ECE 1513: Introduction to Machine Learning

Lecture 9: Generalization and Convolutional Networks

Ali Bereyhi

ali.bereyhi@utoronto.ca

Department of Electrical and Computer Engineering

University of Toronto

Winter 2025

Recap: *Building and Training NN*

`TrainingLoop()`:

1: Build NN $\boldsymbol{y} = f_{\mathbf{w}}(\boldsymbol{x})$ with some hyperparameters and *initial weights*
2: Split training set to mini-batches
3: Specify the loss function $\mathcal{L}$
4: **for** *epochs* = $1, \ldots, E$ **do**
5: Keep applying mini-batch SGD
6: **end for**
7: Return final weights $\mathbf{w}^*$, average training loss, and accuracy on training set

Recap: *Validation*

`Validation()`:

1: Realize NN $\boldsymbol{y} = f_{\mathbf{w}*}(\boldsymbol{x})$ with *trained weights* $\mathbf{w}^*$
2: **for** *sample* i *in validation set* **do**
3: Compute $\boldsymbol{y}_i = f_{\mathbf{w}*}(\boldsymbol{x}_i)$
4: Compute sample loss $\hat{R}_i = \mathcal{L}(\boldsymbol{y}_i, \boldsymbol{v}_i)$
5: Check whether inferred label $\hat{\boldsymbol{v}}_i$ is the same as true label $\boldsymbol{v}_i$
6: **end for**
7: Return average validation loss and accuracy

We repeat training with new hyperparameters till it's tuned

Recap: *Evaluation*

`Evaluation()`:

1: Realize NN $\boldsymbol{y} = f_{\mathbf{w}*}(\boldsymbol{x})$ with *trained weights* $\mathbf{w}^*$ *and tuned hyperparameters*
2: **for** *sample* i *in evaluation set* **do**
3: Compute $\boldsymbol{y}_i = f_{\mathbf{w}*}(\boldsymbol{x}_i)$
4: Compute sample loss $\hat{R}_i = \mathcal{L}(\boldsymbol{y}_i, \boldsymbol{v}_i)$
5: Check whether inferred label $\hat{\boldsymbol{v}}_i$ is the same as true label $\boldsymbol{v}_i$
6: **end for**
7: Return average evaluation loss and accuracy

Today's Agenda: *Generalization and Convolution*

Today's lecture has two parts: in the first part, we talk about

Generalization

In this way, we discuss the following topics

- *Generalization concept and bias-variance trade-off*
- *Memory-capacity trade-off and overfitting*
- *Techniques to reduce overfitting*

In the second part, we learn

Convolution

as an alternative linear operation which enables us to develop

- *Convolutional Neural Networks*

Trained Model on New Data

Once we are finished with training, we use our model for inference

we hope our model works fine on unseen samples as well

Generalization

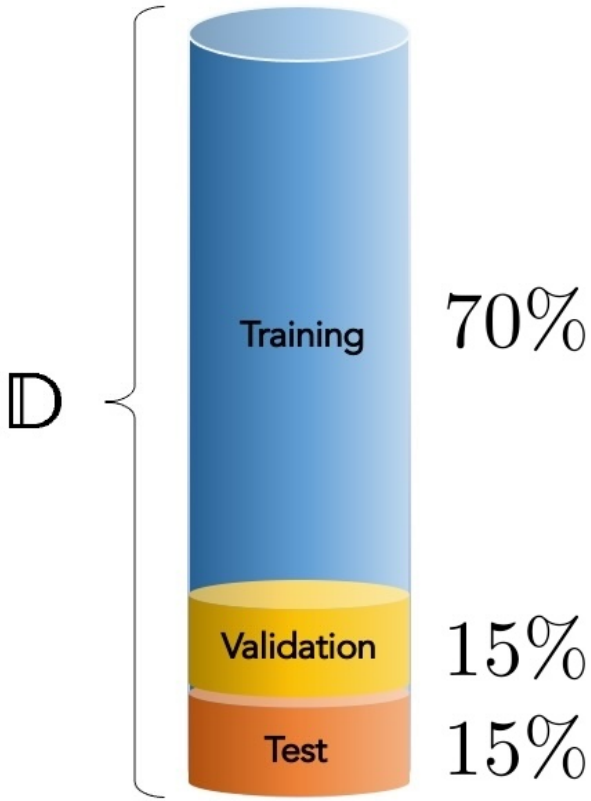
Ability of a trained model to perform well on unseen data

Unseen data is coming from a randomly and its label is unknown to us!

Measuring Generalization

? *How can we measure generalization?*

! *We try samples that we did not use for training*



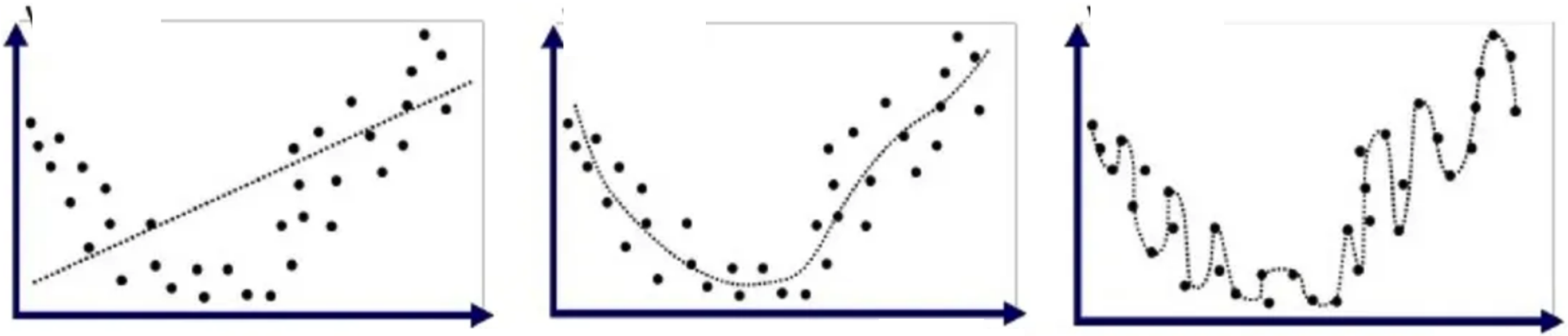
Underfitting and Overfitting

If model does not generalize *we see one of these cases*

- *Training loss is small but validation loss is large!* $\rightsquigarrow$ *Overfitting*
  - ↳ *Model fits dataset* $\rightsquigarrow$ *it memorizes dataset*
  - ↳ *It does not generalize to new data* $\rightsquigarrow$ *it did not learn properly*
- *Training loss is large and validation loss is large!* $\rightsquigarrow$ *Underfitting*
  - ↳ *Model does not fit dataset* $\rightsquigarrow$ *it cannot memorize dataset*
  - ↳ *It does not generalize to new data* $\rightsquigarrow$ *it did not learn properly*

Nowadays, we mainly observe overfitting!

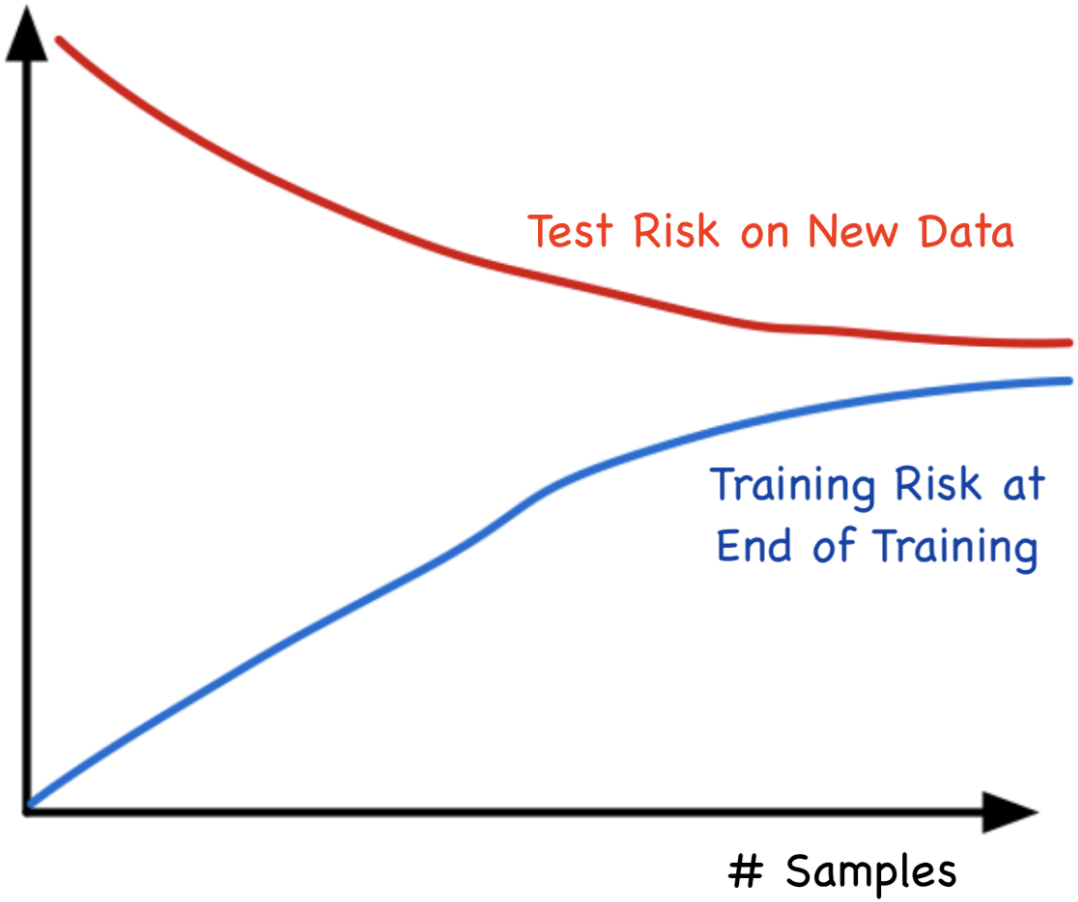
Underfitting and Overfitting



- *Left* $\rightsquigarrow$ *Underfitting*
- *Middle* $\rightsquigarrow$ *Learning fairly*
- *Right* $\rightsquigarrow$ *Overfitting*

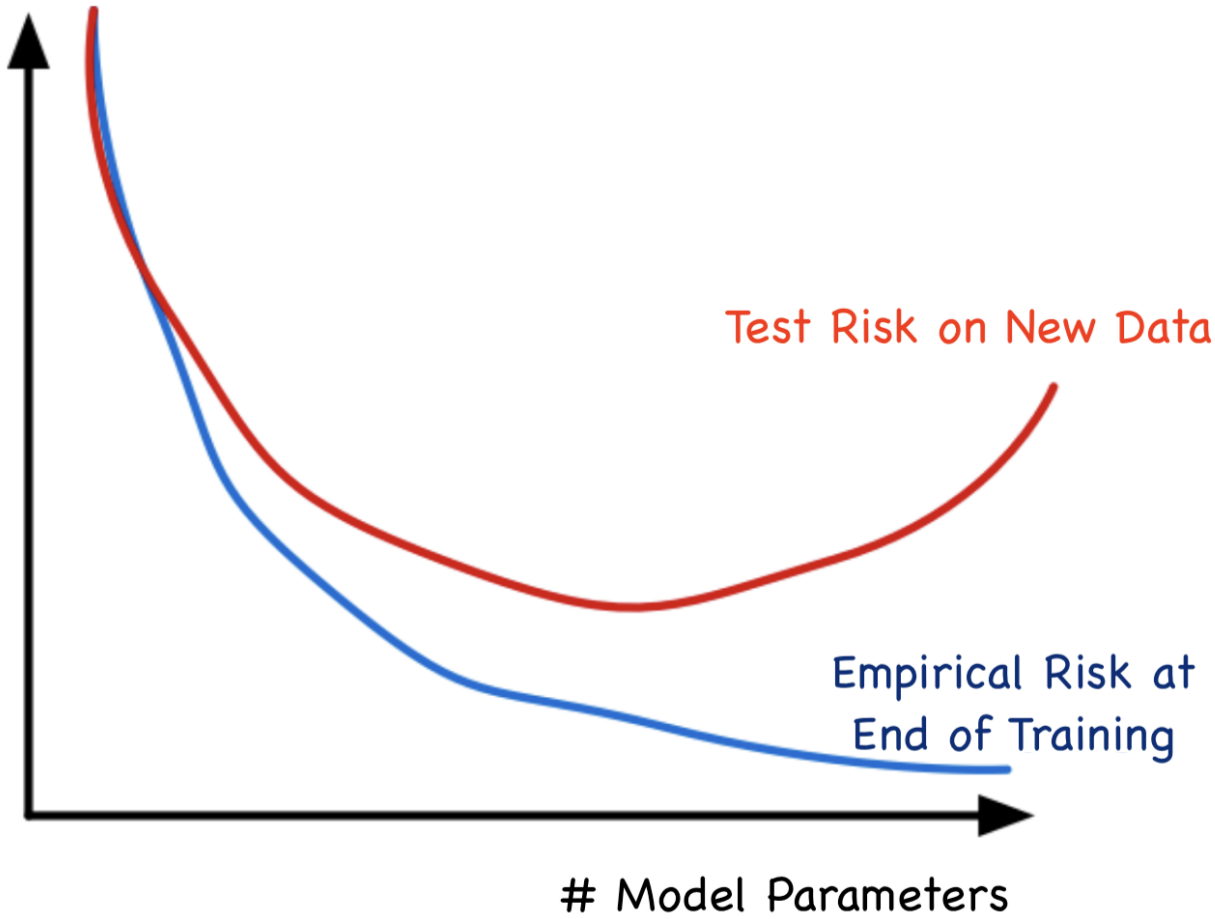
Generalization vs Data

Adding more data reduces chance of memorizing and increase learning ability



Generalization vs Model Capacity

More model parameters increase both memorizing and learning abilities



Formulating Generalization

Let us consider a simple case: *we train a model* $f_{\mathbf{w}}(\cdot)$ *on dataset*

$$\mathbb{D} = \{(\boldsymbol{x}_n, v_n) : n \in [N]\}$$

We evaluate generalization by squared error: for a new point $\boldsymbol{x}$

$$\mathcal{E} = \mathbb{E}\left\{(y - v)^2\right\}$$

- y *is the prediction given by our trained model*

$$y = f_{\mathbf{w}}(\boldsymbol{x})$$

- v *is the label that we do not know*

Generalization Error

Assume that samples are drawn i.i.d. from a data distribution: *each sample*

$$(\boldsymbol{x}, v) \sim P(\boldsymbol{x}, v)$$

We can use Bayes' rule to write

$$P(\boldsymbol{x}, v) = P(\boldsymbol{x})\, P(v|\boldsymbol{x})$$

- $P(\boldsymbol{x})$ *is the distribution of data-points*
- $P(v|\boldsymbol{x})$ *is the distribution of labels when* $\boldsymbol{x}$ *is known*

Attention

We do not have access to $P(\boldsymbol{x})$ *or* $P(v|\boldsymbol{x})$ *in practice! They are hypothetical*

Generalization Error: *Single Sample*

Assume that we only care about sample $(\boldsymbol{x}, v)$: *we know* $\boldsymbol{x}$ *but not* v

$$\begin{aligned}\mathcal{E}(\boldsymbol{x}) &= \mathbb{E}\left\{(y-v)^2 \,|\boldsymbol{x}\right\} \\ &= \mathbb{E}\left\{y^2 - 2yv + v^2|\boldsymbol{x}\right\} \\ &= \mathbb{E}\left\{y^2|\boldsymbol{x}\right\} - 2\mathbb{E}\left\{2v|\boldsymbol{x}\right\} + \mathbb{E}\left\{v^2|\boldsymbol{x}\right\}\end{aligned}$$

Since we know $\boldsymbol{x}$*, we know* $y = f_{\mathbf{w}}(\boldsymbol{x})$ *as well*

$$\mathcal{E}(\boldsymbol{x}) = y^2 - 2y\mathbb{E}\left\{v|\boldsymbol{x}\right\} + \mathbb{E}\left\{v^2|\boldsymbol{x}\right\}$$

Generalization Error: *Single Sample*

The error for a single sample is

$$\mathcal{E}(\boldsymbol{x}) = y^2 - 2y\mathbb{E}\{v|\boldsymbol{x}\} + \mathbb{E}\{v^2|\boldsymbol{x}\}$$

Let's define the mean and variance of v determined by $P(v|\boldsymbol{x})$ as

$$\mu(\boldsymbol{x}) = \mathbb{E}\{v|\boldsymbol{x}\} \qquad \sigma^2(\boldsymbol{x}) = \mathbb{E}\{v^2|\boldsymbol{x}\} - \mu^2(\boldsymbol{x})$$

Then, we can write

$$\begin{aligned}\mathcal{E}(\boldsymbol{x}) &= y^2 - 2y\mu(\boldsymbol{x}) + \mu^2(\boldsymbol{x}) + \sigma^2(\boldsymbol{x}) \\ &= [y - \mu(\boldsymbol{x})]^2 + \sigma^2(\boldsymbol{x}) \\ &= \text{Model Bias}^2 + \text{Label Variance}\end{aligned}$$

Generalization Error: *Bias and Variance*

Generalization Error

Generalization error for single new sample $\boldsymbol{x}$ is

$$\mathcal{E}(\boldsymbol{x}) = [y - \mu(\boldsymbol{x})]^2 + \sigma^2(\boldsymbol{x})$$

The only thing that we can control is y

- *We can hope that our model gives us zero bias $\rightsquigarrow$ unbiased estimator*

$$y = f_{\mathbf{w}}(\boldsymbol{x}) = \mathbb{E}\{v|\boldsymbol{x}\} = \mu(\boldsymbol{x})$$

- *We can do nothing about $\sigma^2(\boldsymbol{x})$ $\rightsquigarrow$ Bayes error*

Bayes Error

Minimum theoretically possible generalization error for a learning problem

Generalization Error: *Expected*

But, we don't care about a single point: *we care about expected error*

$$\mathbb{E}\left\{\mathcal{E}\left(\boldsymbol{x}\right)\right\} = \mathbb{E}\left\{\left[y - \mu\left(\boldsymbol{x}\right)\right]^2\right\} + \mathbb{E}\left\{\sigma^2\left(\boldsymbol{x}\right)\right\}$$

Let's define the mean and variance of the model output

$$\bar{y} = \mathbb{E}\left\{y\right\} \qquad \mathbb{V}\text{ar}\left\{y\right\} = \mathbb{E}\left\{\left(y - \bar{y}\right)^2\right\} = \mathbb{E}\left\{y^2\right\} - \bar{y}^2$$

We can then write

$$\begin{aligned}
\mathbb{E}\left\{\mathcal{E}\left(\boldsymbol{x}\right)\right\} &= \mathbb{E}\left\{\left[y - \bar{y} + \bar{y} - \mu\left(\boldsymbol{x}\right)\right]^2\right\} + \mathbb{E}\left\{\sigma^2\left(\boldsymbol{x}\right)\right\} \\
&= \mathbb{E}\left\{\left[y - \bar{y} + \bar{y} - \mu\left(\boldsymbol{x}\right)\right]^2\right\} + \mathbb{E}\left\{\sigma^2\left(\boldsymbol{x}\right)\right\} \\
&= \mathbb{V}\text{ar}\left\{y\right\} + 2\mathbb{E}\left\{\left(y - \bar{y}\right)\left(\bar{y} - \mu\left(\boldsymbol{x}\right)\right)^2\right\} + \mathbb{E}\left\{\sigma^2\left(\boldsymbol{x}\right)\right\}
\end{aligned}$$

Generalization Error: *Expected*

Let's further define the mean and variance of the label

$$\bar{v} = \mathbb{E}\{v\} = \mathbb{E}\{\mu(\boldsymbol{x})\} \qquad \mathbb{V}\text{ar}\{y\} = \mathbb{E}\left\{(v - \bar{v})^2\right\} = \mathbb{E}\left\{\sigma^2(\boldsymbol{x})\right\}$$

We can then approximately write

$$\begin{aligned}\mathbb{E}\{\mathcal{E}(\boldsymbol{x})\} &\approx (\bar{y} - \bar{v})^2 + \mathbb{V}\text{ar}\{y\} + \mathbb{V}\text{ar}\{v\} \\ &= \text{Model Bias}^2 + \text{Model Variance} + \text{Label Variance}\end{aligned}$$

Generalization Error Components

Generalization error is proportional to the model bias and variance

Generalization Error: *Bias and Variance*

The only thing that we can control is y

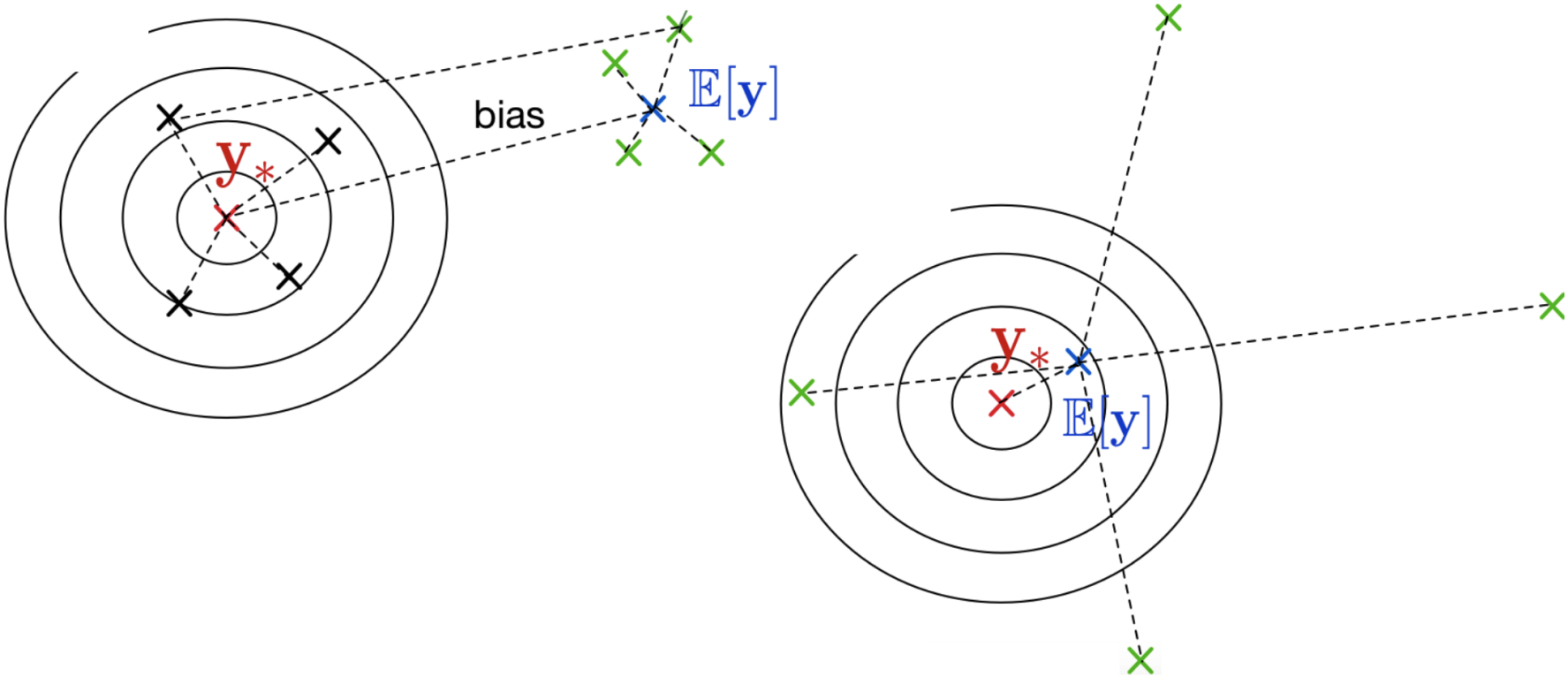
- *We set the model to give us zero bias* $\rightsquigarrow$ *unbiased estimator*

$$\bar{y} = \mathbb{E}\left\{f_{\mathbf{w}}\left(\boldsymbol{x}\right)\right\} = \mathbb{E}\left\{v\right\} = \bar{v}$$

- *But this may lead to higher variance* $\mathbb{V}\mathrm{ar}\left\{y\right\}$
- *There is always a minimum error that we cannot beat* $\rightsquigarrow$ *Bayes error*

Bias and Variance: *Visualization*

We can visualize it as[1]



[1] *Borrowed from ML lecture-notes by R. Grosse*

Tune Hyperparameters

Most basic approach is to tune hyperparameters

Hyperparameter

Any parameter we set to realize our model, e.g., depth, width, etc

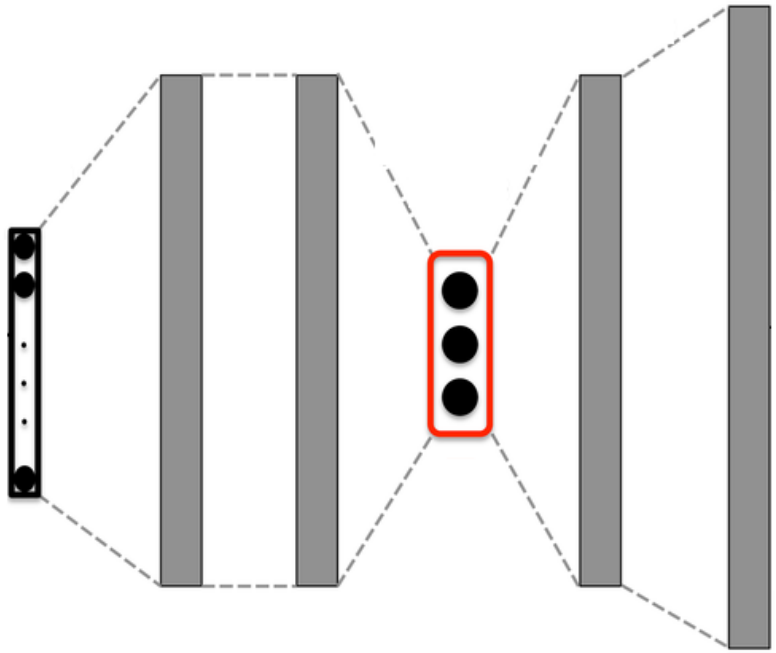
Start with initial setting

- *Train the model and save the training loss*
- *Evaluate it on the validation dataset*
  - ↳ *If the loss is too different from training loss*
    - ↳ *change hyperparameters and re-train*
  - ↳ *Else*
    - ↳ *training is over*

? *How can we search?*

! *Depending on the complexity: random or grid search*

Bottleneck



Bottleneck can drop the model capacity

- *Say we add a 10-neuron layer between two 200-neuron layers*
  - ↳ *Without bottleneck we have 40,000 weights*
  - ↳ *With bottleneck we have 4,000 weights!*

Regularization: *General Form*

In regularization, we add a penalty to empirical risk: *training minimizes*

$$\tilde{R}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(y_n, v_n) + \lambda\Pi(\mathbf{w})$$

Penalty is proportional to the property of weights when model overfits

Example

If we know that weights get large when model overfits we could set

$$\Pi(\mathbf{w}) = \|\mathbf{w}\|^2 = \sum_i w_i^2$$

- *We typically pre-set the regularization function, i.e., penalty*
- *λ is a hyperparameter that we call regularizer and tune by validation*

Regularization: ℓ_p-Norm

Famous choice of penalty is ℓ_p-norm: *we set*

$$\Pi(\mathbf{w}) = \|\mathbf{w}\|_p^p = \sum_i |w_i|^p$$

- *ℓ_2-norm is the classic choice $\rightsquigarrow$ Tikhonov*
  - ↳ *It avoids weights to get very large*
  - ↳ *You did it in Assignment 2 for regression $\rightsquigarrow$ Ridge regression*
- *ℓ_1-norm is another classic choice $\rightsquigarrow$ Lasso*
  - ↳ *I enforces the wrights to be sparse $\rightsquigarrow$ contain too many zeros*
  - ↳ *Great piece of work by Robert Tibshirani* ☺

Regularization and Weight Decay

Let's set $\Pi(\mathbf{w}) = \|\mathbf{w}\|^2/2$ *and write GD*

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \eta \nabla \tilde{R}(\mathbf{w}) \\ &\leftarrow \mathbf{w} - \eta \nabla \left(\frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y_n, v_n) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right)\end{aligned}$$

Now focus on a single w_i*: it would be like a weight decay*

$$\begin{aligned}w_i &\leftarrow w_i - \eta \left(\frac{\partial}{\partial w_i} \hat{R}(\mathbf{w}) + \frac{\lambda}{2} \frac{\partial}{\partial w_i} \|\mathbf{w}\|^2 \right) \\ &\leftarrow w_i - \eta\lambda w_i - \eta \frac{\partial}{\partial w_i} \hat{R}(\mathbf{w}) = w_i (1 - \eta\lambda) - \eta \frac{\partial}{\partial w_i} \hat{R}(\mathbf{w})\end{aligned}$$

Some people call ℓ_2*-norm regularization weight decay*

Stochastic Regularization

We can also apply stochastic regularization: *stating training with SGD*

- *In each gradient computation*
  - ↳ *Apply a stochastic operation on neuron outputs*
- *Average all sample gradients over mini-batch*
- *Apply an iteration of GD*

The most famous approach is dropout

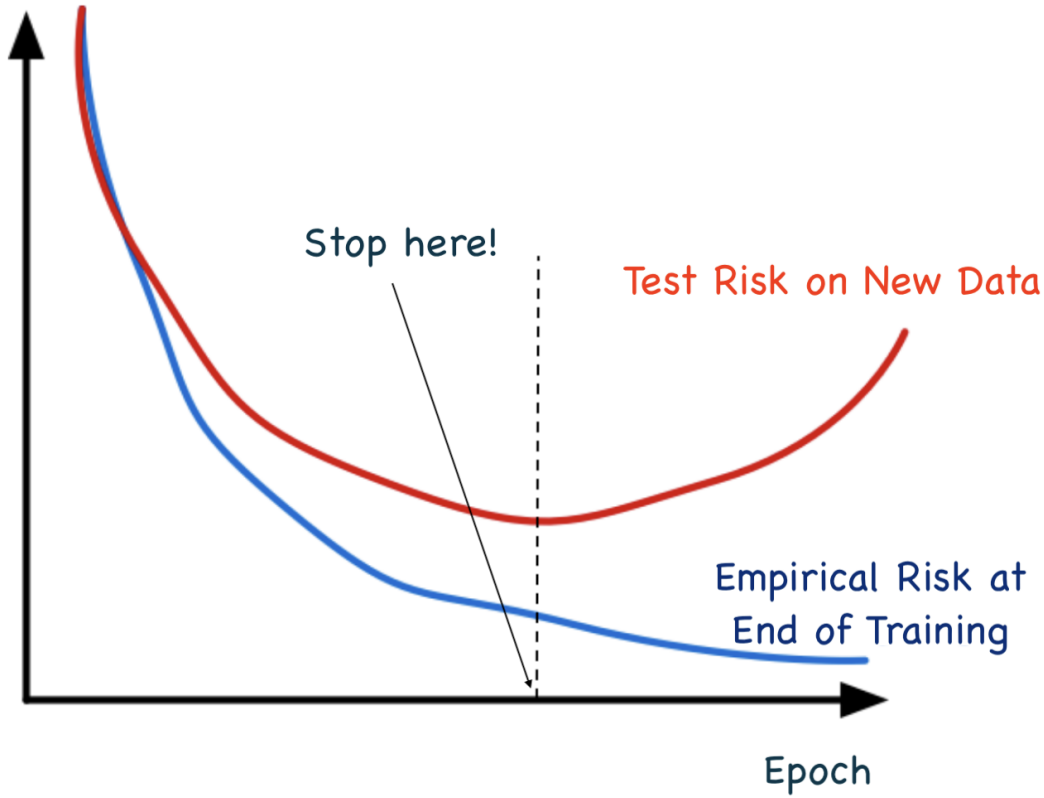
- *In each gradient computation*
  - ↳ *we randomly drop some neurons*

$$y_i \leftarrow m_i y_i$$

where $m_i \in \{0, 1\}$ *is a random mask*

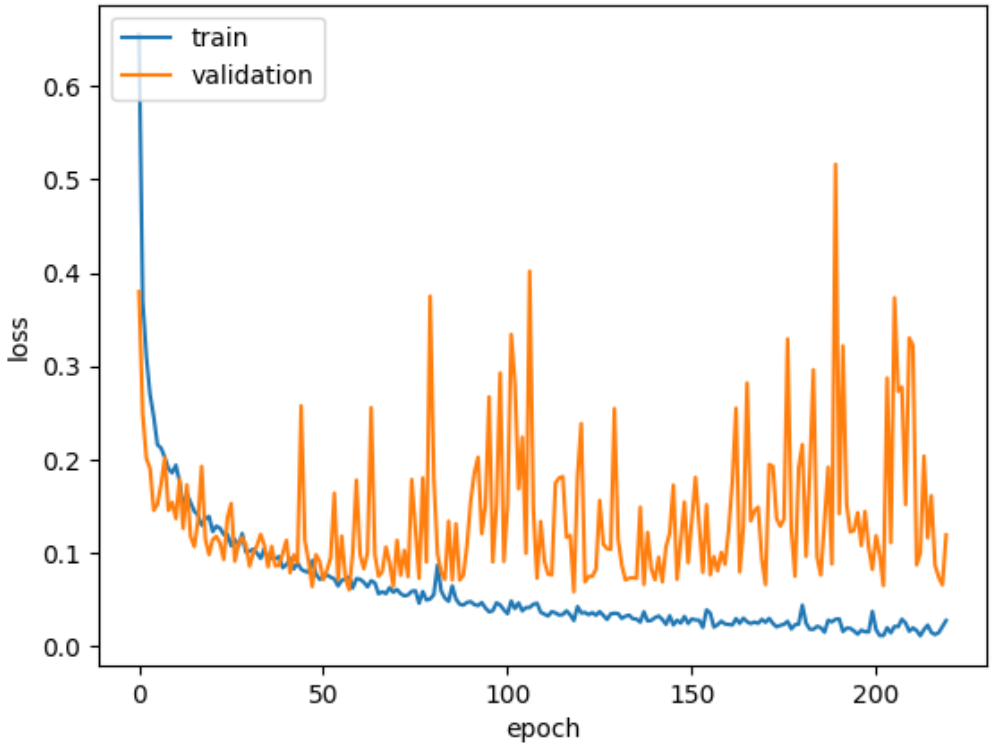
Early Stopping

We may stop when we the model is well-trained: *validate after each epoch*



Early Stopping

In practice though it's hard: *we usually see*



We may use moving average; however, it's not always effective

Ensemble

We could use multiple model: *give the task to m model*

- *Each model predicts a label y_i*
- *We set the average as the final prediction*

$$y = \frac{1}{m} \sum_{i=1}^{m} y_i$$

- ? *How to realize multiple models?*

We can take different approaches

- *Bagging* $\rightsquigarrow$ *use the same model on different data*
- *Stacking* $\rightsquigarrow$ *use different models on the same data*
- *Boosting* $\rightsquigarrow$ *modify the data based on previous predictions and re-train*

Data Augmentation

We can augment to increase samples

Further Read

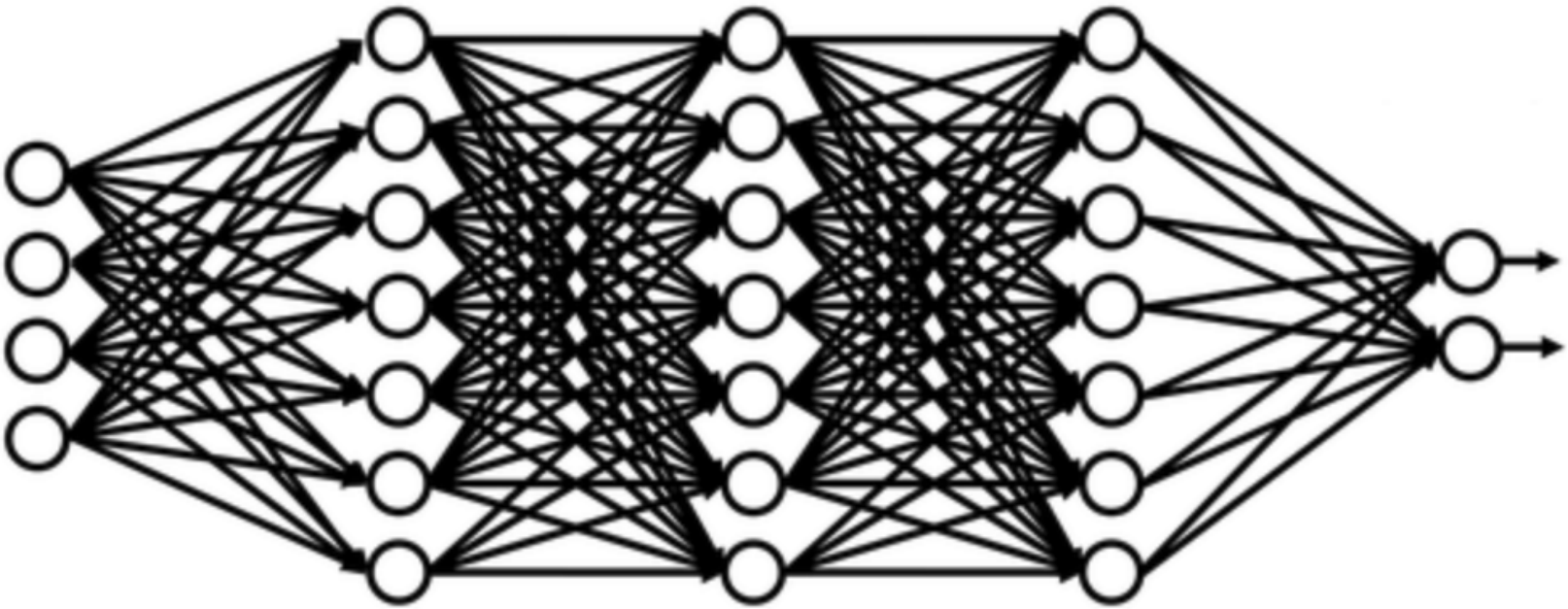
- Goodfellow
  - ↳ Chapter 7

CNNs

Challenges of Fully-Connected Layers

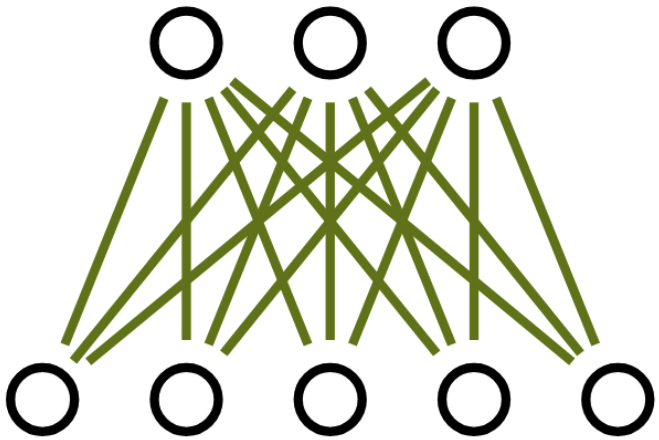
In fully-connected layers we have too much weights

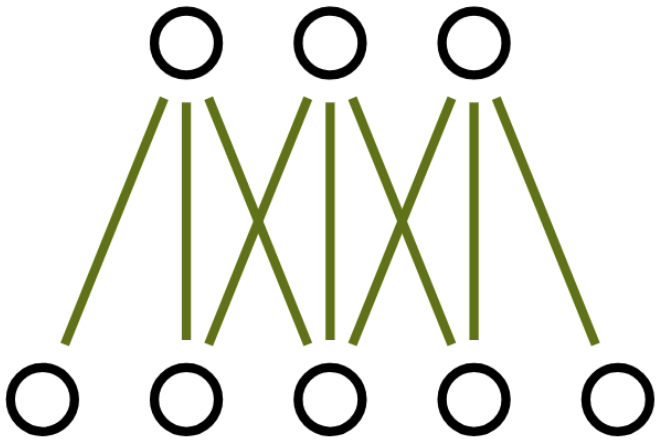
- *Even for a simple task, we end up with high dimensional* $\mathbf{w}$
- *Many of these* $\mathbf{w}$ *do nothing!*



Need for Sparse Linear Operation

We would benefit if *we drop some weights!*





We need a sparsely-weighted linear operation

Mathematical Convolution: *1D Signals*

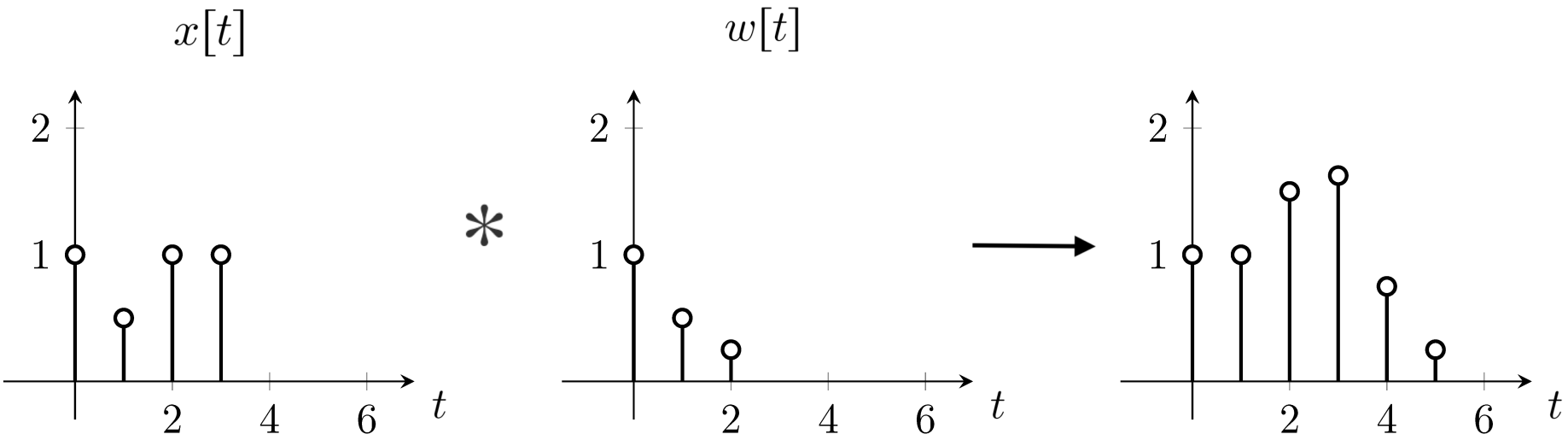
Convolution

For two 1D signals $x[t]$ and $w[t]$, the convolution is defined as

$$(x * w)[t] = \sum_{i=-\infty}^{\infty} x[i]w[t-i]$$

We typically deal with finite signals: *we assume signal is zero anywhere else*

Convolution: *Example*



$$\ldots \qquad (x * w)[0] = \sum_{i=-\infty}^{\infty} x[i]w[-i]$$

$$(x * w)[1] = \sum_{i=-\infty}^{\infty} x[i]w[1-i] \qquad \ldots$$

Convolution: *Sliding Flipped Filter*

We can look at convolution as *sliding a flipped filter over input*

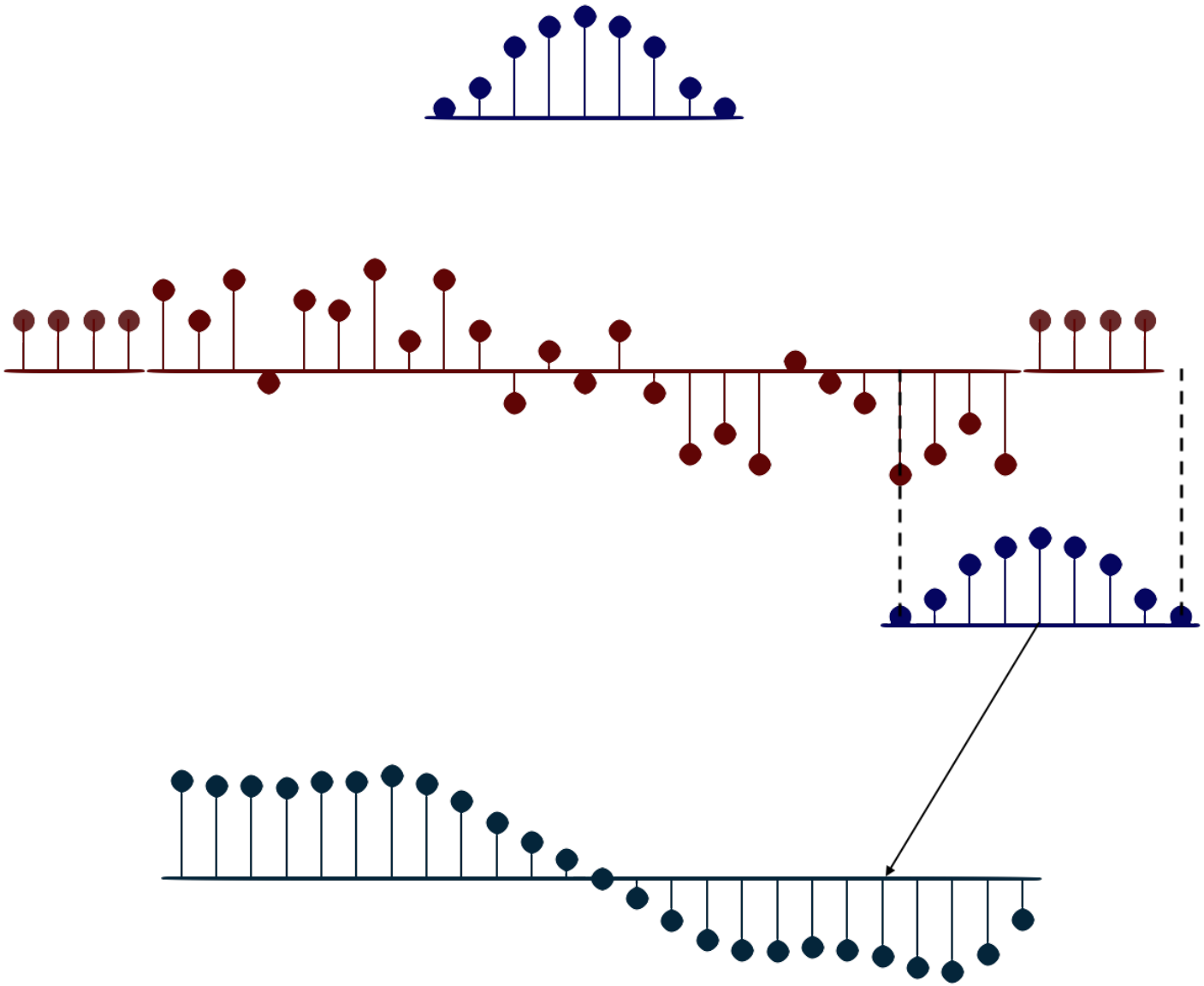
$$(x * w)[t] = \sum_{i=-\infty}^{\infty} x[i]w[t-i]$$

Say $\hat{w}[-t] = w[t]$*; then, we have*

$$(x * w)[t] = \sum_{i=-\infty}^{\infty} x[i]\hat{w}[i-t]$$

This is like sliding $\hat{w}[t]$ *over* $x[t]$ *and averaging*

Convolution: *Sliding Flipped Filter*



Mathematical Convolution: *2D Signals*

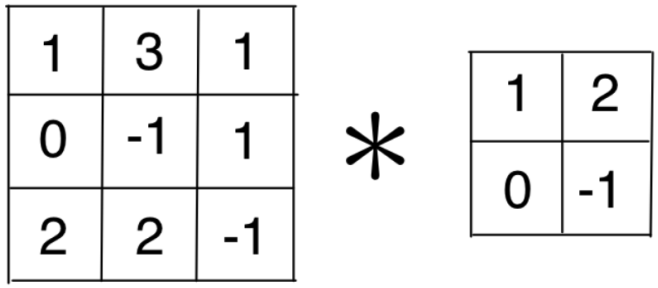
Convolution

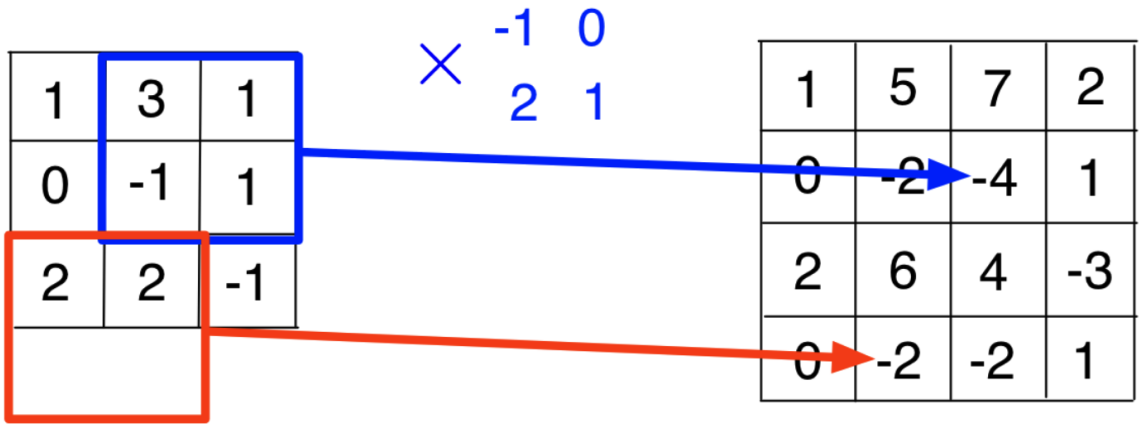
For two 2D signals $\mathbf{X}[t]$ *and* $\mathbf{W}[t]$*, the convolution is defined as*

$$(\mathbf{X} * \mathbf{W})[t, s] = \sum_{i,j=-\infty}^{\infty} \mathbf{X}[i, j]\mathbf{W}[t - i, s - j]$$

Convolution: *Sliding Flipped Filter*

We can again show that *we are sliding the 2D filter after flipping it*



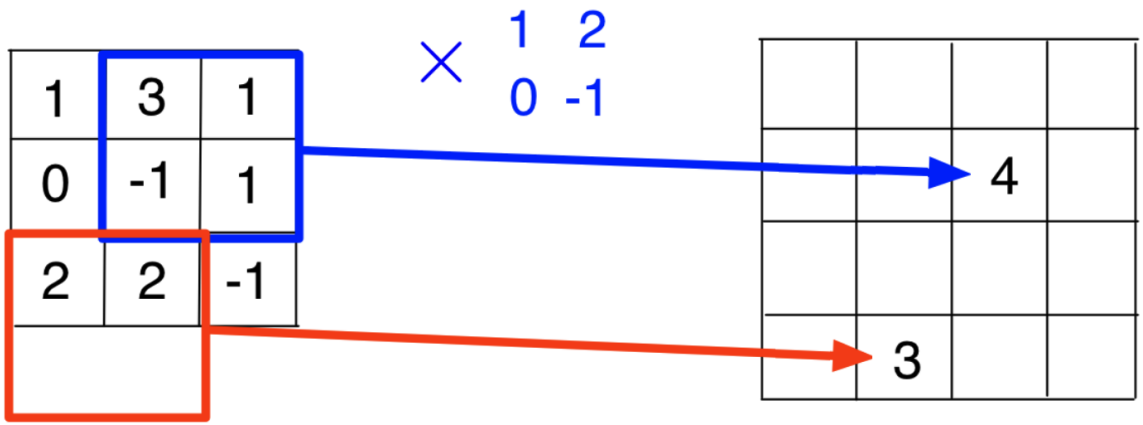


Cross-Correlation: *Convolution in Practice*

Cross-correlation $\equiv$ Convolution without Flipping

For two 2D signals $\mathbf{X}[t]$ and $\mathbf{W}[t]$, the cross-correlation is

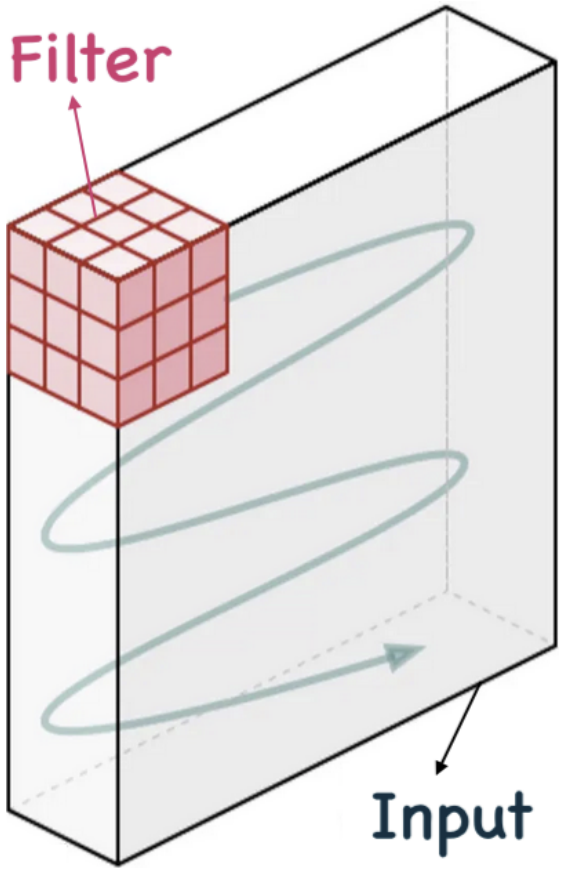
$$(\mathbf{X} * \mathbf{W})[t, s] = \sum_{i,j=-\infty}^{\infty} \mathbf{X}[i, j]\mathbf{W}[i + t, j + s]$$



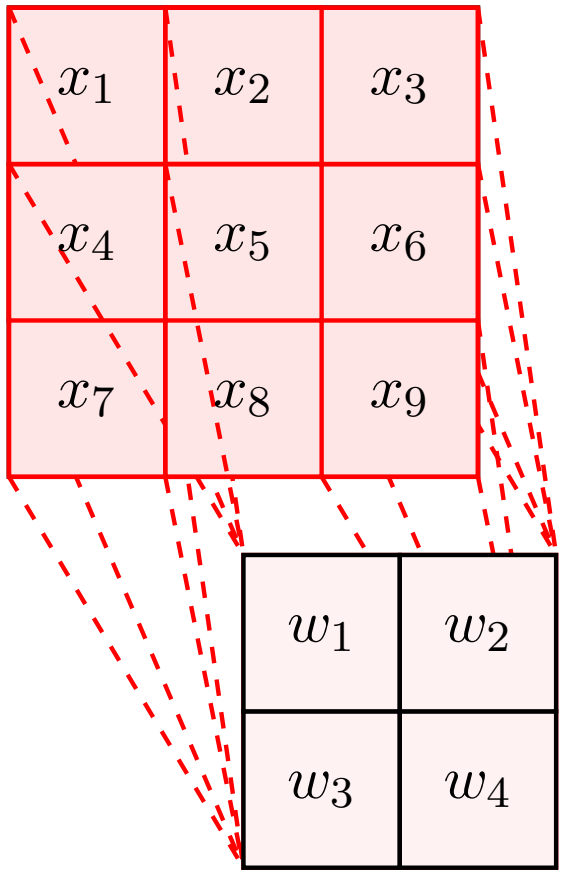
Properties of Convolution

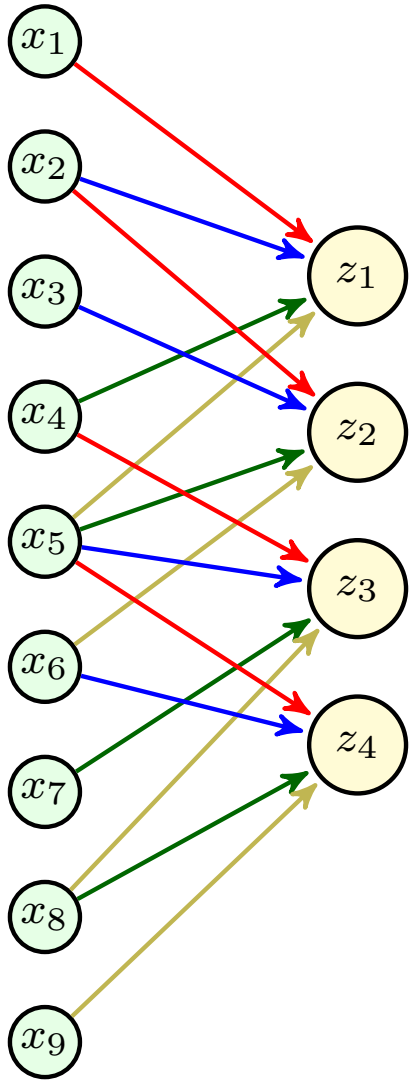
Convolution with 3D Inputs

Convolution can be also done on 3D objects: *we slide 3D filters*



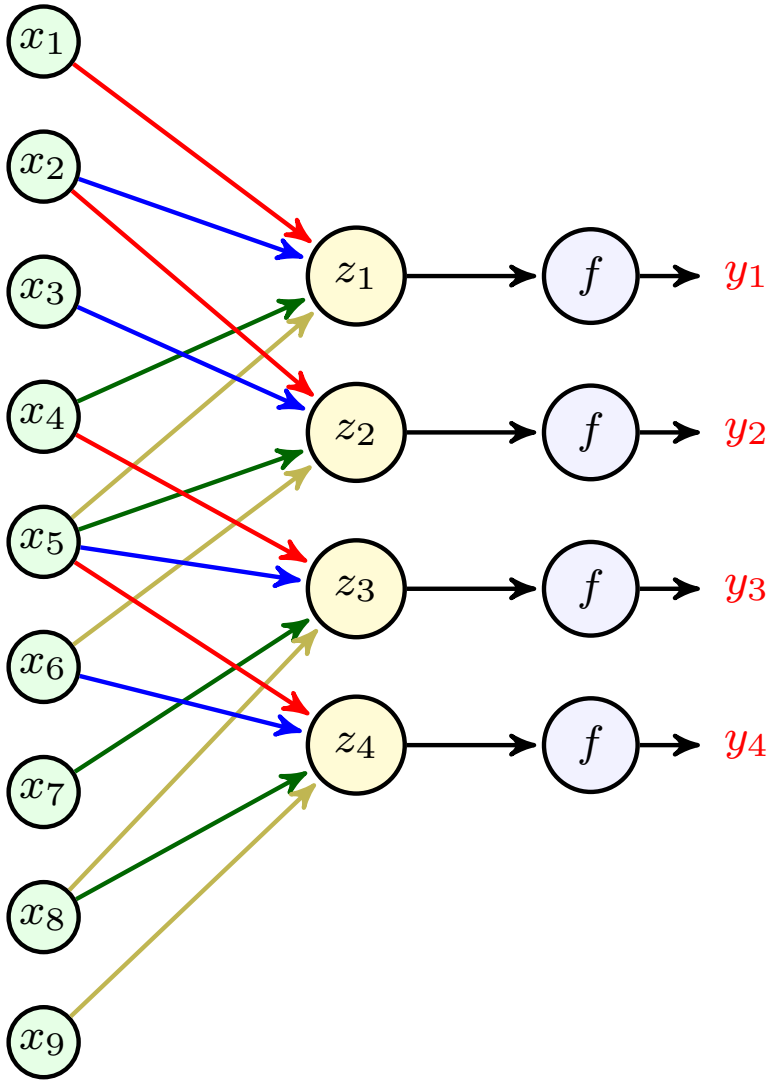
Convolution $\equiv$ *Sparse Linear Operation with Shared Weights*



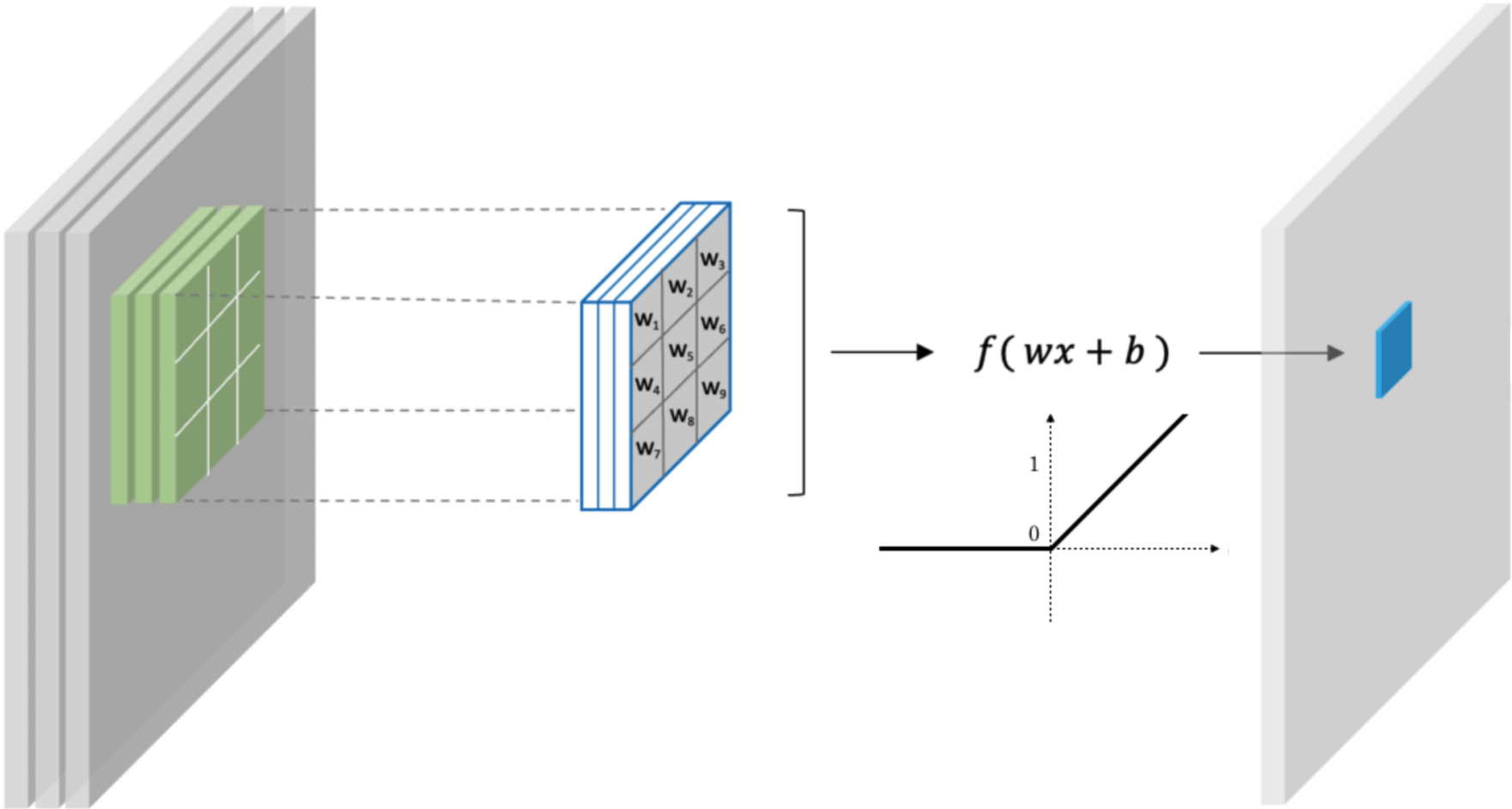


Building Neural Layers with Convolution

We can activate the convolution output to make a partially-connected layer

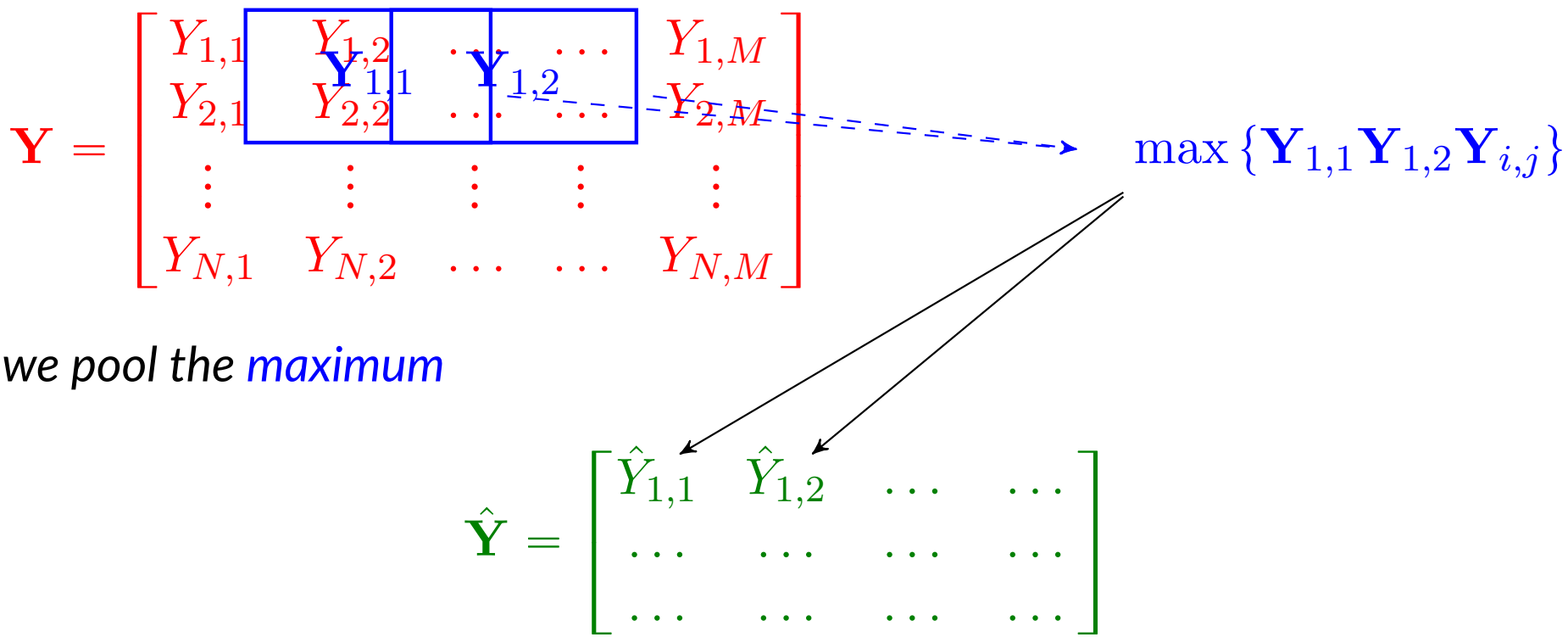


Building Neural Layers with Convolution



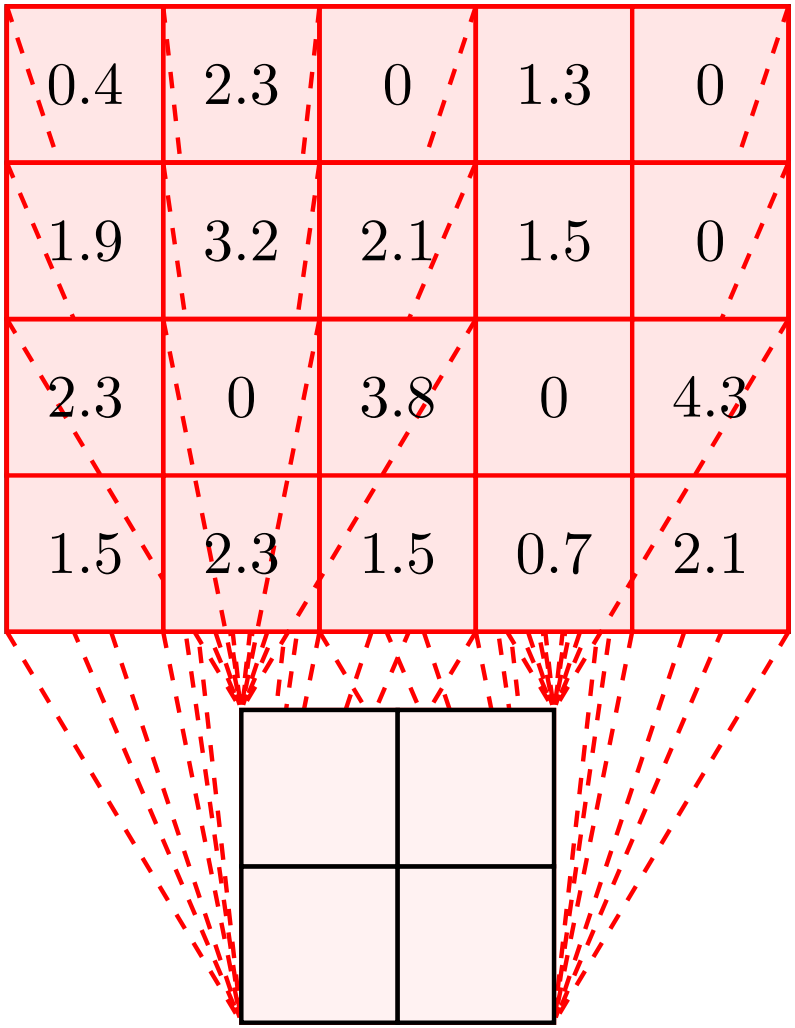
Max-Pooling

Pooling is another convolution like operation: *we compute a fix function in each slide, e.g., in max-pooling*



we pool the maximum

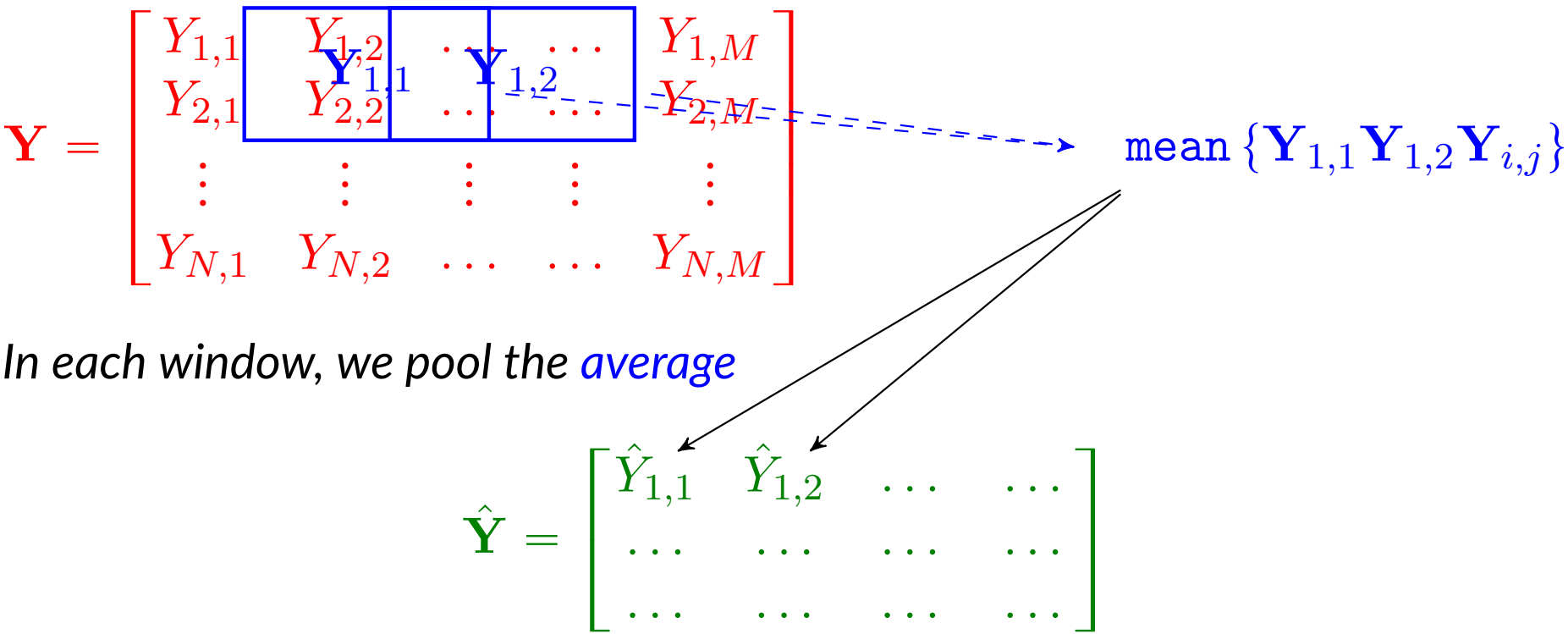
Max Pooling: *Numerical Example*



3.2	3.2	2.1	1.5
3.2	3.8	3.8	4.3
2.3	3.8	3.8	4.3

Mean-Pooling

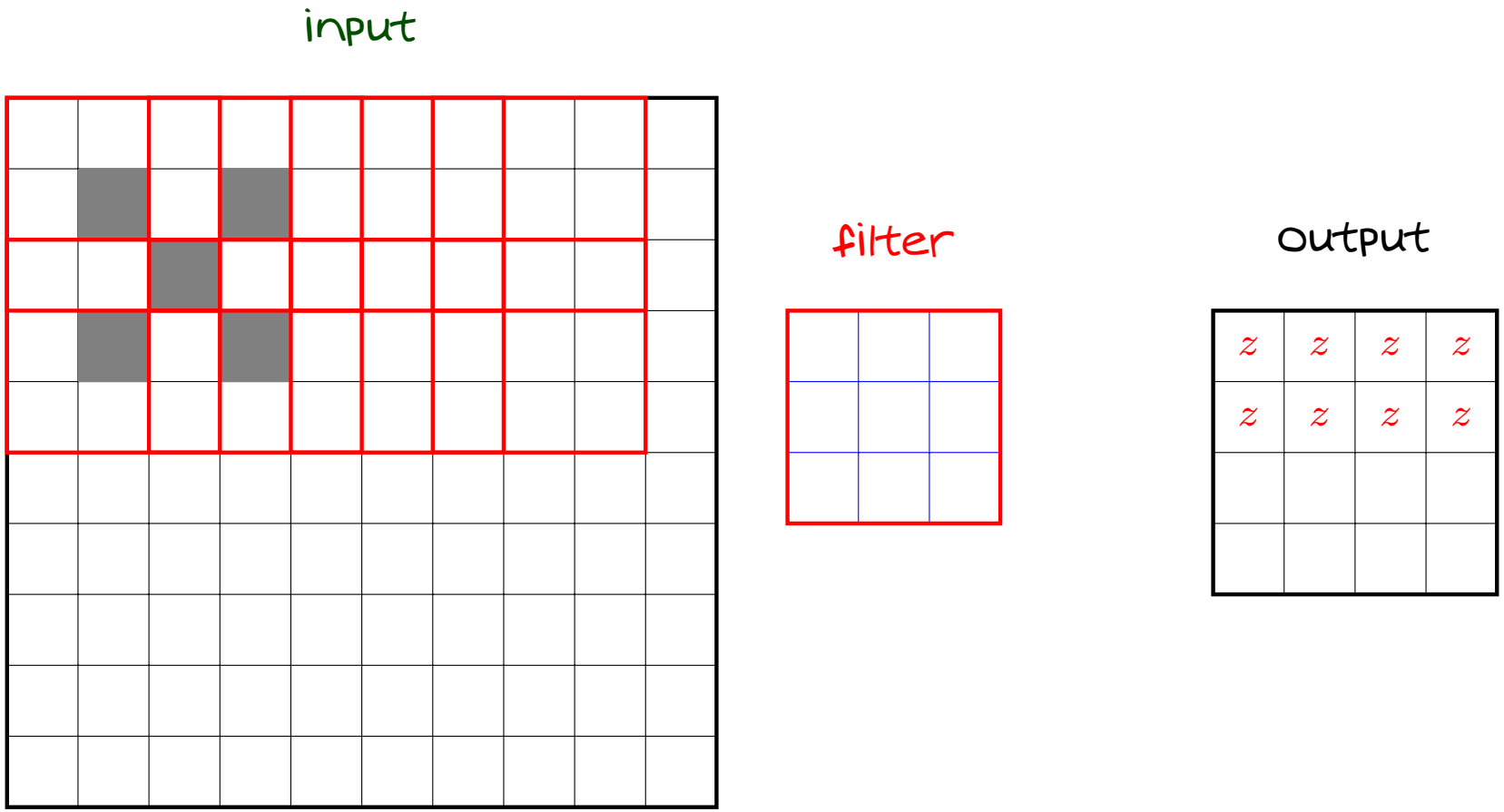
Mean-pooling is another approach in which we compute the average



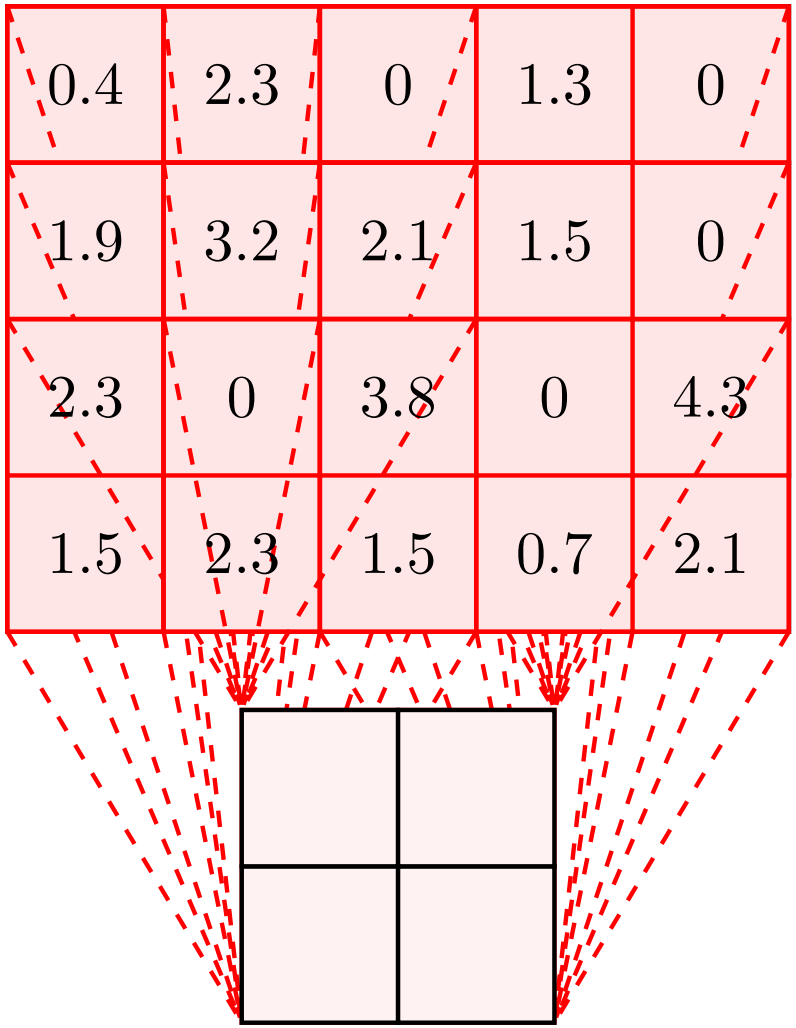
In each window, we pool the average

Convolution *with Stride*

We can perform all operations with stride, e.g., stride 2



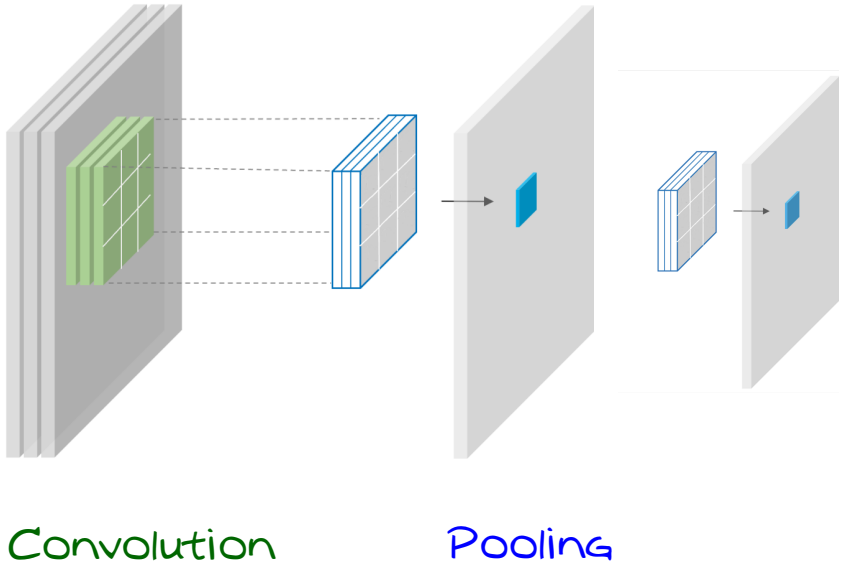
Mean-Pooling: *Numerical Example*



2.05	1.9	1.225	0.7
1.85	2.275	1.85	1.45
1.525	1.9	1.5	1.775

A Convolutional Unit

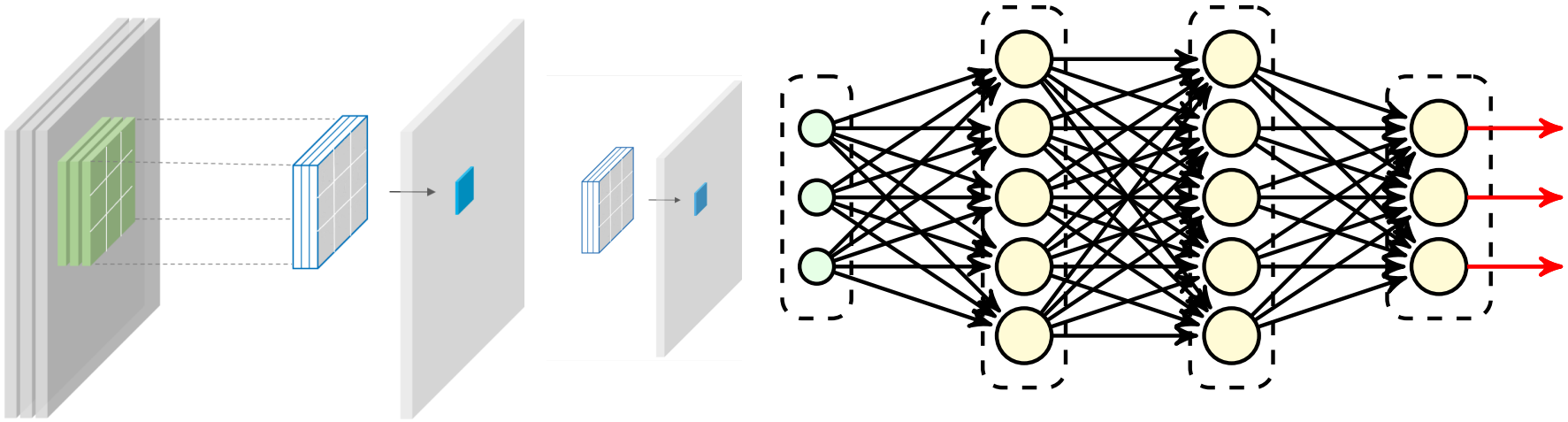
A convolutional unit is made as



- *The convolution performs less complex linear operation*
- *Pooling make the output smooth, i.e., less varying*

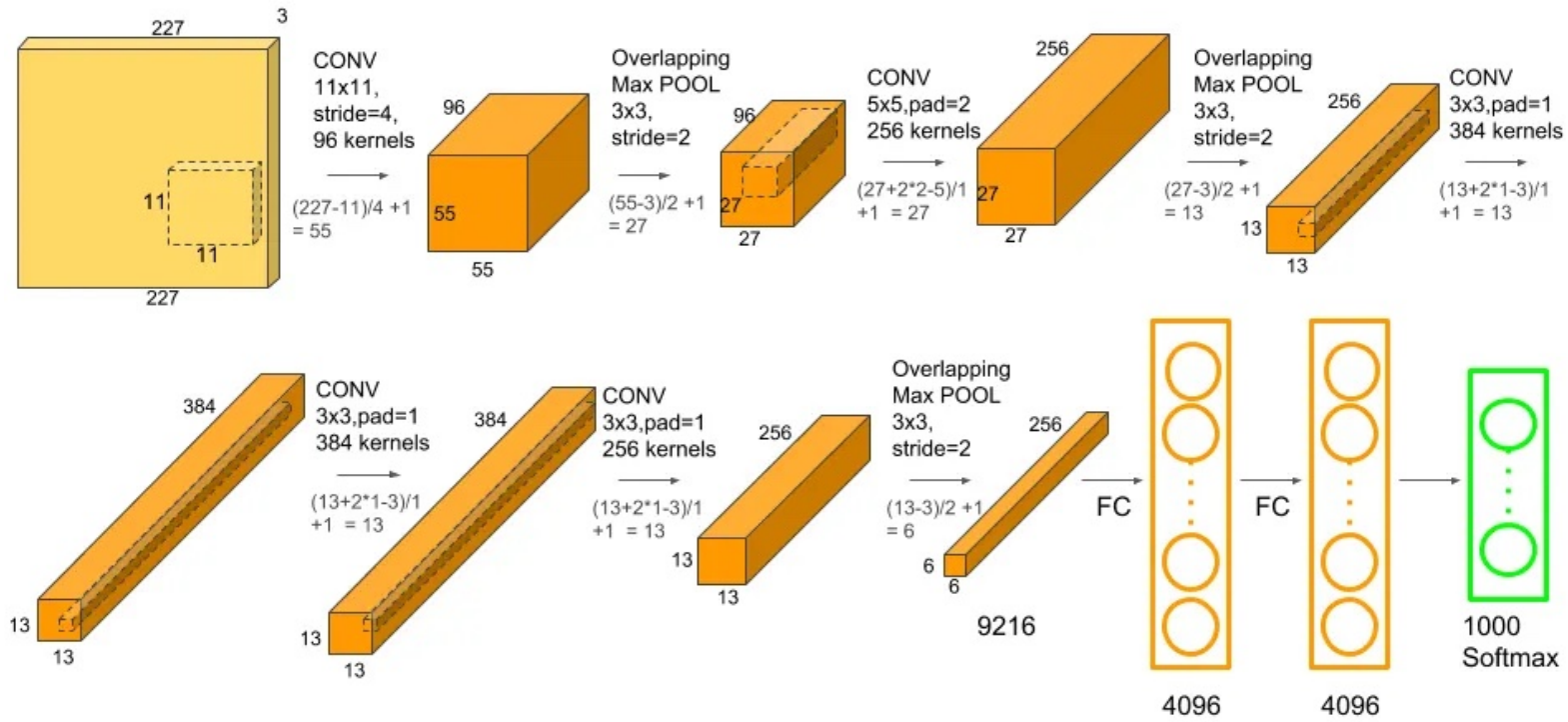
CNN: General Architecture

A simple CNN then looks like this

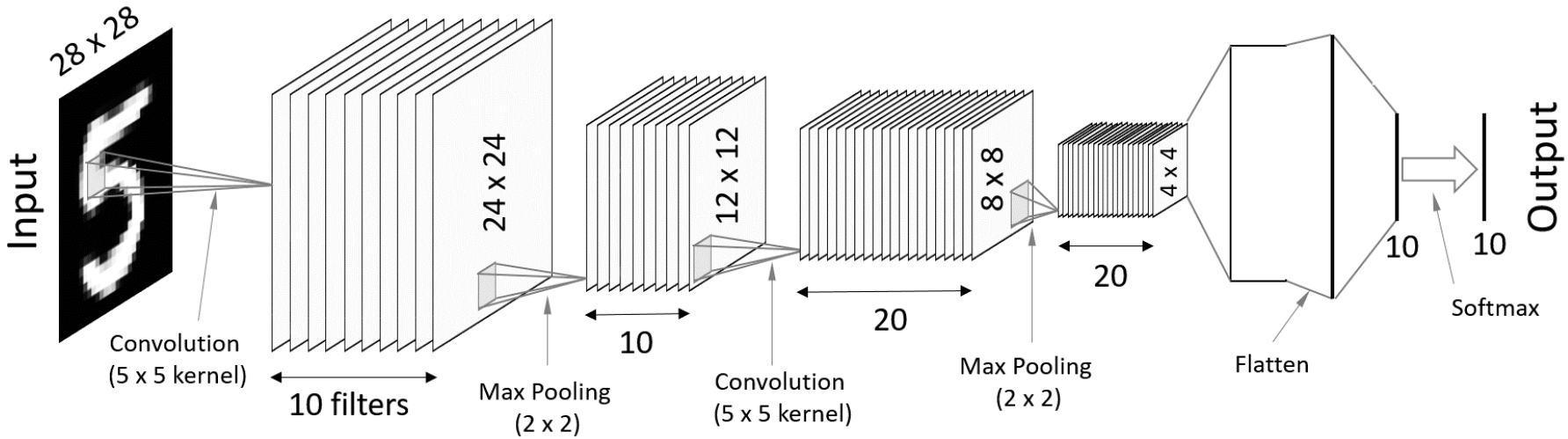


We can repeat the convolutional unit multiple times

Example: *AlexNet*



Example: *Custom CNN for MNIST Classification*



In this network, we do the following

- *We apply convolution with 10 filters*
  - ↳ *We apply pooling with stride 2*
- *We apply convolution with 20 filters*
  - ↳ *We apply pooling with stride 2*
- *We flatten and pass through one fully-connected layer*
  - ↳ *We compute Softmax for classification*

Further Read

- Goodfellow
  - ↳ Chapter 9