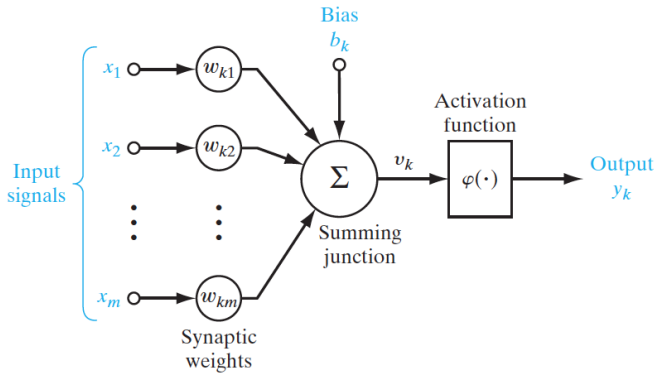
Neural networks and Deep Learning

MATH-412 - Statistical Machine Learning

Formal Neuron Model (McCullock & Pitts, 1943)



$$y = \varphi(\boldsymbol{w}^\top \mathbf{x} + b)$$

where
φ is the activation function
(often denoted σ)

Examples

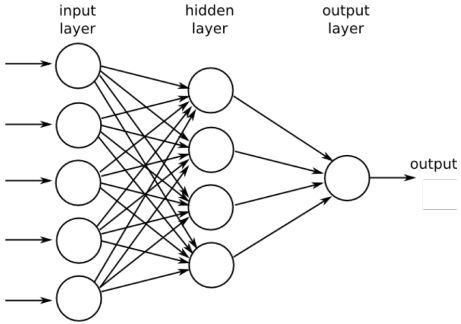
- $\varphi(z) = \text{sign}(z)$ $\rightarrow$ McCullock-Pitts Neuron model (used in the perceptron)
- $\varphi(z) = \sigma^*(z) = (1 + e^{-z})^{-1}$ $\rightarrow$ logistic regression

Two layer neural network

i.e. Neural network with a single hidden layer.

If the activation function of the second layer is linear:

$$\begin{aligned}\hat{y} &= \sum_{j=1}^{k} \alpha_j \, \sigma(\boldsymbol{w}_j^\intercal \mathbf{x} + b_j) \\ &= \boldsymbol{\alpha}^\intercal \sigma(\boldsymbol{W}\mathbf{x} + \mathbf{b})\end{aligned}$$



where k is called **width**

- $\sigma(\boldsymbol{z}) = (\sigma(z_1), \ldots, \sigma(z_k))$ **is applying the function entrywise**
- $\boldsymbol{W} \in \mathbb{R}^{k \times m}$ **the matrix whose** j**th row is** $\boldsymbol{w}_j^\intercal \in \mathbb{R}^m$**.**
- $\mathbf{b} \in \mathbb{R}^m$

Multilayer network (aka Multi-Layer Perceptron)

$$\mathbf{x}^l = f^l(\mathbf{x}^{l-1}) = \sigma^l(\boldsymbol{W}^l\mathbf{x}^{l-1} + \mathbf{b}^l)$$

- $\mathbf{x}^l$ activations of the lth layer
- σ^l activation function at the lth layer
- $\boldsymbol{W}^l$ weights of the lth layer
- $\mathbf{x} = \mathbf{x}^0$ input and $\hat{\boldsymbol{y}}$ output

$$\hat{\boldsymbol{y}} = \sigma^L(\boldsymbol{W}^L \dots \sigma^2(\boldsymbol{W}^2\sigma^1(\boldsymbol{W}^1\mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) \dots + \mathbf{b}^L)$$

$$\hat{\boldsymbol{y}} = f^L \circ \dots \circ f^1(\mathbf{x}) \qquad \text{is a composition of functions.}$$

- L is called **depth**

Forward propagation

$\mathbf{x}^0 \leftarrow \mathbf{x}$
For $l = 1$ to L
    $\mathbf{x}^l \leftarrow f^l(\mathbf{x}^{l-1})$
Endfor

Activation Functions

Univariate activation functions:

- McCullock-Pitts Neuron model (aka perceptron): $\sigma(z) = \text{sign}(z)$
- Logistic function: $\sigma(z) = \sigma^*(z) = (1 + e^{-z})^{-1}$
- Hyperbolic tangent: $\sigma(z) = \tanh(z) = 2\,\sigma^*(2z) - 1$
- Rectified Linear Unit (ReLU): $\sigma(z) = (z)_+ := \max(z, 0)$
- Leaky ReLU: $\sigma_\alpha(z) = (z)_+ - \alpha\,(-z)_+ = \max(z, \alpha z)$ with $0 < \alpha \ll 1$.
- Softplus: $\sigma(z) = \log(1 + e^z)$

Multivariate activation functions:

- Softargmax or "softmax" function[1]: $\sigma(z_1, \ldots, z_K) = \left(\frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \right)_j$

The logistic and the "softmax" are nowadays essentially used in the last layer for classification combined with the *log loss* (often called *cross-entropy loss* in the DL community)

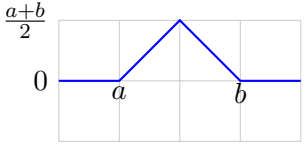
[1] Should not be confused with $\boldsymbol{z} \mapsto \frac{1}{t} \log\left(\sum_{i=1}^d e^{tz_i}\right)$

Good properties of the ReLU

Rectified Linear Unit (ReLU): $\sigma(z) = (z)_+ := \max(z, 0)$

- Multilayer NN with ReLU activations are piecewise linear functions
- In 1D, the following combination of three neurons

$$x \mapsto (x-a)_+ - \big(2x - (a+b)\big)_+ + (x-b)_+$$



is up to normalization the B-spline basis function with knots at $\{a, \frac{a+b}{2}, b\}$.

$\Rightarrow$ In 1D, a two layer neural network therefore has the same expressive power as linear splines *with learnable knots positions*.

$\Rightarrow$ In 1D, any continuous function can be uniformly approximated on a compact set by a two-layer network with ReLU activation.

Approximation Theorem in 1D

$$\mathcal{G}_\sigma^1 := \{g : g(t) = \sigma(\lambda t + \theta) : \lambda, \theta \in \mathbb{R}\}$$

$$\mathcal{H}_\sigma^1 = \text{span}(\mathcal{G}_\sigma^1)$$

Theorem: Leshno et al. (1993)

Let $\sigma \in C(\mathbb{R})$, not a polynomial, then $\mathcal{H}_\sigma^1$ is dense in $(C(K), \|\cdot\|_\infty)$, for any compact set $K \subset \mathbb{R}$.

Corollary: Fully connected NNs with one hidden layer and any non-polynomial, continuous activation function are universal function approximators (depth 2, width $k \to \infty$).

Lemma: Fully connected NNs with one hidden layer and a polynomial activation function are **not** universal function approximators.

Proof: Easier to see for $\sigma \in C^\infty(\mathbb{R})$, then use convolutions.

General Approximation

$$\mathcal{G}_\sigma^d := \{g : g(\mathbf{x}) = \sigma(\boldsymbol{w}^\top \mathbf{x} + b) \text{ for some } \boldsymbol{w} \in \mathbb{R}^d,\, b \in \mathbb{R}\}$$

$$\mathcal{H}_\sigma^d = \text{span}(\mathcal{G}_\sigma^d)$$

Lemma: Pinkus (1999) (simplified)

The fact that $\mathcal{H}_\sigma^1$ is dense in $(C(K_1), \|\cdot\|_\infty)$, for any compact set $K_1 \subset \mathbb{R}$ with

$$\mathcal{H}_\sigma^1 := \text{span}(\mathcal{G}_\sigma^1) = \text{span}\{\sigma(\lambda t + \theta) : \lambda, \theta \in \mathbb{R}\}$$

implies that $\mathcal{H}_\sigma^d$, with

$$\mathcal{H}_\sigma^d := \text{span}(\mathcal{G}_\sigma^d) = \text{span}\{\sigma(\boldsymbol{w}^\top \mathbf{x} + b) : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}\,,$$

is dense in $(C(K), \|\cdot\|_\infty)$ for $K \subset \mathbb{R}^d$, and this for any $d \geq 1$.

Training a feedforward neural network

Given a *feedforward neural network (with no offsets $\mathbf{b}^l$ for simplicity)*

$$F_{\mathcal{W}}(\mathbf{x}) = \sigma^L(\boldsymbol{W}^L \dots \sigma^2(\boldsymbol{W}^2\sigma^1(\boldsymbol{W}^1\mathbf{x}))$$

parameterized by $\mathcal{W} = (\boldsymbol{W}^1, \dots, \boldsymbol{W}^L)$ and a loss function ℓ, we would like to minimize the risk

$$\mathcal{R}(F_{\mathcal{W}}) = \mathbb{E}[\ell(F_{\mathcal{W}}(X), Y)].$$

We can use stochastic gradient descent

$$\boldsymbol{W}^{l,t+1} = \boldsymbol{W}^{l,t} - \eta_t \nabla_{\boldsymbol{W}^l} \ell(F_{\mathcal{W}^t}(\mathbf{x}^t), y^t),$$

where $(\mathbf{x}^t, y^t)$ is the input-output pairs drawn at time t.
This requires to compute

$$\nabla_{\boldsymbol{W}^l} \Big(\ell(\cdot, y^t) \circ f^L \circ \dots \circ f^1 \Big)(\mathbf{x}^t)$$

which will require to use the chain-rule for differentiation.

Chain rules

- If f and g are scalar functions

$$(f \circ g)'(x) = f'(g(x))\, g'(x)$$

- More generally with differentials:

$$d(f \circ g)(x, \cdot) = df(g(x), \cdot) \circ dg(x, \cdot)$$

- For $f : \mathbb{R}^m \to \mathbb{R}^d$ and $g : \mathbb{R}^p \to \mathbb{R}^m$

$$\boldsymbol{J}_{f \circ g}(\mathbf{x}) = \boldsymbol{J}_f\big(g(\mathbf{x})\big)\, \boldsymbol{J}_g(\mathbf{x})$$

where $\boldsymbol{J}_g(\mathbf{x}) \in \mathbb{R}^{m \times p}$ is the Jacobian of g at x defined by

$$\boldsymbol{J}_g(\mathbf{x}) = \begin{pmatrix} \frac{\partial g_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\mathbf{x})}{\partial x_p} \\ \vdots & & \vdots \\ \frac{\partial g_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_m(\mathbf{x})}{\partial x_p} \end{pmatrix} \quad \text{with} \quad g(\mathbf{x}) = \begin{pmatrix} g_1(\mathbf{x}) \\ \vdots \\ g_m(\mathbf{x}). \end{pmatrix}$$

Forward chain rules for compositions

Similarly,

- if we consider $F^l = f^l \circ \ldots \circ f^1 \quad$ with $\quad \mathbf{x}^l = F^l(\mathbf{x})$

then since $F^L = f^L \circ F^{L-1}$ we have

$$\boldsymbol{J}_{F^L}(\mathbf{x}) = \boldsymbol{J}_{f^L}(\mathbf{x}^{L-1})\, \boldsymbol{J}_{F^{L-1}}(\mathbf{x})$$

and

$$\boldsymbol{J}_{F^L}(\mathbf{x}) = \boldsymbol{J}_{f^L}(\mathbf{x}^{L-1})\, \boldsymbol{J}_{f^{L-1}}(\mathbf{x}^{L-2}) \ldots \boldsymbol{J}_{f^1}(\mathbf{x})$$

$$i.e. \quad \text{“} \frac{\partial \mathbf{x}^L}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^L}{\partial \mathbf{x}^{L-1}} \frac{\partial \mathbf{x}^{L-1}}{\partial \mathbf{x}^{L-2}} \cdots \frac{\partial \mathbf{x}^1}{\partial \mathbf{x}^0} \text{”}$$

- This computes $\dfrac{\partial \mathbf{x}^l}{\partial \mathbf{x}}$ for all l.
- What if we need $\dfrac{\partial \mathbf{x}^L}{\partial \mathbf{x}^l}$ for all l ?

Abstract forward propagation of gradients

$\boldsymbol{J}^0 \leftarrow \boldsymbol{I}_p$
For $l = 1$ to L
  $\boldsymbol{J}^l \leftarrow \boldsymbol{J}_{f^l}(\mathbf{x}^{l-1})\, \boldsymbol{J}^{l-1}$
Endfor

Backward chain rules for compositions

Similarly,

- if we consider $\bar{F}^l = f^L \circ \ldots \circ f^{l+1}$ with $\mathbf{x}^l = F^l(\mathbf{x})$

then since $\bar{F}^{l-1} = \bar{F}^l \circ f^l$ we have

$$\bar{\boldsymbol{J}}^{l-1} := \boldsymbol{J}_{\bar{F}^{l-1}}(\mathbf{x}^{l-1}) = \boldsymbol{J}_{\bar{F}^l}(\mathbf{x}^l)\, \boldsymbol{J}_{f^l}(\mathbf{x}^{l-1})$$

$$i.e. \quad \frac{\partial \mathbf{x}^L}{\partial \mathbf{x}^{l-1}} = \frac{\partial \mathbf{x}^L}{\partial \mathbf{x}^l} \frac{\partial \mathbf{x}^l}{\partial \mathbf{x}^{l-1}}$$

Abstract backward propagation of gradients

$\bar{\boldsymbol{J}}^L \leftarrow \boldsymbol{I}_d$
For $l = L$ to 1
  $\bar{\boldsymbol{J}}^{l-1} \leftarrow \bar{\boldsymbol{J}}^l\, \boldsymbol{J}_{f^l}(\mathbf{x}^{l-1})$
Endfor

Note that this requires to have performed a basic forward pass to have computed $\mathbf{x}^l$ for all l.

Computing gradients with respect to the parameters

$$\nabla_{\boldsymbol{W}^l} \ell(F_{\mathcal{W}}(\mathbf{x}), y)$$

Assuming that $x^L := F_{\mathcal{W}}(\mathbf{x})$ is scalar, we have:

$$\begin{aligned}
\nabla^{\intercal}_{\boldsymbol{w}^l_j} \ell(F_{\mathcal{W}}(\mathbf{x}), y) &= \nabla^{\intercal}_{\boldsymbol{w}^l_j} \Big(\ell(\cdot, y) \circ f^L \circ \ldots \circ f^{l+1}\Big)(\underbrace{\sigma^l(\boldsymbol{W}^l \mathbf{x}^{l-1})}_{\mathbf{x}^l}) \\
&= \frac{\partial \ell}{\partial x^L} \frac{\partial x^L}{\partial \mathbf{x}^l} \frac{\partial \mathbf{x}^l}{\partial \boldsymbol{w}^l_j} \\
&= \ell'(x^L, y)\, \boldsymbol{J}_{\bar{F}^l}(\mathbf{x}^l) \frac{\partial \mathbf{x}^l}{\partial \boldsymbol{w}^l_j} \\
&= \ell'(x^L, y)\, [\boldsymbol{J}_{\bar{F}^l}(\mathbf{x}^l)]_j \frac{\partial x^l_j}{\partial \boldsymbol{w}^l_j}
\end{aligned}$$

And applying the chain rule again: $\dfrac{\partial x^l_i}{\partial \boldsymbol{w}^l_j} = (\sigma^l)'(\boldsymbol{w}^{l\intercal}_j \mathbf{x}^{l-1}) \cdot (\mathbf{x}^{l-1})^{\intercal}$

Weight decay

Weight decay=... the DL name for Tikhonov regularization but used in the context of SGD

$$\begin{aligned} \boldsymbol{W}^{l,t+1} &= \boldsymbol{W}^{l,t} - \eta_t \nabla_{\boldsymbol{W}^l} \ell(F_{\mathcal{W}}(\mathbf{x}^t), y^t) - \eta_t \frac{\lambda}{2} \nabla_{\boldsymbol{W}^l} \|\boldsymbol{W}^l\|_F^2 \\ &= \underbrace{(1-\rho_t)\boldsymbol{W}^{l,t}}_{\text{weight decay}} - \eta_t \nabla_{\boldsymbol{W}^l} \ell(F_{\mathcal{W}}(\mathbf{x}^t), y^t) \quad \text{with} \quad \rho_t = \eta_t \lambda. \end{aligned}$$

Dropout

Idea: randomly "drop" subsets of units in the network (Hinton et al., 2012).

More precisely, define "keep" probability π_i^l for unit i in layer l.

- typically: $\pi_i^0 = 0.8$ (inputs) and $\pi_i^{l \geq 1} = 0.5$ (hidden units)
- realization: sampling bit mask and zeroing out activations
- effectively defines an exponential ensemble of networks (each of which is a sub-network of the original one)
- all models share same weights
- standard backpropagation applies

[see the book *Deep Learning* by Goodfellow et al. 2016, Chapter 7.12]

Dropout: Motivation

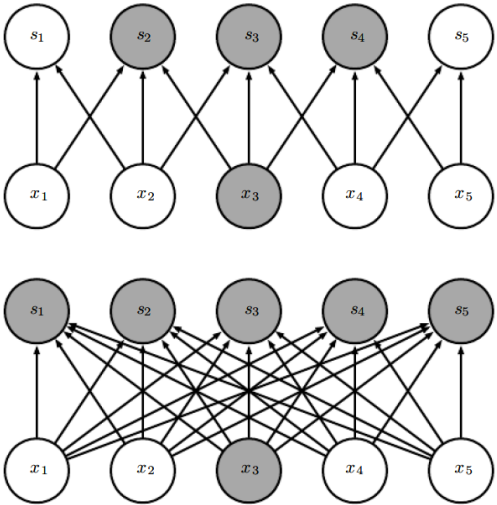
... "overfitting" is greatly reduced by randomly omitting half of the feature detectors on each training case. This prevents complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors. Instead, each neuron learns to detect a feature that is generally helpful for producing the correct answer given the combinatorially large variety of internal contexts in which it must operate.

(Hinton et al., 2012)

Other training techniques that we did not talk about

- Heuristics for initialization
- Batch renormalization
- Other algorithms than SGD: Adagrad, Adam, etc
- Network pruning

Convolution layer vs fully connected layer



(DL, Figure 9.2)

Convolutions

Convolution:

$$(f * h)(x) = (h * f)(x) = \int f(x-z)h(z)dz = \int f(y)h(x-y)dy$$

Cross-correlation:

$$(f \star h)(x) = (f * h(-.))(x) = \int f(x+z)h(z)dz = \int f(y)h(y-x)dy$$

Warning: some libraries use "convolution" for "cross-correlation"!

Equivariance of convolution/cross-correlation
If $f_\tau(x) = f(x-\tau)$, then $(f_\tau * h) = (f * h)_\tau$ i.e.,

if f is translated then $f * h$ is translated.

Thm: A linear operator is *equivariant* for the translations if and only if it is a convolution.

Convolutions III

Discrete cross-correlation in 2D with kernel with finite support

For $I \in \mathbb{R}^{w \times h}$,

$$[I \star K](i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} I(i+m, j+n) K(m,n)$$

Zero-padding

- Previous definition a priori only valid for

$$M+1 \leq i \leq w-M, \quad N+1 \leq i \leq h-N$$

$\rightarrow$ So the convolution yields an image of size $(w-2M) \times (h-2N)$

$\rightarrow$ Zero-padding consist in enlarging the image by adding a number M' of zero columns on the sides of I and a number N' of zero rows above and below I, with $M' \leq M$ and $N' \leq N$ so as to reduce the shrinkage of the image.

Convolutions III

Discrete cross-correlation in 2D with kernel with finite support

For $I \in \mathbb{R}^{w \times h}$,

$$[I \star K](i,j) = \sum_{m=-M}^{M} \sum_{n=-N}^{N} I(i+m, j+n) K(m,n)$$

Nb of param.: $(2M+1)(2N+1)$

Comp. Complexity:

$$\lesssim wh(2M+1)(2N+1)$$

In fact $= O(wh \log[(2M+1)(2N+1)])$
but not worth it for M, N small...

Compare with a fully connected layer:

$$I^{l+1}(i,j) = \sum_{m=1}^{w_l} \sum_{n=1}^{h_l} W^l(i,j,m,n) I^l(m,n)$$

Nb of param.: $w_l\, h_l\, w_{l+1}\, h_{l+1}$

Comp. complexity: $w_l\, h_l\, w_{l+1}\, h_{l+1}$

Pooling

Idea: build in invariance to small local distortions of the image, by keeping only most significant activation over a window.

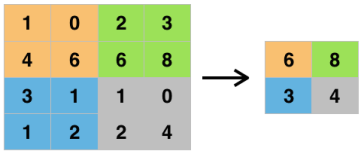
- *strides* s, s' : horizontal and vertical subsampling factors
- M, N horizontal and vertical pooling ranges.

Max pooling

$$I^{l+1}(i,j) = \max_{|m|\leq M, |n|\leq N} I^l(s\,i+m, s'j+n)$$

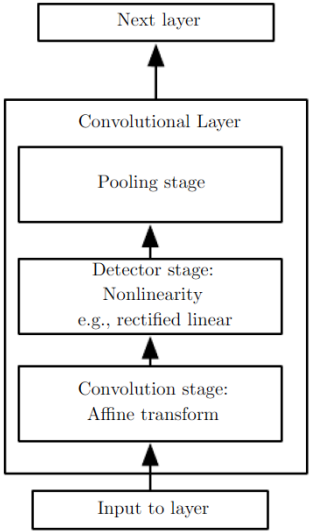
Sum pooling

$$I^{l+1}(i,j) = \sum_{|m|\leq M, |n|\leq N} I^l(s\,i+m, s'j+n)$$



Max-pooling
filter 2x2, stride 2x2

Convolutional Layers: Stages



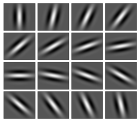
Pooling stage:

locally combine activities

Non-linearity= activation function

Multichannel layers and formulation via tensors

In practice, when working e.g. on images $\sigma^1(I \star K)$ can be viewed as one non-linear filter computed on the image. Filters that can be relevant are *Gabor filters* that are appropriate to detect edges in the image. But each filter correspond to an edge at a certain frequency and with a given orientation, so it makes sense to apply several such filters. So instead of applying a single learnable filter we apply several of them so that I^2 has actually multiple channels.



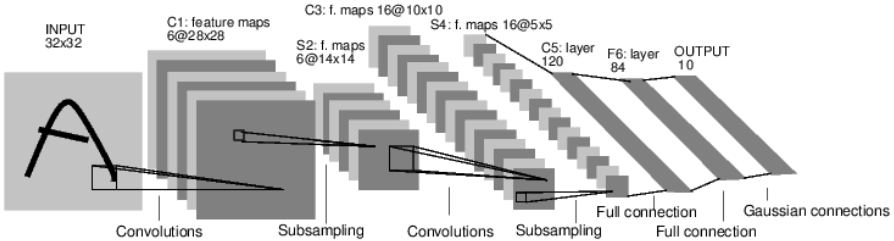
Gabor filters

$$I^2 \in \mathbb{R}^{C \times w \times h}$$

It is natural to obtain the values of a channel c of a new layer by computing linear combination of the convolutions computed on the channels of the previous layer. This leads to tensor convolutions of the form:

$$I^{l+1}(c, i, j) = \sum_{c'=1}^{C^l} \sum_{m:|m|\leq M} \sum_{n:|n|\leq N} I^l(c', i+m, j+n) K^l(c, c', m, n)$$

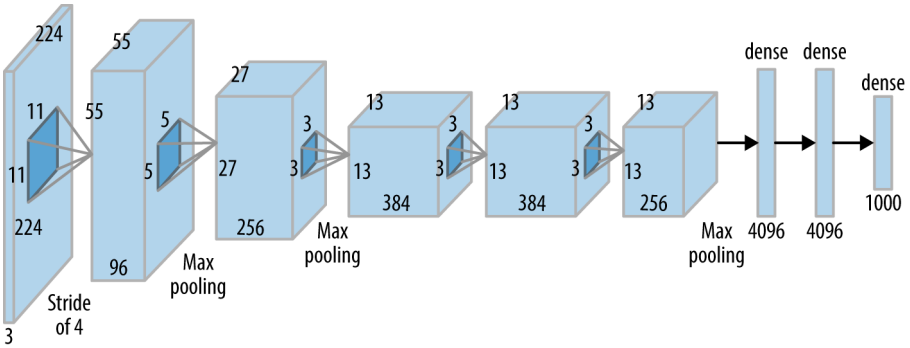
LeNet5 (LeCun et al., 1989, 1998)



Architecture LeNet5

- C1/S2: 6 channels, (5x5 kernels), 2x2 sub (4704 units)
- C3/S4: 16 channels, (6: 6x6x3, 9: 6x6x4 and 1: 6x6x6 kernels), 2x2 sub (1600 units)
- C5: 120 channels, F6: fully-connected
- $\sigma = \tanh$
- output: Gaussian noise model (square loss)

AlexNet (Krizhevsky et al., 2012)



- 60 million parameters and 500,000 neurons
- 5 convolutional layers, some followed by max-pooling
- 2 globally connected layers with a final 1000-way softmax

Convolution: summary

- Equivariance with respect to translations
  - $\rightarrow$ well adapted to images in 2D, times series in 1D
- Decrease of the number of parameters provides
  - $\rightarrow$ computational+memory advantage
  - $\rightarrow$ allows for better generalization
- Mainly using 3×3 and 5×5 convolutions in deeper architectures.

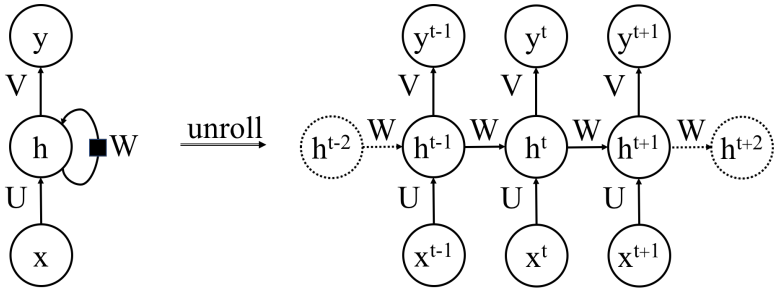
Recurrent Neural Networks (RNN)

Goal: model the relationship between a sequence of input variables $\mathbf{x}^t$ and a sequence of output variables $\boldsymbol{y}^t$

Idea: Introduce hidden variables $\boldsymbol{h}^t$ to implement "short term memory".

Linear dynamical system $\quad \boldsymbol{h}^t = f(\boldsymbol{h}^{t-1}, \mathbf{x}^t) := \sigma(\mathbf{W}\boldsymbol{h}^{t-1} + \mathbf{U}\mathbf{x}^t + \mathbf{b})$

Optionally produce outputs via $\quad \hat{\boldsymbol{y}}^t = \psi(\boldsymbol{h}^t) := \sigma_o(\mathbf{V}\boldsymbol{h}^t + \mathbf{c})$



"Unfolded" RNN

More other models

- Autoencoders
- Long Short Term Memory Networks (LSTM)
- Variational Auto-encoders
- Models with attention mechanisms
- Transformer networks

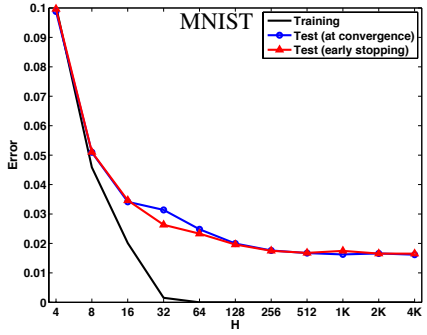
Why is deep learning working?

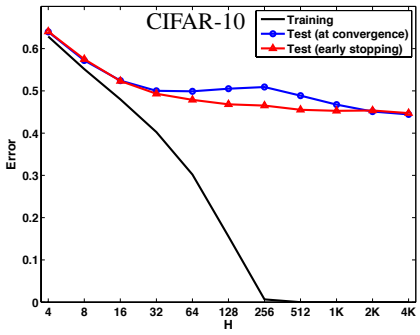
- For larger networks (not necessarily deeper), the optimization problem is empirically easier (fewer/no bad local minima)
- The vanishing/exploding gradients issues are mitigated by the use of ReLU + batch normalization.
- Huge labelled datasets have been made available thanks to the development of crawling engines and "crowd-sourcing", for which generic nonlinear models such as kernel methods did not scale so well computationally and did not generalize so well.
- Some architectural elements inspired from biological systems, e.g. for CNNs our understanding of the mammalian visual system.
- Some empirically elicited good practice
- The use of *graphical processing units* (GPU) and *tensor processing units* (TPU) which allow to perform tensorized calculus very efficiently
- The development of dedicated programming tools and languages such as Theano, TensorFlow and PyTorch.

Deep networks overfit and generalize at the same time

DL model training shows many examples of situations in which

- no regularization was used
- the network fits perfectly (overfits) the training data
- but the network generalizes well to the testing data !





Neyshabur et al. (2017)

Deep learning generalization not explained by classical SLT!

This is puzzling because

- Classical statistical learning theory usually proves that generalization occurs as a consequence of $\mathcal{R}(\hat{h}) - \widehat{\mathcal{R}}_n(\hat{h})$ being small: so the theory does not explain how overfitting and generalizing at the same time is possible.
- And since deep NN have a number of parameters which is of the order of magnitude of the number of data points or (much) larger, the classical learning theory does not apply (huge Rademacher complexity)...

However DL models are not alone:

- Random forests (with no pruning) overfit the training points but can generalize well...
- Ridgeless kernel regression can generalize well in high-dimension in spite of overfitting the data (Liang and Rakhlin, 2018)

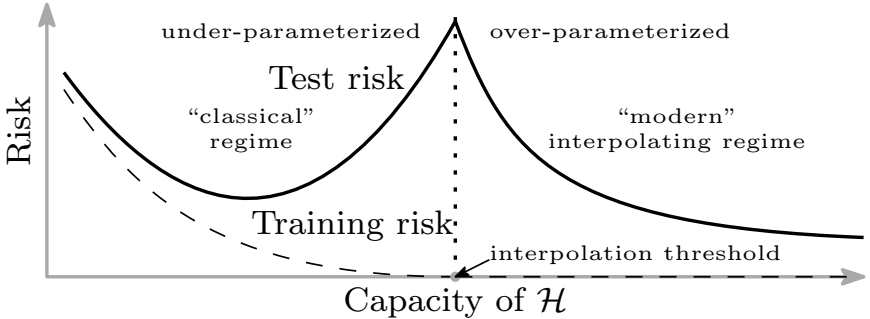
Explanation for why deep learning generalizes well

We don't know for sure, and clearly there are cases where NN do overfit and/or benefit from regularization. In models that are overparameterized, there are many models that can fit exactly the data, and which are somehow interpolating the data but the training algorithm (SGD) converges to a model of low complexity/high smoothness.

Why could this be true?

- Gradient descent algorithm can be shown to maximize the margin for some losses for binary classification and can be shown to converge to some minimal norm solutions in more general settings (for norms that are not necessarily the ℓ_2 norm) (Soudry et al., 2018).
- Some training algorithm (e.g. SGD) have the property that the mutual information between the model and the data remains small and only extract the information relevant to solve the supervised learning problem. (Achille and Soatto, 2018)

Double descent phenomenon



from Belkin et al. (2019a,b)

as model complexity increases, the test error follows the traditional "U-Shaped curve" but beyond the point of interpolation, the error starts to decrease

Conclusions on deep learning

- Learning composition of functions
- Very good empirical performance in spite of limited understanding on large supervised learning problem.
- Good performance in particular for data whose structure can be captured by specific architectures (natural images and imaging data, speech recognition and language processing, structured time series)
- DL is challenging our understanding of generalization mechanisms.
- Learning efficiently with smaller amounts of data is still difficult
- How to do semi-supervised learning efficiently remains an open problem.

References I

Achille, A. and Soatto, S. (2018). Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019a). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

Belkin, M., Hsu, D., and Xu, J. (2019b). Two models of double descent for weak features. *arXiv preprint arXiv:1903.07571*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46.

Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

References II

Liang, T. and Rakhlin, A. (2018). Just interpolate: Kernel" ridgeless" regression can generalize. *arXiv preprint arXiv:1808.00387*.

Neyshabur, B., Tomioka, R., Salakhutdinov, R., and Srebro, N. (2017). Geometry of optimization and implicit regularization in deep learning. *arXiv preprint arXiv:1705.03071*.

Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta numerica*, 8(1):143–195.

Soudry, D., Hoffer, E., Nacson, M. S., Gunasekar, S., and Srebro, N. (2018). The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878.