Introduction to Deep Learning and Transfer Learning

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Global formalism

Input/output

- Goal: infer a function from an input (often tensor) space to an output (often tensor) space, $\mathbf{y} = f(\mathbf{x})$,
- 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 $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- 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, $\mathbf{y} = f(\mathbf{x})$,
- 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 $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- 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, $\mathbf{y} = f(\mathbf{x})$,
- 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 $\hat{\mathbf{y}}$ and obtained output $\mathbf{y}$.
- Example: output should be $[0, 1]$ but is $[0.2, 0.8]$.

Parameters

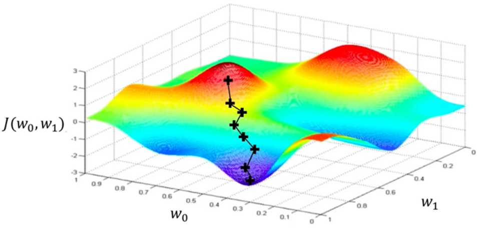
- $f = f_w$ contains **parameters W** to be trained,
- In most cases, an ideal f_w exists but is **hard to find in practice**,
- Learning is a **regression ill-posed** problem.

Global formalism

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

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

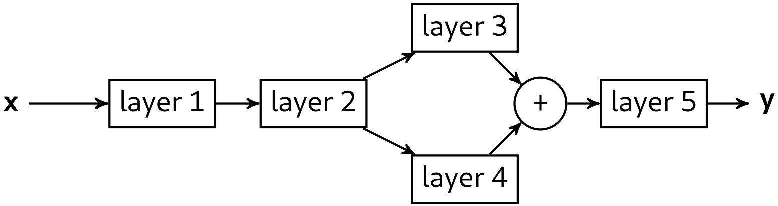


from MIT course `introtodeeplearning.com`

Deep learning

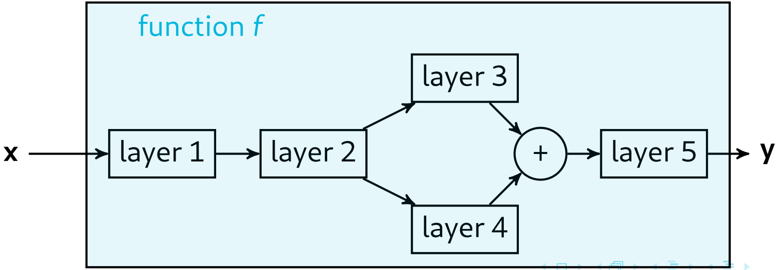
Main idea

- **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)



Main idea

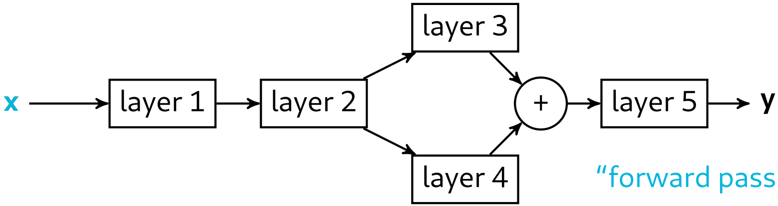
- Compositional Approach: Instead of directly mapping $\mathbf{x}$ to $\mathbf{y}$, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient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

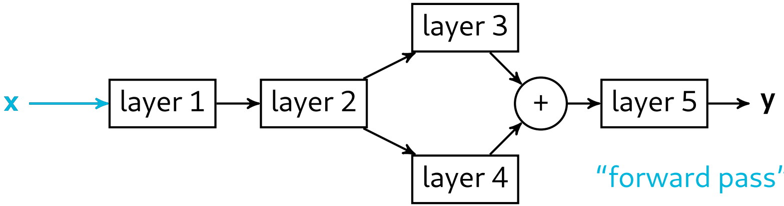


"forward pass"

Deep learning

Main idea

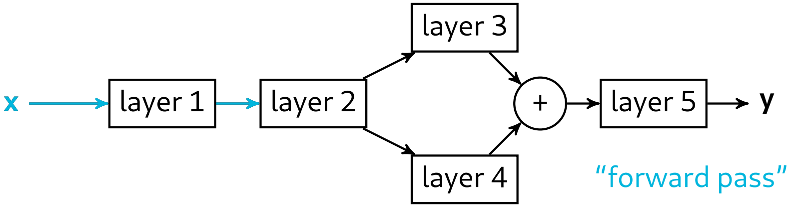
- Compositional Approach: Instead of directly mapping $\mathbf{x}$ to $\mathbf{y}$, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient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



"forward pass"

Main idea

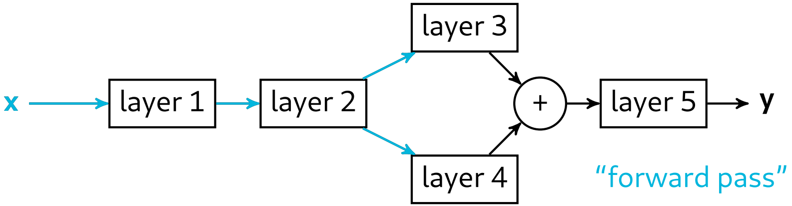
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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



Deep learning

Main idea

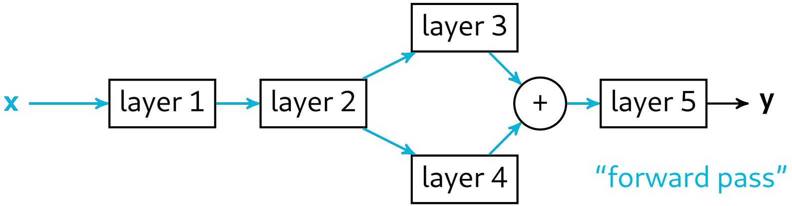
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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



Deep learning

Main idea

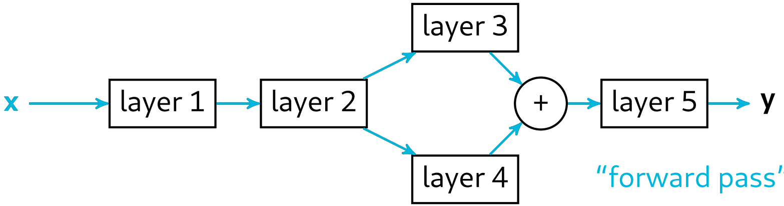
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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

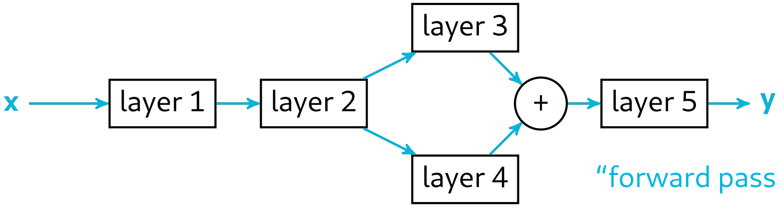


"forward pass"

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)

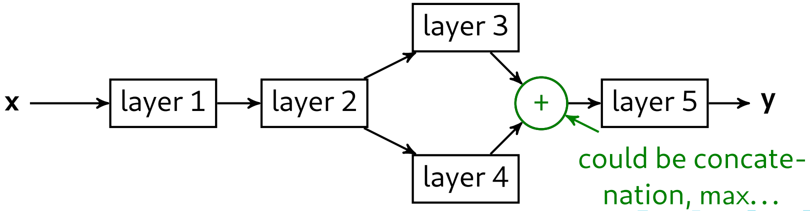


"forward pass"

Deep learning

Main idea

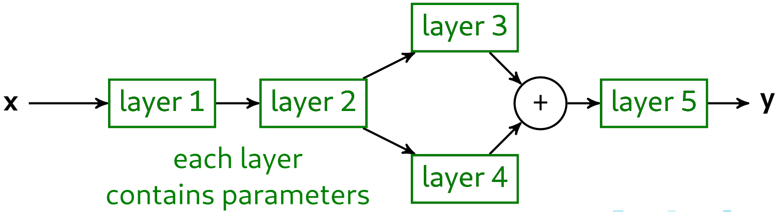
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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



Deep learning

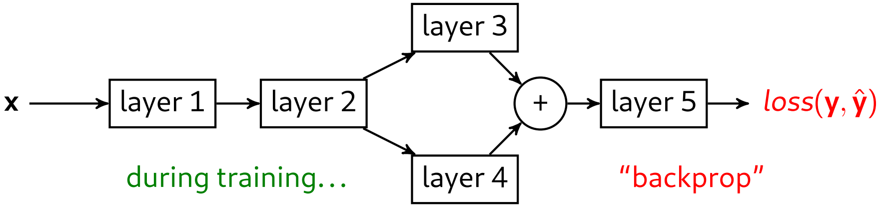
Main idea

- 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)



Main idea

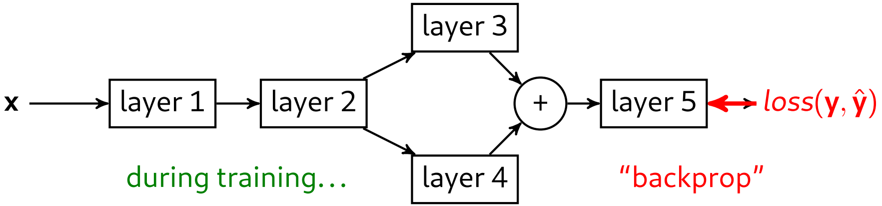
- Compositional Approach: Instead of directly mapping $\mathbf{x}$ to $\mathbf{y}$, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient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



Deep learning

Main idea

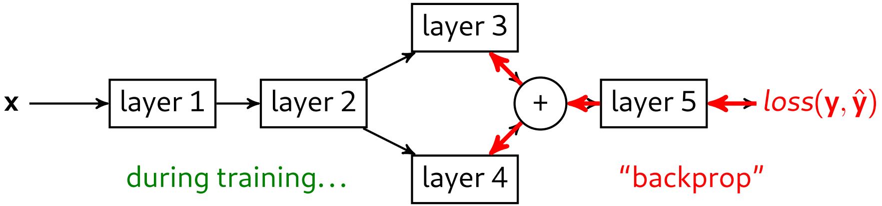
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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



Deep learning

Main idea

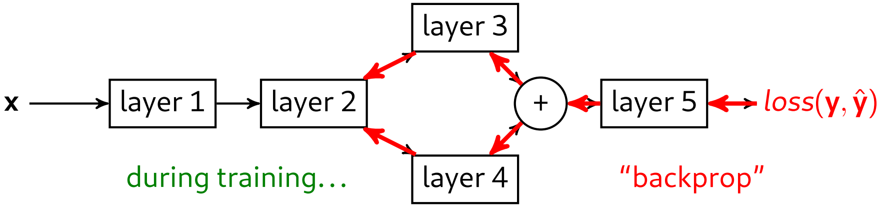
- Compositional Approach: Instead of directly mapping $\mathbf{x}$ to $\mathbf{y}$, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient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



Deep learning

Main idea

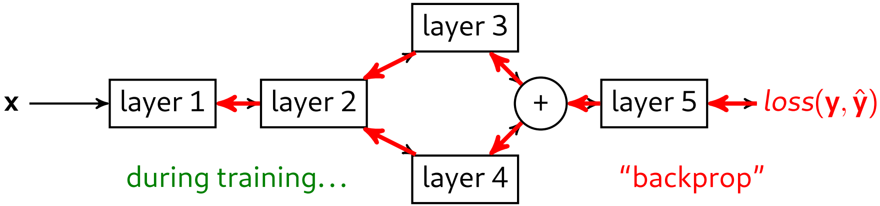
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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



Deep learning

Main idea

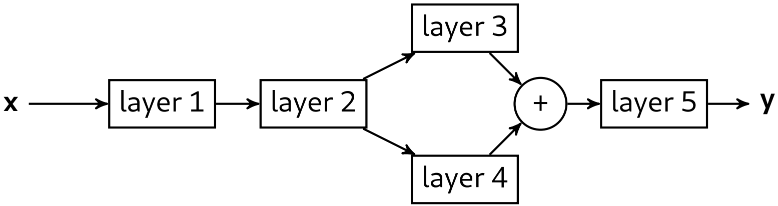
- Compositional Approach: Instead of directly mapping **x** to **y**, express solutions as an assembly of simple mathematical functions called **layers**
- End-to-end learning: Tune all atomic functions together
- Training: Backpropagate throughout the architecture (to compute the gradient of the loss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



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

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

- $\mathbf{x} \mapsto h(\mathbf{Wx} + \mathbf{b})$.
  - 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

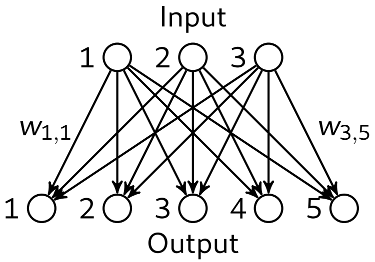
- $\mathbf{x} \mapsto h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
  - h is a nonlinear parameterwise function (often without parameters),
  - $\mathbf{W}$ is a tensor:
    - Can be agnostic of the structure: **fully-connected layers**,
    - Can be structure-dependent: **convolutional layers**.

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**.

Fully connected layer



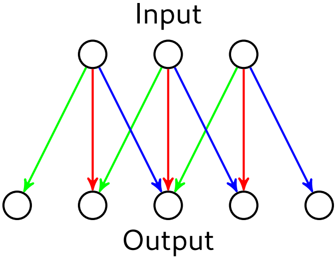
$$\begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \end{pmatrix}$$

Some additional details

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**.

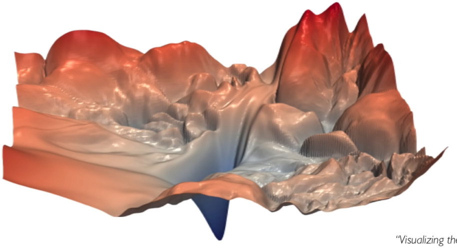
Convolutional layer



$$\begin{pmatrix} w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \end{pmatrix}$$

Some additional details

Training Neural Networks is Difficult



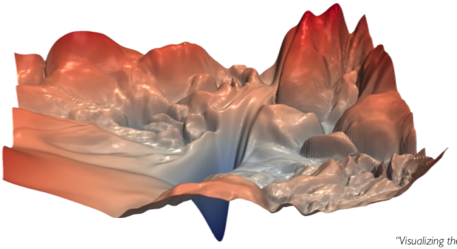
"Visualizing the loss landscape of neural nets". Dec 2017.

Optimization with Differentiable Algorithmic

- Learning rate η : $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
- 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**.

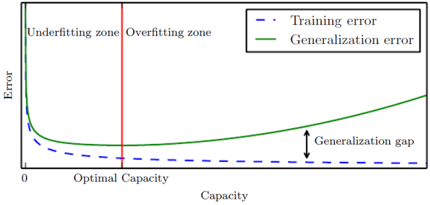
Generalization vs Overfitting

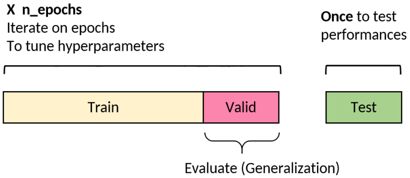
Learning Objectives

- 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% sp lit) to check model generalization



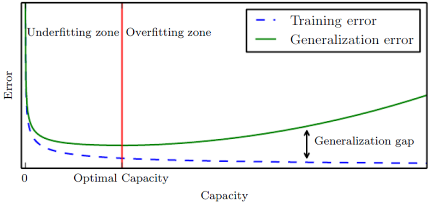


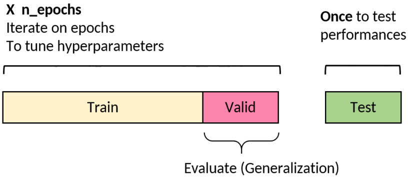
Learning Objectives

- 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% sp lit) to check model generalization

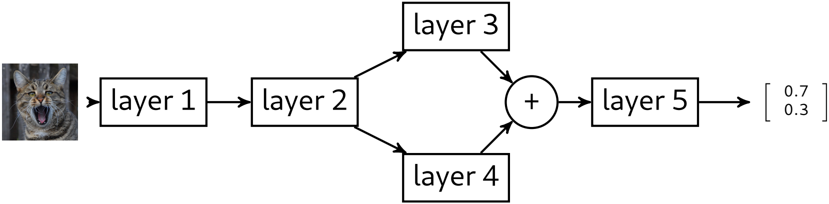




The case of deep learning in classification

Inputs/outputs

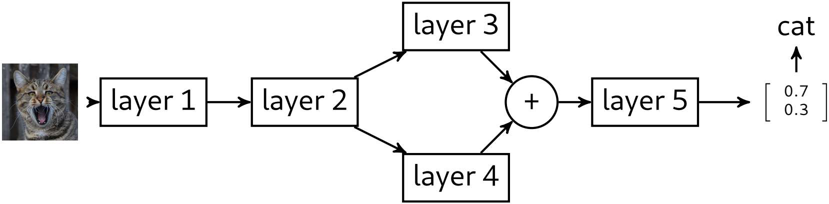
- Often: inputs are **raw signals** or **feature vectors**,
- Often: outputs are vectors which **highest value** indicate the **category of the input**.



The case of deep learning in classification

Inputs/outputs

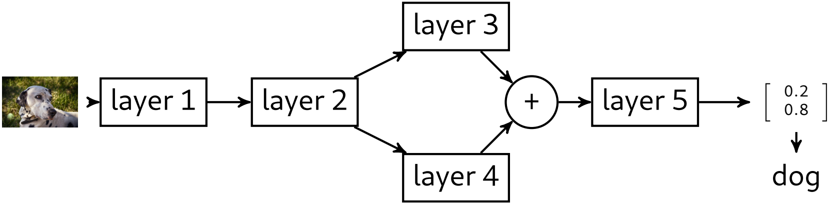
- Often: inputs are **raw signals** or **feature vectors**,
- Often: outputs are vectors which **highest value** indicate the **category of the input**.



The case of deep learning in classification

Inputs/outputs

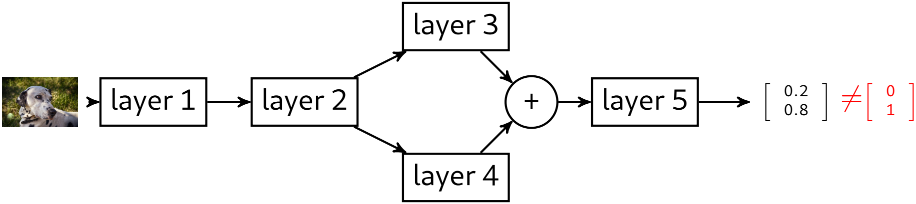
- Often: inputs are **raw signals** or **feature vectors**,
- Often: outputs are vectors which **highest value** indicate the **category of the input**.



The 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**.



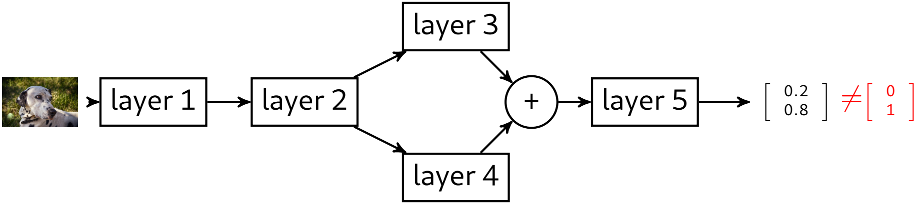
Loss and targets

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

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**.



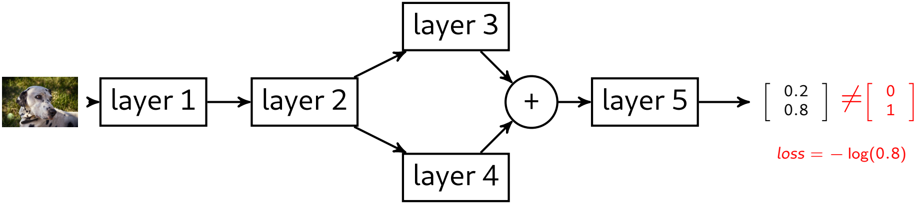
Loss and targets

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

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**.



Loss and targets

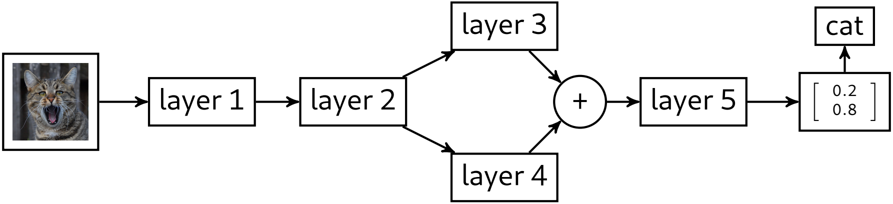
- Labels are encoded as one-hot-bit vectors and called **targets**,
- Outputs are **softmaxed**: $\mathbf{y}_i \leftarrow \exp(\mathbf{y}_i)/\sum_j \exp(\mathbf{y}_j)$,
- Loss is typically **cross-entropy**: $-\log(\hat{\mathbf{y}}^\top \mathbf{y})$.

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.

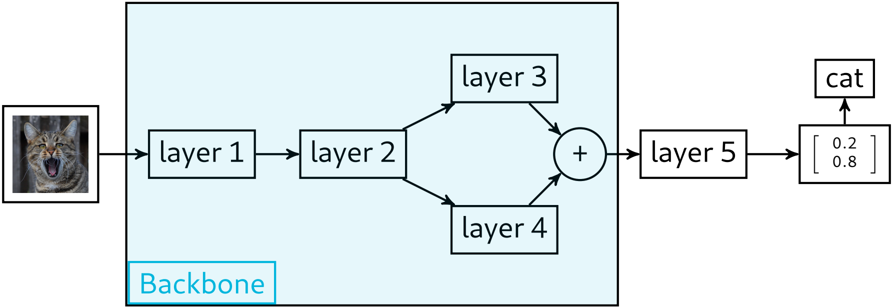


Tranfer Learning and fine-tuning

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.

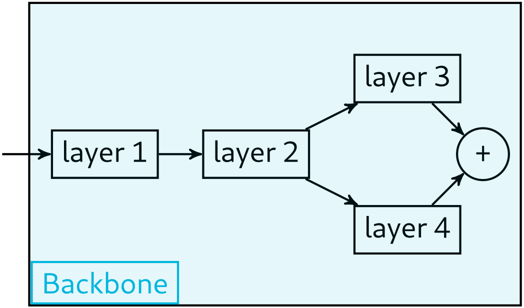


Tranfer Learning and fine-tuning

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.

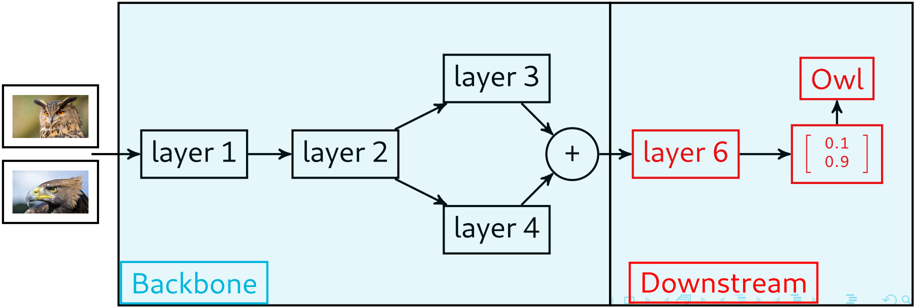


Tranfer Learning and fine-tuning

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.



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)

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)

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.