Music generation using Deep Learning

Gábor Lant
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Budapest, PA 1111
lant.gabor98@gmail.com

Dániel M. Szalai
Department of Networked Systems and Services
Budapest University of Technology and Economics
Budapest, PA 1111
szalaimd@gmail.com

Attila Kádár
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, PA 1111
attilaka98@gmail.com

Abstract

In this paper we try to demonstrate how we can generate music using neural networks from sound files converted to images. The goal is to compare the performance of different types of neural networks and also different types of music to image conversions and how they affect the generated output.

1 Problem summary

Our idea was to convert our music data into pictures and then use some kind of neural network to produce similar pictures. Then we would convert our generated images back to music format and rate the quality. Originally we hoped we could use different styles of music to use as an input for our model and then it would generate a similar music but in a different style, but later we found this to be much more difficult. We will discuss this later in section **??**.

2 Data acquisition, description

For our data-set we had two criteria in mind: we needed .midi files and we wanted to use tracks similar to each other in style. The .midi files allow us to work with large data-sets (25- 100GB as uncompressed .wav files) without actually having to store them as .wavs thus taking up only minimal storage capacity (not counting the preprocessed files). Them being similar in style is not necessary, however we thought that it would be best for our first approach to only consider tracks from the same style. Luckily we found a well-made data-set called Magenta: Maestro **?**, that satisfies all our needs. Maestro is a dataset composed of over 200 hours of virtuosic piano performances.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

3 Data exploration

Our data has the following main features:

- Time: Absolute time, in terms of MIDI clocks, at which this event occurs. Meta-events for which time is not meaningful (for example, song title, copyright information, etc.) have an absolute time of 0.
- Type/Event: Name identifying the type of the record. Record types are text consisting of upper and lower case letters and the underscore ("_"), contain no embedded spaces, and are not enclosed in quotes.
- Channel: The channel identifier.
- Note: The currently played note (integer 21-109, a piano has 88 keys)
- Velocity: The matching note's volume at the moment.

Further details at midicsv's website [1]

4 Data preprocessing

Here we only explain the main steps and ideas of our preprocessing, because we include a well-commented source code, that produces our actual training (70%), validation (20%) and test (10%) data.

4.1 First approach - using the midi files

4.1.1 Encoding

We used a python package called **MidiFile** to process the midi files in the following way:

1. With the constructor of **Midifile** we can read a specific .midi file and get all the MIDI event messages presented in it.
2. From the aquired event messages we built a pandas dataframe which made data handling easier. The structure of the dataframe can be seen on figure 1
3. The time_elapsed column contains the same Time value as it is in the original .midi file. This column will help us to determine the starting and ending pixel of the note appearing on the generated image.
4. We created image files, where the pitch is representsand on the y axis and the time on the x axis (we created images with size of 88x200 pixels storing approximatelly 10 seconds long pieces of music). We used three different coding methods to compare their performance:
   - In the first method we used all the 4 channels of a pixel to store different infromations about every note: its length, velocity, which instrument played that note and finally the current tempo. An example of this encoding can be seen on figure 2. This encoding is very rare, since only the starting of notes are presented on these images, because the length of each note is stored in R channel.
   - The second method is very similar to the one presented above, one difference is that this encoding uses only 3 channels, because we observed that in our dataset the Tempo value is equal in every part of each song so it is not necessary to store that feature. Also in this encoding we don't store the length of notes, the value of R channel is either 125 (which means the beginning of a given note) or 255 (means that a note is sustained). So in this encoding the whole time intervall is painted where a given note is played. An example of this encoding can be seen on figure 3. The index of pixels between the

[1] MIDICSV: Convert MIDI File to and from CSV - Fourmilab. `https://www.fourmilab.ch/webtools/midicsv/`

	message_type	control	value	note	velocity	time_elapsed
3307	note_on	NaN	NaN	38	36	98997.0
3351	note_on	NaN	NaN	38	0	100169.0
4530	note_on	NaN	NaN	38	66	136768.0
4570	note_on	NaN	NaN	38	0	138004.0
5152	note_on	NaN	NaN	38	74	158054.0
...	...	...	...	...	...	...
25719	note_on	NaN	NaN	84	0	841996.0
26690	note_on	NaN	NaN	84	88	864532.0
26693	note_on	NaN	NaN	84	0	864616.0
28943	note_on	NaN	NaN	84	84	916035.0
28949	note_on	NaN	NaN	84	0	916150.0

15788 rows × 6 columns

Figure 1: Pandas dataframe containing midi events.

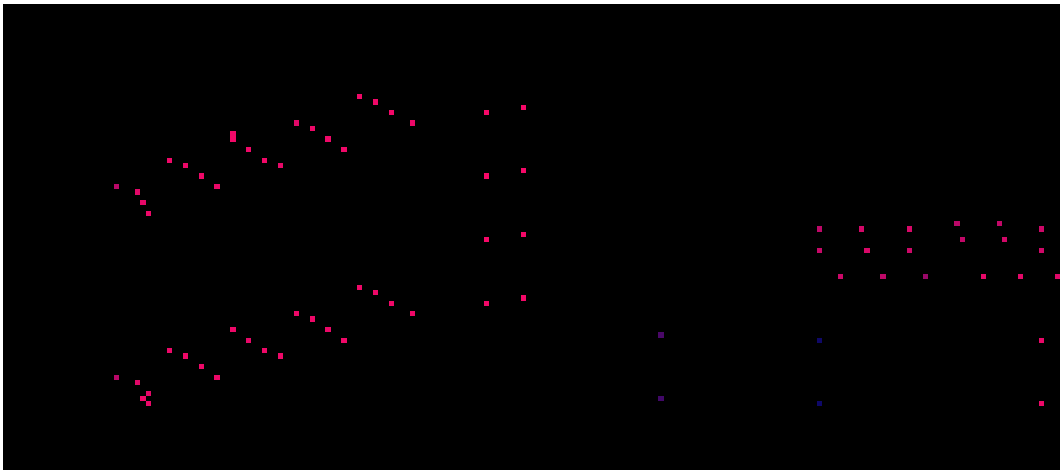


Figure 2: Encoding number 1.

note shuld be painted can be determined as follows:

$$
\begin{aligned}
starting\_time &= tempo \times \frac{\frac{t_0}{division}}{100000} seconds \\
starting\_pixel &= starting\_time \times pixel\_sec \\
ending\_time &= tempo \times \frac{\frac{t_1}{division}}{100000} seconds \\
ending\_pixel &= ending\_time \times pixel\_sec,
\end{aligned}
\tag{1}
$$

where pixel_sec is the time value corresponding to 1 pixel (it is 0.05 sec in our encoding)

Figure 3: Encoding number 2.

- The third encoding is a one-channeled version of the second one, where we doesn't store the velocity and instrument. This encoding was made only for testing the architecture with smaller dimensional inputs. This encoding is presented on figure 4

Figure 4: Encoding number 3.

The code for the first encoding method can be found in **note2img.ipynb** notebook file, while the code for the second and third method can be found in **Midi2Img.ipynb** notebook file.

4.1.2 Decoding

After generating images with different Deep learning architectures it was necessary to create .midi files from the generated images. For this purpose we used two python packages: **Midifile** and **midiutil**. The pitch is represented on the y axis, and the duration of the note can be computed from the x axis in the following way:

$$starting\_time = \frac{1000000 \times pixel\_sec \times staring\_pixel \times division}{tempo})) \tag{2}$$

$$duration = \frac{1000000 \times length \times division}{tempo})), \tag{3}$$

where division is the same division which can be found in the MIDI header message

Using the **midifile.addNote(track, channel, note, time, duration, velocity)** function we could add notes easily to our midi files.

The python code for the decoder can be found in **ImgToMidi.ipynb** notebook.

4.2 Second approach - Hilbert's Curves

In our first approach this will be the way we process the data, but we would like to note here, that we are thinking about different approaches as well. Another promising way can be the usage of Hilbert's curves. In a nuttshell: Hilbert's curves **?** is a way to map the 1D-like spectrogram to a 2D picture, in such way, that the notes close to each other on the spectrogram will also be close on the 2D picture. This approach might work better with convolutional networks since the small matrices, parsing the picture can actually extract features that correlates more.

4.3 Data augmentaion - MIDI approach

As we mentioned we used midi files collected in the MAESTRO dataset, but to widen our datas at our disposal we considered using two types of data augmentation methods:

- **Transposing music:** Before generating images from audio files we used an instrument called MIDICSV [2], which beside converting midi files to csv can also transpose music (shifting the whole melody to another key). So we could produce different (shifted along Y axis) images from the same MIDI file.
- **Adding noise to music:** In the real life it is also possible that a musician makes mistakes while playing (for example playes the wrong notes). So therefore we also added a Gaussian noise to a few notes in the music. The expected value of the noise is 0 and the standard deviation is 1 (actually the noise is a standard normal distribution). We moved the note along the y axis by the integer value of the noise added to it.

5 Network Architectures

5.1 Generative Adversarial Network (GAN)

In our first attempt to generate music using images we tried to use some type of GAN network. This type of architechture uses 2 neural nets to work against each other in an adversarial manner. The first network is the generator takes noise as input and tries to generate a new image from it. The second network is the discriminator. The discriminators job is to learn the dataset and try to differentiate between the generated and the real images. Using the discriminators output as a metric for the generator to become better. The 2 networks should be trained after one another but not simultaneously. Since these networks take a long time to train and optimize we choose an off the shelf model. We used the GAN models found in Erik Linder-Norén's repository as reference.

5.1.1 Different GAN approaches.

We tried to use the simple GAN model with the dataset mentioned in section 2. We didn't have much luck with this, the architechture seemed too simple to learn from our sparse dataset. Also this type of GAN is using Dense layers in the generator network, wich is not optimal to learn the structure of an image. The output was mostly noise with not much resemblance to the original dataset.

Figure 5: GAN generated output

[2] MIDICSV: Convert MIDI File to and from CSV - Fourmilab. `https://www.fourmilab.ch/webtools/midicsv/`

6 DCGAN

To compensate for the first models problems we settled on the DCGAN variant of the network. This type uses Deep Convolutional layers for both the generator and the discrimintator network. The idea was that convolution layers use a matrix of pixels as an input and it can learn the structure of an image much better. During the training this technique did work, however the generated outputs background was mostly gray and our images had black background. This made the discriminators job too easy and the loss didn't really converge to the expected values.

Figure 6: DCGAN generated output.

7 WGAN GF

The third approach was using WGAN GF found in paper . This uses a different method to calculate the error made by the generator network. This method seemed the most promising as the generated output started to look like the expected from around 2000 epochs. We let the network overfit our dataset so it would be as close to the original as possible. The output of the network can be seen in figure 7.

Figure 7: WGAN GF generated output.

8 Long Short Term Memory (LSTM)

As an alternative approach we tried to generate music using sequential recurrent architectures. We chose LSTM networks as the seed of this architecture. Long Short Term Memory networks are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn. It was also a very promissing solution since in music generation it is very important to remember the long-term dependencies in the played music. The key to LSTMs is the cell state (c), runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged (provides ability of remembering long-term dependencies). In addition an LSTM has three gates: input, output, forget gate, to protect and control the cell state. The structure of an LSTM cell can be seen on figure 8

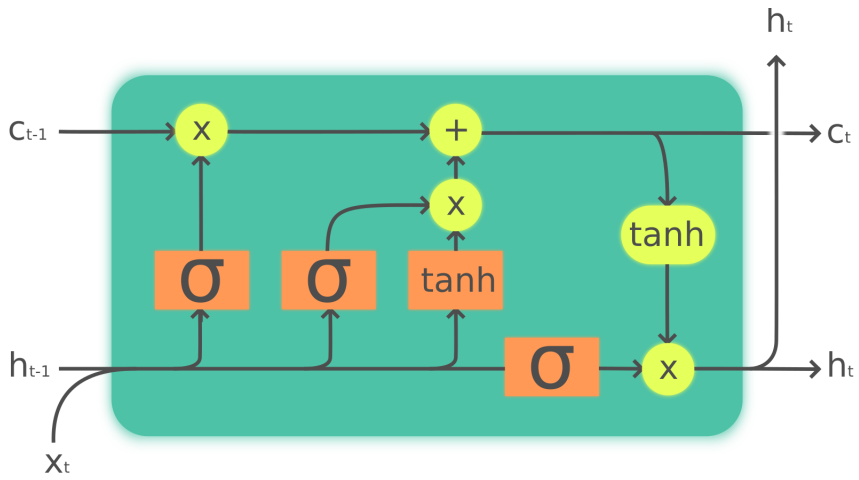


Figure 8: Structure of an LSTM cell.

8.1 Data generator

Since had so many images that couldn't fit in memory we had to use a data generator to read images "batch-by-batch" into the memory. This generator is also practical becaouse it can easily produce fix length data sequences from the images, which is necessary if we want to train an LSTM. For the different LSTM architectures this generator was able to create data sequences with a shape of $[(batch, in\_seq, dim), (batch, out\_seq, dim)]$, where in_seq and out_seq are respectively the size of the input and output sequences, dim is the dimension of a data point at one time stamp.

8.2 Training the basic LSTM model

At first we tried to train a sequential model with only one LSTM layer with 128 neurons and with a Fully connected layer attached to its output. The structure of this model can be seen on figure 9.

The output is a vector corresponding to one column in the generated image (that's why the size of this vector is 88, which is the heigth of the image).

8.2.1 Teacher forcing

Teacher forcing is a technique applied in the training process of a recurrent neural network where the target data (the ground truth) is passed as the next input to the RNN independent on what was the network's previous prediction. Using this method during the training we passed to LSTM 100 columns of an image as input data sequence and we expected to get the next column of the image as the predicted output. As testing procedure we gave a seed as the input of the LSTM (a 100 wide data sequence from the real images) and the LSTM generated the next few columns/time stamps of

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 128)	111104
dense_1 (Dense)	(None, 88)	11352
activation_1 (Activation)	(None, 88)	0

Total params: 122,456
Trainable params: 122,456
Non-trainable params: 0

Figure 9: Basic model with one LSTM layer.

the music. We tried to train this model on the dataset created by the first encoding presented in the section 4. This encoding was too sparse for this LSTM, it learned quickly to generate totally black images without painting any colour pixels onto it.

After training it on the dataset created by the second encoding process (using Teacher forcing) the results started to become a bit different from the first training. However it generated noisy images (as the one presented on figure 10). The first half of the image is the seed and the second (noisy) part is the generated part.

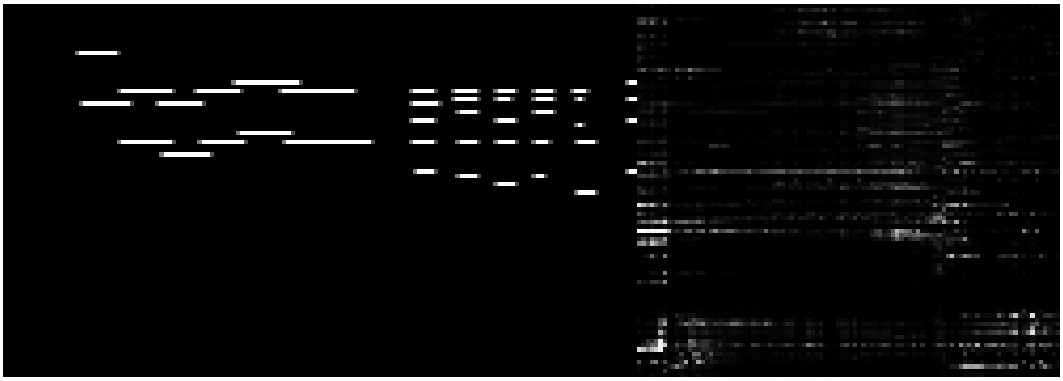


Figure 10: Result using Teacher forcing on basic LSTM.

8.2.2 Custom loss function

At the basic LSTM model we used Mean-Squared Error as loss function, but since that model generated very noise images we considered creating a custom loss function which is able to reduce the noise on the output.

Each value stored in pixel channels are real numbers from the [0, 1] intervall. That is y_true the expected output is also in this range. In this case:

$$\lim_{x \to \infty} \sqrt[x]{y\_true} = 1, \text{for every } y\_true > 0 \text{ value} \tag{4}$$

Now let's consider the following loss function:

$$C(y) = \lim_{x \to \infty} \frac{1}{M} \sum_{i=0}^{M} \alpha(y_i - \sqrt[x]{y\_true_i})^2 \sqrt[x]{y\_true_i} + \beta(y_i - \sqrt[x]{y\_true_i})^2 (1 - \sqrt[x]{y\_true_i}), \tag{5}$$

$$\text{where } M \text{ is the number of datapoints} \tag{6}$$

It can be seen that by adjusting the α and β parameters we can influence how the model punishes the following two cases:

- y_true is close to 1 and y is close to zero (this can be influenced by setting α)
- y_true is 0 and y is close to one (this can be influenced by setting β)

With $\alpha = 1.5$ and $\beta = 0.8$ we got the result presented on figure 11

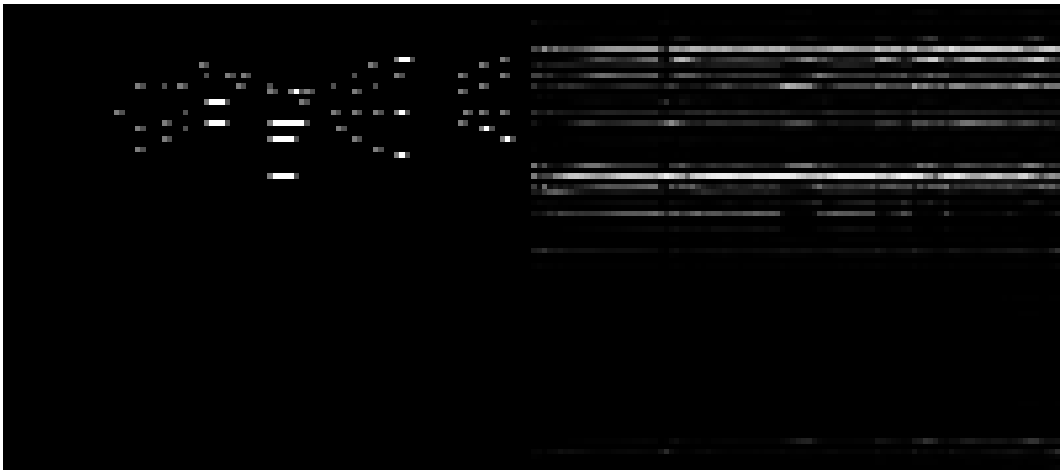


Figure 11: Result using Teacher forcing on LSTM with custom loss.

We noticed that after applying our custom loss function the noise along y axis disappeared, the network learned to continue the music in the same note interval as it was in the seed, it doesn't generate too high or two low notes. But also it can be seen that the network still doesn't know when to stop playing a specific note (the pixel intensity fades away along x axis instead separating notes by black pixels)

As a possible solution to this problem we tried to use the encoder-decoder based Seq2seq LSTM architecture and we stopped using Teacher forcing during the training process.

8.3 Seq2seq LSTM

Seq2seq models are used to generate output with various length from variable length input data sequence. In these models usually there is an encoder RNN which creates an interpretation of the input sequence and passes it to the decoder network, which also has its own input besides the information got from encoder. The output of the decoder will be the generated data. The structure of a Seq2seq model can be seen on figure 12

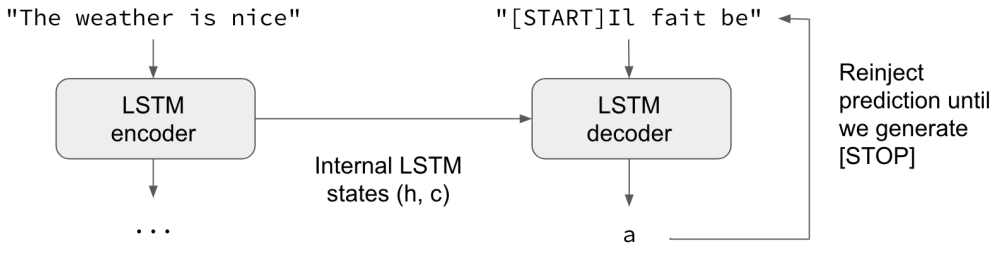


Figure 12: Seq2seq model structure

We applied LSTM cells as the seed of this seq2seq model. Instead of using keras's built-in Sequential model, we used Functional model so we could realize the generation of variable output sequence. The structure of our custom built seq2seg model can be seen on figure 13 In both the encoder and decoder network we put an LSTMCell into an RNN layer. In case of the encoder the number of

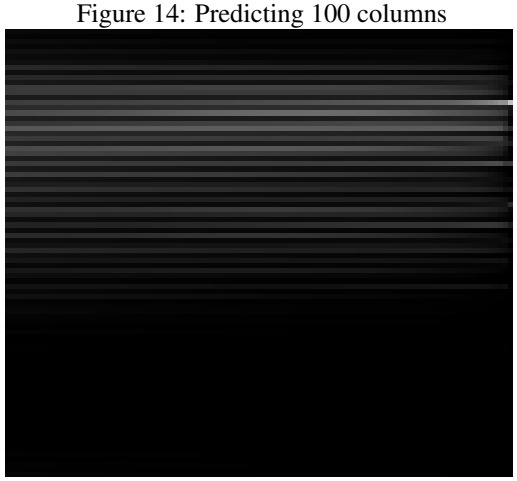
Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	(None, None, 88)	0	
input_8 (InputLayer)	(None, None, 88)	0	
rnn_7 (RNN)	[(None, 128), (None,	111104	input_7[0][0]
rnn_8 (RNN)	[(None, None, 128),	111104	input_8[0][0] rnn_7[0][1] rnn_7[0][2]
dense_7 (Dense)	(None, None, 128)	16512	rnn_8[0][0]
dense_8 (Dense)	(None, None, 88)	11352	dense_7[0][0]

Total params: 250,072
Trainable params: 250,072
Non-trainable params: 0

Figure 13: Seq2seq model summary

unfolded time steps during training are equal to the length of input data sequence. Similarly in case of the decoder the number of unfolded time steps are equal to the length of output sequence. On of the advantages of this model is that it can learn to predict multiple future output in contrast of Teacher forcing method which was trained to predict only one column of the input image using the previous columns. We trained this model to generate the next 10 and 100 columns (time steps) of the music based on the previous 100 columns (which corresponds to approximatelly 5 seconds of music). The results can be seen on figure 14.

Figure 14: Predicting 100 columns



The played notes are separated by black lines, which is a good sign: the model learned to distinguish different pitches. But the "white" lines are fading away here also, which means that the model couldn't learn the possible discrete values of the encoding.

To resolve this problem we considered transforming this into a kind of classification problem: the network needs to predict the class of each pixel of the next column of the image: is it a "silent" (black), "note started" (125/255 intensity gray) or "note sustained" (white) pixel. This classification problem can be realized with a categorical cross entropy loss function and softmax activations at the output fully connected layers.

Another possible direction for further researches can be using bi-directional LSTM cells instead of basic LSTM-s.

9 Submission of papers to NIPS 2016

There is a new style file for papers submitted in 2016!

NIPS requires electronic submissions. The electronic submission site is

`https://cmt.research.microsoft.com/NIPS2016/`

Please read carefully the instructions below and follow them faithfully.

9.1 Style

Papers to be submitted to NIPS 2016 must be prepared according to the instructions presented here. Papers may only be up to eight pages long, including figures. Since 2009 an additional ninth page *containing only acknowledgments and/or cited references* is allowed. Papers that exceed nine pages will not be reviewed, or in any other way considered for presentation at the conference.

The margins in 2016 are the same as since 2007, which allow for $\sim 15\%$ more words in the paper compared to earlier years.

Authors are required to use the NIPS LaTeX style files obtainable at the NIPS website as indicated below. Please make sure you use the current files and not previous versions. Tweaking the style files may be grounds for rejection.

9.2 Retrieval of style files

The style files for NIPS and other conference information are available on the World Wide Web at

`http://www.nips.cc/`

The file `nips_2016.pdf` contains these instructions and illustrates the various formatting requirements your NIPS paper must satisfy.

The only supported style file for NIPS 2016 is `nips_2016.sty`, rewritten for LaTeX 2_ε. **Previous style files for LaTeX 2.09, Microsoft Word, and RTF are no longer supported!**

The new LaTeX style file contains two optional arguments: `final`, which creates a camera-ready copy, and `nonatbib`, which will not load the `natbib` package for you in case of package clash.

At submission time, please omit the `final` option. This will anonymize your submission and add line numbers to aid review. Please do *not* refer to these line numbers in your paper as they will be removed during generation of camera-ready copies.

The file `nips_2016.tex` may be used as a "shell" for writing your paper. All you have to do is replace the author, title, abstract, and text of the paper with your own.

The formatting instructions contained in these style files are summarized in Sections 10, 11, and 12 below.

10 General formatting instructions

The text must be confined within a rectangle 5.5 inches (33 picas) wide and 9 inches (54 picas) long. The left margin is 1.5 inch (9 picas). Use 10 point type with a vertical spacing (leading) of 11 points. Times New Roman is the preferred typeface throughout, and will be selected for you by default. Paragraphs are separated by ½ line space (5.5 points), with no indentation.

The paper title should be 17 point, initial caps/lower case, bold, centered between two horizontal rules. The top rule should be 4 points thick and the bottom rule should be 1 point thick. Allow ¼ inch space above and below the title to rules. All pages should start at 1 inch (6 picas) from the top of the page.

For the final version, authors' names are set in boldface, and each name is centered above the corresponding address. The lead author's name is to be listed first (left-most), and the co-authors'

names (if different address) are set to follow. If there is only one co-author, list both author and co-author side by side.

Please pay special attention to the instructions in Section 12 regarding figures, tables, acknowledgments, and references.

11 Headings: first level

All headings should be lower case (except for first word and proper nouns), flush left, and bold.

First-level headings should be in 12-point type.

11.1 Headings: second level

Second-level headings should be in 10-point type.

11.1.1 Headings: third level

Third-level headings should be in 10-point type.

Paragraphs There is also a `\paragraph` command available, which sets the heading in bold, flush left, and inline with the text, with the heading followed by 1 em of space.

12 Citations, figures, tables, references

These instructions apply to everyone.

12.1 Citations within the text

The `natbib` package will be loaded for you by default. Citations may be author/year or numeric, as long as you maintain internal consistency. As to the format of the references themselves, any style is acceptable as long as it is used consistently.

The documentation for `natbib` may be found at

`http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf`

Of note is the command `\citet`, which produces citations appropriate for use in inline text. For example,

```
\citet{hasselmo} investigated\dots
```

produces

Hasselmo, et al. (1995) investigated…

If you wish to load the `natbib` package with options, you may add the following before loading the `nips_2016` package:

```
\PassOptionsToPackage{options}{natbib}
```

If `natbib` clashes with another package you load, you can add the optional argument `nonatbib` when loading the style file:

```
\usepackage[nonatbib]{nips_2016}
```

As submission is double blind, refer to your own published work in the third person. That is, use "In the previous work of Jones et al. [4]," not "In our previous work [4]." If you cite your other papers that are not widely available (e.g., a journal paper under review), use anonymous author names in the citation, e.g., an author of the form "A. Anonymous."

Table 1: Sample table title

	Part	
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

12.2 Footnotes

Footnotes should be used sparingly. If you do require a footnote, indicate footnotes with a number[3] in the text. Place the footnotes at the bottom of the page on which they appear. Precede the footnote with a horizontal rule of 2 inches (12 picas).

Note that footnotes are properly typeset *after* punctuation marks.[4]

12.3 Figures

All artwork must be neat, clean, and legible. Lines should be dark enough for purposes of reproduction. The figure number and caption always appear after the figure. Place one line space before the figure caption and one line space after the figure. The figure caption should be lower case (except for first word and proper nouns); figures are numbered consecutively.

You may use color figures. However, it is best for the figure captions and the paper body to be legible if the paper is printed in either black/white or in color.

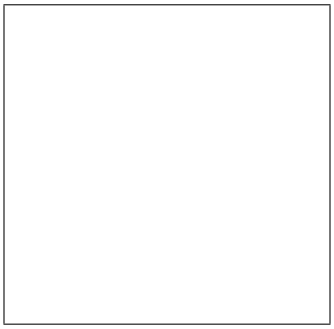


Figure 15: Sample figure caption.

12.4 Tables

All tables must be centered, neat, clean and legible. The table number and title always appear before the table. See Table 1.

Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lower case (except for first word and proper nouns); tables are numbered consecutively.

Note that publication-quality tables *do not contain vertical rules.* We strongly suggest the use of the `booktabs` package, which allows for typesetting high-quality, professional tables:

`https://www.ctan.org/pkg/booktabs`

This package was used to typeset Table 1.

[3] Sample of the first footnote.

[4] As in this example.

13 Final instructions

Do not change any aspects of the formatting parameters in the style files. In particular, do not modify the width or length of the rectangle the text should fit into, and do not change font sizes (except perhaps in the **References** section; see below). Please note that pages should be numbered.

14 Preparing PDF files

Please prepare submission files with paper size "US Letter," and not, for example, "A4."

Fonts were the main cause of problems in the past years. Your PDF file must only contain Type 1 or Embedded TrueType fonts. Here are a few instructions to achieve this.

- You should directly generate PDF files using `pdflatex`.
- You can check which fonts a PDF files uses. In Acrobat Reader, select the menu Files>Document Properties>Fonts and select Show All Fonts. You can also use the program `pdffonts` which comes with `xpdf` and is available out-of-the-box on most Linux machines.
- The IEEE has recommendations for generating PDF files whose fonts are also acceptable for NIPS. Please see `http://www.emfield.org/icuwb2010/downloads/IEEE-PDF-SpecV32.pdf`
- `xfig` "patterned" shapes are implemented with bitmap fonts. Use "solid" shapes instead.
- The `\bbold` package almost always uses bitmap fonts. You should use the equivalent AMS Fonts:

  ```
  \usepackage{amsfonts}
  ```

  followed by, e.g., `\mathbb{R}`, `\mathbb{N}`, or `\mathbb{C}` for $\mathbb{R}$, $\mathbb{N}$ or $\mathbb{C}$. You can also use the following workaround for reals, natural and complex:

  ```
  \newcommand{\RR}{I\!\!R} %real numbers
  \newcommand{\Nat}{I\!\!N} %natural numbers
  \newcommand{\CC}{I\!\!\!\!C} %complex numbers
  ```

  Note that `amsfonts` is automatically loaded by the `amssymb` package.

If your file contains type 3 fonts or non embedded TrueType fonts, we will ask you to fix it.

14.1 Margins in LaTeX

Most of the margin problems come from figures positioned by hand using `\special` or other commands. We suggest using the command `\includegraphics` from the `graphicx` package. Always specify the figure width as a multiple of the line width as in the example below:

```
\usepackage[pdftex]{graphicx} ...
\includegraphics[width=0.8\linewidth]{myfile.pdf}
```

See Section 4.4 in the graphics bundle documentation (`http://mirrors.ctan.org/macros/latex/required/graphics/grfguide.pdf`)

A number of width problems arise when LaTeX cannot properly hyphenate a line. Please give LaTeX hyphenation hints using the `\-` command when necessary.

Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments go at the end of the paper. Do not include acknowledgments in the anonymized submission, only in the final paper.

References

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font

size to `small` (9 point) when listing the references. **Remember that you can use a ninth page as long as it contains *only* cited references.**