10-701: Introduction to Machine Learning Lecture 11 - Convolutional Neural Networks

Henry Chai

2/23/24

Front Matter

- Announcements
  - HW3 released 2/19, due 2/28 at 11:59 PM
  - Project details will be released 3/1 (next Friday)
    - **You must work in groups of 2 or 3 on the project**
- Recommended Readings
  - Zhang, Lipton, Li & Smola, Chapter 7

Recall: Stochastic Gradient Descent for Learning

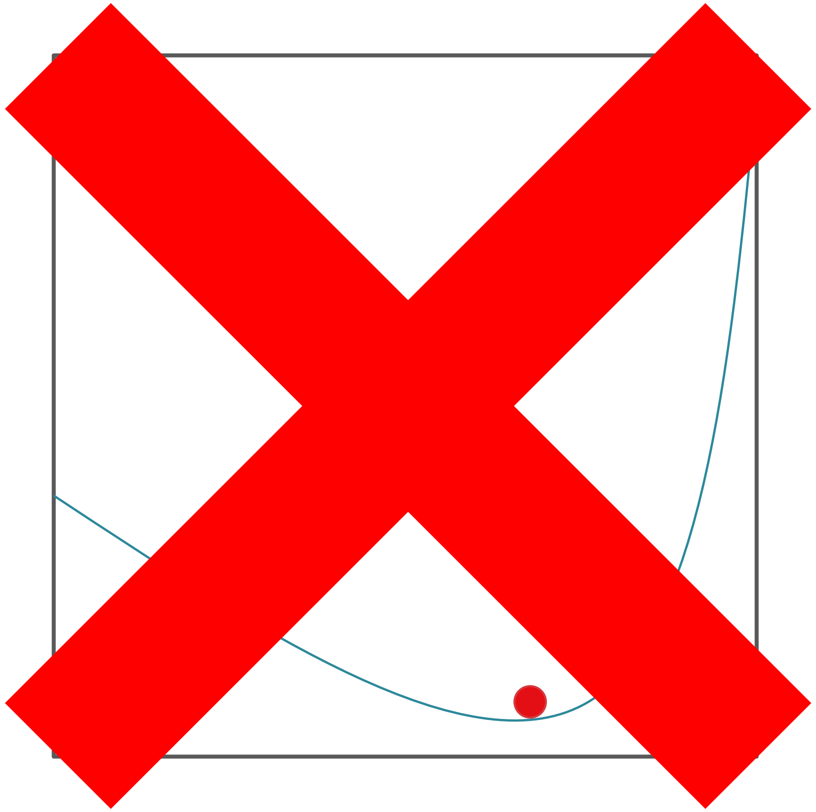
```
- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
- While TERMINATION CRITERION is not satisfied
    - For $i \in \text{shuffle}(\{1, \ldots, N\})$
        - For $l = 1, \ldots, L$
            - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$
            - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
        - Increment $t$: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$
```

Back-propagation

- Input: $W^{(1)}, \ldots, W^{(L)}$ and $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$
- Run forward propagation with $\boldsymbol{x}^{(i)}$ to get $\boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(L)}$
- (Optional) Compute $\ell^{(i)} = \left(o^{(L)} - y^{(i)}\right)^2$
- Initialize: $\boldsymbol{\delta}^{(L)} = 2\left(o_1^{(L)} - y^{(i)}\right)$
- For $l = L - 1, \ldots, 1$
  - Compute $\boldsymbol{\delta}^{(l)} = W^{(l+1)^T} \boldsymbol{\delta}^{(l+1)} \odot \left(1 - \boldsymbol{o}^{(l)} \odot \boldsymbol{o}^{(l)}\right)$
  - Compute $G^{(l)} = \boldsymbol{\delta}^{(l)} \boldsymbol{o}^{(l-1)^T}$
- Output: $G^{(1)}, \ldots, G^{(L)}$, the gradients of $\ell^{(i)}$ w.r.t $W^{(1)}, \ldots, W^{(L)}$

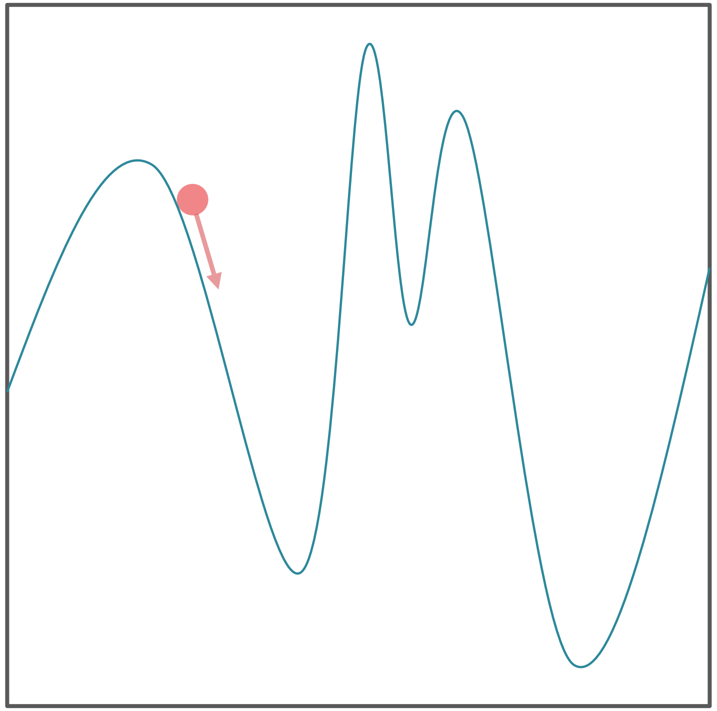
Recall: Gradient Descent

- Iterative method for minimizing functions
- Requires the gradient to exist everywhere



Non-convexity

- Gradient descent is not guaranteed to find a global minimum on non-convex surfaces



Stochastic Gradient Descent for Learning

```
Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta^{(0)}$

Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$

While TERMINATION CRITERION is not satisfied
    For $i \in \text{shuffle}(\{1, \dots, N\})$
        For $l = 1, \dots, L$
            Compute $G^{(l)} = \nabla_{W^{(l)}} \ell^{(i)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$
            Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
        Increment $t$: $t = t + 1$

Output: $W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}$
```

Mini-batch Stochastic Gradient Descent for Learning

```
• Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$
1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
    a. Randomly sample $B$ data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$
    b. Compute the gradient w.r.t. the sampled *batch*,
           $G^{(l)} = \frac{1}{B} \sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) \forall\, l$
    c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_{t}^{(l)} - \eta_{MB}^{(0)} G^{(l)} \forall\, l$
    d. Increment $t$: $t \leftarrow t + 1$
• Output: $W_{t}^{(1)}, \dots, W_{t}^{(L)}$
```

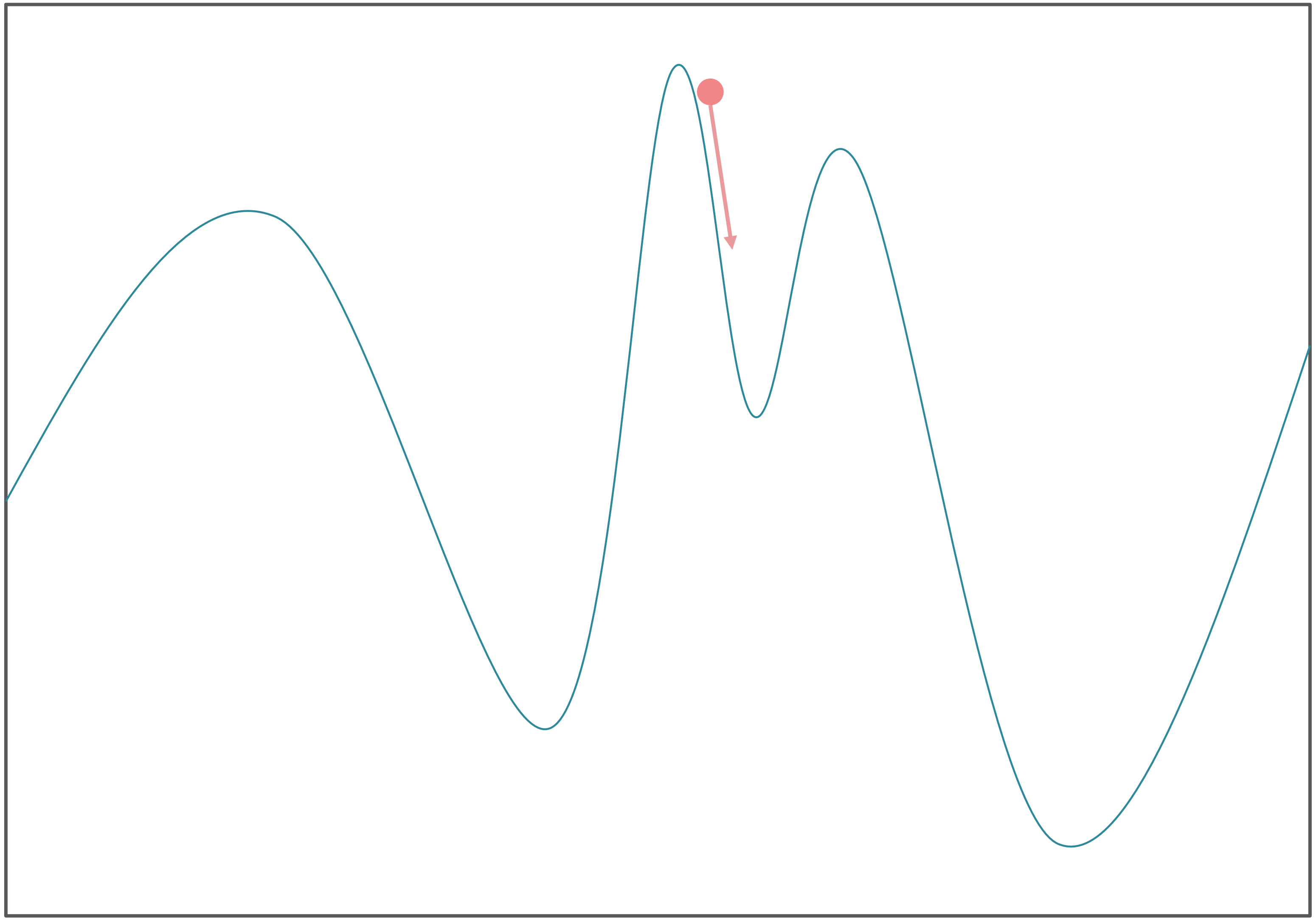
Mini-batch Stochastic Gradient Descent with Momentum for Learning

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B,$ decay parameter β

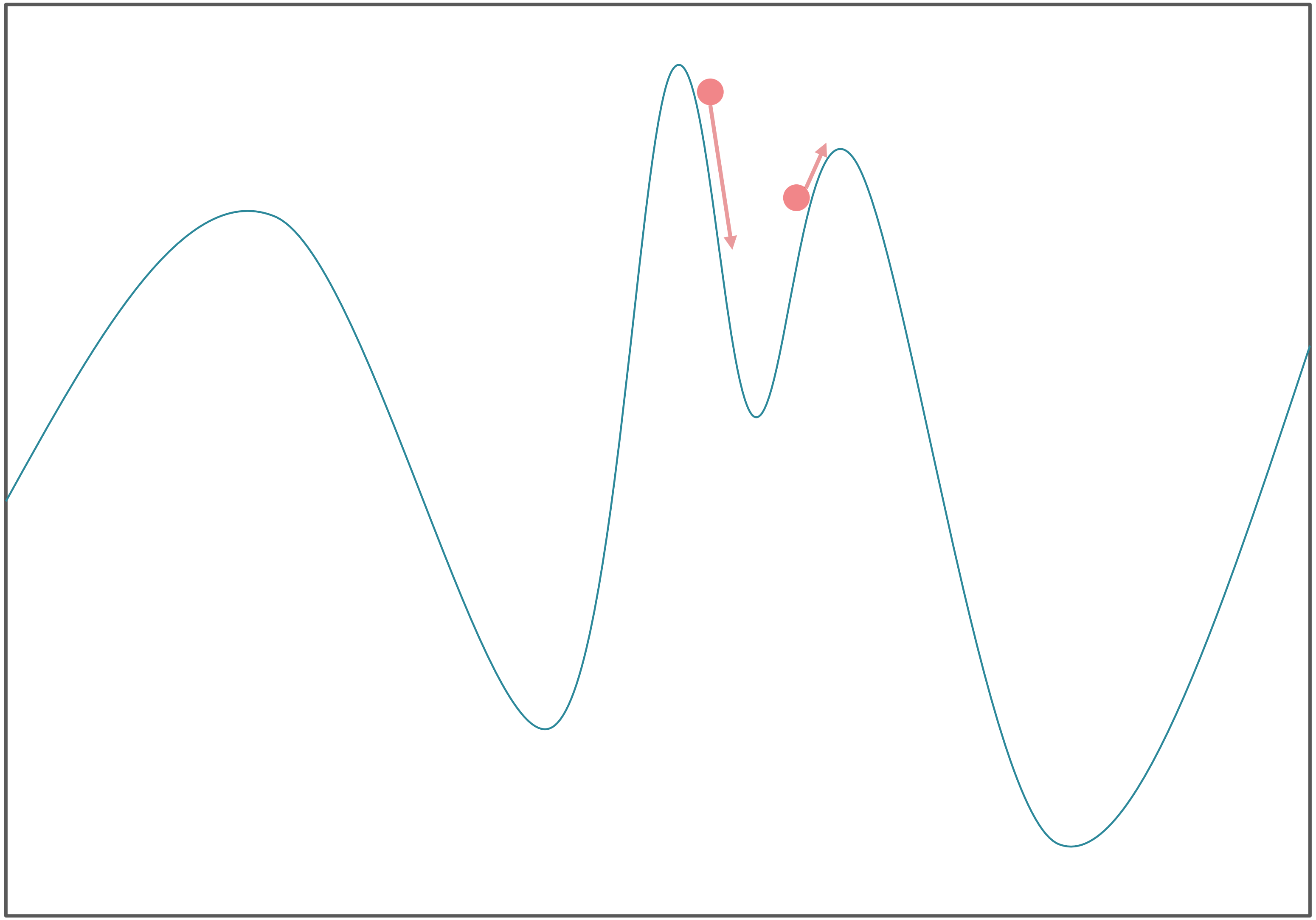
1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$, $G_{-1}^{(l)} = 0 \odot W^{(l)} \, \forall \, l = 1, \ldots, L$
2. While TERMINATION CRITERION is not satisfied
    a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
    b. Compute the gradient w.r.t. the sampled *batch*,
    $$G_t^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right) \forall \, l$$
    c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta_{MB}^{(0)}\left(\beta G_{t-1}^{(l)} + G_t^{(l)}\right) \forall \, l$
    d. Increment t: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \ldots, W_t^{(L)}$

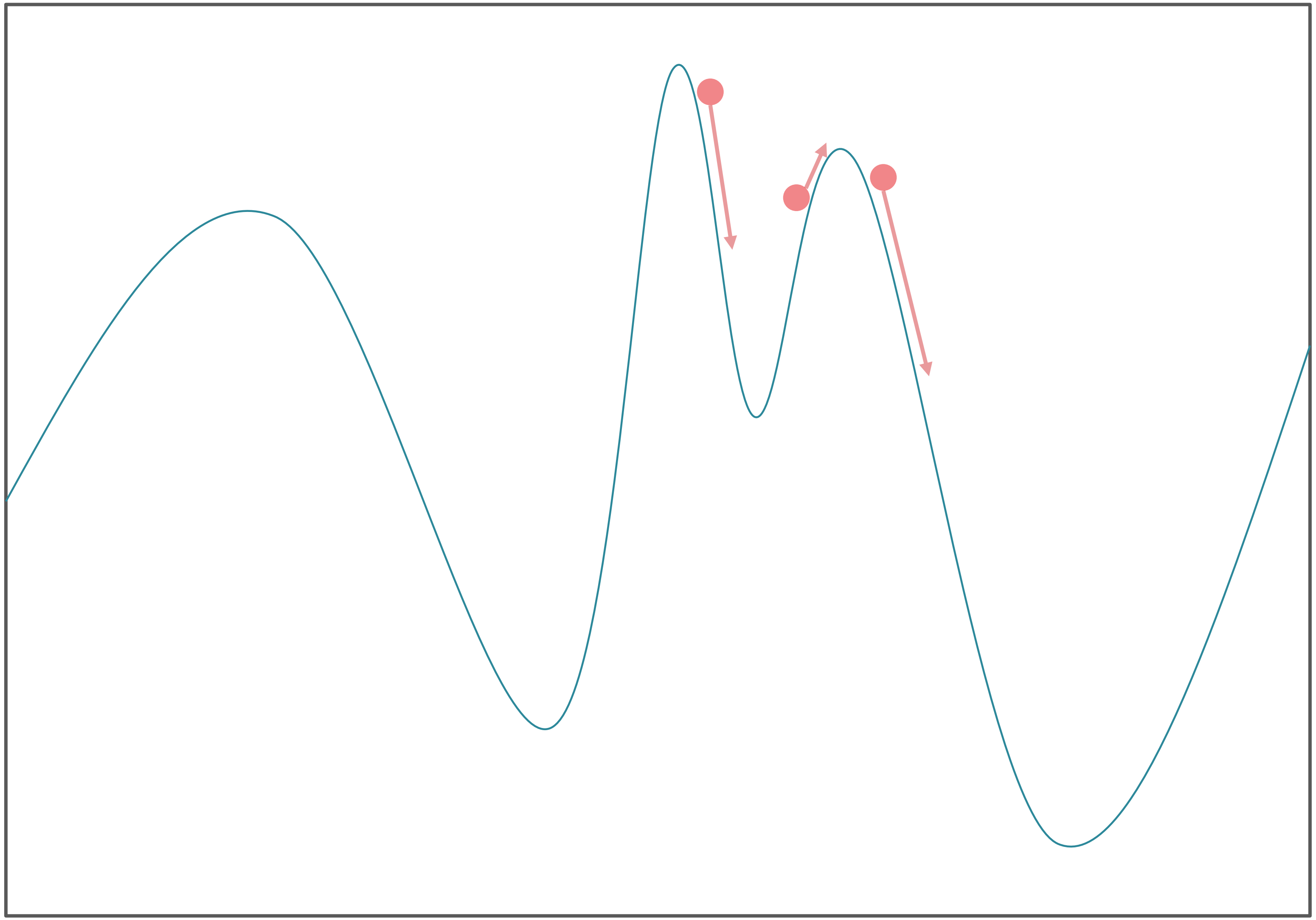
Mini-batch Stochastic Gradient Descent with Momentum for Learning



Mini-batch Stochastic Gradient Descent with Momentum for Learning



Mini-batch Stochastic Gradient Descent with Momentum for Learning



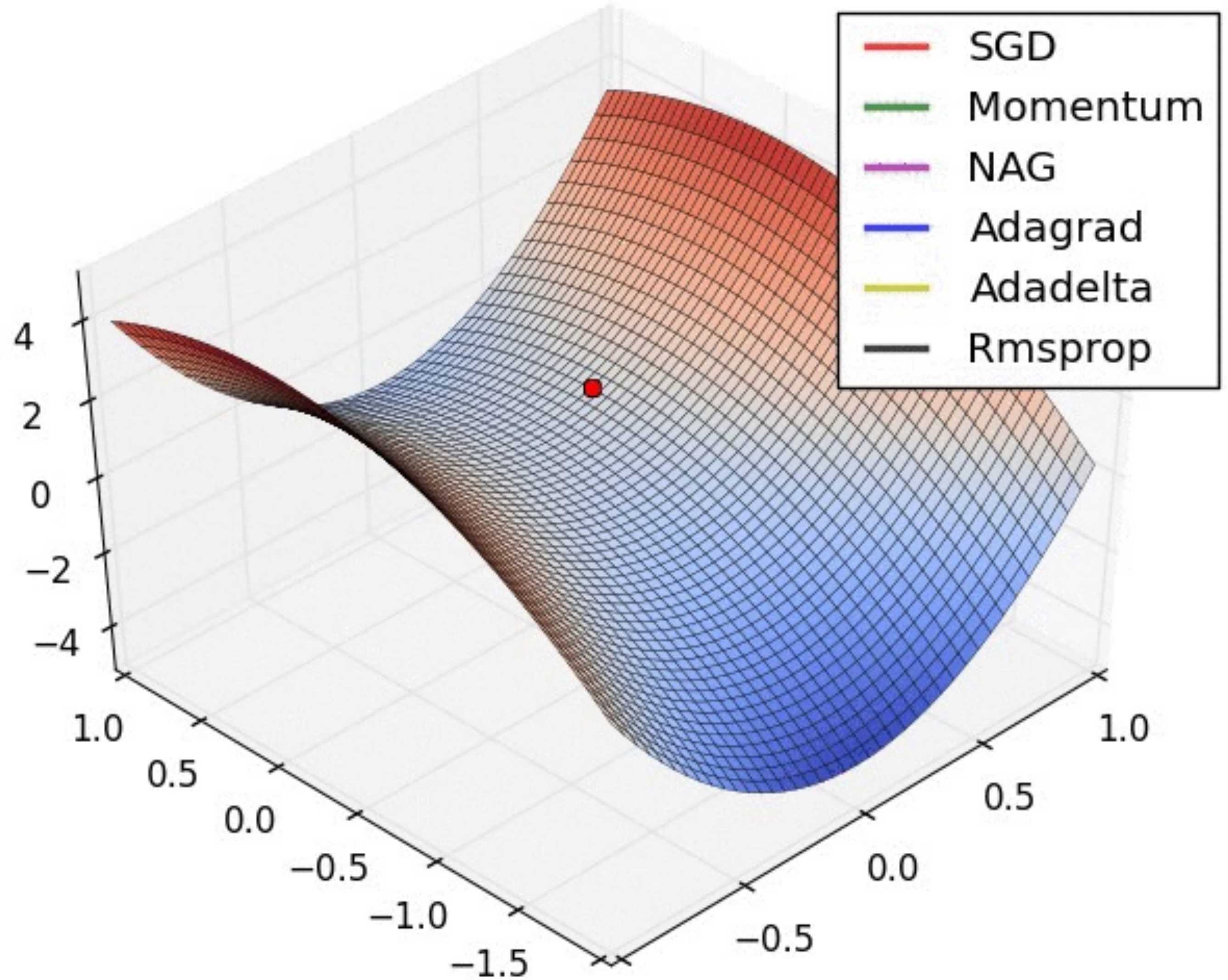
Mini-batch Stochastic Gradient Descent with Root Mean Square Propagation (RMSProp)

```
• Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$, decay parameter $\beta$

1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random
   numbers and set $t = 0$, $S_{-1}^{(l)} = 0 \odot W^{(l)} \ \forall\, l = 1, \dots, L$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample $B$ data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$
   b. Compute the gradient w.r.t. the sampled *batch*,
          $$G_t^{(l)} = \frac{1}{B} \sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)} \left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) \forall\, l$$
   c. Update the scaling factor: $S_t = \beta S_{t-1} + (1 - \beta)(G_t \odot G_t)$
   d. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\gamma}{\sqrt{S_t}} \odot G_t$
   e. Increment $t$: $t \leftarrow t + 1$

• Output: $W_t^{(1)}, \dots, W_t^{(L)}$
```

Mini-batch Stochastic Gradient Descent with Root Mean Square Propagation (RMSProp)



Source: https://www.ruder.io/optimizing-gradient-descent/

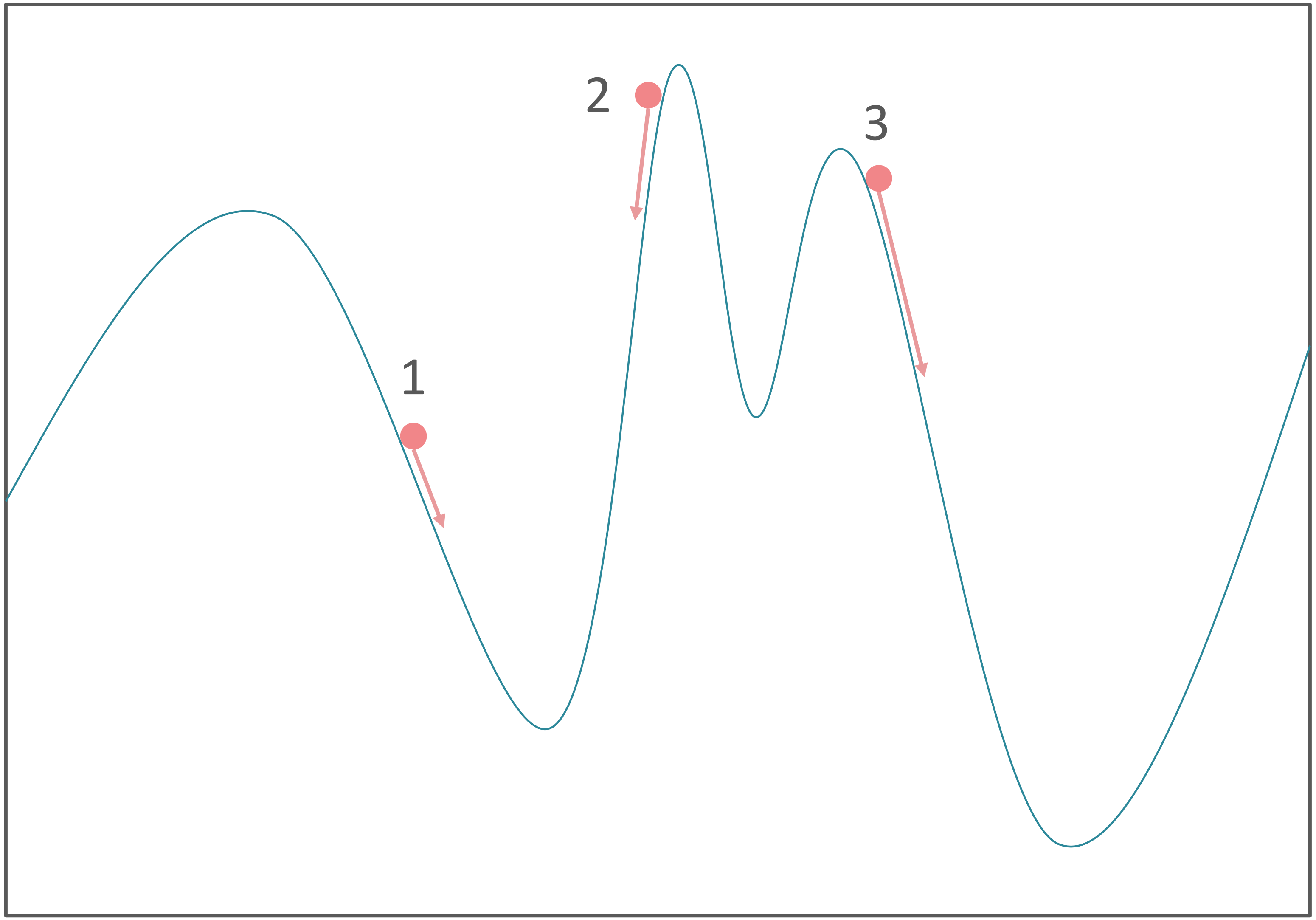
Adam (Adaptive Moment Estimation) = SGD + Momentum + RMSProp

```
• Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta_{MB}^{(0)}, B,$ decay parameters $\beta_1$ and $\beta_2$
1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0, M_{-1} = S_{-1} = 0 \odot W^{(l)} \; \forall\, l = 1, \dots, L$
2. While TERMINATION CRITERION is not satisfied
    a. Randomly sample $B$ data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$
    b. Compute the gradient ($G_t$), momentum and scaling factor
           $M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t$
           $S_t = \beta_2 S_{t-1} + (1 - \beta_2)(G_t \odot G_t)$
    c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \frac{\gamma}{\sqrt{S_t/(1-\beta_2^t)}} \odot (M_t/(1 - \beta_1^t))$
    d. Increment $t$: $t \leftarrow t + 1$
• Output: $W_t^{(1)}, \dots, W_t^{(L)}$
```

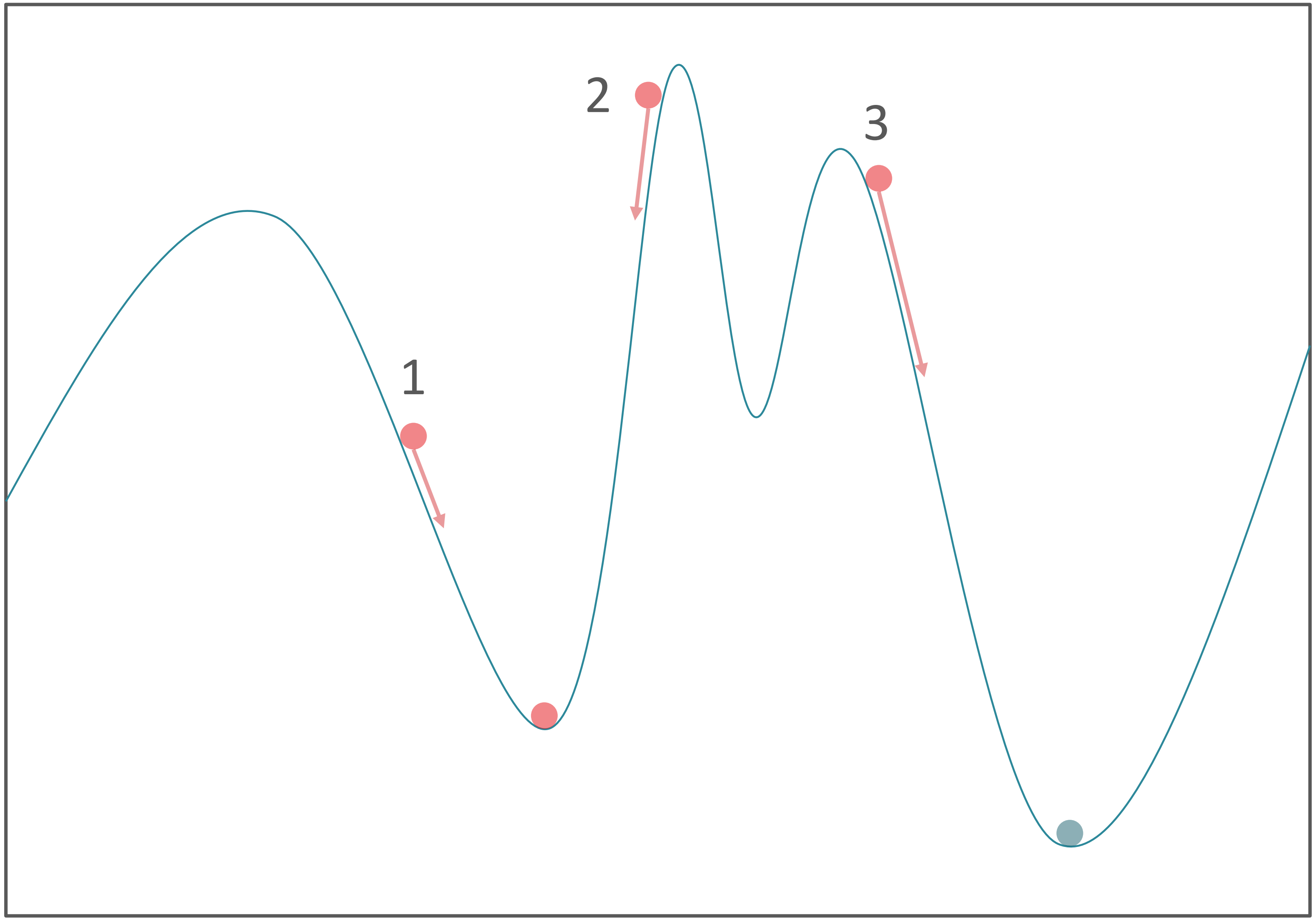
Random Restarts

- Run mini-batch gradient descent (with momentum & adaptive gradients) multiple times, each time starting with a ***different***, ***random*** initialization for the weights.
- Compute the training error of each run at termination and return the set of weights that achieves the lowest training error.

Random Restarts

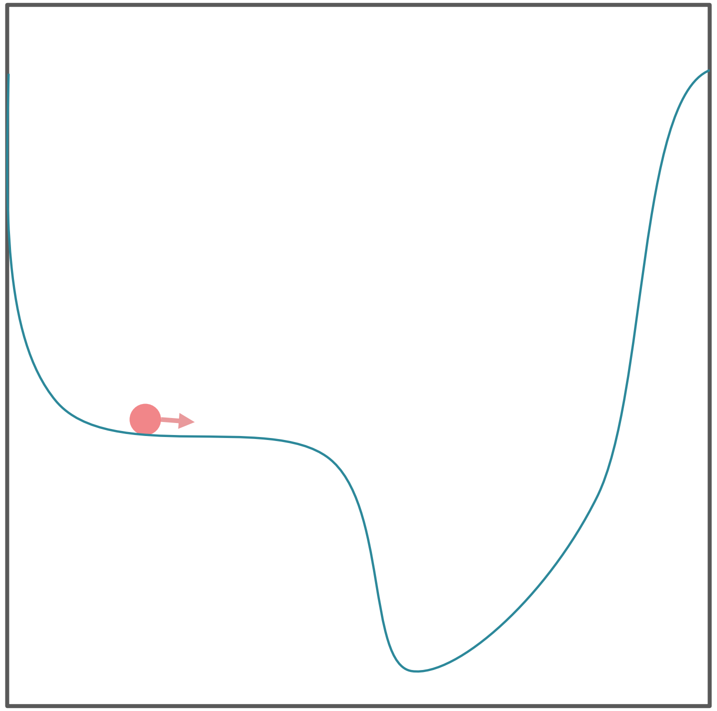


Random Restarts



Terminating Gradient Descent

- For non-convex surfaces, the gradient’s magnitude is often not a good metric for proximity to a minimum



Terminating Gradient Descent "Early"

- For non-convex surfaces, the gradient's magnitude is often not a good metric for proximity to a minimum
- Combine multiple termination criteria e.g. only stop if enough iterations have passed and the improvement in error is small
- Alternatively, terminate early by using a validation data set: if the validation error starts to increase, just stop!
    - Early stopping asks like regularization by **limiting how much of the hypothesis set** is explored

Neural Networks and Regularization

- Minimize $\ell_{AUG}^{(i)}\left(W^{(1)}, \dots, W^{(L)}, \lambda_C\right)$

$$= \ell^{(i)}\left(W^{(1)}, \dots, W^{(L)}\right) + \lambda_C r\left(W^{(1)}, \dots, W^{(L)}\right)$$

e.g. L2 regularization

$$r\left(W^{(1)}, \dots, W^{(L)}\right) = \sum_{l=1}^{L} \sum_{i=0}^{d^{(l-1)}} \sum_{j=1}^{d^{(l)}} \left(w_{j,i}^{(l)}\right)^2$$

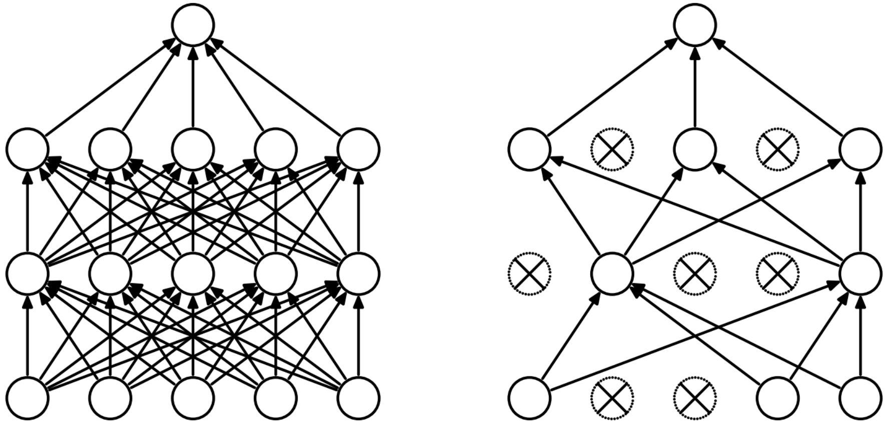
Neural Networks and "Strange" Regularization (Bishop, 1995)

- Jitter
  - In each iteration of gradient descent, add some random noise or "jitter" to each training data point
    - Instead of computing the gradient w.r.t. $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)$, use $\left(\boldsymbol{x}^{(i)} + \boldsymbol{\epsilon}, y^{(i)}\right)$ where $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma^2 I)$
  - Makes neural networks resilient to input noise
  - Has been proven to be equivalent to using a certain kind of regularizer r for some error metrics

Source: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-tikhonov-nc-95.pdf

Neural Networks and "Strange" Regularization (Srivastava et al., 2014)

- Dropout
  - In each iteration of gradient descent, randomly remove some of the nodes in the network
  - Compute the gradient using only the remaining nodes
  - The weights on edges going into and out of "dropped out" nodes are not updated

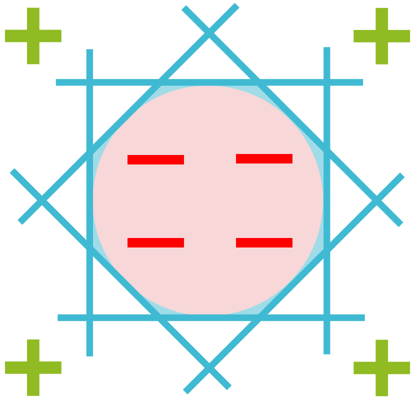


(a) Standard Neural Net (b) After applying dropout.

Source: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

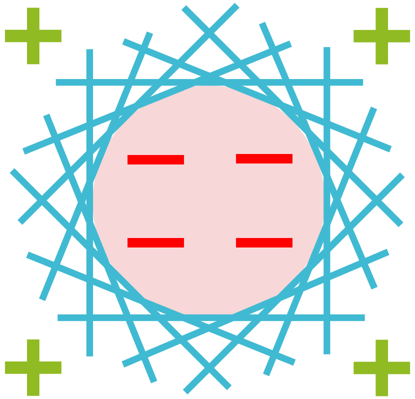
MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



MLPs as Universal Approximators

- Theorem: any function that can be decomposed into perceptrons can be modelled exactly using a 3-layer MLP
- Any smooth decision boundary can be approximated to an arbitrary precision using a finite number of perceptrons



- Theorem: Any smooth decision boundary can be approximated to an arbitrary precision using a 3-layer MLP

NNs as Universal Approximators (Cybenko, 1989 & Hornik, 1991)

- Theorem: Any bounded, continuous function can be approximated to an arbitrary precision using a 2-layer (1 hidden layer) feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.

Source: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F21A09B7475DFB9487990020839A39D2?doi=10.1.1.441.7873&rep=rep1&type=pdf
https://doi.org/10.1016/0893-6080%2891%2990009-T

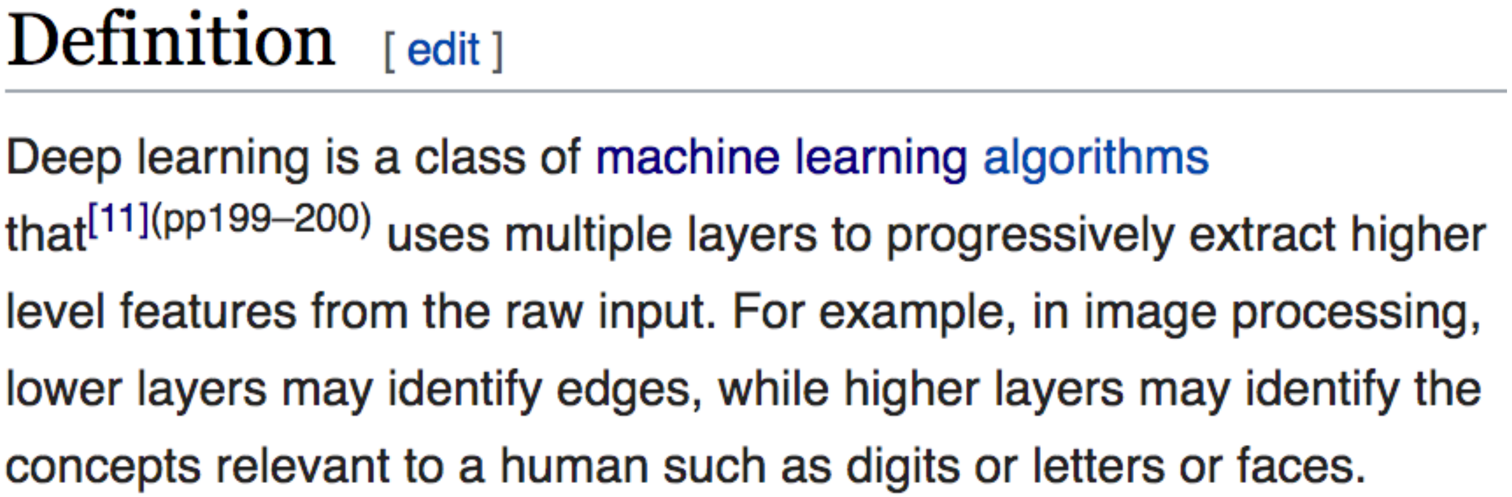
NNs as Universal Approximators (Cybenko, 1988)

- Theorem: Any function can be approximated to an arbitrary precision using a 3-layer (2 hidden layers) feed-forward NN if the activation function, θ, is continuous, bounded and non-constant.

Source: G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Dept. of Computer Science, Tufts University, Medford, MA, 1988.

Deep Learning

- From Wikipedia's page on Deep Learning…



Definition [edit]

Deep learning is a class of machine learning algorithms that[11](pp199–200) uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

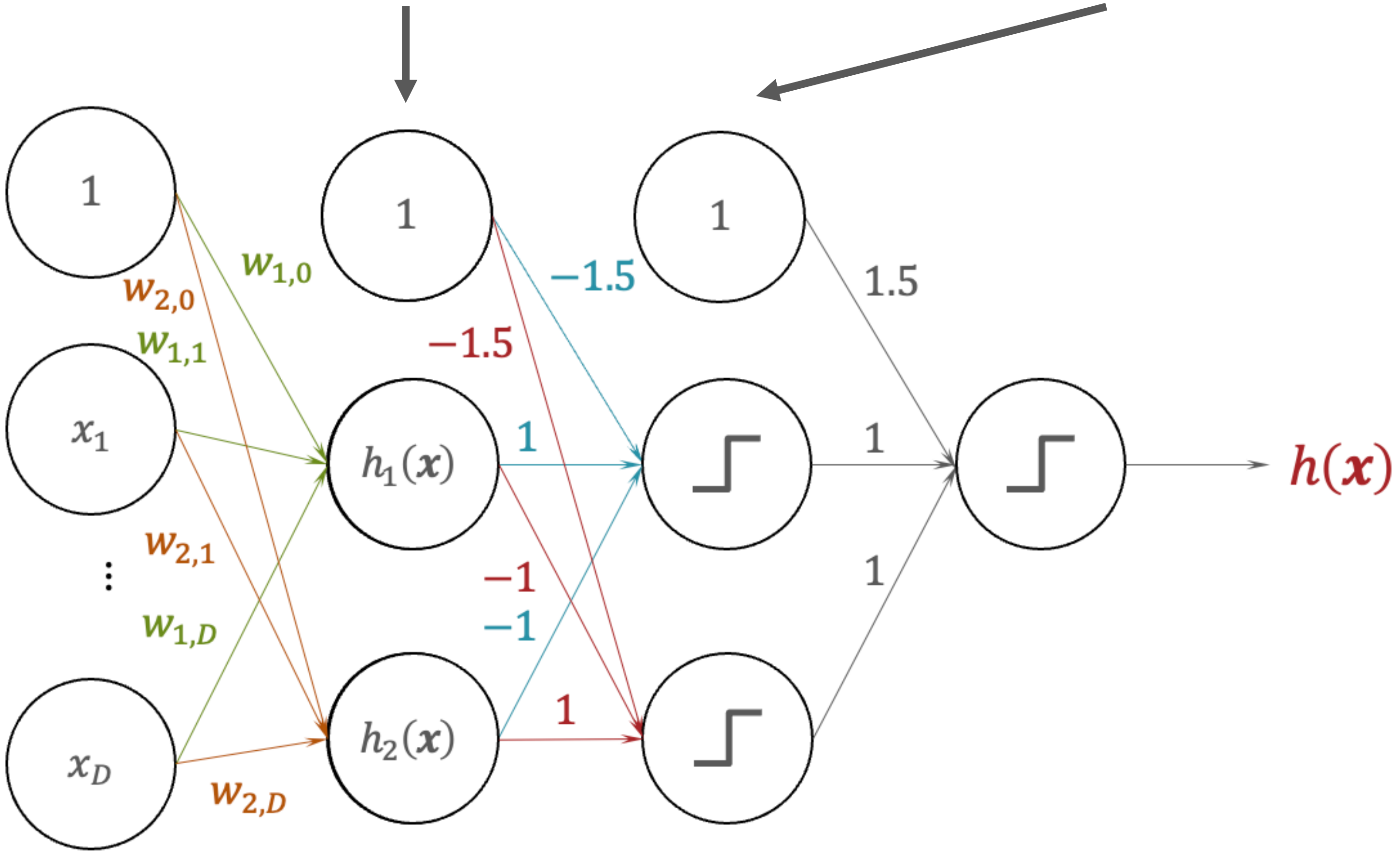
- Deep learning = more than one layer

Source: https://en.wikipedia.org/wiki/Deep_learning

Deep Learning

First layer: computes the perceptrons' predictions

Second layer: combines lower-level components



Convolutional Neural Networks

- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Idea: use the first few layers to identify relevant macro-features, e.g., edges
- Insight: for spatially-structured inputs, many useful macro-features are shift or location-invariant, e.g., an edge in the upper left corner of a picture looks like an edge in the center
- Strategy: learn a *filter* for macro-feature detection in a small window and apply it over the entire image

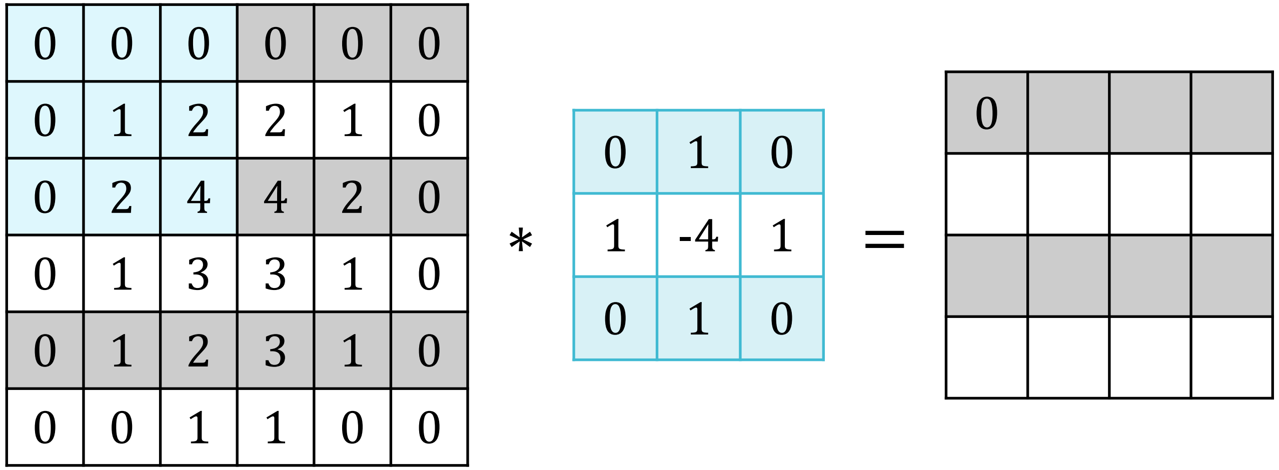
Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 0 \\ \hline 0 & 2 & 4 & 4 & 2 & 0 \\ \hline 0 & 1 & 3 & 3 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

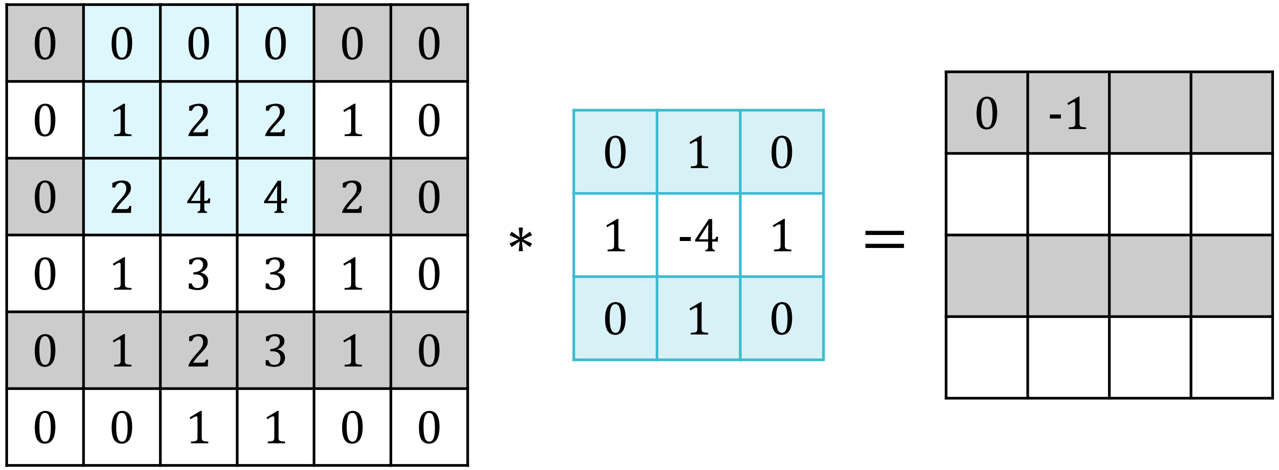


$$\begin{bmatrix}0&0&0&0&0&0\\0&1&2&2&1&0\\0&2&4&4&2&0\\0&1&3&3&1&0\\0&1&2&3&1&0\\0&0&1&1&0&0\end{bmatrix} * \begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix} = \begin{bmatrix}0&&&\\&&&\\&&&\\&&&\end{bmatrix}$$

$$(0 * 0) + (0 * 1) + (0 * 0) + (0 * 1) + (1 * -4)$$
$$+ (2 * 1) + (0 * 0) + (2 * 1) + (4 * 0) = 0$$

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix



$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

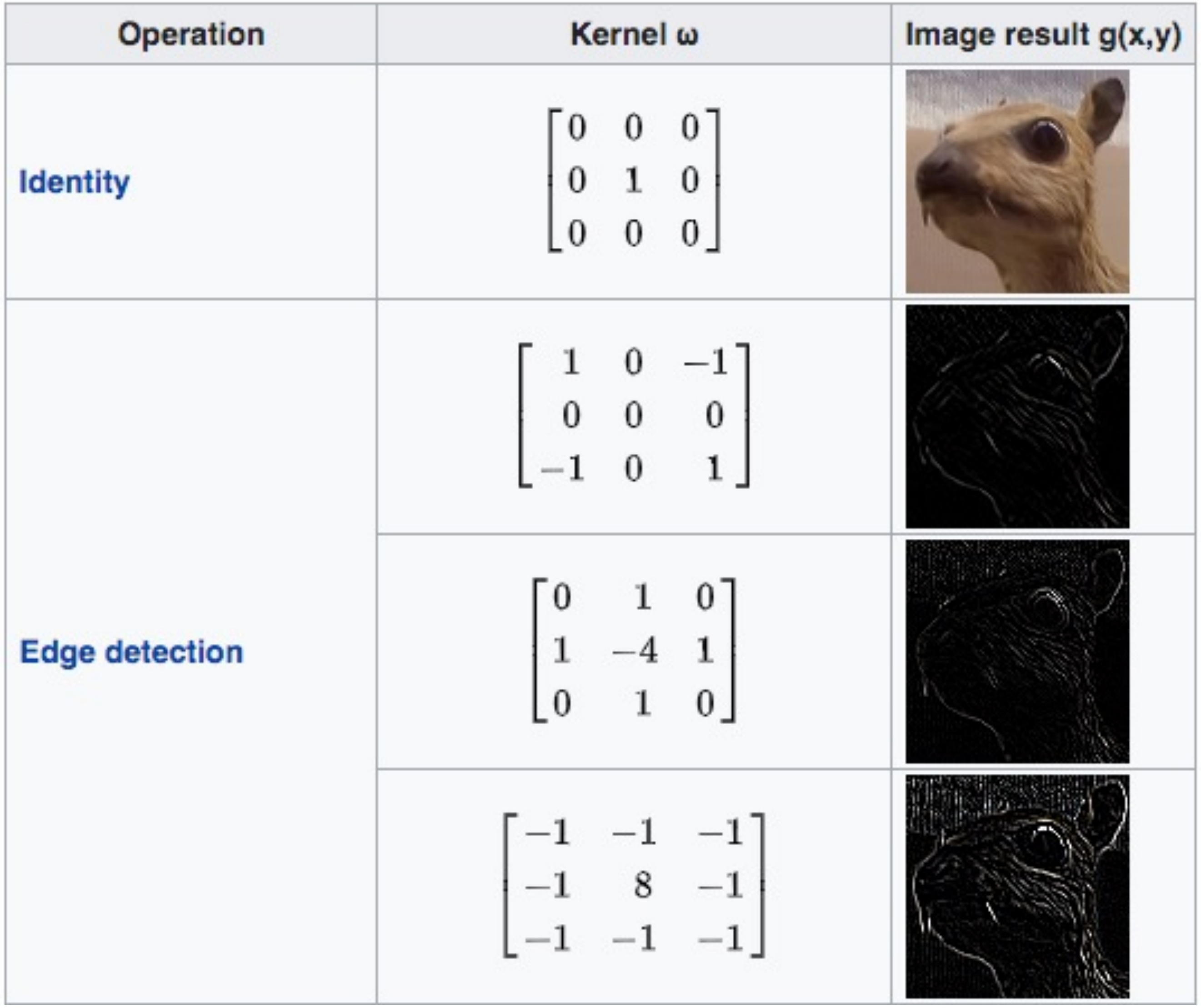
$$(0 * 0) + (0 * 1) + (0 * 0) + (1 * 1) + (2 * -4) + (2 * 1) + (2 * 0) + (4 * 1) + (4 * 0) = -1$$

Convolutional Filters

- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 0 \\ \hline 0 & 2 & 4 & 4 & 2 & 0 \\ \hline 0 & 1 & 3 & 3 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 0 & -1 & -1 & 0 \\ \hline -2 & -5 & -5 & -2 \\ \hline 2 & -2 & -1 & 3 \\ \hline -1 & 0 & -5 & 0 \\ \hline \end{array}$$

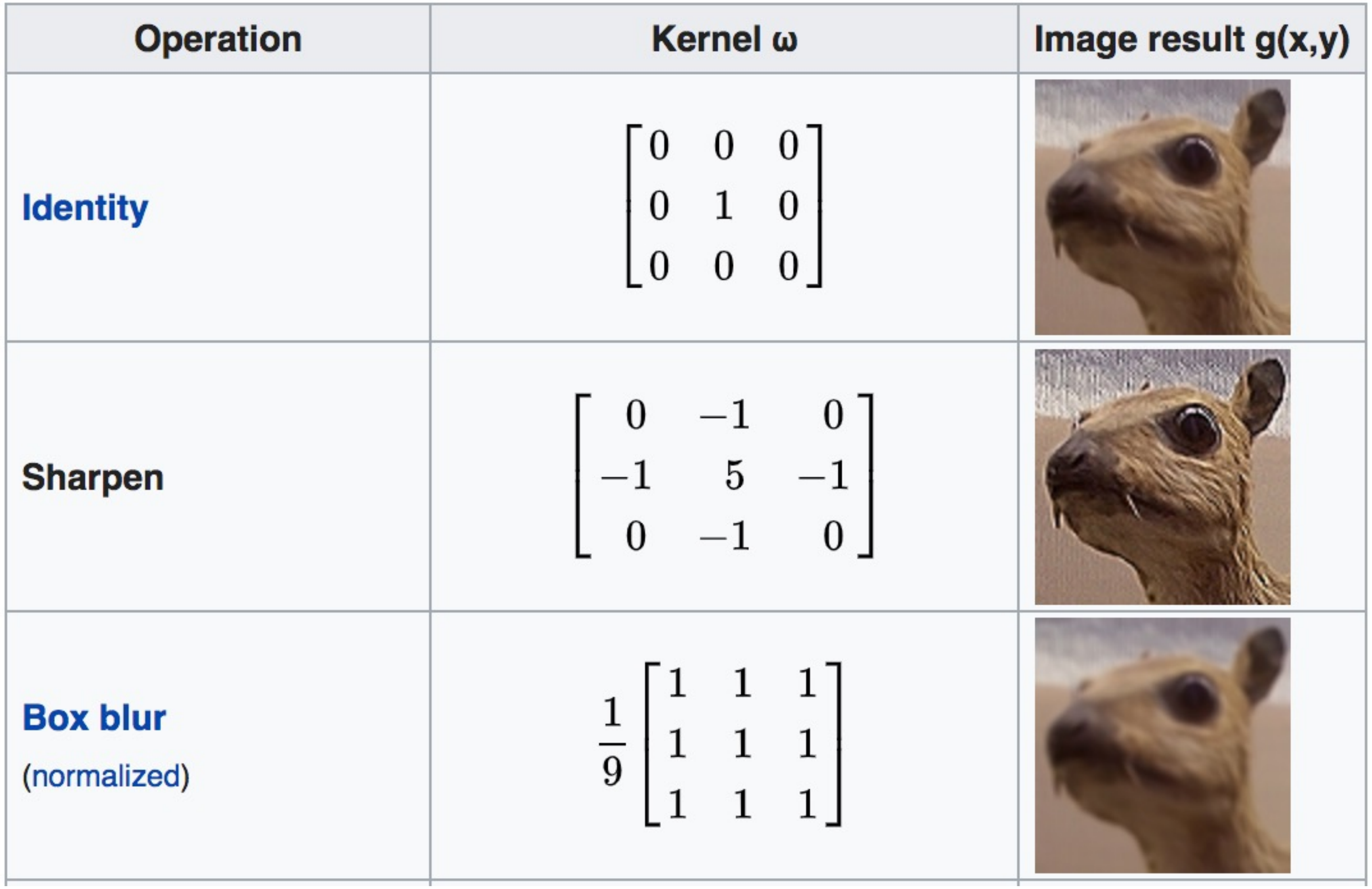
Convolutional Filters



Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

More Filters

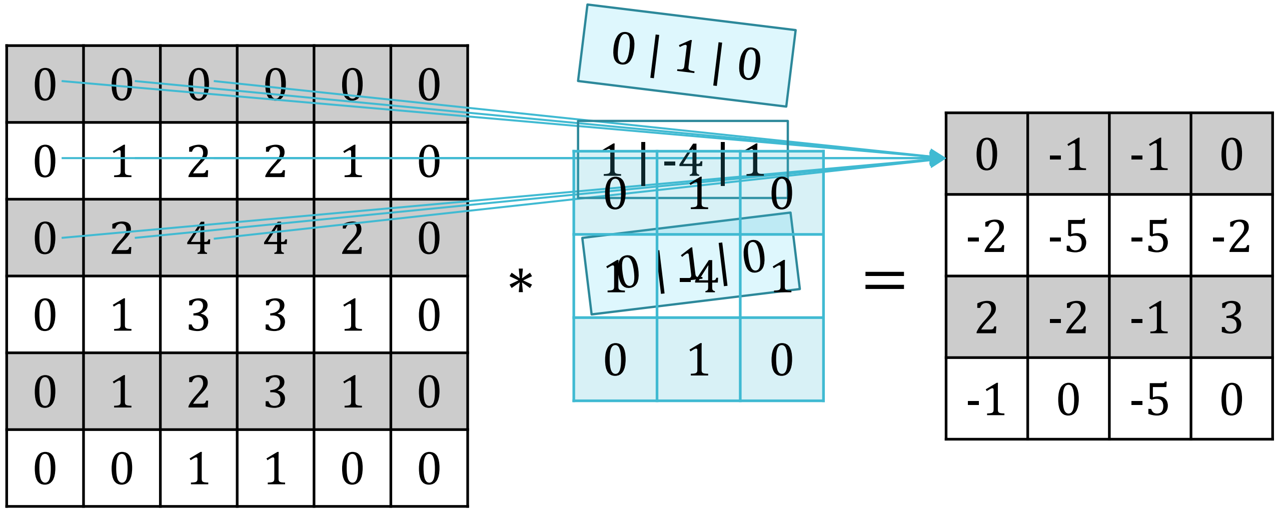


Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Source: https://en.wikipedia.org/wiki/Kernel_(image_processing)

Convolutional Filters

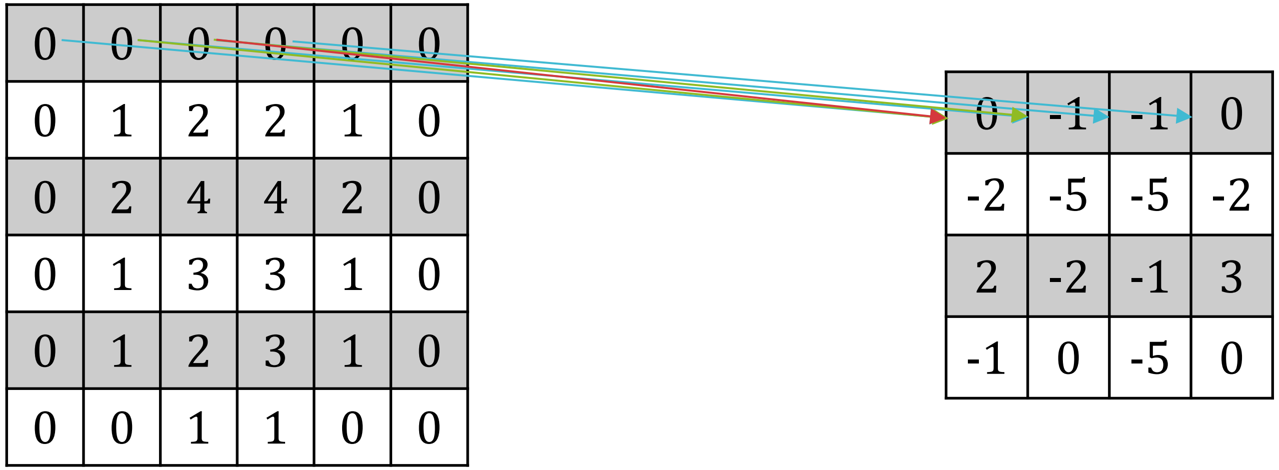
- Images can be represented as matrices: each element corresponds to a pixel and its value is the intensity
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix



$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1 & 0 \\ -2 & -5 & -5 & -2 \\ 2 & -2 & -1 & 3 \\ -1 & 0 & -5 & 0 \end{bmatrix}$$

Convolutional Filters

- Convolutions can be represented by a feed forward neural network where:
  1. Nodes in the input layer are only connected to some nodes in the next layer but not all nodes.
  2. Many of the weights have the same value.



0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

- Many fewer weights than a fully connected layer!
- **Convolution weights are learned using gradient descent/ backpropagation, not prespecified**

Convolutional Filters: Padding

- What if relevant features exist at the border of our image?
- Add zeros around the image to allow for the filter to be applied “everywhere” e.g. a *padding* of 1 with a 3x3 filter preserves image size and allows every pixel to be the center

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	2	2	1	0	0
0	0	2	4	4	2	0	0
0	0	1	3	3	1	0	0
0	0	1	2	3	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

$*$

0	1	0
1	-4	1
0	1	0

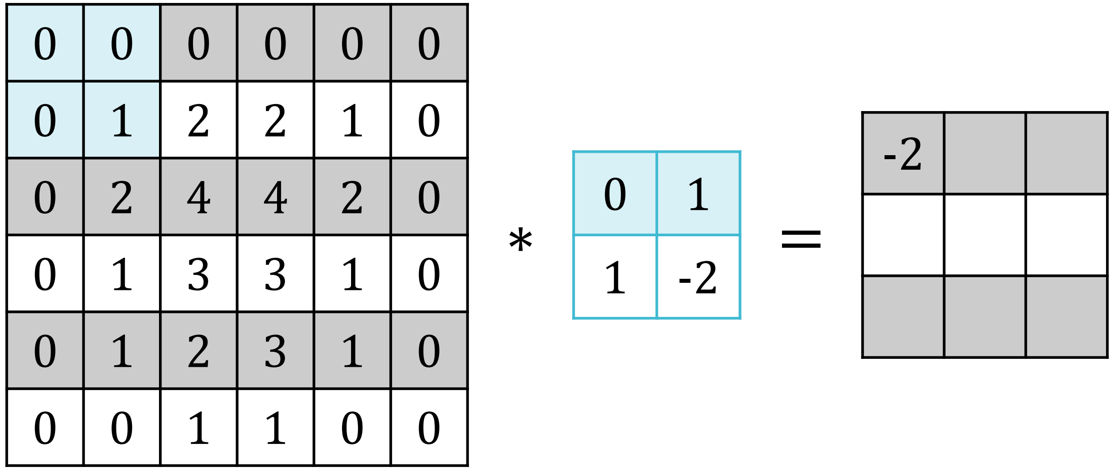
$=$

0	1	2	2	1	0
1	0	-1	-1	0	1
2	-2	-5	-5	-2	2
1	2	-2	-1	3	1
1	-1	0	-5	0	1
0	2	-1	0	2	0

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

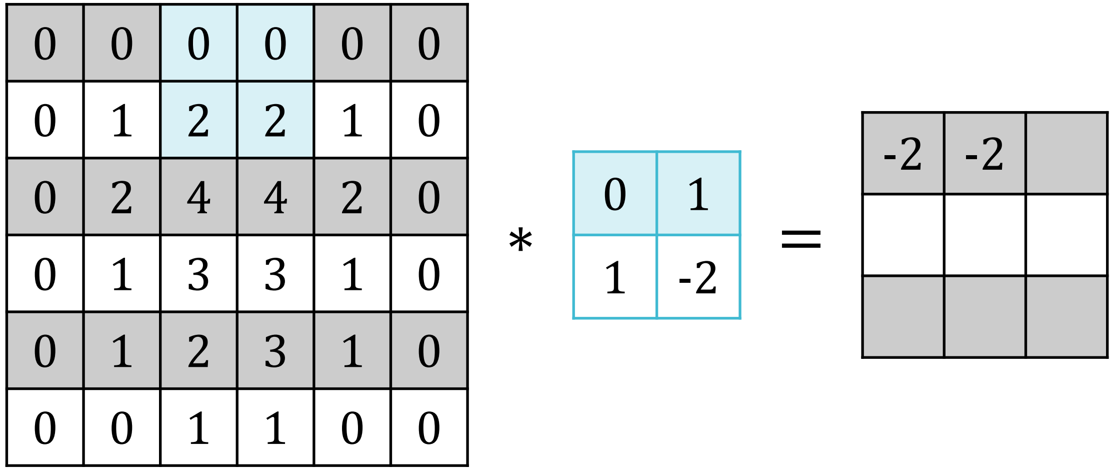


$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 0 \\ \hline 0 & 2 & 4 & 4 & 2 & 0 \\ \hline 0 & 1 & 3 & 3 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & -2 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline -2 & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

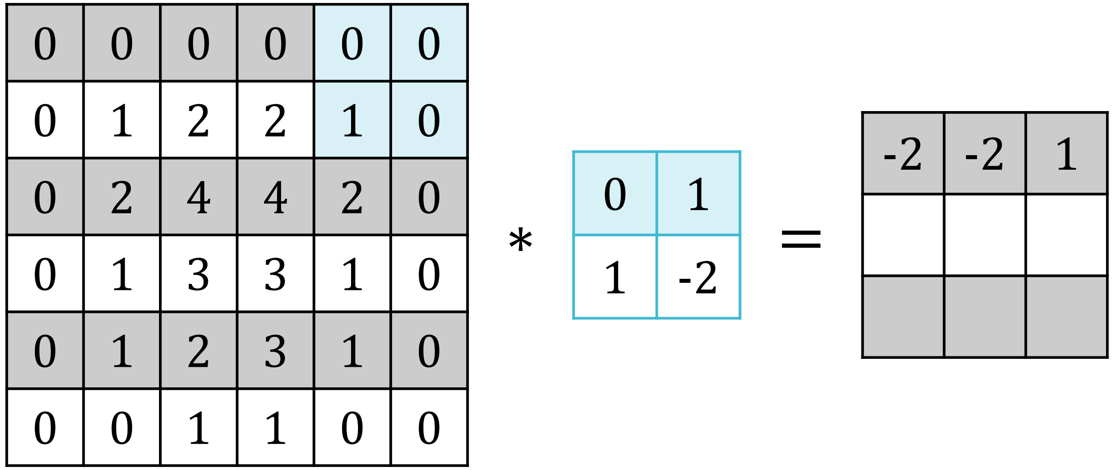


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & \\ & & \\ & & \end{bmatrix}$$

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

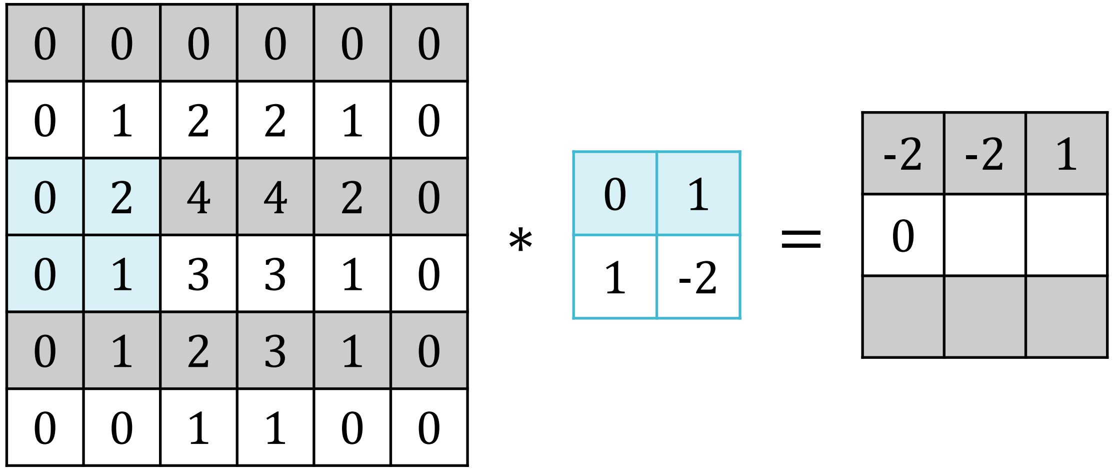


$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 1 \\ & & \\ & & \end{bmatrix}$$

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2



$$\begin{array}{|c|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 & 1 & 0 \\ \hline 0 & 2 & 4 & 4 & 2 & 0 \\ \hline 0 & 1 & 3 & 3 & 1 & 0 \\ \hline 0 & 1 & 2 & 3 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} * \begin{array}{|c|c|}\hline 0 & 1 \\ \hline 1 & -2 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline -2 & -2 & 1 \\ \hline 0 & & \\ \hline & & \\ \hline \end{array}$$

Downsampling: Stride

- Only apply the convolution to some subset of the image

  e.g., every other column and row = a *stride* of 2

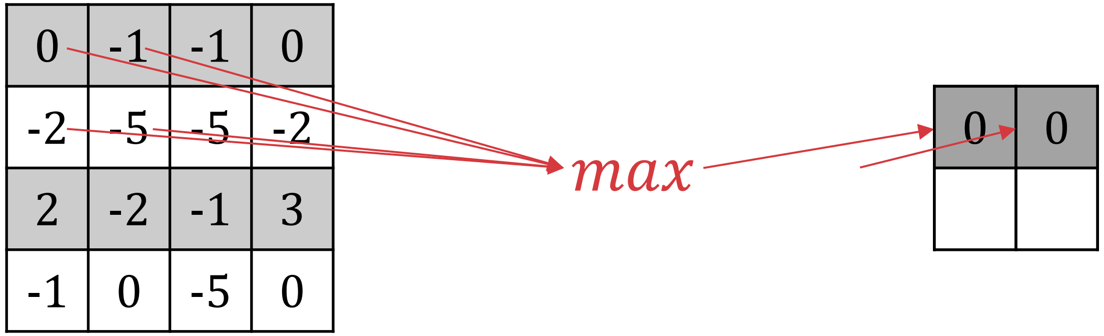
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 & 1 & 0 \\ 0 & 2 & 4 & 4 & 2 & 0 \\ 0 & 1 & 3 & 3 & 1 & 0 \\ 0 & 1 & 2 & 3 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 & 1 \\ 0 & 1 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned

- Many relevant macro-features will tend to span large portions of the image, so taking strides with the convolution tends not to miss out on too much

Downsampling: Pooling

- Combine multiple adjacent nodes into a single node

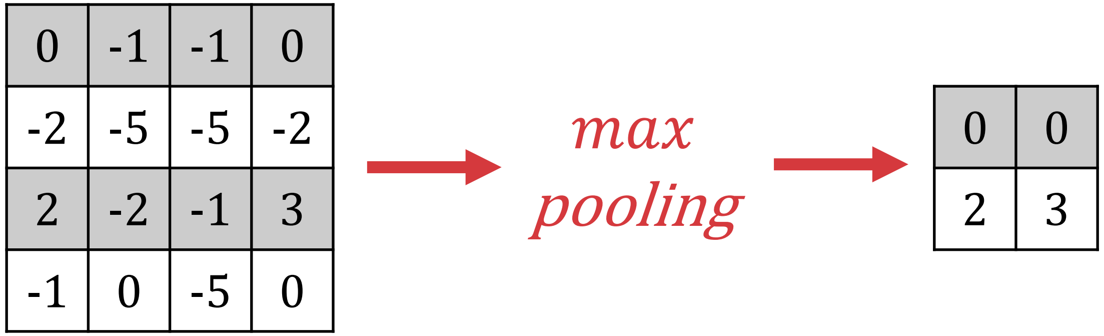


0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

0	0

Downsampling: Pooling

- Combine multiple adjacent nodes into a single node

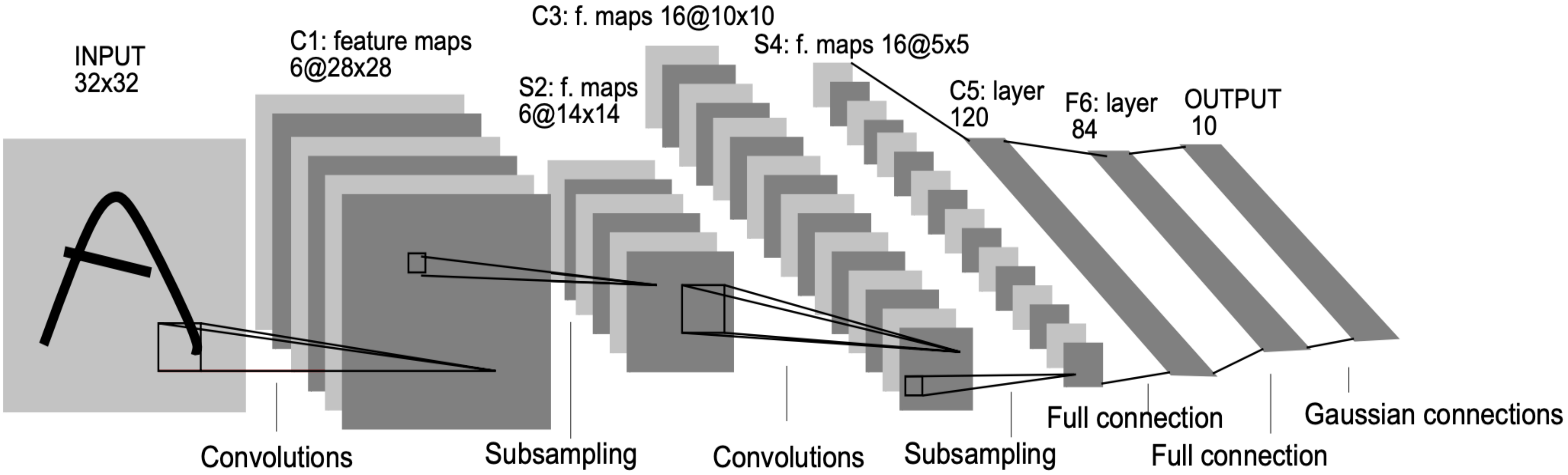


0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

max pooling

0	0
2	3

- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
  - Protects the network from (slightly) noisy inputs



LeNet (LeCun et al., 1998)

Source: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf