Deep learning

Neural networks

Matthew Greenberg

May 2022

This course

1. Introduction
2. Convolutional neural networks (CNNs) for computer vision
3. Text processing and recurrent neural networks (RNNs)

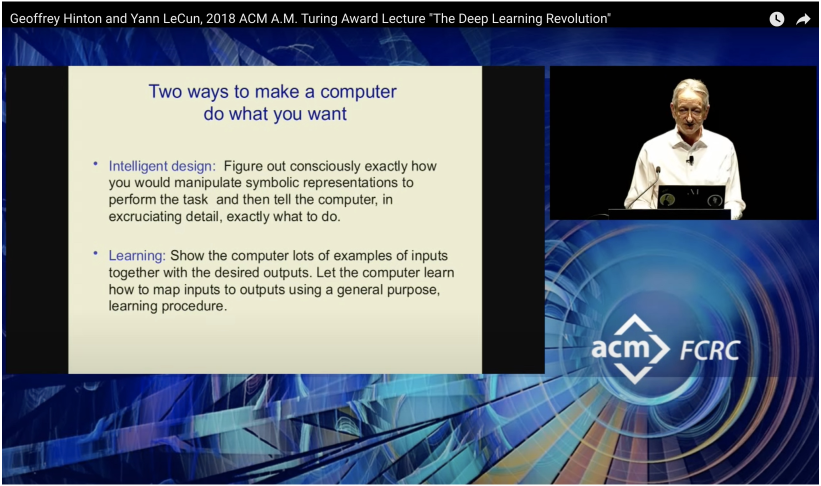
Deep learning

- Probabilistic models based on *neural networks*.
- Transformative for *computer vision*, *natural language processing*, automation, robotics, molecular modeling, simulation...
- Benefits of deep learning most pronounced for analysis of *unstructured, high-dimensional data*.

Using deep learning powered APIs and packages

- Many companies sell access deep learning powered APIs for common data processing and analysis tasks.
- Let's look at how to use Google's cloud vision API...
- Instead of accessing the model via API, you can download it and incorporate it into your code.
- Let's download a transformer model for extractive question answering from HuggingFace...

Artificial intelligence (AI)



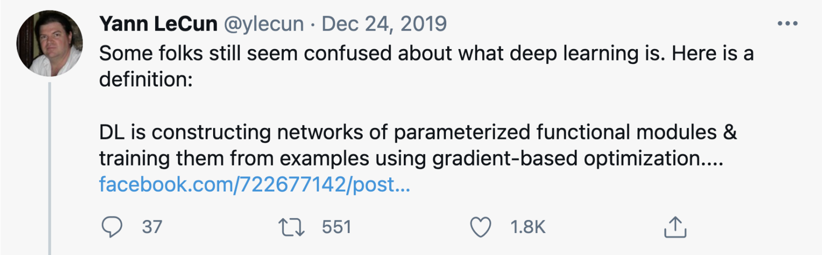
https://www.youtube.com/watch?v=VsnQf7exv5I (11:49)

Machine learning (ML)

Show the computer lots of examples of inputs together with the desired outputs. Let the computer learn how to map inputs to outputs using a general purpose learning procedure.

input	**output**	**input**	**output**
image	*label*	*English text*	*French translation*
	deer	Can I borrow your magazine?	Puis-je emprunter votre magazine?
	frog	Please pass the ketchup.	Veuillez passer le ketchup.
	truck	His shirt was made in turkey.	Sa chemise a été confectionnée en Turquie.

Deep learning (DL)



Networks

Traditional machine learning

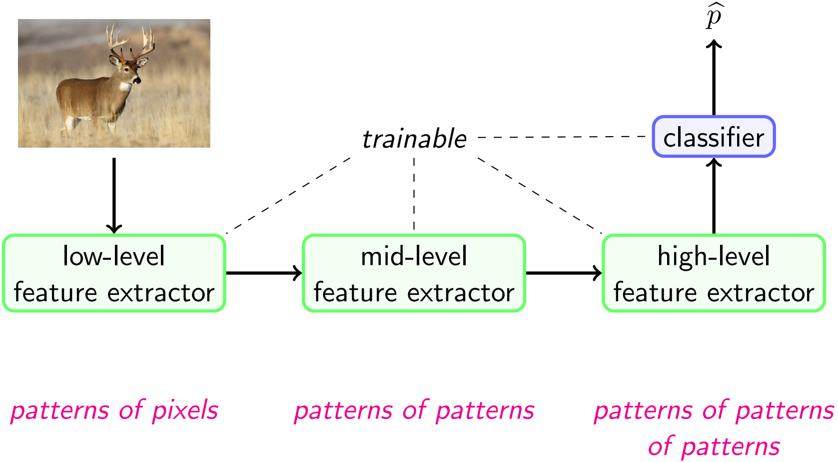
$\rightarrow$ hand-engineered feature extractor $\rightarrow$ trainable classifier $\rightarrow$ $\widehat{p}$

Deep learning

$\rightarrow$ trainable feature extractor $\rightarrow$ trainable classifier $\rightarrow$ $\widehat{p}$

- $\hat{p}$ is a vector of class probabilities.

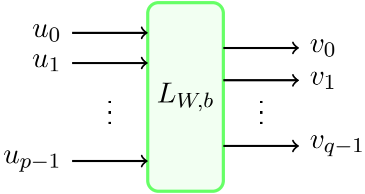
Feature hierarchy



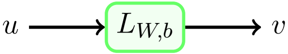
Linear layers

Trainable parameters:

- **weights:** $W \in \mathbb{R}^{p \times q}$
- **biases:** $b \in \mathbb{R}^{q}$



More concisely:



$$v = L_{W,b}(u) = uW + b$$

Compositions of linear layers are linear

$$u \longrightarrow \boxed{L_{W_1,b_1}} \longrightarrow \boxed{L_{W_2,b_2}} \longrightarrow v$$

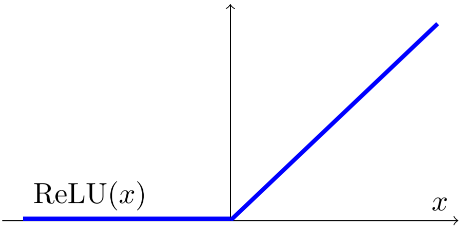
$$\begin{aligned}
L_{W_2,b_2}\big(L_{W_1,b_1}(u)\big) &= L_{W_2,b_2}(uW_1 + b_1) \\
&= (uW_1 + b_1)W_2 + b_2 \\
&= u\underbrace{W_1W_2}_{W} + \underbrace{b_1W_2 + b_2}_{b} \\
&= L_{W,b}(u)
\end{aligned}$$

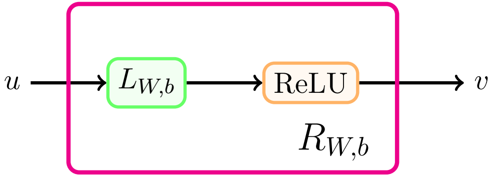
$$u \longrightarrow \boxed{L_{W,b}} \longrightarrow v$$

ReLU-activated linear layers

Rectified linear unit:

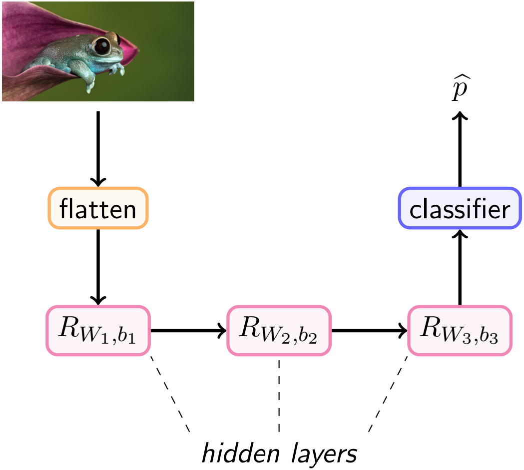
$$\mathrm{ReLU}(x) = \max(x, 0)$$





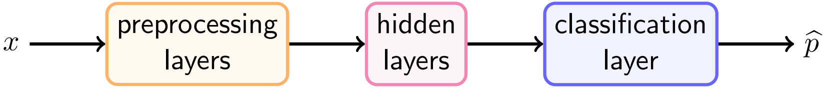
$$R_{W,b}(u) = \mathrm{ReLU}\left(L_{W,b}(u)\right)$$ (apply ReLU componentwise)

A simple classification network

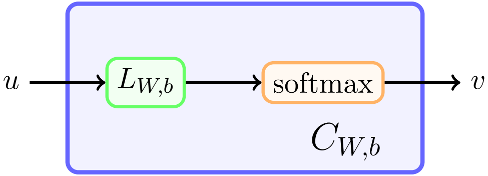


- The **flatten** layer flattens an RGB image tensor with shape $m \times n \times c$ to an mnc-dimensional vector.

The classification layer: softmax-activated linear layer



- The classification layer of a K-class classification network typically performs **logistic regression**, mapping features output by the hidden layers onto class probabilities.



The softmax function

- The **softmax function** converts "raw scores" $u_0, \ldots, u_{p-1}$ into a **distribution vector**, i.e. a vector with nonnegative components that sum to 1:

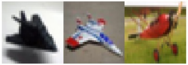
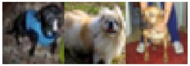
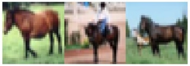
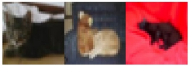
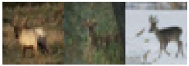
$$\operatorname{softmax}(u_0, \ldots, u_{p-1}) = \frac{1}{\sum_j e^{u_j}} \left(e^{u_0}, \ldots, e^{u_{p-1}}\right)$$

Using pretrained image classification models

- You can download many pretrained image classification models.
- Most are trained on the **ImageNet dataset**, with
  - 1000 object classes,
  - > 1.2 million training images

Example: the CIFAR-10 dataset

- 60000 RGB images, 32×32 pixels
- 10 classes

label/class	**examples**	**label/class**	**examples**
0/airplane		5/dog	
1/automobile		6/frog	
2/bird		7/horse	
3/cat		8/ship	
4/deer		9/truck	

An image classifier for CIFAR-10

$$x \longrightarrow \boxed{f_\theta} \longrightarrow \widehat{p}$$

- input x: tensor with shape $32 \times 32 \times 3$
- target: $y \in \{0, 1, \ldots, 9\}$
- output $\widehat{p}$: vector of length 10, where

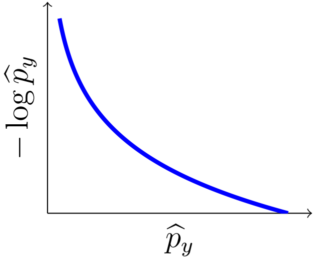
$$\widehat{p}_j = \text{probability that } x \text{ has class } j$$

- trainable parameters: θ, initialized randomly

Training the classifier

- loss function: penalize misclassifications

$$L(y, \widehat{p}) = -\log \widehat{p}_y$$



- Adjust parameters to decrease loss via

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(y, \widehat{p})$$

(α is a positive hyperparameter called the **learning rate**.)

Batches and epochs

- One training **epoch**:
  - Partition training data into N/I **batches** of size I.
  - for $0 \leq k \leq N/I$:
    - Compute
    $$\widehat{p}_i = f_\theta(x_i), \quad kI \leq i < (k+1)I.$$
    - Adjust parameters to decrease average loss over the batch via
    $$\theta \leftarrow \theta - \alpha \nabla_\theta \left\{ \frac{1}{I} \sum_{bI \leq i < (b+1)I} L(y_i, \widehat{p}_i) \right\}.$$
- Train multiple epochs, until convergence.

```
model.fit(x_train, y_train, batch_size=100, epochs=20);
```

```
Epoch 1/20
500/500 [==============================] - 1s 2ms/step - loss: 2.0992 - accuracy: 0.2314
Epoch 2/20
500/500 [==============================] - 1s 2ms/step - loss: 1.8704 - accuracy: 0.3424
Epoch 3/20
500/500 [==============================] - 1s 2ms/step - loss: 1.8263 - accuracy: 0.3621
```

⋮

```
Epoch 18/20
500/500 [==============================] - 1s 2ms/step - loss: 1.7248 - accuracy: 0.4058
Epoch 19/20
500/500 [==============================] - 1s 2ms/step - loss: 1.7164 - accuracy: 0.4129
Epoch 20/20
500/500 [==============================] - 1s 2ms/step - loss: 1.7209 - accuracy: 0.4062
```