`latex-mimosis`

A minimal, modern LaTeX package for typesetting your thesis

by

Bastian Rieck

A document submitted in partial fulfillment of the requirements for the degree of

Technical Report

at

MISKATONIC UNIVERSITY

Abstract

Scientific documents often use LaTeX for typesetting. While numerous packages and templates exist, it makes sense to create a new one. Just because.

Contents

1 Introduction

In which the reasons for creating this package are laid bare for the whole world to see and we encounter some usage guidelines.

This package contains a minimal, modern template for writing your thesis. While originally meant to be used for a Ph. D. thesis, you can equally well use it for your honour thesis, bachelor thesis, and so on—some adjustments may be necessary, though.

1.1 Why?

I was not satisfied with the available templates for LaTeX and wanted to heed the style advice given by people such as Robert Bringhurst [1] or Edward R. Tufte [2, 3]. While there *are* some packages out there that attempt to emulate these styles, I found them to be either too bloated, too playful, or too constraining. This template attempts to produce a beautiful look without having to resort to any sort of hacks. I hope you like it.

1.2 How?

The package tries to be easy to use. If you are satisfied with the default settings, just add

```
\documentclass{mimosis}
```

at the beginning of your document. This is sufficient to use the class. It is possible to build your document using either LaTeX|, XeLaTeX, or LuaLaTeX. I personally prefer one of the latter two because they make it easier to select proper fonts.

1.3 Features

The template automatically imports numerous convenience packages that aid in your typesetting process. Table 1.1 lists the most important ones. Let's briefly discuss some examples below. Please refer to the source code for more demonstrations.

Package	Purpose
`amsmath`	Basic mathematical typography
`amsthm`	Basic mathematical environments for proofs etc.
`booktabs`	Typographically light rules for tables
`bookmarks`	Bookmarks in the resulting PDF
`dsfont`	Double-stroke font for mathematical concepts
`graphicx`	Graphics
`hyperref`	Hyperlinks
`multirow`	Permits table content to span multiple rows or columns
`paralist`	Paragraph ('in-line') lists and compact enumerations
`scrlayer-scrpage`	Page headings
`setspace`	Line spacing
`siunitx`	Proper typesetting of units
`subcaption`	Proper sub-captions for figures

Table 1.1: A list of the most relevant packages required (and automatically imported) by this template.

1.3.1 Typesetting mathematics

This template uses `amsmath` and `amssymb`, which are the de-facto standard for typesetting mathematics. Use numbered equations using the `equation` environment. If you want to show multiple equations and align them, use the `align` environment:

$$V := \{1, 2, \ldots\} \tag{1.1}$$

$$E := \big\{(u, v) \mid \operatorname{dist}(p_u, p_v) \leq \epsilon\big\} \tag{1.2}$$

Define new mathematical operators using `\DeclareMathOperator`. Some operators are already pre-defined by the template, such as the distance between two objects. Please see the template for some examples. Moreover, this template contains a correct differential operator. Use `\diff` to typeset the differential of integrals:

$$f(u) := \int_{v \in \mathbb{D}} \operatorname{dist}(u, v) \, \mathrm{d}v \tag{1.3}$$

You can see that, as a courtesy towards most mathematicians, this template gives you the possibility to refer to the real numbers $\mathbb{R}$ and the domain $\mathbb{D}$ of some function. Take a look at the source for more examples. By the way, the template comes with spacing fixes for the automated placement of brackets.

1.3.2 Typesetting text

Along with the standard environments, this template offers `paralist` for lists within paragraphs. Here's a quick example: The American constitution speaks, among others, of (i) life (ii) liberty (iii) the pursuit of happiness. These should be added in equal measure to your own conduct. To

typeset units correctly, use the `siunitx` package. For example, you might want to restrict your daily intake of liberty to 750 mg.
Likewise, as a small pet peeve of mine, I offer specific operators for *ordinals*. Use `\th` to typeset things like July 4th correctly. Or, if you are referring to the 2nd edition of a book, please use `\nd`. Likewise, if you came in 3rd in a marathon, use `\rd`. This is my 1st rule.

1.4 Changing things

Since this class heavily relies on the `scrbook` class, you can use *their* styling commands in order to change the look of things. For example, if you want to change the text in sections to bold you can just use

```
\setkomafont{sectioning}{\normalfont\bfseries}
```

at the end of the document preamble—you don't have to modify the class file for this. Please consult the source code for more information.

2 Technology

In this chapter I'm describing the technology that I used to implement the set of utilities and experiments described in section ??? This chapter is organized as follows. In 2.1 I'm describing what is TensorFlow and how its computational graphs work. In 2.2 I'm describing Keras, an high-level API for building Machine Learning models without working with a low-level library like TensorFlow. In 2.3 I'm describing CleverHans, a library to generate adversarial examples against a given model. In 2.4 I'm describing scikit-learn, a library for building and training Machine Learning models.

2.1 TensorFlow

TensorFlow is a C++ framework for Machine Learning released by Google. It uses a programming model called Data Flow that aims to allow distributed and parallel computations. While this *paradigm* makes TensorFlow suitable even for research and production environments, it can be quite daunting to use when tinkering. In fact, computational graphs are built and only later executed. This is counter-intuitive at first. For example, in the following snippet of code

```
>>> import tensorflow as tf
>>> symbol = tf.constant(42)
>>> symbol
<tf.Tensor 'Const:0' shape=() dtype=int32>
```

`symbol` doesn't contain a reference to the integer 42. Instead it contains a reference to a node of the computational graph (which will always output 42). In general until you don't run the computational graph it's hard to determine what's going to be the value of a tensor. That's slowing down exploratory analysis.

2.1.1 About computational graphs

TensorFlow computational graphs are a fundamental part of the framework. A computational graph is a directed graph representing a computation involving tensors. A tensor in TensorFlow is a multi-dimensional array: it can be a scalar, a matrix, a batch of RGB images (which is a 4D vector), ... Each node represents an operation on tensors, while edges represent tensors. This distinction between nodes and edges is more important from a formal standpoint than a practical one. Actually thinking of tensors as nodes makes no difference to the best of my knowledge.
Figure 2.1 shows a very simple computational graph. It's the one associated to an addition of two `tf.float32`'s (i.e. two tensors made of 32-bit float numbers). The first two nodes `a` and `b` outputs

a tensor each. Those are used to feed the `add` node that will output the tensor resulting from the sum of `a` and `b`.

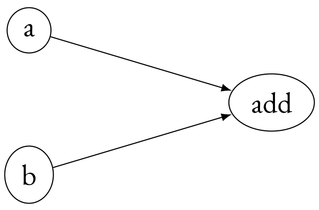

Figure 2.1: An easy computational graph

A more interesting example would be a matrix multiplication. To remain on topic we'll use matrix multiplication in TensorFlow to implement a neural network. The neural network we want to implement consists of 25 input neurons and 10 output neurons. A graphical representation of that is given in Figure 2.2.

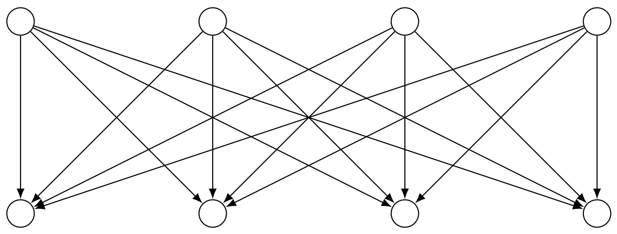

Figure 2.2: A simple neural network

We'll need the weights and the bias and do

$$X * W + b$$

where $W \in \mathbb{R}^{4\times 4}$ and $b \in \mathbb{R}^4$, while X is a *batch* of input vectors in $\mathbb{R}^4$. If you imagine the input is a flattened image of 2 by 2 pixels, this neural network learns how to classify 4 different classes of images.

```
>>> import tensorflow as tf
>>> W = tf.random_normal(shape=(4, 4))
>>> b = tf.random_normal(shape=(4,))
>>> X = tf.placeholder(shape=(None, 4), dtype=tf.float32)
>>> logits = tf.matmul(X, W) + b
>>> probabilities = tf.nn.softmax(logits)
>>>
>>> probabilities
<tf.Tensor 'Softmax:0' shape=(?, 4) dtype=float32>
```

Notice that the Softmax function is applied to the result of the matrix multiplication. This is done for various reasons. One of them is normalize the result of the neural network in $[0, 1]$. Also, it introduces non-linearity in the model allowing the neural network to compute any function – what's known as the "Universal approximation theorem".

To get results out of the graph you need what's called a session in TensorFlow. Basically a session *powers on* the graph, allowing you to run the computational graph with your own data and get actual tensors of actual numbers out of it. In the example above we can use a session to get probabilities out of our neural network for a batch of 2×2 images.

```
>>> import numpy as np
>>> my_batch = np.random.rand(10, 4) # create a batch 10 images of 4x4 pixels
>>> my_batch
array([[0.54485176, 0.33854871, 0.45185129, 0.79884188],
       [0.41204776, 0.23552753, 0.04101023, 0.47883844],
       [0.25544491, 0.7610509 , 0.49307137, 0.6098213 ],
       [0.02545156, 0.70459456, 0.22067103, 0.64743811],
       [0.92359354, 0.96497353, 0.45790538, 0.49380769],
       [0.13330072, 0.22947966, 0.02996348, 0.69954114],
       [0.38397249, 0.30473362, 0.87023559, 0.90153084],
       [0.77056319, 0.94843128, 0.39095345, 0.50572861],
       [0.90112077, 0.19240995, 0.48437166, 0.46200152],
       [0.98589042, 0.2013479 , 0.86091217, 0.55886214]])
>>> with tf.Session() as session:
...     # we're _feeding_ the X placeholder with an actual numpy array
...     session.run(probabilities, feed_dict={X: my_batch})
...
array([[0.03751625, 0.8651346 , 0.05245555, 0.04489357],
       [0.11873736, 0.6301607 , 0.17646323, 0.07463871],
       [0.06252376, 0.82338244, 0.07284217, 0.04125158],
       [0.12086256, 0.6911012 , 0.14362463, 0.04441159],
       [0.03662292, 0.8575108 , 0.04834893, 0.05751737],
       [0.12984137, 0.63064647, 0.1834405 , 0.05607158],
       [0.01711418, 0.93650085, 0.02116078, 0.02522417],
       [0.04886489, 0.83023334, 0.06332602, 0.05757581],
       [0.033136  , 0.84808177, 0.05061754, 0.06816475],
       [0.01250888, 0.9262628 , 0.01797607, 0.0432523 ]], dtype=float32)
>>> #              ^^^^^^^^^ the last image is of the second class with a confidence of 92%
```

Of course building neural networks is pretty useless if you can't train them. In fact the probabilities above are totally non-sense: they're only based on the initial weights and bias randomly extracted from a normal distribution. There's no training set yet. To train a network in TensorFlow we need a `tf.Variable`. A Variable in TensorFlow is a tensor whose value persists across sessions.

2.2 Keras

Keras is a high level library for building Machine Learning models. It consists of a simple API and bindings for a backend of choice, most notably TensorFlow. When you use the library, you're encouraged to use *layers* instead of plain tensors. In fact the whole concept of TensorFlow tensors is hidden away by the library abstractions. To implement your model you stack layers. For example, you would stack a `Dense` layer and an `Activation` one to implement a simple neural network.

```
from keras.models import Sequential

model = Sequential([
  Dense(batch_input_shape=(None, 784), 10),
  Activation('softmax')
])
```

Under the hood, a `Layer` is simply a callable object whose `__call()__` method takes a TensorFlow tensor `X` and manipulates it. For example a simple custom one would be

```
from keras.engine.base_layer import Layer
import tensorflow as tf

class Add42(Layer):
    def __init__(self):
        self.fortytwo = tf.constant(42)

    def __call__(self, X):
        return tf.add(X, self.fortytwo)
```

While the idea is clean and beginner friendly, at the time of this writing the abstraction can become leaky when things goes wrong, needing the programmer to know about TensorFlow computational graphs to debug her program successfully.

2.3 CleverHans

CleverHans is a library written by Google for building adversarial examples. It implements a variety of attacks against neural networks (FGS, T-FGS, Carlini and Wagner, ...) and it's compatible with models built with Keras. It uses the same computational graphs of TensorFlow to generate adversarial examples; again, as you use the library more and more understanding computational graphs becomes a necessity.

2.4 Scikit-learn

Scikit-learn is a Python library written by Google providing a number of models, learning algorithms and other utilities for Machine Learning. It's perfect for fast prototyping as it uses a simple and consistent API and handles numpy arrays or even Python lists.

I've used scikit-learn to borror a couple of decomposition algorithms (PCA, FastICA, ...) without having to implement them. This required a bit of reasoning as reusing scikit-learn algorithms on TensorFlow graphs was not straightforward.

Acronyms

PCA	Principal component analysis
SNF	Smith normal form
TDA	Topological data analysis

Glossary

$\LaTeX$	A document preparation system
$\mathbb{R}$	The set of real numbers

1. R. Bringhurst. *The Elements of Typographic Style*. 4th edition. Hartley & Marks Publishers, Vancouver, British Columbia, Canada, 2012.
2. E. R. Tufte. *Envisioning information*. Graphics Press, Cheshire, CT, USA, 1990.
3. E. R. Tufte. *The visual display of quantitative information*. 2nd edition. Graphics Press, Cheshire, CT, USA, 2001.