



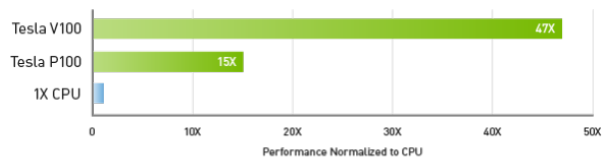
Introduction to Google Colab

2021. 09.07
Jonghwa lee

Deep learning requires a lot of hardware

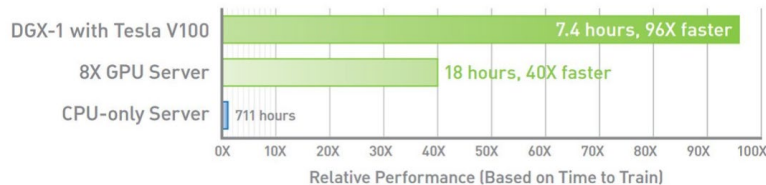
- To efficiently train a deep learning model, your hardware should be able to deal with a **huge amount of data at high speed**
- CPUs are fast but their bandwidth is lower than GPUs
 - GPUs can perform a large number of matrix multiplications at a time
 - It allows you to build and train complex deep learning models on large dataset efficiently
- However, high-performance GPUs are expensive

47X Higher Throughput Than CPU Server on Deep Learning Inference



Workload: ResNet-50 | CPU: 1X Xeon E5-2690v4 @ 2.6 GHz | GPU: Add 1X Tesla P100 or V100

NVIDIA DGX-1 Delivers 96X Faster Training



Workload: ResNet50, 90 epochs to solution | CPU Server: Dual Xeon E5-2699 v4, 2.6GHz

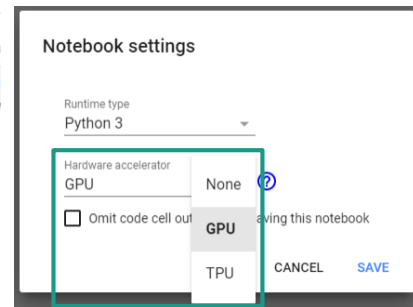
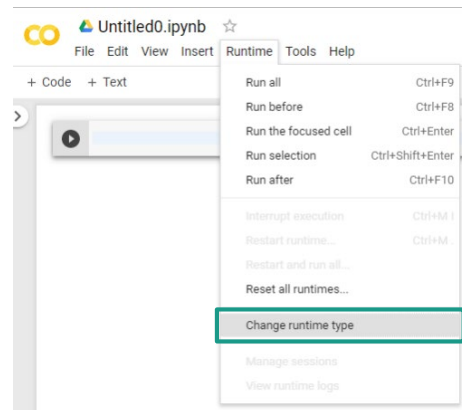
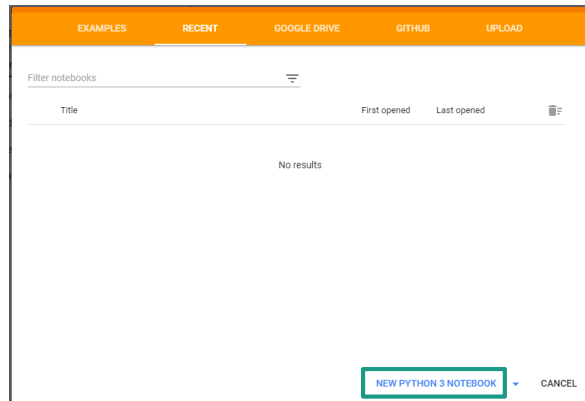


Google Colab to the rescue!

- Google Colab (<https://colab.research.google.com>) is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud
- It currently offers the computing services of an NVIDIA Tesla K80 GPU **for free**
- You can use the computing services for a maximum of 12 hours at a time
 - Basically, you are connected to a virtual machine that lives for a maximum of 12 hours at a time
 - All data including model parameters that aren't saved to the Google Drive before this period will be lost

Getting Started

- Go to <https://colab.research.google.com>
- Create a new Python 3 notebook
- From the 'Runtime > Change runtime type' menu, assign a hardware accelerator to your notebook



Notebook

- Notebook is a **list of cells** which contain either **explanatory text** or **executable code and its output**
- By clicking a cell, you can select the cell that you want to working with
- Text cells help you explain your notebook and code cells you wrote

Text cells use markdown syntax. You can also add math to text cells using LaTeX. Just place the statement within a pair of \$ signs.
For example `$$\sqrt{3x-1}+(1+x)^2$` becomes $\sqrt{3x-1} + (1+x)^2$.

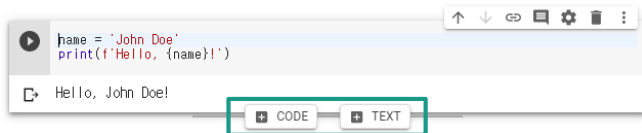
- Code cells contain executable Python code and its output

```
name = 'John Doe'
print(f'Hello, {name}!')
```

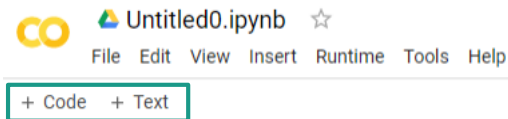
Hello, John Doe!

Working with Cells

- You can add new code or text cells using the **+CODE** and **+TEXT** buttons

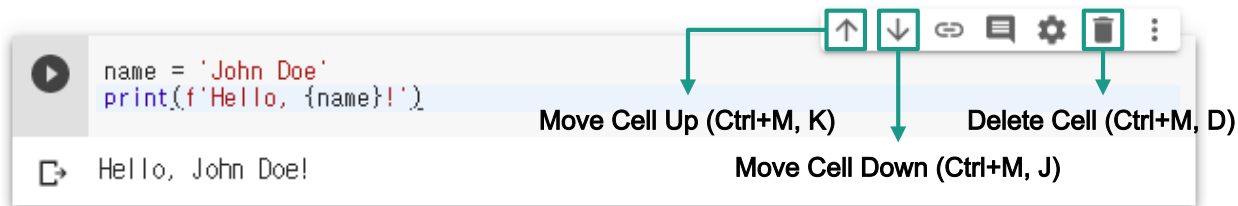


Buttons that appear when you hover between cells



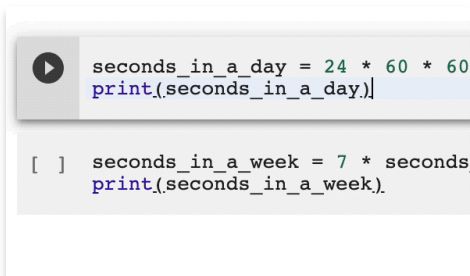
Buttons in the notebook toolbar

- By selecting a cell, you can move up, move down or delete the cell



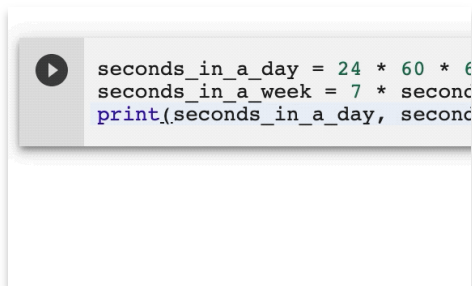
Working with Cells

- You can execute a selected code cell in the following ways:
 - Click the **Play button** in the left of the cell
 - Type **Ctrl+Enter** to run the cell in place
 - Type **Shift+Enter** to run the cell and move focus to the next cell
 - Type **Alt+Enter** to run the cell and insert a new code cell immediately below it



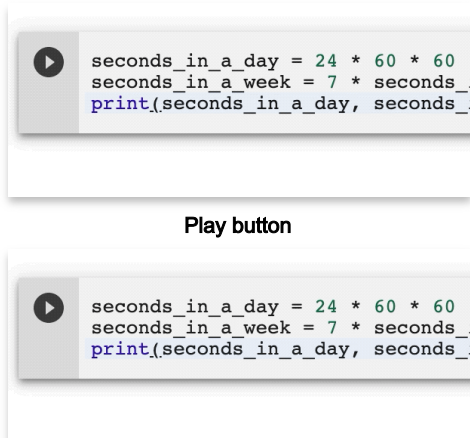
A Jupyter Notebook cell with a play button icon on the left. The cell contains two lines of Python code: `seconds_in_a_day = 24 * 60 * 60` and `print(seconds_in_a_day)`. Below the cell, a new cell is visible, containing the code: `[] seconds_in_a_week = 7 * seconds_in_a_day` and `print(seconds_in_a_week)`. This illustrates the effect of pressing Shift+Enter, which runs the current cell and moves the focus to the next cell below.

Shift + Enter



A Jupyter Notebook cell with a play button icon on the left. The cell contains two lines of Python code: `seconds_in_a_day = 24 * 60 * 60` and `print(seconds_in_a_day, seconds_in_a_week)`. Below the cell, a new empty code cell is inserted, illustrating the effect of pressing Alt+Enter, which runs the current cell and inserts a new cell immediately below it.

Alt + Enter



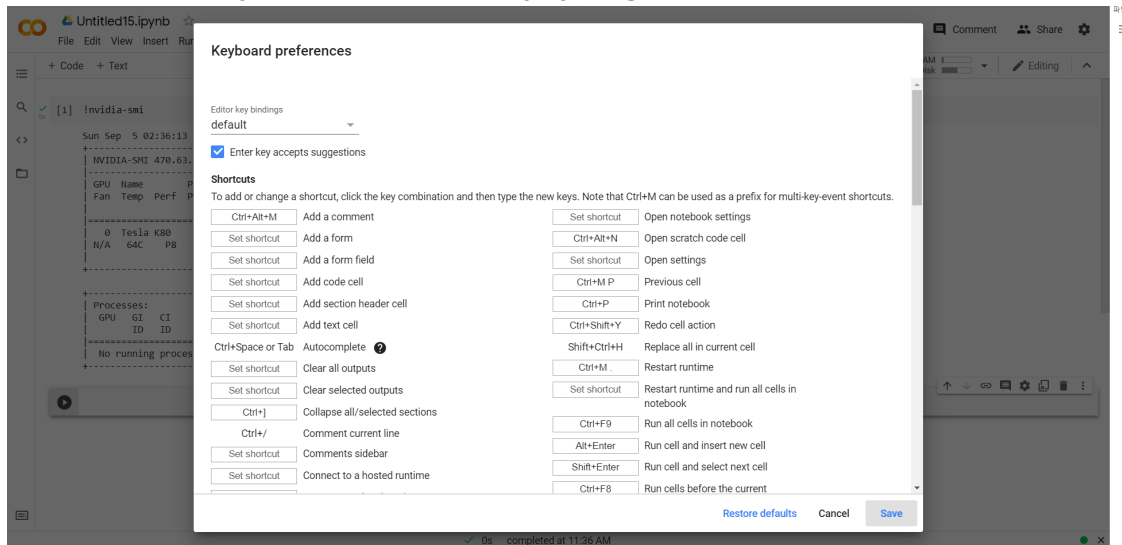
A Jupyter Notebook cell with a play button icon on the left. The cell contains two lines of Python code: `seconds_in_a_day = 24 * 60 * 60` and `print(seconds_in_a_day, seconds_in_a_week)`. This illustrates the effect of clicking the play button, which runs the code in the cell without changing the focus or inserting a new cell.

Play button

Ctrl + Enter

Working with Cells

- You can check list of keyboard shortcuts by typing `ctrl + m + h`





Working with Cells

- You can run any command on the system shell by prefixing it with

```
[1] !ls /
```

```
↳ bin      datalab  home    lib64  opt    run    swift      tmp    var
   boot    dev      lib     media  proc   sbin   sys        tools
   content etc     lib32   mnt    root   srv    tensorflow-2.0.0-rc0  usr
```

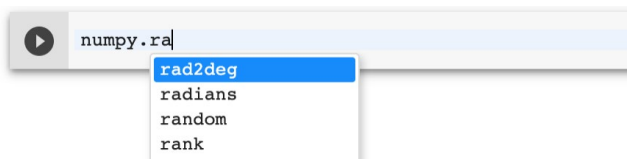
- You can install third-party libraries using package managers as needed

```
[ ] !apt install -y openjdk-11-jdk
    !pip install tqdm
```

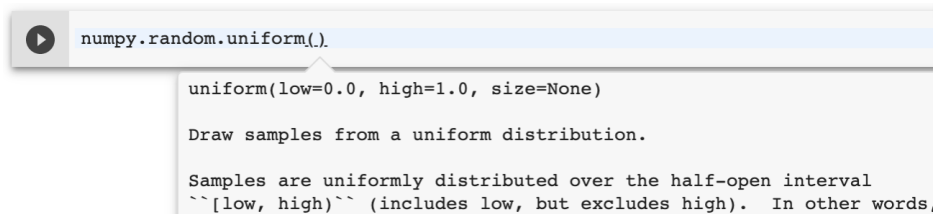
```
↳ Reading package lists... Done
   Building dependency tree
   Reading state information... Done
   The following additional packages will be installed:
     openjdk-11-jdk-headless
   Suggested packages:
     openjdk-11-demo openjdk-11-source visualvm
```

Working with Cells

- You can use tab completion to explore attributes of Python objects, as well as to quickly view documentation strings
 - As an example, if you press Tab after `numpy.ra`, you will see the list of available completions starting with `ra` within the `numpy` module

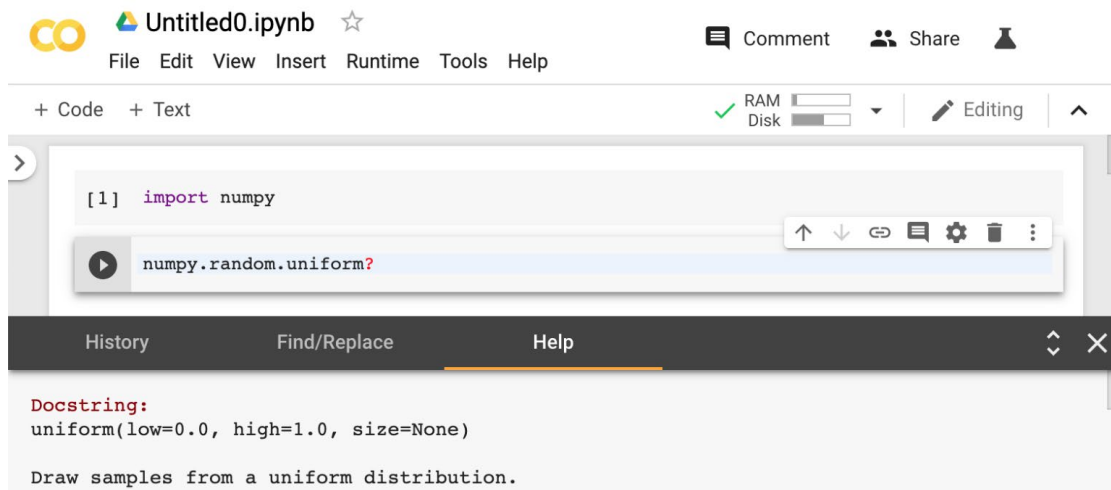


- You can see a pop-up of its documentation string



Working with Cells

- To open the documentation in a persistent pane at the bottom of your screen, add `?` after the object or method name and run the cell



The screenshot displays the JupyterLab interface. At the top, the title bar shows 'Untitled0.ipynb' with a star icon. Below it is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. To the right of the menu bar are icons for 'Comment', 'Share', and a user profile. Below the menu bar, there are tabs for '+ Code' and '+ Text'. On the right side of this bar, there are status indicators for 'RAM' and 'Disk' (both with green checkmarks), a dropdown menu, and a tab for 'Editing'. The main area contains two cells. The first cell is a code cell with the text `[1] import numpy`. The second cell is a help cell, indicated by a play button icon on the left, containing the text `numpy.random.uniform?`. At the bottom of the interface is a dark grey pane with three tabs: 'History', 'Find/Replace', and 'Help' (which is currently selected). The 'Help' pane displays the documentation for `numpy.random.uniform`, starting with 'Docstring:' followed by the function signature `uniform(low=0.0, high=1.0, size=None)` and the description 'Draw samples from a uniform distribution.'

Fetching Notebooks from GitHub

- You can fetch notebooks from GitHub repositories by searching them
 - In this lecture, we will upload all the notebooks to repositories by keai - kaist

The screenshot shows the JupyterLab interface with the 'GitHub' tab selected. The search bar contains 'keai-kaist' and a dropdown menu shows repository suggestions. The 'Lab1/Sep 7/Class1-2.ipynb' file is highlighted in the list below.

1. GitHub tab selected

2. Search bar contains 'keai-kaist'

3. Repository dropdown menu showing suggestions:

- keai-kaist/CS470-2021-2
- keai-kaist/CS470
- keai-kaist/CS470-2021-2
- keai-kaist/CS492F-Spring
- keai-kaist/SEPS32

4. 'Lab1/Sep 7/Class1-2.ipynb' file highlighted in the list