# tutorial 3A

|  |  |
|---|---|
| ↗ course | 🧠 Deep Learning |
| ⁘ Status | new |

## Question 1

Regularization and gradients revisited (Goodfellow et al., Section 7.1 ). Before introducing convolution, we continue to study explicit regularization in deep neural networks.

**a) Consider a loss function with L2 regularization:**

$$L(\theta) = L_{\text{data}}(\theta) + \lambda\|\theta\|_2^2$$

- Compute the gradient of $L(\theta)$ with respect to $\theta$.

$$\nabla_\theta L(\theta) = \nabla_\theta L_{\text{data}}(\theta) + \lambda \nabla_\theta(\theta^\top \theta)$$

$$= \nabla_\theta L_{\text{data}}(\theta) + 2\lambda\theta.$$

$\lambda$ is a hyperparameter called regularization strength. it controles how strongly we penelize large weights.

- Write the corresponding gradient descent update rule.

$$\theta \leftarrow \theta - \epsilon(\nabla_\theta L_{\text{data}}(\theta) + 2\lambda\theta)$$

where $\epsilon$ is the learning rate

sipplifie

$$\theta \leftarrow (1 - 2\epsilon\lambda)\theta - \epsilon\nabla_\theta L_{\text{data}}(\theta)$$

1. First shrink the weights

   $(1 - 2\epsilon\lambda)\theta$

→ this is weight decay

2. Then apply the normal gradient step from the data loss

So L2 regularization does two things:

- Fits the data
- Continuously pulls weights toward zero

That's why it reduces overfitting.

**(b) From the update rule:**
• How does L2 regularization modify parameter updates?

L2 regularization adds an extra term $2\lambda\theta$ to the gradient.

This produces **systematic shrinkage of the weights at every step**.

Concretely:

1. **Multiplicative shrinkage**

   The factor $(1 - 2\epsilon\lambda)$ scales the parameters down each iteration.

   Even if the data gradient were zero, the weights would still decay.

2. **Stronger effect on large weights**

   Since the penalty is proportional to $\theta$, larger weights receive larger shrinkage.

3. **Bias toward small-norm solutions**

   Optimization is pushed toward parameter vectors with smaller L2 norm.

4. **Smoother learned functions**

   Smaller weights reduce sensitivity to input fluctuations, limiting overfitting.

• Does it change the architecture of the network, or only the optimization trajectory?

The number of layers, number of units, activation functions, and connectivity all remain exactly the same. so it does not change the architechure of the network and conectively all remains the same.

how ever it does change the optimization trajectory:

- It modifies the gradient by adding $2\lambda\theta$.
- This alters how parameters move during training.
- It biases learning toward smaller weight magnitudes.

## Question 2

Why convolution? Structured representations for images (Goodfellow et al., Section 9.2 ).
Consider the MNIST dataset. Each image is represented as a vector in $\mathbb{R}^{784}$.

a) Images as structured signals. Conceptually examine digits such as 3, 5, and 9.

- **What local visual patterns (edges, curves, intersections) appear repeatedly?**

  For digits like **3, 5, and 9**, we repeatedly see:

  - Horizontal edges
  - Vertical edges
  - Curved strokes
  - Intersections or junctions
  - Small line segments

  These are **local patterns**. A small patch of pixels often contains meaningful structure independent of the rest of the image.

- **Do these patterns occur at fixed spatial positions?**

No.

A curved stroke or horizontal edge can appear:

- Slightly shifted left or right
- Higher or lower

- Thicker or thinner

The *pattern* is the same, but its **location changes**.

This means:

- The model should recognize a curve anywhere.
- It should not relearn the same curve separately at every possible position.

This motivates **weight sharing**, which convolution provides.

- **Transform a 3 into a 5, then into a 9. Which local modifications are required to transform one digit into another, and which visual components remain unchanged?**

Conceptually:

- **3 → 5**
    - Add a vertical stroke on the left.
    - Modify the top curve slightly.
    - Keep the lower curve similar.
- **5 → 9**
    - Close the upper loop.
    - Remove or weaken the lower horizontal stroke.
    - Keep the top curvature structure.

Key point:

Many **local components remain unchanged**:

- Curved segments
- Horizontal strokes
- Edge fragments

Digits differ by **local modifications**, not a complete reorganization of all pixels.

- Explain why treating each pixel as an independent input feature ignores important
  spatial structure.

Flattening the image into $\mathbb{R}^{784}$:

- Destroys 2D spatial relationships.
- Ignores that nearby pixels are highly correlated.
- Forces the network to learn separate weights for the same pattern at different positions.
- Leads to many redundant parameters.

(b) Fully connected layers and loss of structure.
Consider a fully connected layer mapping

$$\mathbb{R}^{784} \to \mathbb{R}^{H}$$

- **What does the dimension 784 represent?**

It is the number of input pixels in an MNIST image: 28 \times 28 = 784. After flattening, the image becomes a vector with one entry per pixel.

- **What does H represent?**

It is the number of hidden units (neurons) in the fully connected layer, so the output is an H-dimensional feature vector (one activation per hidden unit).

- **What structural information is lost when the image is reshaped into a vector?**

You lose the explicit 2D geometry: which pixels are neighbors, the notion of locality, and the fact that edges and curves are local patterns that can repeat anywhere. After flattening, adjacent entries in the vector are not guaranteed to be adjacent in the image in a meaningful way for learning spatial patterns.

**Using the terminology of Section 9.2:**

- **Does a fully connected layer enforce sparse interactions?**

No. Each hidden unit connects to all 784 pixels, so interactions are dense, not sparse.

- **Does it enforce parameter sharing?**

No. Every connection has its own weight. The model must learn separate weights for the same pattern at different locations.

- **Is it equivariant to translation?**

No. If you shift the digit a few pixels, the input vector changes in a way that does not produce a corresponding shift in the hidden activations. The response can change completely unless the network relearns that invariance from data.
**Explain why a fully connected architecture does not exploit spatial structure.**

Because it ignores locality and weight sharing: it treats a stroke in the top left as unrelated to the same stroke in the bottom right, so it wastes parameters and data learning the same detectors repeatedly.

## Question 3

Convolution as a structured linear operator (Goodfellow et al., Sections 9.1–9.2 ).
In this exercise, we formalize convolution as a linear operation with structural constraints.

a) Hand-compute a simple convolution.
Consider the 4 × 4 input imag

$$X = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 3 & 1 \\ 2 & 2 & 1 & 0 \\ 1 & 0 & 2 & 2 \end{pmatrix}$$

$$K = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$$

Assume stride = 1 and no padding.

$$Y_{ij} = \sum_{a=1}^{2} \sum_{b=1}^{2} X_{i+a-1,\, j+b-1}\, K_{ab}, \qquad Y \in \mathbb{R}^{3\times 3}.$$

$$Y_{11} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 1 \cdot 1 + 2 \cdot (-1) + 0 \cdot 0 + 1 \cdot 1 = 0$$

$$Y_{12} = \begin{pmatrix} 2 & 0 \\ 1 & 3 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 2 \cdot 1 + 0 \cdot (-1) + 1 \cdot 0 + 3 \cdot 1 = 5$$

$$Y_{13} = \begin{pmatrix} 0 & 1 \\ 3 & 1 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 0 \cdot 1 + 1 \cdot (-1) + 3 \cdot 0 + 1 \cdot 1 = 0$$

$$Y_{21} = \begin{pmatrix} 0 & 1 \\ 2 & 2 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 0 \cdot 1 + 1 \cdot (-1) + 2 \cdot 0 + 2 \cdot 1 = 1$$

$$Y_{22} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 1 \cdot 1 + 3 \cdot (-1) + 2 \cdot 0 + 1 \cdot 1 = -1$$

$$Y_{23} = \begin{pmatrix} 3 & 1 \\ 1 & 0 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 3 \cdot 1 + 1 \cdot (-1) + 1 \cdot 0 + 0 \cdot 1 = 2$$

$$Y_{31} = \begin{pmatrix} 2 & 2 \\ 1 & 0 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 2 \cdot 1 + 2 \cdot (-1) + 1 \cdot 0 + 0 \cdot 1 = 0$$

$$Y_{32} = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 2 \cdot 1 + 1 \cdot (-1) + 0 \cdot 0 + 2 \cdot 1 = 3$$

$$Y_{33} = \begin{pmatrix} 1 & 0 \\ 2 & 2 \end{pmatrix} : \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix} = 1 \cdot 1 + 0 \cdot (-1) + 2 \cdot 0 + 2 \cdot 1 = 3$$

$$X * K = \begin{pmatrix} 0 & 5 & 0 \\ 1 & -1 & 2 \\ 0 & 3 & 3 \end{pmatrix}$$

the special output is

$(4 - 2 + 1) \times (4 - 2 + 1) = 3 \times 3.$

**(b) Sparse interactions and parameter sharing.**

- How many input pixels influence one output value?

Each output value $Y_{ij}$ is computed from a $2 \times 2$ patch of the input (because the kernel is $2 \times 2$).

So the number of input pixels that influence one output value is:

$2 \times 2 = 4$.

Only those 4 pixels in the local receptive field affect that single output entry.

- How many parameters does the filter contain?

There are **4 weights** in the filter.

- How many distinct parameters are used across the entire output map?

Even though the output is $3 \times 3 = 9$ values, the **same 4 parameters** are reused at every spatial position.

So the total number of distinct learnable parameters remains 4.

**Explain mathematically why convolution enforces:**

- sparse interactions,

Each output entry is

$$Y_{ij} = \sum_{a=1}^{2} \sum_{b=1}^{2} X_{i+a-1,\, j+b-1} K_{ab}.$$

Only 4 input pixels contribute to each $Y_{ij}$, not all 16 pixels of X.

Therefore the linear operator connecting input to output has many zeros.

Most input pixels do **not** affect a given output value.

That is sparsity.

- parameter sharing.

The same kernel K_{ab} appears in every spatial location:

$$Y_{ij} = \sum_{a,b} X_{i+a-1,\, j+b-1} K_{ab}.$$

Notice:

- $K_{ab}$ does **not** depend on $i, j$.
- The weights are reused everywhere.

Thus one set of 4 parameters is applied across all 9 output positions.

That reuse of identical weights across space is parameter sharing.

**(c) Translation equivariance.**
If the input image shifts one pixel to the right:

- How does the output activation map change?

translating the input and then convolving gives the same result as convolving first and then translating the output by the same amount. This holds because of **weight sharing** and **the same dot product being computed at every spatial location**.

**(d) Hypothesis space and inductive bias.**
Convolution restricts the class of functions the network can represent. This restriction defines a smaller hypothesis space.

- **What is meant by hypothesis space in this context?**

It is the set of all input to output functions the model can represent given its architecture and parameterization. For a conv layer, this is the set of linear maps from images to feature maps that can be written using a small kernel applied everywhere with shared weights.

- **What inductive bias does convolution encode about images?**

It is the set of all input to output functions the model can represent given its architecture and parameterization. For a conv layer, this is the set of linear maps from images to feature maps that can be written using a small kernel applied everywhere with shared weights.

- **Why can restricting the hypothesis space improve generalization?**

Because it removes many functions that fit the training set but do not match how images are actually structured. With fewer degrees of freedom, the model needs less data to learn stable patterns and is less likely to overfit noise.

## Question 4

Receptive fields and hierarchical representation (Goodfellow et al., Sections 9.1–9.2 ).
Assume stride = 1, no padding, and square filters of size $k \times k$.
**(a) For a single convolutional layer:**

- What is the receptive field of one output unit?

The receptive field of one output unit is exactly the input patch it looks at:

$$R_1 = k \times k$$

So one output value depends on a $k \times k$ block of input pixels.

**(b) For two consecutive layers with filter size k:**

- What is the receptive field size $R_2$?

Extend to $L$ layers and derive:

$$R_L = 1 + L(k-1).$$

Each unit in layer 2 looks at a $k \times k$ patch of layer 1 units, and each of those layer 1 units already sees a $k \times k$ patch of the input. With stride 1, the receptive field grows by k-1 per layer:

$$R_2 = 1 + 2(k-1)$$

That is a receptive field of spatial size $R_2 \times R_2$ in the original input.

Extend to L layers:

$$R_L = 1 + L(k-1)$$

Again the receptive field is $R_L \times R_L$.

**(c) Why does increasing depth increase the spatial context available to higher-level**
**features? Explain how this enables hierarchical representations.**

Each added conv layer expands the receptive field, so deeper units integrate information from a larger region of the original image. Early layers learn local primitives like edges and corners, while later layers combine them into larger motifs like curves, parts of digits, and eventually full digit shapes. This works because composition across layers builds complex features from simpler ones while keeping parameter sharing and locality.

## Question 5

1. Pooling and invariance (Goodfellow et al., Section 9.3 ).

**(a) Consider $2 \times 2$ max-pooling with stride $2$.**

- What operation is computed?

You slide a 2×2 window over the input, moving 2 pixels at a time (non overlapping), and output the **maximum** value in each window:

$$y_{i,j} = \max\{x_{2i,2j},\ x_{2i,2j+1},\ x_{2i+1,2j},\ x_{2i+1,2j+1}\}.$$

- How does it affect spatial resolution?

It downsamples by a factor of 2 in each spatial dimension.
If the input is $H \times W$, the output is
$\left\lfloor \frac{H}{2} \right\rfloor \times \left\lfloor \frac{W}{2} \right\rfloor$
(often exactly $H/2 \times W/2$ when $H, W$ are even).

Compare pooling and convolution in terms of:

- learned vs fixed,

Convolution learns its filter weights from data.
Max pooling has no learned parameters; it is fixed.

- linear vs nonlinear,

Convolution is linear in the input.
Max pooling is nonlinear because the max operator is nonlinear.

- information preservation.

Convolution can preserve a lot of information (especially with small stride and enough channels); it transforms but doesn't inherently discard spatial samples.

Max pooling **throws away** information: you keep only the strongest activation per 2×2 region and lose exact positions and non max values.

**(b) Suppose a convolutional layer detects vertical edges.**

- **How does a small shift in the input affect activations?**

A small shift in the input shifts the activations by the same amount. The edge is still detected, just at a neighboring spatial location. That is translation equivariance.

- **How does max-pooling modify this behavior?**

Max pooling makes the response less sensitive to small shifts. If the edge moves within the same $2 \times 2$ pooling window, the pooled output stays the same because the max is unchanged. Only when the edge crosses into a different pooling window does the pooled activation change.

**Explain the difference between equivariance and invariance.**

Equivariance means transforming the input causes a predictable transform of the output, like a shift in gives a shift out. Invariance means transforming the input does not change the output at all, at least for the range of transformations the system is invariant to.

## Question 6

CNNs as structural regularization (Goodfellow et al., Sections 9.2 and 7.1 ).
**(a) Parameter count comparison.**
For a $28 \times 28$ image:

- Parameters in $\mathbb{R}^{784} \to \mathbb{R}^{100}$ fully connected layer?

Weights: $784 \times 100 = 78{,}400$
Biases: 100
Total parameters: $78{,}400 + 100 = 78{,}500$

- Parameters in 100 filters of size $5 \times 5$?

Per filter: weights $5 \times 5 = 25$, plus 1 bias $\Rightarrow 26$
Total parameters: $100 \times 26 = 2{,}600$

Explain the structural reason for the difference.
**(b) Structural vs explicit regularization.**
Compare:

- L2 regularization,
- convolution.

For each:

- Does it restrict magnitude, connectivity, or functional form?
- Does it modify optimization trajectory or hypothesis space?

## L2 regularization

What it does

It restricts **magnitude**: it penalizes large weights, pushing parameters toward small norm solutions.

What it changes

It mainly changes the **optimization trajectory** by adding 2\lambda\theta to the gradient. Indirectly it also shrinks the effective set of functions you end up with, but the architecture's connectivity is unchanged.

So in the requested terms

Magnitude: yes

Connectivity: no

Functional form: weakly, only through smaller weights

Optimization trajectory: yes

Hypothesis space: not explicitly (architecture unchanged), but effectively biased toward smoother, lower norm functions

## Convolution

What it does

It restricts **connectivity and functional form** by design: local receptive fields (sparse interactions) and shared parameters (same filter everywhere). That forces the layer to be a translation equivariant pattern detector.

What it changes

It directly changes the **hypothesis space** because many linear maps are not representable by a conv layer. Training still follows gradients, but within this smaller, structured function class.

So in the requested terms

Magnitude: not inherently

Connectivity: yes (local, sparse)

Functional form: yes (shared weights, translation equivariance)

Optimization trajectory: not the core effect (though it can still affect it)

Hypothesis space: yes, explicitly restricted by architecture

**(c) Generalization and alignment.**
If we train:

- a fully connected network with strong L2,
- a convolutional network with minimal explicit regularization,

**which would generalize better on images?**
Justify using:

- inductive bias,
- effective hypothesis space,
- alignment with image structure.

The convolutional network will usually generalize better on images.

convolution hard codes locality and translation equivariance through local receptive fields and shared weights, so the model is pushed to learn reusable detectors for edges and strokes wherever they appear. L2 only penalizes large weights and does not tell the model that “the same pattern in a different location should be treated similarly”.

a conv layer can only represent functions that are built from the same small filter applied everywhere, so there are far fewer degrees of freedom to fit arbitrary noise. A fully connected layer can represent many more input to output mappings, and strong L2 shrinks them but does not remove those mappings from what the model can express.

MNIST like images are made of local motifs that repeat and shift slightly, so locality plus weight sharing matches the data generating structure. Fully connected models ignore 2D neighborhood structure after flattening, so they waste parameters relearning the same motif at many positions and typically need more data to generalize.

**(d) Limits of convolution.**
Give an example of a task where convolution might be an inappropriate inductive bias.

Convolution is a bad inductive bias when the target function is not driven by local, translation equivariant patterns.

so for example convolution networks would be inappropriate to detect if there is a small dot exactly at the top left corner or to detect symmetry.