

# Introduction to Deep Learning and Transfer Learning



# Course organisation

## Sessions

- 1 Intro Deep Learning,
- 2 Data Augmentation and Self Supervised Learning,
- 3 Quantization,
- 4 Pruning,
- 5 Factorization,
- 6 Distillation,
- 7 Embedded Software and Hardware for DL,
- 8 Presentations for challenge.

2024-02-06

## Introduction to Deep Learning and Transfer Learning

### Course organisation

Course organisation

Sessions

- Intro Deep Learning,
- Data Augmentation and Self Supervised Learning,
- Quantization,
- Pruning,
- Factorization,
- Distillation,
- Embedded Software and Hardware for DL,
- Presentations for challenge.

Introduction to Deep Learning and Transfer Le

2 / 25

# Course organisation

## Sessions

- 1 Intro Deep Learning,
- 2 Data Augmentation and Self Supervised Learning,
- 3 Quantization,
- 4 Pruning,
- 5 Factorization,
- 6 Distillation,
- 7 Embedded Software and Hardware for DL,
- 8 Presentations for challenge.

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ Course organisation

Course organisation

Sessions

- Intro Deep Learning,
- Data Augmentation and Self Supervised Learning,
- Quantization,
- Pruning,
- Factorization,
- Distillation,
- Embedded Software and Hardware for DL,
- Presentations for challenge.

A set of small, semi-transparent navigation icons typically used in Beamer presentations for navigating between slides and sections.

Introduction to Deep Learning and Transfer Le

2 / 25

# Global formalism

## Input/output

- **Goal:** infer a function from an input (often tensor) space to an output (often tensor) space,  $y = f(x)$ ,
- **Example:** input can be an image, output a vector where the largest value indicate the category the image belongs to.

## Error/Loss

- **Loss  $\mathcal{L}$ :** nonnegative measure of the discrepancy between expected output  $\hat{y}$  and obtained output  $y$ .
- **Example:** output should be  $[0, 1]$  but is  $[0.2, 0.8]$ .

## Parameters

- $f = f_w$  contains **parameters  $W$**  to be trained,
- In most cases, an ideal  $f_w$  exists but is **hard to find in practice**,
- Learning is a **regression ill-posed problem**.

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### └ Global formalism

Loss: it's a way to tell the model when it is wrong and train the model accordingly. The model contains parameters (model weights and bias) and usually, given a task, an optimal set of parameters exist but again finding it is ill posed problem (many solutions exists)

## Global formalism

### Input/output

- **Goal:** Infer a function from an input (often tensor) space to an output (often tensor) space,  $y = f(x)$ ,
- **Example:** input can be an image, output a vector where the largest value indicate the category the image belongs to.

### Error/Loss

- **Loss  $\mathcal{L}$ :** nonnegative measure of the discrepancy between expected output  $\hat{y}$  and obtained output  $y$ .
- **Example:** output should be  $[0, 1]$  but is  $[0.2, 0.8]$ .

### Parameters

- $f = f_w$  contains parameters  $W$  to be trained,
- In most cases, an ideal  $f_w$  exists but is hard to find in practice,
- Learning is a regression ill-posed problem.

# Global formalism

## Input/output

- **Goal:** infer a function from an input (often tensor) space to an output (often tensor) space,  $y = f(x)$ ,
- **Example:** input can be an image, output a vector where the largest value indicate the category the image belongs to.

## Error/Loss

- **Loss  $\mathcal{L}$ :** nonnegative measure of the discrepancy between expected output  $\hat{y}$  and obtained output  $y$ .
- **Example:** output should be [0, 1] but is [0.2, 0.8].

## Parameters

- $f = f_w$  contains **parameters  $W$**  to be trained,
- In most cases, an ideal  $f_w$  exists but is **hard to find in practice**,
- Learning is a **regression ill-posed problem**.

# Introduction to Deep Learning and Transfer Learning

2024-02-06

## └ Global formalism

Loss: it's a way to tell the model when it is wrong and train the model accordingly. The model contains parameters (model weights and bias) and usually, given a task, an optimal set of parameters exist but again finding it is ill posed problem (many solutions exists)

### Global formalism

#### Input/output

- **Goal:** Infer a function from an input (often tensor) space to an output (often tensor) space,  $y = f(x)$ ,
- **Example:** input can be an image, output a vector where the largest value indicate the category the image belongs to.

#### Error/Loss

- **Loss  $\mathcal{L}$ :** nonnegative measure of the discrepancy between expected output  $\hat{y}$  and obtained output  $y$ .
- **Example:** output should be [0, 1] but is [0.2, 0.8].

#### Parameters

- $f = f_w$  contains parameters  $W$  to be trained,
- In most cases, an ideal  $f_w$  exists but is hard to find in practice,
- Learning is a regression ill-posed problem.

# Global formalism

## Input/output

- **Goal:** infer a function from an input (often tensor) space to an output (often tensor) space,  $y = f(x)$ ,
- **Example:** input can be an image, output a vector where the largest value indicate the category the image belongs to.

## Error/Loss

- **Loss  $\mathcal{L}$ :** nonnegative measure of the discrepancy between expected output  $\hat{y}$  and obtained output  $y$ .
- **Example:** output should be [0, 1] but is [0.2, 0.8].

## Parameters

- $f = f_w$  contains **parameters  $W$**  to be trained,
- In most cases, an ideal  $f_w$  exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

# Introduction to Deep Learning and Transfer Learning

2024-02-06

## └ Global formalism

Loss: it's a way to tell the model when it is wrong and train the model accordingly. The model contains parameters (model weights and bias) and usually, given a task, an optimal set of parameters exist but again finding it is ill posed problem (many solutions exists)

## Global formalism

### Input/output

- **Goal:** Infer a function from an input (often tensor) space to an output (often tensor) space,  $y = f(x)$ ,
- **Example:** input can be an image, output a vector where the largest value indicate the category the image belongs to.

### Error/Loss

- **Loss  $\mathcal{L}$ :** nonnegative measure of the discrepancy between expected output  $\hat{y}$  and obtained output  $y$ .
- **Example:** output should be [0, 1] but is [0.2, 0.8].

### Parameters

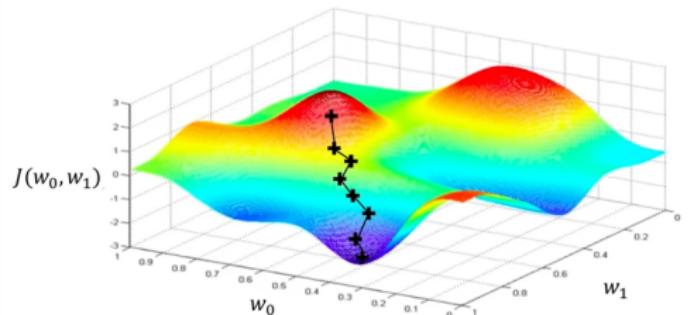
- $f = f_w$  contains **parameters  $W$**  to be trained,
- In most cases, an ideal  $f_w$  exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

# Global formalism

- Loss:  $J(\mathbf{W}) = \sum_i \mathcal{L}(f(\mathbf{x}^{(i)}, \mathbf{W}), \mathbf{y}^{(i)})$ ,  $i = \text{examples}$
- Model parameters:  $\mathbf{W}^* = \text{argmin}(J(\mathbf{W}))$

## Training Algorithm

- Randomly Initialize model weights
- Compute Gradient of the Loss  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Update weights  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Repeat until convergence



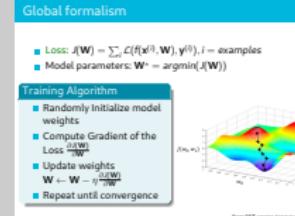
from MIT course [introtodeeplearning.com](http://introtodeeplearning.com)

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### └ Global formalism

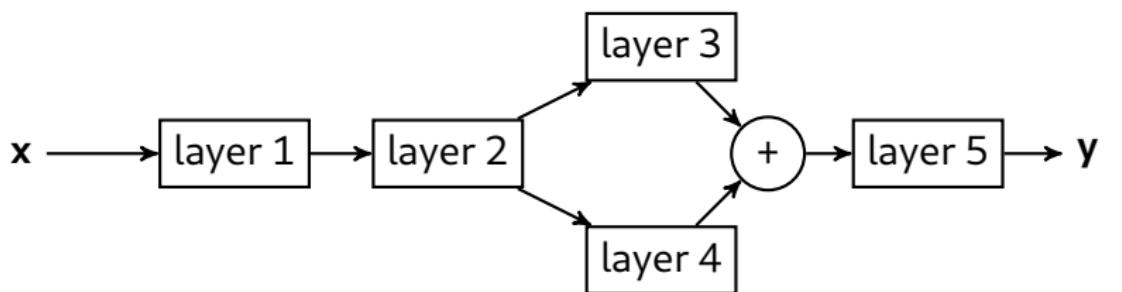
The total loss  $J$  (Empirical Risk, Objective function) is the average of Loss for each input/example and the optimal model parameters are those that minimize it. But how to find them? In other words, how to train the model? Here is a simplified description of the training algorithm at the base of modern DL, gradient descent. Repeat until reaching a local minimum (as illustrated in the figure for a simple example where we have only 2 parameters. We'll see that the function becomes much more complicated for millions of parameters -modern neural networks.)



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



2024-02-06

## Introduction to Deep Learning and Transfer Learning

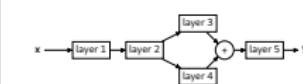
### Deep learning

DL at its core is the ability to learn higher and higher level representations or features from data in an end-to-end fashion. How? By means of a compositional approach of simple mathematical functions (layers). Representations are useful to interpret data: ideally the final representation should be easy to deal with (to classify, to generate data from...). What is new about DL is that we do that in an end-to-end fashion starting from raw data. Also, in DL, we use deep architectures with hidden layers to approximate any complex function  $f$ . So the fundamental blocks of NN are layers, each layer has its own parameters (weights and bias) that need to be trained.

Deep learning

Main idea

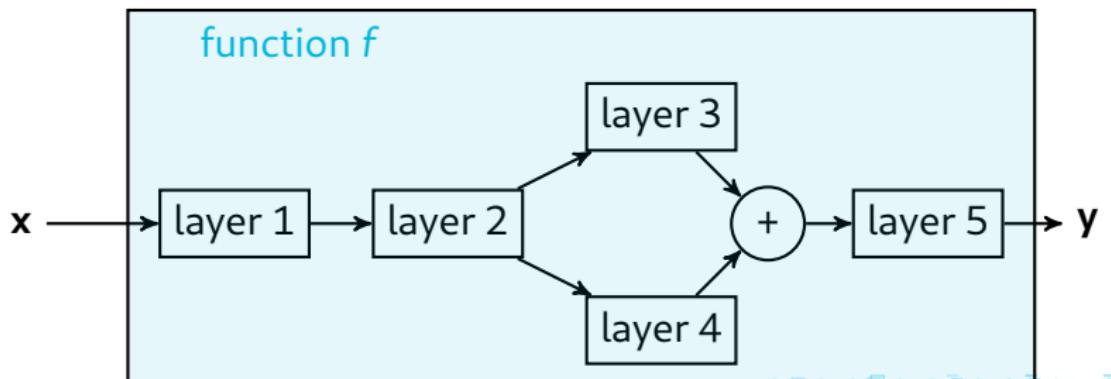
- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



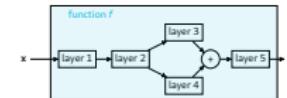
## 2024-02-06 Introduction to Deep Learning and Transfer Learning

### Deep learning

Deep learning

Main idea

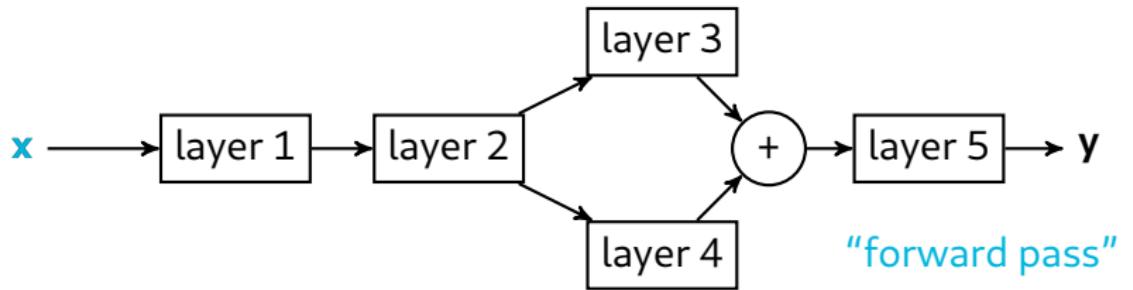
- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

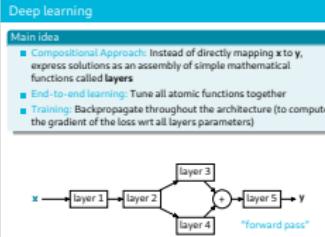


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

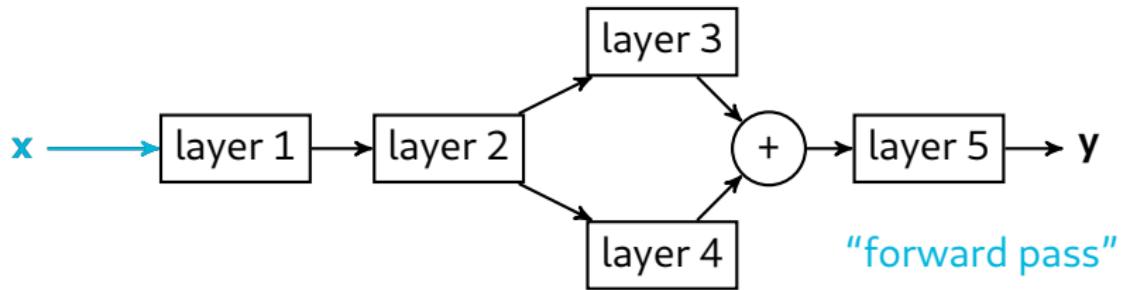
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

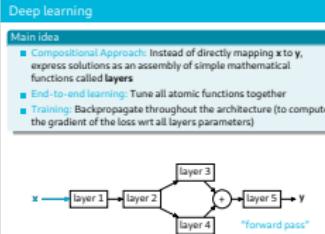


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

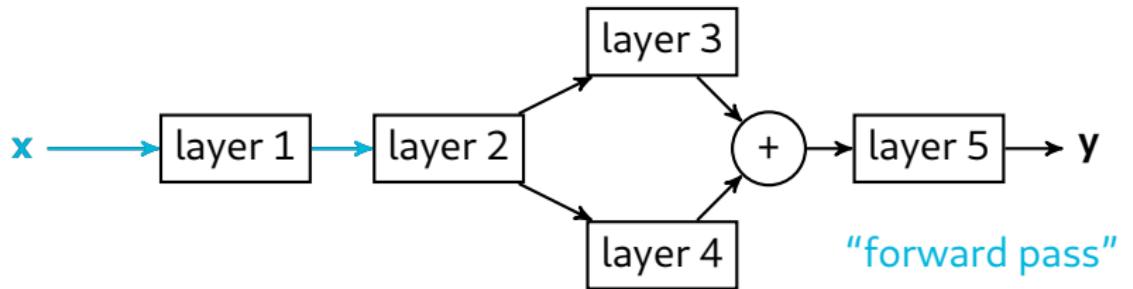
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

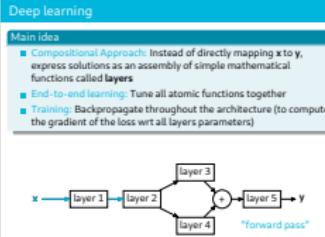


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

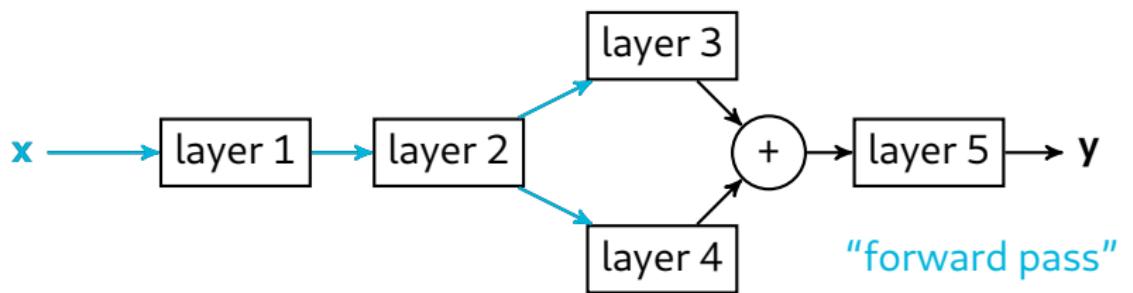


- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

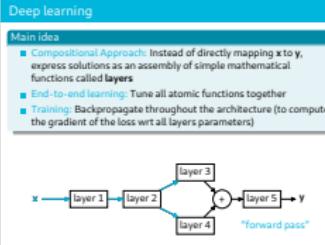


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

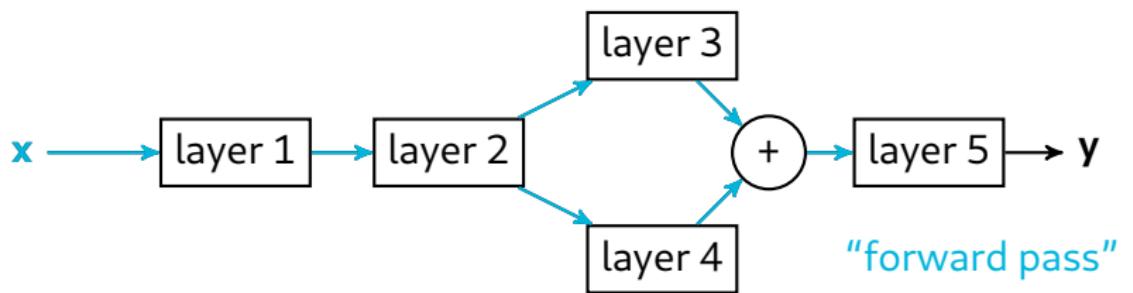
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

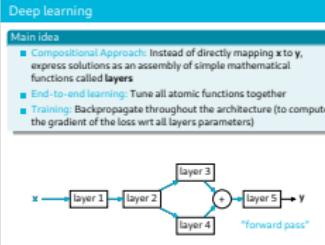


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

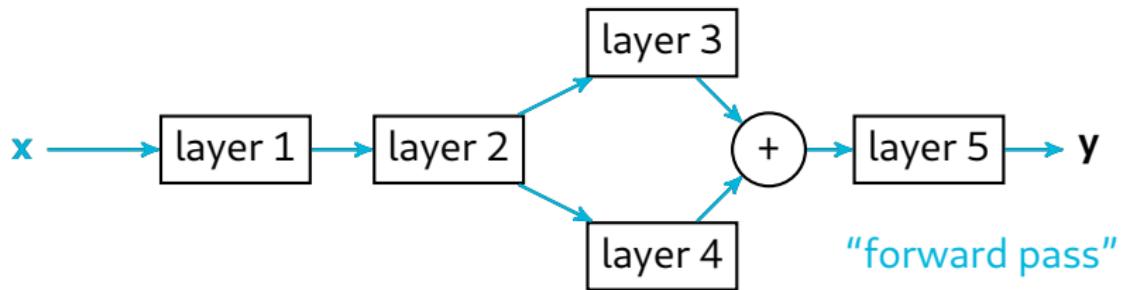


- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

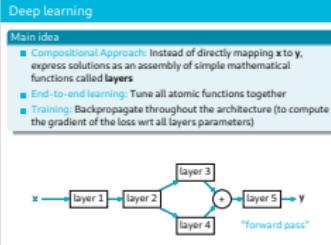


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

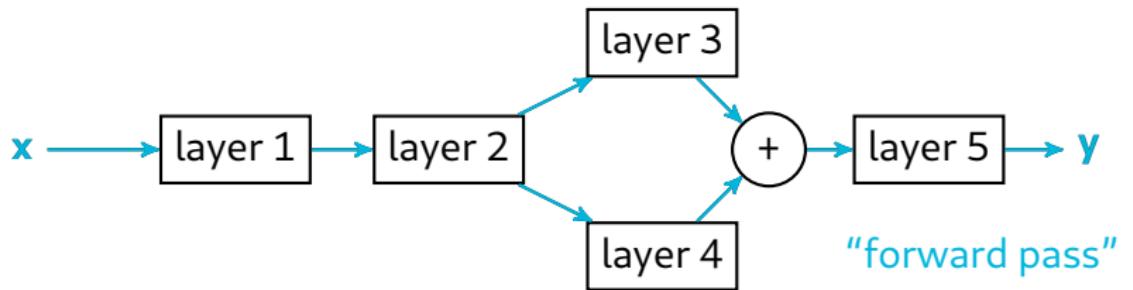


- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

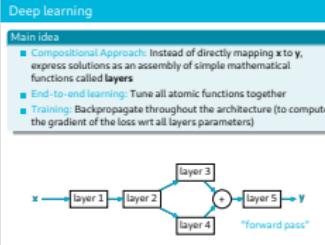


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

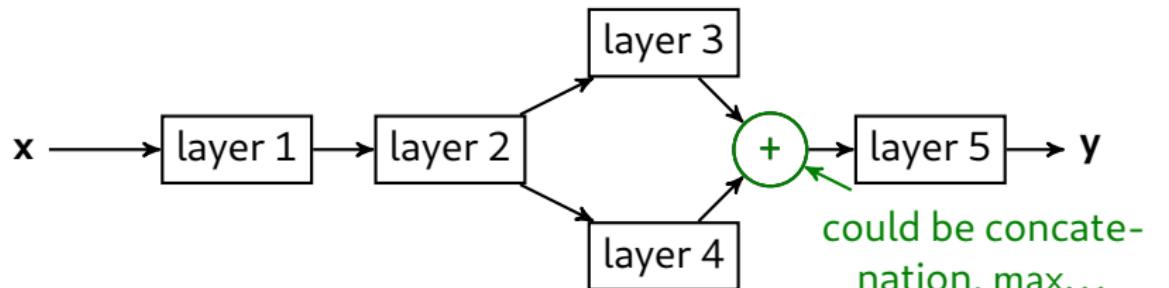


- Deep learning**
- Main idea**
- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
  - **End-to-end learning:** Tune all atomic functions together
  - **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

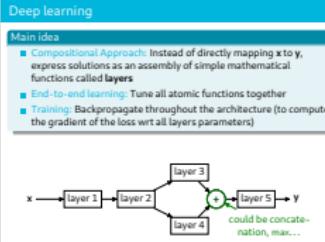


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

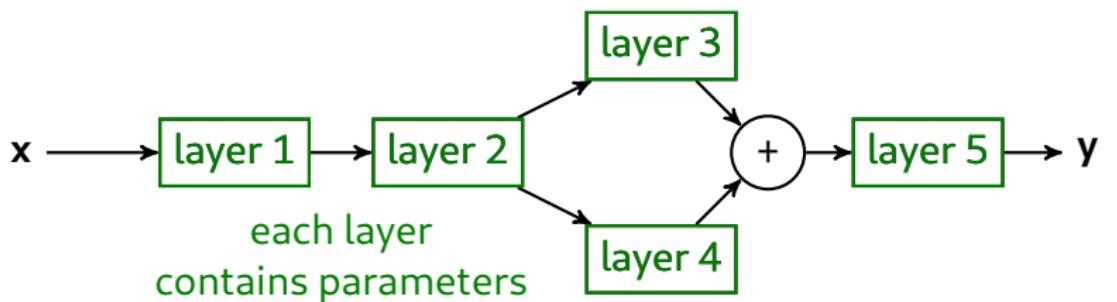
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

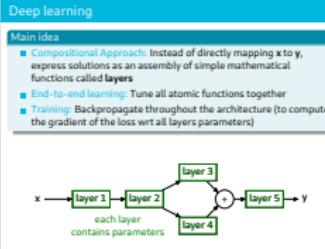
## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



## 2024-02-06 Introduction to Deep Learning and Transfer Learning

### Deep learning

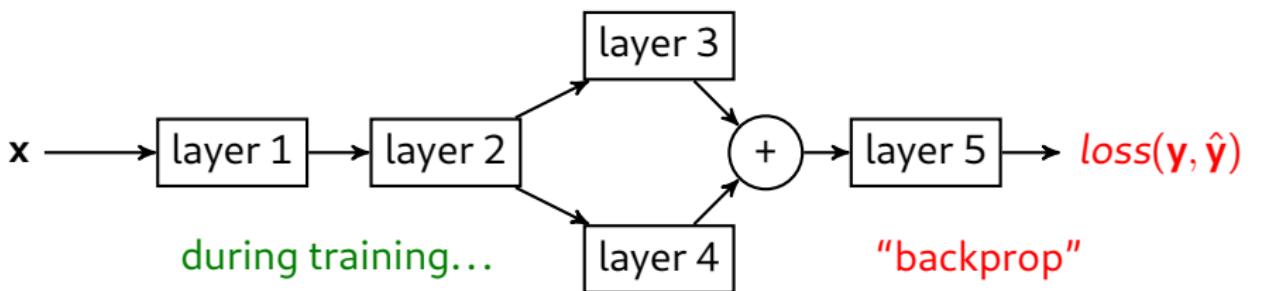


As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

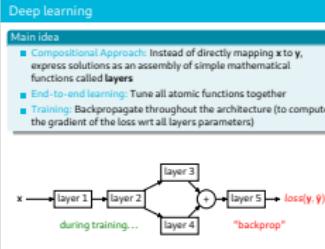


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

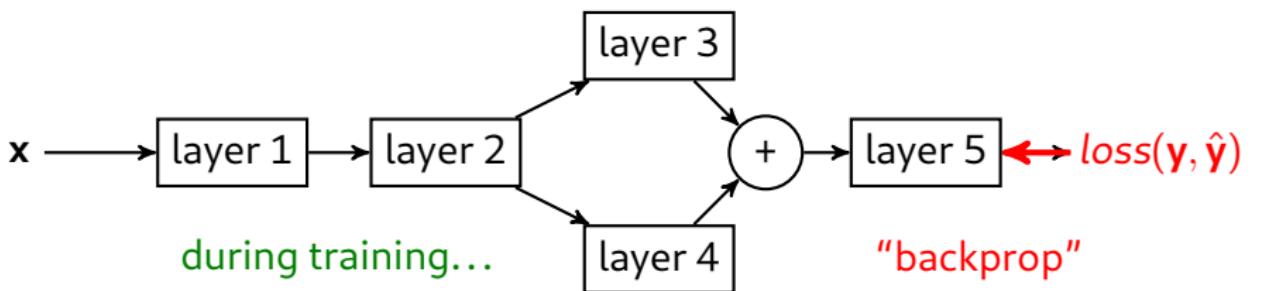
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

## Main idea

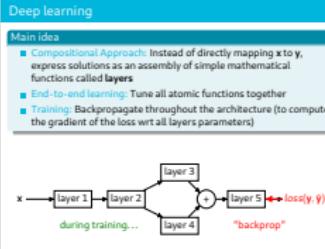
- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning



As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

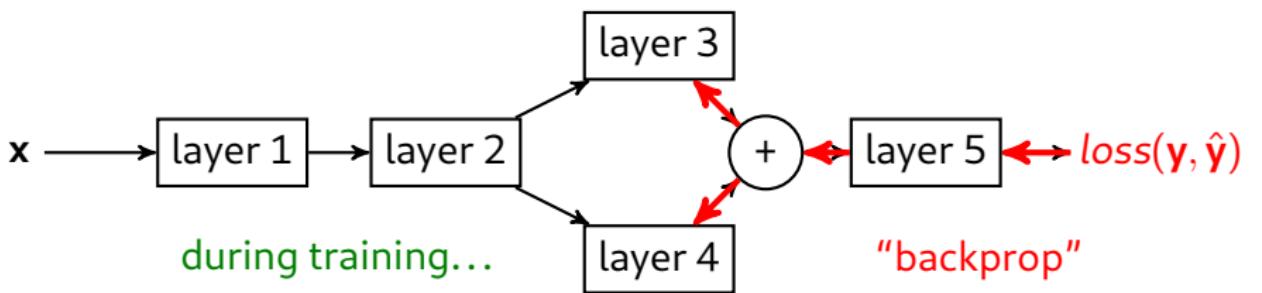


"backprop"

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

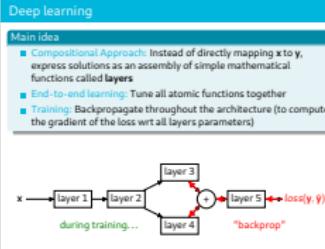


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

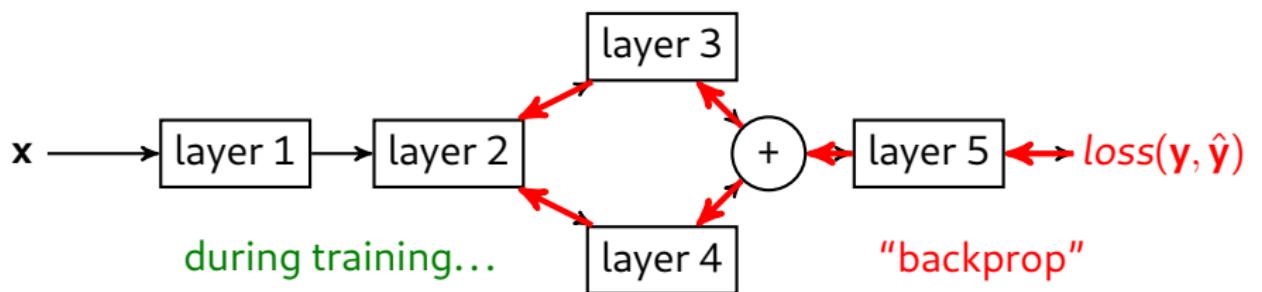
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)

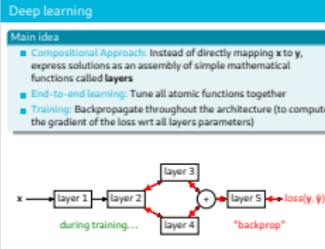


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

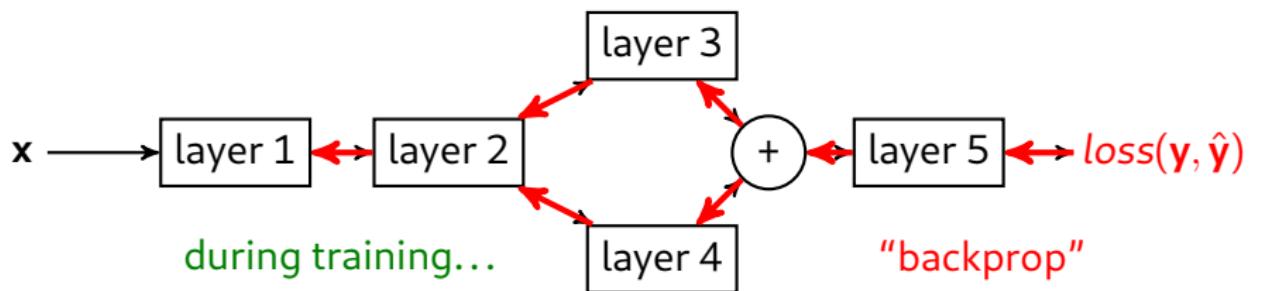
As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



## 2024-02-06 Introduction to Deep Learning and Transfer Learning

### Deep learning

Deep learning
<ul style="list-style-type: none"><li>■ <b>Compositional Approach:</b> Instead of directly mapping <math>x</math> to <math>y</math>, express solutions as an assembly of simple mathematical functions called <b>layers</b></li><li>■ <b>End-to-end learning:</b> Tune all atomic functions together</li><li>■ <b>Training:</b> Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)</li></ul>

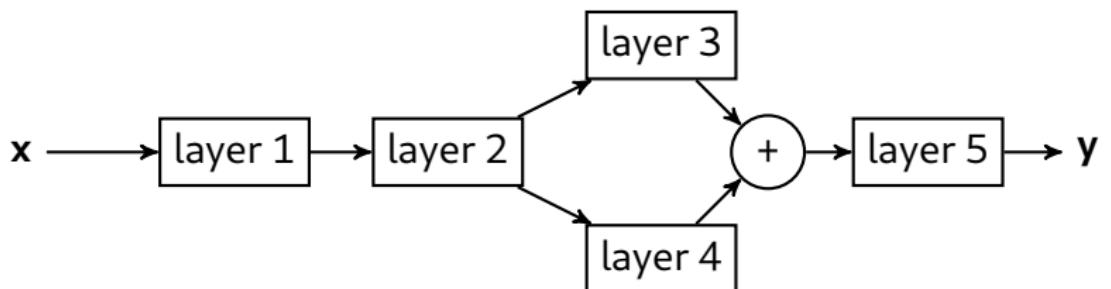


As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.

# Deep learning

## Main idea

- **Compositional Approach:** Instead of directly mapping  $x$  to  $y$ , express solutions as an assembly of simple mathematical functions called **layers**
- **End-to-end learning:** Tune all atomic functions together
- **Training:** Backpropagate throughout the architecture (to compute the gradient of the loss wrt all layers parameters)



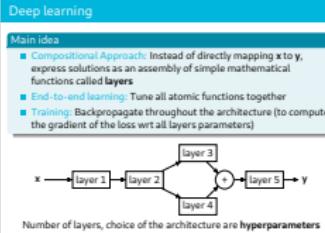
Number of layers, choice of the architecture are **hyperparameters**

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Deep learning

As we have seen the model is trained by calculating the gradient of the loss wrt to each layer parameters. How do we calculate gradients? Using backpropagation: efficient way to compute the gradient of the loss with respect to different layers parameters, using the derivative chain rule. The model weights are then updated with the rule we have seen  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$  so that those responsible for producing the right output are increased, and the other decreased.



# Some additional details

## Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
  - $h$  is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$  is a tensor:
    - Can be agnostic of the structure: fully-connected layers,
    - Can be structure-dependent: convolutional layers.

2024-02-06 Introduction to Deep Learning and Transfer Learning

Some additional details

Non linearity: approximate complex functions! Otherwise combination of linear transformations.

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
  - $h$  is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$  is a tensor:
    - Can be agnostic of the structure: fully-connected layers,
    - Can be structure-dependent: convolutional layers.

Introduction to Deep Learning and Transfer Le

6 / 25

# Some additional details

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Some additional details

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$  is a tensor:
  - Can be agnostic of the structure: fully-connected layers,
  - Can be structure-dependent: convolutional layers.

Some additional details

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$

## Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$  is a tensor:
  - Can be agnostic of the structure: fully-connected layers,
  - Can be structure-dependent: convolutional layers.

# Some additional details

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Some additional details

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$  is a tensor:
  - Can be agnostic of the structure: **fully-connected layers**,
  - Can be structure-dependent: **convolutional layers**.

Some additional details

Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$  is a tensor:
  - Can be agnostic of the structure: fully-connected layers,
  - Can be structure-dependent: convolutional layers.

## Layers

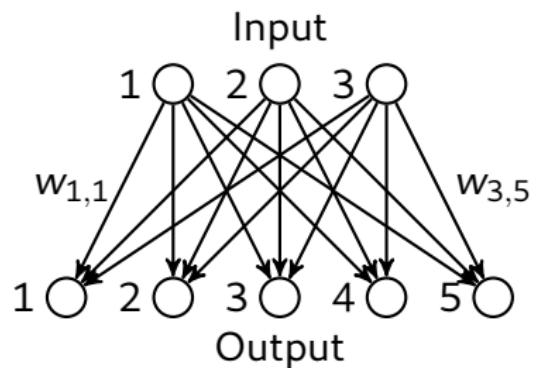
- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$  is a tensor:
  - Can be agnostic of the structure: **fully-connected layers**,
  - Can be structure-dependent: **convolutional layers**.

## Some additional details

## Layers

- $x \mapsto h(Wx + b)$ .
    - $h$  is a nonlinear parameterwise function (often without parameters),
    - $W$  is a tensor:
      - Can be agnostic of the structure: **fully-connected layers**,
      - Can be structure-dependent: **convolutional layers**.

## Fully connected layer



$$\begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \end{pmatrix}$$

## └ Some additional details

The diagram illustrates a **Fully connected layer**. It shows an input layer with 5 neurons (labeled 1, 2, 3, 4, 5) and an output layer with 5 neurons (labeled 1, 2, 3, 4, 5). Every neuron in the input layer is connected to every neuron in the output layer. The connections are labeled with weights:  $W_{1,1}$ ,  $W_{1,2}$ ,  $W_{1,3}$ ,  $W_{1,4}$ ,  $W_{1,5}$  for the first input neuron;  $W_{2,1}$ ,  $W_{2,2}$ ,  $W_{2,3}$ ,  $W_{2,4}$ ,  $W_{2,5}$  for the second; and so on. The output layer is labeled "Output". Above the input layer, the text "Fully connected layer" is written, followed by a large bracket spanning the input layer.

# Some additional details

## Layers

- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ .
- $h$  is a nonlinear parameterwise function (often without parameters),
- $\mathbf{W}$  is a tensor:
  - Can be agnostic of the structure: **fully-connected layers**,
  - Can be structure-dependent: **convolutional layers**.

### Convolutional layer

The diagram illustrates a convolutional layer. On the left, an "Input" layer consists of four white circles. On the right, an "Output" layer consists of three white circles. Colored arrows show connections from the input nodes to the output nodes: two green arrows connect the first input node to the first and second output nodes; two red arrows connect the second input node to the second and third output nodes; one blue arrow connects the third input node to the third output node. To the right of the diagram is a matrix representation of the weights:

$$\begin{pmatrix} & w_7 & w_8 & w_9 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \\ 0 & 0 & w_1 & w_2 & w_3 & w_4 & w_5 \\ & w_1 & w_2 & w_3 & 0 & 0 & 0 \end{pmatrix}$$

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ Some additional details

This slide provides additional details about layers in deep learning. It includes a legend at the top right:

- $x \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$ : A nonlinear parameterwise function (often without parameters).
- $\mathbf{W}$  is a tensor:
- Can be agnostic of the structure: **fully-connected layers**,
- Can be structure-dependent: **convolutional layers**.

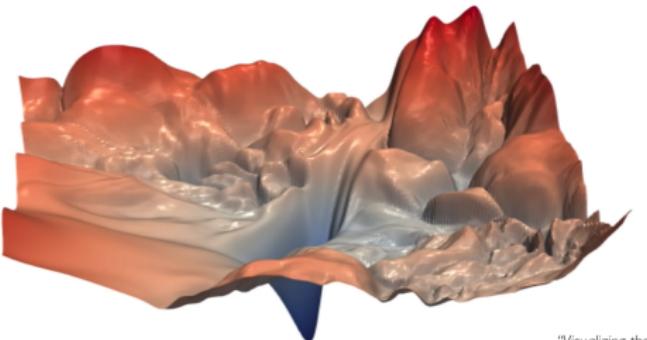
Below the legend are two diagrams of convolutional layers. The first diagram shows a "Convolutional layer" with an "Input" layer of three nodes and an "Output" layer of three nodes. The second diagram shows a more complex convolutional layer with multiple input and output nodes, illustrating how weights are shared across different spatial locations.

Introduction to Deep Learning and Transfer Le

6 / 25

# Some additional details

## Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

## Optimization with Differentiable Algorithmic

- Learning rate  $\eta$  :  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Variants of the **Stochastic Gradient Descent (SGD)** algorithm are used:
  - Use of **moments**,
  - Use of **regularizers**.

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Some additional details

Training a NN is a challenging task: this picture represents the loss landscape of a typical DL network with million of parameters, something very different from the version we have seen before for 2 parameters, extremely complex and with many local minima. Optimization depends of different factors but one of the most crucial one is the learning rate (the fraction of the gradient that is subtracted from the loss) as it determines the convergence of the SGD: it should be large enough to avoid local minima, but small enough to converge. Most of modern implementation use an adaptive lr (increase, decrease during training): try out different adaptive schemes during the lab! Also different optimizers, all variants of SGD can be explored. To increase generalization (or in other words, avoid overfitting) different regularizations techniques. Momentum: help reduce variance: accumulates a decaying moving average of past gradients so the gradient step depends on how aligned past gradients are.

Some additional details

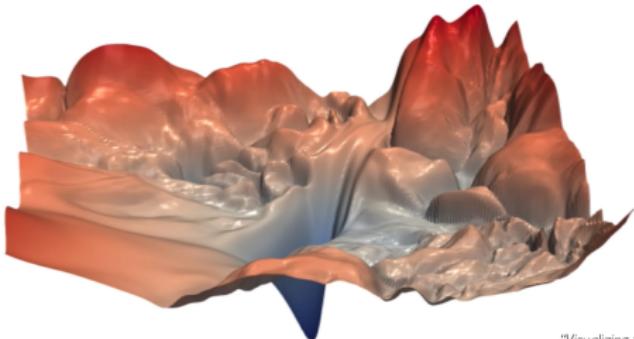
Training Neural Networks is Difficult

Optimization with Differentiable Algorithmic

- Learning rate  $\eta$  :  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- Variants of the Stochastic Gradient Descent (SGD) algorithm are used:
  - Use of moments,
  - Use of regularizers.

## Some additional details

### Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

### Batches

- To accelerate computations, inputs are often treated **concurrently** using small **batches**.

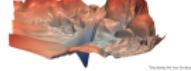
## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Some additional details

Some additional details

Training Neural Networks is Difficult



Batches

- To accelerate computations, inputs are often treated **concurrently** using small **batches**.

Backprop is computationally intensive if performed for each data example. One way to accelerate computation is to compute the gradient of batches (or small group) of training examples. This also gives a better estimate of the gradient, allows for parallelization and higher lr. Of course there is a tradeoff between higher speed (large batches) and better generalization: batch size is a hyperparameter itself. Recap: batch: gradient step, epoch: iteration over the entire dataset (ensemble of batches)

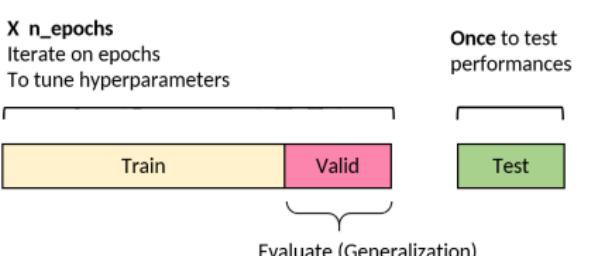
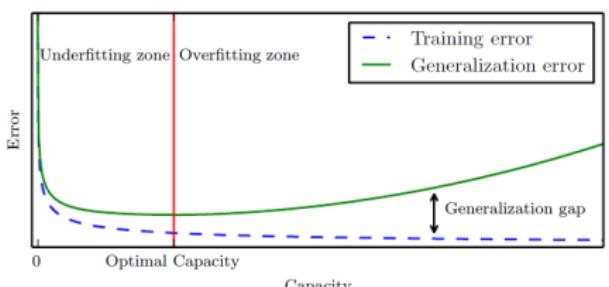
# Generalization vs Overfitting

## Learning Objectives

- Reduce the training error AND reduce the gap between training and **generalization error** (error on new inputs)
- Avoid **overfitting**, increase generalization for better performances on test set

## Validation Set

- Examples from the training distribution NOT observed during training (e.g. 20%, 80% split) to check model generalization

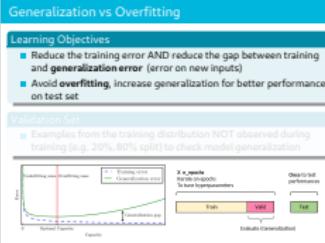


## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Generalization vs Overfitting

All these hyperparameters choices and regularization techniques have the objective to increase generalization or reduce overfitting. The typical model learning curves are showed in the left figure. In the left area when both train and generalization error are high we are in an underfitting regime: the model is not able to express the complexity of the dataset. When the gap between the generalization error and the train error increases we are specializing too much on the dataset (Overfitting regime). One way to assess this is to evaluate the performance on a validation set (split as figure on the right) and one popular regularization technique is the early stopping: stop training at the inflection point. There are many other regularization techniques (dropout: randomly set some model units to zero, also increases robustness; normalization of inputs, batch norm: normalization for intermediate features deeper in the network.)



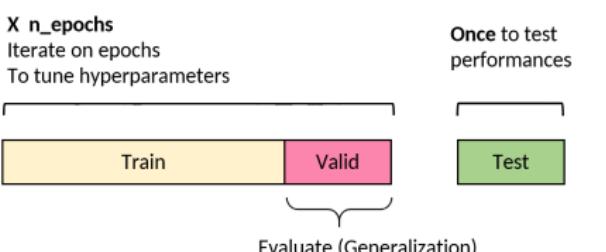
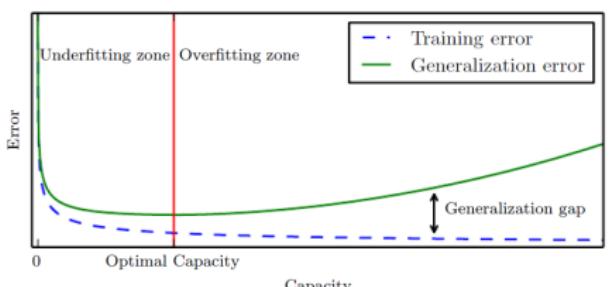
# Generalization vs Overfitting

## Learning Objectives

- Reduce the training error AND reduce the gap between training and **generalization error** (error on new inputs)
- Avoid **overfitting**, increase generalization for better performances on test set

## Validation Set

- Examples from the training distribution NOT observed during training (e.g. 20%, 80% split) to check model generalization



## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Generalization vs Overfitting

All these hyperparameters choices and regularization techniques have the objective to increase generalization or reduce overfitting. The typical model learning curves are showed in the left figure. In the left area when both train and generalization error are high we are in an underfitting regime: the model is not able to express the complexity of the dataset. When the gap between the generalization error and the train error increases we are specializing too much on the dataset (Overfitting regime). One way to assess this is to evaluate the performance on a validation set (split as figure on the right) and one popular regularization technique is the early stopping: stop training at the inflection point. There are many other regularization techniques (dropout: randomly set some model units to zero, also increases robustness; normalization of inputs, batch norm: normalization for intermediate features deeper in the network.)

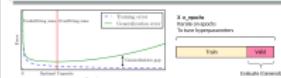
## Generalization vs Overfitting

### Learning Objectives

- Reduce the training error AND reduce the gap between training and generalization error (error on new inputs)
- Avoid overfitting, increase generalization for better performances on test set

### Validation Set

- Examples from the training distribution NOT observed during training (e.g. 20%, 80% split) to check model generalization



Evaluate (Generalization)

# The case of deep learning in classification

## Inputs/outputs

- Often: inputs are raw signals or feature vectors,
- Often: outputs are vectors which highest value indicate the category of the input.

The diagram illustrates a neural network architecture. It starts with an input image of a cat (labeled 'layer 1'). This image is processed by layer 1, then layer 2. From layer 2, two paths emerge: one leading to layer 3 and another leading to layer 4. Both layer 3 and layer 4 feed into a summation node (indicated by a circle with a '+'). The output of this summation node then passes through layer 5. Layer 5 produces a vector output: [ 0.7 | 0.3 ].

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ The case of deep learning in classification

The case of deep learning in classification

Inputs/outputs

- Often: inputs are raw signals or feature vectors;
- Often: outputs are vectors which highest value indicate the category of the input.

The diagram illustrates a neural network architecture. It starts with an input image of a cat (labeled 'layer 1'). This image is processed by layer 1, then layer 2. From layer 2, two paths emerge: one leading to layer 3 and another leading to layer 4. Both layer 3 and layer 4 feed into a summation node (indicated by a circle with a '+'). The output of this summation node then passes through layer 5. Layer 5 produces a vector output: [ 0.7 | 0.3 ].

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

Introduction to Deep Learning and Transfer Le

9/25

# The case of deep learning in classification

## Inputs/outputs

- Often: inputs are raw signals or feature vectors,
- Often: outputs are vectors which highest value indicate the category of the input.

The diagram illustrates a neural network architecture. It starts with an input image of a cat (layer 1). This image is processed by layer 2, which then branches into two paths. One path leads to layer 3, and the other leads to layer 4. Both layer 3 and layer 4 have arrows pointing to a summation node (represented by a circle with a plus sign). The output of this summation node then passes through layer 5. Layer 5 produces a vector output: [0.7, 0.3]. An arrow labeled "cat" points from this vector to the top element, 0.7, indicating it is the predicted category.

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ The case of deep learning in classification

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

The case of deep learning in classification

Inputs/outputs

- Often: inputs are raw signals or feature vectors;
- Often: outputs are vectors which highest value indicate the category of the input.

The diagram illustrates a neural network architecture. It starts with an input image of a cat (layer 1). This image is processed by layer 2, which then branches into two paths. One path leads to layer 3, and the other leads to layer 4. Both layer 3 and layer 4 have arrows pointing to a summation node (represented by a circle with a plus sign). The output of this summation node then passes through layer 5. Layer 5 produces a vector output: [0.7, 0.3]. An arrow labeled "cat" points from this vector to the top element, 0.7, indicating it is the predicted category.

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

# The case of deep learning in classification

## Inputs/outputs

- Often: inputs are raw signals or feature vectors,
- Often: outputs are vectors which highest value indicate the category of the input.

The diagram illustrates a neural network architecture. It starts with a small image of a dog's head labeled "layer 1". This is followed by "layer 2", which is a square box. Arrows point from "layer 1" to "layer 2" and from "layer 2" to a central summation node (a circle with a plus sign). From this summation node, arrows point to "layer 3" (a rectangle) and "layer 4" (a rectangle). Both "layer 3" and "layer 4" have arrows pointing to "layer 5" (a square box). "layer 5" has an arrow pointing to a vector output:  $\begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$ . A downward arrow from this vector points to the word "dog", indicating the predicted category.

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ The case of deep learning in classification

The case of deep learning in classification

Inputs/outputs

- Often: inputs are raw signals or feature vectors;
- Often: outputs are vectors which highest value indicate the category of the input.

This diagram shows a more complex neural network structure. It starts with a small image of a dog's head labeled "layer 1". This is followed by "layer 2", which is a square box. Arrows point from "layer 1" to "layer 2" and from "layer 2" to a central summation node (a circle with a plus sign). From this summation node, arrows point to "layer 3" (a rectangle) and "layer 4" (a rectangle). Both "layer 3" and "layer 4" have arrows pointing to "layer 5" (a square box). "layer 5" has an arrow pointing to a vector output:  $\begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix}$ . A downward arrow from this vector points to the word "dog", indicating the predicted category.

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

9/25

# The case of deep learning in classification

## Inputs/outputs

- Often: inputs are raw signals or feature vectors,
- Often: outputs are vectors which highest value indicate the category of the input.

layer 1 → layer 2 → layer 3 + layer 4 → layer 5 →  $\begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

## Loss and targets

- Labels are encoded as one-hot-bit vectors and called targets,
- Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ ,
- Loss is typically cross-entropy:  $-\log(\hat{y}^\top y)$ .

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### The case of deep learning in classification

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

#### The case of deep learning in classification

Inputs/outputs

- Often: inputs are raw signals or feature vectors;
- Often: outputs are vectors which highest value indicate the category of the input.

Loss and targets

- Labels are encoded as one-hot-bit vectors and called targets,
- Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ ,
- Loss is typically cross-entropy:  $-\log(\hat{y}^\top y)$ .

layer 1 → layer 2 → layer 3 + layer 4 → layer 5 →  $\begin{bmatrix} 0.2 \\ 0.8 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

# The case of deep learning in classification

## Inputs/outputs

- Often: inputs are raw signals or feature vectors,
- Often: outputs are vectors which highest value indicate the category of the input.

## Loss and targets

- Labels are encoded as one-hot-bit vectors and called targets,
- Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ ,
- Loss is typically cross-entropy:  $-\log(\hat{y}^T y)$ .

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### The case of deep learning in classification

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

#### The case of deep learning in classification

Inputs/outputs

- Often: inputs are raw signals or feature vectors;
- Often: outputs are vectors which highest value indicate the category of the input.

Loss and targets

- Labels are encoded as one-hot-bit vectors and called targets,
- Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ ,
- Loss is typically cross-entropy:  $-\log(\hat{y}^T y)$ .

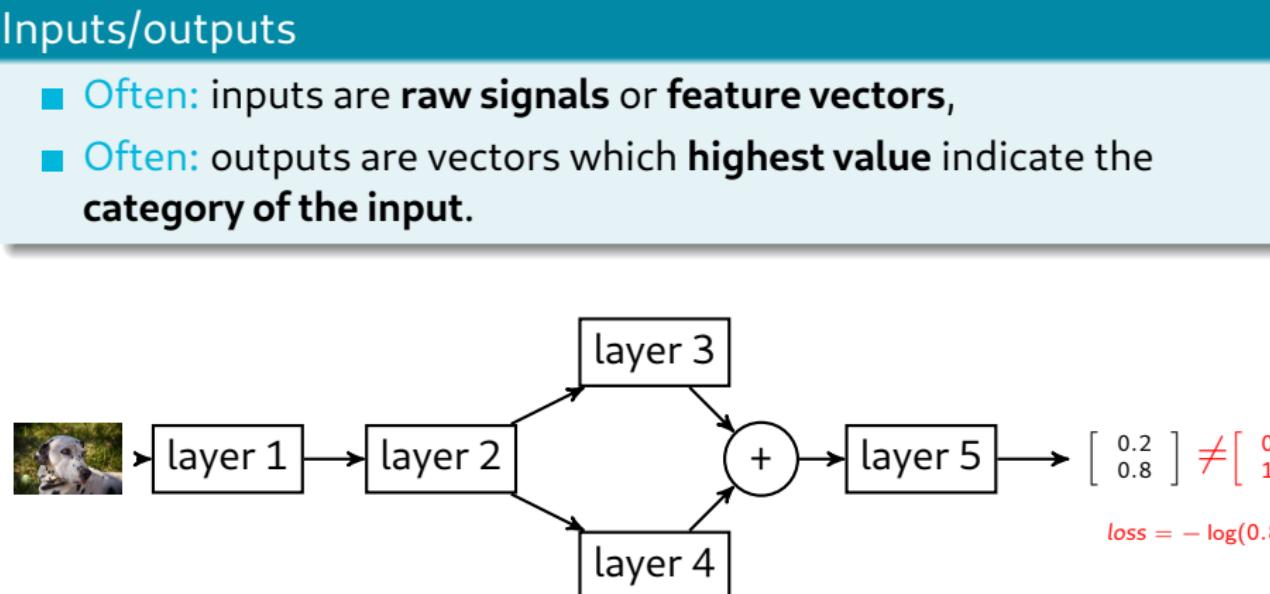
Diagram illustrating the flow of data through a neural network. Inputs pass through layers 1, 2, 3, 4, and 5. Layer 2 branches into paths to layers 3 and 4, which then merge at a summation node before entering layer 5. The final output is a vector [0.2, 0.8], which is shown to be different from the target vector [1, 0].

Diagram illustrating the softmax and cross-entropy loss functions. Labels are one-hot vectors. Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ . Loss is typically cross-entropy:  $-\log(\hat{y}^T y)$ .

# The case of deep learning in classification

## Inputs/outputs

- Often: inputs are raw signals or feature vectors,
- Often: outputs are vectors which highest value indicate the category of the input.



$\text{loss} = -\log(0.8)$

## Loss and targets

- Labels are encoded as one-hot-bit vectors and called targets,
- Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ ,
- Loss is typically cross-entropy:  $-\log(\hat{y}^\top y)$ .

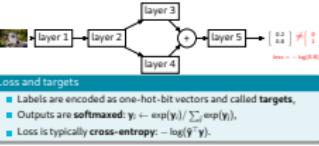
2024-02-06 Introduction to Deep Learning and Transfer Learning

└ The case of deep learning in classification

The case of deep learning in classification

Inputs/outputs

- Often: inputs are raw signals or feature vectors;
- Often: outputs are vectors which highest value indicate the category of the input.



Loss and targets

- Labels are encoded as one-hot-bit vectors and called targets,
- Outputs are softmaxed:  $y_i \leftarrow \exp(y_i) / \sum_j \exp(y_j)$ ,
- Loss is typically cross-entropy:  $-\log(\hat{y}^\top y)$ .

Softmax to generate a probability from numbers. Cross Entropy Loss (Shannon 50 years ago) compare how different the predicted and labels distribution are.

9/25

Introduction to Deep Learning and Transfer Le

# Transfer Learning and fine-tuning

Idea: use **feature vectors** from a **backbone** (pretrained) network to train a **downstream** classifier.

**Two usecases**

- **Fine-tuning:** both the backbone and downstream networks are trained,
- **Transfer Learning:** Only the downstream network is trained.

```
graph LR; Input[Cat Image] --> L1[layer 1]; L1 --> L2[layer 2]; L2 --> L3[layer 3]; L2 --> L4[layer 4]; L3 --> Sum((+)); L4 --> Sum; Sum --> L5[layer 5]; L5 --> Output["[0.2 0.8]"]; Output --> Cat["cat"];
```

2024-02-06 Introduction to Deep Learning and Transfer Learning

Tranfer Learning and fine-tuning

Idea: use feature vectors from a **backbone** (pretrained) network to train a **downstream** classifier.

**Two usecases**

- **Fine-tuning:** both the backbone and downstream networks are trained,
- **Transfer Learning:** Only the downstream network is trained.

An idea that is extensively applied in computer vision and more generally in DL is transfer learning. Consist in exploiting the knowledge of a network pretrained on a large dataset to adapt it for a novel, usually smaller and more specialized dataset/ classification task on dataset. 2 ways: Fine tuning: retrain for a few epochs the whole network on the novel dataset.

Introduction to Deep Learning and Transfer Le

10 / 25

# Transfer Learning and fine-tuning

Idea: use **feature vectors** from a **backbone** (pretrained) network to train a **downstream** classifier.

## Two usecases

- **Fine-tuning:** both the backbone and downstream networks are trained,
- **Transfer Learning:** Only the downstream network is trained.

Diagram illustrating Transfer Learning and fine-tuning:

```
graph LR; Image[Image of a cat] --> L1[layer 1]; L1 --> L2[layer 2]; L2 --> L3[layer 3]; L2 --> L4[layer 4]; L3 --> Sum((+)); L4 --> Sum; Sum --> L5[layer 5]; L5 --> Cat["cat"]; L5 --> Prob["[0.2, 0.8]"];
```

2024-02-06 Introduction to Deep Learning and Transfer Learning

Transfer Learning and fine-tuning

Idea: use feature vectors from a backbone (pretrained) network to train a downstream classifier.

Two usecases

- Fine-tuning: both the backbone and downstream networks are trained.
- Transfer Learning: Only the downstream network is trained.

# Transfer Learning and fine-tuning

Idea: use **feature vectors** from a **backbone** (pretrained) network to train a **downstream** classifier.

## Two usecases

- **Fine-tuning:** both the backbone and downstream networks are trained,
- **Transfer Learning:** Only the downstream network is trained.

The diagram illustrates a neural network architecture. It starts with an input arrow pointing to 'layer 1'. 'layer 1' points to 'layer 2'. 'layer 2' has two output paths: one to 'layer 3' and one to 'layer 4'. 'layer 3' and 'layer 4' both point to a circular summation node with a plus sign (+). A blue box labeled 'Backbone' encloses 'layer 1' and 'layer 2'. The entire network is contained within a light blue rectangular frame.

2024-02-06

## Introduction to Deep Learning and Transfer Learning

### Transfer Learning and fine-tuning

Transfer Learning: chop out from the pretrained network the last dense/fully connected layers: this part is frozen and called backbone and only the final layer(s) are retrained on the novel dataset.

Transfer Learning and fine-tuning

Tranfer Learning and fine-tuning

Idea: use feature vectors from a backbone (pretrained) network to train a downstream classifier.

Two usecases

- Fine-tuning: both the backbone and downstream networks are trained.
- Transfer Learning: Only the downstream network is trained.

The diagram shows a 'Backbone' consisting of 'layer 1' and 'layer 2'. 'layer 3' and 'layer 4' are part of the 'Downstream' network. In 'Fine-tuning', both the backbone and downstream are trained. In 'Transfer Learning', only the downstream network is trained, while the backbone is frozen.

Introduction to Deep Learning and Transfer Le

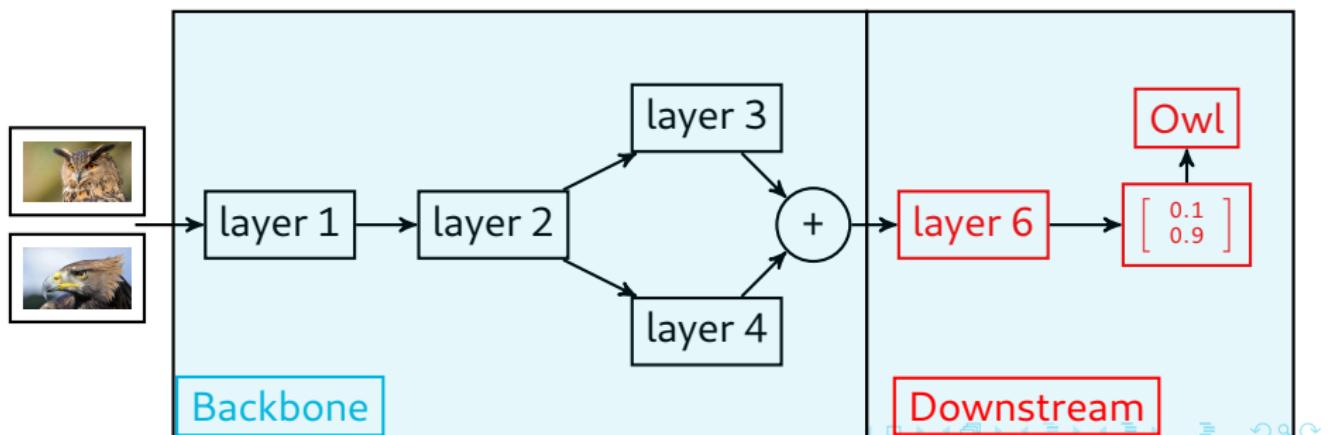
10 / 25

# Transfer Learning and fine-tuning

Idea: use **feature vectors** from a **backbone** (pretrained) network to train a **downstream** classifier.

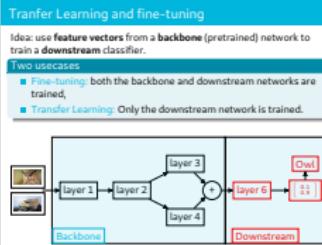
## Two usecases

- **Fine-tuning:** both the backbone and downstream networks are trained,
- **Transfer Learning:** Only the downstream network is trained.



## 2024-02-06 Introduction to Deep Learning and Transfer Learning

### Transfer Learning and fine-tuning



These two techniques can obviously be combined! And are very useful to specialise the network for a precise task like for instance classifying birds, using a backbone trained on a big CV dataset such as ImageNet.

# Hyperparameters

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ Hyperparameters

Hyperparameters

Architecture
■ Number of layers
■ Architecture choice (e.g. ResNet, DenseNet, VGG, ...)

Training
■ Learning rate and scheduling
■ Regularization (e.g. weight decay)
■ Choice of optimizer (e.g. SGD)

## Architecture

- Number of layers
- Architecture choice (e.g. ResNet, DenseNet, VGG, ...)

## Training

- Learning rate and scheduling
- Regularization (e.g. weight decay)
- Choice of optimizer (e.g. SGD)

# Hyperparameters

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ Hyperparameters

Hyperparameters

Architecture
■ Number of layers
■ Architecture choice (e.g. ResNet, DenseNet, VGG, ...)

Training
■ Learning rate and scheduling
■ Regularization (e.g. weight decay)
■ Choice of optimizer (e.g. SGD)

## Architecture

- Number of layers
- Architecture choice (e.g. ResNet, DenseNet, VGG, ...)

## Training

- Learning rate and scheduling
- Regularization (e.g. weight decay)
- Choice of optimizer (e.g. SGD)

# A focus on VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
	<b>LRN</b>	<b>conv3-64</b>			
			conv3-64	conv3-64	conv3-64
			conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
		<b>conv3-128</b>	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
			<b>conv1-256</b>		
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			<b>conv1-512</b>		
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
			<b>conv1-512</b>		
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

2024-02-06 A focus on Deep Learning and Transfer Learning

└ A focus on VGG

A focus on VGG

The diagram illustrates the VGG-16 architecture, which consists of two parallel paths of layers. The layers are organized into sections labeled A through E. 
 - Section A: 11 weight layers, including LRN layers.
 - Section B: 13 weight layers.
 - Section C: 16 weight layers, highlighted with a red box.
 - Section D: 16 weight layers.
 - Section E: 19 weight layers.
 The diagram shows the flow from input to output, including convolutional layers (e.g., conv3-64, conv3-128, conv3-256, conv3-512), max pooling layers, fully connected layers (FC-4096, FC-1000), and a final softmax layer.

Introduction to Deep Learning and Transfer Le

12 / 25

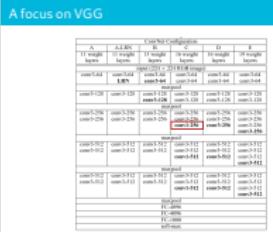
# A focus on VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 <b>conv3-256</b>	conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### └ A focus on VGG



# A focus on VGG

A convolutional layer with 1x1 filters, 3 input feature maps and 2 output feature maps.

The diagram illustrates a convolutional layer operation. On the left, three input feature maps ( $F_{in}$ ) of size  $H \times W$  are shown. A red 1x1 kernel labeled  $k$  is applied to these inputs. The resulting output is two feature maps ( $F_{out}$ ) of size  $H \times W$ . Below the diagram, the text "conv1 : k = 1" is displayed. Labels indicate "input feature maps / input channels" for the inputs and "output feature maps / output channels" for the outputs.

2024-02-06

## Introduction to Deep Learning and Transfer Learning

### └ A focus on VGG

A convolutional layer with 1x1 filters, 3 input feature maps and 2 output feature maps.

The diagram illustrates a convolutional layer operation. On the left, three input feature maps ( $F_{in}$ ) of size  $H \times W$  are shown. A red 1x1 kernel labeled  $k$  is applied to these inputs. The resulting output is two feature maps ( $F_{out}$ ) of size  $H \times W$ . Below the diagram, the text "conv1 : k = 1" is displayed. Labels indicate "input feature maps / input channels" for the inputs and "output feature maps / output channels" for the outputs.

A convolutional layer with 1x1 filters, 3 input feature maps and 2 output feature maps.

The diagram illustrates a convolutional layer operation. On the left, three input feature maps ( $F_{in}$ ) of size  $H \times W$  are shown. A red 1x1 kernel labeled  $k$  is applied to these inputs. The resulting output is two feature maps ( $F_{out}$ ) of size  $H \times W$ . Below the diagram, the text "conv1 : k = 1" is displayed. Labels indicate "input feature maps / input channels" for the inputs and "output feature maps / output channels" for the outputs.

# A focus on VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64	conv3-64	<b>conv3-64</b>	conv3-64	conv3-64
	<b>LRN</b>	<b>conv3-64</b>		conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
		<b>conv3-128</b>	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256		conv3-256	conv3-256	<b>conv1-256</b>	conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512		conv3-512	conv3-512	<b>conv1-512</b>	conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512		conv3-512	conv3-512	<b>conv1-512</b>	conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

2024-02-06 A focus on Deep Learning and Transfer Learning

└ A focus on VGG

A focus on VGG

Introduction to Deep Learning and Transfer Le

15 / 25

# A focus on VGG

A convolutional layer with 3x3 filters, 3 input feature maps and 2 output feature maps.

The diagram illustrates a convolutional layer. On the left, three blue rectangular boxes labeled  $F_{in}$  represent input feature maps, each with height  $H$  and width  $W$ . An arrow points from these to a red 3x3 cube labeled "kernel" with dimension  $k$ . The output is two green rectangular boxes labeled  $F_{out}$ , each with height  $H$  and width  $W$ . Below the diagram, the text "conv3 : k = 3" is written in red.

input feature maps / input channels      convolution filters      output feature maps / output channels

conv3 : k = 3

2024-02-06

# Introduction to Deep Learning and Transfer Learning

## └ A focus on VGG

A convolutional layer with 3x3 filters, 3 input feature maps and 2 output feature maps.

The diagram shows a convolutional layer with the following components:

- Input:** Three blue rectangles labeled  $F_{in}$  representing input feature maps. Each has height  $H$  and width  $W$ .
- Kernel:** A red 3x3 cube labeled "kernel" with dimension  $k$ .
- Output:** Two green rectangles labeled  $F_{out}$  representing output feature maps. Each has height  $H$  and width  $W$ .

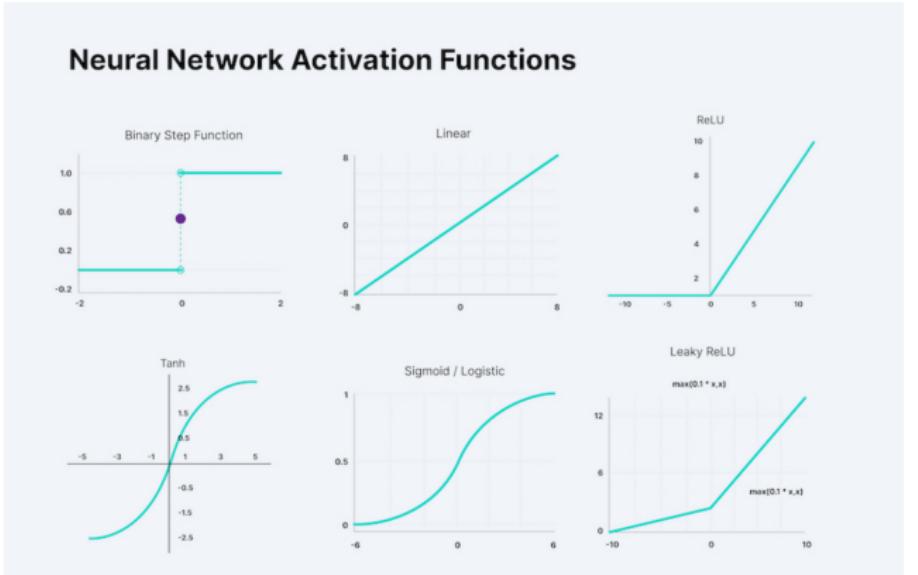
Annotations below the diagram indicate:

- Input feature maps / input channels
- convolution filters
- output feature maps / output channels

conv3 : k = 3

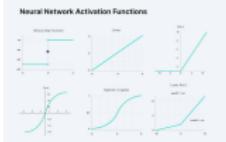
# A focus on VGG

There is a nonlinear activation function (ReLU) after each convolutional layer. Other existing activation functions are: Sigmoid, Tanh, Leaky ReLU, Parametric ReLU, SELU, ELU, GELU, Swish, ...



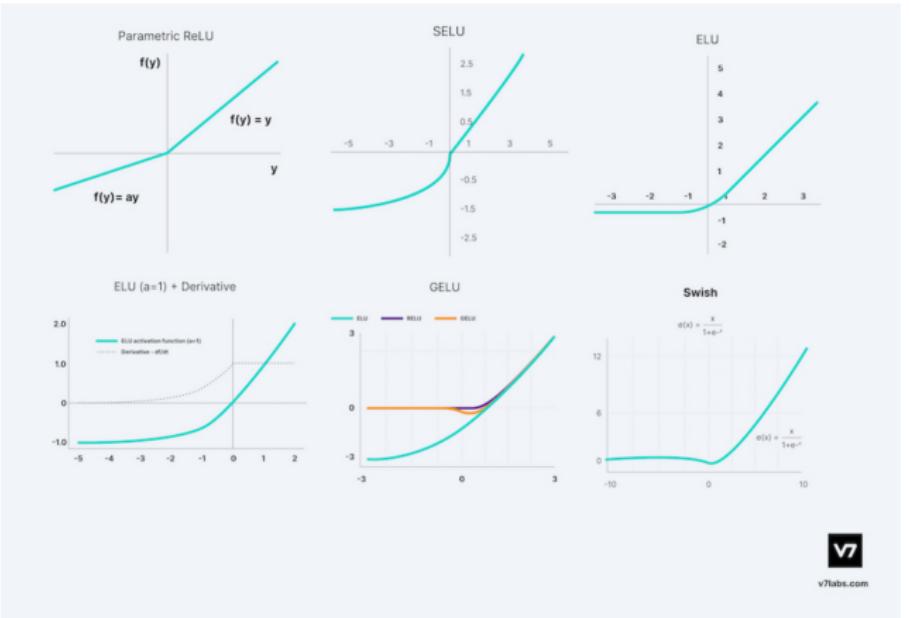
## 2024-02-06 Introduction to Deep Learning and Transfer Learning

### └ A focus on VGG



# A focus on VGG

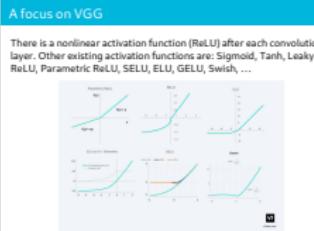
There is a nonlinear activation function (ReLU) after each convolutional layer. Other existing activation functions are: Sigmoid, Tanh, Leaky ReLU, Parametric ReLU, SELU, ELU, GELU, Swish, ...



2024-02-06

## Introduction to Deep Learning and Transfer Learning

### └ A focus on VGG

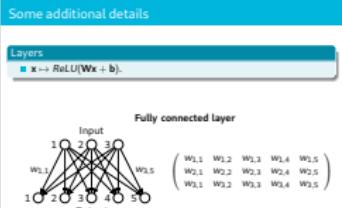
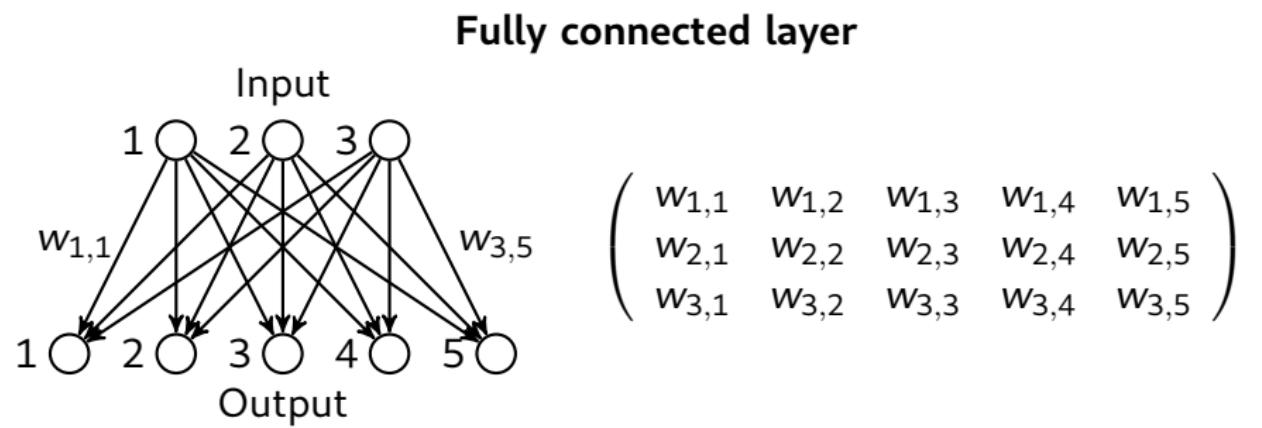


## A focus on VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

## Introduction to Deep Learning and Transfer Learning

## —A focus on VGG



## A focus on VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

## L-06 Introduction to Deep Learning and Transfer Learning

## —A focus on VGG

# A focus on VGG

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ A focus on VGG

maxpool, kernel 2, stride 2

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

2	
---	--

A focus on VGG

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

maxpool, kernel 2, stride 2

Introduction to Deep Learning and Transfer Le

22 / 25

A focus on VGG

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

maxpool, kernel 2, stride 2

2

# A focus on VGG

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ A focus on VGG

maxpool, kernel 2, stride 2

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

maxpool, kernel 2, stride 2


2	3

A focus on VGG

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

maxpool, kernel 2, stride 2


2	3

Introduction to Deep Learning and Transfer Le

22 / 25

# A focus on VGG

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ A focus on VGG

maxpool, kernel 2, stride 2

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

2	3
8	

maxpool, kernel 2, stride 2

A focus on VGG

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

2	3
8	

maxpool, kernel 2, stride 2

Introduction to Deep Learning and Transfer Learning

22 / 25

# A focus on VGG

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ A focus on VGG

maxpool, kernel 2, stride 2

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

2	3
8	6

maxpool, kernel 2, stride 2

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

2	3
8	6

A focus on VGG

1	2	3	1
1	1	1	1
2	3	1	6
8	1	4	5

2	3
8	6

maxpool, kernel 2, stride 2

# A focus on VGG

ConvNet Configuration

A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

input ( $224 \times 224$  RGB image)

conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Navigation icons: back, forward, search, etc.

2024-02-06 A focus on Deep Learning and Transfer Learning

A focus on VGG

ConvNet Configuration

A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers

conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Introduction to Deep Learning and Transfer Le

23 / 25

# A focus on VGG

2024-02-06 Introduction to Deep Learning and Transfer Learning

└ A focus on VGG

A focus on VGG

The diagram illustrates a softmax layer. On the left, under the heading 'FC-1000', there is a vertical list of outputs:  $o_1$ ,  $o_2$ , ...,  $o_{1000}$ . An arrow labeled 'softmax' points from this list to the right. On the right, there is a corresponding list of probabilities:  $y_1$ ,  $y_2$ , ...,  $y_{1000}$ . Below this, the softmax formula is given:  $y_j = \frac{\exp(o_j)}{\sum_{k=1}^{1000} \exp(o_k)}$ .

FC-1000

$o_1$                    $y_1$   
 $o_2$     softmax       $y_2$   
...                      ...  
 $o_{1000}$                  $y_{1000}$

$y_j = \frac{\exp(o_j)}{\sum_{k=1}^{1000} \exp(o_k)}$

# Lab Session 1 and assignment

## Introduction to Deep Learning

- Introduction to Deep Learning in Pytorch
- Train a full DL model from scratch
- Train a downstream model using transfer learning

## Project 1 (oral presentation)

Explore one of the following architectures : ResNet, DenseNet, PreActResNet, VGG.

You have to prepare a 10 minutes (+5 min Q&A) presentation for session 3, in which you explain :

- Description of the architecture
- Hyperparameter search and results
- Study the compromise between architecture size, performance and training time.

## Introduction to Deep Learning and Transfer Learning

2024-02-06

### Lab Session 1 and assignment

Lab Session 1 and assignment

#### Introduction to Deep Learning

- Introduction to Deep Learning in Pytorch
- Train a full DL model from scratch
- Train a downstream model using transfer learning

#### Project 1 (oral presentation)

Explore one of the following architectures : ResNet, DenseNet, PreActResNet, VGG.

You have to prepare a 10 minutes (+5 min Q&A) presentation for session 3, in which you explain :

- Description of the architecture
- Hyperparameter search and results
- Study the compromise between architecture size, performance and training time.