AI Notes

AJ.

March 22, 2017

1 Layered networks and backpropagation

1.1 Setup of the learning system

The example we will use in this note is image manipulation, including image classification, pattern recognition, etc.. For that we use machine learning algorithms, in particular neural networks. Present day machine learning algorithms are functions that map

$$f : \mathcal{W} \times \mathcal{S} \to \mathcal{F}, \tag{1}$$

where $\mathcal{S}$ denotes the space of the orginial images (source), $\mathcal{F}$ stands for the space of the result of the image manipulation (final result), and $\mathcal{W}$ denotes the space of parameters that can be tuned to change the functionality of the map.

The source can be choosen to be $\boldsymbol{P}^d$, where the source image has d independent unit (for an image with n rows and m columns $d = nm$), and the pixel information is an element of $\boldsymbol{P}$. Usually $\boldsymbol{P}$ can be taken isomorphic to either $\boldsymbol{R}^3$ if it consists of three colors, and the color information is continuous; or it is isomorphic to $[0...255]^3$ if the color information is an 8-bit integer.

The space of the result of image manipulation $\mathcal{F}$ can be an image, too, ie. isomorphic to $\mathbf{P}^{d'}$ where $d' = n'm'$ if the final image has n' rows and m' columns; but it can be an abstract space of notions wich is $\boldsymbol{N}^n$ where n is the number of the notions, and an integer number characterizes its weigth, ie. the probability that it describes correctly the image.

The parameter space is $\mathcal{W} \sim \boldsymbol{R}^k$, where there are k parameters, each having a real number value. The parameters should be tuned to fix the functionality of the network, but neither of the parameters have initial preferences.

1.2 Layered networks

Present day neural networks are organized in layers. That means that the full f function is built up as composition from several functions. Each of these composition functions are of the form:

$$f_k : \mathcal{W}_k \times \mathcal{S}_k \to \mathcal{A}_k, \tag{2}$$

where $\mathcal{S}_k$ is the space of the inputs, $\mathcal{A}_k$ is the space of the outputs (axons), and $\mathcal{W}$ is the parameter space. We will denote the elements of $\mathcal{W}_k$ and $\mathcal{A}_k$ as

$$w_k \in \mathcal{W}_k, \qquad A_k \in \mathcal{A}_k. \tag{3}$$

or in component notation

$$w_{ki} \in \mathcal{W}_k, \qquad A_{ka} \in \mathcal{A}_k. \tag{4}$$

The complete f function is a tree-like function composition, meaning that the input of the layer at level k can be any of the outputs of layers $k' < k$. That means $\mathcal{S}_k = \cup_{\ell<k}\mathcal{A}_\ell$, or

$$A_k = f_k(w, A_{k-1}, A_{k-2}, \dots A_0), \tag{5}$$

where $A_0 \equiv S$ is the original image, and the output of the complete network is $F = A_n$.

1.3 Learning as optimization

The learning mechanism is tuning the parameters in a way that the system provides an output closest to the expected one. The difference between the desired and the actual result can be characterized by some scalar number(s), for example a χ^2 value. In general we have an evaluation step that compares A_n with an optimal output and provides some error indicators. Let us call this function L (loss or cost function):

$$L : \mathcal{F} \times \mathcal{F} \to \mathcal{V}, \qquad L(F, \bar{F}) \in \mathcal{V}, \tag{6}$$

where F is the actual output, $\bar{F}$ is the desired output, $\mathcal{V}$ is the space of loss (error) indicators. Learning means to minimize each element of the error function L by tuning the parameters of the network.

Minimization can be performed by different methods, one of the most simple one is the steepest descent method. For that we consider one element of L (ie. we assume that $\mathcal{V}$ is one dimensional), and take it as a function of the parameters of the network. We will modify the parameters of the network along the gradient of the L surface:

$$\delta w = \alpha \nabla L. \tag{7}$$

Since in the linear approximation

$$L(w - \delta w) \approx L(w) - \delta w \nabla L = L(w) - \gamma \nabla L^2 < L(w), \tag{8}$$

we can decrease the error as long as the linear approximation remains true. In practice ∇L has lots of small elements, then we gather the gradients from several input-output pairs.

1.4 Computing the gradient: automatic derivation

For the computation of the gradient we use the hierarchical setup of the network (5) to write

$$dL = \frac{\partial L}{\partial A_n} dA_n, \qquad dA_k = \frac{\partial A_k}{\partial w_k} dw_k + \sum_{\ell=0}^{k-1} \frac{\partial A_k}{\partial A_\ell} dA_\ell. \tag{9}$$

We find that $A_{k<\ell}$ does not depend on w_ℓ. Therefore

$$\frac{\partial A_k}{\partial w_\ell} = \delta^{k\ell} \frac{\partial A_\ell}{\partial w_\ell} + \sum_{\ell_1=\ell}^{k-1} \frac{\partial A_k}{\partial A_{\ell_1}} \frac{\partial A_{\ell_1}}{\partial w^\ell}. \tag{10}$$

This means that the range of the sum becomes smaller by one at each recursion step, and it will terminate at some point, but at the end all terms are proportional to $\partial A_\ell / \partial w_\ell$. The explicit result is

$$\frac{\partial A_k}{\partial w_\ell} = U^{k\ell} \frac{\partial A_\ell}{\partial w_\ell}, \tag{11}$$

where

$$U^{k\ell} = \delta^{k\ell} + \frac{\partial A_k}{\partial A_\ell} + \sum_{\ell < \ell_1 < k} \frac{\partial A_k}{\partial A_{\ell_1}} \frac{\partial A_{\ell_1}}{\partial A_\ell} + \sum_{\ell < \ell_2 < \ell_1 < k} \frac{\partial A_k}{\partial A_{\ell_1}} \frac{\partial A_{\ell_1}}{\partial A_{\ell_2}} \frac{\partial A_{\ell_2}}{\partial A_\ell} + \cdots + \frac{\partial A_k}{\partial A_{k-1}} \frac{\partial A_{k-1}}{\partial A_{k-2}} \cdots \frac{\partial A_{\ell+1}}{\partial A_\ell}. \tag{12}$$

The desired gradient is finally

$$\frac{\partial L}{\partial w_\ell} = \frac{\partial L}{\partial A_n} U^{n\ell} \frac{\partial A_\ell}{\partial w_\ell}. \tag{13}$$

The task is to compute $U^{n\ell}$. There are two simple recursion setups: forward and backward propagation.

1.4.1 Backpropagation

For backpropagation the setup is simpler, since we can directly write up the recursion for $U^{n\ell}$

$$U^{nn} = 1, \qquad U^{n\ell} = \sum_{n > k \geq \ell} U^{nk} \frac{\partial A_k}{\partial A_\ell}. \tag{14}$$

We start with $\ell = n$ with a unit matrix. At each step we decrease ℓ by one, then each terms in the sum is already known, finally we arrive at $\ell = 1$, then all $U^{n\ell}$ is computed.

1.4.2 Forward differentiation

In the forward case we use the recursion expression

$$U^{kk} = 1, \qquad U^{k\ell} = \sum_{k \geq k' > \ell} \frac{\partial A_k}{\partial A_{k'}} U^{k'\ell}. \tag{15}$$

We start with $k = \ell$ with a unit matrix. At each step we increase ℓ by one, finally computing $U^{n\ell}$. Then we start a new recursion to compute $U^{n\ell'}$ for a different ℓ'. As a result we compute all $U^{k\ell}$ matrices which is unnecessary. Only in very special cases can forward propagation be better than backpropagation.

1.4.3 Backpropagation algorithm

For the sake of completeness we write it out backpropagation in component notation, too, suppressing the explicit notation of index summation:

$$U^{nn}_{ab} = \delta_{ab}, \qquad U^{n\ell}_{ab} = \sum_{n > k \geq \ell} U^{nk}_{ac} \frac{\partial A_{kc}}{\partial A_{\ell b}}, \qquad \frac{\partial L}{\partial w_{\ell i}} = \frac{\partial L}{\partial A_{na}} U^{n\ell}_{ab} \frac{\partial A_{\ell b}}{\partial w_{\ell i}}. \tag{16}$$

We can go in this case even a step further, and define a vector

$$J_\ell = \frac{\partial L}{\partial A_n} U^{n\ell}, \tag{17}$$

for which we can set up the recursion

$$J_n = \frac{\partial L}{\partial A_n}, \qquad J_\ell = \sum_{n > k \geq \ell} J_k \frac{\partial A_k}{\partial A_\ell}, \qquad \frac{\partial L}{\partial w_\ell} = J_\ell \frac{\partial A_\ell}{\partial w_\ell}. \tag{18}$$

In the following we will use

$$L = \frac{1}{2} \sum_a \frac{(A_{na} - \bar{A}_{na})^2}{\sigma_a^2}, \tag{19}$$

then

$$J_{na} = \frac{A_{na} - \bar{A}_{na}}{\sigma_a^2}. \tag{20}$$

To perform this recursion we can apply the following algorithm

1. Each k level must have an input slot for adding value to the actual J_k values. As we arrive to the kth level to process it, the J_k vector must be complete.
2. In processing the kth level first we evaluate the gradient vector from J_n, ie. we evaluate

   $$\frac{\partial L}{\partial w_\ell} = J_\ell \frac{\partial A_\ell}{\partial w_\ell}. \tag{21}$$

   Second we list all $\ell < k$ layer depending on the kth layer, and add to their actual J_ℓ vector the contribution

   $$J_\ell \to J_\ell + J_k \frac{\partial A_k}{\partial A_\ell}. \tag{22}$$

3. We start the recursion with filling the input slot of the nth level by J_{na} coming from (20).
4. Process the levels from n to 1 in backward direction.

The process of learning is the following:

1. use the original w parameters to compute the state of the network
2. update the $(\nabla L)_k = \frac{\partial L}{\partial w_k}$ gradient vector at each layer level k with backpropagation
3. go back to step 1 until the gradient is sensible
4. change $w_k \to w_k - \alpha(\nabla L)_k$ with an adequate α value

2 Layer types

In lot of architecture there are parallel layers that process the same input configuration, they do not communicate, and their outputs are processed independently. Then it is worth to pack these layers into a common one having some vector space output.

To the earlier formalism it can be easily adapted, by thinking that the a, b indeices are multi-indices, that is

$$a = (n, m, \alpha), \tag{23}$$

where n, m denotes the 2D position of the object, α indexes the internal space.

Thenll layer hierarchies can be built up from two basic layer types: collection layers and link (link layers). Their properties are the following.

2.1 Collection layer

Its main properties are

- Input: either fixed (source), or sum of outputs of earlier links.
- Performs a nonlinear operation on the inputs to produce the output.
- Does not change geometry.
- Non-tunable, ie. it has a definit functionality.

Therefore the output of the kth collection layer can be computed as

$$A_{ka} = f(\sum_{\ell<k} A_{\ell a}), \tag{24}$$

where f is a nonlinear function, ℓ runs over some of the links. There is now weight here, the backpropagation vector carried over is

$$J_{\ell a} = J_{ka} f'(\sum_{\ell<k} A_{\ell a}). \tag{25}$$

2.2 Link

Its main properties are

- Input: output from one collection layer
- Performs linear operation on the inputs.
- Can change geometry.
- Tunable, the parameters of the linear operation can be changed.
- It is worth to put on GPU.

In basic formula for the output of layer k

$$A_{ka} = \sum_b w_{ab} A_{\ell b} + \bar{w}_a, \tag{26}$$

where ℓ is a single collection layer with $\ell < k$, w and $\bar{w}$ are parameters.

Gradients of the weights

$$\frac{\partial L}{\partial w_{ab}} = \sum_c J_c \frac{\partial A_{kc}}{\partial w_{ab}} = J_a A_{\ell b}, \qquad \frac{\partial L}{\partial \bar{w}_a} = \sum_c J_c \frac{\partial A_{kc}}{\partial \bar{w}_a} = J_a. \tag{27}$$

The carried backpropagation vector:

$$J_{\ell a} = \sum_c J_c \frac{\partial A_{kc}}{\partial A_{\ell a}} = \sum_c J_c w_{cb}. \tag{28}$$

Special types of link layers:

2.2.1 Convolution layer

A specific type of the link layers is convolution layer, where we make a distinction between the spatial (external) and internal indices. In the internal space we still have a full matrix multiplication, but in the external space we just perform a convolution.

So let us denote

$$A_{ka} = A_{ki\alpha}, \tag{29}$$

where k is the layer number, i is the external (multi) index and α is the internal index. Then we have

$$A_{ki\alpha} = \sum_{j\beta} w_{j\alpha\beta} A_{\ell,i+j,\beta} + \bar{w}_\alpha = \sum_{j\beta} w_{j-i,\alpha\beta} A_{\ell j\beta} + \bar{w}_\alpha. \tag{30}$$

Gradients of the weights

$$\frac{\partial L}{\partial w_{i\alpha\beta}} = \sum_{j\gamma} J_{j\gamma} \frac{\partial A_{kj\gamma}}{\partial w_{i\alpha\beta}} = \sum_j J_{j\alpha} A_{\ell,i+j,\beta}, \qquad \frac{\partial L}{\partial \bar{w}_\alpha} = \sum_{j\gamma} J_{j\gamma} \frac{\partial A_{kj\gamma}}{\partial \bar{w}_\alpha} = \sum_j J_{j\alpha}. \tag{31}$$

The carried backpropagation vector:

$$J_{\ell i\alpha} = \sum_{j\gamma} J_{j\gamma} \frac{\partial A_{kj\gamma}}{\partial A_{\ell i\alpha}} = \sum_{j\gamma} J_{j\gamma} w_{i-j,\gamma\alpha}. \tag{32}$$

In case we have a geometry change, then it affects dimensions independently. Let us assume that we start from a dimension of n_{in} sites, and end with a dimension with n_{out} sites. We remark that if the convolution volume is n_{conv}, then the effective starting dimension is $n'_{in} = n_{in} - n_{conv} + 1$. We run over the output sites one by one $i = 0, 1, \dots n_{out} - 1$, and do a convolution starting at i' site in the input layer. If $n'_{in} = n_{out}$ then $i' = i$, otherwise we have to find a map $i'(i)$.

The average step in the input layer is

$$r = \frac{n_{in} - 1}{n_{out} - 1}. \tag{33}$$

We can simply define $i'(i) = [ri]$. Another solution that we concentrate to the center. Then we have to make n_1 times a step $[r]$ and n_2 times a step $[r] + 1$. The formulae determining them read

$$n_1 + n_2 = n_{out}, \qquad [r]n_1 + ([r] + 1)n_2 = n_{in}, \tag{34}$$

which means

$$n_2 = n_{in} - [r]n_{out}, \qquad n_1 = ([r] + 1])n_{out} - n_{in}. \tag{35}$$

We can then arrange the steps that we start with $[n_2/2]$ times $[r] + 1$ long steps, then n_1 times $[r]$ long steps, finally $]n_2/2[$ times $[r] + 1$ long steps.

All in all, we will have an $i'(i)$ transformed multi-index that takes into account the geometry change, too. Then we have

$$A_{ki\alpha} = \sum_{j\beta} w_{j\alpha\beta} A_{\ell,i'+j,\beta} + \bar{w}_\alpha = \sum_{j\beta} w_{j-i',\alpha\beta} A_{\ell j\beta} + \bar{w}_\alpha. \tag{36}$$

Gradients of the weights

$$\frac{\partial L}{\partial w_{i\alpha\beta}} = \sum_{j\gamma} J_{j\gamma} \frac{\partial A_{kj\gamma}}{\partial w_{i\alpha\beta}} = \sum_j J_{j\alpha} A_{\ell,j'+i,\beta}, \qquad \frac{\partial L}{\partial \bar{w}_\alpha} = \sum_{j\gamma} J_{j\gamma} \frac{\partial A_{kj\gamma}}{\partial \bar{w}_\alpha} = \sum_j J_{j\alpha}. \tag{37}$$

The carried backpropagation vector:

$$J_{\ell i\alpha} = \sum_{j\gamma} J_{j\gamma} \frac{\partial A_{kj\gamma}}{\partial A_{\ell i\alpha}} = \sum_{j\gamma} J_{j\gamma} w_{i-j',\gamma\alpha}. \tag{38}$$

This last expression can be easily performed if we define

$$J'_{p\gamma} = \sum_j \delta_{j'p} J_{j\gamma}, \tag{39}$$

that can be built by running through the output indices i, and add $J_{i\gamma}$ to $J'_{i'\gamma}$. Then the backpropagation vector

$$J_{\ell i\alpha} = \sum_{j\gamma} J_{j\gamma} \sum_{p} \delta_{j'p} w_{i-p,\gamma\alpha} = \sum_{p\gamma} J'_{p\gamma} w_{i-p,\gamma\alpha} = \sum_{j\gamma} J'_{i-j,\gamma} w_{j\gamma\alpha}. \tag{40}$$

2.2.2 MaxPooling (downsampling) layers

To change scale it is worth to use pooling. It can be manifested as an averaging, using convolution layer with uniform weights, but it is also possible to realize it as max pooling, where the output is the maximum of some input values:

$$A_{ka} = A_{\ell,b_a}, \qquad \text{where } A_{\ell,b_a} = \max_{|b-a|<R} A_{\ell,b}. \tag{41}$$

Pooling layers do not have internal gradients, and the incoming backpropagation vector J_{ka} is simply copied to that index of the ℓth layer that provided the maximal $A_{\ell,b}$ value:

$$J_{\ell,b} = \begin{cases} J_{ka}, & \text{if} \quad b = b_a \\ 0, & \text{if} \quad b \neq b_a. \end{cases} \tag{42}$$

3 Learning a set

What if we want that our network knows several input-output pairs? Assume we have S_a inputs with corresponding $\bar{F}_a$ outputs for $a = 1 \dots N$, and our network provides a tuneable $F(w, S)$ result for an input S.

What we have to do is to minimize the cumulative los:

$$\chi^2 = \sum_{a=1}^{N} \frac{L(F(w, S_a), \bar{F}_a)}{\sigma_a^2}, \tag{43}$$

where L is the loss function, σ_a characterizes the reliability of the given input-output pair. If the input is damaged, or the output is not certain, we can qualify the corresponding pair as low reliability and weight it with high σ_a. The minimum of the cumulative loss with respect of the weights:

$$\frac{\partial \chi^2}{\partial w_i} = \sum_{a=1}^{N} \frac{1}{\sigma_a^2} \frac{\partial L(F_a, \bar{F}_a)}{\partial w_i}, \tag{44}$$

this latter we have already calculated before. To find the minimum we do a learning step

$$w_i \to w_i - \alpha \frac{\partial \chi^2}{\partial w_i} \tag{45}$$

with some appropriate α.

4 Program details

Here we discuss a possible realization of the network above.

4.1 Vector spaces

- There is no standardized way of treating objects with more indices. In the present realization we represent the geometry of the layers as an int vector

  ```
  class Geometry :  public vector<int>
  ```

  This class contains the N_i dimensions (including both the external and internal spaces), and it provides the "basis vectors": $e_i = \prod_{j<i} N_j$. In addition it stores $maxvalue = \prod_j N_j$.

- To single out an element we use multiindex

  ```
  class MultiIndex :  public vector<int>
  ```

  The vector elements are the indices $0 \leq n_i < N_i$. It contains a reference to the actual Geometry. It also provides the int form of the index $i = \sum_j n_j e_j$. Multiindices can be added, multiplied by an int. Range check should also be indluded.

- It is worth to create a container class that can treat multiindices as indices:

  ```
  class Vector :  public vector<double>
  ```

  The only difference from a normal vector is that the [.] operator is overloaded in order to take a multiindex.

  ```
  class LocalLayer :  public Vector

  class Link :  public Vector
  ```

5 Frequently used loss functions

Here A_a is the output os the last layer, $\bar{F}_a$ is the expected result.

Quadratic loss:

$$L = \frac{1}{2}\sum_a (A_a - \bar{F}_a)^2, \qquad \frac{\partial L}{\partial A_a} = A_a - \bar{F}_a. \tag{46}$$

χ^2 method: σ_a^2 means the error on the given notion, σ^{-2} is proportional the relevance (importance) of the given data:

$$L = \sum_a \frac{(A_a - \bar{F}_a)^2}{2\sigma_a^2}, \qquad \frac{\partial L}{\partial A_a} = \frac{A_a - \bar{F}_a}{\sigma_a^2}. \tag{47}$$

Cross-entropy loss: for output values $\in [0, 1]$

$$L = -\sum_a \left[\bar{F}_a \ln A_a + (1 - \bar{F}_a)\ln(1 - A_a)\right], \qquad \frac{\partial L}{\partial A_a} = -\frac{\bar{F}_a}{A_a} + \frac{1 - \bar{F}_a}{1 - A_a} = \frac{A_a - \bar{F}_a}{A_a(1 - A_a)}. \tag{48}$$

Exponential loss:

$$L = e^{\sum_a \frac{(A_a - \bar{F}_a)^2}{2\sigma_a^2}}, \qquad \frac{\partial L}{\partial A_a} = \frac{A_a - \bar{F}_a}{\sigma_a^2} L. \tag{49}$$

Hellinger diastance:

$$L = \sum_a \frac{(\sqrt{A_a} - \sqrt{\bar{F}_a})^2}{\sigma_a}, \qquad \frac{\partial L}{\partial A_a} = \frac{\sqrt{A_a} - \sqrt{\bar{F}_a}}{\sqrt{A_a}\sigma_a}. \tag{50}$$

Kullback-Leibler divergence: for output values $\in [0, 1]$

$$L = \sum_a \left[\bar{F}_a \ln \frac{\bar{F}_a}{A_a} + A_a - \bar{F}_a\right], \qquad \frac{\partial L}{\partial A_a} = \frac{A_a - \bar{F}_a}{A_a}. \tag{51}$$