IE 534/CS 598 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2018

Lecture 4

We now consider a multi-layer neural network.

$$
\begin{aligned}
Z^1 &= W^1 x + b^1, \\
H^1 &= \sigma(Z^1), \\
Z^\ell &= W^\ell H^{\ell-1} + b^\ell, \quad \ell = 2, \ldots, L, \\
H^\ell &= \sigma(Z^\ell), \quad \ell = 2, \ldots, L, \\
U &= W^{L+1} H^L + b^{L+1}, \\
f(x;\theta) &= F_{\text{softmax}}(U).
\end{aligned} \tag{1}
$$

The neural network has L hidden layers followed by a softmax function. Each layer of the neural network has d_H hidden units. The ℓ-th hidden layer is $H^\ell \in \mathbb{R}^{d_H}$. H^ℓ is produced by applying an element-wise nonlinearity to the input $Z^\ell \in \mathbb{R}^{d_H}$. Using a slight abuse of notation,

$$
\sigma(Z^\ell) = \Big(\sigma(Z_0^\ell), \sigma(Z_1^\ell), \ldots, \sigma(Z_{d_H-1}^\ell) \Big). \tag{2}
$$

The SGD algorithm for updating θ is:

- Randomly select a new data sample (X, Y).
- Compute the forward step $Z^1, H^1, \ldots, Z^L, H^L, U, f(X;\theta)$, and $\rho := \rho\big(f(X;\theta), Y\big)$.
- Calculate the partial derivative

$$\frac{\partial \rho}{\partial U} = -\Bigg(e(Y) - f(X;\theta)\Bigg). \tag{3}$$

- Calculate the partial derivatives $\frac{\partial \rho}{\partial b^{L+1}}$, $\frac{\partial \rho}{\partial W^{L+1}}$, and δ^L.
- For $\ell = L-1, \ldots, 1$:
  - Calculate δ^ℓ via the formula

$$\delta^\ell = \left(W^{\ell+1}\right)^\top \left(\delta^{\ell+1} \odot \sigma'(Z^{\ell+1})\right). \tag{4}$$

  - Calculate the partial derivatives with respect to W^ℓ and b^ℓ.
- Update the parameters θ with a stochastic gradient descent step.

- In principle, the neural network can more accurately fit more complex nonlinear relationships with more layers.
- A "deep neural network" is a highly nonlinear model due to repeated applications of element-wise nonlinearities.
- However, the numerical estimation of the neural network with stochastic gradient descent suffers a limitation called the **vanishing gradient problem** as the number of layers is increased.

- As the number of layers L is increased, the magnitude of the gradient with respect to the parameters in the lower layers becomes small (e.g., $\frac{\partial \rho}{\partial W^\ell}$ for $\ell \ll L$).

- This leads to (stochastic) gradient descent converging extremely slowly.

- Essentially, the lower layers take an impractically long amount of time to train.

Example

Each hidden layer has a single unit (i.e., $d_H = 1$) and $\sigma(\cdot)$ is a sigmoid function. Let's initialize $b^\ell = 0$ and $W^\ell = \frac{1}{2}$. The input dimension $d = 1$ and the output is also one-dimensional. Assume $x = 1$ and let the loss function be $\rho(z, y) = (y - z)^2$.

Then,

$$\left| \frac{\partial \rho}{\partial W^\ell} \right| \leq C 2^{-(L-\ell)}, \tag{5}$$

- The vanishing gradient problem can also occur due to **saturation**.

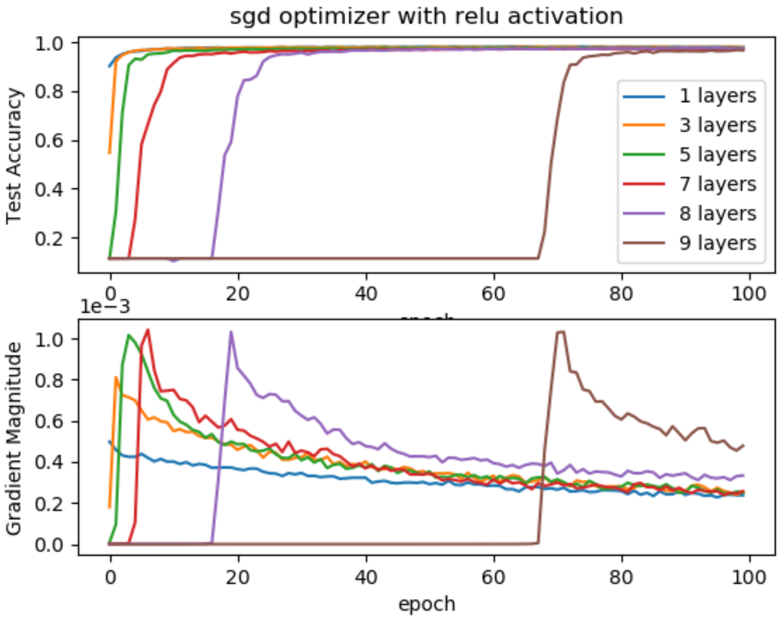
- Saturation occurs when the inputs to the hidden units have very large magnitudes.

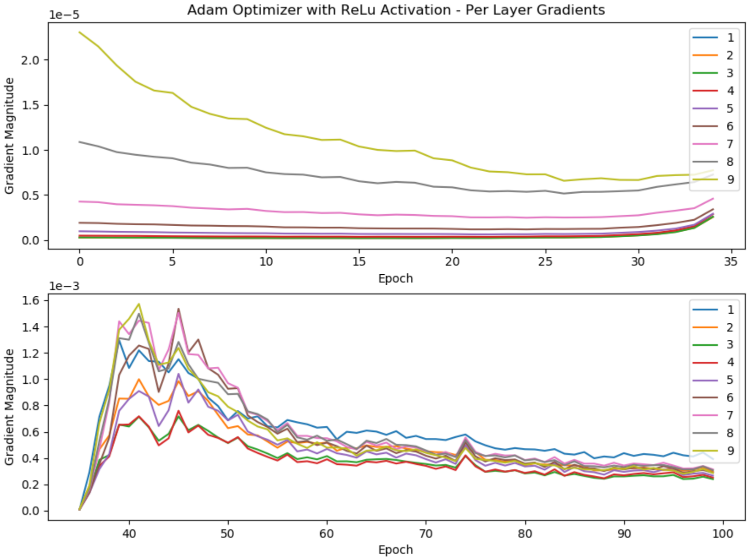
- For example, recall that if $\sigma(\cdot)$ is a sigmoidal function, then its derivative is

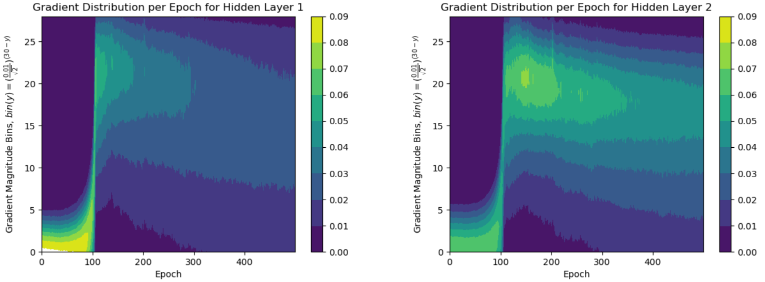
$$\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big). \tag{6}$$

Since $\lim_{\|z\| \to \infty} \sigma(z) \to 0$,

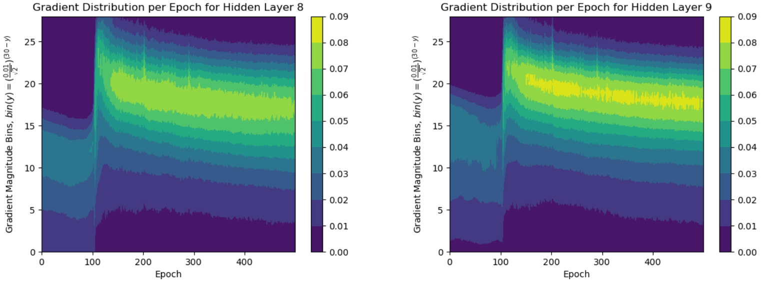
$$\lim_{\|z\| \to \infty} \sigma'(z) = 0. \tag{7}$$


sgd optimizer with relu activation
Test Accuracy
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100
1 layers
3 layers
5 layers
7 layers
8 layers
9 layers
1e−3
Gradient Magnitude
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
epoch

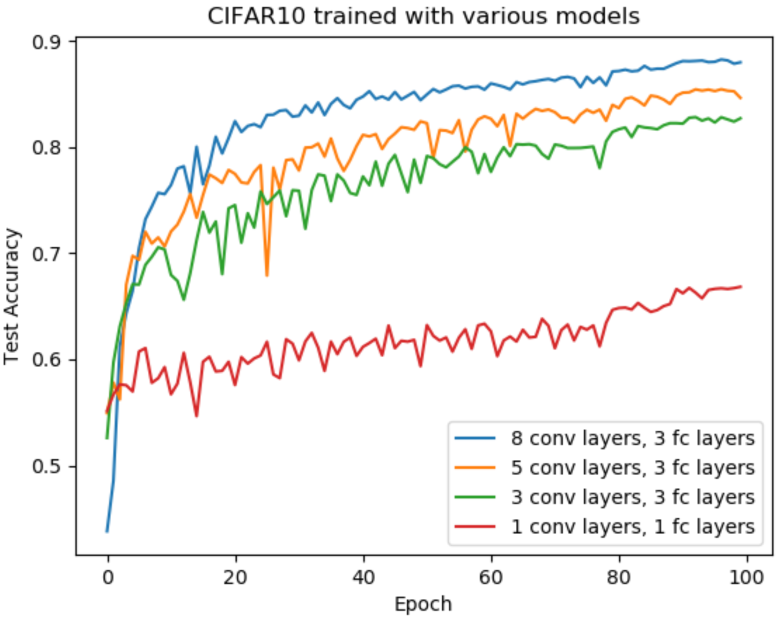

Adam Optimizer with ReLu Activation - Per Layer Gradients
1e−5
Gradient Magnitude
Epoch
1e−3
Gradient Magnitude
Epoch
1
2
3
4
5
6
7
8
9



Contour plots of distribution of gradient magnitudes.



Contour plots of distribution of gradient magnitudes.


CIFAR10 trained with various models
Test Accuracy
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100
Epoch
8 conv layers, 3 fc layers
5 conv layers, 3 fc layers
3 conv layers, 3 fc layers
1 conv layers, 1 fc layers

REGULAR

	kernel	input	output	num weights
conv1	4	3	128	6144
conv2	4	128	128	262144
conv3	4	128	128	262144
conv4	4	128	128	262144
conv5	4	128	128	262144
conv6	3	128	128	147456
conv7	3	128	128	147456
conv8	3	128	128	147456
fc1		2048	500	1024000
fc2		500	500	250000
fc3		500	10	5000
				2776088

SHALLOW

	kernel	input	output	num weights
conv1	4	3	256	12288
conv3	4	256	256	1048576
conv5	4	256	128	524288
fc1		2048	500	1024000
fc2		500	500	250000
fc3		500	10	5000
				2864152

EXTRA SHALLOW

	kernel	input	output	num weights
conv1	4	3	700	33600
conv3	4	700	128	1433600
fc1		2048	500	1024000
fc2		500	500	250000
fc3		500	10	5000
				2746200

2-LAYER

	kernel	input	output	num weights
conv1	5	3	3000	225000
fc1		192000	10	1920000
				2145000

We first consider a convolution network with a single hidden layer. Let the input image be $X \in \mathbb{R}^{d \times d}$ and a filter $K \in \mathbb{R}^{k_y \times k_x}$.

We define a convolution of the matrix X with the filter K as the map $X * K : \mathbb{R}^{d \times d} \times \mathbb{R}^{k_y \times k_x} \to \mathbb{R}^{(d-k_y+1)\times(d-k_x+1)}$ where

$$(X * K)_{i,j} = \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} X_{i+m,j+n}. \tag{8}$$

The hidden layer applies an element-wise nonlinearity $\sigma : \mathbb{R} \to \mathbb{R}$ to each element of the matrix $X * K$. We define the variable $Z \in \mathbb{R}^{(d-k_y+1)\times(d-k_x+1)}$ and the hidden layer $H \in \mathbb{R}^{(d-k_y+1)\times(d-k_x+1)}$ where

$$\begin{aligned} H_{i,j} &= \sigma\big((Z)_{i,j}\big), \\ Z &= X * K. \end{aligned} \tag{9}$$

Y is the label for the image X and takes values in the set $\mathcal{Y} = \{0, 1, \ldots, K-1\}$.

$$
\begin{aligned}
f(x;\theta) &= F_{\text{softmax}}(U), \\
U_k &= W_{k,:,:} \cdot H + b_k,
\end{aligned}
\tag{10}
$$

where $W \in \mathbb{R}^{K \times (d-k_y+1) \times (d-k_x+1)}$, $b \in \mathbb{R}^K$, $U \in \mathbb{R}^K$, and $W_{k,:,:} \cdot H = \sum_{i,j} W_{k,i,j} H_{i,j}$.

The collection of parameters is $\theta = \{K, W, b\}$. The cross-entropy error for a single data sample (X, Y) is

$$
\begin{aligned}
\rho &:= \rho(f(X;\theta), Y) \\
&= -\log\Big(f_Y(X;\theta) \Big).
\end{aligned}
\tag{11}
$$

The single layer convolution network is:

$$
\begin{aligned}
Z &= X * K, \\
H &= \sigma(Z), \\
U_k &= W_{k,:,:} \cdot H + b_k, \quad k = 0, \ldots, K-1, \\
f(x;\theta) &= F_{\mathrm{softmax}}(U). \qquad (12)
\end{aligned}
$$

The cross-entropy error for a single data sample (X, Y) is

$$
\begin{aligned}
\rho &:= \rho(f(X;\theta), Y) \\
&= -\log\Big(f_Y(X;\theta)\Big). \qquad (13)
\end{aligned}
$$

The stochastic gradient descent algorithm for updating θ is:

- Randomly select a new data sample (X, Y).
- Compute the forward step (Z, H, U, ρ).
- Calculate the partial derivatives $(\frac{\partial \rho}{\partial U}, \delta, \frac{\partial \rho}{\partial K})$.
- Update the parameters $\theta = \{K, W, b\}$ with a stochastic gradient descent step:

$$\begin{aligned} b^{(\ell+1)} &= b^{(\ell)} - \alpha^{(\ell)} \frac{\partial \rho}{\partial U}, \\ W_{k,\cdot,\cdot}^{(\ell+1)} &= W_{k,\cdot,\cdot}^{(\ell)} - \alpha^{(\ell)} \frac{\partial \rho}{\partial U_k} H, \\ K^{(\ell+1)} &= K^{(\ell)} - \alpha^{(\ell)} \Big(X * \big(\sigma'(Z) \odot \delta \big) \Big), \end{aligned} \tag{14}$$

where $\alpha^{(\ell)}$ is the learning rate.

- Convolutions are invariant to translations.

- Consider an image $X : \mathbb{Z} \times \mathbb{Z} \to \mathbb{R}$ and the convolution

$$Z_{i,j} = (X * K)_{i,j} = \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} X_{i+m,j+n}. \tag{15}$$

- Let $Y = t(X)$, defined as

$$Y_{i,j} = t(X)_{i,j} = X_{i-b_1,j-b_2}. \tag{16}$$

Then,

$$\begin{aligned}
(Y * K)_{i,j} &= \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} Y_{i+m,j+n} \\
&= \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} X_{i+m-b_1,j+n-b_2} \\
&= (X * K)_{i-b_1,j-b_2} = Z_{i-b_1,j-b_2}.
\end{aligned} \tag{17}$$

We have that

$$
\begin{aligned}
(Y * K)_{i,j} &= (X * K)_{i-b_1,j-b_2} \\
&= Z_{i-b_1,j-b_2} \\
&= t(Z)_{i,j}
\end{aligned}
\tag{18}
$$

Therefore,

$$
t(X) * K = t(X * K). \tag{19}
$$

Shifting the data does not change the output of the convolution operation (up to translations)!