CS4501: Introduction to Computer Vision
Neural Networks (NNs)
Artificial Neural Networks (ANNs)
Multi-layer Perceptrons (MLPs)
Previous

- **Softmax Classifier**
  - Inference vs Training
  - Gradient Descent (GD)
  - Stochastic Gradient Descent (SGD)
  - mini-batch Stochastic Gradient Descent (SGD)

- **Max-Margin Classifier**

- **Regression vs Classification**

- **Issues with Generalization / Overfitting**
  - Regularization / momentum
Today’s Class

Neural Networks

• The Perceptron Model
• The Multi-layer Perceptron (MLP)
• Forward-pass in an MLP (Inference)
• Backward-pass in an MLP (Backpropagation)
Perceptron Model

Frank Rosenblatt (1957) - Cornell University

\[ f(x) = \begin{cases} 
1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\
0, & \text{otherwise} 
\end{cases} \]

More: https://en.wikipedia.org/wiki/Perceptron
Perceptron Model

Frank Rosenblatt (1957) - Cornell University

\[ f(x) = \begin{cases} 
1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\
0, & \text{otherwise} 
\end{cases} \]

More: https://en.wikipedia.org/wiki/Perceptron
Perceptron Model

Frank Rosenblatt (1957) - Cornell University

\[ f(x) = \begin{cases} 
1, & \text{if } \sum_{i=0}^{n} w_i x_i + b > 0 \\
0, & \text{otherwise} 
\end{cases} \]

More: https://en.wikipedia.org/wiki/Perceptron
Activation Functions

- **Step(x)**
- **Tanh(x)**
- **Sigmoid(x)**
- **ReLU(x) = max(0, x)**
Two-layer Multi-layer Perceptron (MLP)
Linear Softmax

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_d \ f_b] \]

\[
g_c = w_{c1}x_{i1} + w_{c2}x_{i2} + w_{c3}x_{i3} + w_{c4}x_{i4} + b_c
\]
\[
g_d = w_{d1}x_{i1} + w_{d2}x_{i2} + w_{d3}x_{i3} + w_{d4}x_{i4} + b_d
\]
\[
g_b = w_{b1}x_{i1} + w_{b2}x_{i2} + w_{b3}x_{i3} + w_{b4}x_{i4} + b_b
\]

\[
f_c = e^{g_c}/(e^{g_c} + e^{g_d} + e^{g_b})
\]
\[
f_d = e^{g_d}/(e^{g_c} + e^{g_d} + e^{g_b})
\]
\[
f_b = e^{g_b}/(e^{g_c} + e^{g_d} + e^{g_b})
\]
**Linear Softmax**

\[ x_i = [x_{i1}, x_{i2}, x_{i3}, x_{i4}] \]

\[ y_i = [1, 0, 0] \]

\[ \hat{y}_i = [f_c, f_d, f_b] \]

\[ g_c = w_{c1}x_{i1} + w_{c2}x_{i2} + w_{c3}x_{i3} + w_{c4}x_{i4} + b_c \]

\[ g_d = w_{d1}x_{i1} + w_{d2}x_{i2} + w_{d3}x_{i3} + w_{d4}x_{i4} + b_d \]

\[ g_b = w_{b1}x_{i1} + w_{b2}x_{i2} + w_{b3}x_{i3} + w_{b4}x_{i4} + b_b \]

\[ w = \begin{bmatrix} w_{c1} & w_{c2} & w_{c3} & w_{c4} \\ w_{d1} & w_{d2} & w_{d3} & w_{d4} \\ w_{b1} & w_{b2} & w_{b3} & w_{b4} \end{bmatrix} \]

\[ b = [b_c, b_d, b_b] \]

\[ f_c = \frac{e^{g_c}}{e^{g_c} + e^{g_d} + e^{g_b}} \]

\[ f_d = \frac{e^{g_d}}{e^{g_c} + e^{g_d} + e^{g_b}} \]

\[ f_b = \frac{e^{g_b}}{e^{g_c} + e^{g_d} + e^{g_b}} \]
Linear Softmax

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \]

\[ y_i = [1 \ 0 \ 0] \]

\[ \hat{y}_i = [f_c \ f_d \ f_b] \]

\[ w = \begin{bmatrix}
    w_{c1} & w_{c2} & w_{c3} & w_{c4} \\
    w_{d1} & w_{d2} & w_{d3} & w_{d4} \\
    w_{b1} & w_{b2} & w_{b3} & w_{b4}
\end{bmatrix} \]

\[ b = [b_c \ b_d \ b_b] \]

\[ g = wx^T + b^T \]

\[ f_c = \frac{e^{g_c}}{e^{g_c} + e^{g_d} + e^{g_b}} \]

\[ f_d = \frac{e^{g_d}}{e^{g_c} + e^{g_d} + e^{g_b}} \]

\[ f_b = \frac{e^{g_b}}{e^{g_c} + e^{g_d} + e^{g_b}} \]
Linear Softmax

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \]

\[ y_i = [1 \ 0 \ 0] \]

\[ \hat{y}_i = [f_c \ f_d \ f_b] \]

\[ g = wx^T + b^T \]

\[ w = \begin{bmatrix} w_{c1} & w_{c2} & w_{c3} & w_{c4} \\ w_{d1} & w_{d2} & w_{d3} & w_{d4} \\ w_{b1} & w_{b2} & w_{b3} & w_{b4} \end{bmatrix} \]

\[ b = [b_c \ b_d \ b_b] \]

\[ f = \text{softmax}(g) \]
Linear Softmax

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_a \ f_b] \]

\[ f = \text{softmax}(wx^T + b^T) \]
Two-layer MLP + Softmax

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_a \ f_b] \]

\[ a_1 = \text{sigmoid}(w_{[1]}x^T + b_{[1]}^T) \]
\[ f = \text{softmax}(w_{[2]}x^T + b_{[2]}^T) \]
N-layer MLP + Softmax

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \]

\[ y_i = [1 \ 0 \ 0] \]

\[ \hat{y}_i = [f_c \ f_d \ f_b] \]

\[ a_1 = \text{sigmoid}(w_{[1]}x^T + b_{[1]}^T) \]

\[ a_2 = \text{sigmoid}(w_{[2]}a_1^T + b_{[2]}^T) \]

\[ \ldots \]

\[ a_k = \text{sigmoid}(w_{[k]}a_{k-1}^T + b_{[k]}^T) \]

\[ \ldots \]

\[ f = \text{softmax}(w_{[n]}a_{n-1}^T + b_{[n]}^T) \]
How to train the parameters?

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_a \ f_b] \]

\[
\begin{align*}
a_1 &= \text{sigmoid}(w_{[1]}^T x_i + b_{[1]}^T) \\
a_2 &= \text{sigmoid}(w_{[2]}^T a_1 + b_{[2]}^T) \\
&\quad \vdots \\
a_k &= \text{sigmoid}(w_{[k]}^T a_{k-1} + b_{[k]}^T) \\
&\quad \vdots \\
f &= \text{softmax}(w_{[n]}^T a_{n-1} + b_{[n]}^T)
\end{align*}
\]
Forward pass (Forward-propagation)

\[ z_i = \sum_{i=0}^{n} w_{1ij}x_i + b_1 \]

\[ a_i = \text{Sigmoid}(z_i) \]

\[ p_1 = \sum_{i=0}^{n} w_{2i}a_i + b_2 \]

\[ y_1 = \text{Sigmoid}(p_i) \]

Loss = \( L(y_1, \hat{y}_1) \)
How to train the parameters?

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad \quad y_i = [1 \ 0 \ 0] \quad \quad \hat{y}_i = [f_c \ f_a \ f_b] \]

\[ a_1 = \text{sigmoid}(w_{[1]} x^T + b_{[1]}^T) \]
\[ a_2 = \text{sigmoid}(w_{[2]} a_1^T + b_{[2]}^T) \]
\[ \vdots \]
\[ a_k = \text{sigmoid}(w_{[k]} a_{k-1}^T + b_{[i]}^T) \]
\[ \vdots \]
\[ f = \text{softmax}(w_{[n]} a_{n-1}^T + b_{[n]}^T) \]

We can still use SGD

We need!

\[ \frac{\partial l}{\partial w_{[k]ij}} \quad \quad \frac{\partial l}{\partial b_{[k]i}} \]
How to train the parameters?

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_d \ f_b] \]

\[ a_1 = \text{sigmoid}(w_{[1]}x^T + b_{[1]}) \]
\[ a_2 = \text{sigmoid}(w_{[2]}a_1^T + b_{[2]}) \]
... 
\[ a_i = \text{sigmoid}(w_{[k]}a_{k-1}^T + b_{[k]}) \]
... 
\[ f = \text{softmax}(w_{[n]}a_{n-1}^T + b_{[n]}) \]

\[ l = \text{loss}(f, y) \]

We can still use SGD

We need!

\[ \frac{\partial l}{\partial w_{[k]ij}} \quad \frac{\partial l}{\partial b_{[k]i}} \]
How to train the parameters?

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_a \ f_b] \]

\[ a_1 = \text{sigmoid}(w_{[1]}x^T + b_{[1]}^T) \]
\[ a_2 = \text{sigmoid}(w_{[2]}a_1^T + b_{[2]}^T) \]
\[ \vdots \]
\[ a_i = \text{sigmoid}(w_{[k]}a_{i-1}^T + b_{[k]}^T) \]
\[ \vdots \]
\[ f = \text{softmax}(w_{[n]}a_{n-1}^T + b_{[n]}^T) \]

\[ l = \text{loss}(f, y) \]

We can still use SGD

We need!

\[
\frac{\partial l}{\partial w_{[k]ij}} \quad \frac{\partial l}{\partial b_{[k]i}}
\]
How to train the parameters?

\[ x_i = [x_{i1} \ x_{i2} \ x_{i3} \ x_{i4}] \quad y_i = [1 \ 0 \ 0] \quad \hat{y}_i = [f_c \ f_a \ f_b] \]

\[
\begin{align*}
    a_1 &= \text{sigmoid}(w_{[1]}x^T + b_{[1]}) \\
    a_2 &= \text{sigmoid}(w_{[2]}a_1^T + b_{[2]}) \\
    &\vdots \\
    a_i &= \text{sigmoid}(w_{[k]}a_{k-1}^T + b_{[k]}) \\
    &\vdots \\
    f &= \text{softmax}(w_{[n]}a_{n-1}^T + b_{[n]}) \\
    l &= \text{loss}(f, y)
\end{align*}
\]

\[
\frac{\partial l}{\partial w_{[k]ij}} = \frac{\partial l}{\partial a_{n-1}} \frac{\partial a_{n-1}}{\partial a_{n-2}} \cdots \frac{\partial a_{k}}{\partial a_{k-1}} \frac{\partial a_{k-1}}{\partial w_{[k]ij}}
\]
Backward pass (Back-propagation)

\[
\frac{\partial L}{\partial x_k} = \left( \frac{\partial}{\partial x_k} \sum_{i=0}^{n} w_{1ij} x_i + b_1 \right) \frac{\partial L}{\partial z_i} \\
\frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i} \text{Sigmoid}(z_i) \frac{\partial L}{\partial a_k} \\
\frac{\partial L}{\partial a_k} = \left( \frac{\partial}{\partial a_k} \sum_{i=0}^{n} w_{2i} a_i + b_2 \right) \frac{\partial L}{\partial p_1} \\
\frac{\partial L}{\partial p_1} = \frac{\partial}{\partial p_1} \text{Sigmoid}(p_i) \frac{\partial L}{\partial \hat{y}_1} \\
\frac{\partial L}{\partial \hat{y}_1} = \frac{\partial}{\partial \hat{y}_1} L(y_1, \hat{y}_1)
\]
(mini-batch) Stochastic Gradient Descent (SGD)

\[ \lambda = 0.01 \]

Initialize \( w \) and \( b \) randomly

\[ \text{for } e = 0, \text{ num\_epochs } \text{ do} \]

\[ \text{for } b = 0, \text{ num\_batches } \text{ do} \]

Compute: \[ \frac{dl(w, b)}{dw} \text{ and } \frac{dl(w, b)}{db} \]

Update \( w \):
\[ w = w - \lambda \frac{dl(w, b)}{dw} \]

Update \( b \):
\[ b = b - \lambda \frac{dl(w, b)}{db} \]

Print: \[ l(w, b) \] // Useful to see if this is becoming smaller or not.

\[ l(w, b) = \sum_{i \in B} -\log f_i,\text{label}(w, b) \]

For Softmax Classifier
Automatic Differentiation

You only need to write code for the forward pass, backward pass is computed automatically.

Frameworks such as Pytorch will “record” the operations performed on tensors and compute gradients through the “recorded” operations when requested.

Pytorch (Facebook -- mostly): https://pytorch.org/

Tensorflow (Google -- mostly): https://www.tensorflow.org/

DyNet (team includes UVA Prof. Yangfeng Ji): http://dynet.io/
Example

- Provided in Assignments 3 and Assignment 4.

- Let’s dissect Assignment 3...
import torch
from torch import nn

# This is the softmax classifier as studied in class.
class Classifier(nn.Module):

    def __init__(self):
        super(Classifier, self).__init__()
        # Linear transformation layer.
        # This computes a = wx + b where:
        # a is a vector of size 10
        # x: is a vector of size 3 x 32 x 32
        # b: is a vector of size 10
        # w: is a matrix of size 10 x (3 * 32 * 32)
        self.linear = nn.Linear(3 * 32 * 32, 10)

        # Softmax operator.
        # This is log(exp(a_i) / sum(a))
        self.log_softmax = nn.LogSoftmax(dim = 1)

    def forward(self, x):
        gx = self.linear(x)
yhat = self.log_softmax(gx)
        return yhat
Defining a Linear Softmax classifier

```python
# This is the softmax classifier as studied in class.
class Classifier(nn.Module):

    def __init__(self):
        super(Classifier, self).__init__()
        self.linear = nn.Linear(3 * 32 * 32, 10)
        self.log_softmax = nn.LogSoftmax(dim = 1)

    def forward(self, x):
        gx = self.linear(x)
        yhat = self.log_softmax(gx)
        return yhat
```
Using a Linear Softmax classifier

# Now let's try using it on a batch 5 of images represented
# as 5 vectors of size 3 * 32 * 32.
dummy_input = torch.rand(5, 3 * 32 * 32)
out = classifier(dummy_input)
print(out.exp())
Training a Linear Softmax classifier

classifier = Classifier()
criterion = nn.NLLLoss()

num_epochs = 30
learningRate = 0.001

classifier.train()

weight = classifier.linear.weight;
bias = classifier.linear.bias;

for epoch in range(0, num_epochs):
    for (i, (x, y)) in enumerate(trainLoader):

        x = x.view(x.shape[0], 3 * 32 * 32)
        yhat = classifier(x)
        loss = criterion(yhat, y)

        loss.backward()

        weight.data.add_(-learningRate * weight.grad.data)
bias.data.add_(-learningRate * bias.grad.data)

    print(loss.item())
What is `trainLoader`?

```python
batch_size = 128

# It additionally has utilities for threaded and multi-parallel data loading.
trainLoader = DataLoader(train_data, batch_size = batch_size,
                        shuffle = True, num_workers = 0)
valLoader = DataLoader(validation_data, batch_size = batch_size,
                        shuffle = False, num_workers = 0)

# Look-up python iterators if you need:
x, y = iter(trainLoader).next()
print('batch-of-images: ', x.shape)
print('batch-of-labels: ', y.shape)
```
Training a Linear Softmax classifier (improved)

classifier = Classifier()
criterion = nn.NLLLoss()

num_epochs = 30
optimizer = torch.optim.SGD(classifier.parameters(), lr = 0.001, momentum = 0.9)

classifier.train()

for epoch in range(0, num_epochs):
    for (i, (x, y)) in enumerate(trainLoader):
        x = x.view(x.shape[0], 3 * 32 * 32)
yhat = classifier(x)
loss = criterion(yhat, y)

    loss.backward()
    optimizer.zero_grad()
    optimizer.step()

    print(loss.item())

This depends on the model but we don’t need it anymore
import torch
from torch import nn

# This is the softmax classifier as studied in class.
class Classifier(nn.Module):

    def __init__(self):
        super(Classifier, self).__init__()
        self.linear1 = nn.Linear(3 * 32 * 32, 512)
        self.sigmoid = nn.Sigmoid()
        self.linear2 = nn.Linear(512, 10)
        self.log_softmax = nn.LogSoftmax(dim = 1)

    def forward(self, x):
        gx = self.linear1(x)
        gxs = self.sigmoid(gx)
        mx = self.linear2(gxs)
        yhat = self.log_softmax(mx)
        return yhat
Questions?