Introduction to Neural Networks

E. Scornet

Fall 2018

Outline

Supervised Learning

Supervised Learning Framework

- Input measurement $\mathbf{X} \in \mathcal{X}$
- Output measurement $Y \in \mathcal{Y}$.
- $(\mathbf{X}, Y) \sim \mathbb{P}$ with $\mathbb{P}$ unknown.
- Training data : $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)

- Often
  - $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \{-1, 1\}$ (classification)
  - or $\mathbf{X} \in \mathbb{R}^d$ and $Y \in \mathbb{R}$ (regression).
- A predictor is a function in $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y} \text{ meas.}\}$

Goal

- Construct a good predictor $\widehat{f}$ from the training data.

- Need to specify the meaning of good.
- Classification and regression are almost the same problem!

Loss and Probabilistic Framework

Loss function for a generic predictor

- Loss function : $\ell(Y, f(\mathbf{X}))$ measures the goodness of the prediction of Y by $f(\mathbf{X})$
- Examples:
  - Prediction loss: $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$
  - Quadratic loss: $\ell(Y, f(\mathbf{X})) = |Y - f(\mathbf{X})|^2$

Risk function

- Risk measured as the average loss for a new couple:

$$\mathcal{R}(f) = \mathbb{E}_{(X,Y)\sim\mathbb{P}}\left[\ell(Y, f(\mathbf{X}))\right]$$

- Examples:
  - Prediction loss: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{P}\left\{Y \neq f(\mathbf{X})\right\}$
  - Quadratic loss: $\mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{E}\left[|Y - f(\mathbf{X})|^2\right]$

- Beware: As $\widehat{f}$ depends on $\mathcal{D}_n$, $\mathcal{R}(\widehat{f})$ is a random variable!

Supervised Learning

Experience, Task and Performance measure

- Training data : $\mathcal{D} = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ (i.i.d. $\sim \mathbb{P}$)
- Predictor: $f : \mathcal{X} \to \mathcal{Y}$ measurable
- Cost/Loss function : $\ell(Y, f(\mathbf{X}))$ measure how well $f(\mathbf{X})$ "predicts" Y
- Risk:

$$\mathcal{R}(f) = \mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \mathbb{E}_X\left[\mathbb{E}_{Y|\mathbf{X}}\left[\ell(Y, f(\mathbf{X}))\right]\right]$$

- Often $\ell(Y, f(\mathbf{X})) = |f(\mathbf{X}) - Y|^2$ or $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$

Goal

- Learn a rule to construct a predictor $\widehat{f} \in \mathcal{F}$ from the training data $\mathcal{D}_n$ s.t. the risk $\mathcal{R}(\widehat{f})$ is small on average or with high probability with respect to $\mathcal{D}_n$.

Best Solution

- The best solution f^* (which is independent of $\mathcal{D}_n$) is
$$f^* = \arg\min_{f \in \mathcal{F}} R(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}\left[\ell(Y, f(\mathbf{X}))\right] = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{\mathbf{X}}\left[\mathbb{E}_{Y|\mathbf{X}}\left[\ell(Y, f(\mathbf{X}))\right]\right]$$

Bayes Predictor (explicit solution)

- In binary classification with $0-1$ loss:
$$f^*(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}\{Y=+1|\mathbf{X}\} \geq \mathbb{P}\{Y=-1|\mathbf{X}\} \\ & \Leftrightarrow \mathbb{P}\{Y=+1|\mathbf{X}\} \geq 1/2 \\ -1 & \text{otherwise} \end{cases}$$
- In regression with the quadratic loss
$$f^*(\mathbf{X}) = \mathbb{E}[Y|\mathbf{X}]$$

Issue: Solution requires to know $\mathbb{E}[Y|\mathbf{X}]$ for all values of $\mathbf{X}$!

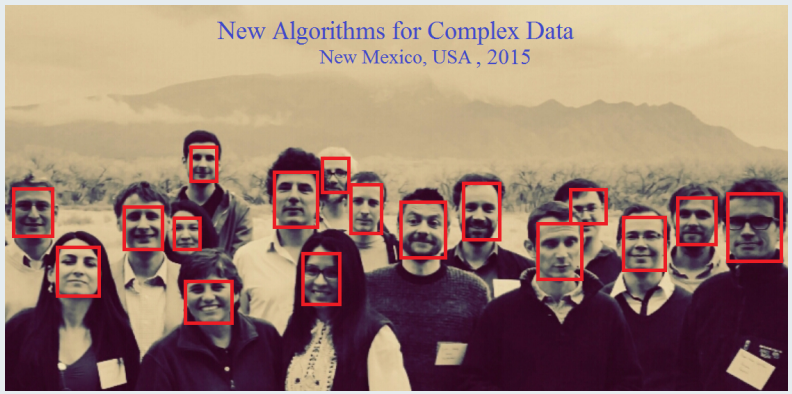
Examples

Spam detection (Text classification)

- Data: email collection
- Input: email
- Output : Spam or No Spam

Examples

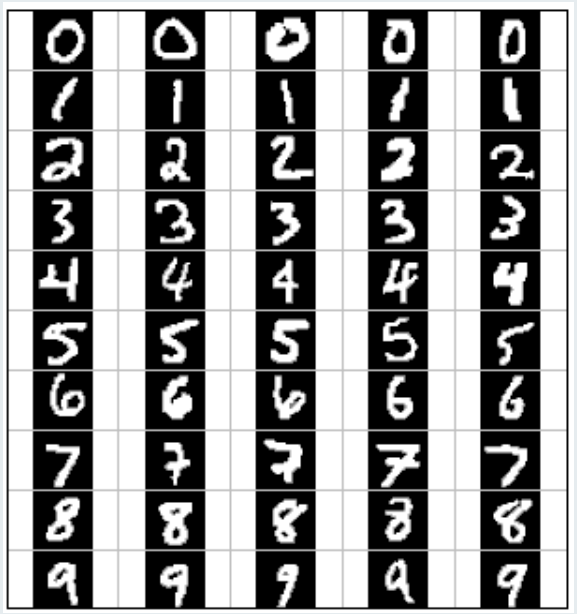
Face Detection



- Data: Annotated database of images
- Input : Sub window in the image
- Output : Presence or no of a face...

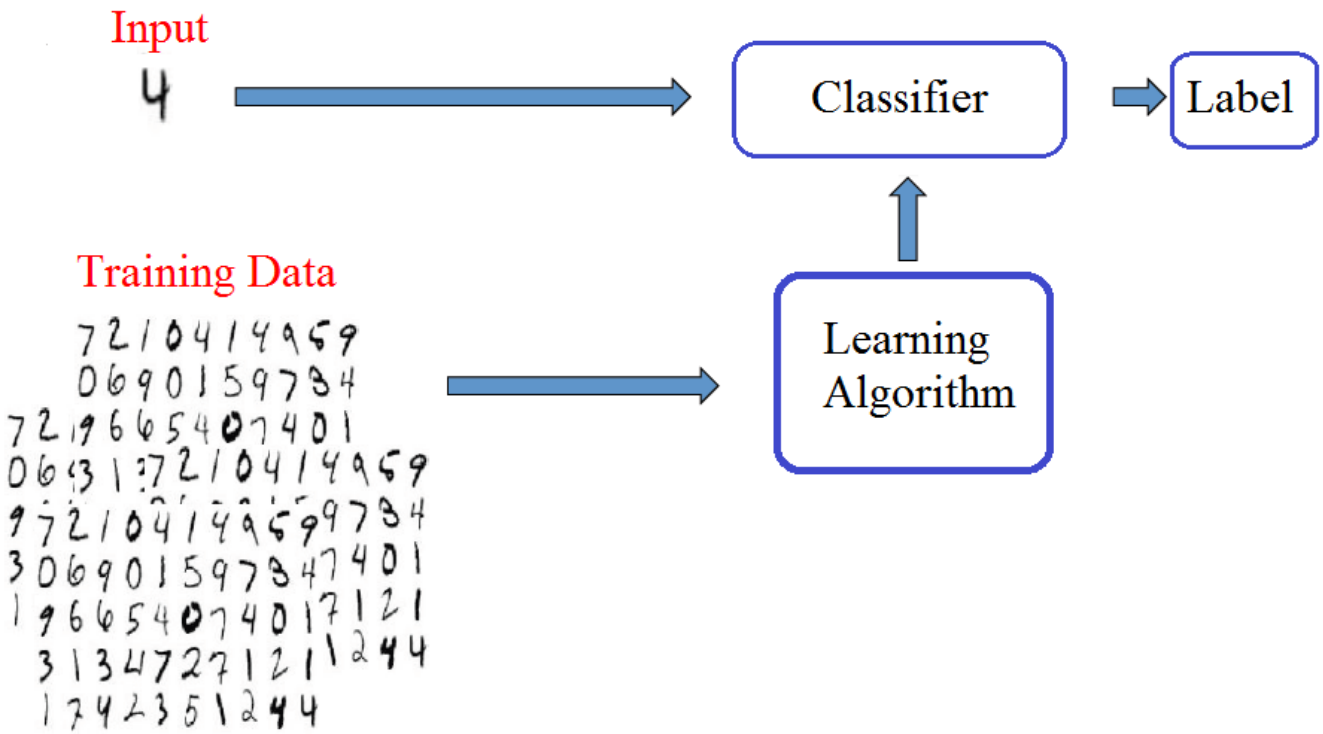
Examples

Number Recognition



- Data: Annotated database of images (each image is represented by a vector of $28 \times 28 = 784$ pixel intensities)
- Input: Image
- Output: Corresponding number

Machine Learning



A definition by Tom Mitchell (http://www.cs.cmu.edu/~tom/)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Unsupervised Learning

Experience, Task and Performance measure

- Training data : $\mathcal{D} = \{\mathbf{X}_1, \ldots, \mathbf{X}_n\}$ (i.i.d. $\sim \mathbf{P}$)
- Task: ???
- Performance measure: ???

- No obvious task definition!

Tasks for this lecture

- **Clustering (or unsupervised classification):** construct a grouping of the data in homogeneous classes.
- **Dimension reduction:** construct a map of the data in a low dimensional space without distorting it too much.

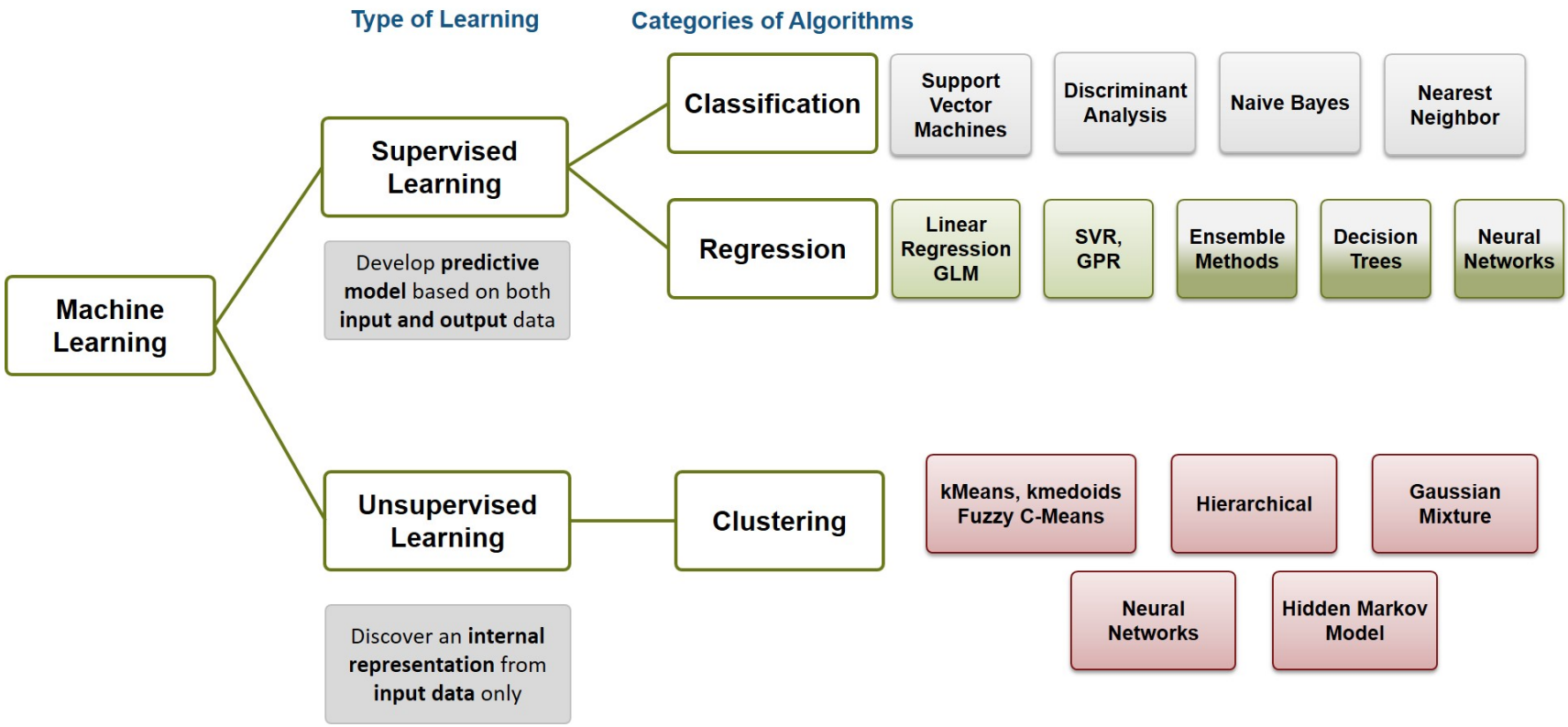
Motivation

- **Marketing:** finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records;
- **Biology:** classification of plants and animals given their features;
- **Libraries:** book ordering;
- **Insurance:** identifying groups of motor insurance policy holders with a high average claim cost; identifying frauds;
- **City-planning:** identifying groups of houses according to their house type, value and geographical location;
- **Internet:** document classification; clustering weblog data to discover groups of similar access patterns.

Marketing

- **Data:** Base of customer data containing their properties and past buying records
- **Goal:** Use the customers *similarities* to find groups.
- **Two directions:**
  - **Clustering:** propose an explicit *grouping* of the customers
  - **Visualization:** propose a representation of the customers so that the groups are *visibles*

Machine Learning



Outline

Outline

Outline

Modelling neurons

The idea of neural networks began unsurprisingly as a model of how neurons in the brain function, termed 'connectionism' and used connected circuits to simulate intelligent behaviour.

- In 1943, portrayed with a simple electrical circuit by neurophysiologist Warren McCulloch and mathematician Walter Pitts.

  ["A logical calculus of the ideas immanent in nervous activity", McCulloch and Pitts 1943]

- Donald Hebb took the idea further by proposing that neural pathways strengthen over each successive use, especially between neurons that tend to fire at the same time.

  [*The organization of behavior: A neuropsychological theory*, Hebb 2005]

First perceptron

Around 50', Frank Rosenblatt, a psychologist at Cornell, was working on understanding the comparatively simpler decision systems present in the eye of a fly, which underlie and determine its flee response.

In an attempt to understand and quantify this process, he proposed the idea of a Perceptron in 1958, calling it Mark I Perceptron. It was a system with a simple input output relationship, modeled on a McCulloch-Pitts neuron to explain the complex decision processes in a brain using a linear threshold gate.

Perceptron machine

A McCulloch-Pitts neuron takes in inputs, takes a weighted sum and returns 0 if the result is below threshold and 1 otherwise.

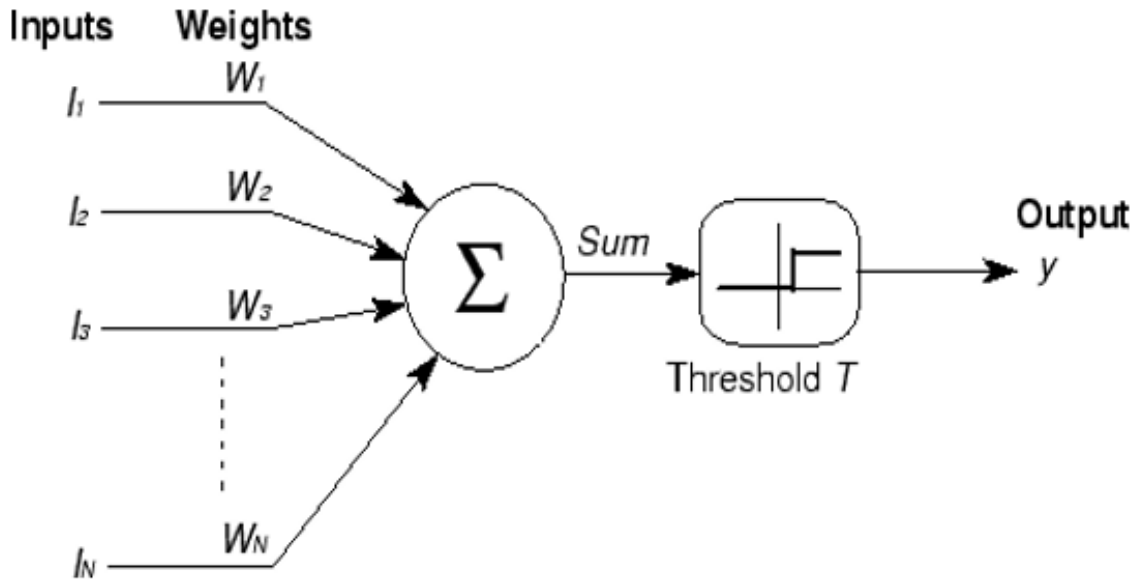


Figure: Most common representation of perceptron

Perceptron machine

A McCulloch-Pitts neuron takes in inputs, takes a weighted sum and returns 0 if the result is below threshold and 1 otherwise.

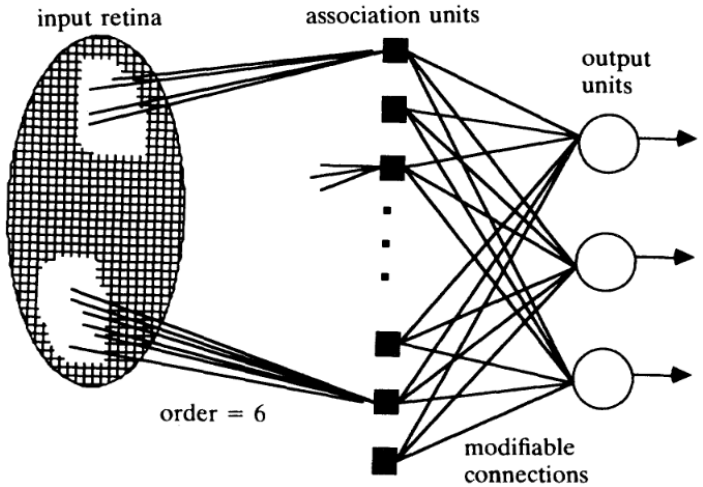
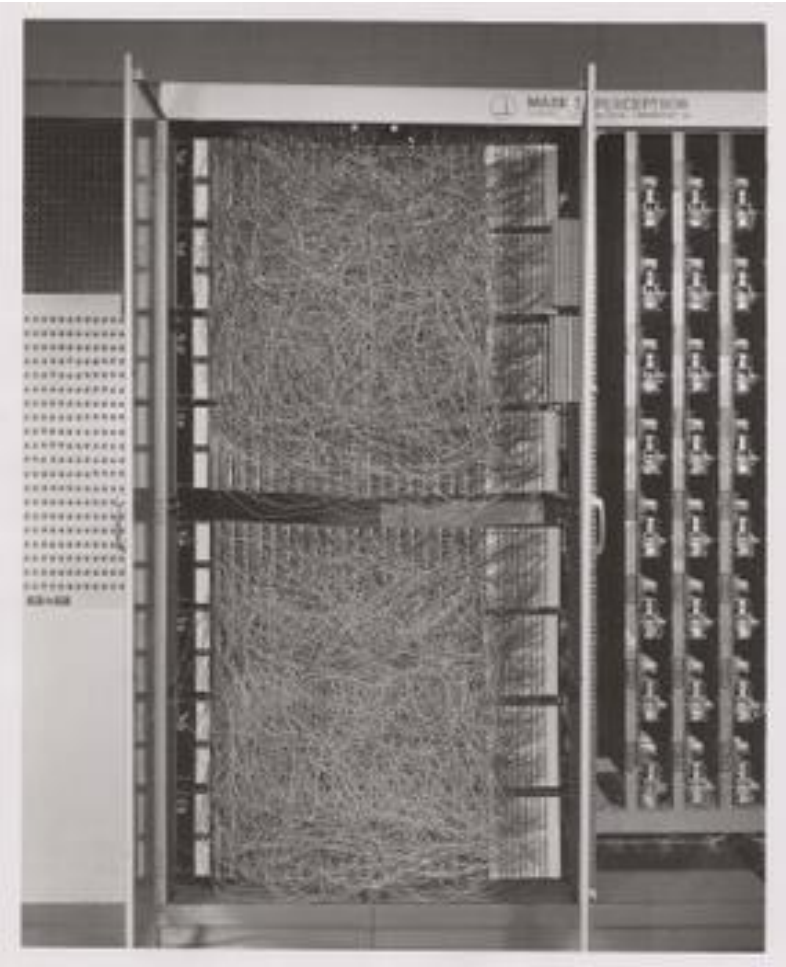


Figure: True representation of perceptron

The connections between the input and the first hidden layer cannot be optimized!

Perceptron Machine

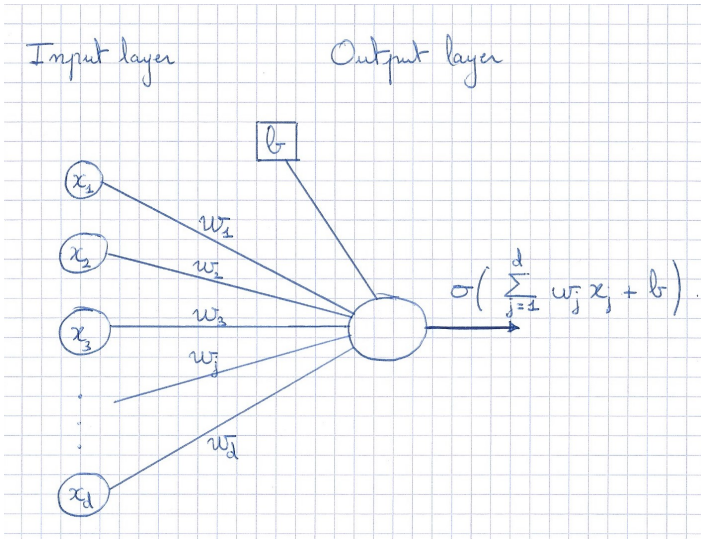


The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20x20 photocells to produce a 400-pixel image.

The main visible feature is a patchboard that allowed experimentation with different combinations of input features. On the right, you can see potentiometers that implement the adaptive weights.

The Perceptron Algorithm - a model

We want to estimate the function $Y = f(X)$. We use the following model.



- Inputs: x_j
- Weights: w_j
- Bias: b
- Activation function: $\sigma(x) = \mathbb{1}_{x>0}$

The estimate is of the form

$$\hat{y} = f(\mathbf{w}, b, \mathbf{x}).$$

How do we estimate $(\mathbf{w}, b)$?

The Perceptron Algorithm

To ease notations, we put $\tilde{\mathbf{w}} = (w_1, \ldots, w_d, b)$ and $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$

Perceptron Algorithm - first (iterative) learning algorithm

- Start with $\tilde{\mathbf{w}} = 0$.
- Repeat over all samples:
  - if $y_i < \tilde{\mathbf{w}}, \tilde{\mathbf{x}}_i >< 0$ modify $\tilde{\mathbf{w}}$ into $\tilde{\mathbf{w}} + y_i \tilde{\mathbf{x}}_i$,
  - otherwise do not modify $\tilde{\mathbf{w}}$.

Exercise

- What is the rational behind this procedure?
- Is this procedure related to a gradient descent method?

Gradient descent:

1. Start with $\tilde{\mathbf{w}} = \tilde{\mathbf{w}}_0$
2. Update $\tilde{\mathbf{w}} \leftarrow \tilde{\mathbf{w}} - \eta \nabla \mathcal{L}(\tilde{\mathbf{w}})$, where $\mathcal{L}$ is the loss to be minimized.
3. Stop when $\tilde{\mathbf{w}}$ does not vary too much.

Exercise

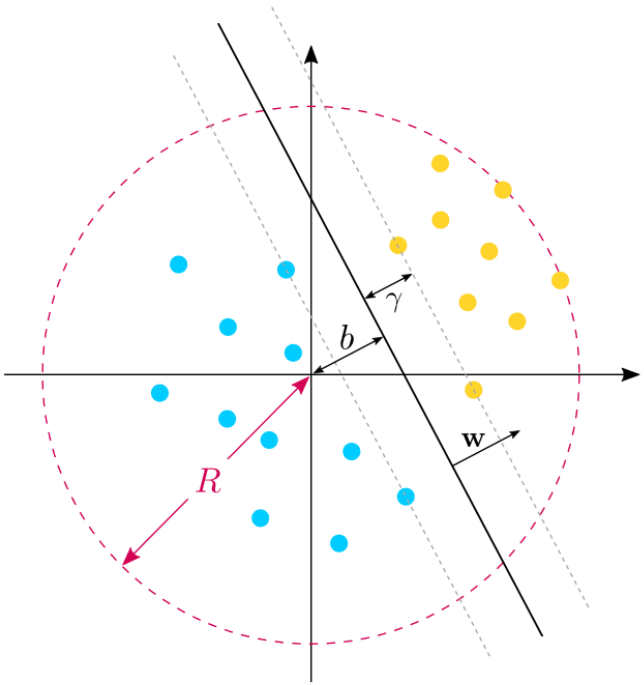


Figure: From `http://image.diku.dk/kstensbo/notes/perceptron.pdf`

Let $R = \max_i \|\mathbf{x}_i\|$. Let $\tilde{\mathbf{w}}^\star$ be the optimal hyperplane of margin

$$\gamma = \min_i y_i \langle \tilde{\mathbf{w}}^\star, \tilde{\mathbf{x}}_i \rangle,$$

with

$$\|\tilde{\mathbf{w}}^\star\| = 1.$$

Theorem (Block 1962; Novikoff 1963)

Assume that the training set $\mathcal{D}_n = \{(\boldsymbol{x}_1, y_i), \ldots, (\boldsymbol{x}_n, y_n)\}$ *is linearly separable* ($\gamma > 0$). *Start with* $\tilde{\boldsymbol{w}}_0 = 0$. *Then the number of updates* k *of the perceptron algorithm is bounded by*

$$k \leq \frac{1 + R^2}{\gamma^2}.$$

Exercise: Prove it!

Perceptron Limitations

Perceptron algorithm

- We have a data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
- We use the previous algorithm to learn the weight vector w.
- We predict using $f_w(\mathbf{x}) = \sigma(\langle w, \mathbf{x} \rangle)$.

What is the main problem of the previous algorithm?

Perceptron Limitations

Perceptron algorithm

- We have a data set $\mathcal{D}_n = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$
- We use the previous algorithm to learn the weight vector w.
- We predict using $f_w(\mathbf{x}) = \sigma(\langle w, \mathbf{x} \rangle)$.

What is the main problem of the previous algorithm?

Limitations

- Can't solve non linearly separable problems... (by design)
- Algorithm not robust to non linear separability!

History

The Perceptron project led by Rosenblatt was funded by the US Office of Naval Research.

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.

Press conference, 7 July 1958, New York Times.

For an extensive study of the perceptron, see [*Principles of neurodynamics. perceptrons and the theory of brain mechanisms*, Rosenblatt 1961]

Moving forward - ADALINE, MADALINE

In 1959 at Stanford, Bernard Widrow and Marcian Hoff developed AdaLinE (ADAptive LINear Elements) and **MAdaLinE** (Multiple AdaLinE) the latter being the first network successfully applied to a real world problem.

[*Adaptive switching circuits*, Bernard Widrow and Hoff 1960]

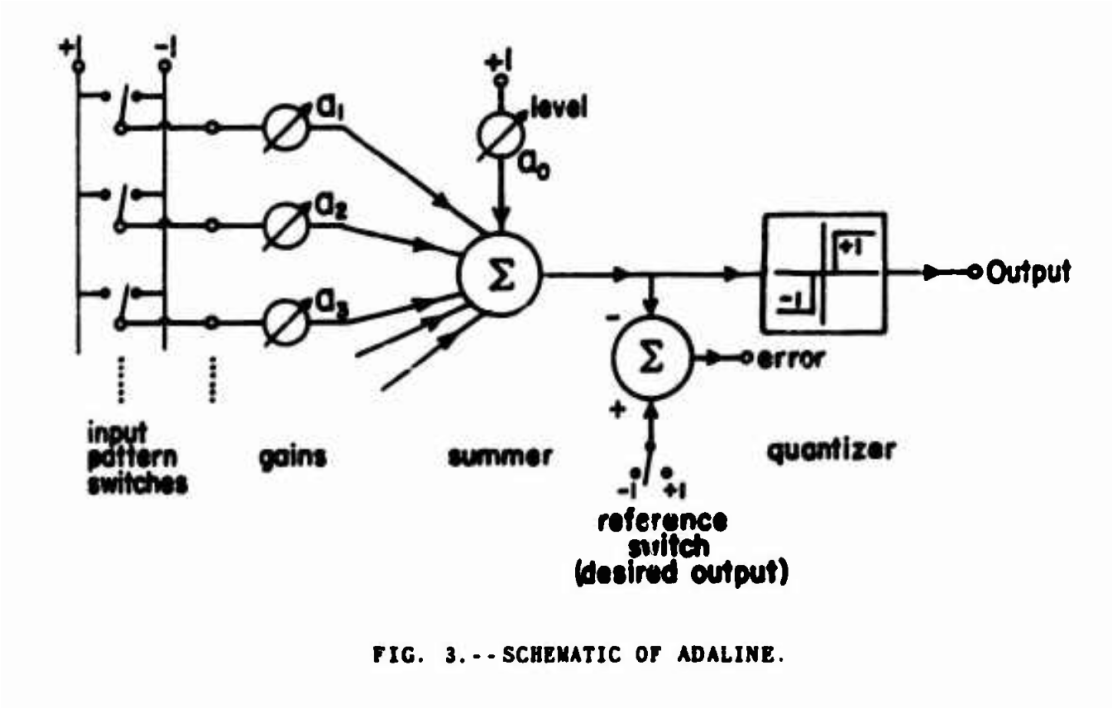


FIG. 3.--SCHEMATIC OF ADALINE.

Main differences with the perceptron:
The loss is the square difference between the sum of the weighted input and the output.
The optimization procedure is a gradient descent: all computations are trivial since the weighted sum is linear as a function of weights.

MADALINE

Many Adalines: network with one hidden layer composed of many Adaline units.

["Madaline Rule II: a training algorithm for neural networks", Winter and Widrow 1988]

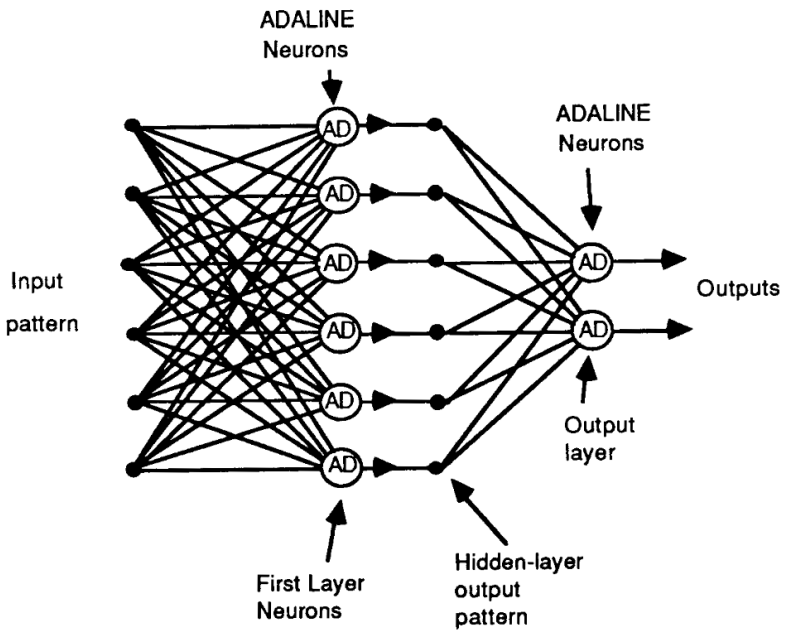


Figure 1: Layered feed-forward ADALINE network.

Applications:

- Speech and pattern recognition

  ["Real-Time Adaptive Speech-Recognition System", Talbert et al. 1963]

- Weather forecasting

  ["Application of the adaline system to weather forecasting", Hu 1964]

- Adaptive filtering and adaptive signal processing

  ["Adaptive signal processing", Bernard and Samuel 1985]

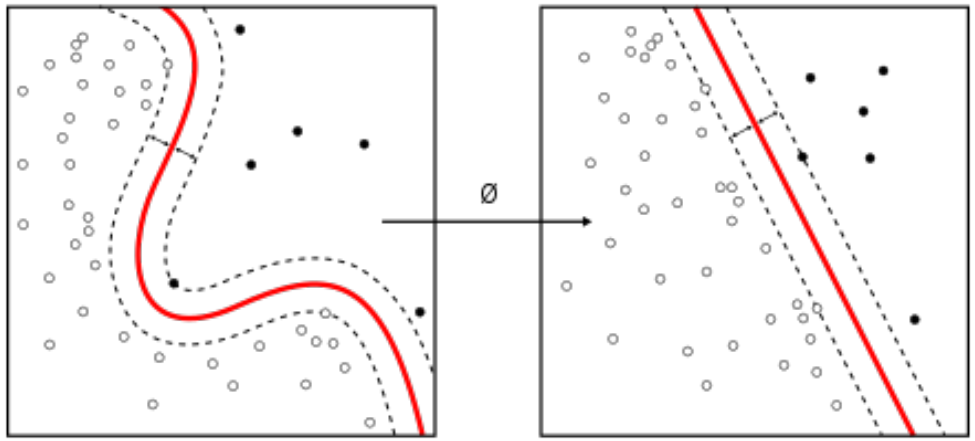
To go further

The kernel perceptron algorithm was already introduced by

["Theoretical foundations of the potential function method in pattern recognition learning", Aizerman 1964].

General idea (work for all methods using only dot product): replace the dot product by a more complex kernel function.

Linear separation in the original space becomes a linear separation in a more complex space, i.e., a non linear separation in the original space.



Margin bounds for the Perceptron algorithm in the general non-separable case were proven by

["Large margin classification using the perceptron algorithm", Freund and Schapire 1999]

and then by

["Perceptron mistake bounds", Mohri and Rostamizadeh 2013]

who extended existing results and gave new L1 bounds.

AI winter

In 1969, Minsky and Papert exhibit the fact that it was difficult for perceptron to detect parity (number of activated pixels) and connectedness (are the pixels connected?). Besides, it was known that they cannot represent simple non linear function such as XOR function.

[*Perceptrons: an introduction to computational geometry*, Minsky and Papert 2017]

There is no reason to suppose that any of [the virtue of perceptrons] carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension is sterile. Perhaps some powerful convergence theorem will be discovered, or some profound reason for the failure to produce an interesting "learning theorem" for the multilayered machine will be found.

Minsky, Papert

It is often believed that Minsky and Papert state the ineffectiveness of neural networks whereas the real story is uneven. Nevertheless, this book is the starting point of the period known as "AI winter", corresponding to a significant decline in funding of neural network research.

To learn more about the controversy between Rosenblatt and Minsky, Papert, see ["A sociological study of the official history of the perceptrons controversy", Olazaran 1996].

AI Winter: the XOR function

Exercise. Consider two binary variables $x_1 \in \{0,1\}$ and $x_2 \in \{0,1\}$. The logical function AND applied to x_1, x_2 is defined as

$$\text{AND} : \{0,1\}^2 \to \{0,1\}$$

$$(x_1, x_2) \mapsto \begin{cases} 1 & \text{if } x_1 = x_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

1) Find a perceptron (i.e., weights and bias) that implements the AND function.

The logical function XOR applied to x_1, x_2 is defined as

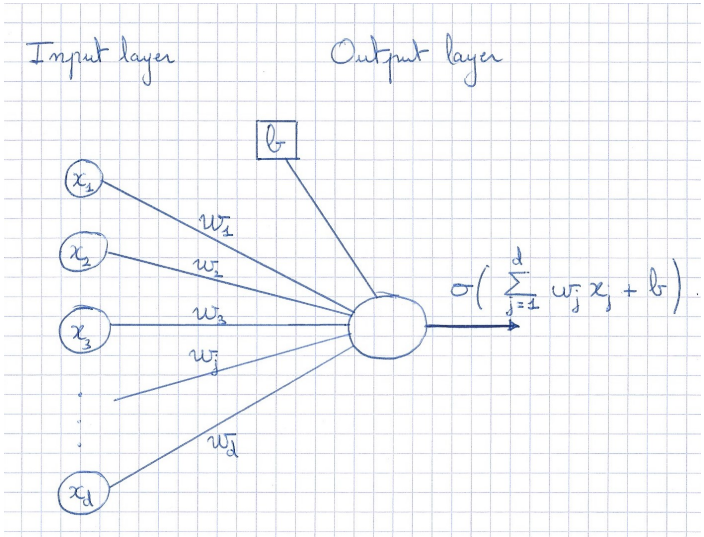
$$\text{XOR} : \{0,1\}^2 \to \{0,1\}$$

$$(x_1, x_2) \mapsto \begin{cases} 0 & \text{if } x_1 = x_2 = 0 \text{ or } x_1 = x_2 = 1 \\ 1 & \text{otherwise} \end{cases}$$

2) Prove that no perceptron can implement the XOR function.

3) Find a neural network with one hidden layer that implements the XOR function.

Nowadays - Logistic Perceptron

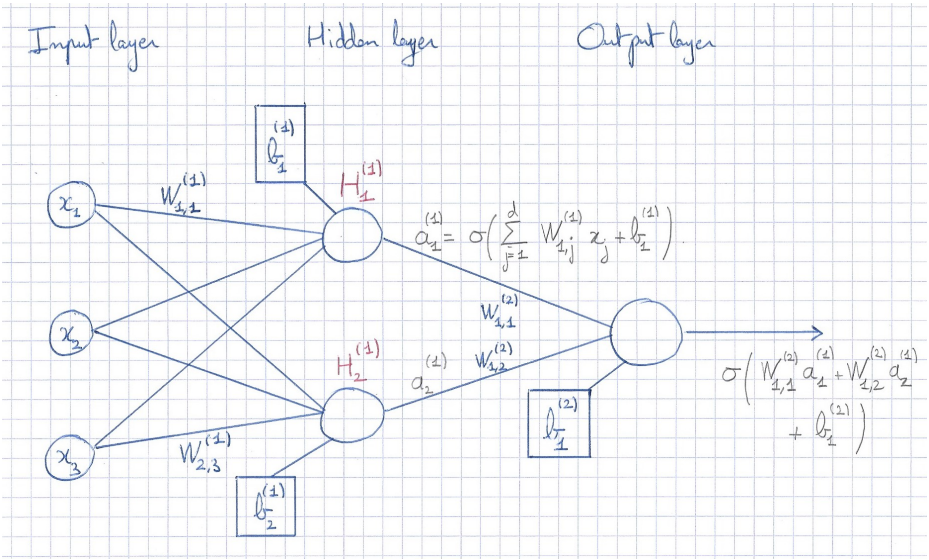


Logistic unit

- Structure:
  - Mix inputs with a weighted sum,
  - Apply the logistic function $\sigma(t) = e^t/(1+e^t)$,
  - Threshold at $1/2$ to make a decision!
- Logistic weights learned by minimizing the -log-likelihood via gradient descent.

Outline

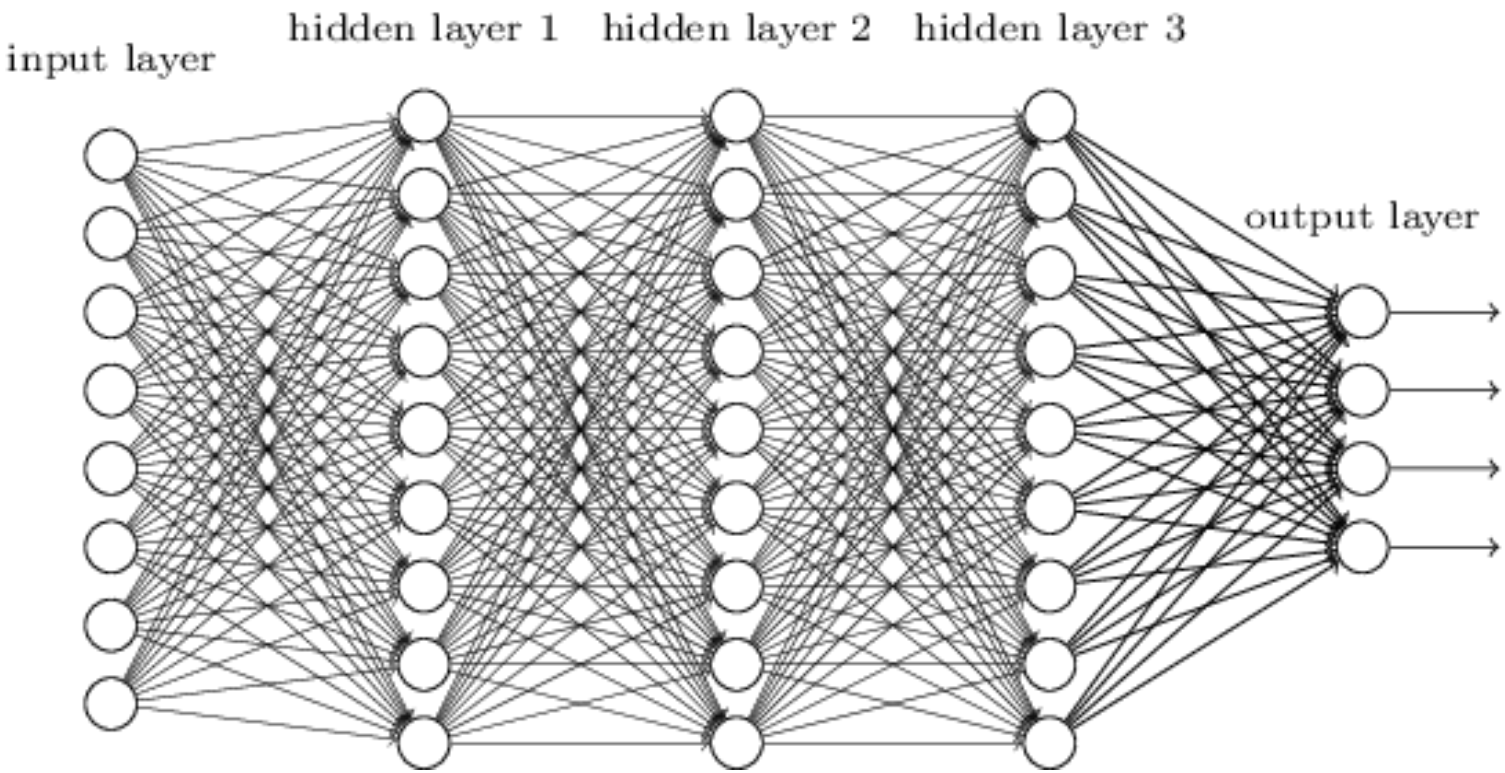
Neural network with one hidden layer



Generic notations:

- $W_{i,j}^{(\ell)}$: weights between the j neuron in the $\ell - 1$ layer and the i neuron of the ℓ layer.
- $b_j^{(\ell)}$: bias of the j neuron of the ℓ layer.
- $a_j^{(\ell)}$: output of the j neuron of the ℓ layer.
- $z_j^{(\ell)}$: input of the j neuron of the ℓ layer, such that $a_j^{(\ell)} = \sigma(z_j^{(\ell)})$.

A first idea



Perceptron algorithm does not work anymore!

Gradient Descent Algorithm

- Empirical risk minimization:

$$\underset{w}{\text{argmin}} \frac{1}{n}\sum_{i=1}^{n} \ell(Y_i, \sigma(\langle w, \mathbf{X}_i\rangle)) \equiv \underset{w}{\text{argmin}} \frac{1}{n}\sum_{i=1}^{n} \ell_i$$

- If ℓ and σ are differentiable almost everywhere:

$$\nabla_w \ell_i = \frac{\partial \ell(Y_i, \sigma(\langle w, \mathbf{X}_i\rangle))}{\partial w} = \mathbf{X}_i \sigma'(\langle w, \mathbf{X}_i\rangle)\partial_2 \ell(Y_i, \sigma(\langle w, \mathbf{X}_i\rangle))$$

(Stochastic) Gradient descent rule:

While $|w_{t+1} - w_t| \geq \varepsilon$ do

- Sample $I_t \subset \{1, \ldots, n\}$
-

$$w_{t+1} = w_t - \eta \frac{1}{|I_t|}\sum_{i \in I_i} \nabla_w \ell_i$$

- If ℓ is convex, this algorithm converges.

Backprop Algorithm

A Clever Gradient Descent Implementation

- Popularized by Rumelhart, McClelland, Hinton in 1986.
- Can be traced back to Werbos in 1974.
- Nothing but the use of chain rule derivation with a touch of dynamic programing.

- Key ingredient to make the Neural Networks work!
- Still at the core of Deep Learning algorithm.

Backpropagation equations

Neural network with L layers, with vector output, with quadratic cost

$$C = \frac{1}{2}\|y - a^{(L)}\|^2.$$

By definition,

$$\delta_j^{(\ell)} = \frac{\partial C}{\partial z_j^{(\ell)}}.$$

The four fundamental equations of backpropagation are given by

$$\begin{aligned}
\delta^{(L)} &= \nabla_a C \odot \sigma'(z^{(L)}), \\
\delta^{(\ell)} &= ((w^{(\ell+1)})^T \delta^{(\ell+1)}) \odot \sigma'(z^{(\ell)}) \\
\frac{\partial C}{\partial b_j^{(\ell)}} &= \delta_j^{(\ell)} \\
\frac{\partial C}{\partial w_{j,k}^{(\ell)}} &= a_k^{(\ell-1)} \delta_j^{(\ell)}.
\end{aligned}$$

Backpropagation Algorithm

Let

$$\delta_j^{(\ell)} = \frac{\partial C}{\partial z_j^{(\ell)}},$$

where $z_j^{(\ell)}$ is the entry of the neuron j of the layer ℓ.

Backpropagation Algorithm

- Initialize randomly weights and bias in the network.
- For each training sample x_i,
  1. **Feedforward:** let x_i go through the network and store the value of activation function and its derivative, for each neuron.
  2. **Output error:** compute the neural network error for x_i.
  3. **Backpropagation:** compute recursively the vectors $\delta^{(\ell)}$ starting from $\ell = L$ to $\ell = 1$.
- Update the weights and bias using the backpropagation equations.

Neural Network terminology

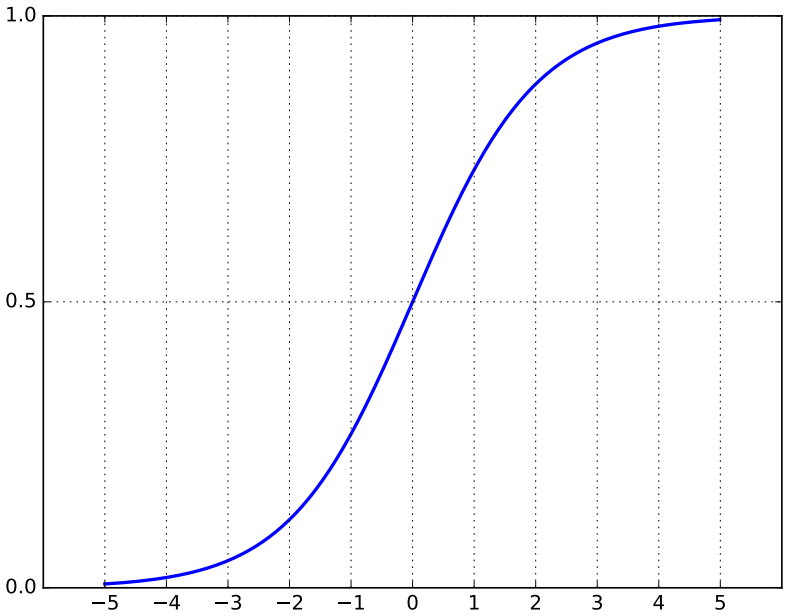
- Epoch: one forward pass and one backward pass of all training examples.
- (Mini) Batch size: number of training examples in one forward/backward pass. The higher the batch size is, the more memory space you'll need.
- Number of iterations: number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

For example, for 1000 training examples, if you set the batch size at 500, it will take 2 iterations to complete 1 epoch.

Outline

Outline

Sigmoid



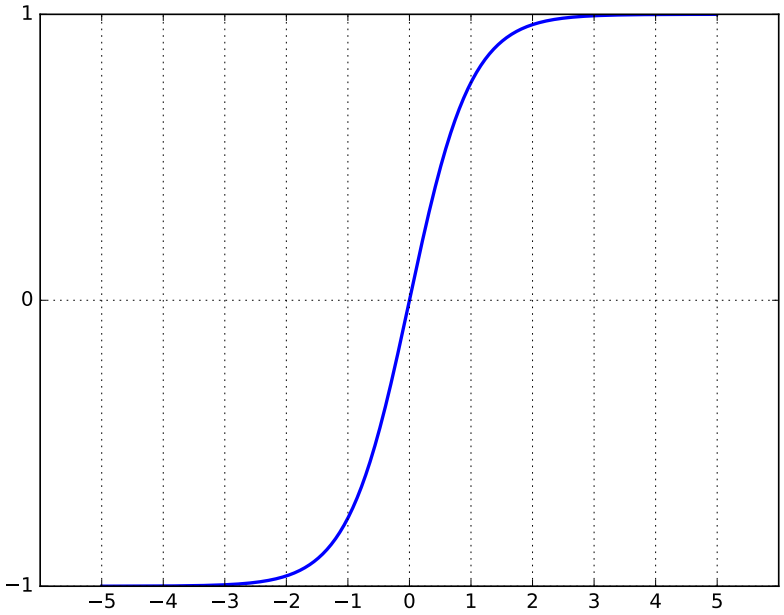
- Sigmoid function

$$x \mapsto \frac{\exp(x)}{1 + \exp(x)}$$

- Problems:

1. Sigmoid is not a zero centered function -> need for rescaling data.
2. Saturated function: gradient killer -> need for rescaling data
3. Plus: exp is a bit computational expensive

Tanh



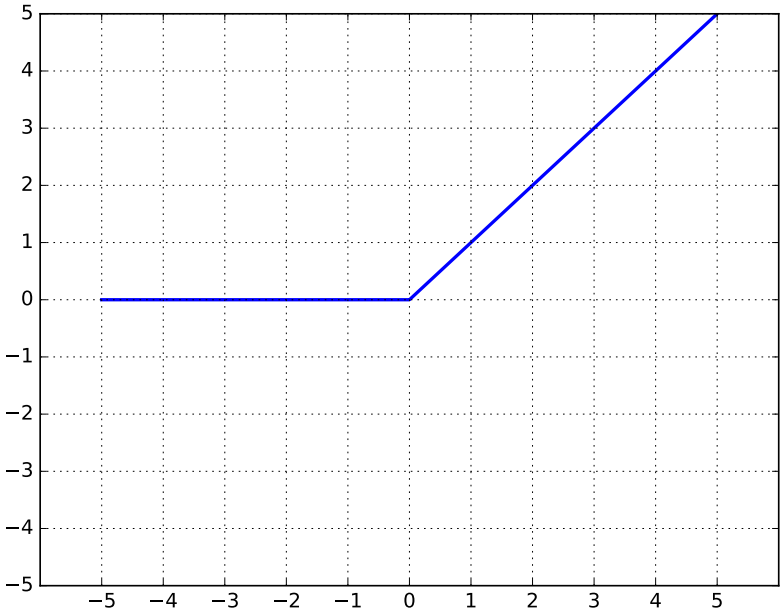
- Hyperbolic tangente function

$$x \mapsto \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- Problems:
  1. Tanh is a zero centered function -> no need for rescaling data
  2. Saturated function: gradient killer -> need for rescaling data
  3. Plus: exp is a bit computational expensive

Note: $\tanh(x) = 2\sigma(2x) - 1$.

Rectified Linear Unit



- ReLU / positive part

$$x \mapsto \max(0, x)$$

- Problems:
  1. Not a saturated function.
  2. Computationally efficient
  3. Empirically, convergence is faster than sigmoid/tanh.
  4. Plus: biologically plausible

A little bit more on ReLU

Introduced by ["Imagenet classification with deep convolutional neural networks", Krizhevsky et al. 2012] in AlexNet

Problems:

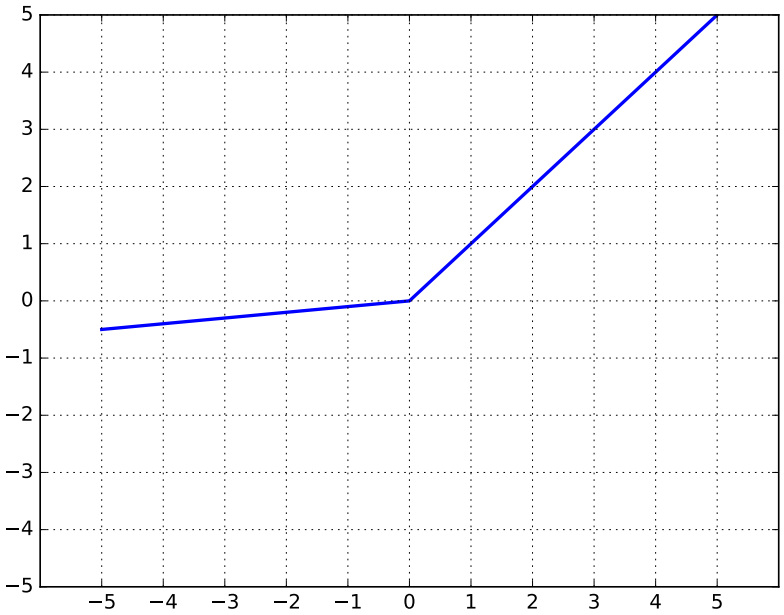
- Not zero centered
- Gradient is null if $x < 0$
- If weights are not properly initialized, ReLU output can be zero.

Usually initial bias for ReLU so that they fire up: useful or not? Mystery...

Related to biology ["Deep sparse rectifier neural networks", Glorot, Bordes, et al. 2011]:

- Most of the time, neurons are inactive.
- when they activate, their activation is proportional to their input.

Leaky ReLU/ Parametric ReLU / Absolute value rectification



$$x \mapsto \max(\alpha x, x)$$

- Leaky ReLU: $\alpha = 0.1$

  ["Rectifier nonlinearities improve neural network acoustic models", Maas et al. 2013]

- Absolute Value Rectification:

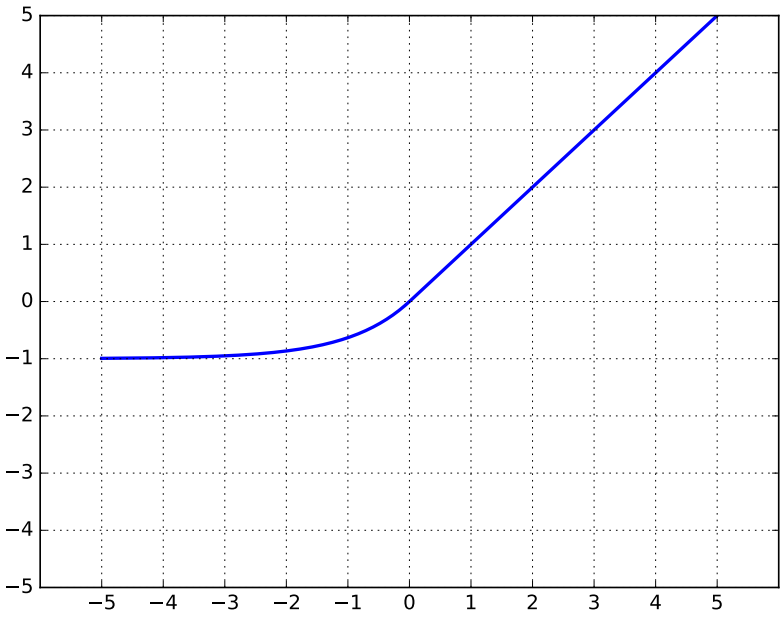
  $$\alpha = -1$$

  ["What is the best multi-stage architecture for object recognition?", Jarrett et al. 2009]

- Parametric ReLU: α optimized during backpropagation. Activation function is learned.

  ["Empirical evaluation of rectified activations in convolutional network", Xu et al. 2015]

ELU



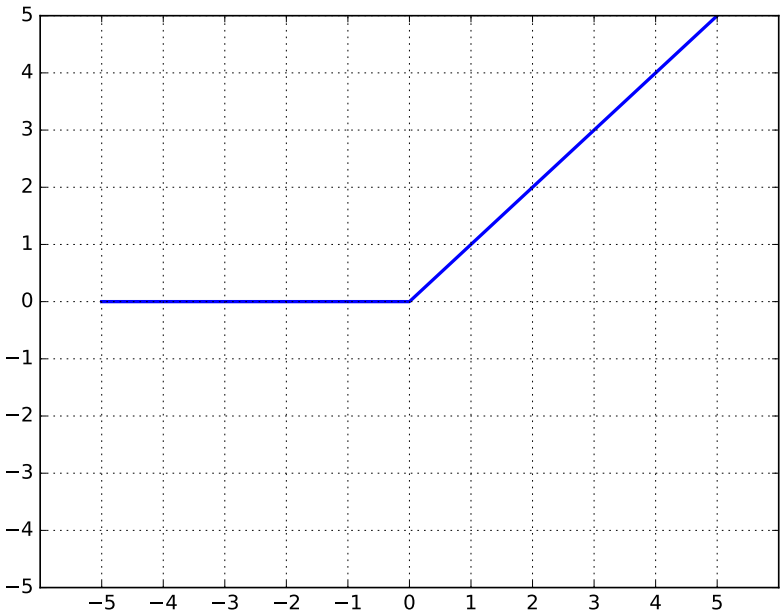
- Exponential Linear Unit

$$x \mapsto \begin{cases} x & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{otherwise} \end{cases}$$

- Negative saturation regime, closer to zero mean output.
- α is set to 1.0.
- Robustness to noise

  ["Fast and accurate deep network learning by exponential linear units (elus)", Clevert et al. 2015]

Maxout



ReLU / positive part

$$x \mapsto \max(w_1 x + b_1, w_2 x + b_2)$$

- Linear regime, not saturating, not dying.
- Number of parameters multiplied by 2 (resp. by k if k entries)
- Learn piecewise linear functions (up to k pieces).

  ["Maxout networks", Goodfellow, Warde-Farley, et al. 2013] ["Deep maxout neural networks for speech recognition", Cai et al. 2013]
- Resist to catastrophic forgetting

  ["An empirical investigation of catastrophic forgetting in gradient-based neural networks", Goodfellow, Mirza, et al. 2013]

Conclusion on activation functions

- Use ReLU.
- Test Leaky ReLU, maxout, ELU.
- Try out Tanh, but not expect too much.
- Do not use sigmoid.

Outline

Output units

- Linear output unit:
$$\hat{y} = W^T h + b$$
$\rightarrow$ Linear regression based on the new variables h.

- Sigmoid output unit, used to predict $\{0, 1\}$ outputs:
$$\mathbb{P}(Y = 1|h) = \sigma(W^T h + b),$$
where $\sigma(t) = e^t/(1 + e^t)$..
$\rightarrow$ Logistic regression based on the new variables h.

- Softmax output unit, used to predict $\{1, \ldots, K\}$:
$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$
where, each z_i is the activation of one neuron of the previous layer, given by $z_i = W_i^T h + b_i$.
$\rightarrow$ Multinomial logistic regression based on the new variables h.

Multinomial logistic regression

Generalization of logistic regression for multiclass outputs: for all $1 \leq k \leq K$,

$$\log\left(\frac{\mathbb{P}[Y_i = k]}{Z}\right) = \beta_k X_i,$$

Hence, for all $1 \leq k \leq K$,

$$\mathbb{P}[Y_i = k] = \frac{1}{Z} e^{\beta_k X_i},$$

where

$$Z = \sum_{k=1}^{K} e^{\beta_k X_i}.$$

Thus,

$$\mathbb{P}[Y_i = k] = \frac{e^{\beta_k X_i}}{\sum_{\ell=1}^{K} e^{\beta_\ell X_i}}.$$

Biology bonus

Softmax, used with cross-entropy:

$$-\log(\mathbb{P}(Y=k|z)) = -\log \mathrm{softmax}(z)_k = z_k - \log\Big(\sum_j \exp(z_j)\Big)$$

$$\simeq z_k - \max_j z_j \simeq 0,$$

if $\mathrm{softmax}(z)_{\hat{y}}$ is maximal $\rightarrow$ no contribution to the cost.

Lateral inhibition: believed to exist between nearby neurons in the cortex. When the difference between the max and the other is large, winner takes all: one neuron is set to 1 and the others go to zero.

More complex models: Conditional Gaussian Mixture: y is multimodal

["On supervised learning from sequential data with applications for speech recognition"; "Generating sequences with recurrent neural networks", Schuster 1999; Graves 2013].

Outline

Cost functions

- Mean Square Error (MSE)

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - f_\theta(\mathbf{X}_i))^2$$

- Mean Absolute Error

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = \frac{1}{n}\sum_{i=1}^{n}|Y_i - f_\theta(\mathbf{X}_i)|$$

- $0-1$ Error

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{Y_i \neq f_\theta(\mathbf{X}_i)}$$

Cost functions

- Cross entropy (or negative log-likelihood):

$$\frac{1}{n}\sum_{i=1}^{n}\ell(Y_i, f_\theta(\mathbf{X}_i)) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{1}_{y_i=k}\log\left([f_\theta(\mathbf{X}_i)]_k\right)$$

Cross-entropy:

- Very popular!
- Should help to prevent saturation phenomenon compared to Mean square error.

$$-\log(\mathbb{P}(Y = y_i|X)) = -\log(\sigma((2y-1)(W^T+b))),$$

  where saturation occurs when $(2y-1)(W^T+b) \gg 1$ that is when the prediction is correct. Otherwise, gradient quickly corrects mistakes.
- May be difficult to compute the gradient when cross-entropy is large

Mean square error should not be used with softmax output units

["Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition", Bridle 1990]

Outline

First very bad idea

Set all weights and bias to the same value:

- All neurons are going to be the same, in each iteration
- Too much symmetry!

Need to break this symmetry!

Small or big weights?

1. First idea: small random numbers, typically
$$0.01 \times \mathcal{N}(0, 10^{-4}).$$
   - work for small networks
   - for big networks ($\sim$ 10 layers) output become dirac in 0: there is no activation at all.

2. Second idea: "big random numbers"
$$\mathcal{N}(0, 10^{-4}).$$
   $\rightarrow$ Saturating phenomenon

In any case, no need to tune the bias: they can be initially set to zero.

Other initialization

Idea: the variance of the input should be the same as the variance of the output.

1. Xavier initialization
   Initialize bias to zero and weights randomly using
   $$\mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right],$$
   where n_j is the size of layer j
   ["Understanding the difficulty of training deep feedforward neural networks", Glorot and Bengio 2010].
   $\rightarrow$ Sadly, it does not work for ReLU (non activated neurons)

2. He et al. initialization
   Initialize bias to zero and weights randomly using
   $$\mathcal{N}\left(0, \frac{\sqrt{2}}{n_j}\right),$$
   where n_j is the size of layer j.
   ["Delving deep into rectifiers: Surpassing human-level performance on imagenet classification", He et al. 2015].

Bonus: ["All you need is a good init", Mishkin and Matas 2015]

Exercise

1. Consider a neural network with two hidden layers (containing n_1 and n_2 neurons respectively) and equipped with linear hidden units. Find a simple sufficient condition on the weights so that the variance of the hidden units stay constant accross layers.
2. Considering the same network, find a simple sufficient condition on the weights so that the gradient stay constant across layers when applying backpropagation procedure.
3. Based on previous questions, propose a simple way to initialize weights.

Outline

Batch normalization

The network converges faster if its input are scaled (mean, variance) and decorrelated.

["Efficient backprop", LeCun et al. 1998]

Hard to decorrelate variables: requiring to compute covariance matrix.

["Batch normalization: Accelerating deep network training by reducing internal covariate shift", Ioffe and Szegedy 2015]

Ideas:

- Improving gradient flows
- Allowing higher learning rates
- Reducing strond dependence on initialization
- Related to regularization (maybe slightly reduces the need for Dropout)

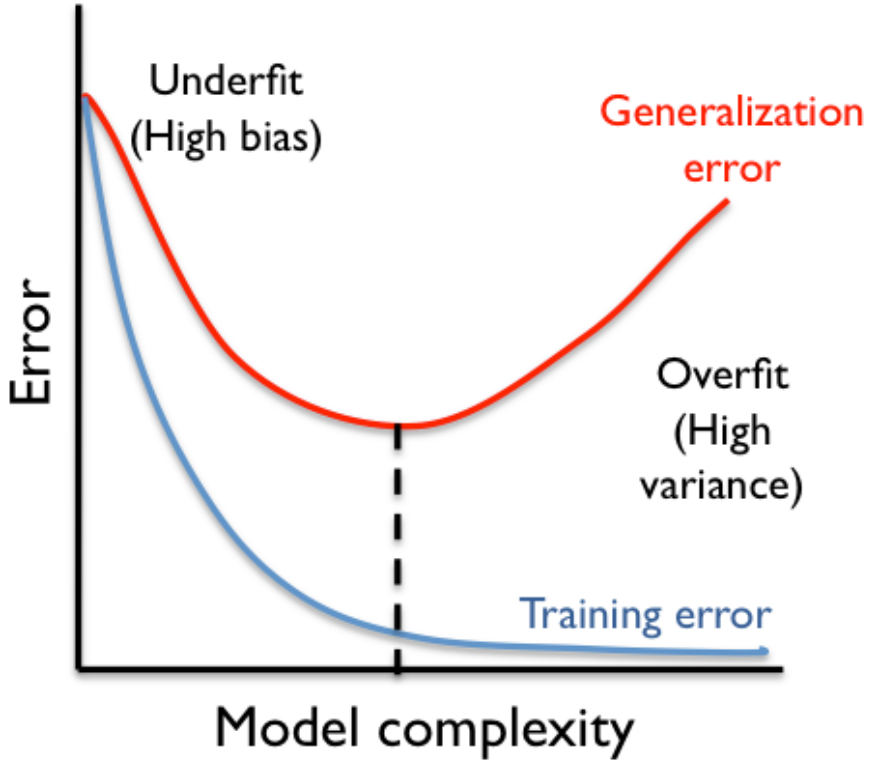
Algorithm

See ["Batch normalization: Accelerating deep network training by reducing internal covariate shift", Ioffe and Szegedy 2015]

1. For every unit in the first layer,
   1. $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$
   2. $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
   3. $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$
   4. $y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$
2. y_i is fed to the next layer and the procedure iterates.
3. Backpropagation is performed on the network parameters including $(\gamma^{(k)}, \beta^{(k)})$. This returns a network.
4. For inference, compute the average over many training batches $\mathcal{B}$:
$$\mathbb{E}_{\mathcal{B}}[x] = \mathbb{E}_{\mathcal{B}}[\mu_{\mathcal{B}}] \quad \text{and} \quad \mathbb{V}_{\mathcal{B}}[x] = \frac{m}{m-1}\mathbb{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2].$$
5. For inference, replace every function $x \mapsto NB_{\gamma,\beta}(x)$ in the network by
$$x \mapsto \frac{\gamma}{\sqrt{\mathbb{V}_{\mathcal{B}}[x] + \varepsilon}} x + \left(\beta - \frac{\gamma \mathbb{E}_{\mathcal{B}}[x]}{\sqrt{\mathbb{V}_{\mathcal{B}}[x] + \varepsilon}}\right).$$

Outline

Regularizing to avoid overfitting



Avoid overfitting by imposing some constraints over the parameter space.

Reducing variance and increasing bias.

Overfitting

Many different manners to avoid overfitting:

- Penalization (L1 or L2)
  Replacing the cost function $\mathcal{L}$ by

$$\tilde{\mathcal{L}}(\theta, X, y) = \mathcal{L}(\theta, X, y) + \text{pen}(\theta).$$

- Early stopping
  Stop the gradient descent procedure when the error on the validation set increases.
- Dropout
  Randomly kill some neurons during optimization and predict with the full network.
- Soft weight sharing
  Reduce the parameter space artificially by imposing explicit constraints

Outline

Constraint the optimization problem

$$\min_{\theta} \mathcal{L}(\theta, X, y), \quad \text{s.t. } \text{pen}(\theta) \leq \textit{cste}.$$

Using Lagrangian formulation, this problem is equivalent to:

$$\min_{\theta} \mathcal{L}(\theta, X, y) + \lambda \, \text{pen}(\theta),$$

where

- $\mathcal{L}(\theta, X, y)$ is the loss function (data-driven term)
- pen is a function that increases when θ becomes more *complex* (penalty term)
- $\lambda \geq 0$ is a constant standing for the strength of the penalty term.

For Neural Networks, pen only penalizes the weights and not the bias: the later being easier to estimate than weights.

Example of penalization

$$\min_{\theta} \mathcal{L}(\theta, X, y) + \text{pen}(\theta),$$

- Ridge

$$\text{pen}(\theta) = \lambda \|\theta\|_2^2$$

["Ridge regression: Biased estimation for nonorthogonal problems", Hoerl and Kennard 1970].
See also ["Lecture notes on ridge regression", Wieringen 2015]

- Lasso

$$\text{pen}(\theta) = \lambda \|\theta\|_1$$

["Regression shrinkage and selection via the lasso", Tibshirani 1996]

- Elastic Net

$$\text{pen}(\theta) = \lambda \|\theta\|_2^2 + \mu \|\theta\|_1$$

["Regularization and variable selection via the elastic net", Zou and Hastie 2005]

Simple case: linear regression

Linear regression

The estimate of linear regression $\hat{\beta}$ is given by

$$\hat{\beta} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - \sum_{j=1}^{d} \beta_j x_i^{(j)})^2,$$

which can be written as

$$\hat{\beta} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \| Y - \mathbb{X}\beta \|_2^2,$$

where $\mathbb{X} \in M_{n,d}(\mathbb{R})$.

Solution:

$$\hat{\beta} = (\mathbb{X}'\mathbb{X})^{-1}\mathbb{X}'Y.$$

Penalized regression

Penalized linear regression

The estimate of linear regression $\hat{\beta}_{\lambda,q}$ is given by

$$\hat{\beta}_{\lambda,q} \in \underset{\beta \in \mathbb{R}^d}{\text{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_q^q.$$

- $q = 2$: Ridge linear regression
- $q = 1$: LASSO

Ridge regression, $q = 2$

Ridge linear regression

The ridge estimate $\hat{\beta}_{\lambda,2}$ is given by

$$\hat{\beta}_{\lambda,2} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_2^2.$$

Solution:

$$\hat{\beta}_{\lambda,2} = (\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\mathbb{X}'Y.$$

This estimate has a bias equal to $-\lambda(\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\beta$, and a variance $\sigma^2(\mathbb{X}'\mathbb{X} + \lambda I)^{-1}\mathbb{X}'\mathbb{X}(\mathbb{X}'\mathbb{X} + \lambda I)^{-1}$. Note that

$$\mathbb{V}[\hat{\beta}_{\lambda,2}] \leq \mathbb{V}[\hat{\beta}].$$

In the case of orthonormal design ($\mathbb{X}'\mathbb{X} = I$), we have

$$\hat{\beta}_{\lambda,2} = \frac{\hat{\beta}}{1+\lambda} = \frac{1}{1+\lambda} X_j' Y.$$

Ridge regression $q = 2$

In general, the **ridge** penalization considers

$$\text{pen}(\theta) = \frac{1}{2}\|\beta\|_2^2 = \frac{1}{2}\sum_{j=1}^{d}\beta_j^2,$$

in the optimization problem

$$\min_{\theta} \mathcal{L}(\beta, X, y) + \text{pen}(\beta),$$

It penalizes the "size" of β.

This is the most widely used penalization

- It's nice and easy
- It allows to "deal" with correlated features.
- It actually helps training! With a ridge penalization, the optimization problem is easier.

Sparsity

There is another desirable property on $\hat{\beta}$

If $\hat{\beta}_j = 0$, then feature j has no impact on the prediction:

$$\hat{y} = \text{sign}(x^\top \hat{\beta} + \hat{b})$$

If we have many features (d is large), it would be nice if $\hat{\beta}$ contained **zeros**, and many of them

- Means that only **few** features are statistically relevant.
- Means that only **few** features are useful to predict the label

Leads to a simpler model, with a "reduced" dimension

How do we enforce sparsity in β?

Sparsity

Tempting to solve

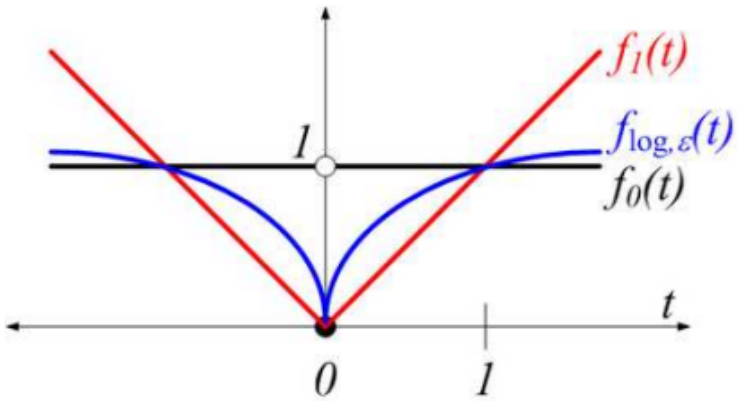
$$\hat{\beta}_{\lambda,0} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_0.$$

where

$$\|\beta\|_0 = \#\{j \in \{1, \dots, d\} : \beta_j \neq 0\}.$$

To solve this, explore **all** possible supports of β. Too long! (NP-hard)

Find a convex proxy of $\|\|_0$: the ℓ_1-**norm** $\|\beta\|_1 = \sum_{j=1}^d |\beta_j|$



LASSO

Ridge linear regression

The LASSO (Least Absolute Selection and Shrinkage Operator) estimate of linear regression $\hat{\beta}_{\lambda,2}$ is given by

$$\hat{\beta}_{\lambda,2} \in \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \|Y - \mathbb{X}\beta\|_2^2 + \lambda \|\beta\|_1.$$

Solution: No close form in the general case

If the X_j are orthonormal then

$$\beta_{\lambda,1} = X_j' Y \left(1 - \frac{\lambda}{2|X_j' Y|}\right)_+,$$

where $(x)_+ = \max(0, x)$.
Thus, in the very specific case of orthogonal design, we can easily show that L1 penalization implies a sparse vector if the parameter λ is properly tuned.

Sparsity - a picture

Why ℓ_2 (ridge) does not induce sparsity?

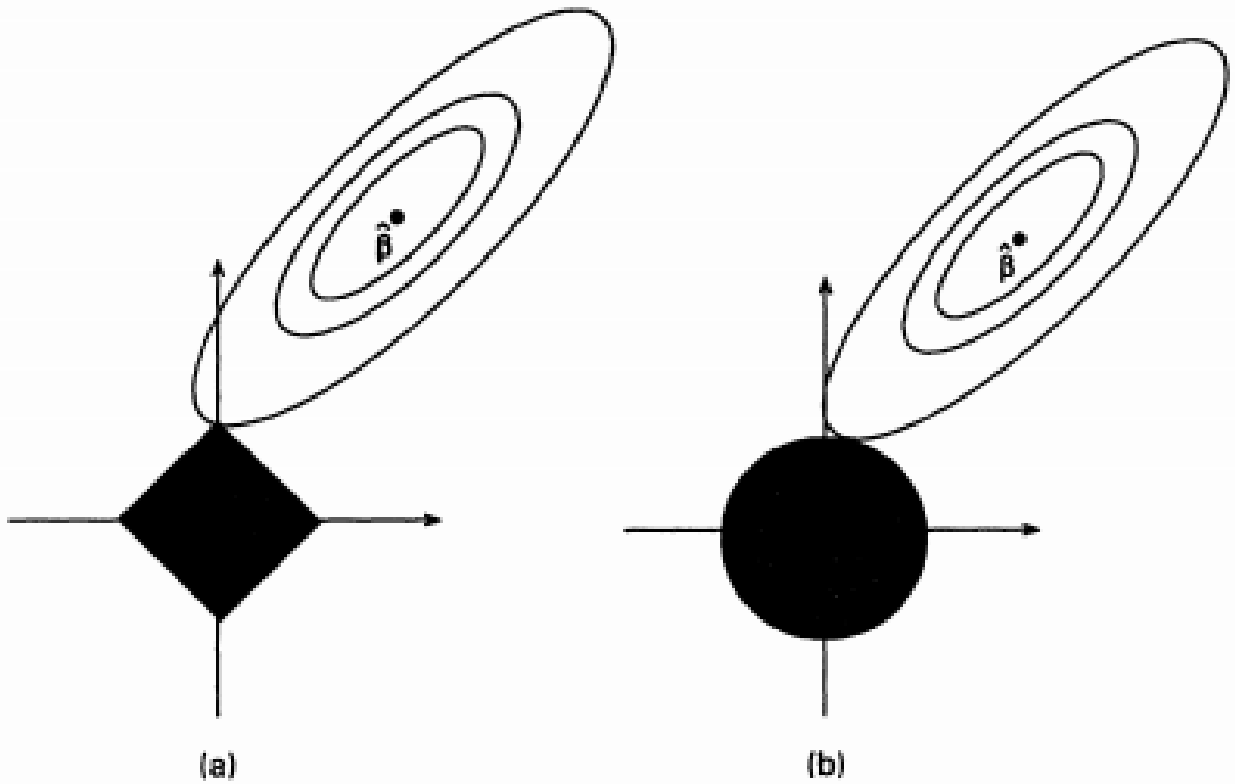


Fig. 2. Estimation picture for (a) the lasso and (b) ridge regression

Outline

Dropout

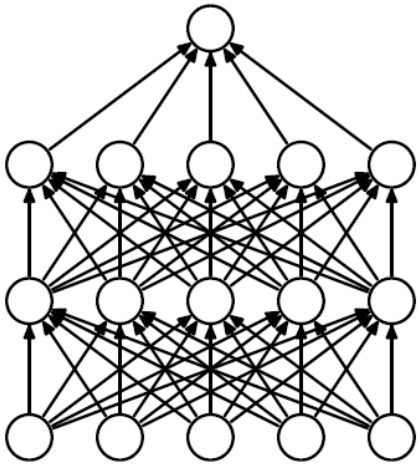
Dropout refers to dropping out units (hidden and visible) in a neural network, i.e., temporarily removing it from the network, along with all its incoming and outgoing connections.

Each unit is independently retained with probability

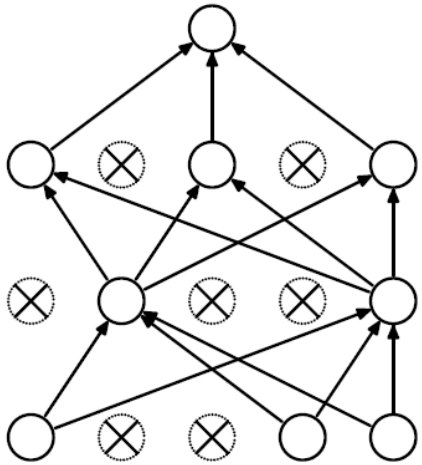
- $p = 0.5$ for hidden units
- $p \in [0.5, 1]$ for input units, usually $p = 0.8$.

["Improving neural networks by preventing co-adaptation of feature detectors", Hinton et al. 2012]

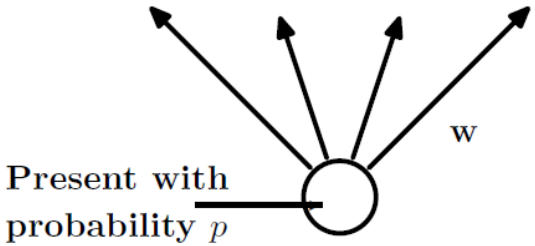
Dropout



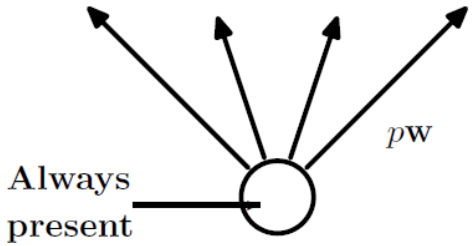
(a) Standard Neural Net



(b) After applying dropout.



(a) At training time



(b) At test time

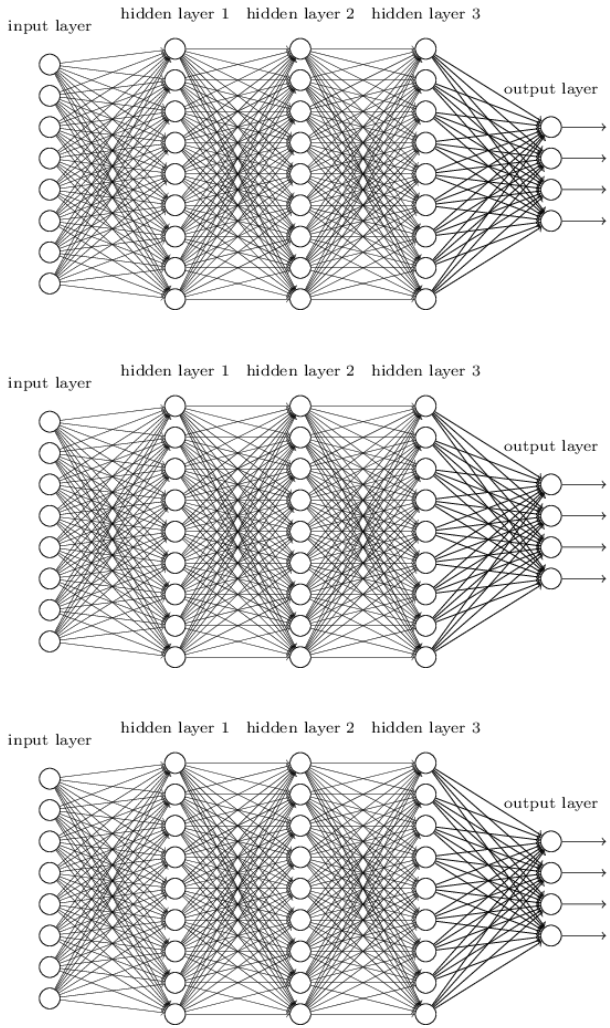
Dropout algorithm

Training step. While *not convergence*

1. Inside one epoch, for each mini-batch of size m,
   1. Sample m different mask. A mask consists in one Bernoulli per node of the network (inner and entry nodes but not output nodes). These Bernoulli variables are *i.i.d.*. Usually
      - ★ the probability of selecting an hidden node is 0.5
      - ★ the probability of selecting an input node is 0.8
   2. For each one of the m observation in the mini-batch,
      - ★ Do a forward pass on the masked network
      - ★ Compute backpropagation in the masked network
      - ★ Compute the gradient for each weights
   3. Update the parameter according to the usual formula.

Prediction step.
Use all neurons in the network with weights given by the previous optimization procedure, times the probability p of being selected (0.5 for inner nodes, 0.8 for input nodes).

Another way of seeing dropout - Ensemble method



Averaging many different neural networks.

Different can mean either:

- randomizing the data set on which we train each network (via subsampling)

  Problem: not enough data to obtain good performance...

- building different network architectures and train each large network separately on the whole training set

  Problem: computationally prohibitive at training time and test time!

Miscelleanous:

["Fast dropout training", Wang and Manning 2013]

["Dropout: A simple way to prevent neural networks from overfitting", Srivastava et al. 2014]

Exercise: linear units

1. Consider a neural networks with linear activation functions. Prove that dropout can be seen as a model averaging method.
2. Given one training example, consider the error of the ensemble of neural network and that of one random neural network sample with dropout:

$$E_{ens} = \frac{1}{2}(y - a_{ens})^2 = \frac{1}{2}(y - \sum_{j=1}^{d} p_i w_i x_i)^2$$

$$E_{single} = \frac{1}{2}(y - a_{single})^2 = \frac{1}{2}(y - \sum_{j=1}^{d} \delta_i w_i x_i)^2,$$

where $\delta_i \in \{0, 1\}$ represents the presence ($\delta_i = 1$) or the absence of a connexion between the input x_i and the output.

Prove that

$$\mathbb{E}[\nabla_w E_{single}] = \nabla_w E_{ens} + \frac{1}{2} \sum_{j=1}^{d} w_i^2 x_i^2 \mathbb{V}(\delta_i).$$

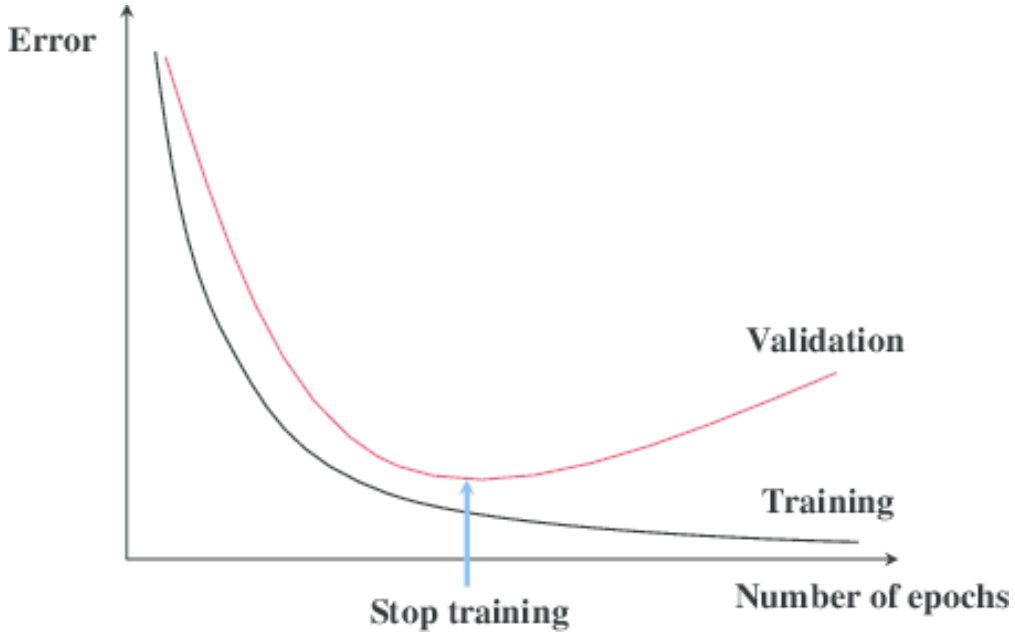
Comment.

Outline

Early stopping

Idea:

- Store the parameter values that lead to the lowest error on the validation set
- Return these values rather than the latest ones.



Early stopping algorithm

Parameters:

- patience p of the algorithm: number of times to observe worsening validation set error before giving up;
- the number of steps n between evaluations.

1. Start with initial random values θ_0.
2. Let $\theta^\star = \theta_0$, $\mathrm{Err}^\star = \infty$, $j = 0$, $i = 0$.
3. While $j < p$
   1. Update θ by running the training algorithm for n steps
   2. $i = i + n$
   3. Compute the error $\mathrm{Err}(\theta)$ on the validation set
   4. If $\mathrm{Err}(\theta) < \mathrm{Err}^\star$
      - ★ $\theta^\star = \theta$
      - ★ $\mathrm{Err}^\star = \mathrm{Err}(\theta)$
      - ★ $j = 0$

      else $j = j + 1$.
4. Return $\theta^\star$ and the overall number of steps $i^\star = i - np$.

How to leverage on early stopping?

First idea: use early stopping to determine the best number of iterations $i^\star$ and train on the whole data set for $i^\star$ iterations.

Let $X^{(train)}, y^{(train)}$ be the training set.

- Split $X^{(train)}, y^{(train)}$ into $X^{(subtrain)}, y^{(subtrain)}$ and $X^{(valid)}, y^{(valid)}$.
- Run early stopping algorithm starting from random θ using $X^{(subtrain)}, y^{(subtrain)}$ for training data and $X^{(valid)}, y^{(valid)}$ for validation data. This returns $i^\star$ the optimal number of steps.
- Set θ to random values again.
- Train on $X^{(train)}, y^{(train)}$ for $i^\star$ steps.

How to leverage on early stopping?

Second idea: use early stopping to determine the best parameters and the training error at the best number of iterations. Starting from $\theta^\star$, train on the whole data set until the error matches the previous early stopping error.

Let $X^{(train)}, y^{(train)}$ be the training set.

- Split $X^{(train)}, y^{(train)}$ into $X^{(subtrain)}, y^{(subtrain)}$ and $X^{(valid)}, y^{(valid)}$.
- Run early stopping algorithm starting from random θ using $X^{(subtrain)}, y^{(subtrain)}$ for training data and $X^{(valid)}, y^{(valid)}$ for validation data. This returns the optimal parameters $\theta^\star$.
- Set $\varepsilon = \mathcal{L}(\theta^\star, X^{(subtrain)}, y^{(subtrain)})$.
- While $\mathcal{L}(\theta^\star, X^{(valid)}, y^{(valid)}) > \varepsilon$, train on $X^{(train)}, y^{(train)}$ for n steps.

To go further

- Early stopping is a very old idea
  - ["Three topics in ill-posed problems", Wahba 1987]
  - ["A formal comparison of methods proposed for the numerical solution of first kind integral equations", Anderssen and Prenter 1981]
  - ["Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping", Caruana et al. 2001]
- But also an active area of research
  - ["Adaboost is consistent", Bartlett and Traskin 2007]
  - ["Boosting algorithms as gradient descent", Mason et al. 2000]
  - ["On early stopping in gradient descent learning", Yao et al. 2007]
  - ["Boosting with early stopping: Convergence and consistency", Zhang, Yu, et al. 2005]
  - ["Early stopping for kernel boosting algorithms: A general analysis with localized complexities", Wei et al. 2017]

More on reducing overfitting averaging

- Soft-weight sharing:
  ["Simplifying neural networks by soft weight-sharing", Nowlan and Hinton 1992]
- Model averaging:
  Average over: random initialization, random selection of minibatches, hyperparameters, or outcomes of nondeterministic neural networks.
- Boosting neural networks by incrementally adding neural networks to the ensemble
  ["Training methods for adaptive boosting of neural networks", Schwenk and Bengio 1998]
- Boosting has also been applied interpreting an individual neural network as an ensemble, incrementally adding hidden units to the networks
  ["Convex neural networks", Bengio et al. 2006]

Outline

Pipeline for neural networks

- Step 1: Preprocessing the data (substract mean, divide by standard deviation). More complex if data are images.
- Step 2: Choose the architecture (number of layers, number of nodes per layer)
- Step 3:
  1. First, run the network and see if the loss is reasonable (compare with dumb classifier: uniform for classification, mean for regression)
  2. Add some regularization and check that the error on the training set increases.
  3. On a small portion of data, make sure you can overfit when turning down the regularization.
  4. Find the best learning rate
     1. The error does not change too much $\rightarrow$ learning rate too small
     2. The error explodes, NaN $\rightarrow$ learning rate too high
     3. Find a rough range $[10^{-5}, 10^{-3}]$.

Playing with neural network:

`http://playground.tensorflow.org/`

Outline

Denseness results

["Capabilities of three-layered perceptrons", Irie and Miyake 1988]:

Continuous Neural Networks with one single hidden layer can approximate any function $f : \mathbb{R} \to \mathbb{R}$ absolutely integrable and with bounded variation provided that the activation function itself is absolutely integrable and with bounded variation.

Total variation

The total variation of a function f on an interval $[a, b]$ is

$$V_a^b(f) = \sup_{n; x_1 \leq \ldots \leq x_n} \sum_{i=1}^{n-1} |f(x_{i+1}) - f(x_i)|.$$

If f is differentiable and Riemannian integrable, then $V_a^b(f) = \int_a^b |f'(t)|dt$.

Denseness results

["There exists a neural network that does not make avoidable mistakes", Gallant and White 1988]:
Connexion between a neural network with cosine squasher activation function and the decomposition in Fourier series.

Cosine squasher ϕ:

$$\phi(x) = \begin{cases} 0, & \text{if } x \leq -\pi/2 \\ \frac{\cos(x+3\pi/2)+1}{2}, & \text{if } -\pi/2 < x \leq \pi/2 \\ 1, & \text{if } x > \pi/2 \end{cases}$$

["Approximation by superpositions of a sigmoidal function", Cybenko 1989]:
Neural networks with one single hidden layer and sigmoidal activation function ϕ can approximate any continuous function on the unit cube.

Sigmoidal function

A function ϕ is sigmoidal if it satisfies

- $\lim_{x \to -\infty} \phi(x) = 0$
- $\lim_{x \to \infty} \phi(x) = 1$

Denseness results

["Multilayer feedforward networks are universal approximators", Hornik et al. 1989]:
Neural networks with one single hidden layer and any squashing function ϕ can approximate any measurable function.

Squashing function

A function ϕ is squashing if it is non-decreasing and satisfies

- $\lim_{x \to -\infty} \phi(x) = -1$
- $\lim_{x \to \infty} \phi(x) = 1$

["Approximation capabilities of multilayer feedforward networks", Hornik 1991]:

Neural Networks with a single hidden layer and any bounded and non constant activation function can approximate any function in L^p, provided a sufficient number of hidden units.

In addition, if the activation function is continuous, such neural networks can approximate continuous functions uniformly on compacts.

Denseness results

["Theory of the backpropagation neural network", Hecht-Nielsen 1992]:

Neural Networks with one single hidden layer can approximate any function $f \in L^2(R^n \rightarrow R^m)$.

["Multilayer feedforward networks with a non-polynomial activation function can approximate any function", Leshno et al. 1992]:

Neural networks with one single hidden layer and with locally bounded piecewise continuous activation function can approximate any continuous function if and only if the network's activation function is non polynomial.

["Some new results on neural network approximation", Hornik 1993]:

Neural networks with a single hidden layer can uniformly approximate continuous function on compacts if activation function is locally Riemann integrable and nonpolynomial.

Second result

Let B be a bounded set in $\mathbb{R}^d$ containing $x = 0$. Let Γ_B be the set of functions f on B such that, there exists a complex-valued measure $\tilde{F}$ satisfying $\int |w||\tilde{F}|(dw) < \infty$ such that, for all $x \in B$

$$f(x) = f(0) + \int (e^{i\langle w,x\rangle} - 1)\tilde{F}(dw). \quad (1)$$

For all $C > 0$, let $\Gamma_{C,B}$ be the set of all functions f in Γ_B such that, for some $\tilde{F}$ satisfying (1),

$$\int \sup_{x\in B} |\langle w, x\rangle||\tilde{F}|(dw) \leq C.$$

Second result

Let

$$f_n(x) = \sum_{k=1}^{n} c_k \phi(\langle a_k, x_k \rangle + b_k) + c_0.$$

Theorem Barron 1993

For every function $f \in \Gamma_{B,C}$, every sigmoidal function ϕ, ervery probability measure μ, and every $n \geq 1$, there exists a linear combination of sigmoidal functions $f_n(x)$ such that

$$\int_B (f(x) - f_n(x))^2 \mu(dx) \leq \frac{(2C)^2}{n}.$$

The coefficients may be restricted to satisfy $c_0 = f(0)$ and

$$\sum_{k=1}^{n} |c_k| \leq 2C.$$

General framework

Assumption on data

$\mathcal{D}_n = \{(X_i, Y_i), i = 1, \ldots, n\}$ is composed of $i.i.d.$ pairs (X_i, Y_i) where $X \in [-1, 1]^d$ and $Y \in [a, b]$.

Define the regression function f as $f(x) = \mathbb{E}[Y|X = x]$.

Since the function f is bounded, we will consider the clip version $\bar{f}_n$ of an estimate f_n defined as

$$\bar{f}_n(x, \theta) = clip(f_n(x, \theta)) = \begin{cases} f_n(x, \theta), & \text{if } f_n(x, \theta) \in [a, b] \\ a, & \text{if } f_n(x, \theta) < a \\ b, & \text{if } f_n(x, \theta) > b \end{cases}$$

Distance between sigmoid function ϕ_τ and sign

$$dist(\phi_\tau, \text{sign}) = \inf_{0<\varepsilon\leq 1/2} (2\varepsilon + \sup_{|z|\geq\varepsilon} |\phi(\tau z) - \text{sign}(z)|).$$

General framework

Define the parameter space $\Theta_{n,\tau,C} \subset \mathbb{R}^{n(d+2)+1}$ by the set of a_k, b_k, c_k, c_0 for $k = 1, \dots, n$ satisfying

$$|a_k|_1 \leq \tau, \quad |b_k| \leq \tau, \sum_{i=1}^{n} |c_k| \leq C, c_0 \in [a - C, b + C].$$

Let $\Theta_{n,\varepsilon,\tau,C}$ be a finite ε-covering of $\Theta_{n,\tau,C}$ in the sense that for all $\theta \in \Theta_{n,\tau,C}$, there exists $\theta^\star \in \Theta_{n,\varepsilon,\tau,C}$ such that for $k = 1, \dots, n$

$$|a_k - a_k^\star|_1 \leq \varepsilon, \quad |b_k - b_k^\star| \leq \varepsilon,$$

$$\sum_{k=1}^{n} |c_k - c_k^\star| \leq C\varepsilon, \quad |c_0 - c_0^\star| \leq C\varepsilon.$$

General framework

Let

$$f_n(x) = \sum_{k=1}^{n} c_k \phi(\langle a_k, x_k \rangle + b_k) + c_0.$$

Recall that $\bar{f}_n(x, \theta) = clip(f_n(x, \theta))$ and let

$$\hat{\theta}_{n,N,C,\varepsilon} = \underset{\theta \in \Theta_{n,\varepsilon,\tau_n,C}}{\text{argmin}} \left(\frac{1}{N} \sum_{i=1}^{N} (Y_i - \bar{f}_n(X_i, \theta))^2 \right).$$

Let $\hat{f}_{n,N,C,\varepsilon}$ be defined as

$$\hat{f}_{n,N,C,\varepsilon}(x) = \bar{f}_n(x, \hat{\theta}_{n,N,C})$$

General framework

Assume that

- $$\log \textbf{Card}(\Theta_{n,\varepsilon,\tau_n,C}) = O(nd \log(CndN)),$$
- $$\varepsilon = O(n^{-1/2}),$$
- $$dist(\phi_{\tau_n}, \text{sign}) \leq 1/\sqrt{n}.$$

Theorem Barron 1994

Assume that the target function f satisfies

$$C_f = \int |w|_1 |\tilde{F}(w)| dw \leq C,$$

There exists ε such that

$$\mathbb{E}\|f - \hat{f}_{n,N,C,\varepsilon}\|^2 \leq O\Big(\frac{C^2}{n}\Big) + O\Big(\frac{nd}{N} \log N\Big),$$

which is of order

$$O\Big(C\Big(\frac{d \log N}{N}\Big)^{1/2}\Big)$$

for $n \sim C(N/(d \log N))^{1/2}$


Mark A Aizerman. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and remote control* 25 (1964), pp. 821–837.


RS Anderssen and PM Prenter. "A formal comparison of methods proposed for the numerical solution of first kind integral equations". In: *The ANZIAM Journal* 22.4 (1981), pp. 488–500.


Andrew R Barron. "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information theory* 39.3 (1993), pp. 930–945.


Andrew R Barron. "Approximation and estimation bounds for artificial neural networks". In: *Machine learning* 14.1 (1994), pp. 115–133.


Yoshua Bengio et al. "Convex neural networks". In: *Advances in neural information processing systems*. 2006, pp. 123–130.


Hans-Dieter Block. "The perceptron: A model for brain functioning. i". In: *Reviews of Modern Physics* 34.1 (1962), p. 123.


John S Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition". In: *Neurocomputing*. Springer, 1990, pp. 227–236.


Widrow Bernard and D Stearns Samuel. "Adaptive signal processing". In: *Englewood Cliffs, NJ, Prentice-Hall, Inc* 1 (1985), p. 491.


Peter L Bartlett and Mikhail Traskin. "Adaboost is consistent". In: *Journal of Machine Learning Research* 8.Oct (2007), pp. 2347–2368.


Rich Caruana, Steve Lawrence, and C Lee Giles. "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping". In: *Advances in neural information processing systems*. 2001, pp. 402–408.


Meng Cai, Yongzhe Shi, and Jia Liu. "Deep maxout neural networks for speech recognition". In: *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE. 2013, pp. 291–296.


Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).


George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.


Yoav Freund and Robert E Schapire. "Large margin classification using the perceptron algorithm". In: *Machine learning* 37.3 (1999), pp. 277–296.


Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.


Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.


Ian J Goodfellow, Mehdi Mirza, et al. “An empirical investigation of catastrophic forgetting in gradient-based neural networks”. In: *arXiv preprint arXiv:1312.6211* (2013).


Ian J Goodfellow, David Warde-Farley, et al. “Maxout networks”. In: *arXiv preprint arXiv:1302.4389* (2013).


Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850* (2013).


A Ronald Gallant and Halbert White. “There exists a neural network that does not make avoidable mistakes”. In: *Proceedings of the Second Annual IEEE Conference on Neural Networks, San Diego, CA, I*. 1988.

Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

Robert Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

Geoffrey E Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580* (2012).

Arthur E Hoerl and Robert W Kennard. “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

Kurt Hornik. “Some new results on neural network approximation”. In: *Neural networks* 6.8 (1993), pp. 1069–1072.


Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.


Michael Jen-Chao Hu. "Application of the adaline system to weather forecasting". PhD thesis. Department of Electrical Engineering, Stanford University, 1964.


Bunpei Irie and Sei Miyake. "Capabilities of three-layered perceptrons". In: *IEEE International Conference on Neural Networks*. Vol. 1. 641-648. 1988, p. 218.


Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).


Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. "What is the best multi-stage architecture for object recognition?" In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 2146–2153.


Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.


Yann LeCun et al. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.


Moshe Leshno et al. “Multilayer feedforward networks with a non-polynomial activation function can approximate any function”. In: (1992).


Llew Mason et al. “Boosting algorithms as gradient descent”. In: *Advances in neural information processing systems*. 2000, pp. 512–518.


Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.


Dmytro Mishkin and Jiri Matas. “All you need is a good init”. In: *arXiv preprint arXiv:1511.06422* (2015).


Marvin Minsky and Seymour A Papert. *Perceptrons: an introduction to computational geometry*. MIT press, 2017.


Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.


Mehryar Mohri and Afshin Rostamizadeh. “Perceptron mistake bounds”. In: *arXiv preprint arXiv:1305.0208* (2013).


Steven J Nowlan and Geoffrey E Hinton. "Simplifying neural networks by soft weight-sharing". In: *Neural computation* 4.4 (1992), pp. 473–493.


Albert B Novikoff. *On convergence proofs for perceptrons*. Tech. rep. STANFORD RESEARCH INST MENLO PARK CA, 1963.


Mikel Olazaran. "A sociological study of the official history of the perceptrons controversy". In: *Social Studies of Science* 26.3 (1996), pp. 611–659.


Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.


Holger Schwenk and Yoshua Bengio. "Training methods for adaptive boosting of neural networks". In: *Advances in neural information processing systems*. 1998, pp. 647–653.


Michael Schuster. "On supervised learning from sequential data with applications for speech recognition". In: *Daktaro disertacija, Nara Institute of Science and Technology* 45 (1999).


Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.


LR Talbert, GF Groner, and JS Koford. “Real-Time Adaptive Speech-Recognition System”. In: *The Journal of the Acoustical Society of America* 35.5 (1963), pp. 807–807.


Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288.


Grace Wahba. “Three topics in ill-posed problems”. In: *Inverse and ill-posed problems*. Elsevier, 1987, pp. 37–51.


Bernard Widrow and Marcian E Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.


Wessel N van Wieringen. “Lecture notes on ridge regression”. In: *arXiv preprint arXiv:1509.09169* (2015).


Sida Wang and Christopher Manning. “Fast dropout training”. In: *international conference on machine learning*. 2013, pp. 118–126.


Capt Rodney Winter and B Widrow. “Madaline Rule II: a training algorithm for neural networks”. In: *Second Annual International Conference on Neural Networks*. 1988, pp. 1–401.


Yuting Wei, Fanny Yang, and Martin J Wainwright. "Early stopping for kernel boosting algorithms: A general analysis with localized complexities". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6067–6077.


Bing Xu et al. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).


Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. "On early stopping in gradient descent learning". In: *Constructive Approximation* 26.2 (2007), pp. 289–315.


Tong Zhang, Bin Yu, et al. "Boosting with early stopping: Convergence and consistency". In: *The Annals of Statistics* 33.4 (2005), pp. 1538–1579.


Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320.