Neural Networks II

Data Structures

Structures of Data in Real World

Data in real world often have structures

- Continuity in space
- Continuity in time
- Causal (and noncausal) relationships
- ...

Dismentling Structures: Restoring Independence for Inference

Structures in data often lead to dependency

Most statistical inference require independence

Sometimes we have to dismentle the structures for proper inference

- Blocking or stratification
- Washout periods between experiments
- Randomized controlled trials
- ...

Harnessing Structure: Architectures that Generalize

Sometimes knowing the structures help

- Image classification (convolutional neural networks)
- Language translation (attention and transformer)
- Social network anslysis (graph neural networks)
- ...

Main message for today: Architectures that encode prior knowledge need *fewer* parameters and generalize better.

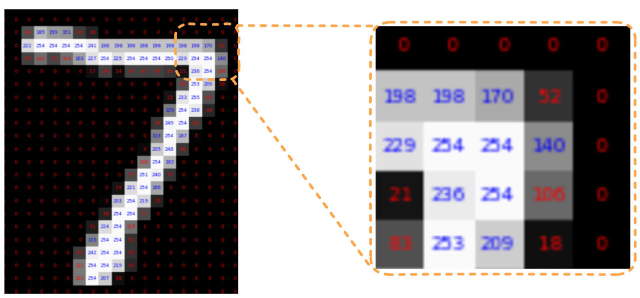
Convolution Neural Networks

Example: Image classification

Structures of Images

- Local continuity for most pixels
- Objects/parts are more important than single pixel for prediction
- ...

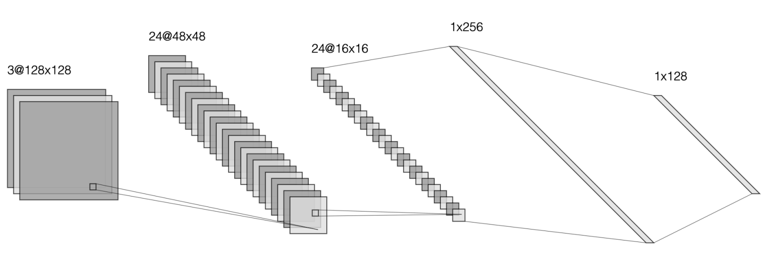
Images in Computers' Eyes



- Input: $\mathbf{X} \in \mathbb{R}^{m \times m}$
  - $x_{i,j}$: integer between 0 and 256
  - $\mathbf{X} \in \mathbb{R}^{m \times m \times 3}$ for colored image (RGB)
- Goal: Classify the image into digits $(0-9)$

Images from the Modified National Institute of Standards and Technology database (MNIST) dataset

Convolutional Neural Networks



Immage made with NN-SVG

Convolution Layer

Recall that a layer of MLP takes the form

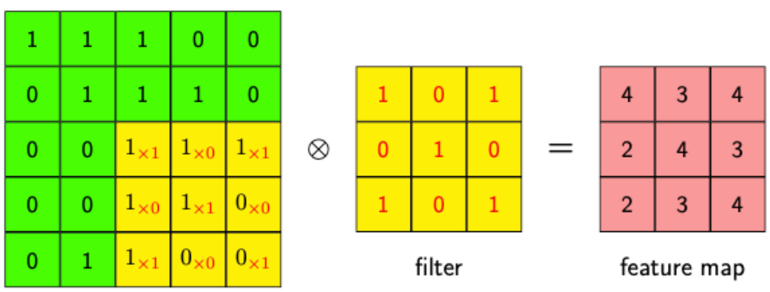
$$\mathbf{z}^{(l)} = \phi\left(\mathbf{W}^{(l)}\,\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}\right),$$

A convolution layer replaces the big matrix multiplication $\mathbf{W}^{(l)} \in \mathbb{R}^{p\times p}$ or $\mathbf{W}^{(l)} \in \mathbb{R}^{p\times W^2}$ with a *convolution*, for each i, j,

$$z_{ij}^{(1)} = \sum_{u=1}^{k}\sum_{v=1}^{k} w_{uv}^{(k,c)}\; x_{i+u,j+v}^{c} + b^k$$

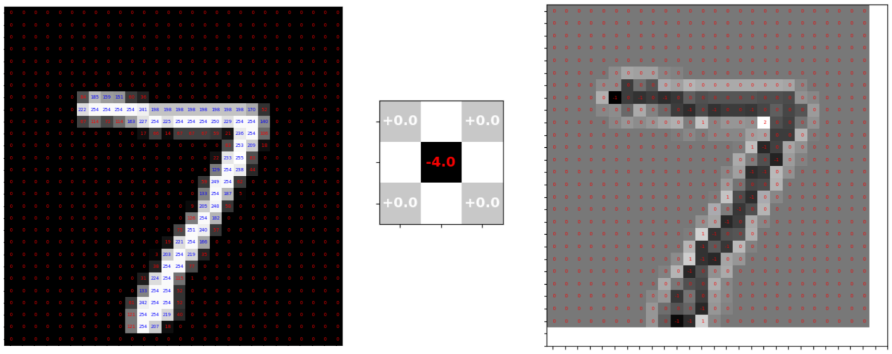
- Local *receptive field*: filter size $k \times k$ (e.g. 3×3) looks only at nearby pixels.
- Weight sharing: the same kernel slides across the image
- Usually followed by a ReLU activation

Convolution



- Input dimension M (often **W**idth or **H**eight)
- Kernel/filter size K (typically 3 or 5)
- Stride S
- Padding P
- Output dimension $(M + 2P - K)/S + 1$

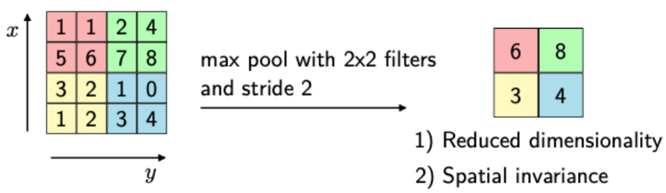
Convolution Layer: Example



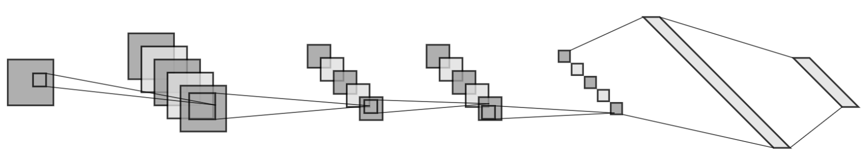
Pooling Layer

Pooling: Down-sample while keeping the most salient signal.

- Max-Pool: $z_{ij} = \max_{u,v} x_{(i+u)(j+v)}$ (translation/shift invariance)
- Average-Pool $z_{ij} = k^{-2} \sum_{u,v} x$ (smooths signals)



Convolutional Neural Networks: Example



Immage made with NN-SVG

CNN v.s. FNN

In the example, you will find

1. A convolutional neural network

```
self.conv = nn.Sequential(
            nn.Conv2d(1, 32, 3, padding=1), nn.ReLU(),
```

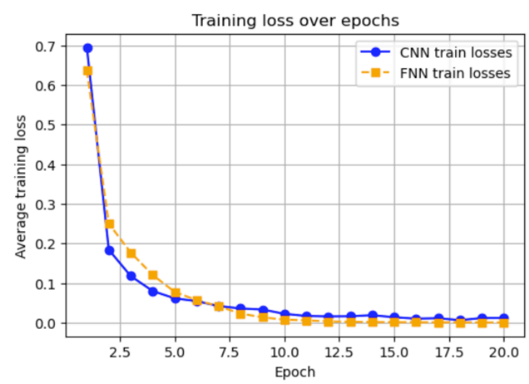
```
            nn.MaxPool2d(2),                     # 14×14
            nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(),
            nn.MaxPool2d(2)                      # 7×7
        )
        self.fc = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64*7*7, 128), nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 10)
        )
```

2. An FNN/MLP

```
self.net = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 512), nn.ReLU(),
            nn.Linear(512,  32), nn.ReLU(),
            nn.Linear( 32,  10)
        )
```

Performance

- Number of parameters (trainable weights)
  - CNN: 421,642
  - FNN: 418,666
- Train losses look fine for both models
- Misclassification on test sets:
  - CNN: 18.1%
  - FNN: 47.1%

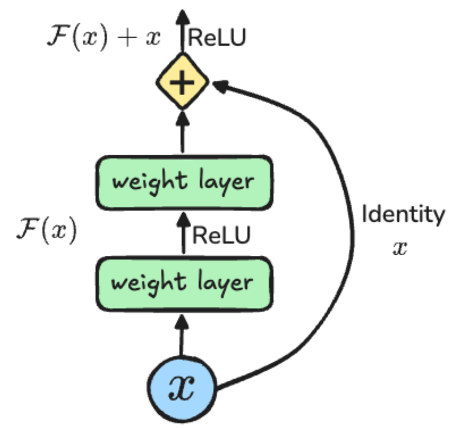


Convolutional Neural Networks: Milestones

Year	Architecture	# Layers
1998	LeNet-5	5
2012	AlexNet	8
2014	VGG	19
2015	ResNet	**152**
2017	SENet	**152**

Key to Go Deep: ResNet (He et al. 2015)

- Prior to ResNet: deeper models yield larger training error
- Contradictory to known facts:
  - Deeper models can capture more complex mappings
  - CNNs already pass the interpolation threshold (second descent) (AlexNet has around 60 million parameters)
- Hypothesis: large training error is a result of failures in optimization
  - Possible cause: vanishing gradients in backpropogation
- Solution: explicitly include an identity mapping to let gradient flows
  Desired mapping $H(x)$ "Residual"/nonlinear mapping $F(x)$



Key to Go Deep: Dataset

- ImageNet
- CIFAR-10 and CIFAR-100
- Torchvision ships with many models with pre-trained weights

Attention and Transformer

Example: Translation

Machine Translation Examples

French to English

- **FR:** Demain, je donnerai le livre à mon ami.
- **EN:** I will give the book to my friend tomorrow.

Spanish to English

- **ES:** Ayer, María le envió una carta a su hermano desde México.
- **EN:** Maria sent her brother a letter from Mexico yesterday.

Chinese to English

- **ZH:**
- **EN:** I borrowed an interesting book from the library.

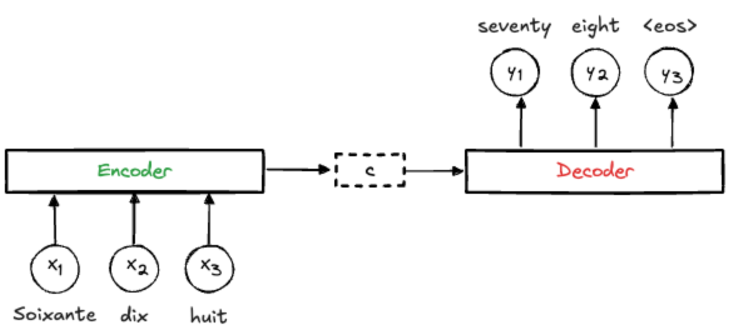
Text in the eyes of computers: Tokenizer playground

French numbers to English using Transformer

Sequence-to-Sequence Task

- Machine Translation (e.g., French to English sentence translation)
- Music Generation (input: musical themes or motifs, output: extended music sequences)
- Code Generation from Natural Language (input: problem description, output: source code)
- ...

Key idea: **Encode** the input into a context state c then **decode** step-by-step with RNN/Transformer.



Link to illustration

Recurrent Neural Networks

Encoder

$$\mathbf{h}_t = f\big(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}\big),$$

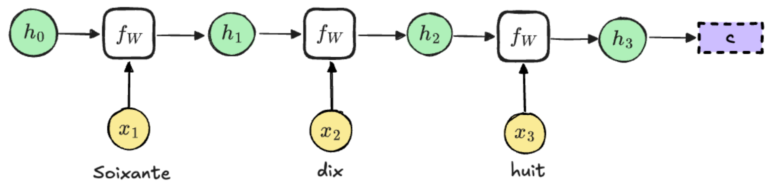
Decoder

$$\mathbf{s}_t = g\big(\mathbf{U}_{sc}\mathbf{c} + \mathbf{U}_{sy}\mathbf{y}_{t-1} + \mathbf{U}_{ss}\mathbf{s}_{t-1}\big)$$

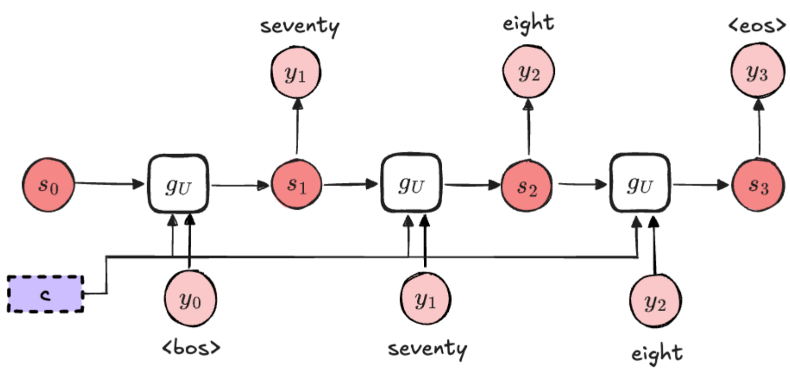
- Shares parameters across all time steps
- Trained with *Back-Propagation Through Time* (BPTT).
- Suffers from gradient vanishing for long T.

Recurrent Neural Networks

Encoder



Decoder



[Link to illustration]

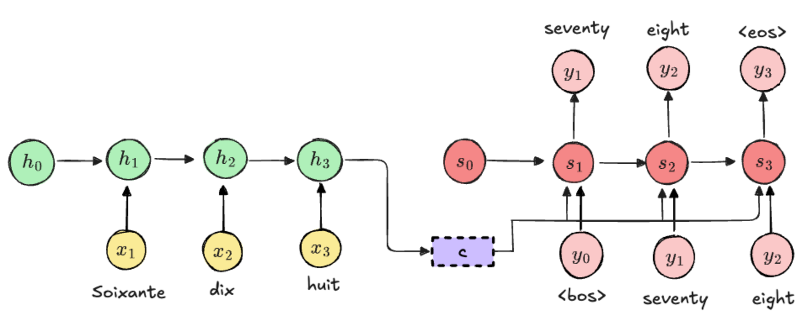
Problem with the Context

One problem with using RNN is that

- All information of the inputs are stored in one context vector c.
- All decodings are based on c

This could have a few

- Might fail for long sequences
- Might not be optimal for sequential decoding



[Link to illustration]

Attention: Step 1

1. Compute alignment scores $e_{1,i} \in \mathbb{R}$

$$e_{1,i} = f_{\text{att}}(s_0, h_i)$$

2. Apply softmax to obtain attention weights (weights sum up to 1)

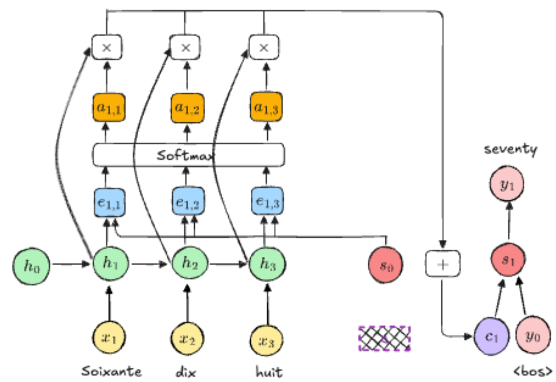
$$a_{1,i} = \frac{\exp(e_{1,i})}{\sum_{j=1}^{3} \exp(e_{1,j})}$$

3. Compute context vector as the weighted sum of hidden states

$$c_1 = \sum_{i=1}^{3} a_{1,i} h_i$$

4. Decode the new state

$$s_1 = g\big(\mathbf{U}_{sc} c_1 + \mathbf{U}_{sy} y_0 + \mathbf{U}_{ss} s_0\big)$$



[Link to illustration]

Attention: Step t

1. Compute alignment scores $e_{t,i} \in \mathbb{R}$

$$e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$$

2. Apply softmax to obtain attention weights (weights sum up to 1)

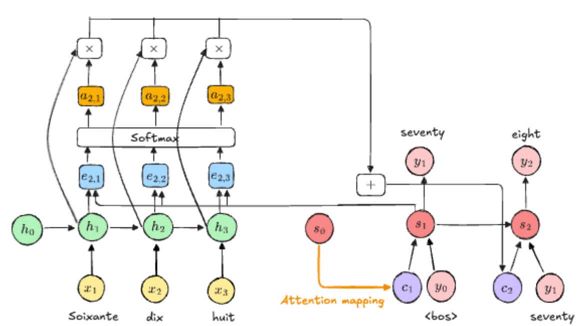
$$a_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{3} \exp(e_{t,j})}$$

3. Compute context vector as the weighted sum of hidden states

$$c_t = \sum_{i=1}^{3} a_{t,i} h_i$$

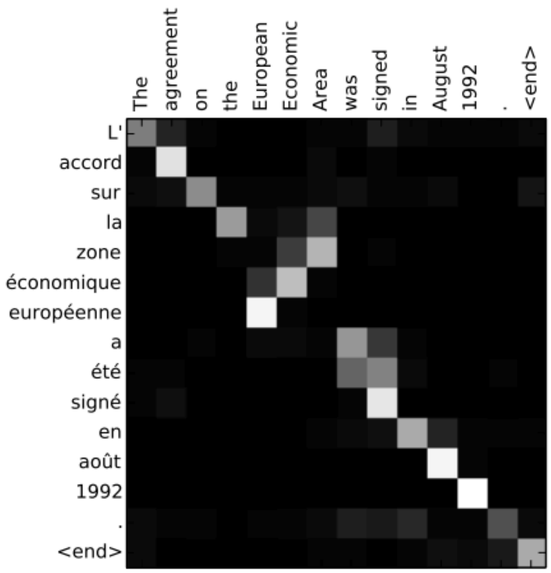
4. Decode the new state

$$s_t = g\big(\mathbf{U}_{sc} c_t + \mathbf{U}_{sy} y_{t-1} + \mathbf{U}_{ss} s_{t-1}\big)$$



[Link to illustration]

Attention: Example



Source: Figure 2(a) from Bahdanau et al. (2015)

Self-Attention

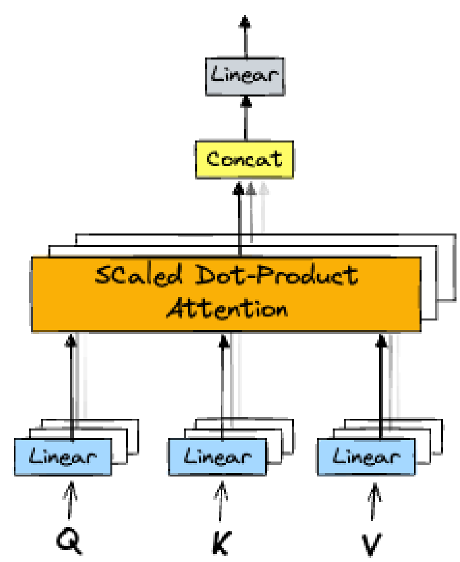
Observation: Hidden states and decoding states are both functions of input $\mathbf{X}$

Self-Attention: new architectures based on the input $\mathbf{X}$

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V,$$

where the query vector Q, key vector K, and value vector V are all functions of input $\mathbf{X}$

For more detail, see Lecture 8 of Stanford CS231n by Fei-Fei Li et al.



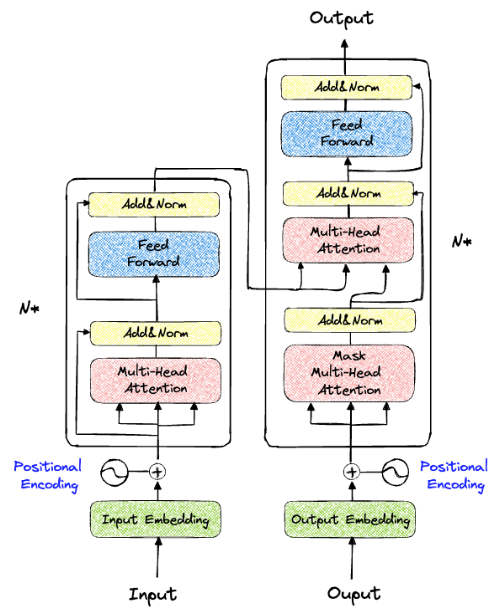
Link to illustration

Transformer

Stacking multi-head self-attention layers yields the **Transformer** (Vaswani et al., 2017).

Key advantages:

- Parallelizable (vs. sequential RNN).
- Captures long-range dependencies.
- Scales with data



Link to illustration

Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

- Pretraining (foundation model) & Fine-tuning (for specific tasks)
- Interactive demo: The Annotated Transformer.

Summary

Real-world data come with *structure*:

- **Grid-like** → images (2-D pixels), audio spectrograms (1-D or 2-D).
- **Sequential** → text, time-series, DNA.

Neural layers that *leverage* structures learn faster and generalize better:

Data type	Good default layer	Key idea
Grid	Convolution	local receptive field & weight sharing

Data type	Good default layer	Key idea
Sequence	Recurrent / Transformer	hidden state or attention over positions