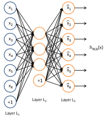
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x^{(i)}, y^{(i)}) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(i)}, y^{(i)}) = \delta_i^{(l+1)}.$$

Autoencoders

An **autoencoder** learns the identity function:

- Input: unlabeled data.
- Output = input.
- Idea: limit the number of hidden layers to discover structure in the data.
- Learn a *compressed* representation of the input.



Source: UFLDL tutorial.

Can also learn a *sparse* network by including supplementary constraints on the weights.

Example (UFLDL)

- Train an autoencoder on 10×10 images with one hidden layer.
- Each hidden unit i computes:

$$a_i^{(2)} = f\left(\sum_{j=1}^{100} W_{ij}^{(1)} x_j + b_j^{(1)}\right).$$

- Think of $a_i^{(2)}$ as some non-linear feature of the input x.

Problem: Find x that maximally activates $a_i^{(2)}$ over $\|x\|_2 \leq 1$.

Claim:

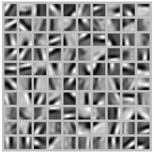
$$x_j = \frac{W_{ij}^{(1)}}{\sqrt{\sum_{j=1}^{100} (W_{ij}^{(1)})^2}}.$$

(Hint: Use Cauchy–Schwarz).

We can now display the image maximizing $a_i^{(2)}$ for each i.

Example (cont.)

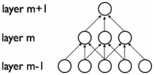
100 hidden units on 10x10 pixel inputs:



The different hidden units have learned to detect edges at different positions and orientations in the image.

Sparse neural networks

- So far we discussed *dense* neural networks.
- Dense networks have a lot of parameters to learn. Can be inefficient or impossible to train.
- *Sparse* models have been proposed in the literature.
- Some of these models find inspiration from how the early visual system is wired up in biology.

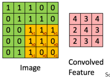


Using convolutions

- Idea: Certain signals are *stationary*, i.e., their statistical properties do not change in space or time.
- For example, images often have similar statistical properties in different regions in space.
- That suggests that the features that we learn at one part of an image can also be applied to other parts of the image.
- Can "convolve" the learned features with the larger image.

Example: 96×96 image.

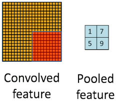
- Learn features on small 8×8 patches sampled randomly (e.g. using a sparse autoencoder).
- Run the trained model through all 8×8 patches of the image to get the feature activations.



Source: UFLDL tutorial.

Pooling features

- Once can also *pool* the features obtained via convolution.
- For example, to describe a large image, one natural approach is to aggregate statistics of these features at various locations.
- E.g. compute the mean, max, etc. over different regions.
- Can lead to more robust features. Can lead to invariant features.
- For example, if the pooling regions are contiguous, then the pooling units will be "translation invariant", i.e., they won't change much if objects in the image are undergo a (small) translation.



Neural networks with scikit-learn

Need to install the 0.18-dev version (`http://scikit-learn.org/stable/developers/contributing.html#retrieving-the-latest-code`).

- `sklearn.neural_network.MLPClassifier`
- `sklearn.neural_network.MLPRegressor`