BUILDING A MLP THE HARD WAY

JOSEPH D. VIVIANO

1. Standardize input data

See `make_mnist_proc`. Image data is typically scaled so that each channel has zero mean.

IMPORTANT: the mean of each channel should be calculated on the TRAINING data only and applied to the VALID and TEST data. Otherwise you have information leakage between your TRAIN, VALID, and TEST sets which can give you higher (but invalid) results!

2. Weight initialization

See `_init_W`. Neural network weights should be initialized to have small random normally distributed values. Actually, people do even fancier things (i.e., Xavier initialization) but for this task, it is fine to just draw the weights from a gaussian.

Later on: try setting all the weights to zero and see what happens when you train the model!

3. Forward propagation

See `fprop`. Fill in the 4 calculations for the forward pass.

Below are the equations!
The dimension of $\mathbf{b}^{(1)}$ is: $\mathbf{b^{(1)}} \in \mathbb{R}^{d_h}$.

$$\mathbf{h}^a = \mathbf{W}^{(1)T} \cdot \mathbf{x} + \mathbf{b}^{(1)} \tag{3.1}$$

$$\mathbf{h}^s = g(\mathbf{h}^a) \tag{3.2}$$

Where $g(x)$ is the activation function (i.e., ReLU nonlinearity) applied element wise to the hidden layer. $g(x) = max(0, x)$.

The dimensions of $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$ are: $\mathbf{W}^{(2)} \in \mathbb{R}^{m \times d_h}$, and $\mathbf{b}^{(2)} \in \mathbb{R}^m$.

$$\mathbf{o}^a = \mathbf{W}^{(2)T} \cdot \mathbf{h^s} + \mathbf{b}^{(2)} \tag{3.3}$$

$$\mathbf{o}^a_k = \mathbf{W}^{(2)T}_k \cdot \mathbf{h^s} + b^{(2)}_k \tag{3.4}$$

Date: Jul 2019.

Let's define an m-class softmax: $softmax(x) = \frac{e^{x_i}}{\sum_{i=1}^{m} e^{x_i}}$. Therefore,

$$\mathbf{o}_k^s = \frac{e^{\mathbf{o}_k^a}}{\sum_{i=1}^{m} e^{\mathbf{o}_k^a}} \tag{3.5}$$

$$\sum_{i=1}^{m} \mathbf{o}_k^s = \sum_{i=1}^{m} \frac{e^{\mathbf{o}_k^a}}{\sum_{i=1}^{m} e^{\mathbf{o}_k^a}} \tag{3.6}$$

$$= \frac{\sum_{i=1}^{m} e^{\mathbf{o}_k^a}}{\sum_{i=1}^{m} e^{\mathbf{o}_k^a}} \tag{3.7}$$

$$= 1. \tag{3.8}$$

Since $e^x > 0$, the result will always be positive. This is crucial because we need the softmax to produce a probability distribution over our m output classes.

4. Backpropagation (with clues!)

See `bprop`. Fill in the 8 calculations for the backward pass. Fill in the remaining 4 equations to do the parameter updates.

4.1. Optimization.

The empirical risk $\hat{R}$ associated with the loss function (with a regulatrization term) is:

$$\hat{R} = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), y) + \lambda \Omega(\theta) \tag{4.1}$$

We proceed via gradient descent, i.e., we are going to optimize the parameters of the network such that we minimize this risk generated by some function f_θ shown some dataset D_n.

$$\theta^* = argmin \hat{R}_\lambda(f_\theta, D_n) \tag{4.2}$$

The gradient descent update looks like this:

$$\theta \leftarrow \theta - \eta \frac{\partial \hat{R}\lambda}{\partial \theta} + \eta \frac{\partial}{\partial \theta} \Omega(\theta) \quad \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \left(\frac{1}{n} \sum_{i=1}^{n} L(f_\theta(\mathbf{x}^{(i)}), y^{(1)}) \right) + \eta \frac{\partial}{\partial \theta} \Omega(\theta) \tag{4.3}$$

4.2. Loss of the prediction and it's derivative.

Now that we have done a forward pass to generate our predictions, we need to calculate the loss so we can figure out how to update the weights. We define the loss function (you don't need to implement this, but you need to call it appropriately). It is here for explaination, and in the code, as well.

$$L(\mathbf{o}^a, y) = -\sum_{k=1}^{M} y_k \log \left(\frac{e^{\mathbf{o}_k^a}}{\sum_{i=1}^{m} e^{\mathbf{o}_k^a}} \right) \tag{4.4}$$

Now we are ready to take the derivative with respect to the output layer when we have the correct class k (vs. the incorrect class i):

$$\frac{\partial L}{\partial \mathbf{o}^a_k} = -1 + \frac{e^{\mathbf{o}^a_k}}{\sum_{i \neq k}^{m} e^{\mathbf{o}^a_i} + e^{\mathbf{o}^a_k}} \tag{4.5}$$

$$= \frac{e^{\mathbf{o}^a_k}}{\sum_{k=1}^{m} e^{\mathbf{o}^a_k}} - 1 \tag{4.6}$$

$$= \mathbf{o}^s_k - 1 \tag{4.7}$$

Since $onehot_m(y) = 1$ when m is the target and is 0 otherwise, we see that the above is true. We can similarly take the derivative with respect to the output layer when we have the incorrect class i:

$$\frac{\partial L}{\partial \mathbf{o}^a_i} = \frac{e^{\mathbf{o}^a_i}}{\sum_{i \neq k}^{m} e^{\mathbf{o}^a_i} + e^{\mathbf{o}^a_k}} \tag{4.8}$$

$$= \mathbf{o}^s_i - 0 \tag{4.9}$$

See `softmax_backward` and `_softmax` for an implementation.

4.3. Backward through the weights.

$$\frac{\partial L}{\partial \mathbf{h}^s} = \frac{\partial L}{\partial \mathbf{o}^a} \frac{\partial \mathbf{o}^a}{\partial \mathbf{h}^s} \tag{4.10}$$

We have aleady defined $\frac{\partial L}{\partial \mathbf{o}^a}$. The gradient of $\frac{\partial \mathbf{o}^a}{\partial \mathbf{h}^s}$ is given by:

$$\frac{\partial \mathbf{o}^a}{\partial \mathbf{h}^s} = \frac{\partial}{\partial \mathbf{h}^s}(\mathbf{W}^{(2)T}\mathbf{h}^s + \mathbf{b}^{(2)}) \tag{4.11}$$

$$= \mathbf{W}^{(2)T} \tag{4.12}$$

4.4. Backward update to the second weight matrix and bias term.

Note that all **bold** variables represent matrices.

We have already defined $\frac{\partial L}{\partial o^a_k}$. $\frac{\partial o^a_k}{\partial W^{(2)}_{kj}}$ and $\frac{\partial \mathbf{o}^a_k}{\partial b^{(2)}_k}$ are given by:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{o}^a} \frac{\partial \mathbf{o}^a}{\partial \mathbf{W}^{(2)}} \tag{4.13}$$

$$\frac{\partial \mathbf{o}^a}{\partial \mathbf{W}^{(2)}} = \frac{\partial}{\partial \mathbf{W}^{(2)}}(\mathbf{W}^{(2)T}\mathbf{h}^s + \mathbf{b}^{(2)}) \tag{4.14}$$

$$= \mathbf{h}^{sT} \tag{4.15}$$

and

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{o}^a} \frac{\partial \mathbf{o}^a}{\partial \mathbf{b}^{(2)}} \tag{4.16}$$

$$\begin{aligned} \frac{\partial \mathbf{o}^a}{\partial \mathbf{b}^{(2)}} &= \frac{\partial}{\partial \mathbf{b}^{(2)}} (\mathbf{W}^{(2)T} \mathbf{h}^s + \mathbf{b}^{(2)}) && (4.17) \\ &= \mathbf{1} && (4.18) \end{aligned}$$

4.5. **Backward through the ReLU.**

$$\frac{\partial L}{\partial \mathbf{h}^a} = \frac{\partial L}{\partial \mathbf{h}^s} \frac{\partial \mathbf{h}^s}{\partial \mathbf{h}} \tag{4.19}$$

Where:

$$\frac{\partial \mathbf{h}^s}{\partial \mathbf{h}^a} = \mathbf{I}_{\{h_j^a > 0\}} \tag{4.20}$$

Where $\mathbf{I} \in \mathbb{R}^{d_h}$.

4.6. **Backward update to the first weights and biases.** We have already defined $\frac{\partial L}{\partial h_j^a}$.

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial L}{\partial \mathbf{h}^a} \frac{\partial \mathbf{h}^a}{\mathbf{W}^{(1)}} \tag{4.21}$$

$$\begin{aligned} \frac{\partial \mathbf{h}^a}{\mathbf{W}^{(1)}} &= \frac{\partial}{\mathbf{W}^{(1)}} (\mathbf{W}^{(1)} \mathbf{x}^T + \mathbf{b}^{(1)}) && (4.22) \\ &= \mathbf{x}^T && (4.23) \end{aligned}$$

and

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{\partial L}{\partial \mathbf{h}^a} \frac{\partial \mathbf{h}^a}{\mathbf{b}^{(1)}} \tag{4.24}$$

$$\begin{aligned} \frac{\partial \mathbf{h}^a}{\partial \mathbf{b}^{(1)}} &= \frac{\partial}{\mathbf{b}^{(1)}} (\mathbf{W}^{(1)} \mathbf{x}^T + \mathbf{b}^{(1)}) && (4.25) \\ &= \mathbf{1} && (4.26) \end{aligned}$$

Where $\mathbf{h}^a \in \mathbb{R}^{d_h}$, $\mathbf{W}^{(1)} \in \mathbb{R}^{d_h \times d}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{d_h}$, $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{1} \in \mathbb{R}^{d_h}$

4.7. **Gradient of Regularizers.** You don't have to implement this, but we're showing it to you here.

What you see below is called *elastic net* regularization. We have an l1 and l2 penalty on the weights of the model:

$$\begin{aligned} L(\theta) &= L(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) && (4.27) \\ &= \lambda_{11} ||\mathbf{W}^{(1)}||_1 + \lambda_{12} ||\mathbf{W}^{(1)}||_2^2 + \lambda_{21} ||\mathbf{W}^{(2)}||_1 + \lambda_{21} ||\mathbf{W}^{(2)}||_2^2 && (4.28) \end{aligned}$$

The gradient of the regularizers are added to the unregularized gradient of the parameters to update. The regularizers affect only the parameters $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

$$\Delta_{W^{(1)}} = -\frac{\partial L}{\partial \mathbf{W}^{(1)}} - \lambda_{11} sign(w_j^{(1)}) - \lambda_{12}[2\mathbf{W}^{(1)}] \tag{4.29}$$

$$\Delta_{W^{(2)}} = -\frac{\partial L}{\partial \mathbf{W}^{(2)}} - \lambda_{21} sign(w_j^{(2)}) - \lambda_{22}[2\mathbf{W}^{(2)}] \tag{4.30}$$

5. Training loop

See `train`.

Split the data into training, testing, and validation data appropriately. Sample minibatches of data to train the model using a forward pass and backward pass.

This should give you a good idea of how a typical ”training loop” should be structured.

6. Batch vs minibatch training & Gradient Checking

See `exp1`.

Here, we can explore the differences between training with stochastic gradient descent, minibatch stochastic gradient descent, and batch gradient descent. You can use `time.time` to see how this decision effects execution time.

This function will also do gradient checking (optional: see `grad_check`).

7. Network capacity

See `exp2`.

The hyperparamters you choose for your network (number of nodes in the hidden layer, regularization terms, learning rate, etc.) all effect the capactiy of your network.

Here you can play with the `options` dictionary and visualize the effect on the decision boundary for the two circles dataset. If you want to make your life easy, you can make a list of options and loop through them all. Inspect the outputs to understand how different decisions affect your model.

8. Experiment tracking

See `exp3`.

It is important to learn how to monitor a deep learning experiment. Remember, we want to build models that generalize well. Your task here is to train a model using good hyperparameters and plot the results obtained from training the model for the train, valid, and test sets. It is instructive to look at both the accuracy of the model, and the loss obtained, over epochs. These are called ”training curves” and are common in deep learning research.

Université de Montréal
Email address: joseph@viviano.ca