Samy Bengio

IDIAP Research Institute, Martigny, Switzerland, and
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
bengio@idiap.ch
http://www.idiap.ch/~bengio

January 17, 2006
1. Generalities
2. Multi Layer Perceptrons
3. Gradient Descent
4. Classification
5. Tricks of the Trade
An ANN is a set of units (neurons) connected to each other
Each unit may have multiple inputs but have one output
Each unit performs 2 functions:
  - integration: $s = f(x; \theta)$
  - transfer: $y = g(s)$
Artificial Neural Networks: Functions

- Example of integration function: \( s = \theta_0 + \sum_i x_i \cdot \theta_i \)

- Examples of transfer functions:
  - tanh: \( y = \tanh(s) \)
  - sigmoid: \( y = \frac{1}{1 + \exp(-s)} \)

- Some units receive **inputs** from the outside world.
- Some units generate **outputs** to the outside world.
- The other units are often named **hidden**.
- Hence, from the outside, an ANN can be viewed as a **function**.
- There are various forms of ANNs. The most popular is the Multi Layer Perceptron (MLP).
Transfer Functions (Graphical View)

\[ y = \tanh(x) \]

\[ y = \text{sigmoid}(x) \]

\[ y = x \]
Multi Layer Perceptrons

- An MLP is a function: $\hat{y} = \text{MLP}(x; \theta)$
- The parameters $\theta = \{w_{i,j}^l, b_i^l : \forall i, j, l\}$
- From now on, let $x_i(p)$ be the $i^{th}$ value in the $p^{th}$ example represented by vector $x(p)$ (and when possible, let us drop $p$).
- Each layer $l$ ($1 \leq l \leq M$) is fully connected to the previous layer
  - **Integration**: $s_i^l = b_i^l + \sum_j y_j^{l-1} \cdot w_{i,j}^l$
  - **Transfer**: $y_i^l = \tanh(s_i^l)$ or $y_i^l = \text{sigmoid}(s_i^l)$ or $y_i^l = s_i^l$
  - The output of the zeroth layer contains the inputs $y_i^0 = x_i$
  - The output of the last layer $M$ contains the outputs $\hat{y}_i = y_i^M$
An MLP can approximate any continuous functions.

However, it needs to have at least 1 hidden layer (sometimes easier with 2), and enough units in each layer.

Moreover, we have to find the correct value of the parameters $\theta$.

This is an NP-complete problem!!!!

How can we find these parameters?

Answer: optimize a given criterion using a gradient method.

Note: capacity controlled by the number of parameters.
Separability

- Linear
- Linear + Sigmoid
- Linear + Sigmoid + Linear
- Linear + Sigmoid + Linear + Sigmoid + Linear
1. Generalities
2. Multi Layer Perceptrons
3. Gradient Descent
4. Classification
5. Tricks of the Trade
Gradient Descent

Objective: minimize a criterion $C$ over a set of data $D_n$:

$$C(D_n, \theta) = \sum_{p=1}^{n} L(y(p), \hat{y}(p))$$

where

$$\hat{y}(p) = \text{MLP}(x(p); \theta)$$

We are searching for the best parameters $\theta$:

$$\theta^* = \arg \min_{\theta} C(D_n, \theta)$$

Gradient descent: an iterative procedure where, at each iteration $s$ we modify the parameters $\theta$:

$$\theta^{s+1} = \theta^s - \eta \frac{\partial C(D_n, \theta^s)}{\partial \theta^s}$$

where $\eta$ is the learning rate. WARNING: local optima.
Gradient Descent: The Basics

Chain Rule

- If \( a = f(b) \) and \( b = g(c) \)
- Then \( \frac{\partial a}{\partial c} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c} = f'(b) \cdot g'(c) \)
Gradient Descent: The Basics

Sum Rule

- if \( a = f(b, c) \) and \( b = g(d) \) and \( c = h(d) \)

- then
  \[
  \frac{\partial a}{\partial d} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial d} + \frac{\partial a}{\partial c} \cdot \frac{\partial c}{\partial d}
  \]

  \[
  \frac{\partial a}{\partial d} = \frac{\partial f(b, c)}{\partial b} \cdot g'(d) + \frac{\partial f(b, c)}{\partial c} \cdot h'(d)
  \]
Gradient Descent Basics (Graphical View)
Gradient Descent: Criterion

First: we need to pass the gradient through the criterion.

The global criterion $C$ is:

$$C(D_n, \theta) = \sum_{p=1}^{n} L(y(p), \hat{y}(p))$$

Example: the mean squared error criterion (MSE):

$$L(y, \hat{y}) = \sum_{i=1}^{d} \frac{1}{2}(y_i - \hat{y}_i)^2$$

And the derivative with respect to the output $\hat{y}_i$:

$$\frac{\partial L(y, \hat{y})}{\partial \hat{y}_i} = \hat{y}_i - y_i$$
Gradient Descent: Last Layer

- **Second**: derivative wrt the parameters of the last layer $M$

\[
\hat{y}_i = y^M_i = \tanh(s^M_i)
\]

\[
s^M_i = b^M_i + \sum_j y^{M-1}_j \cdot w^M_{i,j}
\]

Hence the derivative with respect to $w^M_{i,j}$ is:

\[
\frac{\partial \hat{y}_i}{\partial w^M_{i,j}} = \frac{\partial \hat{y}_i}{\partial s^M_i} \cdot \frac{\partial s^M_i}{\partial w^M_{i,j}} = (1 - (y^M_i)^2) \cdot y^{M-1}_j
\]

And the derivative with respect to $b^M_i$ is:

\[
\frac{\partial \hat{y}_i}{\partial b^M_i} = \frac{\partial \hat{y}_i}{\partial s^M_i} \cdot \frac{\partial s^M_i}{\partial b^M_i} = (1 - (y^M_i)^2) \cdot 1
\]
Gradient Descent: Other Layers

- **Third**: derivative wrt to the output of a hidden layer $y^l_j$

\[
\frac{\partial \hat{y}_i}{\partial y^l_j} = \sum_k \frac{\partial \hat{y}_i}{\partial y^{l+1}_k} \cdot \frac{\partial y^{l+1}_k}{\partial y^l_j}
\]

where

\[
\frac{\partial y^{l+1}_k}{\partial y^l_j} = \frac{\partial y^{l+1}_k}{\partial s^{l+1}_k} \cdot \frac{\partial s^{l+1}_k}{\partial y^l_j} = (1 - (y^{l+1}_k)^2) \cdot w^{l+1}_{k,j}
\]

and

\[
\frac{\partial \hat{y}_i}{\partial y^M_i} = 1 \text{ and } \frac{\partial \hat{y}_i}{\partial y^M_{k \neq i}} = 0
\]
Gradient Descent: Other Parameters

- **Fourth**: derivative wrt the parameters of hidden layer $y_j^l$

\[
\frac{\partial \hat{y}_i}{\partial w_{j,k}^l} = \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{j,k}^l} = \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot y_k^{l-1}
\]

and

\[
\frac{\partial \hat{y}_i}{\partial b_j^l} = \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial b_j^l} = \frac{\partial \hat{y}_i}{\partial y_j^l} \cdot 1
\]
Gradient Descent: Global Algorithm

For each iteration

1. Initialize gradients $\frac{\partial C}{\partial \theta_i} = 0$ for each $\theta_i$
2. For each example $z(p) = (x(p), y(p))$
   1. Forward phase: compute $\hat{y}(p) = \text{MLP}(x(p), \theta)$
   2. Compute $\frac{\partial L(y(p), \hat{y}(p))}{\partial \hat{y}(p)}$
3. For each layer $l$ from $M$ to 1:
   1. Compute $\frac{\partial \hat{y}(p)}{\partial y_j^l}$
   2. Compute $\frac{\partial y_j^l}{\partial b_j^l}$ and $\frac{\partial y_j^l}{\partial w_{j,k}^l}$
   3. Accumulate gradients:
      
      \[
      \frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial b_j^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial b_j^l}
      \]
      \[
      \frac{\partial C}{\partial w_{j,k}^l} = \frac{\partial C}{\partial w_{j,k}^l} + \frac{\partial C}{\partial L} \cdot \frac{\partial L}{\partial \hat{y}(p)} \cdot \frac{\partial \hat{y}(p)}{\partial y_j^l} \cdot \frac{\partial y_j^l}{\partial w_{j,k}^l}
      \]
3. Update the parameters: $\theta_i^{s+1} = \theta_i^s - \eta \cdot \frac{\partial C}{\partial \theta_i^s}$
Gradient Descent: An Example (1)

Let us start with a simple MLP:

```
0.7 0.6
-0.3
1.2
1.1
linear
tanh
tanh
0.3
2.3
-0.6
1.3
-0.3
0.7
0.6
1.1
1.3
```

Samy Bengio
Statistical Machine Learning from Data
Gradient Descent: An Example (2)

We forward one example and compute its MSE:
We backpropagate the gradient everywhere:
Gradient Descent: An Example (4)

We modify each parameter with learning rate 0.1:

We modify each parameter with learning rate 0.1:
Gradient Descent: An Example (5)

We forward the same example and compute its (smaller) MSE:

\[
\text{MSE} = 0.27
\]
MLP are Universal Approximators

- It can be shown that, under reasonable assumptions, one can approximate any smooth function with an MLP with one layer of hidden units.
- First intuition:
  - Let us consider a classification task
  - Let us consider hard transfert functions for hidden units:
    \[ y = \text{step}(s) = \begin{cases} 
    1 & \text{if } s > 0 \\
    0 & \text{otherwise}
    \end{cases} \]
  - Let us consider linear transfert functions for output units:
    \[ y = s \]
  - First attempt:
    \[ \hat{y} = c + \sum_{i=1}^{N} v_i \cdot \text{sign} \left( \sum_{j=1}^{M} x_j \cdot w_{i,j} + b_i \right) \]
Illustration: Universal Approximators
Illustration: Universal Approximators
Illustration: Universal Approximators
Illustration: Universal Approximators
Illustration: Universal Approximators

... but what about that?
Let us consider simple functions of two variables $y(x_1, x_2)$.

Fourier decomposition:

$$y(x_1, x_2) \approx \sum_s A_s(x_1) \cos(sx_2)$$

where coefficients of $A_s$ are functions of $x_1$.

Further Fourier decomposition:

$$y(x_1, x_2) \approx \sum_s \sum_l A_{s,l} \cos(lx_1) \cos(sx_2)$$

We know that $\cos(\alpha) \cos(\beta) = \frac{1}{2} \cos(\alpha + \beta) + \frac{1}{2} \cos(\alpha - \beta)$:

$$y(x_1, x_2) \approx \sum_s \sum_l A_{s,l} \left[ \frac{1}{2} \cos(lx_1 + sx_2) + \frac{1}{2} \cos(lx_1 - sx_2) \right]$$
Universal Approximation by Cosines

- The cos function can be approximated with linear combinations of step functions:

\[ f(z) = f_0 + \sum_i (f_{i+1} - f_i) \text{step}(z - z_i) \]

- So \( y(x_1, x_2) \) can be approximated by a linear combination of step functions whose arguments are linear combinations of \( x_1 \) and \( x_2 \), and which can be approximated by tanh functions.
1 Generalities

2 Multi Layer Perceptrons

3 Gradient Descent

4 Classification

5 Tricks of the Trade
ANN for Binary Classification

- One output with target coded as \{-1, 1\} or \{0, 1\} depending on the last layer output function (linear, sigmoid, tanh, ...)
- For a given output, the associated class corresponds to the nearest target.
- How to obtain class posterior probabilities:
  - use a sigmoid with targets \{0, 1\}
  - if the model is correctly trained (with, for instance MSE criterion)

\[ \hat{y}(x) = E[Y|X = x] = 1 \cdot P(Y = 1|X = x) + 0 \cdot P(Y = 0|X = x) \]

- the output will thus encode \( P(Y = 1|X = x) \)
- Note: we do not optimize directly the classification error...
ANN for Multiclass Classification

- Simplest solution: **one-hot encoding**
  - One output per class, coded for instance as (0, · · · , 1, · · · , 0)
  - For a given output, the associated class corresponds to the index of the maximum value in the output vector
  - How to obtain class posterior probabilities:
    - use a softmax: \( \hat{y}_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)} \)
    - each output \( i \) will encode \( P(Y = i|X = x) \)

- Otherwise: each class corresponds to a different **binary code**
  - For example for a 4-class problem, we could have an 8-dim code for each class
  - For a given output, the associated class corresponds to the nearest code (according to a given distance)
  - Example: Error Correcting Output Codes (ECOC)
Let us represent a 4-class problem with 6 bits:

- class 1: 1 1 0 0 0 1
- class 2: 1 0 0 0 1 0
- class 3: 0 1 0 1 0 0
- class 4: 0 0 1 0 0 0

We then create 6 classifiers (or 1 classifier with 6 outputs)
For example: the first classifier will try to separate classes 1 and 2 from classes 3 and 4
Error Correcting Output Codes

- Given our 4-class problem represented with 6 bits:
  - class 1: 1 1 0 0 0 1
  - class 2: 1 0 0 0 1 0
  - class 3: 0 1 0 1 0 0
  - class 4: 0 0 1 0 0 0

- When a new example comes, we compute the distance between the code obtained by the 6 classifiers and the 4 classes:
  - obtained: 0 1 1 1 1 0
  - distances: (let us use Manhattan distance)
    - to class 1: 5
    - to class 2: 4
    - to class 3: 2
    - to class 4: 3
What is a Good Error Correcting Output Code

- How to devise a good error correcting output code?
- Maximize the *minimum Hamming distance* between any pair of code words.
- A good ECOC should satisfy two properties:
  - **Row separation.** (Hamming distance)
  - **Column separation.** Column functions should be as uncorrelated as possible with each other.
1 Generalities

2 Multi Layer Perceptrons

3 Gradient Descent

4 Classification

5 Tricks of the Trade
Tricks of the Trade

A good book to make ANNs working


Content:
- Stochastic Gradient
- Initialization
- Learning Rate and Learning Rate Decay
- Weight Decay
The gradient descent technique is **batch**:
- First accumulate the gradient from all examples, then adjust the parameters
- What if the data set is very big, and contains redundencies?

**Other solution:** **stochastic** gradient descent
- Adjust the parameters after each example instead
- Stochastic: we approximate the full gradient with its estimate at each example
- Nevertheless, convergence proofs exist for such method.
- Moreover: **much faster** for large data sets!!!

**Other gradient techniques:** second order methods such as **conjugate gradient**: good for small data sets
How should we initialize the parameters of an ANN?

One common problem: **saturation**

When the weighted sum is big, the output of the tanh (or sigmoid) saturates, and the gradient tends towards 0.
Hence, we should initialize the parameters such that the average weighted sum is in the linear part of the transfer function:

- See Leon Bottou’s thesis for details

- input data: normalized with zero mean and unit variance,
- targets:
  - regression: normalized with zero mean and unit variance,
  - classification:
    - output transfer function is tanh: 0.6 and -0.6
    - output transfer function is sigmoid: 0.8 and 0.2
    - output transfer function is linear: 0.6 and -0.6

- parameters: uniformly distributed in $\left[ \frac{-1}{\sqrt{\text{fan in}}}, \frac{1}{\sqrt{\text{fan in}}} \right]$
Learning Rate and Learning Rate Decay

- How to select the learning rate $\eta$?
- If $\eta$ is too big: the optimization diverges
- If $\eta$ is too small: the optimization is very slow and may be stuck into local minima
- One solution: **progressive decay**
  - initial learning rate $\eta = \eta_0$
  - learning rate decay $\eta_d$
  - At each iteration $s$:

$$
\eta(s) = \frac{\eta_0}{(1 + s \cdot \eta_d)}
$$
Learning Rate Decay (Graphical View)

The graph shows the function $\frac{1}{1 + 0.1 \times x}$, which is used to illustrate the decay of the learning rate with respect to the iteration step $x$. The y-axis represents the value of the function, while the x-axis represents the iteration steps from 0 to 100.
Weight Decay

- One way to control the capacity: **regularization**
- For MLPs, when the weights tend to 0, sigmoid or tanh functions are **almost linear**, hence with low capacity
- **Weight decay**: penalize solutions with high weights and bias (in amplitude)

$$C(D_n, \theta) = \sum_{p=1}^{n} L(y(p), \hat{y}(p)) + \frac{\beta}{2} \sum_{j=1}^{|\theta|} \theta_j^2$$

where $\beta$ controls the weight decay.

- Easy to implement:

$$\theta_j^{s+1} = \theta_j^s - \sum_{p=1}^{n} \eta \frac{\partial L(y(p), \hat{y}(p))}{\partial \theta_j^s} - \eta \cdot \beta \cdot \theta_j^s$$
Examples of Training Criteria

- **Mean-squared error, for regression:**
  \[ L(y, \hat{y}) = \sum_{i=1}^{d} \frac{1}{2}(y_i - \hat{y}_i)^2 \]

- **Cross-entropy criterion, for classification (targets ∈ \{-1, 1\}):**
  \[ L(y, \hat{y}) = \log(1 + \exp(-y_i\hat{y}_i)) \]
  
  **Hard version:**
  \[ L(y, \hat{y}) = \|1 - y_i\hat{y}_i\|_+ \]
Examples of Training Criteria

- $\log(1 + \exp(-x))$
- $|1 - x| + 0.5x^2$
- $0.5x^2$