IE 534/CS 598 Deep Learning

University of Illinois at Urbana-Champaign

Fall 2018

Lecture 4

We now consider a multi-layer neural network.

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

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,

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

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

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

  - 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,

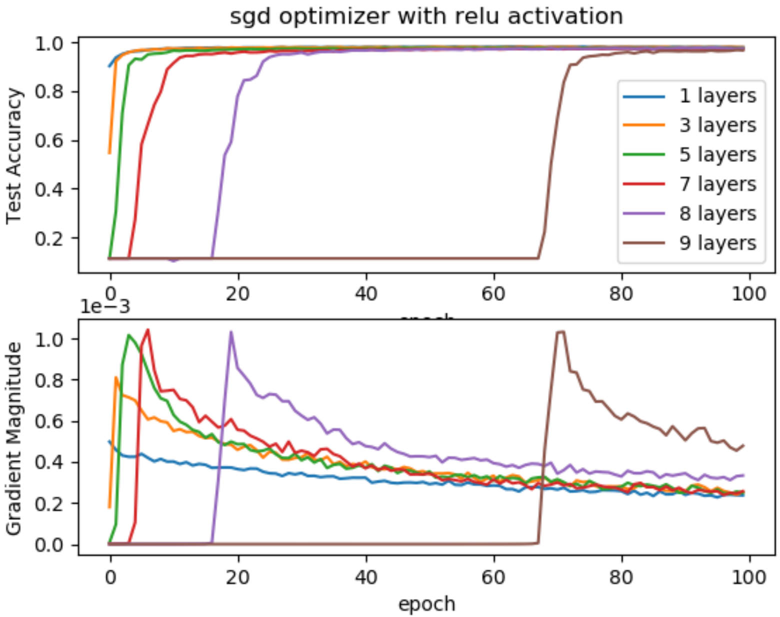
$$\left| \frac{\partial \rho}{\partial W^\ell} \right| \leq C 2^{-(L-\ell)}, \tag{5}$$

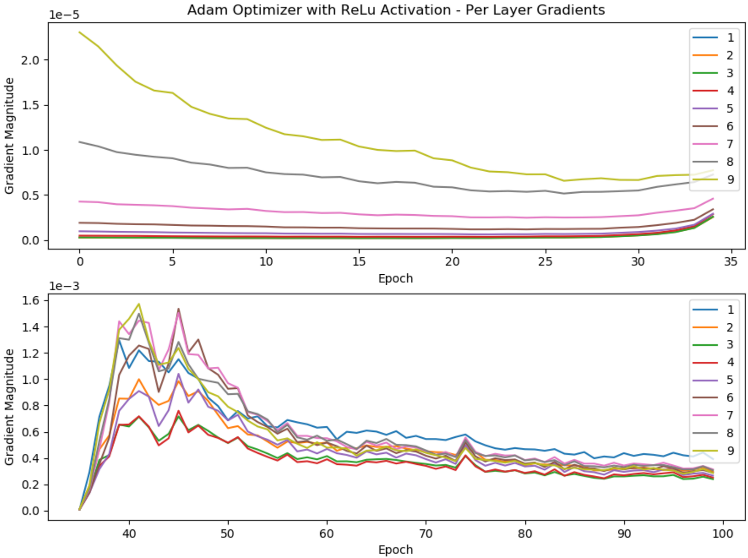
- The vanishing gradient problem can also occur due to **saturation**.
- Saturation occurs when the inputs to the hidden units have very large magnitudes.
- For example, recall that if $\sigma(\cdot)$ is a sigmoidal function, then its derivative is

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

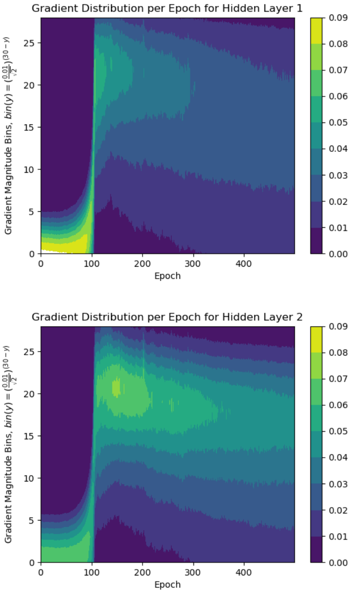
Since $\lim_{\|z\| \to \infty} \sigma(z) \to 0$,

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

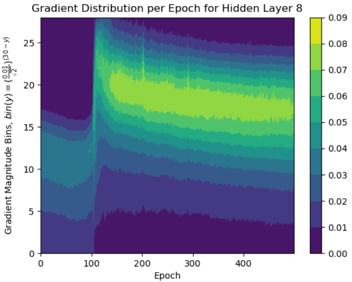

sgd optimizer with relu activation
Test Accuracy
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100
epoch
1 layers
3 layers
5 layers
7 layers
8 layers
9 layers
1e−3
Gradient Magnitude
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
epoch

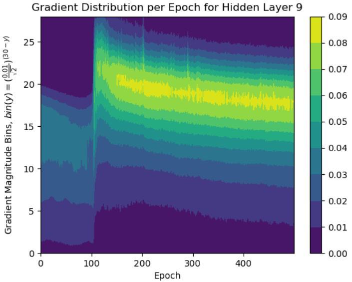


Adam Optimizer with ReLu Activation - Per Layer Gradients
1e−5
Gradient Magnitude
Epoch
1
2
3
4
5
6
7
8
9
1e−3
Gradient Magnitude
Epoch
1
2
3
4
5
6
7
8
9

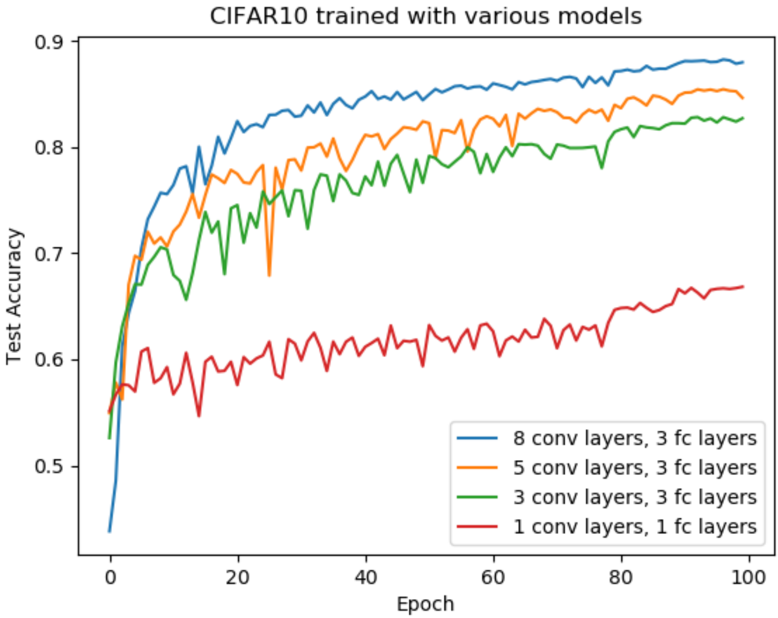


Contour plots of distribution of gradient magnitudes.





Contour plots of distribution of gradient magnitudes.


CIFAR10 trained with various models
Test Accuracy
Epoch
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100
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:

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

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

- The feature maps for the hidden layer are represented by a variable $H \in \mathbb{R}^{(d-k_y+1)\times(d-k_x+1)\times C}$.
- The convolution layer has an array (or "stack") of C filters where each filter is of size $k_y \times k_x$.
- The filters are given by the variable $K \in \mathbb{R}^{d_y \times d_x \times C}$.

The hidden layer H is given by:

$$H_{i,j,p} = \sigma\left(\sum_{m=0}^{k_y^\ell - 1} \sum_{n=0}^{k_x^\ell - 1} K_{m,n,p}^\ell X_{i+m,j+n} \right). \tag{15}$$

Therefore,

$$\begin{aligned} H_{:,:,p} &= \sigma\Big(Z_{:,:,p}, \Big), \\ Z_{:,:,p} &= X_{:,:} * K_{:,:,p}. \end{aligned} \tag{16}$$

The output of the network is simply the softmax function applied to a linear function of the hidden layer H:

$$
\begin{aligned}
f(x;\theta) &= F_{\mathrm{softmax}}(U), \\
U_k &= W_{k,:,:,:} \cdot H + b_k,
\end{aligned}
\tag{17}
$$

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

The single layer convolution network **with multiple channels** is:

$$
\begin{aligned}
Z_{:,:,p} &= X_{:,:} * K_{:,:,p}, \\
H_{:,:,p} &= \sigma\left(Z_{:,:,p} \right), \\
U_k &= W_{k,:,:,:} \cdot H + b_k, \\
f(x;\theta) &= F_{\text{softmax}}(U).
\end{aligned} \tag{18}
$$

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

$$
\begin{aligned}
\rho &:= \rho(f(X;\theta), Y) \\
&= -\log\left(f_Y(X;\theta) \right).
\end{aligned} \tag{19}
$$

Define

$$\begin{aligned} \delta_{i,j,p} &:= \frac{\partial \rho}{\partial H_{i,j,p}} = \sum_{k=0}^{K-1} \frac{\partial \rho}{\partial U_k} W_{k,i,j,p} \\ &= \frac{\partial \rho}{\partial U} \cdot W_{:,i,j,p}. \end{aligned} \tag{20}$$

The backpropagation algorithm is essentially the same as before, with

$$\frac{\partial \rho}{\partial K_{:,:,p}} = X * \left(\sigma'(Z_{:,:,p}) \odot \delta_{:,:,p}\right), \tag{21}$$

and

$$\begin{aligned} \frac{\partial \rho}{\partial b} &= \frac{\partial \rho}{\partial U}, \\ \frac{\partial \rho}{\partial W_{k,:,:,:}} &= \frac{\partial \rho}{\partial U_k} H. \end{aligned} \tag{22}$$

Multi-layer convolution networks:

- The input image is $X \in \mathbb{R}^{d \times d \times C^0}$.
- The ℓ-th convolution layer contains C^ℓ "feature maps".
- The number of feature maps C^ℓ is often called the "number of channels" for layer ℓ.
- The ℓ-th hidden layer is $H^\ell \in \mathbb{R}^{d_y^\ell \times d_x^\ell \times C^\ell}$. The first feature map $H^0 = X$.
- The filters for the ℓ-layer are given by the variable $K^\ell \in \mathbb{R}^{d_y^\ell \times d_x^\ell \times C^\ell \times C^{\ell-1}}$.

$$H^\ell_{i,j,p} = \sigma\left(\sum_{p'=0}^{C^{\ell-1}-1} \sum_{m=0}^{k_y^\ell - 1} \sum_{n=0}^{k_x^\ell - 1} K^\ell_{m,n,p,p'} H^{\ell-1}_{i+m,j+n,p'} \right). \tag{23}$$

The height d_y^ℓ and width d_x^ℓ of the feature maps in the ℓ-th layer depend upon the height $d_y^{\ell-1}$ and width $d_x^{\ell-1}$ of the feature maps in the previous layer and the size of the filters $k_y^\ell \times k_x^\ell$:

$$\begin{aligned} d_y^\ell &= d_y^{\ell-1} - k_y^\ell + 1, \\ d_x^\ell &= d_x^{\ell-1} - k_x^\ell + 1. \end{aligned} \tag{24}$$

The size of the features maps monotonically decreases as the number of layers increase. In particular,

$$\begin{aligned} d_y^\ell &= d + \sum_{i=1}^{\ell} (-k_y^i + 1), \\ d_x^\ell &= d + \sum_{i=1}^{\ell} (-k_x^i + 1). \end{aligned} \tag{25}$$

- Typically followed by 1-2 fully-connected layers.
- ReLU hidden units
- Common sizes for filters: 3×3, 5×5, 8×8.
- Pooling, strides, padding.
- 3-d convolutions.

Why do convolution networks work well for image recognition?

- Invariant to translations.
- Shared weights vs. fully-connected
- Learn about all weights no matter where image is located in the image.
- Much fewer parameters than fully-connected.

- Convolutions are invariant to translations.

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

$$Z_{i,j} = (X * K)_{i,j} = \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} X_{i+m,j+n}. \tag{26}$$

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

$$Y_{i,j} = t(X)_{i,j} = X_{i-b_1,j-b_2}. \tag{27}$$

Then,

$$\begin{aligned}
(Y * K)_{i,j} &= \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} Y_{i+m,j+n} \\
&= \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} K_{m,n} X_{i+m-b_1,j+n-b_2} \\
&= (X * K)_{i-b_1,j-b_2} = Z_{i-b_1,j-b_2}.
\end{aligned} \tag{28}$$

We have that

$$
\begin{aligned}
(Y * K)_{i,j} &= (X * K)_{i-b_1,j-b_2} \\
&= Z_{i-b_1,j-b_2} \\
&= t(Z)_{i,j} \qquad (29)
\end{aligned}
$$

Therefore,

$$t(X) * K = t(X * K). \qquad (30)$$

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

Importance of understanding backpropagation:

- Low-level implementation
- Implementation of new methods and models
- Understanding backpropagation $\longrightarrow$ vanishing gradient problem $\longrightarrow$ new models and training methods.
- "Standard" automatic differentiation code may not be optimal.

Automatic differentiation computes the chain rule for a composition of functions

$$g(x) = f^N\left(f^{N-1}\big(\cdots f^1(x)\big)\right). \tag{31}$$

$y^n = f^n\big(\cdots f^1(x)\big)$, $y^0 = x$, and $D^n = \frac{\partial y^n}{\partial y^{n-1}}$. Then,

$$\frac{\partial y^N}{\partial x} = D^N D^{N-1} \cdots D^1. \tag{32}$$

Suppose $g(x) : \mathbb{R}^d \to \mathbb{R}^K$, $f^1 : \mathbb{R}^d \to \mathbb{R}^H$, $f^n : \mathbb{R}^H \to \mathbb{R}^H$ for $1 < n < N$, and $f^N : \mathbb{R}^H \to \mathbb{R}^K$.

What is the optimal way to compute $\frac{\partial y^N}{\partial x}$?

The partial derivative of $g(x) : \mathbb{R}^d \to \mathbb{R}^K$ is:

$$\frac{\partial y^N}{\partial x} = D^N D^{N-1} \cdots D^1. \tag{33}$$

Forward mode:

$$\mathcal{O}\left(H^2 d + (N-2) H^2 d + KHd \right). \tag{34}$$

Reverse mode:

$$\mathcal{O}\left(H^2 K + (N-2) K H^2 + KHd \right). \tag{35}$$

Forward mode is better if $K > d$.

Reverse mode is better if $K < d$.

- In fact, backpropagation is an example of reverse mode differentiation.
- PyTorch and TensorFlow use reverse mode automatic differentiation.
- The optimal sequence of chain rule operations to compute the Jacobian ("Optimal Jacobian accumulation") is NP-complete. (See Naumann, *Mathematical Programming*, 2008.)
- Therefore, automatic differentiation algorithms may not be optimal.
- For example, consider the softmax function.

Other features of PyTorch:

- Dynamic, define-**by**-run. TensorFlow is static, define-**and**-run.
- Implementation in C++.
- Safeguards against in-place operations.
- Memory management: automatically frees memory when possible.
- See "Automatic differentiation in PyTorch" by Paszke et al., NIPS, 2017.

3-d convolutions:

Let the input image be $X \in \mathbb{R}^{d \times d \times d}$ and a filter $K \in \mathbb{R}^{k_y \times k_x \times k_z}$.

We define a 3-dimensional convolution of the matrix X with the filter K as the map

$$X * K : \mathbb{R}^{d \times d \times d} \times \mathbb{R}^{k_y \times k_x \times k_z} \to \mathbb{R}^{(d-k_y+1)\times(d-k_x+1)\times(d-k_z+1)}, \quad (36)$$

where

$$(X * K)_{i,j,q} = \sum_{m=0}^{k_y-1} \sum_{n=0}^{k_x-1} \sum_{r=0}^{k_z-1} K_{m,n,r} X_{i+m,j+n,q+r}. \quad (37)$$