EE 628
Deep Learning
Fall 2019

Lecture 6

10/03/2019

Assistant Prof. Sergul Aydore

Department of Electrical and Computer Engineering

Overview

- Last lecture we covered
  - Backpropagation
  - Optimization Algorithms

- Today, we will cover
  - More on Optimization Algorithms
  - Convolutional layers

Momentum

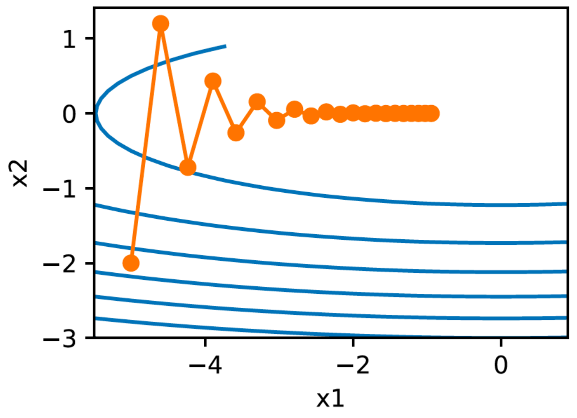
- Let’s start with an example $f(\mathbf{x}) = 0.1\, x_1^2 + 2x_2^2$

Momentum

- Let's start with an example $f(\mathbf{x}) = 0.1\, x_1^2 + 2x_2^2$
- Let's implement GD on this objective function and demonstrate the iterative trajectory of the updates with learning rate 0.4

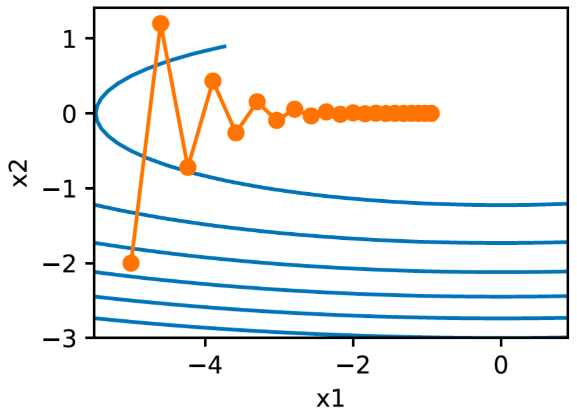
Momentum

- Let's start with an example $f(\mathbf{x}) = 0.1\, x_1^2 + 2x_2^2$
- Let's implement GD on this objective function and demonstrate the iterative trajectory of the updates with learning rate 0.4
  - The slope of the objective function has a larger value in the vertical direction
  - Therefore, the variable will move more in Vertical direction



Momentum

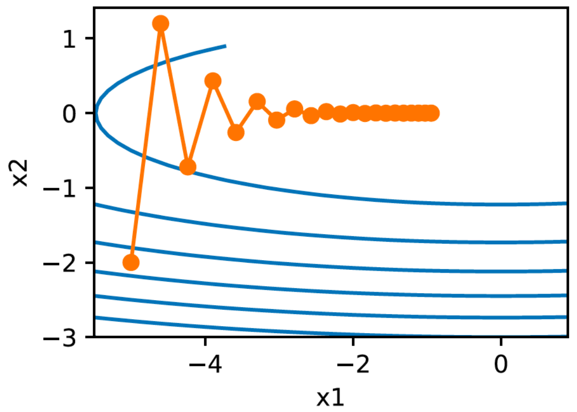
- Let’s start with an example $f(\mathbf{x}) = 0.1\, x_1^2 + 2x_2^2$
- Let’s implement GD on this objective function and demonstrate the iterative trajectory of the updates with learning rate 0.4
  - The slope of the objective function has a larger value in the vertical direction
  - Therefore, the variable will move more in Vertical direction



- Small learning rate will cause the variable move slower toward the optimal solution

Momentum

- Let's start with an example $f(\mathbf{x}) = 0.1\, x_1^2 + 2x_2^2$
- Let's implement GD on this objective function and demonstrate the iterative trajectory of the updates with learning rate 0.4
  - The slope of the objective function has a larger value in the vertical direction
  - Therefore, the variable will move more in Vertical direction



- Small learning rate will cause the variable move slower toward the optimal solution
- Large learning rate will make the variable overshoot in vertical direction

Momentum

- Momentum method was proposed to address this problem

x1

Momentum

- Momentum method was proposed to address this problem
- At time step $t > 0$, momentum modifies the steps at each iteration as follows

$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta_t \mathbf{g}_t,$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{v}_t,$$

Momentum

- Momentum method was proposed to address this problem
- At time step $t > 0$, momentum modifies the steps at each iteration as follows

$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta_t \mathbf{g}_t,$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{v}_t,$$

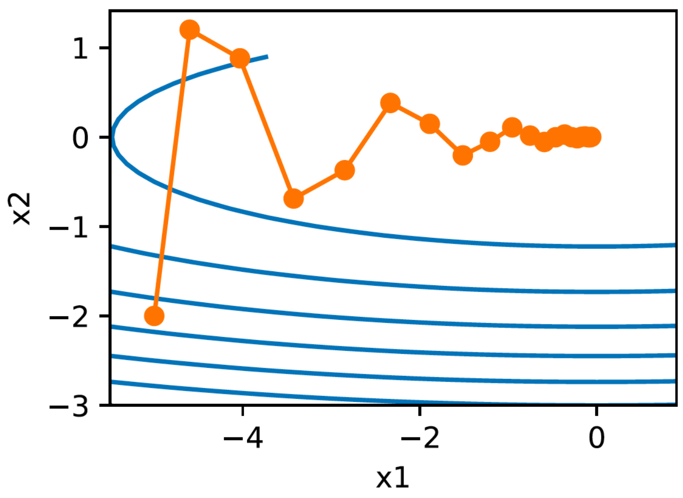
- When we implement this on the previous example we see that the trajectory is smoother in vertical direction and approaches the optimal solution faster in the horizontal direction.

Momentum

- Momentum method was proposed to address this problem
- At time step $t > 0$, momentum modifies the steps at each iteration as follows

$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta_t \mathbf{g}_t,$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{v}_t,$$

- When we implement this on the previous example we see that the trajectory is smoother in vertical direction and approaches the optimal solution faster in the horizontal direction.



Momentum

- To understand momentum, we can expanding the velocity variable $\mathbf{v}_t$

Momentum

- To understand momentum, we can expanding the velocity variable $\mathbf{v}_t$ over time:

$$
\begin{aligned}
\mathbf{v}_t &= \eta_t \mathbf{g}_t + \gamma \mathbf{v}_{t-1}, \\
&= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \gamma \mathbf{v}_{t-1}, \\
&\dots \\
&= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \ldots + \gamma^{t-1} \eta_1 \mathbf{g}_1.
\end{aligned}
$$

Momentum

- To understand momentum, we can expanding the velocity variable $\mathbf{v}_t$ over time:

$$\begin{aligned}\mathbf{v}_t &= \eta_t \mathbf{g}_t + \gamma \mathbf{v}_{t-1}, \\ &= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \gamma \mathbf{v}_{t-1}, \\ &\ldots \\ &= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \ldots + \gamma^{t-1} \eta_1 \mathbf{g}_1.\end{aligned}$$

- $\mathbf{v}_t$ is a weighted sum over all past gradients multiplied by the according learning rate

Momentum

- To understand momentum, we can expanding the velocity variable $\mathbf{v}_t$ over time:

$$
\begin{aligned}
\mathbf{v}_t &= \eta_t \mathbf{g}_t + \gamma \mathbf{v}_{t-1}, \\
&= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \gamma \mathbf{v}_{t-1}, \\
&\ldots \\
&= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \ldots + \gamma^{t-1} \eta_1 \mathbf{g}_1.
\end{aligned}
$$

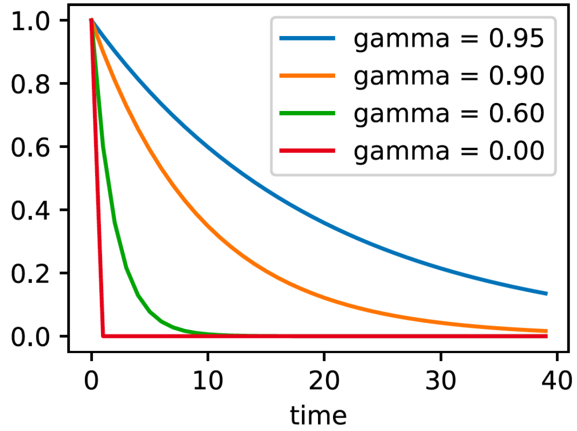
- $\mathbf{v}_t$ is a weighted sum over all past gradients multiplied by the according learning rate
- What happens when γ is too large or small?

Momentum

- To understand momentum, we can expanding the velocity variable $\mathbf{v}_t$ over time:

$$\begin{aligned}\mathbf{v}_t &= \eta_t \mathbf{g}_t + \gamma \mathbf{v}_{t-1}, \\ &= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \gamma \mathbf{v}_{t-1}, \\ &\dots \\ &= \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \ldots + \gamma^{t-1} \eta_1 \mathbf{g}_1.\end{aligned}$$

- $\mathbf{v}_t$ is a weighted sum over all past gradients multiplied by the according learning rate
- What happens when γ is too large or small?



Adagrad

- So far we showed that each variable uses the same learning rate

Adagrad

- So far we showed that each variable uses the same learning rate
- For example, if we assume that the objective function is f and the independent variable is a two-dimensional vector $[x_1, x_2]$, the update for each element is $x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1} \qquad x_2 \leftarrow x_2 - \eta \frac{\partial f}{\partial x_2}$

Adagrad

- So far we showed that each variable uses the same learning rate
- For example, if we assume that the objective function is f and the independent variable is a two-dimensional vector $[x_1, x_2]$, the update for each element is $x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1}$ $\quad x_2 \leftarrow x_2 - \eta \frac{\partial f}{\partial x_2}$
- We saw that this would be a problem when there is a big difference between x_1 and x_2.

Adagrad

- So far we showed that each variable uses the same learning rate
- For example, if we assume that the objective function is f and the independent variable is a two-dimensional vector $[x_1, x_2]$, the update for each element is $x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1}$ $\quad x_2 \leftarrow x_2 - \eta \frac{\partial f}{\partial x_2}$
- We saw that this would be a problem when there is a big difference between x_1 and x_2.
- The momentum relies on the exponentially weighted moving average to make the direction of the independent variable more consistent

Adagrad

- So far we showed that each variable uses the same learning rate
- For example, if we assume that the objective function is f and the independent variable is a two-dimensional vector $[x_1, x_2]$, the update for each element is $x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1} \qquad x_2 \leftarrow x_2 - \eta \frac{\partial f}{\partial x_2}$
- We saw that this would be a problem when there is a big difference between x_1 and x_2.
- The momentum relies on the exponentially weighted moving average to make the direction of the independent variable more consistent
- Now, we introduce Adagrad that adjusts the learning rate according to the gradient value of the independent variable in each direction

Adagrad

- Adagrad uses the cumulative variable $\mathbf{s}_t$ obtained from a square by element operation on the mini-batch stochastic gradient $\mathbf{g}_t$.

Adagrad

- Adagrad uses the cumulative variable $\mathbf{s}_t$ obtained from a square by element operation on the mini-batch stochastic gradient $\mathbf{g}_t$.
- At time step 0, $\mathbf{s}_0 = 0$.

Adagrad

- Adagrad uses the cumulative variable $\mathbf{s}_t$ obtained from a square by element operation on the mini-batch stochastic gradient $\mathbf{g}_t$.
- At time step 0, $\mathbf{s}_0 = 0$.
- At time step t, we first sum the results of the square by element operation for the mini-batch gradient $\mathbf{g}_t$ to get the variable $\mathbf{s}_t$:

$$\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t,$$

Adagrad

- Adagrad uses the cumulative variable $\mathbf{s}_t$ obtained from a square by element operation on the mini-batch stochastic gradient $\mathbf{g}_t$.
- At time step 0, $\mathbf{s}_0 = 0$.
- At time step t, we first sum the results of the square by element operation for the mini-batch gradient $\mathbf{g}_t$ to get the variable $\mathbf{s}_t$:

$$\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t,$$

- Next, we readjust the learning rate of each element in the independent variable of the objective function using element operations

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t,$$

Adagrad

- if an element in the independent variable of the objective function has a constant and large partial derivative, the learning rate of this element will drop faster.

Adagrad

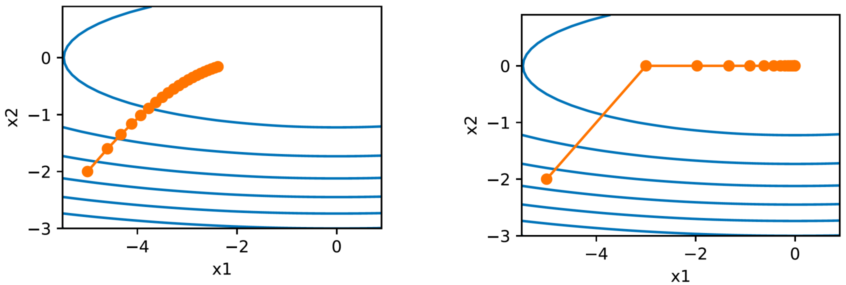
- if an element in the independent variable of the objective function has a constant and large partial derivative, the learning rate of this element will drop faster.
- On the contrary, if the partial derivative of such an element remains small, then its learning rate will decline more slowly.

Adagrad

- if an element in the independent variable of the objective function has a constant and large partial derivative, the learning rate of this element will drop faster.
- On the contrary, if the partial derivative of such an element remains small, then its learning rate will decline more slowly.
- when the learning rate declines very fast during early iteration, yet the current solution is still not desirable, Adagrad might have difficulty finding a useful solution because the learning rate will be too small at later stages of iteration.

Adagrad

- if an element in the independent variable of the objective function has a constant and large partial derivative, the learning rate of this element will drop faster.
- On the contrary, if the partial derivative of such an element remains small, then its learning rate will decline more slowly.
- when the learning rate declines very fast during early iteration, yet the current solution is still not desirable, Adagrad might have difficulty finding a useful solution because the learning rate will be too small at later stages of iteration.
- If we use Adagrad for the example $f(\mathbf{x}) = 0.1\, x_1^2 + 2x_2^2$, we get



Which one uses a larger learning rate?

RMSProp

- RMSProp made a small modification to Adagrad to prevent learning rate from declining very fast during early iteration

RMSProp

- RMSProp made a small modification to Adagrad to prevent learning rate from declining very fast during early iteration
- RMSProp uses the exponentially weighted moving average on the square by element results of these gradients.

RMSProp

- RMSProp made a small modification to Adagrad to prevent learning rate from declining very fast during early iteration
- RMSProp uses the exponentially weighted moving average on the square by element results of these gradients.
- Specifically, given the hyperparameter $0 \leq \gamma < 1$, RMSProp is computed at time step $t > 0$:

$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1 - \gamma)\mathbf{g}_t \odot \mathbf{g}_t.$$

RMSProp

- RMSProp made a small modification to Adagrad to prevent learning rate from declining very fast during early iteration
- RMSProp uses the exponentially weighted moving average on the square by element results of these gradients.
- Specifically, given the hyperparameter $0 \leq \gamma < 1$, RMSProp is computed at time step $t > 0$:

$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1 - \gamma)\mathbf{g}_t \odot \mathbf{g}_t.$$

- Like Adagrad, RMSProp readjust the learning rate of each element in the independent variable of the object function as:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t,$$

RMSProp

- If we expand the definition of $\mathbf{s}_t$, we see that:

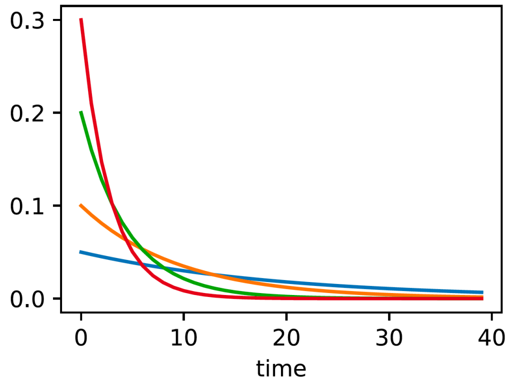
$$\begin{aligned}\mathbf{s}_t &= (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{s}_{t-1} \\ &= (1-\gamma)\left(\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{g}_{t-1} \odot \mathbf{g}_{t-1}\right) + \gamma^2 \mathbf{s}_{t-2} \\ &\cdots \\ &= (1-\gamma)\left(\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{g}_{t-1} \odot \mathbf{g}_{t-1} + \cdots + \gamma^{t-1}\mathbf{g}_1 \odot \mathbf{g}_1\right).\end{aligned}$$

RMSProp

- If we expand the definition of $\mathbf{s}_t$, we see that:

$$
\begin{aligned}
\mathbf{s}_t &= (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{s}_{t-1} \\
&= (1-\gamma)\left(\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{g}_{t-1} \odot \mathbf{g}_{t-1}\right) + \gamma^2 \mathbf{s}_{t-2} \\
&\cdots \\
&= (1-\gamma)\left(\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{g}_{t-1} \odot \mathbf{g}_{t-1} + \cdots + \gamma^{t-1} \mathbf{g}_1 \odot \mathbf{g}_1\right).
\end{aligned}
$$

- We visualize these weights in the past 40 time steps for various γ:



Adadelta

- In addition to RMSProp, Adadelta is another common optimization algorithm that helps improve the chances of finding useful solutions at later stages of iteration.

Adadelta

- In addition to RMSProp, Adadelta is another common optimization algorithm that helps improve the chances of finding useful solutions at later stages of iteration.
- The interesting thing is that there is no learning rate hyperparameter in the Adadelta algorithm.

Adadelta

- In addition to RMSProp, Adadelta is another common optimization algorithm that helps improve the chances of finding useful solutions at later stages of iteration.
- The interesting thing is that there is no learning rate hyperparameter in the Adadelta algorithm.
- Given the hyperparameter $0 \leq \rho < 1$, we compute $\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1 - \rho) \boldsymbol{g}_t \odot \boldsymbol{g}_t.$

Adadelta

- In addition to RMSProp, Adadelta is another common optimization algorithm that helps improve the chances of finding useful solutions at later stages of iteration.
- The interesting thing is that there is no learning rate hyperparameter in the Adadelta algorithm.
- Given the hyperparameter $0 \leq \rho < 1$, we compute $\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1-\rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$
- Unlike RMSprop, Adadelta maintains an additional state variable $\Delta\mathbf{x}_t$ to compute the variation of the independent variable

$$\boldsymbol{g}_t' \leftarrow \sqrt{\frac{\Delta\boldsymbol{x}_{t-1} + \epsilon}{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t,$$

Adadelta

- In addition to RMSProp, Adadelta is another common optimization algorithm that helps improve the chances of finding useful solutions at later stages of iteration.
- The interesting thing is that there is no learning rate hyperparameter in the Adadelta algorithm.
- Given the hyperparameter $0 \leq \rho < 1$, we compute $\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1-\rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$
- Unlike RMSprop, Adadelta maintains an additional state variable $\Delta\mathbf{x}_t$ to compute the variation of the independent variable

$$\boldsymbol{g}_t' \leftarrow \sqrt{\frac{\Delta\boldsymbol{x}_{t-1}+\epsilon}{\boldsymbol{s}_t+\epsilon}} \odot \boldsymbol{g}_t,$$

- Next we update the independent variable $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}_t'.$

Adadelta

- In addition to RMSProp, Adadelta is another common optimization algorithm that helps improve the chances of finding useful solutions at later stages of iteration.
- The interesting thing is that there is no learning rate hyperparameter in the Adadelta algorithm.
- Given the hyperparameter $0 \leq \rho < 1$, we compute $\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1-\rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$
- Unlike RMSprop, Adadelta maintains an additional state variable $\Delta\mathbf{x}_t$ to compute the variation of the independent variable

$$\boldsymbol{g}_t' \leftarrow \sqrt{\frac{\Delta\boldsymbol{x}_{t-1}+\epsilon}{\boldsymbol{s}_t+\epsilon}} \odot \boldsymbol{g}_t,$$

- Next we update the independent variable $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}_t'.$
- Finally, we use $\Delta\mathbf{x}_t$ to record EWMA on the squares of elements of $\boldsymbol{g}'$

$$\Delta\boldsymbol{x}_t \leftarrow \rho\Delta\boldsymbol{x}_{t-1} + (1-\rho)\boldsymbol{g}_t' \odot \boldsymbol{g}_t'.$$

Adam

- Created on the basis of RMSProp, Adam also uses EWMA on the mini-batch stochastic gradient

Adam

- Created on the basis of RMSProp, Adam also uses EWMA on the mini-batch stochastic gradient
- Adam uses a momentum variable $\boldsymbol{v}_t$ and variable $\boldsymbol{s}_t$, which is an EWMA on the squares of elements in the mini-batch SGD from RMSProp

$$\boldsymbol{v}_t \leftarrow \beta_1 \boldsymbol{v}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t. \qquad \boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$$

Adam

- Created on the basis of RMSProp, Adam also uses EWMA on the mini-batch stochastic gradient
- Adam uses a momentum variable $\boldsymbol{v}_t$ and variable $\boldsymbol{s}_t$, which is an EWMA on the squares of elements in the mini-batch SGD from RMSProp $\boldsymbol{v}_t \leftarrow \beta_1 \boldsymbol{v}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t.$ $\boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$
- Notice that $\boldsymbol{v}_t = \underbrace{(1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i}}_{1 - \beta_1^t} \boldsymbol{g}_i$

What happens when t is small?

Adam

- Created on the basis of RMSProp, Adam also uses EWMA on the mini-batch stochastic gradient
- Adam uses a momentum variable $\boldsymbol{v}_t$ and variable $\boldsymbol{s}_t$, which is an EWMA on the squares of elements in the mini-batch SGD from RMSProp

$$\boldsymbol{v}_t \leftarrow \beta_1 \boldsymbol{v}_{t-1} + (1-\beta_1)\boldsymbol{g}_t. \qquad \boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1-\beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$$

- Notice that $\boldsymbol{v}_t = \underbrace{(1-\beta_1)\sum_{i=1}^{t} \beta_1^{t-i}}_{1-\beta_1^t} \boldsymbol{g}_i$

What happens when t is small?

- When t is small, the sum of the mini-batch stochastic gradient weights from each previous time step will be small. To eliminate this effect, we perform bias correction:

$$\hat{\boldsymbol{v}}_t \leftarrow \frac{\boldsymbol{v}_t}{1-\beta_1^t}, \qquad \hat{\boldsymbol{s}}_t \leftarrow \frac{\boldsymbol{s}_t}{1-\beta_2^t}.$$

Adam

- Next, the Adam algorithm will use the bias-corrected variables to re-adjust the learning rate of each element.

$$\boldsymbol{g}_t' \leftarrow \frac{\eta \hat{\boldsymbol{v}}_t}{\sqrt{\hat{\boldsymbol{s}}_t} + \epsilon},$$

Adam

- Next, the Adam algorithm will use the bias-corrected variables to re-adjust the learning rate of each element.

$$\boldsymbol{g}_t' \leftarrow \frac{\eta \hat{\boldsymbol{v}}_t}{\sqrt{\hat{\boldsymbol{s}}_t} + \epsilon},$$

- Finally, we update the independent variable as

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}_t'.$$

Convolutional Neural Networks

- So far, we simply threw away the spatial structure in images by flattening each image into a 1D vector, and fed it into a fully-connected network

Convolutional Neural Networks

- So far, we simply threw away the spatial structure in images by flattening each image into a 1D vector, and fed it into a fully-connected network
- Ideally, we would find a way to leverage our prior knowledge that nearby pixels are more related to each other.

Convolutional Neural Networks

- So far, we simply threw away the spatial structure in images by flattening each image into a 1D vector, and fed it into a fully-connected network
- Ideally, we would find a way to leverage our prior knowledge that nearby pixels are more related to each other.
- Now, we introduce convolutional neural networks (CNNs), a powerful family of neural networks that were designed for precisely this purpose.

Convolutional Neural Networks

- So far, we simply threw away the spatial structure in images by flattening each image into a 1D vector, and fed it into a fully-connected network
- Ideally, we would find a way to leverage our prior knowledge that nearby pixels are more related to each other.
- Now, we introduce convolutional neural networks (CNNs), a powerful family of neural networks that were designed for precisely this purpose.
- In addition to their strong predictive performance,
  - convolutional neural networks tend to be computationally efficient,
  - both because they tend to require fewer fewer parameters than dense architectures
  - also because convolutions are easy to parallelize across GPU cores

From Dense Layers to Convolutions

- A few key principles for building neural networks for computer vision:
  1. **Translation Invariance**: Our vision systems should, in some sense, respond similarly to the same object regardless of where it appears in the image

From Dense Layers to Convolutions

- A few key principles for building neural networks for computer vision:
  1. **Translation Invariance**: Our vision systems should, in some sense, respond similarly to the same object regardless of where it appears in the image
  2. **Locality**: Our visions systems should, in some sense, focus on local regions, without regard for what else is happening in the image at greater distances.

Constraining the MLP

- Let's consider how MLP would look like with $h \times w$ images as inputs, and hidden representations as $h \times w$ matrices.

Constraining the MLP

- Let's consider how MLP would look like with $h \times w$ images as inputs, and hidden representations as $h \times w$ matrices.
- To have each of the hw hidden nodes receive input from each of the hw inputs, we would switch from using weight matrices to representing our parameters as four-dimensional weight tensors.

Constraining the MLP

- Let's consider how MLP would look like with $h \times w$ images as inputs, and hidden representations as $h \times w$ matrices.
- To have each of the hw hidden nodes receive input from each of the hw inputs, we would switch from using weight matrices to representing our parameters as four-dimensional weight tensors.
- We could formally express this dense layer as follows:

$$h[i,j] = u[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[k,l] = u[i,j] + \sum_{a,b} V[i,j,a,b] \cdot x[i+a, j+b]$$

Constraining the MLP

- Let's consider how MLP would look like with $h \times w$ images as inputs, and hidden representations as $h \times w$ matrices.
- To have each of the hw hidden nodes receive input from each of the hw inputs, we would switch from using weight matrices to representing our parameters as four-dimensional weight tensors.
- We could formally express this dense layer as follows:

$$h[i,j] = u[i,j] + \sum_{k,l} W[i,j,k,l] \cdot x[k,l] = u[i,j] + \sum_{a,b} V[i,j,a,b] \cdot x[i+a,j+b]$$

- For any given location (i,j) in the hidden layer $h[i,j]$, we compute its value by summing over pixels in x, centered around (i,j) and weighted by $V[i,j,a,b]$.

Constraining the MLP

- Let’s invoke the first principle- **translation invariance**.

Constraining the MLP

- Let’s invoke the first principle- **translation invariance**.
- This implies that a shift in the inputs x should simply lead to a shift in the activations h.

Constraining the MLP

- Let’s invoke the first principle- **translation invariance**.
- This implies that a shift in the inputs x should simply lead to a shift in the activations h.
- This is only possible if V and u don’t actually depend on (i, j).

Constraining the MLP

- Let’s invoke the first principle- **translation invariance**.
- This implies that a shift in the inputs x should simply lead to a shift in the activations h.
- This is only possible if V and u don’t actually depend on (i, j).
- In other words, we have $V[i, j, a, b] = V[a, b]$ and u is a constant.

Constraining the MLP

- Let’s invoke the first principle- **translation invariance**.
- This implies that a shift in the inputs x should simply lead to a shift in the activations h.
- This is only possible if V and u don’t actually depend on (i, j).
- In other words, we have $V[i, j, a, b] = V[a, b]$ and u is a constant.
- As a result we can simplify the definition for h:

$$h[i, j] = u + \sum_{a,b} V[a, b] \cdot x[i + a, j + b]$$

Constraining the MLP

- Let's invoke the first principle- **translation invariance**.
- This implies that a shift in the inputs x should simply lead to a shift in the activations h.
- This is only possible if V and u don't actually depend on (i, j).
- In other words, we have $V[i, j, a, b] = V[a, b]$ and u is a constant.
- As a result we can simplify the definition for h:

$$h[i, j] = u + \sum_{a,b} V[a, b] \cdot x[i + a, j + b]$$

- This is a convolution! We also reduced the number of parameters.

Constraining the MLP

- Now let's invoke the second principle – **locality**.

Constraining the MLP

- Now let's invoke the second principle – **locality**.
- We believe we should not have to look very far away from (i, j) in order to glean relevant information to assess what is going on at $h[i, j]$.

Constraining the MLP

- Now let's invoke the second principle – **locality**.
- We believe we should not have to look very far away from (i, j) in order to glean relevant information to assess what is going on at $h[i, j]$.
- This means that outside some range $|a|, |b| > \Delta$, we should set $V[a, b] = 0$.

Constraining the MLP

- Now let's invoke the second principle – **locality**.
- We believe we should not have to look very far away from (i, j) in order to glean relevant information to assess what is going on at $h[i, j]$.
- This means that outside some range $|a|, |b| > \Delta$, we should set $V[a, b] = 0$.
- Equivalently, we can write $h[i, j]$ as

$$h[i,j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a,b] \cdot x[i+a, j+b]$$

Constraining the MLP

- Now let's invoke the second principle – **locality**.
- We believe we should not have to look very far away from (i, j) in order to glean relevant information to assess what is going on at $h[i, j]$.
- This means that outside some range $|a|, |b| > \Delta$, we should set $V[a, b] = 0$.
- Equivalently, we can write $h[i, j]$ as

$$h[i, j] = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} V[a, b] \cdot x[i + a, j + b]$$

- This, in a nutshell is the convolutional layer.

Convolutions

- In mathematics, the convolution between two functions is defined as:

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$

Convolutions

- In mathematics, the convolution between two functions is defined as:

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$

- For discrete variables, the integral turns into a sum

$$[f \circledast g](i) = \sum f(a)g(i-a)$$

Convolutions

- In mathematics, the convolution between two functions is defined as:

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$

- For discrete variables, the integral turns into a sum

$$[f \circledast g](i) = \sum_a f(a)g(i-a)$$

- For two dimensional arrays, we have a corresponding sum with indices (i, j) for f and $(i-a, j-b)$ for g respectively.

Convolutions

- In mathematics, the convolution between two functions is defined as:

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$

- For discrete variables, the integral turns into a sum

$$[f \circledast g](i) = \sum_a f(a)g(i-a)$$

- For two dimensional arrays, we have a corresponding sum with indices (i, j) for f and $(i-a, j-b)$ for g respectively.
- This looks similar to the definition $h[i,j] = u + \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$

Convolutions

- In mathematics, the convolution between two functions is defined as:
$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$
- For discrete variables, the integral turns into a sum
$$[f \circledast g](i) = \sum_a f(a)g(i-a)$$
- For two dimensional arrays, we have a corresponding sum with indices (i, j) for f and $(i-a, j-b)$ for g respectively.
- This looks similar to the definition $h[i,j] = u + \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$
- In order to match the signs, we can use $\tilde{V} = V[-a, -b]$ to obtain $h = x \circledast \tilde{V}$

Convolutions

- In mathematics, the convolution between two functions is defined as:

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$

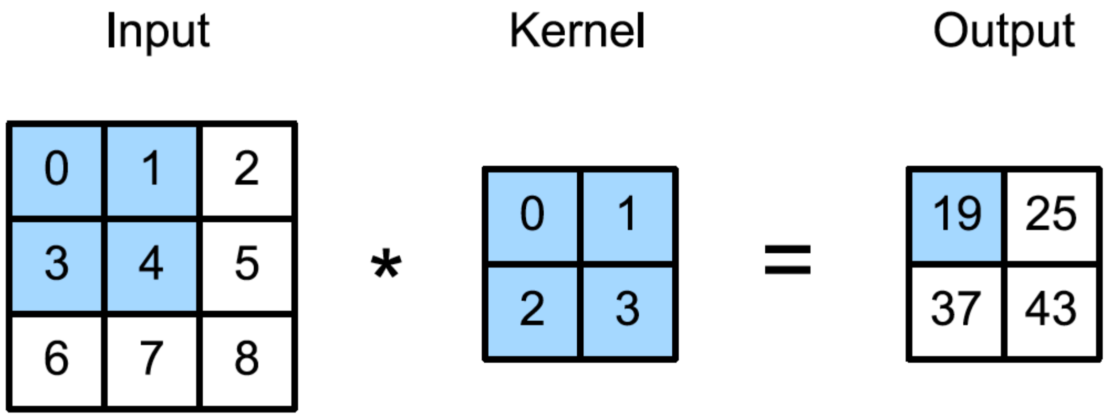
- For discrete variables, the integral turns into a sum

$$[f \circledast g](i) = \sum_a f(a)g(i-a)$$

- For two dimensional arrays, we have a corresponding sum with indices (i, j) for f and $(i-a, j-b)$ for g respectively.
- This looks similar to the definition $h[i,j] = u + \sum_{a,b} V[a,b] \cdot x[i+a, j+b]$
- In order to match the signs, we can use $\tilde{V} = V[-a, -b]$ to obtain $h = x \circledast \tilde{V}$
- Also note that the original definition is actually a *cross-correlation*.

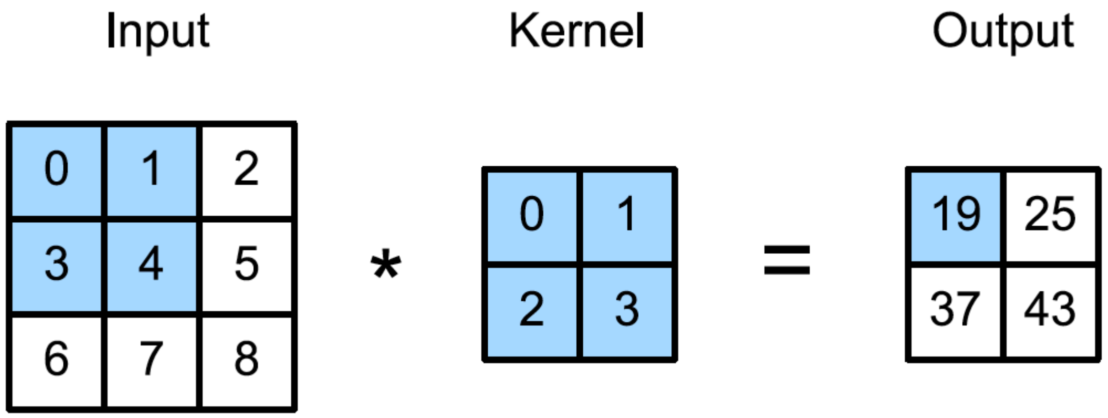
Convolutions for Images

- Let's see how convolutions work in practice.



Convolutions for Images

- Let’s see how convolutions work in practice.



- Start the notebook IN_CLASS_convolutions

Padding and Stride

- In general, assuming the input shape is (n_h, n_w) and the convolution kernel window shape is $(k_h,\ k_w)$, then the output shape will be:

Padding and Stride

- In general, assuming the input shape is (n_h, n_w) and the convolution kernel window shape is $(k_h,\ k_w)$, then the output shape will be:

$$(n_h - k_h + 1) \times (n_w - k_w + 1).$$

Padding and Stride

- In general, assuming the input shape is (n_h, n_w) and the convolution kernel window shape is $(k_h,\ k_w)$, then the output shape will be:

$$(n_h - k_h + 1) \times (n_w - k_w + 1).$$

- This might result in having much smaller images at the output of convolutional layer and we might lose any interesting information in the boundaries.

Padding and Stride

- In general, assuming the input shape is (n_h, n_w) and the convolution kernel window shape is $(k_h,\ k_w)$, then the output shape will be:

$$(n_h - k_h + 1) \times (n_w - k_w + 1).$$

- This might result in having much smaller images at the output of convolutional layer and we might lose any interesting information in the boundaries.
  - Solution: **Padding**

Padding and Stride

- In general, assuming the input shape is (n_h, n_w) and the convolution kernel window shape is $(k_h,\ k_w)$, then the output shape will be:

$$(n_h - k_h + 1) \times (n_w - k_w + 1).$$

- This might result in having much smaller images at the output of convolutional layer and we might lose any interesting informarion in the boundaries.
  - Solution: **Padding**
- In some cases, we want to reduce the resolution drastically

Padding and Stride

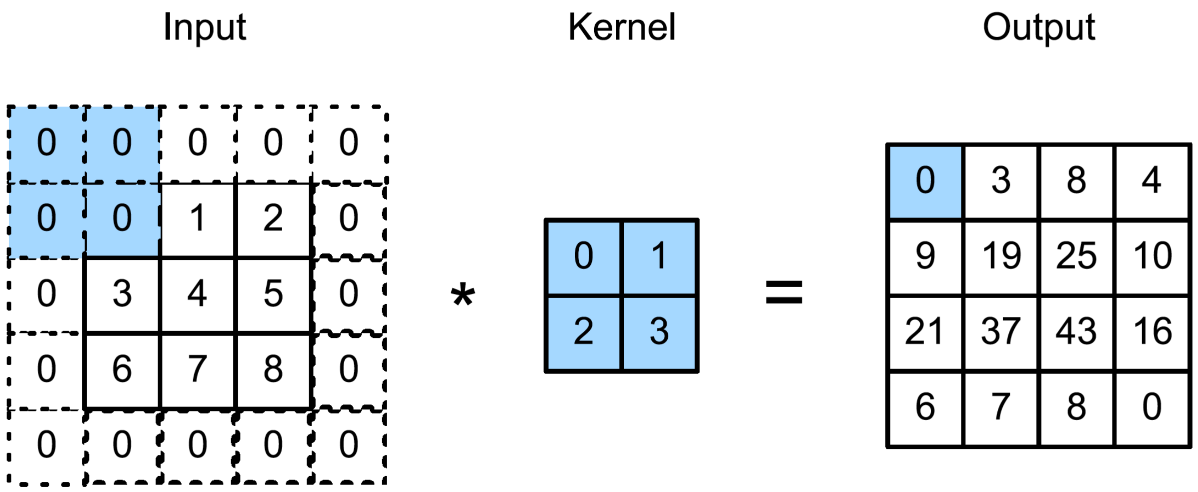
- In general, assuming the input shape is (n_h, n_w) and the convolution kernel window shape is $(k_h,\ k_w)$, then the output shape will be:

$$(n_h - k_h + 1) \times (n_w - k_w + 1).$$

- This might result in having much smaller images at the output of convolutional layer and we might lose any interesting informarion in the boundaries.
  - Solution: **Padding**
- In some cases, we want to reduce the resolution drastically
  - Solution: **Strides**

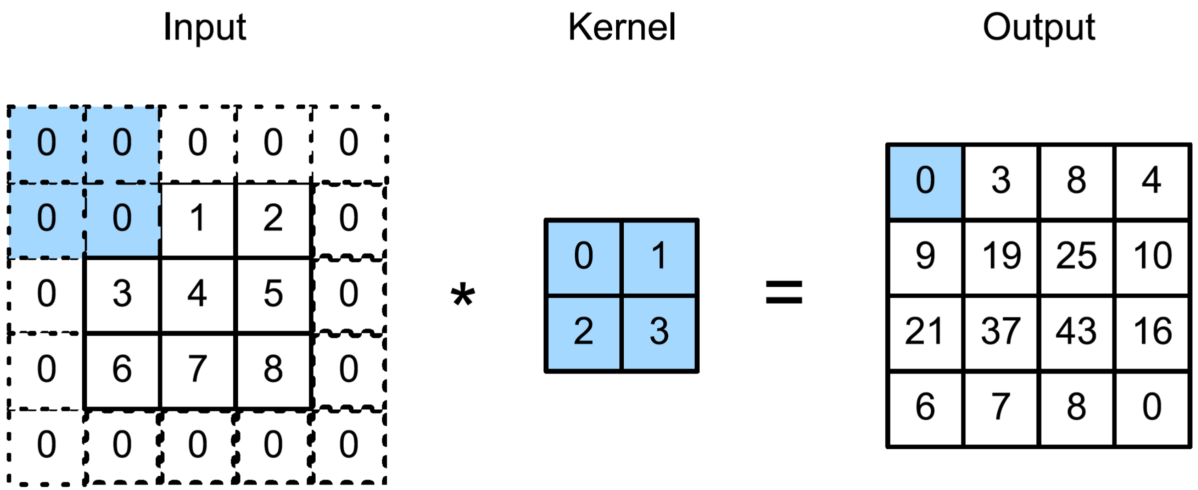
Padding

- Adding extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image.



Padding

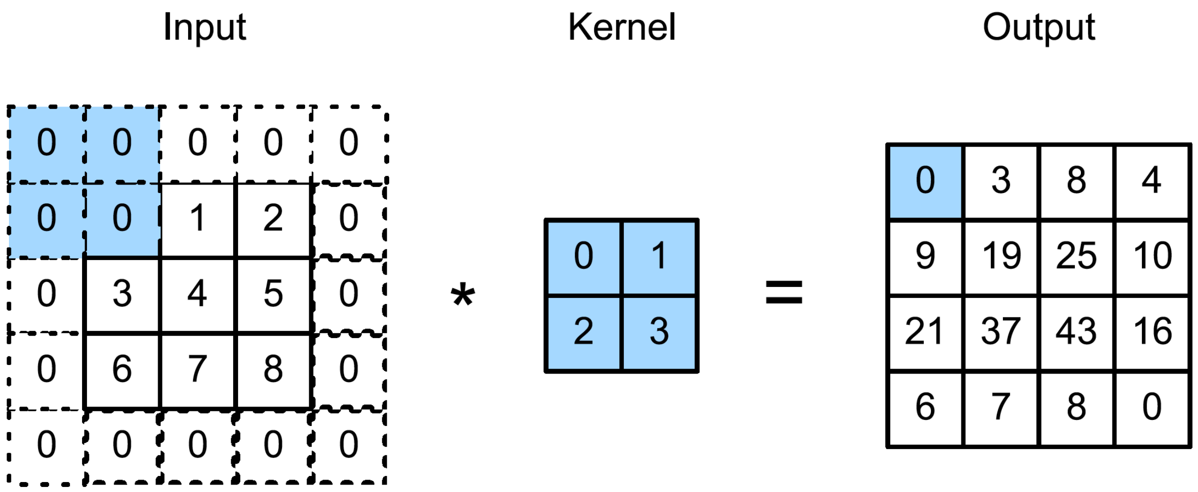
- Adding extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image.



- What is the size of the output after padding?

Padding

- Adding extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image.



- What is the size of the output after padding?
- What do you think is a good number of padding?

Stride

- When computing the convolution, we start with the convolution window at the top-left corner of the input array, and then slide it over all locations both down and to the right.

Stride

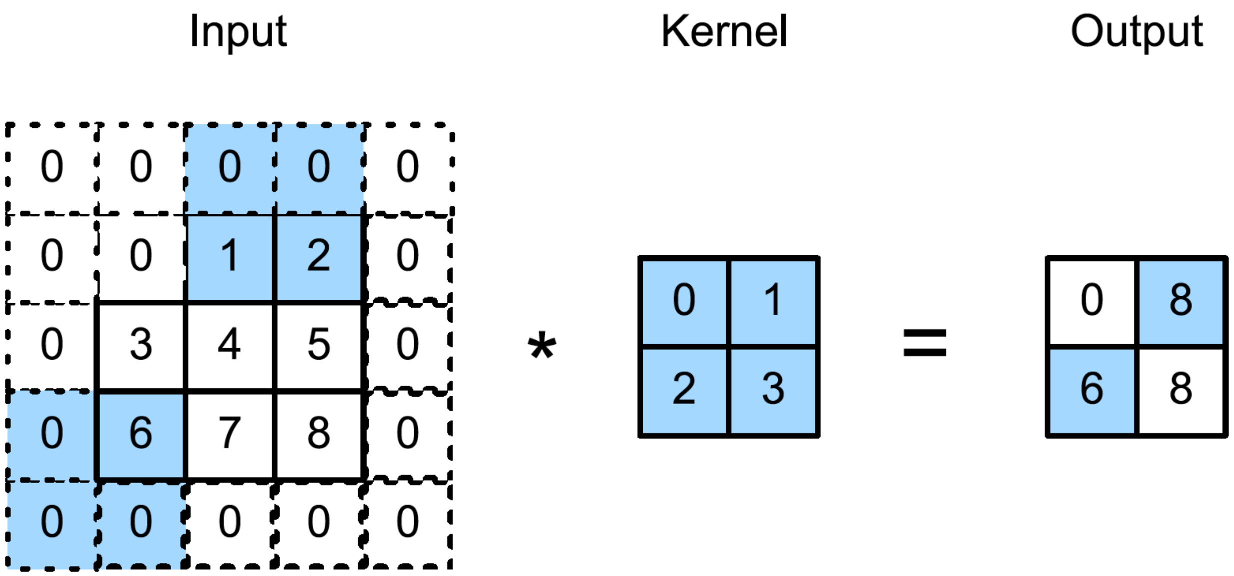
- When computing the convolution, we start with the convolution window at the top-left corner of the input array, and then slide it over all locations both down and to the right.
- In previous examples, we default to sliding one pixel at a time.

Stride

- When computing the convolution, we start with the convolution window at the top-left corner of the input array, and then slide it over all locations both down and to the right.
- In previous examples, we default to sliding one pixel at a time.
- However, sometimes, we move our window more than one pixel at a time, skipping the intermediate locations.

Stride

- When computing the convolution, we start with the convolution window at the top-left corner of the input array, and then slide it over all locations both down and to the right.
- In previous examples, we default to sliding one pixel at a time.
- However, sometimes, we move our window more than one pixel at a time, skipping the intermediate locations.
- We refer to the number of rows and columns traversed per slide as the *stride*.



Example for strides with 3 and 2 for height and width, respectively.

What is the size of the output after proper amount of padding?