Deep Learning

Lecturer: Hofmann Thomas

Scribe: Yao Zhang

https://zhims.github.io/datascience.html

Test version, updated October 28, 2019

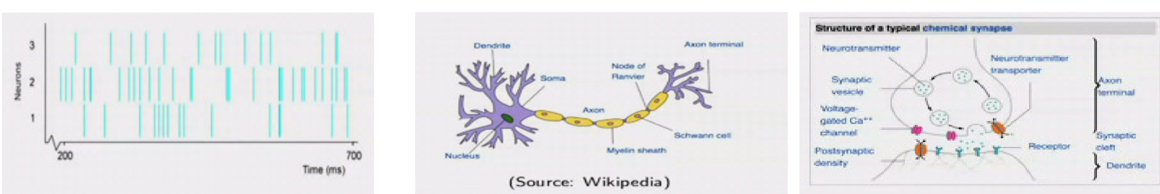
1 Deep Neural Networks

Biological neural networks:

- Neurons: basic functional & structural units of nerous system
- Cells connected by nervous fibers
- Signaling via electrical impulses
- Human brain:
  - $\sim$ 100 billion neurons
  - $\sim$ 100 trillion connections
  - very large scale system

Figure 1

Biological neural networks:



(a) Neurons (b) Anaomy (c) Synapses

Figure 2

- Neurons: all-or-none principle = action potential (spike)
- Anatomy: Soma, dendrite, axon
- Functional: many inputs ($\sim 10^3 - 10^5$), one output
- Synapses: plasticity (strengthening & weakening)

Connectome

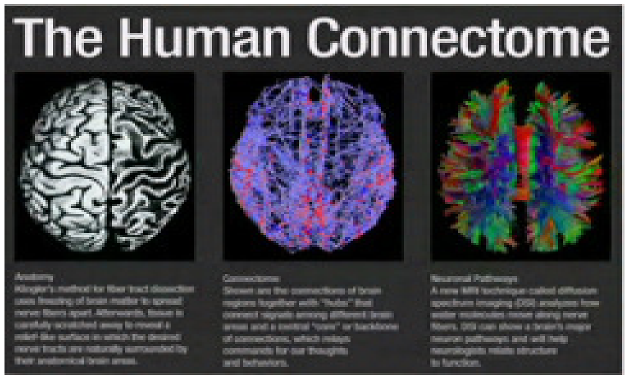


Figure 3

- Scientific challenge: decipher brain connectivity
- Small scale connectivity vs. overall "wiring" (white matter pathways)
- Network analysis: Rich club(2001) $\downdownarrows$

1.1 Backpropagation

Learning in neural network = gradient-based optimization (with very few exception).

Definition 1.1. *Gradient of objective with regard to parameters* θ :

$$\nabla_\theta \mathcal{R} = \begin{pmatrix} \frac{\partial \mathcal{R}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \mathcal{R}}{\partial \theta_d} \end{pmatrix} \tag{1.1}$$

Definition 1.2. *Steepest descent and stochastic gradient decent*

$$\theta\left(t+1\right) \leftarrow \theta\left(t\right) - \eta \nabla_\theta \mathcal{R}\left(\mathcal{S}\right) \tag{1.2}$$

where

1. *here* t = *0,1,2,... is an iteration index*
2. $\mathcal{S}$ = *all training data* $\Rightarrow$ *steepest descent*
3. $\mathcal{S}$ = *mini batch data* $\Rightarrow$ *SGD*

Computational challenge: how to to compute $\nabla_\theta \mathcal{R}$? Exploit composition structure of network = *backpropagation.*

Basic steps:

1. perform a forward pass (for given training input X) ro compute activations for all units
2. compute gradient of $\mathcal{R}$ with respect to (w.r.t) output layer activations (for given target y)
3. iteratively propagate activation gradient information from outputs to inputs
4. compute local gradients of activations w.r.t weights

Vector-valued function (map) $F : \mathbb{R}^n \to \mathbb{R}^m$: each component function has gradient $\nabla F_i \in \mathbb{R}^n, i \in \{1, 2, ..., m\}$

Definition 1.3. *(Jacobin matrix)*

$$J_F \triangleq \begin{pmatrix} \nabla^T F_1 \\ \nabla^T F_2 \\ \vdots \\ \nabla^T F_m \end{pmatrix} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \frac{\partial F_m}{\partial x_2} & \cdots & \frac{\partial F_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{1.3}$$

derivate of outputs with regard to inputs, i.e. $(J_F)_{ij} = \frac{\partial F_i}{\partial x_j}$.

Vector-valued function $G : \mathbb{R}^n \to \mathbb{R}^q$, $H : \mathbb{R}^q \to \mathbb{R}^m, F \triangleq H \circ G$. Componentwise rule:

$$\frac{\partial F_i}{\partial x_j}|_{x=x_0} = \frac{\partial (H \circ G)_i}{\partial x_j}|_{x=x_0} = \sum_{k=1}^{q} \frac{\partial H_i}{\partial z_k}|_{z=G(x_0)} \cdot \frac{\partial G_k}{\partial x_j}|_{x=x_0} \tag{1.4}$$

Lemma 1.1.1 (Jacobi matrix chain rule)**.**

$$J_{H \circ G}|_{x=x_0} = J_H|_{z=G(x_0)} \cdot J_G|_{x=x_0} \tag{1.5}$$

Proof. content... □

2 Convolutional Neural Networks

Definition 2.1 (Integral operator)**.** *A transform T expressible with the kernel H and $t_1, t_2 \in \mathbb{R} \bigcup \{-\infty, \infty\}$ such that for any function f (for with Tf exists)*

$$(Tf)(u) = \int_{t_1}^{t_2} H(u, t) f(t) dt \tag{2.1}$$

is called an integral operator.

Example 2.1 (Fourier transform)**.**

$$(\mathcal{F}f)(u) \triangleq \int_{-\infty}^{\infty} e^{-2\pi i t u} f(t)\, dt \tag{2.2}$$

Definition 2.2 (Convolution). *Given two functions f, h, their convolution is defined as*

$$(f * h)(u) \triangleq \int_{-\infty}^{\infty} h(u-t) f(t)\, dt = \int_{-\infty}^{\infty} f(u-t) h(t)\, dt \tag{2.3}$$

Remark 2.1.

1. *integral operator with kernel $H(u,t) = h(u-t)$*
2. *shift-invariant as $H(u-s, t-s) = h(u-t) = H(u,t) \quad (\forall s)$*

   Proof. content... □
3. *convolution operator is commutative*

   Proof. content... □
4. *existence depends on properties of f, h*
5. *typical use $f =$ signal, $h =$ fast decaying kernel function*

Definition 2.3 (Linear transform). *T is linear, if for all functions f, g and the scalars α, β,*

$$T(\alpha f + \beta g) = \alpha T f + \beta T g \tag{2.4}$$

Definition 2.4 (Translation invariant transform). *T is translation (or shift) invariant, if for any f and scalar τ,*

$$f_\tau(t) \triangleq f(t+\tau), \quad (T f_\tau)(t) \triangleq (Tf)(t+\tau) \tag{2.5}$$

Remark 2.2. *content...*

Theorem 2.1. *Any linear, translation-invariant transformation T can be written as convolution with a suitable h.*

Proof. content... □

Signal processing with neural networks:

1. Transforms in deep networks: linear + simple non-linearity
2. Many signals (audio, image, etc.) obey translation invariance ⇒ invariant feature maps: shift in input = shift in feature map

1 + 2 in above:

1. ⇒ learn convolutions, not (full connectivity) weight matrices
2. ⇒ convolutional layers for signal processing

For all practical purposes: signal are sampled, i.e. discrete.

Definition 2.5 (Discrete convolution (1-D)). *For $f, h : \mathbb{Z} \to \mathbb{R}$, we can define the discrete convolution via*

$$(f * h)[u] \triangleq \sum_{t=-\infty}^{\infty} f[t]\, h[u-t] \tag{2.6}$$

Remark 2.3.

1. *use of rectangular brackets to suggest "arrays"*
2. *2D case:*
$$\text{content...} \tag{2.7}$$
3. *typical: h with finite support (window size)*

Example 2.2. *Small Gaussian kernel with support* $[-2:2] \subset \mathbb{Z}$

$$h[t] = \frac{1}{16}\begin{cases} 6 & t=0 \\ 4 & |t|=1 \\ 1 & |t|=2 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

Consequence: convolution sum can be truncated:

$$\begin{aligned}(f*h)[u] &= \sum_{t=u-2}^{u+2} f[t]\,h[u-t] = \sum_{t=-2}^{2} h[t]\,f[u-t] \\ &= \frac{6f[u]\ +4f[u-1]+4f[u+1]+f[u-2]+f[u+2]}{16}\end{aligned} \tag{2.9}$$

Remark 2.4. *content...*

Definition 2.6 (Discrete cross-correlation)**.** *Let* $f,h:\mathbb{Z}\to\mathbb{R}$*, then*

$$(h*f)[u] \triangleq \sum_{t=\infty}^{\infty} h[t]\,f[u+t] \tag{2.10}$$

Remark 2.5.

1. *Def. 2.6 also called a "sliding inner product",* $u+t$ *instead of* $u-t$
2. *note that cross-correlation and convolution are closely related:*

$$\begin{aligned}(h*f)[u] &= \sum_{t=\infty}^{\infty} h[t]\,f[u+t] \\ &= (h*f)[u] = \sum_{t=\infty}^{\infty} h[-t]\,f[u-t] \\ &= \left(\overline{h}*f\right)[u] \\ &= \left(f*\overline{h}\right)[u]\end{aligned} \tag{2.11}$$

where $\overline{h}[t] \triangleq h[-t]$.

Only difference: kernel flipped over, but not non-commutative.

Convolution via matrices:

1. In practice: signal f and kernel h have finite support
2. Without loss of generality (w.l.o.g) $f[t]=0$ for $t\notin[1:n]$, $h[t]=0$ for $t\notin[1:m]$

3. We can think of f and h as vectors and define:

$$(f * h) = \underbrace{\begin{pmatrix} h_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ h_2 & h_1 & 0 & 0 & \cdots & 0 & 0 \\ h_3 & h_2 & h_1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & h_m & h_{m-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & h_m \end{pmatrix}}_{\triangleq H_n^h \in \mathbb{R}^{(n+m-1)\times n}} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \tag{2.12}$$

Remark 2.6. *content...*

Definition 2.7 (Toeplitz matrix)**.** *A matrix $H \in \mathbb{R}^{k\times n}$ is a Toeplitz matrix, if there exists $n+k-1$ numbers c_l ($l \in [-(n-1):(k-1)] \subset \mathbb{Z}$) such that*

$$H_{ij} = c_{i-j} \tag{2.13}$$

Remark 2.7.

1. *in plain English, all NE-SE diagonals are constant*
2. *if $m \ll n$: additional sparseness (band matrix of width m)*
3. *H_n^h has only m degrees of freedom*
4. *locality (sparseness $m \ll n$) and weight sharing (kernel)*

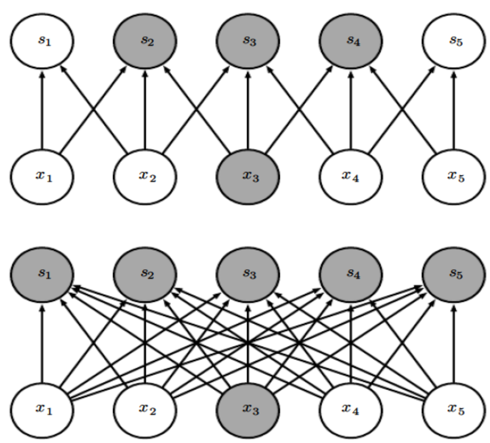


Figure 4: Sparse vs dense connectivity

Convolutions in higher dimensions: generalize concept of convolution to:

1. 2D: e.g. images, spectograms
2. 3D: e.g. color or multi-spectral images, voxel images, video
3. or even higher dimensions

Replace vector by:

1. matrices or fields (e.g. in discrete case)

$$(F * G)[i,j] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} F[i-k, j-l] \cdot G[k,l] \tag{2.14}$$

2. tensors: for 3D and higher

Different options for border handling:

1. our definition: padding with zeros = same padding
2. only retain values from windows fully contained in support of signal f = valid padding

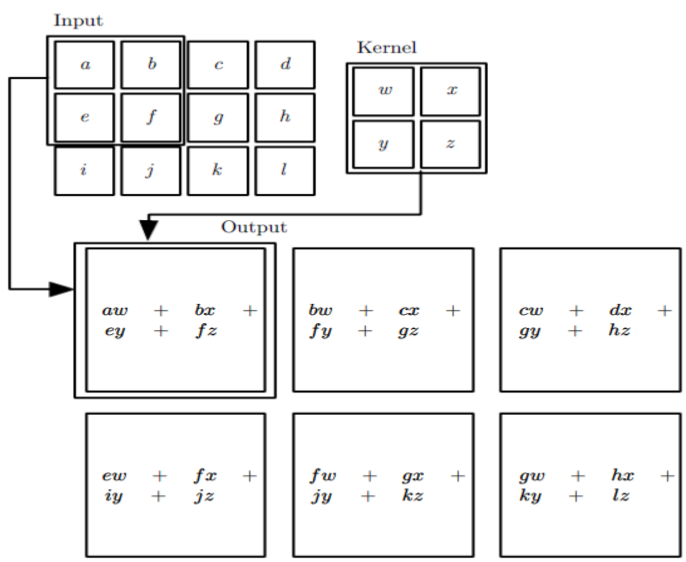


Figure 5

layout:

1. Convolved signal inherits topology of original signal
2. Hence: units in a convolutional layer are typically arranged on the same grid (1D, 2D, 3D,...)

Exploit structural sparseness in computing $\frac{\partial x_i^l}{\partial x_j^{l-1}}$:

1. receptive filed of $x_i^l : \mathcal{I}_i^l \triangleq \left\{ j : W_{ij}^l \neq 0 \right\}$, where W^l is the Toeplitz matrix of the convolution
2. obviously $\frac{\partial x_i^l}{\partial x_j^{l-1}} = 0$ for $j \notin \mathcal{I}_i^l$

Weight sharing in computing $\frac{\partial \mathcal{R}}{\partial h_j^l}$, where h_j^l is a kernel weight

$$\frac{\partial \mathcal{R}}{\partial h_j^l} = \sum_i \frac{\partial \mathcal{R}}{\partial x_i^l} \frac{\partial x_i^l}{\partial h_j^l} \tag{2.15}$$

Weight is re-used for every unit within target layer $\Rightarrow$ additive combination of derivatives in chain rule. nesting of convolutions: receptive fields grow.

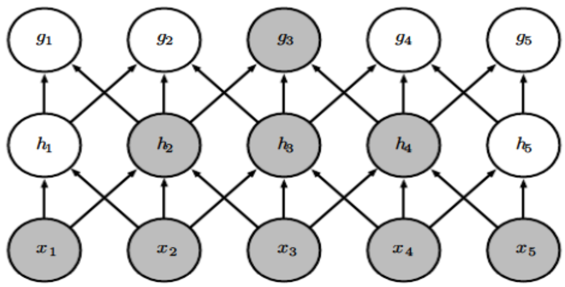


Figure 6

FFT (Fast Fourier Transform): compute convolutions fast(er).

1. Fourier transform of signal $f \to (\mathcal{F}f)$ and kernel $h \to (\mathcal{F}h)$
2. pointwise multiplication and inverse Fourier transform:
$$(f * h) = \mathcal{F}^{-1}\left((\mathcal{F}f) \cdot (\mathcal{F}h)\right) \tag{2.16}$$
3. FFT: signal of length n, can be done in $O\left(n \ \log n\right)$
4. pays off, if many channels (amortizes computation of $\mathcal{F}f$)
5. small kernels ($m < \log n$): favor time / space domain

Remark 2.8. *content...*

Stages:

1. Non-linearites: detector stage. As always: scalar non-linearities (activation function)
2. Pooling stage: locally combine activities

Most frequently used pooling function:max pooling.

Definition 2.8 (Max Pooling)**.** *Define window size r (e.g. 3 or 3×3), then*

$$\begin{aligned} 1D: \quad & x_i^{\max} = \max\left\{x_{i+k} : 0 \leqslant k < r\right\}, \\ 2D: \quad & x_{ij}^{\max} = \max\left\{x_{i+k,j+l} : 0 \leqslant k, l < r\right\} \end{aligned} \tag{2.17}$$

Remark 2.9.

1. *maximum over a small patch of units*

2. *other functions are possible: average, soft-maximization*

Max-pooling: invariance

1. set of invertible transformations $\mathcal{T}$: group w.r.t composition
2. $\mathcal{T}-$invariance through maximization $f_{\mathcal{T}}(x) \triangleq \max_{\tau \in \mathcal{T}} f(\tau x)$

Proposition 2.1. *$f_{\mathcal{T}}$ is invariant under $\tau \in \mathcal{T}$.*

Proof.

$$f_{\mathcal{T}}(\tau x) = \max_{\rho \in \mathcal{T}} f(\rho(\tau x)) = \max_{\rho \in \mathcal{T}} (f(\rho \circ \tau) x) = \max_{\sigma \in \mathcal{T}} f(\sigma x) \tag{2.18}$$

as $\forall \sigma, \sigma = \rho \circ \tau$ with $\rho = \sigma \circ \tau^{-1}$. □

sub-sampling(also known as (aka) strides):

1. often, it is desirable to reduce the size of feature maps
2. sub-sampling: reduce temporal/spatial resolution. Often: combined with (max-)pooling (aka. stride)
3. example: max-pool, filter 2×2, stride 2×2
4. disadvantage: loss of information

Learn multiple convolution kernel (or filters) = multiple channels:

1. typically: all channels use same window size
2. channels form additional dimension for next layer (e.g. 2D signal $\times$ channels = 3D tensor)
3. number of channels: design parameter

http://cs231n.github.io/assets/conv-demo/index.html

Note that kernels (across channels) form a linear map:

$$h : \mathbb{R}^{r^2 \times d} \to \mathbb{R}^k \tag{2.19}$$

where $r \times r$ is the window size and d is the depth.

Convolutional networks: multiple, stacked feature maps

$$\underbrace{y[r]}_{r-th\ channel}[s,t] = \sum_u \sum_{\Delta s, \Delta t} \underbrace{w[r,u][\Delta s, \Delta t]}_{parameters} \underbrace{x[u]}_{u-th\ channel}[s + \Delta s, t + \Delta t] \tag{2.20}$$

1. x, y tensor, 3-rd order

2. number of parameters:

$$\underbrace{\#r \cdot \#u}_{\textit{fully connected}} \quad \cdot \quad \underbrace{\#\Delta s \cdot \#\Delta t}_{\textit{window size}} \tag{2.21}$$

3. pointwise non-linearities (e.g. ReLU)
4. interleaved with: pooling (e.g. max, average)
5. optionally: downsampling (use of strides)

Convolutional pyramid:
Typical use of convolution in vision: sequence of convolutions that

1. reduce spatial dimensions (sub-sampling)
2. increase number of channels

⇒ smaller, but more feature maps.

Architecture LeNet5 [?] [?]

1. C1/S2: 6 channels, 5×5 kernels, 2×2 sub (4704 units)
2. C3/S4: 16 channels,6×6 kernels, 2×2 sub (1600 units)
3. C5: 120 channels, F6: fully-connected
4. output: Gaussian noise model (squared loss)

AlexNet: [?]

1. Pyramidal architecture: reduce spatial resolution, increase channels with depth
2. Challenge: many channels (width) + large windows + depth
3. Number of parameters
   (a) 384 to 384 channels with 3×3 windows: > 1.3 M
   (b) $13 \times 13 \times 384$ tensor to 4096, fully connected: > 265 M

Deep ConvNets: key challenges

1. avoid blow-up of model size (e.g. # parameters)
2. preserve computational efficiency of learning (e.g. gradients)
3. allow for large depth (as it is known to be a plus)
4. allow for sufficient width (as it is known to be a plus, too)

Very deep convolutional networks: VGG

1. use very small receptive fields (maximally 3×3)
2. avoid downsampling/pooling
3. stacking small receptive fields: more depth, fewer parameters
4. example: $3 \cdot (3 \times 3) = 27 < 49(7 \times 7)$

Many channels needed for high accuracy, typically $k \sim 200 - 1000$ (e.g. AlexNet: 2×192).

Observation (motivated by Arora et al, 2013 [?]): when convolving, dimension reduction across channels may be acceptable.

Dimension reduction: m channels of a $1 \times 1 \times k$ convolution $m \leq k$:

$$x_{ij}^{+} = \sigma\left(W x_{ij}\right), \quad W \in \mathbb{R}^{m \times k} \tag{2.22}$$

1. 1×1 convolution = no convolution
2. inception module (Szegedy et al. [?])
3. network within a network (Lin et al, [?])
4. i.e. W is shared for all (i, j) (translation invariance)