Architectures = prior on the function space

Change of paradigm:
- classical ML: design features by hand
- deep RL: meta-design of features → design architectures able to express features to learn

An architecture:

Random projections

Do dimensionality reduction

Light 6n

Sensors

coding protocol, random p(x)

R*

Rut

activities

as many as neurons

ReLU

x

activites = random proj. p*(x)
In some cases most of the performance is due to the architecture and not the training.

(Extreme ML)
- Work only for small architectures because the architecture was found beforehand.

\[ x \xrightarrow{\text{random weights}} \text{sum (one layer)} \xrightarrow{\text{random non-linear Features}} \]

\[ \text{if-random weights: good reconstruction of } x \rightarrow \text{keep more information} \]
\[ \text{if-trained: kind of work (anti-space) } \rightarrow \text{keep only information relevant for the task} \]

For deeper networks:
- In each layer: 90% of neurons: randomly initialized (artificial)
- 10% are trained

\[ x \xrightarrow{\text{architecture ideas}} \]

Architectures
- CNN: move to Fourier space

\[ \text{Image input} \xrightarrow{\text{Fourier}} \]

\[ \text{add shallow layers } M^4 M^3 \ldots \text{no gain in expressive power} \]

\[ \text{But the distribution over Fourier space is different} \]
\[ M \sim \mathcal{L}_2 \quad M \sim \mathcal{N}(0,1) \sim p(F) \]
\[ M^4 M \]

Random initialization:
- random: but which law?
Exploding/Vanishing Gradients

\[
\left| \begin{array}{c}
-1 \\
\times
\end{array} \right| \rightarrow x
\]

\[
\text{Layer 1} \rightarrow \text{Layer 2} \rightarrow x
\]

amplitude \rightleftharpoons \text{input amplitude}

\[
\|x\| \rightarrow \gamma \|x\|
\]

\[
\gamma = 10
\]

\[
\text{if 10 layers}
\]

\[
\rightarrow \text{issues}
\]

\[
F \approx 0
\]

\[
\text{Some issue with gradients}
\]

\[
\frac{1}{\gamma} \approx 0 \rightarrow \text{not possible to train}
\]

\[
\frac{1}{\gamma} \approx 0
\]

\[
\text{Xavier Glorot's initilization}
\]

\[
x_1 \rightarrow \cdots \rightarrow x_n
\]

\[
\text{Inputs}
\]

\[
\text{0-}\gamma 1 \text{ neuron}
\]

\[
\text{Weights}
\]

\[
\text{Hidden inputs}
\]

\[
\text{Mean: } \mathbb{E} \mathbb{E}[w] = \mathbb{E}\left[ \sum_i w_i x_i \right] = 0
\]

\[
\text{Variance: } \mathbb{E} \mathbb{E}\left[ s^2 \right] = \mathbb{E}\left[ \sum_i w_i x_i \right]^2 = \mathbb{E}\left[ \sum_i w_i^2 x_i^2 \right] (\text{to})
\]

\[
= d \mathbb{E}[x^2]
\]

\[
\text{Width of activation function } \gamma:
\]

\[
\text{ReLU}
\]

\[
\rightarrow \text{variance } \frac{\gamma}{2}
\]

\[
\Rightarrow \times \sqrt{2}
\]

\[
\text{activating a specific correlated factor}
\]

\[
\text{Bases: initialized to 0}
\]

\[
\text{Other approaches: Kaiming He etc.}
\]

\[
\left| \begin{array}{c}
1 \\
\times
\end{array} \right| \rightarrow \left| \begin{array}{c}
1 \\
\times
\end{array} \right| \rightarrow \text{ReLU}
\]

\[
\text{Forward pass}
\]

\[
\text{Backprop}
\]
\[ J = \frac{df}{dx} \] Jacobian properties

\[ \text{well-schur at init} \]

\[ \text{matrix dim objects} \times \text{dim inputs} \]

\[ \frac{\partial}{\partial x} \]

\[ x \rightarrow y \rightarrow \mathbf{D} \rightarrow y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4 \]

\[ \text{var}(\frac{df}{dx}) = \frac{1}{n_0} \]

\[ \text{var}(\frac{df}{dx}) = \mathbb{E} \left[ \left( \frac{df}{dx} \right)^2 \right] \]

\[ \alpha \leq \frac{1}{\beta} \leq \frac{1}{\gamma} \]

\[ \Rightarrow \text{what matters: } \sum \mathbf{v}_i \text{ as small as possible} \]

\[ \Rightarrow \text{better: as many neurons as possible} \]

\[ \Rightarrow \text{avoid thin layers} \]

\[ \Rightarrow \text{consider same-width layers} \]

\[ \Rightarrow \text{avoid thin layers} \]

\[ \text{Design easy-to-train architectures} \]

\[ \text{Deeper architectures: more difficult to train} \]

\[ \text{Normalization} \]

- SELU activation \( P \): nice properties:
  - keeps the variance

\[ x \rightarrow y \rightarrow \mathbf{D} \rightarrow y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_4 \]

\[ \text{update? } \delta W = \frac{\partial}{\partial W} \frac{dx}{\partial x} = \frac{\partial}{\partial x} \frac{dx}{\partial x} \]

\[ g = \mathbb{E}[x] \]

\[ \text{Learning rate is lost} \]

\[ \text{Set init low enough} \]

\[ \sum \text{for few layers} \]

\[ \text{Scaling parameters} \]

\[ \frac{A_y - A}{\epsilon} + b \]

\[ \text{batchnorm} \]

\[ \text{instance-norm} \]

\[ \text{layer-norm} \]

\[ \text{Fixup init} \]
Architecture 600

- CNN: convolutions

2D image \( x \times N \times B \) in grid

Local filters applied at all locations

Features

\[ \text{input size} \rightarrow \text{num filters} \rightarrow \text{filter size} \rightarrow \text{output size} \]

\( \times \text{num filters} \)

\( \times \text{num meshes} \)

Standard block in computer vision

Repeat

Divide by resolution/2

Hierarchical model

More "agnostic" less "accidental"

Anti-encoders: unsupervised setup

"Self-supervised": build a supervised task from codes built from the input

Input \[ \times \]

Smaller-dim representation (dim reduction)

Training criterion:

\[ \| x - x' \| \]

Generative models: cf. chapter 6
Segmentation

Optimization

ResNet

\[ x' = x + F(x) \]

\[ g = F_2(x_2) + F_2(x_3) + F_2(x) + x \]

\[ \frac{\partial L}{\partial w} = -\frac{1}{3\gamma} \left( \ldots \right) + \frac{\partial L}{\partial F_1} \frac{\partial F_1}{\partial w} \]

\[ \Rightarrow \text{much faster training} \]

Auxiliary losses

Inception

\[ x \rightarrow \text{Inception} \rightarrow x' \]

Graph-MN

Task: 1 input = a graph

Examples:
- molecule
- social graph
- 3D simulation mesh
Spectral methods

Fourier domain

\[\text{CNN} \rightarrow \mathcal{M}(p) \times (p-p') \text{ d}p\]

\[\Delta \text{ (Laplacian)} \]

\[\Delta = \text{eigenvalues of } \Delta \]

\[\text{choice of mask} \rightarrow \text{multiplication of Fourier transforms} \]

\[\Delta \text{ n-neighborhood} \]
Attention mechanism

**Basic:**

- \( g = w_a x + w_b b \)
- If \( c = 0 \): \( x_a + 0 b \)
- If \( c = 1 \): \( 0x + 1b \)
- \( w_a = f(c) \)

**Many inputs:**

- \( x \in \text{context} \)
- \( x = \text{softmax} \)
- \( x \in \mathbb{R} \)

\( \sum_{x} x = 1 \)

**Keys:**

- \( \text{keys} \) and \( \text{values} \)
- \( \text{describe instances} \)
- \( \text{state their importance} \)

**Common use (Transformers):**

- \( k, v, x \)
- \( \text{similarities between inputs} \)
- \( \text{softmax} \)
- \( \text{weights} \)
Fully-connected graph:

- each word = a node

\[ \Rightarrow \text{graph attention network (GAT)} \]

\[ \Rightarrow \text{listen to all nodes according to how similar they are to you} \]

Recurrent networks:

- involve time;
- input time series \( x = (x_1, x_2, ..., x_t) \)

\[ s_t \] \[ \text{message (history summary)} \] \[ \text{output} \]

\[ \uparrow \] \[ \uparrow \] \[ \uparrow \]

\[ x_1 \] \[ x_2 \] \[ x_3 \] \[ x_t \]

- all tasks are identical; share the same parameters

Block: based on attention

\[ \text{GRU, LSTM} \]

message

\[ x_t \]

- combine message
- \( a \) message
- \( + (a)(x_t) \)

\[ a = g(\text{message}, x_t) \]

"optimising the message"

- \( a \approx 1 \)
- \( a \approx 0 \) for very important new event