Recall

We have:

\[ a^{(2)}_1 = f(W^{(1)}_{11} x_1 + W^{(1)}_{12} x_2 + W^{(1)}_{13} x_3 + b^{(1)}_1) \]
\[ a^{(2)}_2 = f(W^{(1)}_{21} x_1 + W^{(1)}_{22} x_2 + W^{(1)}_{23} x_3 + b^{(1)}_2) \]
\[ a^{(2)}_3 = f(W^{(1)}_{31} x_1 + W^{(1)}_{32} x_2 + W^{(1)}_{33} x_3 + b^{(1)}_3) \]

\[ h_{W,b} = a^{(3)}_1 = f(W^{(2)}_{11} a^{(2)}_1 + W^{(2)}_{12} a^{(2)}_2 + W^{(2)}_{13} a^{(2)}_3 + b^{(2)}). \]

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 \( l \) \((l = 1, \ldots, n_l)\).
- Observations \((x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)}) \in \mathbb{R}^{s_1} \times \mathbb{R}^{s_1}\).

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

Let

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

Define

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

(average squared error with Ridge penalty).

Note:

- The Ridge penalty prevents overfitting.
- We do not penalize the bias terms \( b^{(l)}_i \).
- 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 $J(W, b; x^{(i)}, y^{(i)})$.

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.

Can also learn a sparse network by including supplementary constraints on the weights.
Example (UFLDL)

- Train an autoencoder on $10 \times 10$ images with one hidden layer.
- Each hidden unit $i$ computes:
  \[ a^{(2)}_i = f \left( \sum_{j=1}^{100} W^{(1)}_{ij} x_j + b^{(1)}_i \right). \]
- Think of $a^{(2)}_i$ as some non-linear feature of the input $x$.

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

**Claim:**
\[ x_j = \frac{W^{(1)}_{ij}}{\sqrt{\sum_{j=1}^{100} (W^{(1)}_{ij})^2}}. \]
(Hint: Use Cauchy–Schwarz).

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

---

**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:** $96 \times 96$ image.
- Learn features on small $8 \times 8$ patches sampled randomly (e.g., using a sparse autoencoder).
- Run the trained model through all $8 \times 8$ patches of the image to get the feature activations.
Pool 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 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`