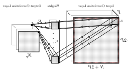
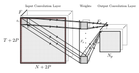
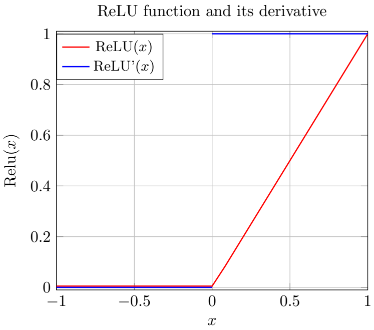
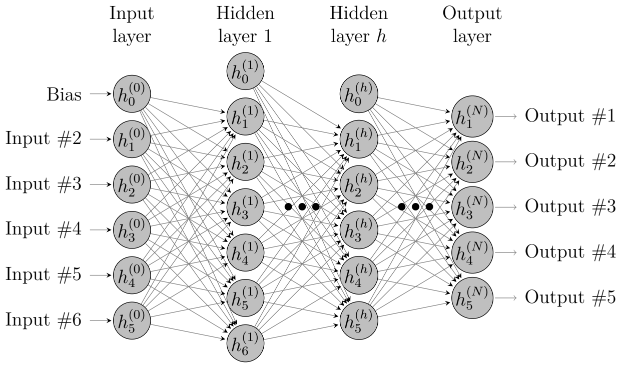


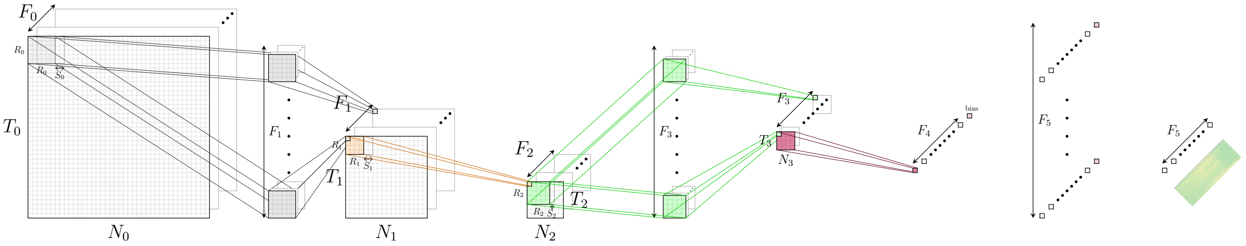
Deep learning: Technical introduction

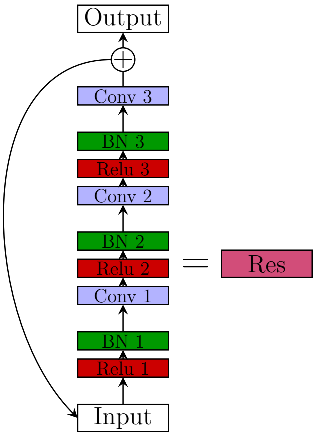


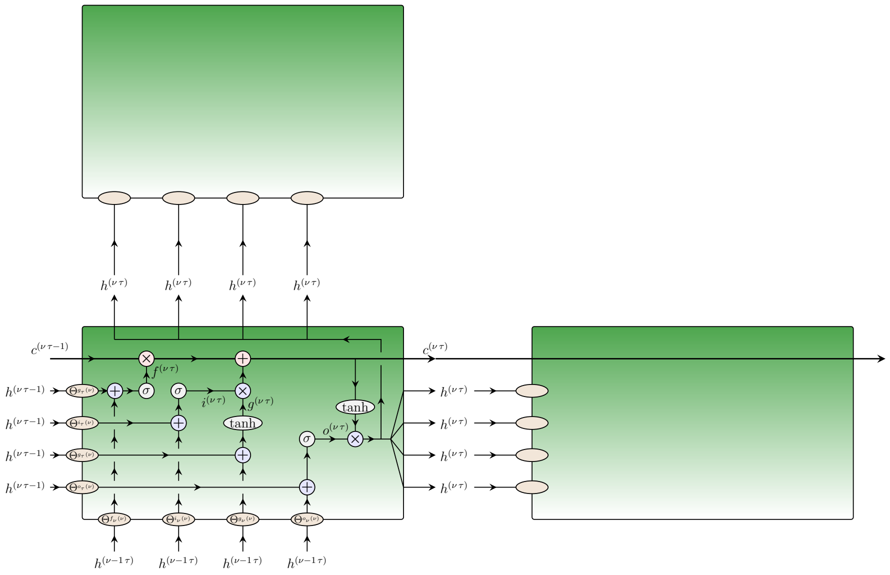
Thomas Epelbaum











August 31, 2017

Contents

Chapter 1

Preface

I started learning about deep learning fundamentals in February 2017. At this time, I knew nothing about backpropagation, and was completely ignorant about the differences between a Feedforward, Convolutional and a Recurrent Neural Networ.

As I navigated through the humongous amount of data available on deep learning online, I found myself quite frustated when it came to really understand what deep learning is, and not just applying it with some available library.

In particular, the backpropagation update rules are seldom derived, and never in index form. Unfortunately for me, I have an "index" mind: seeing a 4 Dimensional convolution formula in matrix form does not do it for me. Since I am also stupid enough to like recoding the wheel in low level programming languages, the matrix form cannot be directly converted into working code either.

I therefore started some notes for my personal use, where I tried to rederive everything from scratch in index form.

I did so for the vanilla Feedforward network, then learned about L1 and L2 regularization , dropout[1], batch normalization[2], several gradient descent optimization techniques... Then turned to convolutional networks, from conventional single digit number of layer conv-pool architectures[3] to recent VGG[4] ResNet[5] ones, from local contrast normalization and rectification to bacthnorm... And finally I studied Recurrent Neural Network structures[6], from the standard formulation to the most recent LSTM one[7].

As my work progressed, my notes got bigger and bigger, until a point when I realized I might have enough material to help others starting their own deep learning journey.

This work is bottom-up at its core. If you are searching a working Neural Network in 10 lines of code and 5 minutes of your time, you have come to the wrong place. If you can mentally multiply and convolve 4D tensors, then I have nothing to convey to you either.

If on the other hand you like(d) to rederive every tiny calculation of every theorem of every class that you stepped into, then you might be interested by what follow!

Chapter 2

Acknowledgements

This work has no benefit nor added value to the deep learning topic on its own. It is just the reformulation of ideas of brighter researchers to fit a peculiar mindset: the one of prefering formulas with ten indices but where one knows precisely what one is manipulating rather than (in my opinion sometimes opaque) matrix formulations where the dimension of the objects are rarely if ever specified.

Among the brighter people from whom I learned online are Andrew Ng. His Coursera class (here) was the first contact I got with Neural Network, and this pedagogical introduction allowed me to build on solid ground.

I also wish to particularly thanks Hugo Larochelle, who not only built a wonderful deep learning class (here), but was also kind enough to answer emails from a complete beginner and stranger!

The Standford class on convolutional networks (here) proved extremely valuable to me, so did the one on Natural Language processing (here).

I also benefited greatly from Sebastian Ruder's blog (here), both from the blog pages on gradient descent optimization techniques and from the author himself.

I learned more about LSTM on colah's blog (here), and some of my drawings are inspired from there.

I also thank Jonathan Del Hoyo for the great articles that he regularly shares on LinkedIn.

Many thanks go to my collaborators at Mediamobile, who let me dig as deep as I wanted on Neural Networks. I am especially indebted to Clément, Nicolas Jessica and Céline.

Thanks to Jean-Michel Loubes and Fabrice Gamboa, from whom I learned a great deal on probability theory and statistics.

I end this list with my employer, Mediamobile, which has been kind enough to let me work on this topic with complete freedom.

Chapter 3

Introduction

This note aims at presenting the three most common forms of neural network architectures. It does so in a technical though hopefully pedagogical way, buiding up in complexity as one progresses through the chapters.

Chapter 4 starts with the first type of network introduced historically: a regular feedforward neural network, itself an evolution of the original perceptron [8] algorithm. One should see the latter as a non-linear regression, and feedforward networks schematically stack perceptron layers on top of one another.

We will thus introduce in chapter 4 the fundamental building blocks of the simplest neural network layers: weight averaging and activation functions. We will also introduce gradient descent as a way to train the network when joint with the backpropagation algorithm, as a way to minimize a loss function adapted to the task at hand (classification or regression). The more technical details of the backpropagation algorithm are found in the appendix of this chapter, alongside with an introduction to the state of the art feedforward neural network, the ResNet. One can finally find a short matrix description of the feedforward network.

In chapter 5, we present the second type of neural network studied: the convolutional networks, particularly suited to treat images and label them. This implies presenting the mathematical tools related to this network: convolution, pooling, stride... As well as seeing the modification of the builiding block introduced in chapter 4. Several convolutional architectures are then presented, and the appendices once again detail the difficult steps of the main text.

Chapter 6 finally presents the network architecture suited for data with a temporal structure – as time series for instance, the recurrent neural network.

There again, the novelties and the modifications of the material introduced in the two previous chapters are detailed in the main text, while the appendices give all what one needs to understand the most cumbersome formula of this kind of network architecture.

Chapter 4

Feedforward Neural Networks

Contents

4.1 Introduction

In this section we review the first type of neural network that has been developed historically: a regular Feedforward Neural Network (FNN). This network does not take into account any particular structure that the input data might have. Nevertheless, it is already a very powerful machine learning tool, especially when used with the state of the art regularization techniques. These techniques – that we are going to present as well – allowed to circumvent the training issues that people experienced when dealing with "deep" architectures: namely the fact that neural networks with an important number of hidden states and hidden layers have proven historically to be very hard to train (vanishing gradient and overfitting issues).

4.2 FNN architecture

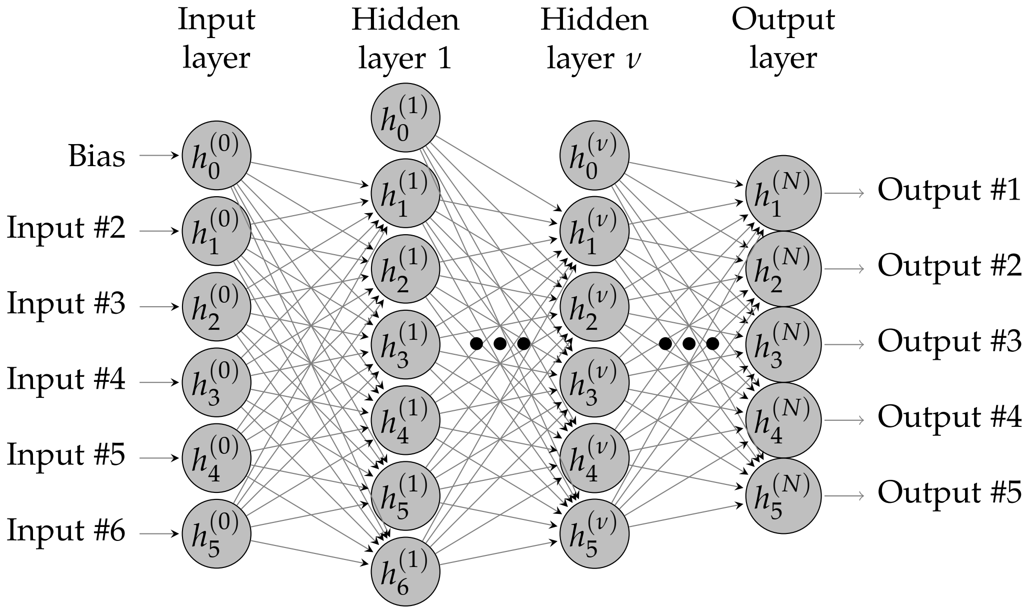


Figure 4.1: Neural Network with $N+1$ layers ($N-1$ hidden layers). for simplicity of notations, the index referencing the training set has not been indicated. Shallow architectures use only one hidden layer. Deep learning amounts to take several hidden layers, usually containing the same number of hidden neurons. This number should be on the ballpark of the average of the number of input and output variables.

A FNN is formed by one input layer, one (shallow network) or more (deep network, hence the name deep learning) hidden layers and one output layer. Each layer of the network (except the output one) is connected to a following layer. This connectivity is central to the FNN structure and has two main features in its simplest form: a weight averaging feature and an activation feature. We will review these features extensively in the following

4.3 Some notations

In the following, we will call

- N the number of layers (not counting the input) in the Neural Network.
- $T_{\rm train}$ the number of training examples in the training set.
- $T_{\rm mb}$ the number of training examples in a mini-batch (see section 4.7).
- $t \in \llbracket 0, T_{\rm mb} - 1 \rrbracket$ the mini-batch training instance index.
- $\nu \in \llbracket 0, N \rrbracket$ the number of layers in the FNN.
- F_ν the number of neurons in the ν'th layer.
- $X_f^{(t)} = h_f^{(0)(t)}$ where $f \in \llbracket 0, F_0 - 1 \rrbracket$ the input variables.
- $y_f^{(t)}$ where $f \in [0, F_N - 1]$ the output variables (to be predicted).
- $\hat{y}_f^{(t)}$ where $f \in [0, F_N - 1]$ the output of the network.
- $\Theta_f^{(\nu)f'}$ for $f \in [0, F_\nu - 1]$, $f' \in [0, F_{\nu+1} - 1]$ and $\nu \in [0, N - 1]$ the weights matrices
- A bias term can be included. In practice, we will see when talking about the batch-normalization procedure that we can omit it.

4.4 Weight averaging

One of the two main components of a FNN is a weight averaging procedure, which amounts to average the previous layer with some weight matrix to obtain the next layer. This is illustrated on the figure 4.2

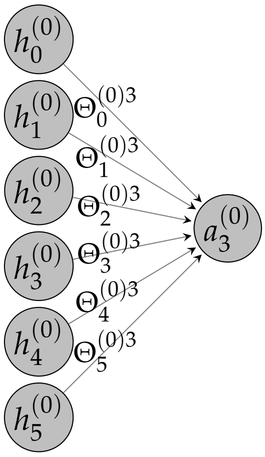


Figure 4.2: Weight averaging procedure.

Formally, the weight averaging procedure reads:

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu-1+\epsilon} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)} , \tag{4.1}$$

where $\nu \in [\![0, N-1]\!]$, $t \in [\![0, T_{\text{mb}}-1]\!]$ and $f \in [\![0, F_{\nu+1}-1]\!]$. The ϵ is here to include or exclude a bias term. In practice, as we will be using batch-normalization, we can safely omit it ($\epsilon = 0$ in all the following).

4.5 Activation function

The hidden neuron of each layer is defined as

$$h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right) , \tag{4.2}$$

where $\nu \in [\![0, N-2]\!]$, $f \in [\![0, F_{\nu+1}-1]\!]$ and as usual $t \in [\![0, T_{\text{mb}}-1]\!]$. Here g is an activation function – the second main ingredient of a FNN – whose non-linearity allow to predict arbitrary output data. In practice, g is usually taken to be one of the functions described in the following subsections.

4.5.1 The sigmoid function

the sigmoid function takes its value in $]0,1[$ and reads

$$g(x) = \sigma(x) = \frac{1}{1+e^{-x}} . \tag{4.3}$$

Its derivative is

$$\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right) \ . \tag{4.4}$$

This activation function is not much used nowadays (except in RNN-LSTM networks that we will present later in chapter 6).

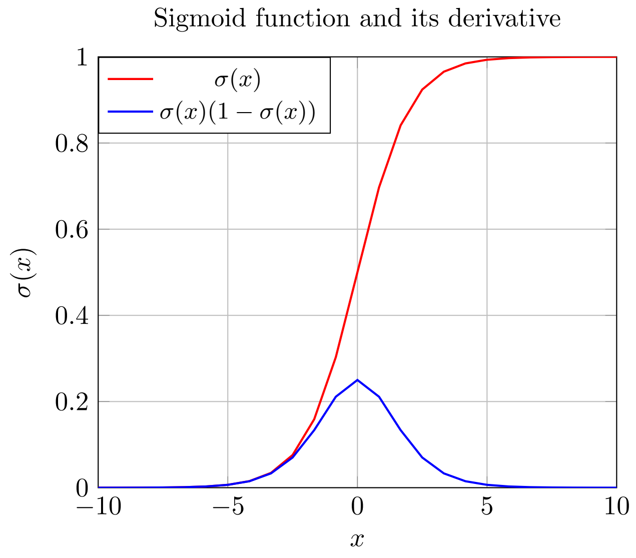


Figure 4.3: the sigmoid function and its derivative.

4.5.2 The tanh function

the tanh function takes its value in $]-1,1[$ and reads

$$g(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \ . \tag{4.5}$$

Its derivative is

$$\tanh'(x) = 1 - \tanh^2(x) \ . \tag{4.6}$$

This activation function has seen its popularity drop due to the use of the activation function presented in the next section.

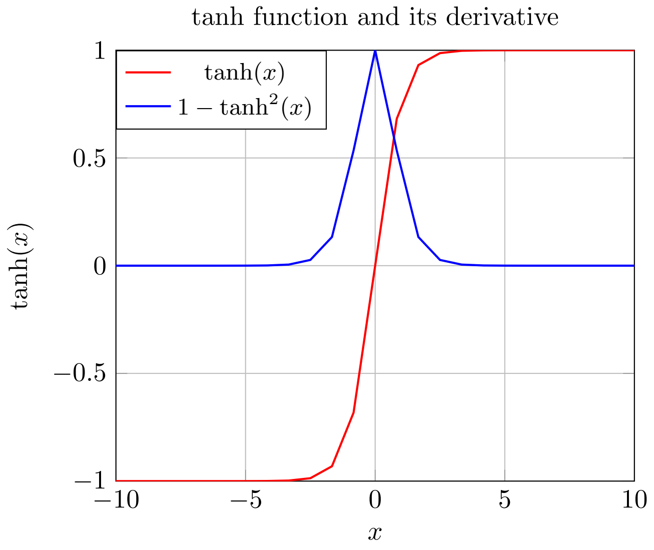


Figure 4.4: the tanh function and its derivative.

It is nevertherless still used in the standard formulation of the RNN-LSTM model (6).

4.5.3 The ReLU function

the ReLU –for Rectified Linear Unit – function takes its value in $[0, +\infty$ and reads

$$g(x) = \text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} . \tag{4.7}$$

Its derivative is

$$\text{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} . \tag{4.8}$$

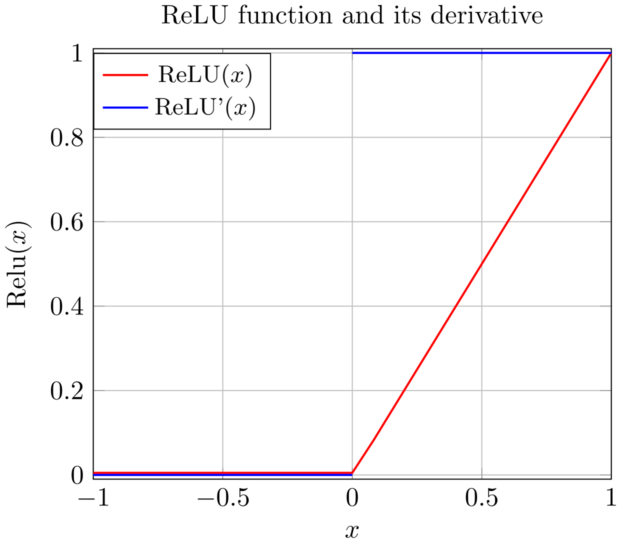


Figure 4.5: the ReLU function and its derivative.

This activation function is the most extensively used nowadays. Two of its more common variants can also be found : the leaky ReLU and ELU – Exponential Linear Unit. They have been introduced because the ReLU activation function tends to "kill" certain hidden neurons: once it has been turned off (zero value), it can never be turned on again.

4.5.4 The leaky-ReLU function

the leaky-ReLU –for Linear Rectified Linear Unit – function takes its value in $]-\infty,+\infty[$ and is a slight modification of the ReLU that allows non-zero value for the hidden neuron whatever the x value. It reads

$$g(x) = 1 - \text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0.01\,x & x < 0 \end{cases} . \tag{4.9}$$

Its derivative is

$$1 - \text{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ 0.01 & x < 0 \end{cases} . \tag{4.10}$$

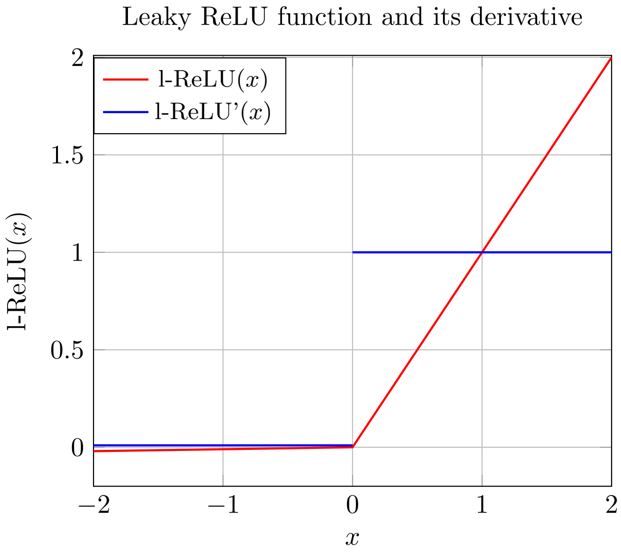


Figure 4.6: the leaky-ReLU function and its derivative.

A variant of the leaky-ReLU can also be found in the litterature : the Parametric-ReLU, where the arbirtrary 0.01 in the definition of the leaky-ReLU is replaced by an α coefficient, that can be computed via backpropagation

$$g(x) = \text{Parametric} - \text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ \alpha\, x & x < 0 \end{cases} . \tag{4.11}$$

Its derivative is

$$\text{Parametric} - \text{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ \alpha & x < 0 \end{cases} . \tag{4.12}$$

4.5.5 The ELU function

The ELU –for Exponential Linear Unit – function takes its value between $]-\infty, +\infty[$ and is inspired by the leaky-ReLU philosophy: non-zero values for all x's. But it presents the advantage of being $\mathcal{C}^1$.

$$g(x) = \text{ELU}(x) = \begin{cases} x & x \geq 0 \\ e^x - 1 & x < 0 \end{cases} . \tag{4.13}$$

Its derivative is

$$\text{ELU}'(x) = \begin{cases} 1 & x \geq 0 \\ e^x & x < 0 \end{cases} . \tag{4.14}$$

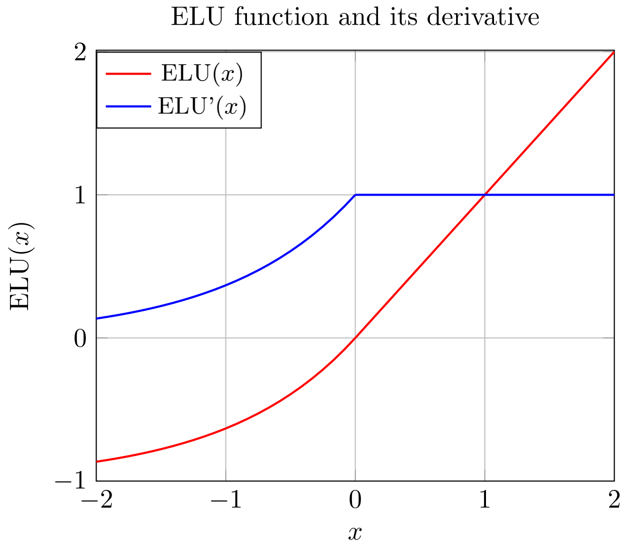


Figure 4.7: the ELU function and its derivative.

4.6 FNN layers

As illustrated in figure 4.1, a regular FNN is composed by several specific layers. Let us explicit them one by one.

4.6.1 Input layer

The input layer is one of the two places where the data at disposal for the problem at hand come into place. In this chapter, we will be considering a input of size F_0, denoted $X^{(t)}_f$, with[1] $t \in [\![0, T_{\rm mb} - 1]\!]$ (size of the mini-batch, more on that when we will be talking about gradient descent techniques), and $f \in [\![0, F_0 - 1]\!]$. Given the problem at hand, a common procedure could be to center the input following the procedure

$$\tilde{X}^{(t)}_f = X^{(t)}_f - \mu_f \, , \tag{4.15}$$

with

$$\mu_f = \frac{1}{T_{\rm train}} \sum_{t=0}^{T_{\rm train}-1} X^{(t)}_f \, . \tag{4.16}$$

[1] To train the FNN, we jointly compute the forward and backward pass for $T_{\rm mb}$ samples of the training set, with $T_{\rm mb} \ll T_{\rm train}$. In the following we will thus have $t \in [\![0, T_{\rm mb} - 1]\!]$.

This correspond to compute the mean per data types over the training set. Following our notations, let us recall that

$$X_f^{(t)} = h_f^{(t)(0)} . \tag{4.17}$$

4.6.2 Fully connected layer

The fully connected operation is just the conjonction of the weight averaging and the activation procedure. Namely, $\forall \nu \in [\![0, N-1]\!]$

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu - 1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)} . \tag{4.18}$$

and $\forall \nu \in [\![0, N-2]\!]$

$$h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right) . \tag{4.19}$$

for the case where $\nu = N-1$, the activation function is replaced by an output function.

4.6.3 Output layer

The ouput of the FNN reads

$$h_f^{(t)(N)} = o(a_f^{(t)(N-1)}) , \tag{4.20}$$

where o is called the output function. In the case of the Euclidean loss function, the output function is just the identity. In a classification task, o is the softmax function

$$o\left(a_f^{(t)(N-1)}\right) = \frac{e^{a_f^{(t)(N-1)}}}{\sum\limits_{f'=0}^{F_{N-1}-1} e^{a_{f'}^{(t)(N-1)}}} \tag{4.21}$$

4.7 Loss function

The loss function evaluates the error performed by the FNN when it tries to estimate the data to be predicted (second place where the data make their appearance). For a regression problem, this is simply a mean square error (MSE) evaluation

$$J(\Theta) = \frac{1}{2T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \sum_{f=0}^{F_N-1} \left(y_f^{(t)} - h_f^{(t)(N)} \right)^2 , \tag{4.22}$$

while for a classification task, the loss function is called the cross-entropy function

$$J(\Theta) = -\frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \sum_{f=0}^{F_N-1} \delta^f_{y^{(t)}} \ln h_f^{(t)(N)} , \tag{4.23}$$

and for a regression problem transformed into a classification one, calling C the number of bins leads to

$$J(\Theta) = -\frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \sum_{f=0}^{F_N-1} \sum_{c=0}^{C-1} \delta^c_{y_f^{(t)}} \ln h_{fc}^{(t)(N)} . \tag{4.24}$$

For reasons that will appear clear when talking about the data sample used at each training step, we denote

$$J(\Theta) = \sum_{t=0}^{T_{\rm mb}-1} J_{\rm mb}(\Theta) . \tag{4.25}$$

4.8 Regularization techniques

On of the main difficulties when dealing with deep learning techniques is to get the deep neural network to train efficiently. To that end, several regularization techniques have been invented. We will review them in this section

4.8.1 L2 regularization

L2 regularization is the most common regularization technique that on can find in the litterature. It amounts to add a regularizing term to the loss function in the following way

$$J_{\rm L2}(\Theta) = \lambda_{\rm L2} \sum_{\nu=0}^{N-1} \left\| \Theta^{(\nu)} \right\|^2_{\rm L2} = \lambda_{\rm L2} \sum_{\nu=0}^{N-1} \sum_{f=0}^{F_{\nu+1}-1} \sum_{f'=0}^{F_\nu-1} \left(\Theta_f^{(\nu)f'} \right)^2 . \tag{4.26}$$

This regularization technique is almost always used, but not on its own. A typical value of λ_{L2} is in the range $10^{-4} - 10^{-2}$. Interestingly, this L2 regularization technique has a Bayesian interpretation: it is Bayesian inference with a Gaussian prior on the weights. Indeed, for a given ν, the weight averaging procedure can be considered as

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)} + \epsilon \,, \tag{4.27}$$

where ϵ is a noise term of mean 0 and variance σ^2. Hence the following Gaussian likelihood for all values of t and f:

$$\mathcal{N}\left(a_f^{(t)(i)} \middle| \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)}, \sigma^2\right) \,. \tag{4.28}$$

Assuming all the weights to have a Gaussian prior of the form $\mathcal{N}\left(\Theta_{f'}^{(\nu)f} \middle| \lambda_{\text{L2}}^{-1}\right)$ with the same parameter λ_{L2}, we get the following expression

$$\begin{aligned}\mathcal{P} &= \prod_{t=0}^{T_{\text{mb}}-1} \prod_{f=0}^{F_{\nu+1}-1} \left[\mathcal{N}\left(a_f^{(t)(\nu)} \middle| \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)}, \sigma^2\right) \prod_{f'=0}^{F_\nu-1} \mathcal{N}\left(\Theta_{f'}^{(\nu)f} \middle| \lambda_{\text{L2}}^{-1}\right)\right] \\ &= \prod_{t=0}^{T_{\text{mb}}-1} \prod_{f=0}^{F_{\nu+1}-1} \left[\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(a_f^{(t)(\nu)} - \sum_{f'=0}^{F_i-1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)}\right)^2}{2\sigma^2}} \prod_{f'=0}^{F_\nu-1} \sqrt{\frac{\lambda_{\text{L2}}}{2\pi}} e^{-\frac{\left(\Theta_{f'}^{(\nu)f}\right)^2 \lambda_{\text{L2}}}{2}}\right] \,.\end{aligned} \tag{4.29}$$

Taking the log of it and forgetting most of the constant terms leads to

$$\mathcal{L} \propto \frac{1}{T_{\text{mb}}\sigma^2} \sum_{t=0}^{T_{\text{mb}}-1} \sum_{f=0}^{F_{\nu+1}-1} \left(a_f^{(t)(\nu)} - \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{(\nu)f} h_{f'}^{(t)(\nu)}\right)^2 + \lambda_{\text{L2}} \sum_{f=0}^{F_{\nu+1}-1} \sum_{f'=0}^{F_\nu-1} \left(\Theta_{f'}^{(\nu)f}\right)^2 \,, \tag{4.30}$$

and the last term is exactly the L2 regulator for a given *nu* value (see formula (4.26)).

4.8.2 L1 regularization

L1 regularization amounts to replace the L2 norm by the L1 one in the L2 regularization technique

$$J_{\text{L1}}(\Theta) = \lambda_{\text{L1}} \sum_{\nu=0}^{N-1} \left\|\Theta^{(\nu)}\right\|_{\text{L1}} = \lambda_{\text{L1}} \sum_{\nu=0}^{N-1} \sum_{f=0}^{F_{\nu+1}-1} \sum_{f'=0}^{F_\nu-1} \left|\Theta_{f'}^{(\nu)f}\right| \,. \tag{4.31}$$

It can be used in conjonction with L2 regularization, but again these techniques are not sufficient on their own. A typical value of λ_{L1} is in the range $10^{-4} - 10^{-2}$. Following the same line as in the previous section, one can show that L1 regularization is equivalent to Bayesian inference with a Laplacian prior on the weights

$$\mathcal{F}\left(\Theta^{(\nu)f}_{f'}\middle|0,\lambda^{-1}_{\text{L1}}\right) = \frac{\lambda_{\text{L1}}}{2} e^{-\lambda_{\text{L1}}\left|\Theta^{(\nu)f}_{f'}\right|} . \tag{4.32}$$

4.8.3 Clipping

Clipping forbids the L2 norm of the weights to go beyond a pre-determined threshold C. Namely after having computed the update rules for the weights, if their L2 norm goes above C, it is pushed back to C

$$\text{if } \left\|\Theta^{(\nu)}\right\|_{\text{L2}} > C \longrightarrow \Theta^{(\nu)f}_{f'} = \Theta^{(\nu)f}_{f'} \times \frac{C}{\left\|\Theta^{(\nu)}\right\|_{\text{L2}}} . \tag{4.33}$$

This regularization technique avoids the so-called exploding gradient problem, and is mainly used in RNN-LSTM networks. A typical value of C is in the range $10^0 - 10^1$. Let us now turn to the most efficient regularization techniques for a FNN: dropout and Batch-normalization.

4.8.4 Dropout

A simple procedure allows for better backpropagation performance for classification tasks: it amounts to stochastically drop some of the hidden units (and in some instances even some of the input variables) for each training example.

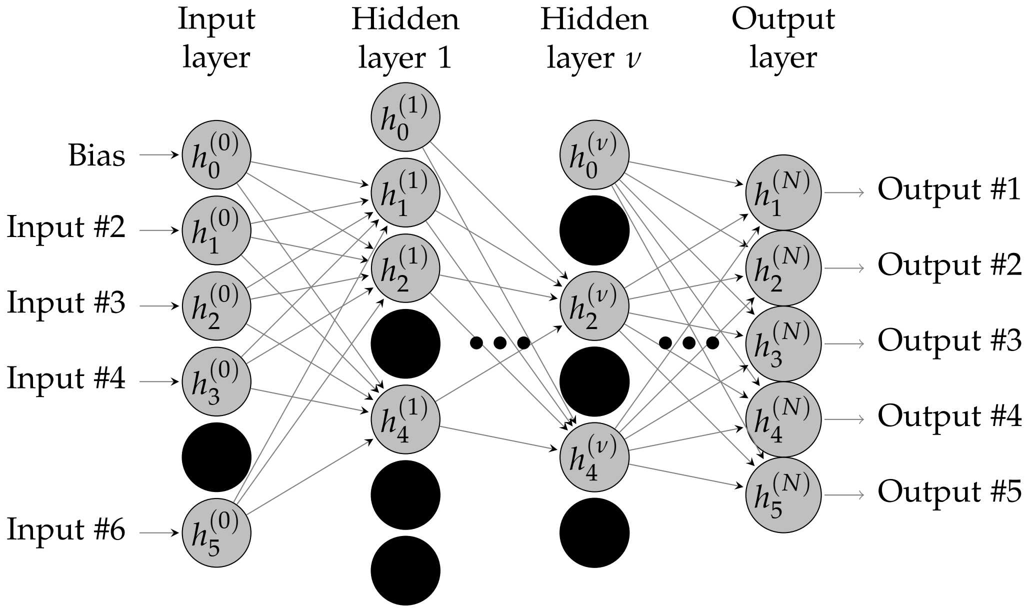


Figure 4.8: The neural network of figure 4.1 with dropout taken into account for both the hidden layers and the input. Usually, a different (lower) propability for turning off a neuron is adopted for the input thanthe one adopted for the hidden layers.

This amounts to do the following change: for $\nu \in [\![1, N-1]\!]$

$$h_f^{(\nu)} = m_f^{(\nu)} g\left(a_f^{(\nu)}\right) \tag{4.34}$$

with $m_f^{(i)}$ following a p Bernouilli distribution with usually $p = \frac{1}{5}$ for the mask of the input layer and $p = \frac{1}{2}$ otherwise. Dropout[1] has been the most succesful regularization technique until the appearance of Batch Normalization.

4.8.5 Batch Normalization

Batch normalization[2] amounts to joinlty normalize the mini-batch set per data types, and does so at each input of a FNN layer. In the original paper, the authors argued that this step should be done after the convolutional layers, but in practice it has been shown to be more efficient after the non-linear step. In our case, we will thus consider $\forall i \in [\![0, N-2]\!]$

$$\tilde{h}_f^{(t)(\nu)} = \frac{h_f^{(t)(\nu+1)} - \hat{h}_f^{(\nu)}}{\sqrt{\left(\hat{\sigma}_f^{(\nu)}\right)^2 + \epsilon}} \,, \tag{4.35}$$

with

$$\hat{h}_f^{(\nu)} = \frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} h_f^{(t)(\nu+1)} \tag{4.36}$$

$$\left(\hat{\sigma}_f^{(\nu)}\right)^2 = \frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \left(h_f^{(t)(\nu+1)} - \hat{h}_f^{(\nu)}\right)^2 \,. \tag{4.37}$$

To make sure that this transformation can represent the identity transform, we add two additional parameters (γ_f, β_f) to the model

$$y_f^{(t)(\nu)} = \gamma_f^{(\nu)}\, \tilde{h}_f^{(t)(\nu)} + \beta_f^{(\nu)} = \tilde{\gamma}_f^{(\nu)}\, h_f^{(t)(\nu)} + \tilde{\beta}_f^{(\nu)} \,. \tag{4.38}$$

The presence of the $\beta_f^{(\nu)}$ coefficient is what pushed us to get rid of the bias term, as it is naturally included in batchnorm. During training, one must compute a running sum for the mean and the variance, that will serve for the evaluation of the cross-validation and the test set (calling e the number of iterations/epochs)

$$\mathbb{E}\left[h_f^{(t)(\nu+1)}\right]_{e+1} = \frac{e\mathbb{E}\left[h_f^{(t)(\nu)}\right]_e + \hat{h}_f^{(\nu)}}{e+1} \,, \tag{4.39}$$

$$\mathbb{V}ar\left[h_f^{(t)(\nu+1)}\right]_{e+1} = \frac{e\mathbb{V}ar\left[h_f^{(t)(\nu)}\right]_e + \left(\hat{\sigma}_f^{(\nu)}\right)^2}{e+1} \tag{4.40}$$

and what will be used at test time is

$$\mathbb{E}\left[h_f^{(t)(\nu)}\right] = \mathbb{E}\left[h_f^{(t)(\nu)}\right] \,, \qquad \mathbb{V}ar\left[h_f^{(t)(\nu)}\right] = \frac{T_{\rm mb}}{T_{\rm mb}-1}\mathbb{V}ar\left[h_f^{(t)(\nu)}\right] \,. \tag{4.41}$$

so that at test time

$$y_f^{(t)(\nu)} = \gamma_f^{(\nu)} \frac{h_f^{(t)(\nu)} - E[h_f^{(t)(\nu)}]}{\sqrt{Var\left[h_f^{(t)(\nu)}\right] + \epsilon}} + \beta_f^{(\nu)} \,. \tag{4.42}$$

In practice, and as advocated in the original paper, on can get rid of dropout without loss of precision when using batch normalization. We will adopt this convention in the following.

4.9 Backpropagation

Backpropagation[9] is the standard technique to decrease the loss function error so as to correctly predict what one needs. As it name suggests, it amounts to backpropagate through the FNN the error performed at the output layer, so as to update the weights. In practice, on has to compute a bunch of gradient terms, and this can be a tedious computational task. Nevertheless, if performed correctly, this is the most useful and important task that one can do in a FN. We will therefore detail how to compute each weight (and Batchnorm coefficients) gradients in the following.

4.9.1 Backpropagate through Batch Normalization

Backpropagation introduces a new gradient

$$\delta^f_{f'} J^{(tt')(\nu)}_f = \frac{\partial y^{(t')(\nu)}_{f'}}{\partial h^{(t)(\nu+1)}_f} \, . \tag{4.43}$$

we show in appendix 4.C that

$$J^{(tt')(\nu)}_f = \tilde{\gamma}^{(\nu)}_f \left[\delta^{t'}_t - \frac{1 + \tilde{h}^{(t')(\nu)}_f \tilde{h}^{(t)(\nu)}_f}{T_{\rm mb}} \right] \, . \tag{4.44}$$

4.9.2 error updates

To backpropagate the loss error through the FNN, it is very useful to compute a so-called error rate

$$\delta^{(t)(\nu)}_f = \frac{\partial}{\partial a^{(t)(\nu)}_f} J(\Theta) \, , \tag{4.45}$$

We show in Appendix 4.B that $\forall \nu \in [\![0, N-2]\!]$

$$\delta^{(t)(\nu)}_f = g'\left(a^{(t)(\nu)}_f\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \Theta^{(\nu+1)f'}_f J^{(tt')(\nu)}_f \delta^{(t')(\nu+1)}_{f'} \, , \tag{4.46}$$

the value of $\delta^{(t)(N-1)}_f$ depends on the loss used. We show also in appendix 4.A that for the MSE loss function

$$\delta^{(t)(N-1)}_f = \frac{1}{T_{\rm mb}} \left(h^{(t)(N)}_f - y^{(t)}_f \right) \, , \tag{4.47}$$

and for the cross entropy loss function

$$\delta_f^{(t)(N-1)} = \frac{1}{T_{\text{mb}}}\left(h_f^{(t)(N)} - \delta^f_{y^{(t)}}\right) . \tag{4.48}$$

To unite the notation of chapters 4, 5 and 6, we will call

$$\mathcal{H}_{ff'}^{(t)(\nu+1)} = g'\left(a_f^{(t)(\nu)}\right)\Theta_f^{(\nu+1)f'} , \tag{4.49}$$

so that the update rule for the error rate reads

$$\delta_f^{(t)(\nu)} = \sum_{t'=0}^{T_{\text{mb}}-1} J_f^{(tt')(\nu)} \sum_{f'=0}^{F_{\nu+1}-1} \mathcal{H}_{ff'}^{(t)(\nu+1)}\delta_{f'}^{(t)(\nu+1)} . \tag{4.50}$$

4.9.3 Weight update

Thanks to the computation of the error rates, the derivation of the error rate is straghtforward. We indeed get $\forall \nu \in [\![1, N-1]\!]$

$$\Delta_{f'}^{\Theta(\nu)f} = \frac{1}{T_{\text{mb}}}\sum_{t=0}^{T_{\text{mb}}-1}\sum_{f''=0}^{F_{\nu+1}-1}\sum_{f'''=0}^{F_\nu}\frac{\partial\Theta_{f'''}^{(\nu)f''}}{\partial\Theta_{f'}^{(\nu)f}} y_{f'''}^{(t)(\nu-1)}\delta_{f''}^{(t)(\nu)} = \sum_{t=0}^{T_{\text{mb}}-1}\delta_f^{(t)(\nu)}y_{f'}^{(t)(\nu-1)} . \tag{4.51}$$

and

$$\Delta_{f'}^{\Theta(0)f} = \sum_{t=0}^{T_{\text{mb}}-1}\delta_f^{(t)(0)}h_{f'}^{(t)(0)} . \tag{4.52}$$

4.9.4 Coefficient update

The update rule for the Batchnorm coefficient can easily be computed thanks to the error rate. It reads

$$\Delta_f^{\gamma(\nu)} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{f'=0}^{F_{\nu+1}-1}\frac{\partial a_{f'}^{(t)(\nu+1)}}{\partial\gamma_f^{(i)}}\delta_{f'}^{(t)(\nu+1)} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{f'=0}^{F_{\nu+1}-1}\Theta_f^{(\nu+1)f'}\tilde{h}_f^{(t)(i)}\delta_{f'}^{(t)(\nu+1)} , \tag{4.53}$$

$$\Delta_f^{\beta(\nu)} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{f'=0}^{F_{\nu+1}-1}\frac{\partial a_{f'}^{(t)(\nu+1)}}{\partial\beta_f^{(i)}}\delta_{f'}^{(t)(\nu+1)} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{f'=0}^{F_{\nu+1}-1}\Theta_f^{(\nu+1)f'}\delta_{f'}^{(t)(\nu+1)} , \tag{4.54}$$

4.10 Which data sample to use for gradient descent?

From the beginning we have denoted $T_{\rm mb}$ the sample of the data from which we train our model. This procedure is repeated a large number of time (each time is called an epoch). But in the litterature there exists three way to sample from the data: Full-batch, Stochastic and Mini-batch gradient descent. We explicit these terms in the following sections.

4.10.1 Full-batch

Full-batch takes the whole training set at each epoch, such that the loss function reads

$$J(\Theta) = \sum_{t=0}^{T_{\rm train}-1} J_{\rm train}(\Theta) \ . \tag{4.55}$$

This choice has the advantage to be numerically stable, but it so coslty in computation time that it is rarely if ever used.

4.10.2 Stochastic Gradient Descent (SGD)

SGD amounts to take only one exemplary of the training set at each epoch

$$J(\Theta) = J_{\rm SGD}(\Theta) \ . \tag{4.56}$$

This choice leads to faster computations, but is so numerically unstable that the most standard choice by far is Mini-batch gradient descent.

4.10.3 Mini-batch

Mini-batch gradient descent is a compromise between stability and time efficiency, and is the middle-ground between Full-batch and Stochastic gradient descent: $1 \ll T_{\rm mb} \ll T_{\rm train}$. Hence

$$J(\Theta) = \sum_{t=0}^{T_{\rm mb}-1} J_{\rm mb}(\Theta) \ . \tag{4.57}$$

All the calculations in this note have been performed using this gradient descent technique.

4.11 Gradient optimization techniques

Once the gradients for backpropagation have been computed, the question of how to add them to the existing weights arise. The most natural choice would be to take

$$\Theta^{(\nu)f}_{f'} = \Theta^{(\nu)f}_{f'} - \eta \Delta^{\Theta(i)f}_{f'} \; . \tag{4.58}$$

where η is a free parameter that is generally initialized thanks to cross-validation. It can also be made epoch dependent (with usually a slow exponentially decaying behaviour). When using Mini-batch gradient descent, this update choice for the weights presents the risk of having the loss function being stuck in a local minimum. Several method have been invented to prevent this risk. We are going to review them in the next sections.

4.11.1 Momentum

Momentum[10] introduces a new vector $v_{\rm e}$ and can be seen as keeping a memory of what where the previous updates at prior epochs. Calling e the number of epochs and forgetting the f, f', ν indices for the gradients to ease the notations, we have

$$v_{\rm e} = \gamma v_{\rm e-1} + \eta \Delta^{\Theta} \; , \tag{4.59}$$

and the weights at epoch e are then updated as

$$\Theta_e = \Theta_{e-1} - v_{\rm e} \; . \tag{4.60}$$

γ is a new parameter of the model, that is usually set to 0.9 but that could also be fixed thanks to cross-validation.

4.11.2 Nesterov accelerated gradient

Nesterov accelerated gradient[11] is a slight modification of the momentum technique that allows the gradients to escape from local minima. It amounts to take

$$v_{\rm e} = \gamma v_{\rm e-1} + \eta \Delta^{\Theta - \gamma v_{\rm e-1}} \; , \tag{4.61}$$

and then again

$$\Theta_e = \Theta_{e-1} - v_{\rm e} \; . \tag{4.62}$$

Until now, the parameter η that controls the magnitude of the update has been set globally. It would be nice to have a fine control of it, so that different weights can be updated with different magnitudes.

4.11.3 Adagrad

Adagrad[12] allows to fine tune the different gradients by having individual learning rates η. Calling for each value of f, f', i

$$v_{\mathrm{e}} = \sum_{e'=0}^{e-1} \left(\Delta_{e'}^{\Theta}\right)^2 \,, \tag{4.63}$$

the update rule then reads

$$\Theta_e = \Theta_{e-1} - \frac{\eta}{\sqrt{v_{\mathrm{e}} + \epsilon}} \Delta_e^{\Theta} \,. \tag{4.64}$$

One advantage of Adagrad is that the learning rate η can be set once and for all (usually to 10^{-2}) and does not need to be fine tune via cross validation anymore, as it is individually adapted to each weight via the v_{e} term. ϵ is here to avoid division by 0 issues, and is usually set to 10^{-8}.

4.11.4 RMSprop

Since in Adagrad one adds the gradient from the first epoch, the weight are forced to monotonically decrease. This behaviour can be smoothed via the Adadelta technique, which takes

$$v_{\mathrm{e}} = \gamma v_{\mathrm{e}-1} + (1-\gamma)\Delta_e^{\Theta} \,, \tag{4.65}$$

with γ a new parameter of the model, that is usually set to 0.9. The Adadelta update rule then reads as the Adagrad one

$$\Theta_e = \Theta_{e-1} - \frac{\eta}{\sqrt{v_{\mathrm{e}} + \epsilon}} \Delta_e^{\Theta} \,. \tag{4.66}$$

η can be set once and for all (usually to 10^{-3}).

4.11.5 Adadelta

Adadelta[13] is an extension of RMSprop, that aims at getting rid of the η parameter. To do so, a new vector update is introduced

$$m_{\mathrm{e}} = \gamma m_{\mathrm{e}-1} + (1-\gamma) \left(\frac{\sqrt{m_{\mathrm{e}-1} + \epsilon}}{\sqrt{v_{\mathrm{e}} + \epsilon}} \Delta_e^{\Theta}\right)^2 \,, \tag{4.67}$$

and the new update rule for the weights reads

$$\Theta_e = \Theta_{e-1} - \frac{\sqrt{m_{\rm e-1}+\epsilon}}{\sqrt{v_{\rm e}+\epsilon}}\Delta_e^{\Theta} \; . \tag{4.68}$$

The learning rate has been completely eliminated from the update rule, but the procedure for doing so is add hoc. The next and last optimization technique presented seems more natural and is the default choice on a number of deep learning algorithms.

4.11.6 Adam

Adam[14] keeps track of both the gradient and its square via two epoch dependent vectors

$$m_{\rm e} = \beta_1 m_{\rm e-1} + (1-\beta_1)\Delta_e^{\Theta} \; , \qquad v_{\rm e} = \beta_2 v_{\rm e} + (1-\beta_2)\left(\Delta_e^{\Theta}\right)^2 \; , \tag{4.69}$$

with β_1 and β_2 parameters usually respectively set to 0.9 and 0.999. But the robustness and great strength of Adam is that it makes the whole learning process weakly dependent of their precise value. To avoid numerical problems during the first steps, these vector are rescaled

$$\hat{m}_{\rm e} = \frac{m_{\rm e}}{1-\beta_1^e} \; , \qquad \hat{v}_{\rm e} = \frac{v_{\rm e}}{1-\beta_2^e} \; . \tag{4.70}$$

before entering into the update rule

$$\Theta_e = \Theta_{e-1} - \frac{\eta}{\sqrt{\hat{v}_{\rm e}+\epsilon}}\hat{m}_{\rm e} \; . \tag{4.71}$$

This is the optimization technique implicitly used throughout this note, alongside with a learning rate decay

$$\eta_e = e^{-\alpha_0}\eta_{e-1} \; , \tag{4.72}$$

α_0 determinde by cross-validation, and η_0 usually initialized in the range $10^{-3} - 10^{-2}$.

4.12 Weight initialization

Without any regularization, training a neural network can be a daunting task because of the fine-tuning of the weight initial conditions. This is one of the reasons why neural networks have experienced out of mode periods. Since dropout and Batch normalization, this issue is less pronounced, but one should not initialize the weigt in a symmetric fashion (all zero for instance), nor should one intialize them too large. A good heuristic is

$$\left[\Theta_{f}^{(\nu)f'}\right]_{\text{init}} = \sqrt{\frac{6}{F_i + F_{i+1}}} \times \mathcal{N}(0,1) \ . \tag{4.73}$$

Appendix

4.A Backprop through the output layer

Recalling the MSE loss function

$$J(\Theta) = \frac{1}{2T_{\text{mb}}} \sum_{t=0}^{T_{\text{mb}}-1} \sum_{f=0}^{F_N-1} \left(y_f^{(t)} - h_f^{(t)(N)}\right)^2 \ , \tag{4.74}$$

we instantaneously get

$$\delta_f^{(t)(N-1)} = \frac{1}{T_{\text{mb}}} \left(h_f^{(t)(N)} - y_f^{(t)}\right) \ . \tag{4.75}$$

Things are more complicated for the cross-entropy loss function of a regression problem transformed into a multi-classification task. Assuming that we have C classes for all the values that we are trying to predict, we get

$$\delta_{fc}^{(t)(N-1)} = \frac{\partial}{\partial a_{fc}^{(t)(N-1)}} J(\Theta) = \sum_{t'=0}^{T_{\text{mb}}-1} \sum_{f'=0}^{F_N-1} \sum_{d=0}^{C-1} \frac{\partial h_{f'd}^{(t')(N)}}{\partial a_{fc}^{(t)(N-1)}} \frac{\partial}{\partial h_{f'd}^{(t')(N)}} J(\Theta) \ . \tag{4.76}$$

Now

$$\frac{\partial}{\partial h_{f'd}^{(t')(N)}} J(\Theta) = -\frac{\delta^{d}_{y_{f'}^{(t')}}}{T_{\text{mb}} h_{f'd}^{(t')(N)}} \ , \tag{4.77}$$

and

$$\frac{\partial h_{f'd}^{(t')(N)}}{\partial a_{fc}^{(t)(N-1)}} = \delta^f_{f'} \delta^t_{t'} \left(\delta^c_d h_{fc}^{(t)(N)} - h_{fc}^{(t)(N)} h_{fd}^{(t)(N)}\right) \ , \tag{4.78}$$

so that

$$\begin{aligned}\delta_{fc}^{(t)(N-1)} &= -\frac{1}{T_{\rm mb}}\sum_{d=0}^{C-1}\frac{\delta^{d}_{y_f^{(t)}}}{h_{fd}^{(t)(N)}}\left(\delta^c_d h_{fc}^{(t)(N)} - h_{fc}^{(t)(N)}h_{fd}^{(t)(N)}\right)\\ &= \frac{1}{T_{\rm mb}}\left(h_{fc}^{(t)(N)} - \delta^c_{y_f^{(t)}}\right) . \end{aligned} \tag{4.79}$$

For a true classification problem, we easily deduce

$$\delta_{fc}^{(t)(N-1)} = \frac{1}{T_{\rm mb}}\left(h_{f}^{(t)(N)} - \delta^f_{y^{(t)}}\right) . \tag{4.80}$$

4.B Backprop through hidden layers

To go further we need

$$\begin{aligned}\delta_f^{(t)(\nu)} &= \frac{\partial}{\partial a_f^{(t)(\nu)}}J^{(t)}(\Theta) = \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\frac{\partial a_{f'}^{(t')(\nu+1)}}{\partial a_f^{(t)(\nu)}}\delta_{f'}^{(t')(\nu+1)}\\ &= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{f''=0}^{F_\nu}\Theta_{f''}^{(\nu+1)f'}\frac{\partial y_{f''}^{(t')(\nu)}}{\partial a_f^{(t)(\nu)}}\delta_{f'}^{(t')(\nu+1)}\\ &= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{f''=0}^{F_\nu}\Theta_{f''}^{(\nu+1)f'}\frac{\partial y_{f''}^{(t')(\nu)}}{\partial h_f^{(t)(\nu+1)}}g'\left(a_f^{(t)(\nu)}\right)\delta_{f'}^{(t')(\nu+1)} , \end{aligned} \tag{4.81}$$

so that

$$\delta_f^{(t)(\nu)} = g'\left(a_f^{(t)(\nu)}\right)\sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\Theta_f^{(\nu+1)f'}J_f^{(tt')(\nu)}\delta_{f'}^{(t)(\nu+1)} , \tag{4.82}$$

4.C Backprop through BatchNorm

We saw in section 4.9.1 that batch normalization implies among other things to compute the follwowing gradient.

$$\frac{\partial y_{f'}^{(t')(\nu)}}{\partial h_f^{(t)(\nu+1)}} = \gamma_f^{(\nu)}\frac{\partial \tilde{h}_{f'}^{(t)(\nu)}}{\partial h_f^{(t)(\nu+1)}} . \tag{4.83}$$

We propose to do just that in this section. Firstly

$$\frac{\partial h_{f'}^{(t')(\nu+1)}}{\partial h_{f}^{(t)(\nu+1)}} = \delta_t^{t'} \delta_f^{f'} \; , \qquad \frac{\partial \hat{h}_{f'}^{(\nu)}}{\partial h_{f}^{(t)(\nu+1)}} = \frac{\delta_f^{f'}}{T_{\rm mb}} \; . \tag{4.84}$$

Secondly

$$\frac{\partial \left(\hat{\sigma}_{f'}^{(\nu)}\right)^2}{\partial h_{f}^{(t)(\nu+1)}} = \frac{2\delta_f^{f'}}{T_{\rm mb}} \left(h_f^{(t)(\nu+1)} - \hat{h}_f^{(\nu)}\right) \; , \tag{4.85}$$

so that we get

$$\begin{aligned}\frac{\partial \tilde{h}_{f'}^{(t)(\nu)}}{\partial h_{f}^{(t)(\nu+1)}} &= \frac{\delta_f^{f'}}{T_{\rm mb}} \left[\frac{T_{\rm mb}\delta_t^{t'} - 1}{\left(\left(\hat{\sigma}_f^{(\nu)}\right)^2 + \epsilon\right)^{\frac{1}{2}}} - \frac{\left(h_f^{(t')(\nu+1)} - \hat{h}_f^{(\nu)}\right)\left(h_f^{(t)(\nu+1)} - \hat{h}_f^{(\nu)}\right)}{\left(\left(\hat{\sigma}_f^{(\nu)}\right)^2 + \epsilon\right)^{\frac{3}{2}}} \right] \\ &= \frac{\delta_f^{f'}}{\left(\left(\hat{\sigma}_f^{(\nu)}\right)^2 + \epsilon\right)^{\frac{1}{2}}} \left[\delta_t^{t'} - \frac{1 + \tilde{h}_f^{(t')(\nu)} \tilde{h}_f^{(t)(\nu)}}{T_{\rm mb}} \right] \; .\end{aligned} \tag{4.86}$$

To ease the notation recall that we denoted

$$\tilde{\gamma}_f^{(\nu)} = \frac{\gamma_f^{(\nu)}}{\left(\left(\hat{\sigma}_f^{(\nu)}\right)^2 + \epsilon\right)^{\frac{1}{2}}} \; . \tag{4.87}$$

so that

$$\frac{\partial y_{f'}^{(t)(\nu)}}{\partial h_{f}^{(t)(\nu+1)}} = \tilde{\gamma}_f^{(\nu)} \delta_f^{f'} \left[\delta_t^{t'} - \frac{1 + \tilde{h}_f^{(t')(\nu)} \tilde{h}_f^{(t)(\nu)}}{T_{\rm mb}} \right] \; . \tag{4.88}$$

4.D FNN ResNet (non standard presentation)

The state of the art architecture of convolutional neural networks (CNN, to be explained in chapter 5) is called ResNet[5]. Its name comes from its philosophy: each hidden layer output y of the network is a small – hence the term residual – modification of its input ($y = x + F(x)$), instead of a total modification ($y = H(x)$) of its input x. This philosophy can be imported to the FNN case. Representing the operations of weight averaging, activation function and batch normalization in the following way

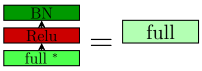


Figure 4.9: Schematic representation of one FNN fully connected layer.

In its non standard form presented in this section, the residual operation amounts to add a skip connection to two consecutive full layers

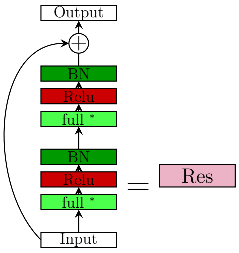


Figure 4.10: Residual connection in a FNN.

Mathematically, we had before (calling the input $y^{(t)(\nu-1)}$)

$$
\begin{aligned}
y_f^{(t)(\nu+1)} &= \gamma_f^{(\nu+1)}\tilde{h}_f^{(t)(\nu+2)} + \beta_f^{(\nu+1)}\,, & a_f^{(t)(\nu+1)} &= \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{(\nu+1)f} y_f^{(t)(\nu)} \\
y_f^{(t)(\nu)} &= \gamma_f^{(\nu)}\tilde{h}_f^{(t)(\nu+1)} + \beta_f^{(\nu)}\,, & a_f^{(t)(\nu)} &= \sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{(\nu)f} y_f^{(t)(\nu-1)}\,,
\end{aligned} \tag{4.89}
$$

as well as $h_f^{(t)(\nu+2)} = g\left(a_f^{(t)(\nu+1)}\right)$ and $h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right)$. In ResNet, we now have the slight modification

$$
y_f^{(t)(\nu+1)} = \gamma_f^{(\nu+1)}\tilde{h}_f^{\nu+2} + \beta_f^{(\nu+1)} + y_f^{(t)(\nu-1)}\,. \tag{4.90}
$$

The choice of skipping two and not just one layer has become a standard for empirical reasons, so as the decision not to weight the two paths (the trivial skip one and the two FNN layer one) by a parameter to be learned by backpropagation

$$y_f^{(t)(\nu+1)} = \alpha \left(\gamma_f^{(\nu+1)} \tilde{h}_f^{(t)(\nu+2)} + \beta_f^{(\nu+1)} \right) + (1-\alpha)\, y_{f'}^{(t)(\nu-1)} \,. \qquad (4.91)$$

This choice is called highway nets[15], and it remains to be theoretically understood why it leads to worst perfomance than ResNet, as the latter is a particular instance of the former. Going back to the ResNet backpropagation algorithm, this changes the gradient through the skip connection in the following way

$$\begin{aligned}
\delta_f^{(t)(\nu-1)} &= \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_\nu-1} \frac{\partial a_{f'}^{(t')(\nu)}}{\partial a_f^{(t)(\nu-1)}} \delta_{f'}^{(t')(\nu)} + \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+2}-1} \frac{\partial a_{f'}^{(t')(\nu+2)}}{\partial a_f^{(t)(\nu-1)}} \delta_{f'}^{(t')(\nu+2)} \\
&= \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_\nu-1} \sum_{f''=0}^{F_{\nu-1}-1} \Theta_{f''}^{(\nu)f'} \frac{\partial y_{f''}^{(t')(\nu-1)}}{\partial a_f^{(t)(\nu-1)}} \delta_{f'}^{(t')(\nu)} \\
&+ \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+2}-1} \sum_{f''=0}^{F_{\nu-1}-1} \Theta_{f''}^{(\nu+2)f'} \frac{\partial y_{f''}^{(t')(\nu+1)}}{\partial a_f^{(t)(\nu-1)}} \delta_{f'}^{(t')(\nu+2)} \\
&= g'\left(a_f^{(t)(\nu-1)}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_\nu-1} \sum_{f''=0}^{F_{\nu-1}-1} \Theta_{f''}^{(\nu)f'} J_f^{(tt')(\nu)} \delta_{f'}^{(t')(\nu)} \\
&+ g'\left(a_f^{(t)(\nu-1)}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+2}-1} \sum_{f''=0}^{F_{\nu-1}-1} \Theta_{f''}^{(\nu+2)f'} J_f^{(tt')(\nu)} \delta_{f'}^{(t')(\nu+2)} \,, \qquad (4.92)
\end{aligned}$$

so that

$$\begin{aligned}
\delta_f^{(t)(\nu-1)} &= g'\left(a_f^{(t)(\nu-1)}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f''=0}^{F_{\nu-1}-1} J_f^{(tt')(\nu)} \\
&\times \left[\sum_{f'=0}^{F_\nu-1} \Theta_{f''}^{(\nu)f'} \delta_{f'}^{(t')(\nu)} + \sum_{f'=0}^{F_{\nu+2}-1} \Theta_{f''}^{(\nu+2)f'} \delta_{f'}^{(t')(\nu+2)} \right] . \qquad (4.93)
\end{aligned}$$

This formulation has one advantage: it totally preserves the usual FNN layer structure of a weight averaging (WA) followed by an activation function (AF) and then a batch normalization operation (BN). It nevertheless has one disadvantage: the backpropagation gradient does not really flow smootly from one error rate to the other. In the following section we will present the standard ResNet formulation of that takes the problem the other way around : it allows the gradient to flow smoothly at the cost of "breaking" the natural FNN building block.

4.E FNN ResNet (more standard presentation)

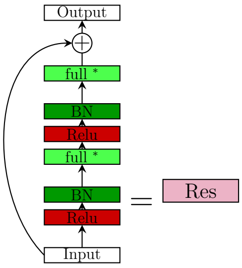


Figure 4.11: Residual connection in a FNN, trivial gradient flow through error rates.

In the more standard form of ResNet, the skip connections reads

$$a_f^{(t)(\nu+2)} = a_f^{(t)(\nu+2)} + a_f^{(t)(\nu)} , \tag{4.94}$$

and the updated error rate reads

$$\delta_f^{(t)(\nu)} = g'\left(a_f^{(t)(\nu)}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f''=0}^{F_\nu-1} J_f^{(tt')(\nu)} \sum_{f'=0}^{F_{\nu+1}-1} \Theta_{f''}^{(\nu+1)f'} \delta_{f'}^{(t')(\nu+1)} + \delta_f^{(t')(\nu+2)} . \tag{4.95}$$

4.F Matrix formulation

In all this chapter, we adopted an "index" formulation of the FNN. This has upsides and downsides. On the positive side, one can take the formula as written here and go implement them. On the downside, they can be quite cumbersome to read.

Another FNN formulation is therefore possible: a matrix one. To do so, one has to rewrite

$$h_f^{(t)(\nu)} \mapsto h_{ft}^{(\nu)} \mapsto h^{(\nu)} \in \mathcal{M}(F_\nu, T_{\rm mb}) . \tag{4.96}$$

In this case the weight averaging procedure (4.18) can be written as

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{(\nu)f} h_{f't}^{(\nu)} \mapsto a^{(\nu)} = \Theta^{(\nu)} h^{(\nu)} . \tag{4.97}$$

The upsides and downsides of this formulation are the exact opposite of the index one: what we gained in readibility, we lost in terms of direct implementation in low level programming languages (C for instance). For FNN, one can use a high level programming language (like python), but this will get quite intractable when we talk about Convolutional networks. Since the whole point of the present work was to introduce the index notation, and as one can easily find numerous derivation of the backpropagation update rules in matrix form, we will stick with the index notation in all the following, and now turn our attention to convolutional networks.

Chapter 5

Convolutional Neural Networks

Contents

5.1 Introduction

In this chapter we review a second type of neural network that is presumably the most popular one: Convolutional Neural Networks (CNN). CNN are particularly adapted for image classification, be it numbers or animal/car/... category. We will review the novelty involved when dealing with CNN when compared to FNN. Among them are the fundamental building blocks of CNN: convolution and pooling. We will in addition see what modification have to be taken into account for the regularization techniques introduced in the FNN part. Finally, we will present the most common CNN architectures that are used in the litterature: from LeNet to ResNet.

5.2 CNN architecture

A CNN is formed by several convolution and pooling operations, usually followed by one or more fully connected layers (those being similar to the traditional FNN layers). We will clarify the new terms introduced thus far in the following sections.

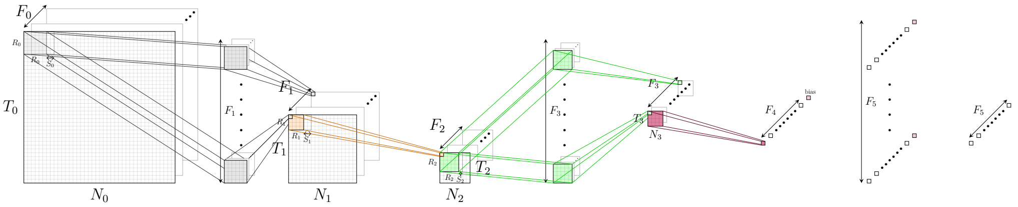


Figure 5.1: A typical CNN architecture (in this case LeNet inspired): convolution operations are followed by pooling operations, until the size of each feature map is reduced to one. Fully connected layers can then be introduced.

5.3 CNN specificities

5.3.1 Feature map

In each layer of a CNN, the data are no longer labeled by a single index as in a FNN. one should see the FNN index as equivalent to the label of a given image in a layer of a CNN. This label is the feature map. In each feature map $f \in [\![0, F_\nu - 1]\!]$ of the ν'th layer, the image is fully characterized by two additional indices corresponding to its height $k \in T_\nu - 1$ and its width $j \in N_\nu - 1$. a given f, j, k thus characterizes a unique pixel of a given feature map. Let us now review the different layers of a CNN

5.3.2 Input layer

We will be considering a input of F_0 channels. In the standard image treatment, these channels can correspond to the RGB colors ($F_0 = 3$). Each image in each channel will be of size $N_0 \times T_0$ (width×height). The input will be denoted $X^{(t)}_{f\,j\,k}$, with $t \in [\![0, T_{\rm mb} - 1]\!]$ (size of the Mini-batch set, see chapter 4), $j \in [\![0, N_0 - 1]\!]$ and $k \in [\![0, T_0 - 1]\!]$. A standard input treatment is to center the data following either one of the two following procedure

$$\tilde{X}^{(t)}_{fjk} = X^{(t)}_{ijk} - \mu_f \,, \qquad\qquad \tilde{X}^{(t)}_{fjk} = X^{(t)}_{ijk} - \mu_{fjk} \tag{5.1}$$

with

$$\mu_f = \frac{1}{T_{\text{train}} T_0 N_0} \sum_{t=0}^{T_{\text{train}}-1} \sum_{j}^{N_0-1} \sum_{k}^{T_0-1} X^{(t)}_{fjk} \,, \tag{5.2}$$

$$\mu_{fjk} = \frac{1}{T_{\text{train}}} \sum_{t=0}^{T_{\text{train}}-1} X^{(t)}_{fjk} \,. \tag{5.3}$$

This correspond to either compute the mean per pixel over the training set, or to also average over every pixel. This procedure should not be followed for regression tasks. To conclude, figure 5.2 shows what the input layer looks like.

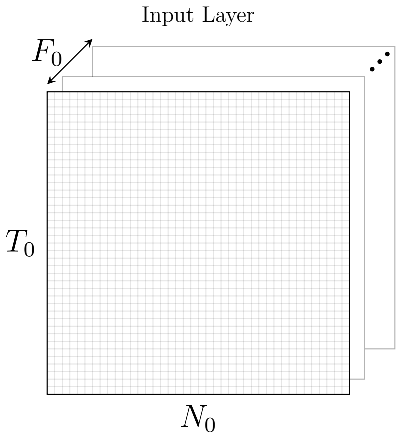


Figure 5.2: The Input layer

5.3.3 Padding

As we will see when we proceed, it may be convenient to "pad" the feature maps in order to preserve the width and the height of the images though several hidden layers. the padding operation amounts to add 0's around the original image. With a padding of size P, we add P zeros at the beginning of each row and column of a given feature map. This is illustrated in the following figure

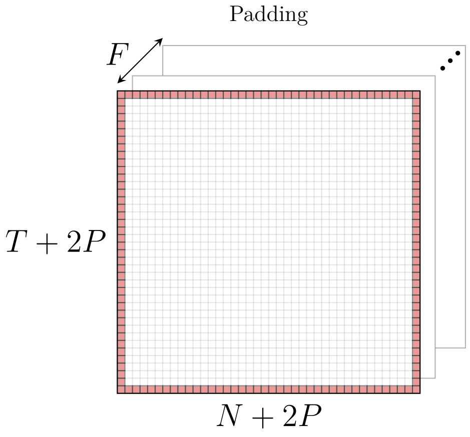


Figure 5.3: Padding of the feature maps. The zeros added correspond to the red tiles, hence a padding of size $P = 1$.

5.3.4 Convolution

The convolution operation that gives its name to the CNN is the fundamental buiding block of this type of network. It amounts to convolute a feature map of an input hidden layer with a weight matrix to give rise to an output feature map. The weight is really a four dimensional tensor, one dimension (F) being the number of feature maps of the convolutional input layer, another (F_p) the number of feature maps of the convolutional output layer. The two others gives the size of the receptive field in the width and the height direction. The receptive field allows one to convolute a subset instead of the whole input image. It aims at searching similar patterns in the input image, no matter where the pattern is (translational invariance). The width and the height of the output image are also determined by the stride: it is simply the number of pixel by which one slides in the vertical and/or the horizontal direction before applying again the convolution operation. A good picture being worth a thousand words, here is the convolution operation in a nuthsell

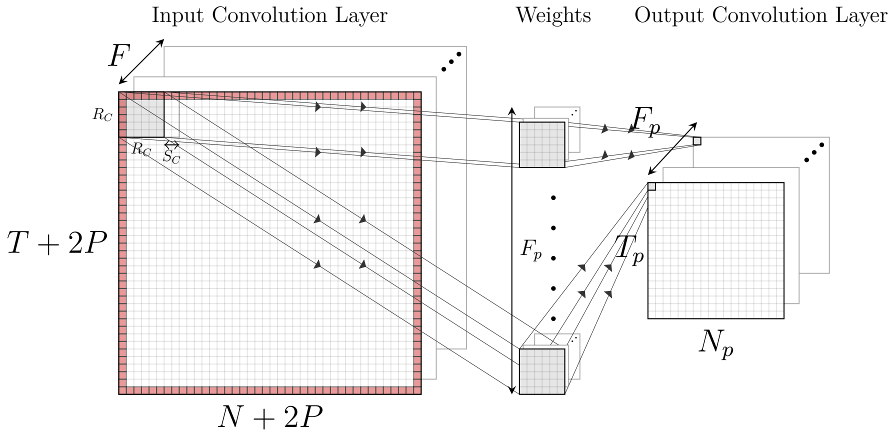


Figure 5.4: The covolution operation

Here R_C is the size of the convolutional receptive field (we will see that the pooling operation also has a receptive field and a stride) and S_C the convolutional stride. The widths and heights of the output image can be computed thanks to the input height T and output width N

$$N_p = \frac{N+2P-R_C}{S_C}+1 \,, \qquad\qquad T_p = \frac{T+2P-R_C}{S_C}+1 \,. \tag{5.4}$$

It is common to introduce a padding to preserve the widths and heights of the input image $N = N_p = T = T_p$, so that in these cases $S_C = 1$ and

$$P = \frac{R_C - 1}{2} \,. \tag{5.5}$$

For a given layer n, the convolution operation mathematically reads (similar in spirit to the weight averaging procedure of a FNN)

$$a^{(t)(\nu)}_{f\,l\,m} = \sum_{\nu=0}^{F_\nu-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{(o)f}_{f\,j\,k} h^{(t)(\nu)}_{f\,S_C l+j\,S_C m+k} \,, \tag{5.6}$$

where o characterizes the $o+1$ convolution in the network. Here ν denotes the ν'th hidden layer of the network (and thus belongs to $\llbracket 0, N-1 \rrbracket$), and $f \in \llbracket 0, F_{\nu+1}-1 \rrbracket$, $l \in \llbracket 0, N_{\nu+1}-1 \rrbracket$ and $m \in \llbracket 0, T_{\nu+1}-1 \rrbracket$. Thus $S_C l + j \in \llbracket 0, N_\nu - 1 \rrbracket$ and $S_C l + j \in \llbracket 0, T_\nu - 1 \rrbracket$. One then obtains the hidden units via a ReLU (or other, see chapter 4) activation function application. Taking padding into account, it reads

$$h^{(t)(\nu+1)}_{f\,l+P\,m+P} = g\left(a^{(t)(\nu)}_{f\,l\,m}\right) \,. \tag{5.7}$$

5.3.5 Pooling

The pooling operation, less and less used in the current state of the art CNN, is fundamentally a dimension reduction opearation. It amounts either to average or to take the maximum of a sub-image – characterized by a pooling receptive field R_P and a stride S_P – of the input feature map F to obtain an output feature map $F_p = F$ of width $N_p < N$ and height $T_p < T$. To be noted: the padded values of the input hidden layers are not taken into account during the pooling operation (hence the $+P$ indices in the following formulas)

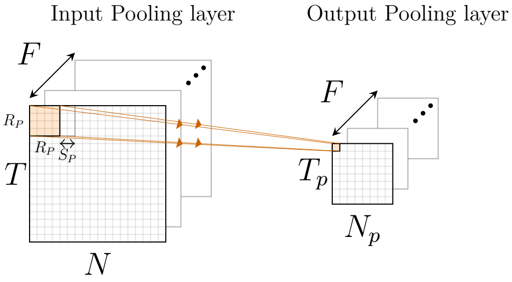


Figure 5.5: The pooling operation

The average pooling procedure reads for a given ν'th pooling operation

$$a^{(t)(\nu)}_{f\,l\,m} = \sum_{j,k=0}^{R_P-1} h^{(t)(\nu)}_{f\,S_Pl+j+P\,S_Pm+k+P}\,, \tag{5.8}$$

while the max pooling reads

$$a^{(t)(\nu)}_{f\,l\,m} = \max_{j,k=0}^{R_P-1} h^{(t)(\nu)}_{f\,S_Pl+j+P\,S_Pm+k+P}\,. \tag{5.9}$$

Here ν denotes the ν'th hidden layer of the network (and thus belongs to $\llbracket 0, N-1 \rrbracket$), and $f \in \llbracket 0, F_{\nu+1}-1 \rrbracket$, $l \in \llbracket 0, N_{\nu+1}-1 \rrbracket$ and $m \in \llbracket 0, T_{\nu+1}-1 \rrbracket$. Thus $S_Pl + j \in \llbracket 0, N_\nu - 1 \rrbracket$ and $S_Pl + j \in \llbracket 0, T_\nu - 1 \rrbracket$. Max pooling is extensively used in the litterature, and we will therefore adopt it in all the following. denoting $j^{(t)(p)}_{flm}$, $k^{(t)(p)}_{flm}$ the indices at which the l,m maximum of the f feature map of the tth batch sample is reached, we have

$$h^{(t)(\nu+1)}_{f\,l+P\,m+P} = a^{(t)(\nu)}_{f\,l\,m} = h^{(t)(\nu)}_{f\,S_Pl+j^{(t)(p)}_{flm}+P\,S_Pm+k^{(t)(p)}_{flm}+P}\,. \tag{5.10}$$

5.3.6 Towards fully connected layers

At some point in a CNN the convolutional receptive field is equal to the width and the height of the input image. In this case, the convolution operation becomes a kind of weight averaging procedure (as in a FNN)

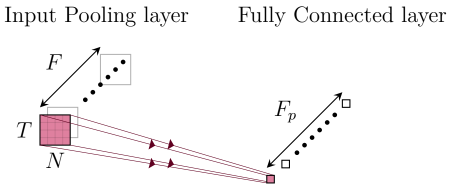


Figure 5.6: Fully connected operation to get images of width and height 1.

This weight averaging procedure reads

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu - 1} \sum_{l=0}^{N-1} \sum_{m=0}^{T-1} \Theta_{f'lm}^{(o)f} h_{f'l+Pm+P}^{(t)(\nu)} , \tag{5.11}$$

and is followed by the activation function

$$h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right) , \tag{5.12}$$

5.3.7 fully connected layers

After the previous operation, the remaining network is just a FNN one. The weigh averaging procedure reads

$$a_f^{(t)(\nu)} = \sum_{f'=0}^{F_\nu - 1} \Theta_{f'}^{(o)f} h_{f'}^{(t)(\nu)} , \tag{5.13}$$

and is followed as usual by the activation function

$$h_f^{(t)(\nu+1)} = g\left(a_f^{(t)(\nu)}\right) , \tag{5.14}$$

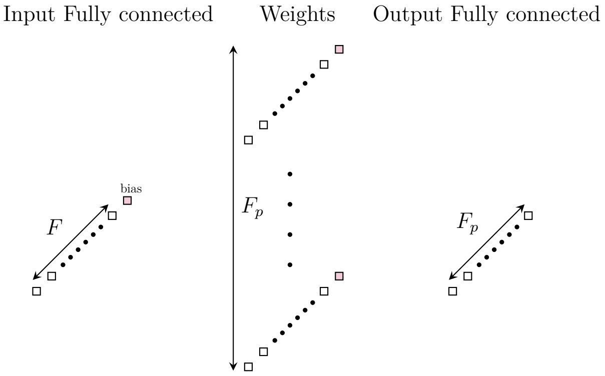


Figure 5.7: Fully connected operation, identical to the FNN operations.

5.3.8 Output connected layer

Finally, the output is computed as in a FNN

$$a_f^{(t)(N-1)} = \sum_{f'=0}^{F_N-1} \Theta_{f'}^{(o)f} h_{f'}^{(t)(N-1)} , \qquad h_f^{(t)(N)} = o\left(a_f^{(t)(N-1)}\right) , \tag{5.15}$$

where as in a FNN, o is either the L2 or the cross-entropy loss function (see chapter 4).

5.4 Modification to Batch Normalization

In a CNN, Batch normalization is modified in the following way (here, contrary to a regular FNN, not all the hidden layers need to be Batch normalized. Indeed this operation is not performed on the output of the pooling layers. We will hence use different names ν and n for the regular and batch normalized hidden layers)

$$\tilde{h}_{f\,l\,m}^{(t)(n)} = \frac{h_{f\,l\,m}^{(t)(\nu)} - \hat{h}_f^{(n)}}{\sqrt{\left(\hat{\sigma}_f^{(n)}\right)^2 + \epsilon}} , \tag{5.16}$$

with

$$\hat{h}_f^{(n)} = \frac{1}{T_{\rm mb}N_nT_n}\sum_{t=0}^{T_{\rm mb}-1}\sum_{l=0}^{N_n-1}\sum_{m=0}^{T_n-1} h_{f\,l\,m}^{(t)(\nu)} \tag{5.17}$$

$$\left(\hat{\sigma}_f^{(n)}\right)^2 = \frac{1}{T_{\rm mb}N_nT_n}\sum_{t=0}^{T_{\rm mb}-1}\sum_{l=0}^{N_n-1}\sum_{m=0}^{T_n-1} \left(h_{f\,l\,m}^{(t)(\nu)} - \hat{h}_f^{(n)}\right)^2 . \tag{5.18}$$

The identity transform can be implemented thanks to the two additional parameters (γ_f, β_f)

$$y_{f\,l\,m}^{(t)(n)} = \gamma_f^{(n)}\,\tilde{h}_{f\,l\,m}^{(t)(n)} + \beta_f^{(n)} . \tag{5.19}$$

For the evaluation of the cross-validation and the test set (calling e the number of iterations/epochs), one has to compute

$$\mathbb{E}\left[h_{f\,l\,m}^{(t)(\nu)}\right]_{e+1} = \frac{e\mathbb{E}\left[h_{f\,l\,m}^{(t)(\nu)}\right]_e + \hat{h}_f^{(n)}}{e+1} , \tag{5.20}$$

$$\mathbb{V}ar\left[h_{f\,l\,m}^{(t)(\nu)}\right]_{e+1} = \frac{i\mathbb{V}ar\left[h_{f\,l\,m}^{(t)(\nu)}\right]_e + \left(\hat{\sigma}_f^{(n)}\right)^2}{e+1} \tag{5.21}$$

and what will be used at test time is $\mathbb{E}\left[h_{f\,l\,m}^{(t)(\nu)}\right]$ and $\frac{T_{\rm mb}}{T_{\rm mb}-1}\mathbb{V}ar\left[h_{f\,l\,m}^{(t)(\nu)}\right]$.

5.5 Network architectures

We will now review the standard CNN architectures that have been used in the litterature in the past 20 years, from old to very recent (end of 2015) ones. To allow for an easy pictural representation, we will adopt the following schematic representation of the different layers.

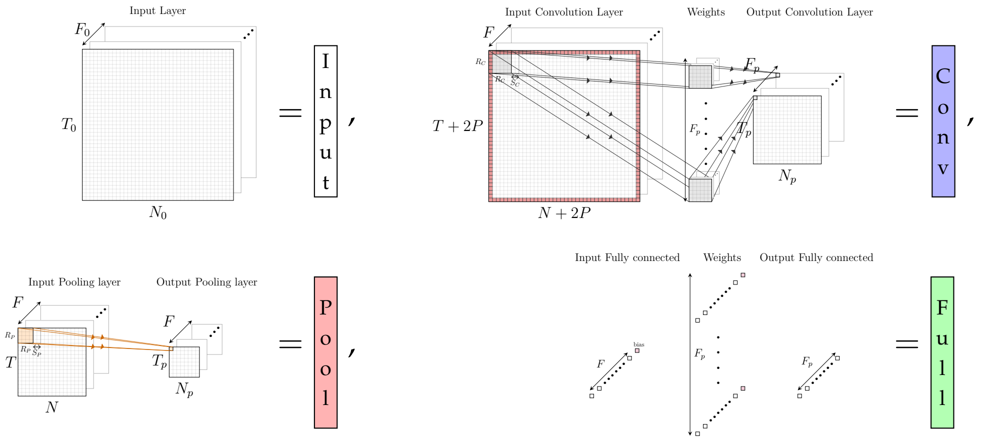


Figure 5.8: Schematic representation of the different layer

5.5.1 Realistic architectures

In realistic architectures, every fully connected layer (except the last one related to the output) is followed by a ReLU (or other) activation and then a batch normalization step (these two data proccessing steps can be inverted, as it was the case in the original BN implementation).

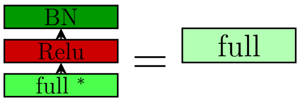


Figure 5.9: Realistic Fully connected operation

The same holds for convolutional layers

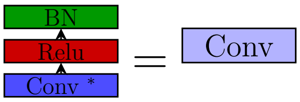


Figure 5.10: Realistic Convolution operation

We will adopt the simplified right hand side representation, keeping in mind that the true structure of a CNN is richer. With this in mind – and mentionning in passing [16] that details recent CNN advances, let us now turn to the first popular CNN used by the deep learning community.

5.5.2 LeNet

The LeNet[3] (end of the 90's) network is formed by an input, followed by two conv-pool layers and then a fully-connected layer before the final output. It can be seen in figure 5.1

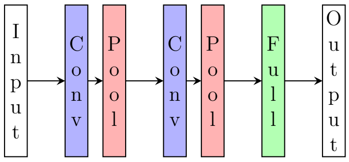


Figure 5.11: The LeNet CNN

When treating large images (224 × 224), this implie to use large size of receptive fields and strides. This has two downsides. Firstly, the number or parameter in a given weight matrix is proportional to the size of the receptive field, hence the larger it is the larger the number of parameter. The network can thus be more prone to overfit. Second, a large stride and receptive field means a less subtle analysis of the fine structures of the images. All the subsequent CNN implementations aim at reducing one of these two issues.

5.5.3 AlexNet

The AlexNet[17] (2012) saw no qualitative leap in the CNN theory, but due to better processors was able to deal with more hidden layers.

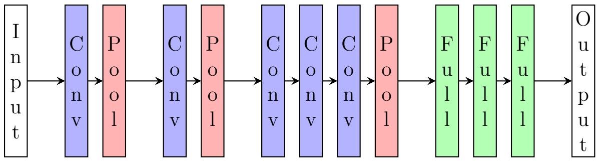


Figure 5.12: The AlexNet CNN

This network is still commonly used, though less since the arrival of the VGG network.

5.5.4 VGG

The VGG[4] network (2014) adopted a simple criteria: only 2×2 paddings of stride 2 and 3×3 convolutions of stride 1 with a padding of size 1, hence preserving the size of the image's width and height through the convolution operations.

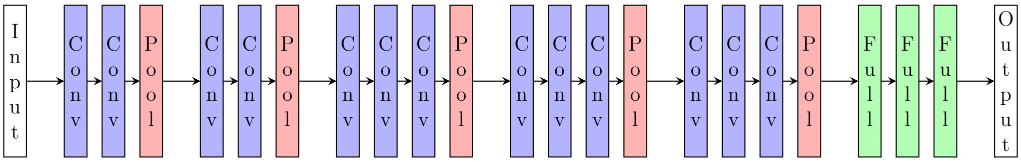


Figure 5.13: The VGG CNN

This network is the standard one in most of the deep learning packages dealing with CNN. It is no longer the state of the art though, as a design innovation has taken place since its creation.

5.5.5 GoogleNet

The GoogleNet[18] introduced a new type of "layer" (wich is in reality a combination of existing layers): the inception layer (in reference to the movie by Christopher Nolan). Instead of passing from one layer of a CNN to the next by a simple pool, conv or fully-connected (fc) operation, one averages the result of the following architecture.

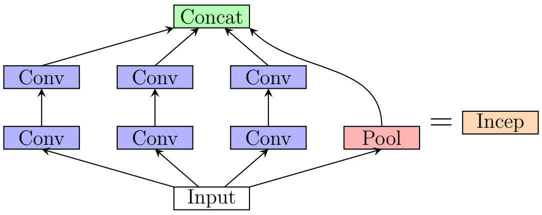


Figure 5.14: The Inception module

We won't enter into the details of the concat layer, as the Google Net illustrated on the following figure is (already!) no longer state of the art.

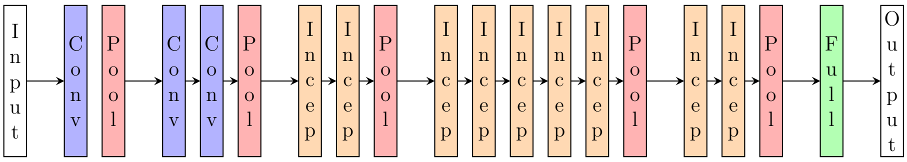


Figure 5.15: The GoogleNet CNN

Indeed, the idea of averaging the result of several conv-pool operations to obtain the next hidden layer of a CNN as been exploited but greatly silmplified by the state of the art CNN : The ResNet.

5.5.6 ResNet

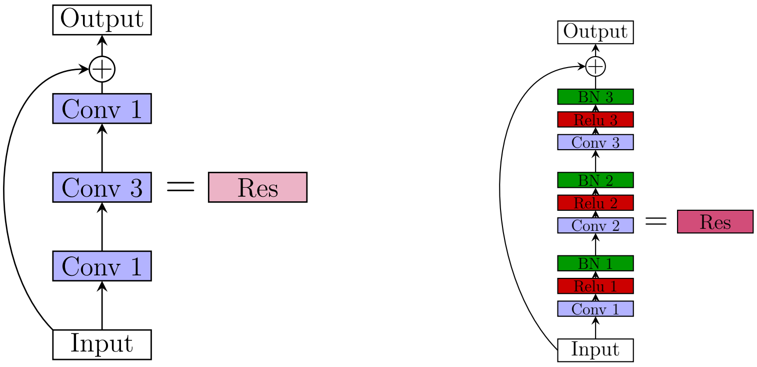


Figure 5.16: The Bottleneck Residual architecture. Schematic representation on the left, realistic one on the right. It amounts to a 1×1 conv of stride 1 and padding 0, then a standard VGG conv and again a 1×1 conv. Two main modifications in our presentation of ResNet: BN operations have been put after ReLU ones, and the final ReLU is before the plus operation.

The ResNet[5] takes back the simple idea of the VGG net to always use the same size for the convolution operations (except for the first one). It also takes into account an experimental fact: the fully connected layer (that usually contains most of the parameters given their size) are not really necessary to perform well. Removing them leads to a great decrease of the number of parameters of a CNN. In addition, the pooling operation is also less and less popular and tend to be replaced by convolution operations. This gives the basic ingredients of the ResNet fundamental building block, the Residual module of figure 5.16.

Two important points have to be mentioned concerning the Residual module. Firstly, a usual conv-conv-conv structure would lead to the following output (forgeting about batch normalization for simplicity and only for the time being, and denoting that there is no need for padding in 1 × 1 convolution operations)

$$\begin{aligned}
h^{(t)(1)}_{f\,l+P\,m+P} &= g\left(\sum_{f'=0}^{F_0-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(0)f}_{f'jk}h^{(t)(0)}_{f'\,S_Cl+j\,S_Cm+k}\right)\\
h^{(t)(2)}_{flm} &= g\left(\sum_{f'=0}^{F_1-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(0)f}_{f'jk}h^{(t)(1)}_{f'\,S_Cl+j\,S_Cm+k}\right)\\
h^{(t)(3)}_{flm} &= g\left(\sum_{f'=0}^{F_2-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(0)f}_{f'jk}h^{(t)(2)}_{f'\,S_Cl+j\,S_Cm+k}\right)\,,
\end{aligned}\tag{5.22}$$

whereas the Residual module modifies the last previous equation to (implying that the width, the size and the number of feature size of the input and the output being the same)

$$\begin{aligned}
h^{(t)(4)}_{flm} &= h^{(t)(0)}_{flm} + g\left(\sum_{f'=0}^{F_2-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(0)f}_{f'jk}h^{(t)(2)}_{f'\,S_Cl+j\,S_Cm+k}\right)\\
&= h^{(t)(0)}_{flm} + \delta h^{(t)(0)}_{flm}\,.
\end{aligned}\tag{5.23}$$

Instead of trying to fit the input, one is trying to fit a tiny modification of the input, hence the name residual. This allows the network to minimally modify the input when necessary, contrary to traditional architectures. Secondly, if the number of feature maps is important, a 3 × 3 convolution with stride 1 could be very costly in term of execution time and prone to overfit (large number of parameters). This is the reason of the presence of the 1 × 1 convolution, whose aim is just to prepare the input to the 3 × 3 conv to reduce the number of feature maps, number which is then restored with the final 1 × 1 conv of the Residual module. The first 1 × 1 convolution thus reads as a weight averaging operation

$$h^{(t)(1)}_{fl+Pm+P} = g\left(\sum_{f'=0}^{F_0-1} \Theta^{(0)f}_{f'} h^{(t)(0)}_{f'l\,m}\right) , \tag{5.24}$$

but is designed such that $f \in F_1 \ll F_0$. The second 1×1 convolution reads

$$h^{(t)(3)}_{flm} = g\left(\sum_{i=0}^{F_1-1} \Theta^{(2)f}_{i} h^{(t)(1)}_{i\,l\,m}\right) , \tag{5.25}$$

with $f \in F_0$, restoring the initial feature map size. The ResNet architecture is then the stacking of a large number (usually 50) of Residual modules, preceded by a conv-pool layer and ended by a pooling operation to obtain a fully connected layer, to which the output function is directly applied. This is illustrated in the following figure.

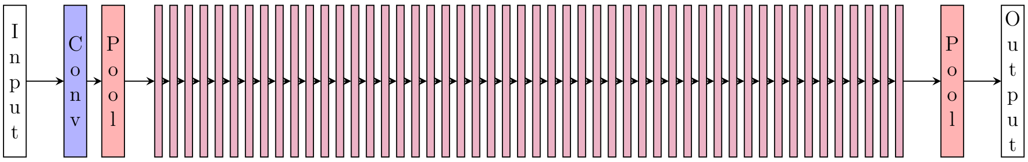


Figure 5.17: The ResNet CNN

The ResNet CNN has accomplished state of the art results on a number of popular training sets (CIFAR, MNIST...). In practice, we will present in the following the bacpropagation algorithm for CNN having standard (like VGG) architectures in mind.

5.6 Backpropagation

In a FNN, one just has to compute two kind of backpropagations : from output to fully connected (fc) layer and from fc to fc. In a traditional CNN, 4 new kind of propagations have to be computed: fc to pool, pool to conv, conv to conv and conv to pool. We present the corresponding error rates in the next sections, postponing their derivation to the appendix. We will consider as in a FNN a network with an input layer labelled 0, N-1 hidden layers labelled i and an output layer labelled N ($N+1$ layers in total in the network).

5.6.1 Backpropagate through Batch Normalization

As in FNN, backpropagation introduces a new gradient

$$\delta^{f}_{f'} J^{(tt')(n)}_{fll'mm'} = \frac{\partial y^{(t')(n)}_{f'\,l'\,m'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} \; . \tag{5.26}$$

we show in appendix 5.A that for pool and conv layers

$$J^{(tt')(n)}_{fll'mm'} = \tilde{\gamma}^{(n)}_{f} \left[\delta^{t'}_{t} \delta^{l'}_{l} \delta^{m'}_{m} - \frac{1 + \tilde{h}^{(t')(n)}_{f\,l'\,m'} \tilde{h}^{(t)(n)}_{f\,l\,m}}{T_{\text{mb}} N_n T_n} \right] \; , \tag{5.27}$$

while we find the FNN result as expected for fc layers

$$J^{(tt')(n)}_{f} = \tilde{\gamma}^{(n)}_{f} \left[\delta^{t'}_{t} - \frac{1 + \tilde{h}^{(t')(n)}_{f} \tilde{h}^{(t)(n)}_{f}}{T_{\text{mb}}} \right] \; . \tag{5.28}$$

5.6.2 Error updates

We will call the specific CNN error rates (depending wether we need padding or not)

$$\delta^{(t)(\nu)}_{fl(+P)m(+P)} = \frac{\partial}{\partial a^{(t)(i)}_{f\,l\,m}} J(\Theta) \; , \tag{5.29}$$

5.6.2.1 Backpropagate from output to fc

Backpropagate from output to fc is schematically illustrated on the following plot

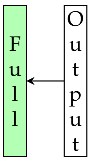


Figure 5.18: Backpropagate from output to fc.

As in FNN, we find for the L2 loss function

$$\delta^{(t)(N-1)}_{f} = \frac{1}{T_{\text{mb}}} \left(h^{(t)(N)}_{f} - y^{(t)}_{f} \right) \; , \tag{5.30}$$

and for the cross-entropy one

$$\delta_f^{(t)(N-1)} = \frac{1}{T_{\rm mb}} \left(h_f^{(t)(N)} - \delta^f_{y^{(t)}} \right) , \tag{5.31}$$

5.6.2.2 Backpropagate from fc to fc

Backpropagate from fc to fc is schematically illustrated on the following plot

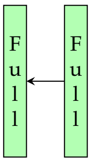


Figure 5.19: Backpropagate from fc to fc.

As in FNN, we find

$$\delta_f^{(t)(\nu)} = g'\left(a_f^{(t)(\nu)}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \Theta_f^{(o)f'} J_f^{(tt')(n)} \delta_{f'}^{(t)(\nu+1)} , \tag{5.32}$$

5.6.2.3 Backpropagate from fc to pool

Backpropagate from fc to pool is schematically illustrated on the following plot

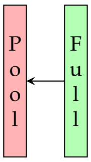


Figure 5.20: Backpropagate from fc to pool.

We show in appendix 5.B that this induces the following error rate

$$\delta_{flm}^{(t)(\nu)} = \sum_{f'=0}^{F_{\nu+1}-1} \Theta_{f\,l\,m}^{(o)f'} \delta_{f'}^{(t)(\nu+1)} , \tag{5.33}$$

5.6.2.4 Backpropagate from pool to conv

Backpropagate from pool to conv is schematically illustrated on the following plot

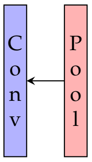


Figure 5.21: Backpropagate from pool to conv.

We show in appendix 5.A that this induces the following error rate (calling the pooling layer the pth one)

$$\delta^{(t)(\nu)}_{fl+Pm+P} = g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'} \times J^{(tt')(n)}_{f\,S_Pl'+j^{(t')(p)}_{fl'm'}+P\,S_Pm'+k^{(t')(p)}_{fl'm'}+P\,l+P\,m+P} . \tag{5.34}$$

Note that we have padded this error rate.

5.6.2.5 Backpropagate from conv to conv

Backpropagate from conv to conv is schematically illustrated on the following plot

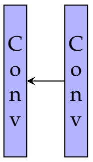


Figure 5.22: Backpropagate from conv to conv.

We show in appendix 5.B that this induces the following error rate

$$\delta^{(t)(\nu)}_{fl+Pm+P} = g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \delta^{(t')(\nu+1)}_{f'l'+Pm'+P} \times \Theta^{(o)f'}_{f\,j\,k}\, J^{(tt')(n)}_{f\,S_Cl'+j\,S_Cm'+k\,l+P\,m+P} \tag{5.35}$$

Note that we have padded this error rate.

5.6.2.6 Backpropagate from conv to pool

Backpropagate from conv to pool is schematically illustrated on the following plot

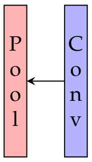


Figure 5.23: Backpropagate from conv to pool.

We show in appendix 5.B that this induces the following error rate

$$\delta^{(t)(\nu)}_{flm} = \sum_{f'=0}^{F_{\nu+1}-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{(o)f}_{f'jk} \delta^{(t)(\nu+1)}_{f\,\frac{l+P-j}{S_C}+P\,\frac{m+P-k}{S_C}+P} \,. \tag{5.36}$$

5.6.3 Weight update

For the weight updates, we will also consider separately the weights between fc to fc layer, fc to pool, conv to conv, conv to pool and conv to input.

5.6.3.1 Weight update from fc to fc

For the two layer interactions

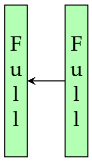


Figure 5.24: Weight update between two fc layers.

We have the weight update that reads

$$\Delta^{\Theta(o)f}_{f'} = \sum_{t=0}^{T_{\rm mb}-1} y^{(t)(n)}_{f'} \delta^{(t)(\nu)}_{f} \tag{5.37}$$

5.6.3.2 Weight update from fc to pool

For the two layer interactions

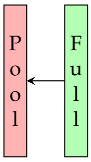


Figure 5.25: Weight update between a fc layer and a pool layer.

We have the weight update that reads

$$\Delta^{\Theta(o)f}_{f'jk} = \sum_{t=0}^{T_{\text{mb}}-1} h^{(t)(\nu)}_{f'j+Pk+P}\delta^{(t)(\nu)}_{f} \tag{5.38}$$

5.6.3.3 Weight update from conv to conv

For the two layer interactions

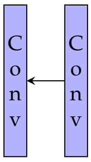


Figure 5.26: Weight update between two conv layers.

We have the weight update that reads

$$\Delta^{\Theta(o)f}_{f'jk} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{l=0}^{T_{\nu+1}-1}\sum_{m=0}^{N_{\nu+1}-1} y^{(t)(n)}_{f'\,l+j\,m+k}\delta^{(t)(\nu)}_{f\,l+P\,m+P} \tag{5.39}$$

5.6.3.4 Weight update from conv to pool and conv to input

For the two layer interactions

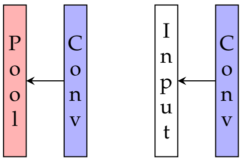


Figure 5.27: Weight update between a conv and a pool layer, as well as between a conv and the input layer.

$$\Delta^{\Theta(o)f}_{f'jk} = \sum_{t=0}^{T_{\rm mb}-1} \sum_{l=0}^{T_{\nu+1}-1} \sum_{m=0}^{N_{\nu+1}-1} h^{(t)(\nu)}_{f'\,l+j\,m+k} \delta^{(t)(\nu)}_{f\,l+P\,m+P} \tag{5.40}$$

5.6.4 Coefficient update

For the Coefficient updates, we will also consider separately the weights between fc to fc layer, fc to pool, cont to pool and conv to conv .

5.6.4.1 Coefficient update from fc to fc

For the two layer interactions

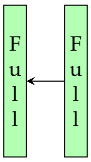


Figure 5.28: Coefficient update between two fc layers.

We have

$$\begin{aligned}
\Delta^{\gamma(n)}_{f} &= \sum_{t=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \Theta^{(o)f'}_{f} \tilde{h}^{(t)(n)}_{f} \delta^{(t)(\nu)}_{f'} \,, \\
\Delta^{\beta(n)}_{f} &= \sum_{t=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \Theta^{(o)f'}_{f} \delta^{(t)(\nu)}_{f'} \,,
\end{aligned} \tag{5.41}$$

5.6.4.2 Coefficient update from fc to pool and conv to pool

For the two layer interactions

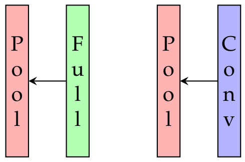


Figure 5.29: Coefficient update between a fc layer and a pool as well as a conv and a pool layer.

$$\begin{aligned}
\Delta_f^{\gamma(n)} &= \sum_{t=0}^{T_{\rm mb}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1} \tilde{h}^{(t)(n)}_{f\,S_Pl+j^{(t)(p)}_{flm}+P\,S_Pm+k^{(t)(p)}_{flm}+P}\delta^{(t)(\nu)}_{flm}\,,\\
\Delta_f^{\beta(n)} &= \sum_{t=0}^{T_{\rm mb}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1} \delta^{(t)(\nu)}_{flm}\,, \qquad (5.42)
\end{aligned}$$

5.6.4.3 Coefficient update from conv to conv

For the two layer interactions

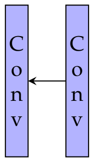


Figure 5.30: Coefficient update between two conv layers.

We have

$$\begin{aligned}
\Delta_f^{\gamma(n)} &= \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{fjk}\,\tilde{h}^{(t)(n)}_{fl+j\,m+k}\delta^{(t)(\nu)}_{f'\,l+P\,m+P}\,,\\
\Delta_f^{\beta(n)} &= \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{fjk}\,\delta^{(t)(\nu)}_{f'\,l+P\,m+P}\,. \qquad (5.43)
\end{aligned}$$

Let us now demonstrate all these formulas!

Appendix

5.A Backprop through BatchNorm

For Backpropagation, we will need

$$\frac{\partial y^{(t')(n)}_{f'\,l'\,m'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} = \gamma_f^{(n)} \frac{\partial \tilde{h}^{(t')(n)}_{f'\,l'\,m'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} \,. \tag{5.44}$$

Since

$$\frac{\partial h^{(t')(\nu)}_{f'\,l'\,m'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} = \delta^{t'}_{t} \delta^{f'}_{f} \delta^{l'}_{l} \delta^{m'}_{m} \,, \qquad \frac{\partial \hat{h}^{(n)}_{f'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} = \frac{\delta^{f'}_{f}}{T_{\mathrm{mb}} N_n T_n} \,; \tag{5.45}$$

and

$$\frac{\partial \left(\hat{\sigma}^{(n)}_{f'}\right)^2}{\partial h^{(t)(\nu)}_{f\,l\,m}} = \frac{2\delta^{f'}_{f}}{T_{\mathrm{mb}} N_n T_n} \left(h^{(t)(\nu)}_{f\,l\,m} - \hat{h}^{(n)}_{f}\right) \,, \tag{5.46}$$

we get

$$\begin{aligned}
\frac{\partial \tilde{h}^{(t')(n)}_{f'\,l'\,m'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} &= \frac{\delta^{f'}_{f}}{T_{\mathrm{mb}} N_n T_n} \left[\frac{T_{\mathrm{mb}} N_n T_n \delta^{t'}_{t} \delta^{l'}_{l} \delta^{m'}_{m} - 1}{\left(\left(\hat{\sigma}^{(n)}_{f}\right)^2 + \epsilon\right)^{\frac{1}{2}}} - \frac{\left(h^{(t')(\nu)}_{f\,l'\,m'} - \hat{h}^{(n)}_{f}\right)\left(h^{(t)(\nu)}_{f\,l\,m} - \hat{h}^{(n)}_{f}\right)}{\left(\left(\hat{\sigma}^{(n)}_{f}\right)^2 + \epsilon\right)^{\frac{3}{2}}} \right] \\
&= \frac{\delta^{f'}_{f}}{\left(\left(\hat{\sigma}^{(n)}_{f}\right)^2 + \epsilon\right)^{\frac{1}{2}}} \left[\delta^{t'}_{t} \delta^{l'}_{l} \delta^{m'}_{m} - \frac{1 + \tilde{h}^{(t')(n)}_{f\,l'\,m'} \tilde{h}^{(t)(n)}_{f\,l\,m}}{T_{\mathrm{mb}} N_n T_n} \right] \,.
\end{aligned} \tag{5.47}$$

To ease the notation we will denote

$$\tilde{\gamma}^{(n)}_{f} = \frac{\gamma^{(n)}_{f}}{\left(\left(\hat{\sigma}^{(n)}_{f}\right)^2 + \epsilon\right)^{\frac{1}{2}}} \,. \tag{5.48}$$

so that

$$\delta^{f}_{f'} J^{(tt')(n)}_{f\,l\,m\,l'\,m'} = \frac{\partial y^{(t')(n)}_{f'\,l'\,m'}}{\partial h^{(t)(\nu)}_{f\,l\,m}} = \tilde{\gamma}^{(n)}_{f} \delta^{f'}_{f} \left[\delta^{t'}_{t} \delta^{l'}_{l} \delta^{m'}_{m} - \frac{1 + \tilde{h}^{(t')(n)}_{f\,l'\,m'} \tilde{h}^{(t)(n)}_{f\,l\,m}}{T_{\mathrm{mb}} N_n T_n} \right] \,. \tag{5.49}$$

5.B Error rate updates: details

we have for backpropagation from fc to pool

$$\begin{aligned}
\delta^{(t)(\nu)}_{flm} &= \frac{\partial}{\partial a^{(t)(\nu)}_{flm}} J^{(t)}(\Theta) = \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \frac{\partial a^{(t')(\nu+1)}_{f'}}{\partial a^{(t)(\nu)}_{flm}} \delta^{(t')(\nu+1)}_{f'} \\
&= \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \sum_{f''=0}^{F_{\nu}-1} \sum_{j=0}^{N_{\nu+1}} \sum_{k=0}^{T_{\nu+1}} \Theta^{(o)f'}_{f''\,j\,k} \frac{\partial h^{(t)(\nu+1)}_{f''j+Pk+P}}{\partial h^{(t)(\nu+1)}_{fl+Pm+P}} \delta^{(t')(\nu+1)}_{f'} \\
&= \sum_{f'=0}^{F_{\nu+1}-1} \Theta^{(o)f'}_{f\,l\,m} \delta^{(t)(\nu+1)}_{f'} \;,
\end{aligned} \tag{5.50}$$

for backpropagation from pool to conv

$$\begin{aligned}
\delta^{(t)(\nu)}_{fl+Pm+P} &= \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \frac{\partial a^{(t')(\nu+1)}_{f'l'm'}}{\partial a^{(t)(\nu)}_{f\,l\,m}} \delta^{(t')(\nu+1)}_{f'l'm'} = \\
&= \sum_{t'=0}^{T_{\rm mb}-1} \sum_{f'=0}^{F_{\nu+1}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \frac{\partial y^{(t')(n)}_{f'\,S_P l'+j^{(t')(p)}_{f'l'm'}+P\,S_P m'+k^{(t')(p)}_{f'l'm'}+P}}{\partial h^{(t)(\nu+1)}_{f\,l+P\,m+P}} g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \delta^{(t')(\nu+1)}_{f'l'm'} \\
&= \tilde{\gamma}^{(n)}_{f} g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'} \\
&\left[\delta^{t'}_{t} \delta^{S_P l'+j^{(p)}_{t'fl'm'}}_{l} \delta^{S_P m'+k^{(p)}_{t'fl'm'}}_{m} - \frac{1+\tilde{h}^{(t')(n)}_{f\,S_P l'+j^{(t')(p)}_{fl'm'}+P\,S_P m'+k^{(t')(p)}_{fl'm'}+P} \tilde{h}^{(t)(n)}_{f\,l+P\,m+P}}{T_{\rm mb} N_n T_n} \right] \\
&= g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'} \\
&\times J^{(tt')(n)}_{f\,S_P l'+j^{(t')(p)}_{fl'm'}+P\,S_P m'+k^{(t')(p)}_{fl'm'}+P\,l+P\,m+P}
\end{aligned} \tag{5.51}$$

for backpropagation from conv to conv

$$\begin{aligned}
\delta^{(t)(\nu)}_{fl+Pm+P} &= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\frac{\partial a^{(t')(\nu+1)}_{f'l'm'}}{\partial a^{(t)(\nu)}_{f\,l\,m}}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\sum_{f''=0}^{F_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(o)f'}_{f''jk}\\
&\times\frac{\partial y^{(t')(n)}_{f''\,l'+j\,m'+k}}{\partial h^{(t)(\nu+1)}_{f\,l+P\,m+P}}g'\left(a^{(t)(\nu)}_{f\,l\,m}\right)\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\,,
\end{aligned}\tag{5.52}$$

so

$$\begin{aligned}
\delta^{(t)(\nu)}_{fl+Pm+P} &= \tilde{\gamma}^{(n)}_f g'\left(a^{(t)(\nu)}_{f\,l\,m}\right)\sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(o)f'}_{f\,j\,k}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&\times\left[\delta^{t'}_{t}\delta^{l'+j}_{l+P}\delta^{m'+k}_{m+P}-\frac{1+\tilde{h}^{(t')(n)}_{f\,l'+j\,m'+k}\tilde{h}^{(t)(n)}_{f\,l+P\,m+P}}{T_{\rm mb}N_nT_n}\right]\\
&= g'\left(a^{(t)(\nu)}_{f\,l\,m}\right)\sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&\times\Theta^{(o)f'}_{f\,j\,k}J^{(tt')(n)}_{f\,l'+j\,m'+k\,l+P\,m+P}\,,
\end{aligned}\tag{5.53}$$

and for backpropagation from conv to pool (taking the stride equal to 1 to simplify the derivation)

$$\begin{aligned}
\delta^{(t)(\nu)}_{flm} &= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\frac{\partial a^{(t')(\nu+1)}_{f'l'm'}}{\partial a^{(t)(\nu)}_{f\,l\,m}}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\sum_{f''=0}^{F_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(o)f'}_{f''jk}\frac{\partial h^{(t')(\nu+1)}_{f''\,l'+j\,m'+k}}{\partial h^{(t)(\nu+1)}_{f\,l+P\,m+P}}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&= \sum_{f'=0}^{F_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(o)f'}_{f\,j\,k}\delta^{(t)(\nu+1)}_{f'l+2P-j\,m+2P-k}\,.
\end{aligned}\tag{5.54}$$

And so on and so forth.

5.C Weight update: details

Fc to Fc

$$\Delta^{\Theta(o)f}_{f'} = \frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \sum_{f''=0}^{F_{\nu+1}-1} \sum_{f'''=0}^{F_\nu} \frac{\partial \Theta^{(o)f''}_{f'''}}{\partial \Theta^{(o)f}_{f'}} y^{(t)(n)}_{f'''} \delta^{(t)(\nu)}_{f''} = \sum_{t=0}^{T_{\rm mb}-1} \delta^{(t)(\nu)}_{f} y^{(t)(n)}_{f'} \ . \tag{5.55}$$

Fc to pool

$$\begin{aligned}\Delta^{\Theta(o)f}_{f'jk} &= \frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \sum_{f''=0}^{F_{\nu+1}-1} \sum_{f'''=0}^{F_\nu} \sum_{j'=0}^{N_{\nu+1}} \sum_{k'=0}^{T_{\nu+1}} \frac{\partial \Theta^{(13)f''}_{f'''j'k'}}{\partial \Theta^{(o)f}_{f'jk}} h^{(t)(\nu)}_{f'''j'+Pk'+P} \delta^{(t)(\nu)}_{f''} \\ &= \sum_{t=0}^{T_{\rm mb}-1} \delta^{(t)(\nu)}_{f} h^{(t)(\nu)}_{f'j+Pk+P} \ .\end{aligned} \tag{5.56}$$

and for conv to conv

$$\begin{aligned}\Delta^{\Theta(o)f}_{f'jk} &= \sum_{t=0}^{T_{\rm mb}-1} \sum_{f''=0}^{F_{\nu+1}-1} \sum_{l=0}^{T_{\nu+1}-1} \sum_{m=0}^{N_{\nu+1}-1} \frac{\partial a^{(t)(\nu)}_{f''\,l\,m}}{\partial \Theta^{(o)f}_{f'\,j\,k}} \delta^{(t)(\nu)}_{f''\,l+P\,m+P} \\ &= \sum_{t=0}^{T_{\rm mb}-1} \sum_{f''=0}^{F_{\nu+1}-1} \sum_{l=0}^{T_{\nu+1}-1} \sum_{m=0}^{N_{\nu+1}-1} \sum_{f'''=0}^{F_{\nu+1}-1} \sum_{j'=0}^{R_C-1} \sum_{k'=0}^{R_C-1} \frac{\partial \Theta^{(o)f''}_{f'''\,j'\,k'}}{\partial \Theta^{(o)f}_{f'\,j\,k}} \\ &\times y^{(t)(n)}_{f'''\,S_C l+j'\,S_C m+k'} \delta^{(t)(\nu)}_{f''\,l+P\,m+P} \\ &= \sum_{t=0}^{T_{\rm mb}-1} \sum_{l=0}^{T_{\nu+1}-1} \sum_{m=0}^{N_{\nu+1}-1} y^{(t)(n)}_{f'\,S_C l+j\,S_C m+k} \delta^{(t)(\nu)}_{f\,l+P\,m+P} \ .\end{aligned} \tag{5.57}$$

similarly for conv to pool and conv to input

$$\Delta^{\Theta(o)f}_{f'jk} = \sum_{t=0}^{T_{\rm mb}-1} \sum_{l=0}^{T_{\nu+1}-1} \sum_{m=0}^{N_{\nu+1}-1} h^{(t)(\nu)}_{f'\,S_C l+j\,S_C m+k} \delta^{(t)(\nu)}_{f\,l+P\,m+P} \ . \tag{5.58}$$

5.D Coefficient update: details

Fc to Fc

$$\Delta^{\gamma(n)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1} \frac{\partial a^{(t)(\nu+1)}_{f'}}{\partial \gamma^{(n)}_{f}} \delta^{(t)(\nu+1)}_{f'} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1} \Theta^{(o)f'}_{f} \tilde{h}^{(t)(n)}_{f} \delta^{(t)(\nu+1)}_{f'} \,, \tag{5.59}$$

$$\Delta^{\beta(n)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1} \frac{\partial a^{(t)(\nu+1)}_{f'}}{\partial \beta^{(n)}_{f}} \delta^{(t)(\nu+1)}_{f'} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1} \Theta^{(o)f'}_{f} \delta^{(t)(\nu+1)}_{f'} \,, \tag{5.60}$$

fc to pool and conv to pool

$$\Delta^{\gamma(n)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1} \tilde{h}^{(t)(n)}_{f\,S_P l + j^{(t)(p)}_{flm} + P\, S_P m + k^{(t)(p)}_{flm} + P} \delta^{(t)(\nu+1)}_{flm} \tag{5.61}$$

$$\Delta^{\beta(n)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1} \delta^{(t)(\nu+1)}_{flm} \,, \tag{5.62}$$

conv to conv

$$\Delta^{\gamma(n)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1} \frac{a^{(t)(\nu+1)}_{f'lm}}{\partial \gamma^{(n)}_{f}} \delta^{(t)(\nu+1)}_{f'lm} \tag{5.63}$$

$$= \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{fjk} \tilde{h}^{(t)(n)}_{fl+j\,m+k} \delta^{(t)(\nu+1)}_{f'lm} \tag{5.64}$$

$$\Delta^{\beta(n)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{fjk} \delta^{(t)(\nu+1)}_{f'lm} \,, \tag{5.65}$$

5.E Practical Simplification

When implementing a CNN, it turns out that some of the error rate computation can be very costly (in term of execution time) if naively encoded. In this section, we sketch some improvement that can be performed on the pool to conv, conv to conv error rate implementation, as well as ones on coefficient updates.

5.E.1 pool to conv Simplification

Let us expand the batch normalization term of the pool to conv error rate to see how we can simplify it (calling the pooling the pth one)

$$\delta^{(t)(\nu)}_{flm} = \tilde{\gamma}^{(n)}_f g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \sum_{t'=0}^{T_{\rm mb}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'} \left[\delta^{t'}_t \delta_l^{S_P l'+j^{(p)}_{t'fl'm'}} \delta_m^{S_P m'+k^{(p)}_{t'fl'm'}} - \frac{1+\tilde{h}^{(t')(n)}_{f\,S_P l'+j^{(t')(p)}_{fl'm'}+P\,S_P m'+k^{(t')(p)}_{fl'm'}+P}\tilde{h}^{(t)(n)}_{f\,l+P\,m+P}}{T_{\rm mb}N_n T_n}\right]. \tag{5.66}$$

Numerically, this implies that for each t, f, l, m one needs to perform 3 loops (on t', l', m'), hence a 7 loop process. This can be reduced to 4 at most in the following way. Defining

$$\mu^{(1)}_f = \sum_{t'=0}^{T_{\rm mb}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'}\,, \tag{5.67}$$

and

$$\mu^{(2)}_f = \sum_{t'=0}^{T_{\rm mb}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'} \tilde{h}^{(t')(n)}_{f\,S_P l'+j^{(t')(p)}_{fl'm'}+P\,S_P m'+k^{(t')(p)}_{fl'm'}+P}\,, \tag{5.68}$$

we have introduced two new variables that can be computed in four loops, but three of them are independent of the ones needed to compute $\delta^{(t)(\nu)}_{flm}$. For the last term, the δ functions "kill" 3 loops and we are left with

$$\delta^{(t)(\nu)}_{flm} = \tilde{\gamma}^{(n)}_f g'\left(a^{(t)(\nu)}_{f\,l\,m}\right) \left\{ \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{fl'm'} \delta_l^{S_P l'+j^{(p)}_{t'fl'm'}} \delta_m^{S_P m'+k^{(p)}_{t'fl'm'}} - \frac{\mu^{(1)}_f + \mu^{(2)}_f \tilde{h}^{(t)(n)}_{f\,l+P\,m+P}}{T_{\rm mb}N_n T_n} \right\}, \tag{5.69}$$

which requires only 4 loops to be computed.

5.E.2 Convolution Simplification

Let us expand the batch normalization term of the conv to conv error rate to see how we can simplify it

$$\delta^{(t)(\nu)}_{flm} = \tilde{\gamma}^{(n)}_f g'\left(a^{(t)(iv)}_{f\,l\,m}\right) \sum_{t'=0}^{T_{\text{mb}}-1} \sum_{f'=0}^{F_{\nu+1}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{f\,j\,k} \delta^{(t')(\nu+1)}_{f'l'm'} \left[\delta^{t'}_t \delta^{l'+j}_{l+P} \delta^{m'+k}_{m+P} - \frac{1 + \tilde{h}^{(t')(n)}_{f\,l'+j\,m'+k} \tilde{h}^{(t)(n)}_{f\,l+P\,m+P}}{T_{\text{mb}} N_n T_n}\right] . \tag{5.70}$$

If left untouched, one now needs 10 loops (on t, f, l, m and t', f', l', m', j, k) to compute $\delta^{(t)(\nu)}_{flm}$! This can be reduced to 7 loops at most in the following way. First, we define

$$\lambda^{(t)(1)}_{flm} = \sum_{f'=0}^{F_{\nu+1}-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{f\,j\,k} \delta^{(t')(\nu+1)}_{f'l+P-j\,m+P-k} , \tag{5.71}$$

which is a convolution operation on a shifted index $\delta^{(\nu+1)}$. This is second most expensive operation, but librairies are optimized for this and it implies two of the smallest loops (those on R_C). Then we compute (four loops each)

$$\lambda^{(2)}_{ff'} = \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{f\,j\,k} , \qquad \lambda^{(3)}_{f'} = \sum_{t'=0}^{T_{\text{mb}}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{f'l'm'} , \tag{5.72}$$

and (2 loops)

$$\lambda^{(4)}_f = \sum_{f'=0}^{F_{\nu+1}-1} \lambda^{(2)}_{ff'} \lambda^{(3)}_{f'} . \tag{5.73}$$

Finally, we compute

$$\lambda^{(5)}_{ff'jk} = \sum_{t'=0}^{T_{\text{mb}}-1} \sum_{l'=0}^{N_{\nu+1}-1} \sum_{m'=0}^{T_{\nu+1}-1} \delta^{(t')(\nu+1)}_{f'l'm'} \tilde{h}^{(t')(n)}_{f\,l'+j\,m'+k} , \tag{5.74}$$

$$\lambda^{(6)}_f = \sum_{f'=0}^{F_{\nu+1}-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{(o)f'}_{f\,j\,k} \lambda^{(5)}_{ff'jk} . \tag{5.75}$$

$\lambda^{(6)}_f$ only requires four loops, and $\lambda^{(5)}_{ff'jk}$ is the most expensive operation. but it is also a convolution operation that implies two of the smalles loops (those on

R_C). With all these newly introduced λ, we obtain

$$\delta^{(t)(\nu)}_{flm} = \tilde{\gamma}^{(n)}_f g'\left(a^{(t)(\nu)}_{f\,l\,m}\right)\left\{\lambda^{(t)(1)}_{flm} - \frac{\lambda^{(4)}_f + \lambda^{(6)}_f \tilde{h}^{(t)(n)}_{f\,l+P\,m+P}}{T_{\text{mb}}N_n T_n}\right\}, \tag{5.76}$$

which only requires four loops to be computed.

5.E.3 Coefficient Simplification

To compute

$$\Delta^{\gamma(n)}_f = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(o)f'}_{fjk}\tilde{h}^{(t)(n)}_{fl+j\,m+k}\delta^{(t)(\nu+1)}_{f'lm} \tag{5.77}$$

$$\Delta^{\beta(n)}_f = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\Theta^{(o)f'}_{fjk}\delta^{(t)(\nu+1)}_{f'lm}, \tag{5.78}$$

we will first define

$$\nu^{(1)}_{f'fjk} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\tilde{h}^{(t)(n)}_{fl+j\,m+k}\delta^{(t)(\nu+1)}_{f'lm}, \quad \nu^{(2)}_{f'} = \sum_{t=0}^{T_{\text{mb}}-1}\sum_{l=0}^{N_{\nu+1}-1}\sum_{m=0}^{T_{\nu+1}-1}\delta^{(t)(\nu+1)}_{f'lm}, \tag{5.79}$$

so that

$$\Delta^{\gamma(n)}_f = \sum_{f'=0}^{F_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\nu^{(1)}_{f'fjk}\Theta^{(o)f'}_{fjk} \tag{5.80}$$

$$\Delta^{\beta(n)}_f = \sum_{f'=0}^{F_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\nu^{(2)}_{f'}\Theta^{(o)f'}_{fjk}, \tag{5.81}$$

5.F Batchpropagation through a ResNet module

For pedagogical reasons, we introduced the ResNet structure without "breaking" the conv layers. Nevertheless, a more standard choice is depicted in the following figure

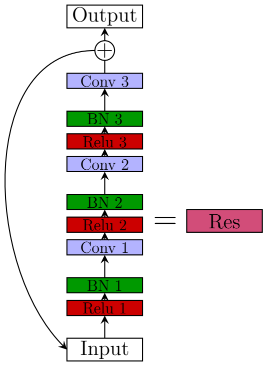


Figure 5.31: Batchpropagation through a ResNet module.

Batchpropagation through this ResNet module presents no particular difficulty. Indeed, the update rules imply the usual conv to conv backpropagation derived in the main part of this note. The only novelty is the error rate update of the input layer of the ResNet, as it now reads (assuming that the input of this ResNet module is the output of another one)

$$\begin{aligned}
\delta^{(t)(\nu)}_{fl+Pm+P} &= \sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\frac{\partial a^{(t')(\nu+1)}_{f'l'm'}}{\partial a^{(t)(\nu)}_{f\,l\,m}}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&+\sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+3}-1}\sum_{l'=0}^{N_{\nu+3}-1}\sum_{m'=0}^{T_{\nu+3}-1}\frac{\partial a^{(t')(\nu+3)}_{f'l'm'}}{\partial a^{(t)(\nu)}_{f\,l\,m}}\delta^{(t')(\nu+3)}_{f'l'+Pm'+P}\\
&= g'\left(a^{(t)(\nu)}_{f\,l\,m}\right)\sum_{t'=0}^{T_{\rm mb}-1}\sum_{f'=0}^{F_{\nu+1}-1}\sum_{l'=0}^{N_{\nu+1}-1}\sum_{m'=0}^{T_{\nu+1}-1}\sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1}\delta^{(t')(\nu+1)}_{f'l'+Pm'+P}\\
&\times\Theta^{(o)f'}_{f\,j\,k}J^{(tt')(n)}_{f\,l'+j\,m'+k\,l+P\,m+P}+\delta^{(t)(\nu+3)}_{fl+Pm+P}\,. \qquad (5.82)
\end{aligned}$$

This new term is what allows the error rate to flow smoothly from the output to the input in the ResNet CNN, as the additional connexion in a ResNet is like a skip path to the convolution chains. Let us mention in passing that some architecture connects every hidden layer to each others[19].

5.G Convolution as a matrix multiplication

Thanks to the simplifications introduced in appendix 5.E, we have reduced all convolution, pooling and tensor multiplication operations to at most 7 loops operations. Nevertheless, in high abstraction programming languages such as python, it is still way too much to be handled smoothly. But there exists additional tricks to "reduce" the dimension of the convolution operation, such as one has only to encode three for loops (2D matrix multiplication) at the end of the day. Let us begin our presentation of these tricks with a 2D convolution example

5.G.1 2D Convolution

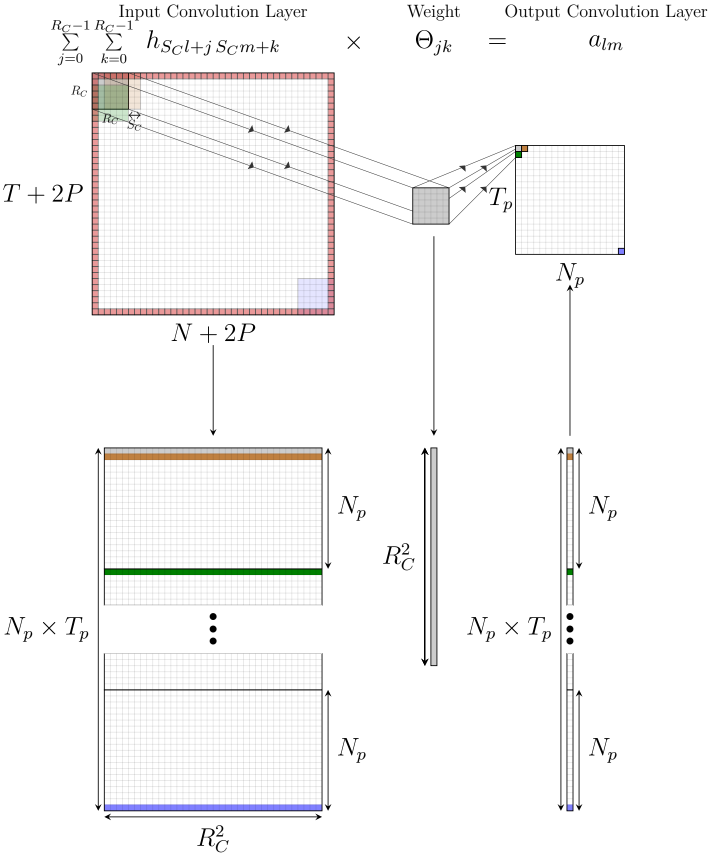


Figure 5.32: :2D convolution as a 2D matrix multiplication

A 2D convolution operation reads

$$a_{lm} = \sum_{j=0}^{R_C-1}\sum_{k=0}^{R_C-1} \Theta_{jk} h_{S_Cl+j\,S_Cm+k} \,. \tag{5.83}$$

and it involves 4 loops. the trick to reduce this operation to a 2D matrix multiplication is to redefine the h matrix by looking at each h indices are going to be multiplied by Θ for each value of l and m. Then the assiociated h values are stored into a $N_pT_p \times R_C^2$ matrix. flatening out the Θ matrix, we are left with a matrix multiplication, the flattened Θ matrix being of size $R_C^2 \times 1$. This is illustrated on figure 5.32

5.G.2 4D Convolution

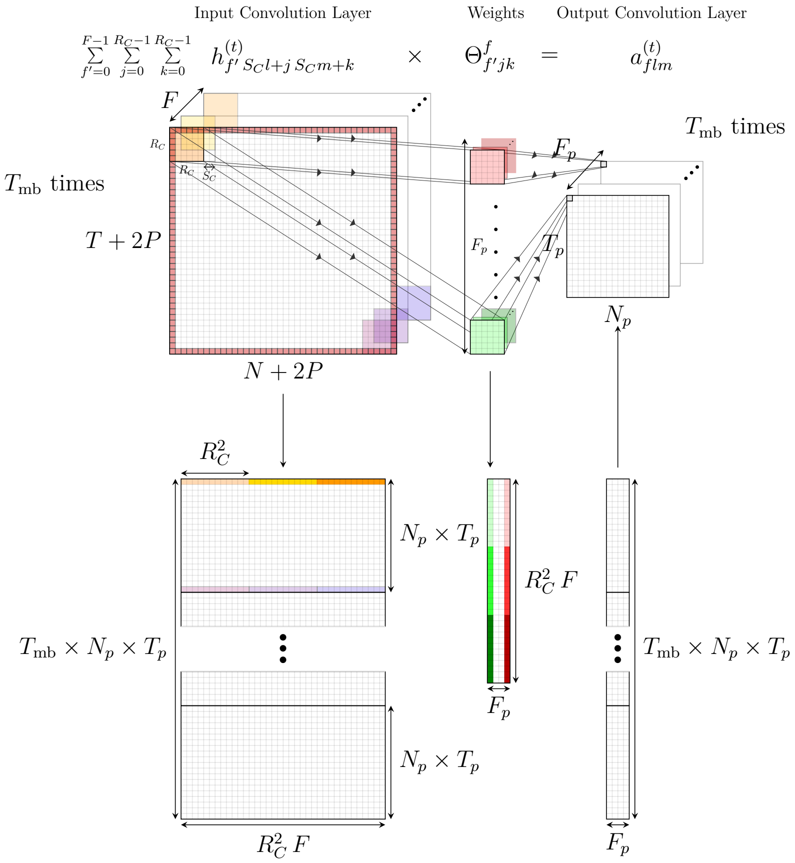


Figure 5.33: 4D convolution as a 2D matrix multiplication

Following the same lines, the adding of the input and output feature maps as well as the batch size poses no particular conceptual difficulty, as illustrated on figure 5.33, corresponding to the 4D convolution

$$a^{(t)}_{flm} = \sum_{f'=0}^{F_p-1} \sum_{j=0}^{R_C-1} \sum_{k=0}^{R_C-1} \Theta^{f}_{f'jk} h^{(t)}_{f' S_C l+j\, S_C m+k} \,. \tag{5.84}$$

5.H Pooling as a row matrix maximum

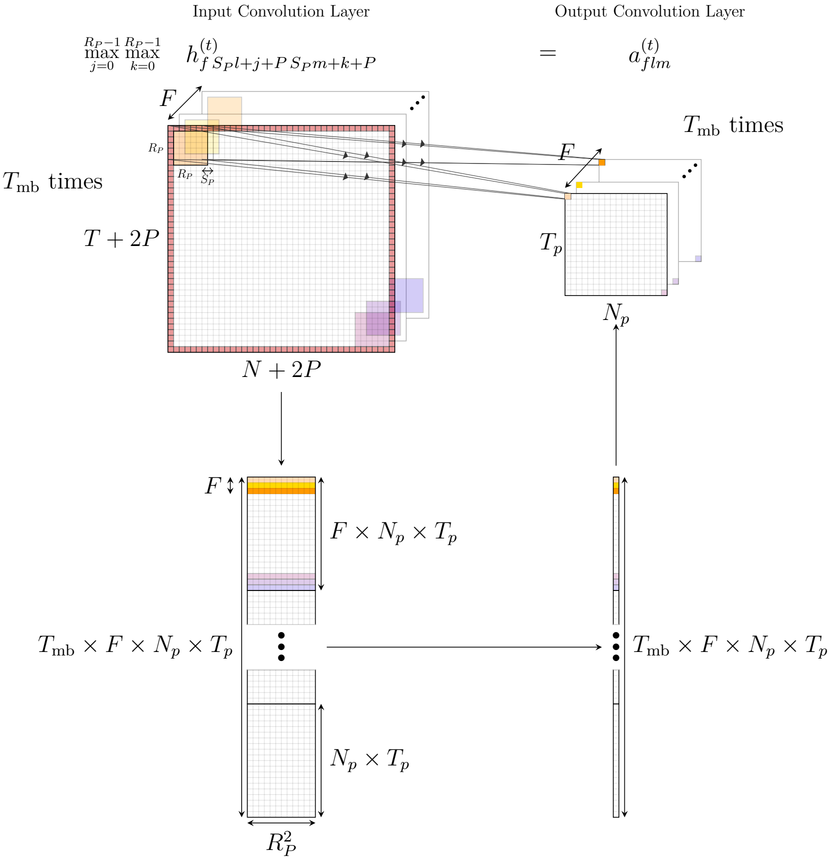

Input Convolution Layer
Output Convolution Layer
$\max_{j=0}^{R_P-1} \max_{k=0}^{R_P-1} h^{(t)}_{f\, S_P l+j+P\, S_P m+k+P}$
$=$
$a^{(t)}_{flm}$
F
$T_{\rm mb}$ times
R_P
R_P
S_P
$T_{\rm mb}$ times
$T+2P$
F
T_p
N_p
$N+2P$
F
$F \times N_p \times T_p$
$T_{\rm mb} \times F \times N_p \times T_p$
$T_{\rm mb} \times F \times N_p \times T_p$
$N_p \times T_p$
R_P^2

Figure 5.34: 4D pooling as a 2D matrix multiplication

The pooling operation can also be simplified, seeing it a the maximum search on the rows of a flattened 2D matrix. This is illustrated on figure 5.34

$$a^{(t)}_{flm} = \max_{j=0}^{R_P-1} \max_{k=0}^{R_P-1} h^{(t)}_{f S_P l+j\, S_P m+k} \,. \tag{5.85}$$

Chapter 6

Recurrent Neural Networks

Contents

6.1 Introduction

In this chapter, we review a third kind of Neural Network architecture: Recurrent Neural Networks[6]. By contrast with the CNN, this kind of network introduces a real architecture novelty : instead of forwarding only in a "spatial" direction, the data are also forwarded in a new – time dependent – direction. We will present the first Recurrent Neural Network (RNN) architecture, as well as the current most popular one: the Long Short Term Memory (LSTM) Neural Network.

6.2 RNN-LSTM architecture

6.2.1 Forward pass in a RNN-LSTM

In figure 4.1, we present the RNN architecture in a schematic way

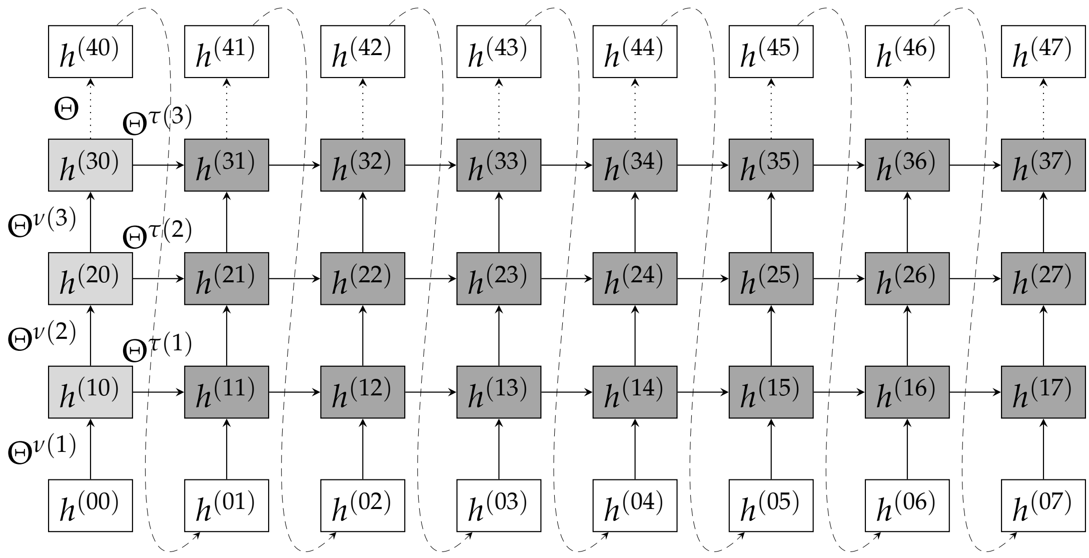


Figure 6.1: RNN architecture, with data propagating both in "space" and in "time". In our exemple, the time dimension is of size 8 while the "spatial" one is of size 4.

The real novelty of this type of neural network is that the fact that we are trying to predict a time serie is encoded in the very architecture of the network. RNN have first been introduced mostly to predict the next words in a sentance (classification task), hence the notion of ordering in time of the prediction. But this kind of network architecture can also be applied to regression problems. Among others things one can think of stock prices evolution, or temperature forecasting. In contrast to the precedent neural networks that we introduced, where we defined (denoting ν as in previous chapters the layer index in the spatial direction)

$$\begin{aligned} a_f^{(t)(\nu)} &= \text{Weight Averaging } \left(h_f^{(t)(\nu)}\right) \;, \\ h_f^{(t)(\nu+1)} &= \text{Activation function } \left(a_f^{(t)(\nu)}\right) \;, \end{aligned} \tag{6.1}$$

we now have the hidden layers that are indexed by both a "spatial" and a "temporal" index (with T being the network dimension in this new direction), and the general philosophy of the RNN is (now the a is usually characterized by a c for cell state, this denotation, trivial for the basic RNN architecture will make more sense when we talk about LSTM networks)

$$\begin{aligned} c_f^{(t)(\nu\tau)} &= \text{Weight Averaging } \left(h_f^{(t)(\nu\tau-1)}, h_f^{(t)(\nu-1\tau)}\right) \;, \\ h_f^{(t)(\nu\tau)} &= \text{Activation function } \left(c_f^{(t)(\nu\tau)}\right) \;, \end{aligned} \tag{6.2}$$

6.2.2 Backward pass in a RNN-LSTM

The backward pass in a RNN-LSTM has to respect a certain time order, as illustrated in the following figure

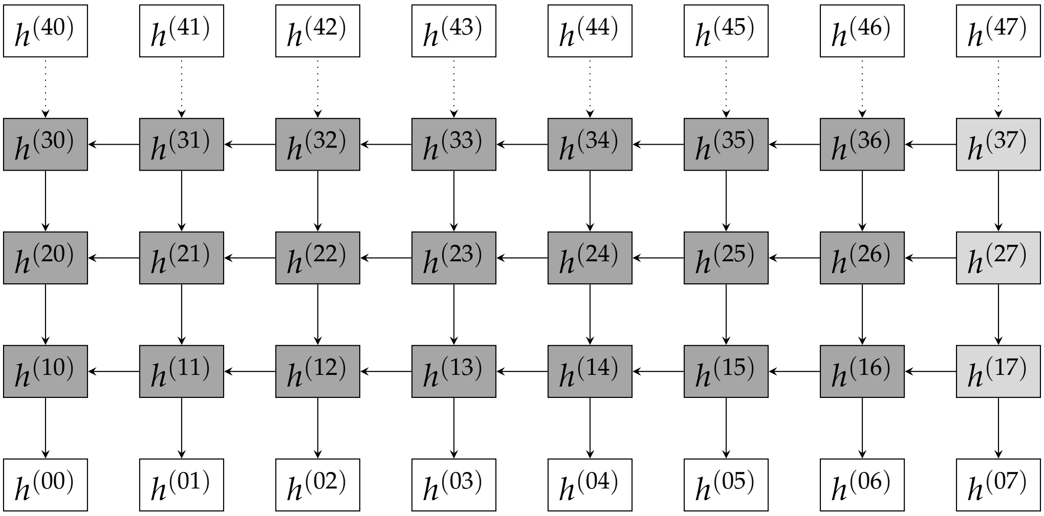


Figure 6.2: Architecture taken, backward pass. Here what cannot compute the gradient of a layer without having computed the ones that flow into it

With this in mind, let us now see in details the implementation of a RNN and its advanced cousin, the Long Short Term Memory (LSTM)-RNN.

6.3 Extreme Layers and loss function

These part of the RNN-LSTM networks just experiences trivial modifications. Let us see them

6.3.1 Input layer

In a RNN-LSTM, the input layer is recursively defined as

$$h_f^{(t)(0\tau+1)} = \left(\tilde{h}_f^{(t)(0\tau)}, h_f^{(t)(N-1\tau)}\right) . \tag{6.3}$$

where $\tilde{h}_f^{(t)(0\tau)}$ is $h_f^{(t)(0\tau)}$ with the first time column removed.

6.3.2 Output layer

The output layer of a RNN-LSTM reads

$$h_f^{(t)(N\tau)} = o\left(\sum_{f'=0}^{F_{N-1}-1} \Theta^f_{f'} h_f^{(t)(N-1\tau)}\right) , \tag{6.4}$$

where the output function o is as for FNN's and CNN's is either the identity (regression task) or the cross-entropy function (classification task).

6.3.3 Loss function

The loss function for a regression task reads

$$J(\Theta) = \frac{1}{2T_{\text{mb}}} \sum_{t=0}^{T_{\text{mb}}-1} \sum_{\tau=0}^{T-1} \sum_{f=0}^{F_N-1} \left(h_f^{(t)(N\tau)} - y_f^{(t)(\tau)}\right)^2 . \tag{6.5}$$

and for a classification task

$$J(\Theta) = -\frac{1}{T_{\text{mb}}} \sum_{t=0}^{T_{\text{mb}}-1} \sum_{\tau=0}^{T-1} \sum_{c=0}^{C-1} \delta^c_{y_c^{(t)(\tau)}} \ln\left(h_f^{(t)(N\tau)}\right) . \tag{6.6}$$

6.4 RNN specificities

6.4.1 RNN structure

RNN is the most basic architecture that takes – thanks to the way it is built in – into account the time structure of the data to be predicted. Zooming on one hidden layer of 6.1, here is what we see for a simple Recurrent Neural Network.

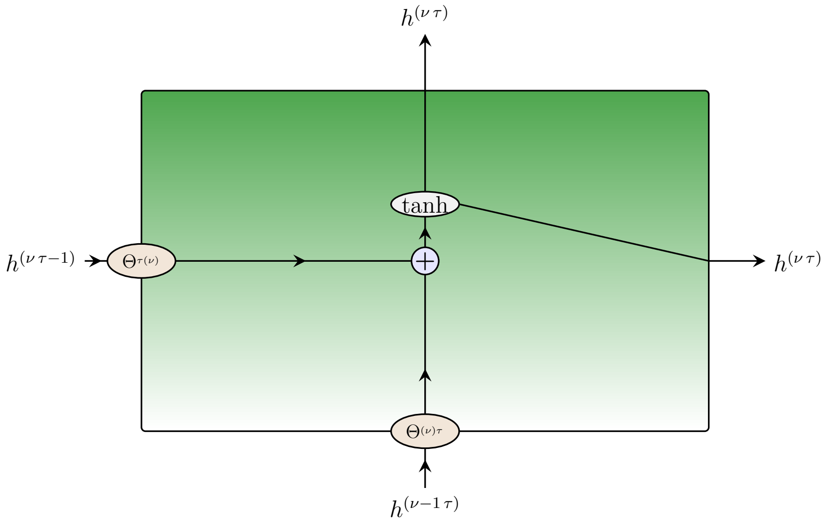


Figure 6.3: RNN hidden unit details

And here is how the output of the hidden layer represented in 6.3 enters into the subsequent hidden units

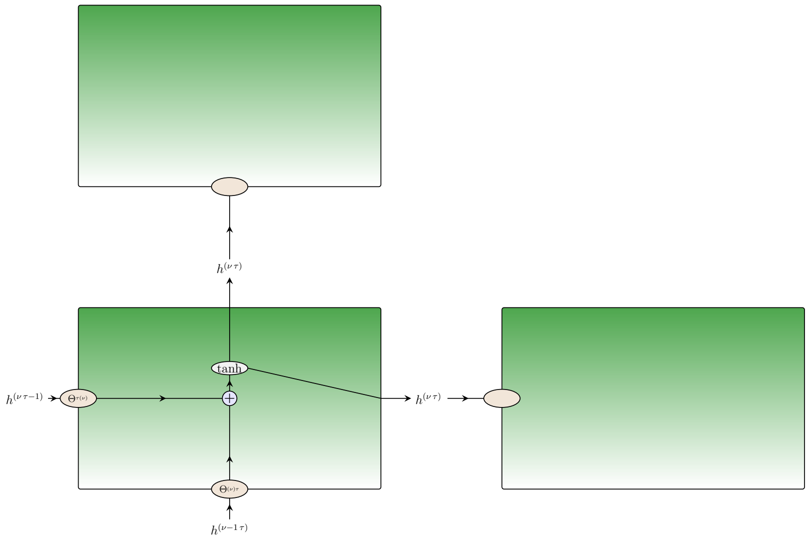


Figure 6.4: How the RNN hidden unit interact with each others

Lest us now mathematically express what is reprensented in figures 6.3 and 6.4.

6.4.2 Forward pass in a RNN

In a RNN, the update rules read for the first time slice (spatial layer at the extreme left of figure 6.1)

$$h_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)}\right) , \tag{6.7}$$

and for the other ones

$$h_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_{\nu}-1} \Theta_{f'}^{\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) . \tag{6.8}$$

6.4.3 Backpropagation in a RNN

The backpropagation philosophy will remain unchanged : find the error rate updates, from which one can deduce the weight updates. But as for the hidden layers, the δ now have both a spatial and a temporal component. We will thus have to compute

$$\delta_f^{(t)(\nu\tau)} = \frac{\delta}{\delta h_f^{(t)(\nu+1\tau)}} J(\Theta) , \tag{6.9}$$

to deduce

$$\Delta_{f'}^{\Theta\text{index}f} = \frac{\delta}{\delta \Delta_{f'}^{\Theta\text{index}f}} J(\Theta) , \tag{6.10}$$

where the index can either be nothing (weights of the ouput layers), $\nu(\nu)$ (weights between two spatially connected layers) or $\tau(\nu)$ (weights between two temporally connected layers). First, it is easy to compute (in the same way as in chapter 1 for FNN) for the MSE loss function

$$\delta_f^{(t)(N-1\tau)} = \frac{1}{T_{\text{mb}}}\left(h_f^{(t)(N\tau)} - y_f^{(t)(\tau)}\right) , \tag{6.11}$$

and for the cross entropy loss function

$$\delta_f^{(t)(N-1)} = \frac{1}{T_{\text{mb}}}\left(h_f^{(t)(N\tau)} - \delta^f_{y^{(t)(\tau)}}\right) . \tag{6.12}$$

Calling

$$\mathcal{T}_f^{(t)(\nu\tau)} = 1 - \left(h_f^{(t)(\nu\tau)}\right)^2 , \tag{6.13}$$

and

$$\mathcal{H}_{ff'}^{(t')(\nu\tau)_a} = \mathcal{T}_{f'}^{(t')(\nu+1\tau)} \Theta_f^{a(\nu+1)f'} , \tag{6.14}$$

we show in appendix 6.B.1 that (if $\tau+1$ exists, otherwise the second term is absent)

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\text{mb}}} J_f^{(tt')(\nu\tau)} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}_{ff'}^{(t')(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}} \delta_{f'}^{(t')(\nu-\epsilon\tau+\epsilon)} . \tag{6.15}$$

where $b_0 = \nu$ and $b_1 = \tau$.

6.4.4 Weight and coefficient updates in a RNN

To complete the backpropagation algorithm, we need

$$\Delta_{f'}^{\nu(\nu)f} , \qquad \Delta_{f'}^{\tau(\nu)f} , \qquad \Delta_{f'}^{f} , \qquad \Delta_f^{\beta(\nu\tau)} , \qquad \Delta_f^{\gamma(\nu\tau)} . \tag{6.16}$$

We show in appendix 6.B.2 that

$$\Delta_{f'}^{\nu(\nu-)f} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\text{mb}}-1} \mathcal{T}_f^{(t)(\nu\tau)} \delta_f^{(t)(\nu-1\tau)} h_{f'}^{(t)(\nu-1\tau)} , \tag{6.17}$$

$$\Delta_{f'}^{\tau(\nu)f} = \sum_{\tau=1}^{T-1} \sum_{t=0}^{T_{\text{mb}}-1} \mathcal{T}_f^{(t)(\nu\tau)} \delta_f^{(t)(\nu-1\tau)} h_{f'}^{(t)(\nu\tau-1)} , \tag{6.18}$$

$$\Delta_{f'}^{f} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\text{mb}}-1} h_{f'}^{(t)(N-1\tau)} \delta_f^{(t)(N-1\tau)} , \tag{6.19}$$

$$\Delta_f^{\beta(\nu\tau)} = \sum_{t=0}^{T_{\text{mb}}-1} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}_{ff'}^{(t')(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}} \delta_{f'}^{(t')(\nu-\epsilon\tau+\epsilon)} , \tag{6.20}$$

$$\Delta_f^{\gamma(\nu\tau)} = \sum_{t=0}^{T_{\text{mb}}-1} \tilde{h}_f^{(t)(\nu\tau)} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}_{ff'}^{(t')(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}} \delta_{f'}^{(t')(\nu-\epsilon\tau+\epsilon)} . \tag{6.21}$$

6.5 LSTM specificities

6.5.1 LSTM structure

In a Long Short Term Memory Neural Network[7], the state of a given unit is not directly determined by its left and bottom neighbours. Instead, a cell state is updated for each hidden unit, and the output of this unit is a probe of the cell state. This formulation might seem puzzling at first, but it is philosophically similar to the ResNet approach that we briefly encounter in the appendix of chapter 4: instead of trying to fit an input with a complicated function, we try to fit tiny variation of the input, hence allowing the gradient to flow in a smoother manner in the network. In the LSTM network, several gates are thus introduced : the input gate $i_f^{(t)(\nu\tau)}$ determines if we allow new information $g_f^{(t)(\nu\tau)}$ to enter into the cell state. The output gate $o_f^{(t)(\nu\tau)}$ determines if we set or not the output hidden value to 0, or really probes the current cell state. Finally, the forget state $f_f^{(t)(\nu\tau)}$ determines if we forget or not the past cell state. All theses concepts are illustrated on the figure 6.5, which is the LSTM counterpart of the RNN structure of section 6.4.1. This diagram will be explained in details in the next section.

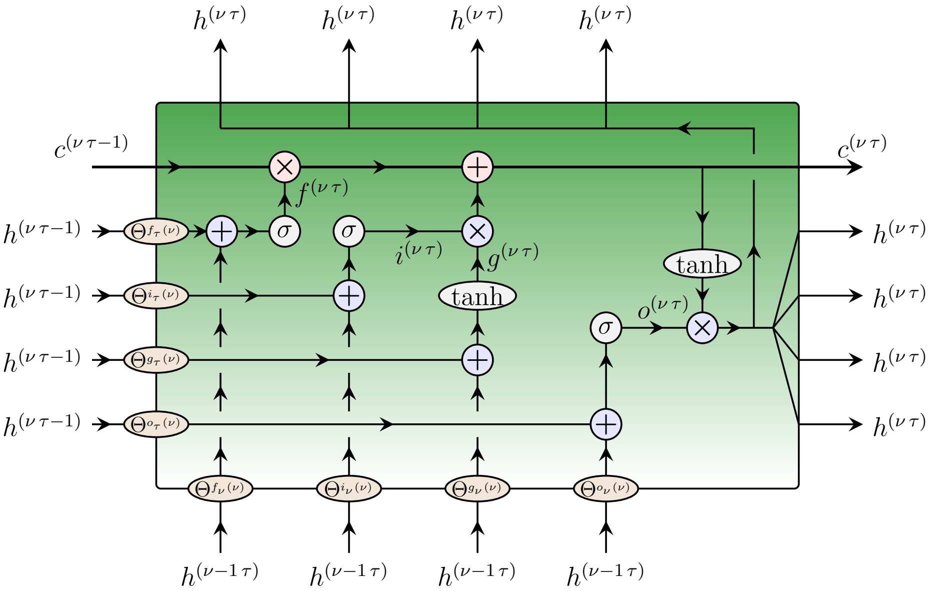


Figure 6.5: LSTM hidden unit details

In a LSTM, the different hidden units interact in the following way

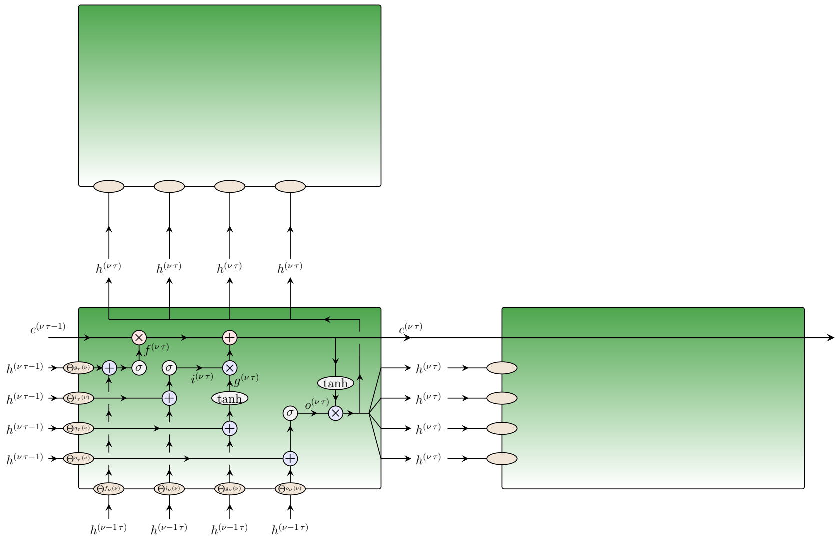


Figure 6.6: How the LSTM hidden unit interact with each others

6.5.2 Forward pass in LSTM

Considering all the $\tau - 1$ variable values to be 0 when $\tau = 0$, we get the following formula for the input, forget and output gates

$$i_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{i_\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{i_\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.22}$$

$$f_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{f_\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{f_\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.23}$$

$$o_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{o_\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{o_\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) . \tag{6.24}$$

The sigmoid function is the reason why the i, f, o functions are called gates: they take their values between 0 and 1, therefore either allowing or forbidding information to pass through the next step. The cell state update is then performed in the following way

$$g_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{g_\nu(\nu)f} h_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{g_\tau(\nu)f} h_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.25}$$

$$c_f^{(t)(\nu\tau)} = f_f^{(t)(\nu\tau)} c_f^{(t)(\nu\tau-1)} + i_f^{(t)(\nu\tau)} g_f^{(t)(\nu\tau)} , \tag{6.26}$$

and as announced, hidden state update is just a probe of the current cell state

$$h_f^{(t)(\nu\tau)} = o_f^{(t)(\nu\tau)} \tanh\left(c_f^{(t)(\nu\tau)}\right) . \tag{6.27}$$

These formula singularly complicates the feed forward and especially the backpropagation procedure. For completeness, we will us nevertheless carefully derive it. Let us mention in passing that recent studies tried to replace the tanh activation function of the hidden state $h_f^{(t)(\nu\tau)}$ and the cell update $g_f^{(t)(\nu\tau)}$ by Rectified Linear Units, and seems to report better results with a proper initialization of all the weight matrices, argued to be diagonal

$$\Theta_{f'}^{f}(\text{init}) = \frac{1}{2}\delta_{f'}^{f}\left(+\sqrt{\frac{6}{F_{\text{in}} + F_{\text{out}}}}\right) , \tag{6.28}$$

with the bracket term here to possibly (or not) include some randomness into the initialization

6.5.3 Batch normalization

In batchnorm The update rules for the gates are modified as expected

$$i_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{i_\nu(\nu-)f} y_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{i_\tau(-\nu)f} y_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.29}$$

$$f_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{f_\nu(\nu-)f} y_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{f_\tau(-\nu)f} y_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.30}$$

$$o_f^{(t)(\nu\tau)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{o_\nu(\nu-)f} y_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{o_\tau(-\nu)f} y_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.31}$$

$$g_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{g_\nu(\nu-)f} y_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{g_\tau(-\nu)f} y_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.32}$$

where

$$y_f^{(t)(\nu\tau)} = \gamma_f^{(\nu\tau)} \tilde{h}_f^{(t)(\nu\tau)} + \beta_f^{(\nu\tau)} \,, \tag{6.33}$$

as well as

$$\tilde{h}_f^{(t)(\nu\tau)} = \frac{h_f^{(t)(\nu\tau)} - \hat{h}_f^{(\nu\tau)}}{\sqrt{\left(\sigma_f^{(\nu\tau)}\right)^2 + \epsilon}} \tag{6.34}$$

and

$$\hat{h}_f^{(\nu\tau)} = \frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} h_f^{(t)(\nu\tau)} \,, \quad \left(\sigma_f^{(\nu\tau)}\right)^2 = \frac{1}{T_{\rm mb}} \sum_{t=0}^{T_{\rm mb}-1} \left(h_f^{(t)(\nu\tau)} - \hat{h}_f^{(\nu\tau)}\right)^2 \,. \tag{6.35}$$

It is important to compute a running sum for the mean and the variance, that will serve for the evaluation of the cross-validation and the test set (calling e the number of iterations/epochs)

$$\mathbb{E}\left[h_f^{(t)(\nu\tau)}\right]_{e+1} = \frac{e\mathbb{E}\left[h_f^{(t)(\nu\tau)}\right]_e + \hat{h}_f^{(\nu\tau)}}{e+1} \,, \tag{6.36}$$

$$\mathbb{V}ar\left[h_f^{(t)(\nu\tau)}\right]_{e+1} = \frac{e\mathbb{V}ar\left[h_f^{(t)(\nu\tau)}\right]_e + \left(\hat{\sigma}_f^{(\nu\tau)}\right)^2}{e+1} \tag{6.37}$$

and what will be used at the end is $\mathbb{E}\left[h_f^{(t)(\nu\tau)}\right]$ and $\frac{T_{\rm mb}}{T_{\rm mb}-1}\mathbb{V}ar\left[h_f^{(t)(\nu\tau)}\right]$.

6.5.4 Backpropagation in a LSTM

The backpropagation In a LSTM keeps the same structure as in a RNN, namely

$$\delta_f^{(t)(N-1\tau)} = \frac{1}{T_{\rm mb}} \left(h_f^{(t)(N\tau)} - y_f^{(t)(\tau)}\right) \,, \tag{6.38}$$

and (shown in appendix 6.C.1)

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\rm mb}} J_f^{(tt')(\nu\tau)} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}_{ff'}^{(t')(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}} \delta_{f'}^{(t')(\nu-\epsilon\tau+\epsilon)} \,. \tag{6.39}$$

What changes is the form of $\mathcal{H}$, now given by

$$\begin{aligned}
\mathcal{O}^{(t)(\nu\tau)}_f &= h^{(t)(\nu\tau)}_f \left(1 - o^{(t)(\nu\tau)}_f\right) \;, \\
\mathcal{I}^{(t)(\nu\tau)}_f &= o^{(t)(\nu\tau)}_f \left(1 - \tanh^2\left(c^{(t)(\nu\tau)}_f\right)\right) g^{(t)(\nu\tau)}_f i^{(t)(\nu\tau)}_f \left(1 - i^{(t)(\nu\tau)}_f\right) \;, \\
\mathcal{F}^{(t)(\nu\tau)}_f &= o^{(t)(\nu\tau)}_f \left(1 - \tanh^2\left(c^{(t)(\nu\tau)}_f\right)\right) c^{(t)(\nu\tau-1)}_f f^{(t)(\nu\tau)}_f \left(1 - f^{(t)(\nu\tau)}_f\right) \;, \\
\mathcal{G}^{(t)(\nu\tau)}_f &= o^{(t)(\nu\tau)}_f \left(1 - \tanh^2\left(c^{(t)(\nu\tau)}_f\right)\right) i^{(t)(\nu\tau)}_f \left(1 - \left(g^{(t)(\nu\tau)}_f\right)^2\right) \;,
\end{aligned} \tag{6.40}$$

and

$$\begin{aligned}
H^{(t)(\nu\tau)_a}_{ff'} &= \Theta^{o_a(\nu+1)f'}_f \mathcal{O}^{(t)(\nu+1\tau)}_{f'} + \Theta^{f_a(\nu+1)f'}_f \mathcal{F}^{(t)(\nu+1\tau)}_{f'} \\
&\quad + \Theta^{g_a(\nu+1)f'}_f \mathcal{G}^{(t)(\nu+1\tau)}_{f'} + \Theta^{i_a(\nu+1)f'}_f \mathcal{I}^{(t)(\nu+1\tau)}_{f'} \;.
\end{aligned} \tag{6.41}$$

6.5.5 Weight and coefficient updates in a LSTM

As for the RNN, (but with the $\mathcal{H}$ defined in section 6.5.4), we get for $\nu = 1$

$$\Delta^{\rho_\nu(\nu)f}_{f'} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\text{mb}}-1} \rho^{(\nu\tau)(t)}_f \delta^{(\nu\tau)(t)}_f h^{(\nu-1\tau)(t)}_{f'} \;, \tag{6.42}$$

$$\tag{6.43}$$

and otherwise

$$\Delta^{\rho_\nu(\nu)f}_{f'} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\text{mb}}-1} \rho^{(\nu\tau)(t)}_f \delta^{(\nu\tau)(t)}_f y^{(\nu-1\tau)(t)}_{f'} \;, \tag{6.44}$$

$$\rho^{(\nu\tau)(t)}_f \delta^{(\nu\tau)(t)}_f y^{(\nu-1\tau)(t)}_{f'} \;, \tag{6.45}$$

$$\Delta^{\rho_\tau(\nu)f}_{f'} = \sum_{\tau=1}^{T-1} \sum_{t=0}^{T_{\text{mb}}-1} \rho^{(\nu\tau)(t)}_f \delta^{(\nu\tau)(t)}_f y^{(\nu\tau-1)(t)}_{f'} \;, \tag{6.46}$$

$$\Delta^{\beta(\nu\tau)}_f = \sum_{t=0}^{T_{\text{mb}}-1} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}^{(t)(\nu-\epsilon\tau+\epsilon)_{b\epsilon}}_{ff'} \delta^{(t)(\nu-\epsilon\tau+\epsilon)}_{f'} \;, \tag{6.47}$$

$$\Delta^{\gamma(\nu\tau)}_f = \sum_{t=0}^{T_{\text{mb}}-1} \tilde{h}^{(t)(\nu\tau)}_f \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}^{(t)(\nu-\epsilon\tau+\epsilon)_{b\epsilon}}_{ff'} \delta^{(t)(\nu-\epsilon\tau+\epsilon)}_{f'} \;. \tag{6.48}$$

and

$$\Delta^{f}_{f'} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\rm mb}-1} y^{(t)(N-1\tau)}_{f'} \delta^{(t)(N-1\tau)}_{f} \; . \tag{6.49}$$

Appendix

6.A Backpropagation trough Batch Normalization

For Backpropagation, we will need

$$\frac{\partial y^{(t')(\nu\tau)}_{f'}}{\partial h^{(t)(\nu\tau)}_{f}} = \gamma^{(\nu\tau)}_{f} \frac{\partial \tilde{h}^{(t)(\nu\tau)}_{f'}}{\partial h^{(t)(\nu\tau)}_{f}} \; . \tag{6.50}$$

Since

$$\frac{\partial h^{(t')(\nu\tau)}_{f'}}{\partial h^{(t)(\nu\tau)}_{f}} = \delta^{t'}_{t} \delta^{f'}_{f} \; , \qquad\qquad \frac{\partial \hat{h}^{(\nu\tau)}_{f'}}{\partial h^{(t)(\nu\tau)}_{f}} = \frac{\delta^{f'}_{f}}{T_{\rm mb}} \; ; \tag{6.51}$$

and

$$\frac{\partial \left(\hat{\sigma}^{(\nu\tau)}_{f'}\right)^2}{\partial h^{(t)(\nu\tau)}_{f}} = \frac{2\delta^{f'}_{f}}{T_{\rm mb}} \left(h^{(t)(\nu\tau)}_{f} - \hat{h}^{(\nu\tau)}_{f}\right) \; , \tag{6.52}$$

we get

$$\begin{aligned} \frac{\partial \tilde{h}^{(t')(\nu\tau)}_{f'}}{\partial h^{(t)(\nu\tau)}_{f}} &= \frac{\delta^{f'}_{f}}{T_{\rm mb}} \left[\frac{T_{\rm mb}\delta^{t'}_{t} - 1}{\left(\left(\hat{\sigma}^{(\nu\tau)}_{f}\right)^2 + \epsilon\right)^{\frac{1}{2}}} - \frac{\left(h^{(t')(\nu\tau)}_{f} - \hat{h}^{(\nu\tau)}_{f}\right)\left(h^{(t)(\nu\tau)}_{f} - \hat{h}^{(\nu\tau)}_{f}\right)}{\left(\left(\hat{\sigma}^{(\nu\tau)}_{f}\right)^2 + \epsilon\right)^{\frac{3}{2}}} \right] \\ &= \frac{\delta^{f'}_{f}}{\left(\left(\hat{\sigma}^{(\nu\tau)}_{f}\right)^2 + \epsilon\right)^{\frac{1}{2}}} \left[\delta^{t'}_{t} - \frac{1 + \tilde{h}^{(t')(\nu\tau)}_{f} \tilde{h}^{(t)(\nu\tau)}_{f}}{T_{\rm mb}} \right] \; . \end{aligned} \tag{6.53}$$

To ease the notation we will denote

$$\tilde{\gamma}_f^{(\nu\tau)} = \frac{\gamma_f^{(\nu\tau)}}{\left(\left(\hat{\sigma}_f^{(\nu\tau)}\right)^2 + \epsilon\right)^{\frac{1}{2}}} . \tag{6.54}$$

so that

$$\frac{\partial y_{f'}^{(t')(\nu\tau)}}{\partial h_f^{(t)(\nu\tau)}} = \tilde{\gamma}_f^{(\nu\tau)} \delta_f^{f'} \left[\delta_t^{t'} - \frac{1 + \tilde{h}_f^{(t')(\nu\tau)} \tilde{h}_f^{(t)(\nu\tau)}}{T_{\text{mb}}} \right] . \tag{6.55}$$

This modifies the error rate backpropagation, as well as the formula for the weight update (y's instead of h's). In the following we will use the formula

$$J_f^{(tt')(\nu\tau)} = \tilde{\gamma}_f^{(\nu\tau)} \left[\delta_t^{t'} - \frac{1 + \tilde{h}_f^{(t')(\nu\tau)} \tilde{h}_f^{(t)(\nu\tau)}}{T_{\text{mb}}} \right] . \tag{6.56}$$

6.B RNN Backpropagation

6.B.1 RNN Error rate updates: details

Recalling the error rate definition

$$\delta_f^{(t)(\nu\tau)} = \frac{\delta}{\delta h_f^{(t)(\nu+1\tau)}} J(\Theta) , \tag{6.57}$$

we would like to compute it for all existing values of ν and τ. As computed in chapter 4, one has for the maximum ν value

$$\delta_f^{(t)(N-1\tau)} = \frac{1}{T_{\text{mb}}} \left(h_f^{(t)(N\tau)} - y_f^{(t)(\tau)} \right) . \tag{6.58}$$

Now since (taking Batch Normalization into account)

$$h_f^{(t)(N\tau)} = o\left(\sum_{f'=0}^{F_{N-1}-1} \Theta_{f'}^{f} y_f^{(t)(N-1\tau)} \right) , \tag{6.59}$$

and

$$h_f^{(t)(\nu\tau)} = \tanh\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta_{f'}^{\nu(\nu)f} y_{f'}^{(t)(\nu-1\tau)} + \sum_{f'=0}^{F_\nu-1} \Theta_{f'}^{\tau(\nu)f} y_{f'}^{(t)(\nu\tau-1)}\right) , \tag{6.60}$$

we get for

$$\begin{aligned}\delta_f^{(t)(N-2\tau)} &= \sum_{t'=0}^{T_{\rm mb}} \left[\sum_{f'=0}^{F_N-1} \frac{\delta h_{f'}^{(t')(N\tau)}}{\delta h_f^{(t)(N-1\tau)}} \delta_{f'}^{(t')(N-1\tau)}\right.\\ &\left. + \sum_{f'=0}^{F_{N-1}-1} \frac{\delta h_{f'}^{(t')(N-1\tau+1)}}{\delta h_f^{(t)(N-1\tau)}} \delta_{f'}^{(t')(N-2\tau+1)}\right] .\end{aligned} \tag{6.61}$$

Let us work out explicitly once (for a regression cost function and a trivial identity output function)

$$\begin{aligned}\frac{\delta h_{f'}^{(t')(N\tau)}}{\delta h_f^{(t)(N-1\tau)}} &= \sum_{f''=0}^{F_{N-1}-1} \Theta_{f''}^{f'} \frac{\delta y_{f''}^{(t')(N-1\tau)}}{\delta h_f^{(t)(N-1\tau)}}\\ &= \Theta_f^{f'} J_f^{(tt')(N-1\tau)} .\end{aligned} \tag{6.62}$$

as well as

$$\begin{aligned}\frac{\delta h_{f'}^{(t')(N-1\tau+1)}}{\delta h_f^{(t)(N-1\tau)}} &= \left[1 - \left(h_{f'}^{(t')(N-1\tau+1)}\right)^2\right] \sum_{f''=0}^{F_{N-1}-1} \Theta_{f''}^{\tau(N-1)f'} \frac{\delta y_{f''}^{(t')(N-1\tau)}}{\delta h_f^{(t)(N-1\tau)}}\\ &= \mathcal{T}_{f'}^{(t')(N-1\tau+1)} \Theta_f^{\tau(N-1)f'} J_f^{(tt')(N-1\tau)} .\end{aligned} \tag{6.63}$$

Thus

$$\begin{aligned}\delta_f^{(t)(N-2\tau)} &= \sum_{t'=0}^{T_{\rm mb}} J_f^{(tt')(N-1\tau)} \left[\sum_{f'=0}^{F_N-1} \Theta_f^{f'} \delta_{f'}^{(t')(N-1\tau)}\right.\\ &\left. + \sum_{f'=0}^{F_{N-1}-1} \mathcal{T}_{f'}^{(t')(N-1\tau+1)} \Theta_f^{\tau(N-1)f'} \delta_{f'}^{(t')(N-2\tau+1)}\right] .\end{aligned} \tag{6.64}$$

Here we adopted the convention that the $\delta^{(t')(N-2\tau+1)}$'s are 0 if $\tau = T$. In a similar way, we derive for $\nu \leq N-1$

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\rm mb}} J_f^{(tt')(\nu\tau)} \left[\sum_{f'=0}^{F_{\nu+1}-1} \mathcal{T}_{f'}^{(t')(\nu+1\tau)} \Theta_f^{\nu(\nu+1)f'} \, \delta_{f'}^{(t')(\nu\tau)} \right.$$
$$\left. + \sum_{f'=0}^{F_\nu-1} \mathcal{T}_{f'}^{(t')(\nu\tau+1)} \Theta_f^{\tau(\nu)f'} \, \delta_{f'}^{(t')(\nu-1\tau+1)} \right] . \tag{6.65}$$

Defining

$$\mathcal{T}_{f'}^{(t')(N\tau)} = 1 \, , \qquad \qquad \Theta_f^{\nu(N)f'} = \Theta_f^{f'} \, , \tag{6.66}$$

the previous $\delta_f^{(t)(\nu-1\tau)}$ formula extends to the case $\nu = N-1$. To unite the RNN and the LSTM formulas, let us finally define (with a either τ or ν

$$\mathcal{H}_{ff'}^{(t')(\nu\tau)_a} = \mathcal{T}_{f'}^{(t')(\nu+1\tau)} \Theta_f^{a(\nu+1)f'} \, , \tag{6.67}$$

thus (defining $b_0 = \nu$ and $b_1 = \tau$)

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\rm mb}} J_f^{(tt')(\nu\tau)} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}_{ff'}^{(t')(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}} \delta_{f'}^{(t')(\nu-\epsilon\tau+\epsilon)} \, . \tag{6.68}$$

6.B.2 RNN Weight and coefficient updates: details

We want here to derive

$$\Delta_{f'}^{\nu(\nu)f} = \frac{\partial}{\partial \Theta_{f'}^{\nu(\nu)f}} J(\Theta) \qquad \qquad \Delta_{f'}^{\tau(\nu)f} = \frac{\partial}{\partial \Theta_{f'}^{\tau(\nu)f}} J(\Theta) \, . \tag{6.69}$$

We first expand

$$\Delta_{f'}^{\nu(\nu)f} = \sum_{\tau=0}^{T-1} \sum_{f''=0}^{F_\nu-1} \sum_{t=0}^{T_{\rm mb}-1} \frac{\partial h_{f''}^{(t)(\nu\tau)}}{\partial \Theta_{f'}^{\nu(\nu)f}} \delta_{f''}^{(t)(\nu-1\tau)} \, ,$$
$$\Delta_{f'}^{\tau(\nu)f} = \sum_{\tau=0}^{T-1} \sum_{f''=0}^{F_\nu-1} \sum_{t=0}^{T_{\rm mb}-1} \frac{\partial h_{f''}^{(t)(\nu\tau)}}{\partial \Theta_{f'}^{\tau(\nu)f}} \delta_{f''}^{(t)(\nu-1\tau)} \, , \tag{6.70}$$

so that

$$\Delta^{\nu(\nu)f}_{f'} = \sum_{\tau=0}^{T-1}\sum_{t=0}^{T_{\rm mb}-1} \mathcal{T}^{(t)(\nu\tau)}_{f}\delta^{(t)(\nu-1\tau)}_{f}h^{(t)(\nu-1\tau)}_{f'} \ , \tag{6.71}$$

$$\Delta^{\tau(\nu)f}_{f'} = \sum_{\tau=1}^{T-1}\sum_{t=0}^{T_{\rm mb}-1} \mathcal{T}^{(t)(\nu\tau)}_{f}\delta^{(t)(\nu-1\tau)}_{f}h^{(t)(\nu\tau-1)}_{f'} \ . \tag{6.72}$$

We also have to compute

$$\Delta^{f}_{f'} = \frac{\partial}{\partial\Theta^{f}_{f'}}J(\Theta) \ . \tag{6.73}$$

We first expand

$$\Delta^{f}_{f'} = \sum_{\tau=0}^{T-1}\sum_{f''=0}^{F_N-1}\sum_{t=0}^{T_{\rm mb}-1} \frac{\partial h^{(t)(N\tau)}_{f''}}{\partial\Theta^{f}_{f'}}\delta^{(t)(N-1\tau)}_{f''} \tag{6.74}$$

so that

$$\Delta^{f}_{f'} = \sum_{\tau=0}^{T-1}\sum_{t=0}^{T_{\rm mb}-1} h^{(t)(N-1\tau)}_{f'}\delta^{(t)(N-1\tau)}_{f} \ . \tag{6.75}$$

Finally, we need

$$\Delta^{\beta(\nu\tau)}_{f} = \frac{\partial}{\partial\beta^{(\nu\tau)}_{f}}J(\Theta) \qquad\qquad \Delta^{\gamma(\nu\tau)}_{f} = \frac{\partial}{\partial\gamma^{(\nu\tau)}_{f}}J(\Theta) \ . \tag{6.76}$$

First

$$\begin{aligned}
\Delta^{\beta(\nu\tau)}_{f} &= \sum_{t=0}^{T_{\rm mb}-1}\left[\sum_{f'=0}^{F_{\nu+1}-1}\frac{\partial h^{(t)(\nu+1\tau)}_{f'}}{\partial\beta^{(\nu\tau)}_{f}}\delta^{(t)(\nu\tau)}_{f'} + \sum_{f'=0}^{F_{\nu}-1}\frac{\partial h^{(t)(\nu\tau+1)}_{f'}}{\partial\beta^{(\nu\tau)}_{f}}\delta^{(t)(\nu-1\tau+1)}_{f'}\right] \ , \\
\Delta^{\gamma(\nu\tau)}_{f} &= \sum_{t=0}^{T_{\rm mb}-1}\left[\sum_{f'=0}^{F_{\nu+1}-1}\frac{\partial h^{(t)(\nu+1\tau)}_{f'}}{\partial\gamma^{(\nu\tau)}_{f}}\delta^{(t)(\nu\tau)}_{f'} + \sum_{f'=0}^{F_{\nu}-1}\frac{\partial h^{(t)(\nu\tau+1)}_{f'}}{\partial\gamma^{(\nu\tau)}_{f}}\delta^{(t)(\nu-1\tau+1)}_{f'}\right] \ .
\end{aligned} \tag{6.77}$$

So that

$$\begin{aligned}\Delta^{\beta(\nu\tau)}_{f} &= \sum_{t=0}^{T_{\rm mb}-1}\left[\sum_{f'=0}^{F_{\nu+1}-1}\mathcal{T}^{(t)(\nu+1\tau)}_{f'}\Theta^{\nu(\nu+1)f'}_{f}\delta^{(t)(\nu\tau)}_{f'}\right.\\ &\left.+\sum_{f'=0}^{F_{\nu}-1}\mathcal{T}^{(t)(\nu\tau+1)}_{f'}\Theta^{\tau(\nu)f'}_{f}\delta^{(t)(\nu-1\tau+1)}_{f'}\right]\,, \qquad (6.78)\\ \Delta^{\gamma(\nu\tau)}_{f} &= \sum_{t=0}^{T_{\rm mb}-1}\left[\sum_{f'=0}^{F_{\nu+1}-1}\mathcal{T}^{(t)(\nu+1\tau)}_{f'}\Theta^{\nu(\nu+1)f'}_{f}\tilde{h}^{(t)(\nu\tau)}_{f}\delta^{(t)(\nu\tau)}_{f'}\right.\\ &\left.+\sum_{f'=0}^{F_{\nu}-1}\mathcal{T}^{(t)(\nu\tau+1)}_{f'}\Theta^{\tau(\nu)f'}_{f}\tilde{h}^{(t)(\nu\tau)}_{f}\delta^{(t)(\nu-1\tau+1)}_{f'}\right]\,, \qquad (6.79)\end{aligned}$$

which we can rewrite as

$$\Delta^{\beta(\nu\tau)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{\epsilon=0}^{1}\sum_{f'=0}^{F_{\nu+1-\epsilon}-1}\mathcal{H}^{(t)(\nu-\epsilon\tau+\epsilon)b_{\epsilon}}_{ff'}\delta^{(t)(\nu-\epsilon\tau+\epsilon)}_{f'}\,, \qquad (6.80)$$

$$\Delta^{\gamma(\nu\tau)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\tilde{h}^{(t)(\nu\tau)}_{f}\sum_{\epsilon=0}^{1}\sum_{f'=0}^{F_{\nu+1-\epsilon}-1}\mathcal{H}^{(t)(\nu-\epsilon\tau+\epsilon)b_{\epsilon}}_{ff'}\delta^{(t)(\nu-\epsilon\tau+\epsilon)}_{f'}\,. \qquad (6.81)$$

6.C LSTM Backpropagation

6.C.1 LSTM Error rate updates: details

As for the RNN

$$\delta^{(t)(N-1\tau)}_{f} = \frac{1}{T_{\rm mb}}\left(h^{(t)(N\tau)}_{f} - y^{(t)(\tau)}_{f}\right)\,. \qquad (6.82)$$

Before going any further, it will be useful to define

$$\begin{aligned}\mathcal{O}^{(t)(\nu\tau)}_{f} &= h^{(t)(\nu\tau)}_{f}\left(1-o^{(t)(\nu\tau)}_{f}\right)\,,\\ \mathcal{I}^{(t)(\nu\tau)}_{f} &= o^{(t)(\nu\tau)}_{f}\left(1-\tanh^{2}\left(c^{(t)(\nu\tau)}_{f}\right)\right)g^{(t)(\nu\tau)}_{f}i^{(t)(\nu\tau)}_{f}\left(1-i^{(t)(\nu\tau)}_{f}\right)\,,\\ \mathcal{F}^{(t)(\nu\tau)}_{f} &= o^{(t)(\nu\tau)}_{f}\left(1-\tanh^{2}\left(c^{(t)(\nu\tau)}_{f}\right)\right)c^{(t)(\nu\tau-1)}_{f}f^{(t)(\nu\tau)}_{f}\left(1-f^{(t)(\nu\tau)}_{f}\right)\,,\\ \mathcal{G}^{(t)(\nu\tau)}_{f} &= o^{(t)(\nu\tau)}_{f}\left(1-\tanh^{2}\left(c^{(t)(\nu\tau)}_{f}\right)\right)i^{(t)(\nu\tau)}_{f}\left(1-\left(g^{(t)(\nu\tau)}_{f}\right)^{2}\right)\,, \qquad (6.83)\end{aligned}$$

and

$$\begin{aligned} H_{ff'}^{(t)(\nu\tau)_a} &= \Theta_f^{o_a(\nu+1)f'} \mathcal{O}_{f'}^{(t)(\nu+1\tau)} + \Theta_f^{f_a(\nu+1)f'} \mathcal{F}_{f'}^{(t)(\nu+1\tau)} \\ &\quad + \Theta_f^{g_a(\nu+1)f'} \mathcal{G}_{f'}^{(t)(\nu+1\tau)} + \Theta_f^{i_a(\nu+1)f'} \mathcal{I}_{f'}^{(t)(\nu+1\tau)} \, . \end{aligned} \tag{6.84}$$

As for RNN, we will start off by looking at

$$\begin{aligned} \delta_f^{(t)(N-2\tau)} &= \sum_{t'=0}^{T_{\text{mb}}} \left[\sum_{f'=0}^{F_N-1} \frac{\delta h_{f'}^{(t')(N\tau)}}{\delta h_f^{(t)(N-1\tau)}} \delta_{f'}^{(t')(N-1\tau)} \right. \\ &\quad \left. + \sum_{f'=0}^{F_{N-1}-1} \frac{\delta h_{f'}^{(t')(N-1\tau+1)}}{\delta h_f^{(t)(N-1\tau)}} \delta_{f'}^{(t')(N-2\tau+1)} \right] \, . \end{aligned} \tag{6.85}$$

We will be able to get our hands on the second term with the general formula, so let us first look at

$$\frac{\delta h_{f'}^{(t')(N\tau)}}{\delta h_f^{(t)(N-1\tau)}} = \Theta_f^{f'} \, J_f^{(tt')(N-1\tau)} \, , \tag{6.86}$$

which is is similar to the RNN case. Let us put aside the second term of $\delta_f^{(t)(N-2\tau)}$, and look at the general case

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\text{mb}}} \left[\sum_{f'=0}^{F_{\nu+1}-1} \frac{\delta h_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} \delta_{f'}^{(t')(\nu\tau)} + \sum_{f'=0}^{F_\nu-1} \frac{\delta h_{f'}^{(t')(\nu\tau+1)}}{\delta h_f^{(t)(\nu\tau)}} \delta_{f'}^{(t')(\nu-1\tau+1)} \right] \, , \tag{6.87}$$

which involves to study in details

$$\begin{aligned} \frac{\delta h_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} &= \frac{\delta o_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} \tanh c_{f'}^{(t')(\nu+1\tau)} \\ &\quad + \frac{\delta c_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} o_{f'}^{(t')(\nu+1\tau)} \left[1 - \tanh^2 c_{f'}^{(t')(\nu+1\tau)} \right] \, . \end{aligned} \tag{6.88}$$

Now

$$\begin{aligned}\frac{\delta o_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} &= o_{f'}^{(t')(\nu+1\tau)}\left[1-o_{f'}^{(t')(\nu+1\tau)}\right]\sum_{f''=0}^{F_\nu-1}\Theta_{f''}^{o_\nu(\nu+1)f'}\frac{\delta y_{f'}^{(t')(\nu\tau)}}{\delta h_f^{(t)(\nu\tau)}}\\ &= o_{f'}^{(t')(\nu+1\tau)}\left[1-o_{f'}^{(t')(\nu+1\tau)}\right]\Theta_f^{o_\nu(\nu+1)f'}J_f^{(tt')(\nu\tau)}\,, \qquad (6.89)\end{aligned}$$

and

$$\begin{aligned}\frac{\delta c_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} &= \frac{\delta i_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}}g_{f'}^{(t')(\nu+1\tau)} + \frac{\delta g_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}}i_{f'}^{(t')(\nu+1\tau)}\\ &+ \frac{\delta f_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}}c_{f'}^{(t')(\nu\tau)}\,. \qquad (6.90)\end{aligned}$$

We continue our journey

$$\begin{aligned}\frac{\delta i_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} &= i_{f'}^{(t')(\nu+1\tau)}\left[1-i_{f'}^{(t')(\nu+1\tau)}\right]\Theta_f^{i_\nu(\nu+1)f'}J_f^{(tt')(\nu\tau)}\,,\\ \frac{\delta f_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} &= f_{f'}^{(t')(\nu+1\tau)}\left[1-f_{f'}^{(t')(\nu+1\tau)}\right]\Theta_f^{f_\nu(\nu+1)f'}J_f^{(tt')(\nu\tau)}\,,\\ \frac{\delta g_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} &= \left[1-\left(g_{f'}^{(t')(\nu+1\tau)}\right)^2\right]\Theta_f^{g_\nu(\nu+1)f'}J_f^{(tt')(\nu\tau)}\,, \qquad (6.91)\end{aligned}$$

and our notations now come handy

$$\frac{\delta h_{f'}^{(t')(\nu+1\tau)}}{\delta h_f^{(t)(\nu\tau)}} = J_f^{(tt')(\nu\tau)}H_{ff'}^{(t)(\nu\tau)_\nu}\,. \qquad (6.92)$$

This formula also allows us to compute the second term for $\delta_f^{(t)(N-2\tau)}$. In a totally similar manner

$$\frac{\delta h_{f'}^{(t')(\nu\tau+1)}}{\delta h_f^{(t)(\nu\tau)}} = J_f^{(tt')(\nu\tau)}H_{ff'}^{(t)(\nu-1\tau+1)_\tau}\,. \qquad (6.93)$$

Going back to our general formula

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\rm mb}} J_f^{(tt')(\nu\tau)} \left[\sum_{f'=0}^{F_{\nu+1}-1} H_{ff'}^{(t)(\nu\tau)_\nu} \delta_{f'}^{(t')(\nu\tau)} \right. \\ \left. + \sum_{f'=0}^{F_\nu-1} H_{ff'}^{(t)(\nu-1\tau+1)_\tau} \delta_{f'}^{(t')(\nu-1\tau+1)} \right] , \tag{6.94}$$

and as in the RNN case, we re-express it as (defining $b_0 = \nu$ and $b_1 = \tau$)

$$\delta_f^{(t)(\nu-1\tau)} = \sum_{t'=0}^{T_{\rm mb}} J_f^{(tt')(\nu\tau)} \sum_{\epsilon=0}^{1} \sum_{f'=0}^{F_{\nu+1-\epsilon}-1} \mathcal{H}_{ff'}^{(t')(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}} \delta_{f'}^{(t')(\nu-\epsilon\tau+\epsilon)} . \tag{6.95}$$

This formula is also valid for $\nu = N-1$ if we define as for the RNN case

$$\mathcal{H}_{f'}^{(t')(N\tau)} = 1 , \qquad \Theta_f^{\nu(N)f'} = \Theta_f^{f'} , \tag{6.96}$$

6.C.2 LSTM Weight and coefficient updates: details

We want to compute

$$\Delta_{f'}^{\rho_\nu(\nu)f} = \frac{\partial}{\partial \Theta_{f'}^{\rho_\nu(\nu)f}} J(\Theta) \qquad \Delta_{f'}^{\rho_\tau(\nu)f} = \frac{\partial}{\partial \Theta_{f'}^{\rho_\tau(\nu)f}} J(\Theta) , \tag{6.97}$$

with $\rho = (f,i,g,o)$. First we expand

$$\Delta_{f'}^{\rho_\nu(\nu)f} = \sum_{\tau=0}^{T-1} \sum_{f''=0}^{F_\nu-1} \sum_{t=0}^{T_{\rm mb}-1} \frac{\partial h_{f''}^{(\nu\tau)(t)}}{\partial \Theta_{f'}^{\rho_\nu(\nu)f}} \frac{\partial}{\partial h_{f''}^{(\nu\tau)(t)}} J(\Theta) \\ = \sum_{\tau=0}^{T-1} \sum_{f''=0}^{F_\nu-1} \sum_{t=0}^{T_{\rm mb}-1} \frac{\partial h_{f''}^{(\nu\tau)(t)}}{\partial \Theta_{f'}^{\rho_\nu(\nu)f}} \delta_{f''}^{(\nu\tau)(t)} , \tag{6.98}$$

so that (with $\rho^{(\nu\tau)} = (\mathcal{F}, \mathcal{I}, \mathcal{G}, \mathcal{O})$) if $\nu = 1$

$$\Delta_{f'}^{\rho_\nu(\nu-)f} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\rm mb}-1} \rho_f^{(\nu\tau)(t)} \delta_f^{(\nu\tau)(t)} h_{f'}^{(\nu-1\tau)(t)} , \tag{6.99}$$

and else

$$\Delta^{\rho_\nu(\nu-)f}_{f'} = \sum_{\tau=0}^{T-1}\sum_{t=0}^{T_{\rm mb}-1} \rho_f^{(\nu\tau)(t)}\delta_f^{(\nu\tau)(t)}y_{f'}^{(\nu-1\tau)(t)} \;, \tag{6.100}$$

$$\Delta^{\rho_\tau(\nu)f}_{f'} = \sum_{\tau=1}^{T-1}\sum_{t=0}^{T_{\rm mb}-1} \rho_f^{(\nu\tau)(t)}\delta_f^{(\nu\tau)(t)}y_{f'}^{(\nu\tau-1)(t)} \;. \tag{6.101}$$

We will now need to compute

$$\Delta^{\beta(\nu\tau)}_{f} = \frac{\partial}{\partial\beta_f^{(\nu\tau)}}J(\Theta) \qquad\qquad \Delta^{\gamma(\nu\tau)}_{f} = \frac{\partial}{\partial\gamma_f^{(\nu\tau)}}J(\Theta)\;. \tag{6.102}$$

For that we need to look at

$$\begin{aligned}\Delta^{\beta(\nu\tau)}_{f} &= \sum_{f'=0}^{F_{\nu+1}-1}\sum_{t'=0}^{T_{\rm mb}-1}\frac{\partial h_{f'}^{(\nu+1\tau)(t')}}{\partial\beta_f^{(\nu\tau)}}\delta_{f'}^{(\nu\tau)(t')} + \sum_{f'=0}^{F_{\nu}-1}\sum_{t'=0}^{T_{\rm mb}-1}\frac{\partial h_{f'}^{(\nu\tau+1)(t')}}{\partial\beta_f^{(\nu\tau)}}\delta_{f'}^{(\nu-1\tau+1)(t')}\\ &= \sum_{t=0}^{T_{\rm mb}-1}\left\{\sum_{f'=0}^{F_{\nu+1}-1}H_{ff'}^{(t)(\nu\tau)}\delta_{f'}^{(t)(\nu\tau)} + \sum_{f'=0}^{F_{\nu}-1}H_{ff'}^{(t)(\nu-1\tau+1)}\delta_{f'}^{(t)(\nu\tau+1)}\right\}\;.\end{aligned} \tag{6.103}$$

and

$$\Delta^{\gamma(\nu\tau)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\tilde{h}_f^{(t)(\nu\tau)}\left\{\sum_{f'=0}^{F_{\nu+1}-1}H_{ff'}^{(t)(\nu\tau)}\delta_{f'}^{(t)(\nu\tau)} + \sum_{f'=0}^{F_{\nu}-1}H_{ff'}^{(t)(\nu-1\tau+1)}\delta_{f'}^{(t)(\nu-1\tau+1)}\right\}\;, \tag{6.104}$$

which we can rewrite as

$$\Delta^{\beta(\nu\tau)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\sum_{\epsilon=0}^{1}\sum_{f'=0}^{F_{\nu+1-\epsilon}-1}\mathcal{H}_{ff'}^{(t)(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}}\delta_{f'}^{(t)(\nu-\epsilon\tau+\epsilon)}\;, \tag{6.105}$$

$$\Delta^{\gamma(\nu\tau)}_{f} = \sum_{t=0}^{T_{\rm mb}-1}\tilde{h}_f^{(t)(\nu\tau)}\sum_{\epsilon=0}^{1}\sum_{f'=0}^{F_{\nu+1-\epsilon}-1}\mathcal{H}_{ff'}^{(t)(\nu-\epsilon\tau+\epsilon)_{b_\epsilon}}\delta_{f'}^{(t)(\nu-\epsilon\tau+\epsilon)}\;. \tag{6.106}$$

Finally, as in the RNN case

$$\Delta^{f}_{f'} = \frac{\partial}{\partial \Theta^{f}_{f'}} J(\Theta) \; . \tag{6.107}$$

We first expand

$$\Delta^{f}_{f'} = \sum_{\tau=0}^{T-1} \sum_{f''=0}^{F_N-1} \sum_{t=0}^{T_{\rm mb}-1} \frac{\partial h^{(t)(N\tau)}_{f''}}{\partial \Theta^{f}_{f'}} \delta^{(t)(N-1\tau)}_{f''} \tag{6.108}$$

so that

$$\Delta^{f}_{f'} = \sum_{\tau=0}^{T-1} \sum_{t=0}^{T_{\rm mb}-1} h^{(t)(N-1\tau)}_{f'} \delta^{(t)(N-1\tau)}_{f} \; . \tag{6.109}$$

6.D Peephole connexions

Some LSTM variants probe the cell state to update the gate themselves. This is illustrated in figure 6.7

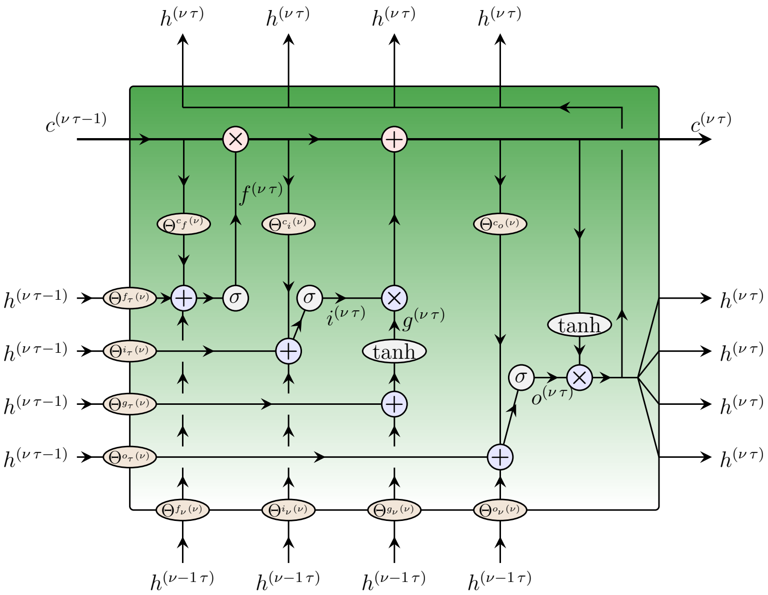


Figure 6.7: LSTM hidden unit with peephole

Peepholes modify the gate updates in the following way

$$i_f^{(\nu\tau)(t)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta^{i_\nu(\nu)f}_{f'} h^{(\nu-1\tau)(t)}_{f'} + \sum_{f'=0}^{F_\nu-1}\left[\Theta^{i_\tau(\nu)f}_{f'} h^{(\nu\tau-1)(t)}_{f'} + \Theta^{c_i(\nu)f}_{f'} c^{(\nu\tau-1)(t)}_{f'}\right]\right) , \tag{6.110}$$

$$f_f^{(\nu\tau)(t)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta^{f_\nu(\nu)f}_{f'} h^{(\nu-1\tau)(t)}_{f'} + \sum_{f'=0}^{F_\nu-1}\left[\Theta^{f_\tau(\nu)f}_{f'} h^{(\nu\tau-1)(t)}_{f'} + \Theta^{c_f(\nu)f}_{f'} c^{(\nu\tau-1)(t)}_{f'}\right]\right) , \tag{6.111}$$

$$o_f^{(\nu\tau)(t)} = \sigma\left(\sum_{f'=0}^{F_{\nu-1}-1} \Theta^{o_\nu(\nu)f}_{f'} h^{(\nu-1\tau)(t)}_{f'} + \sum_{f'=0}^{F_\nu-1}\left[\Theta^{o_\tau(\nu)f}_{f'} h^{(\nu\tau-1)(t)}_{f'} + \Theta^{c_o(\nu)f}_{f'} c^{(\nu\tau)(t)}_{f'}\right]\right) , \tag{6.112}$$

which also modifies the LSTM backpropagation algorithm in a non-trivial way. As it as been shown that different LSTM formulations lead to pretty similar results, we leave to the reader the derivation of the backpropagation update rules as an exercise.

Chapter 7

Conclusion

We have come to the end of our journey. I hope this note lived up to its promises, and that the reader now understands better how a neural network is designed and how it works under the hood. To wrap it up, we have seen the architecture of the three most common neural networks, as well as the careful mathematical derivation of their training formulas.

Deep Learning seems to be a fastly evolving field, and this material might be out of date in a near future, but the index approach adopted will still allow the reader – as it as helped the writer – to work out for herself what is behind the next state of the art architectures.

Until then, one should have enough material to encode from scratch its own FNN, CNN and RNN-LSTM, as the author did as an empirical proof of his formulas.

Bibliography

[1] **Nitish Srivastava et al.** "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.

[2] **Sergey Ioffe and Christian Szegedy**. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: (Feb. 2015).

[3] **Yann Lecun et al.** "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[4] **Karen Simonyan and Andrew Zisserman**. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[5] **Kaiming He et al.** "Deep Residual Learning for Image Recognition". In: 7 (Dec. 2015).

[6] **Alex Graves**. *Supervised Sequence Labelling with Recurrent Neural Networks*. 2011.

[7] **Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins**. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Comput.* 12.10 (Oct. 2000), pp. 2451–2471. ISSN: 0899-7667. DOI: 10.1162/089976600300015015. URL: http://dx.doi.org/10.1162/089976600300015015.

[8] **F. Rosenblatt**. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* (1958), pp. 65–386.

[9] **Yann LeCun et al.** "Effiicient BackProp". In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. ISBN: 3-540-65311-2. URL: http://dl.acm.org/citation.cfm?id=645754.668382.

[10] **Ning Qian**. "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1 (1999), pp. 145 –151. ISSN: 0893-6080. DOI: http://dx.doi.org/10.1016/S0893-6080(98)00116-6. URL: http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[11] **Yurii Nesterov**. "A method for unconstrained convex minimization problem with the rate of convergence O (1/k2)". In: *Doklady an SSSR*. Vol. 269. 3. 1983, pp. 543–547.

[12] **John Duchi, Elad Hazan, and Yoram Singer**. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12 (July 2011), pp. 2121–2159. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2021068.

[13] **Matthew D. Zeiler**. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). URL: http://dblp.uni-trier.de/db/journals/corr/corr1212.html#abs-1212-5701.

[14] **Diederik Kingma and Jimmy Ba**. "Adam: A Method for Stochastic Optimization". In: (Dec. 2014).

[15] **Rupesh K. Srivastava, Klaus Greff, and Jurgen Schmidhuber**. "Highway Networks". In: (). URL: http://arxiv.org/pdf/1505.00387v1.pdf.

[16] **Jiuxiang Gu et al.** "Recent Advances in Convolutional Neural Networks". In: *CoRR* abs/1512.07108 (2015).

[17] **Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton**. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[18] **Christian Szegedy et al.** "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842.

[19] **Gao Huang et al.** "Densely Connected Convolutional Networks". In: (July 2017).