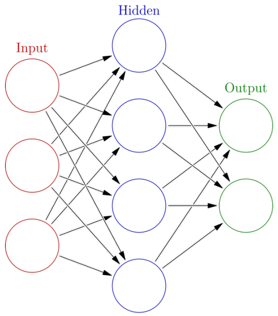
AI in Built Environment
DCP4300

Lec07-08: Deep Learning

Part C

Dr. Chaofeng Wang
Jianhao Gao (TA)
University of Florida
College of Design Construction and Planning

Build a neural network in TensorFlow/PyTorch



Software and Platforms

scikit-learn (traditional ML)

Tensorflow (deep neural networks)

PyTorch (deep neural networks)

They are packaged as Python modules,
already installed inside the Python running on Google Colab.

You have to install them if you want to run codes on your own computer instead of Google Colab.

```
pip install scikit-learn

pip install --upgrade pip
pip install tensorflow
```

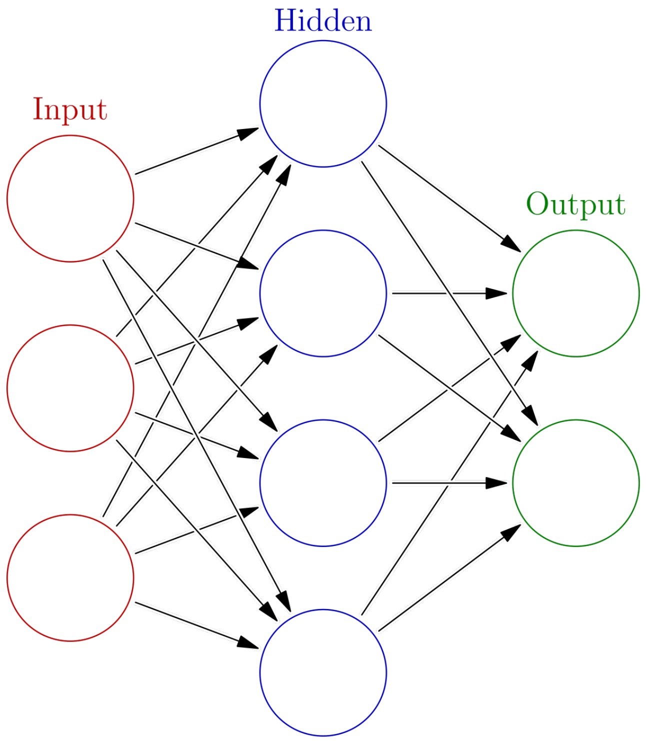
Instruction on PyTorch installation:
https://pytorch.org/get-started/locally/

We will use this notebook for demonstration:

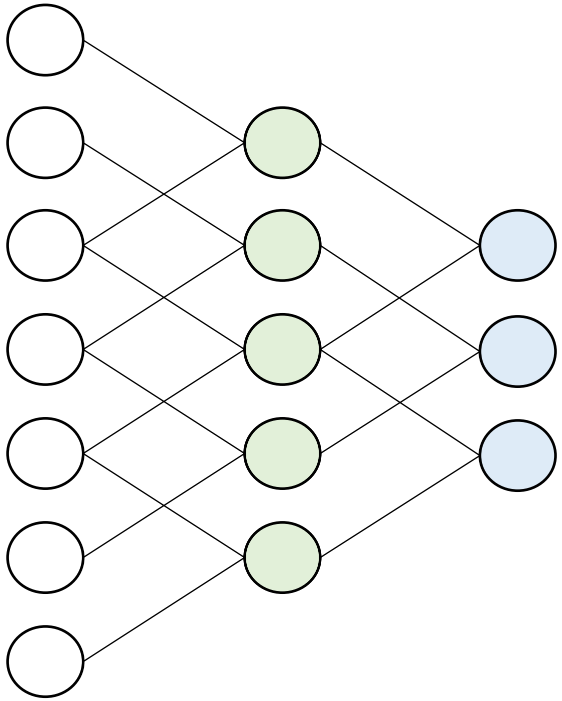
https://colab.research.google.com/drive/1L7f72NpyGN-KWNZzm6d9bElxebqmkTaf?usp=sharing

We’ll jump between the notebook and slides.

Fully (densely) connected layers



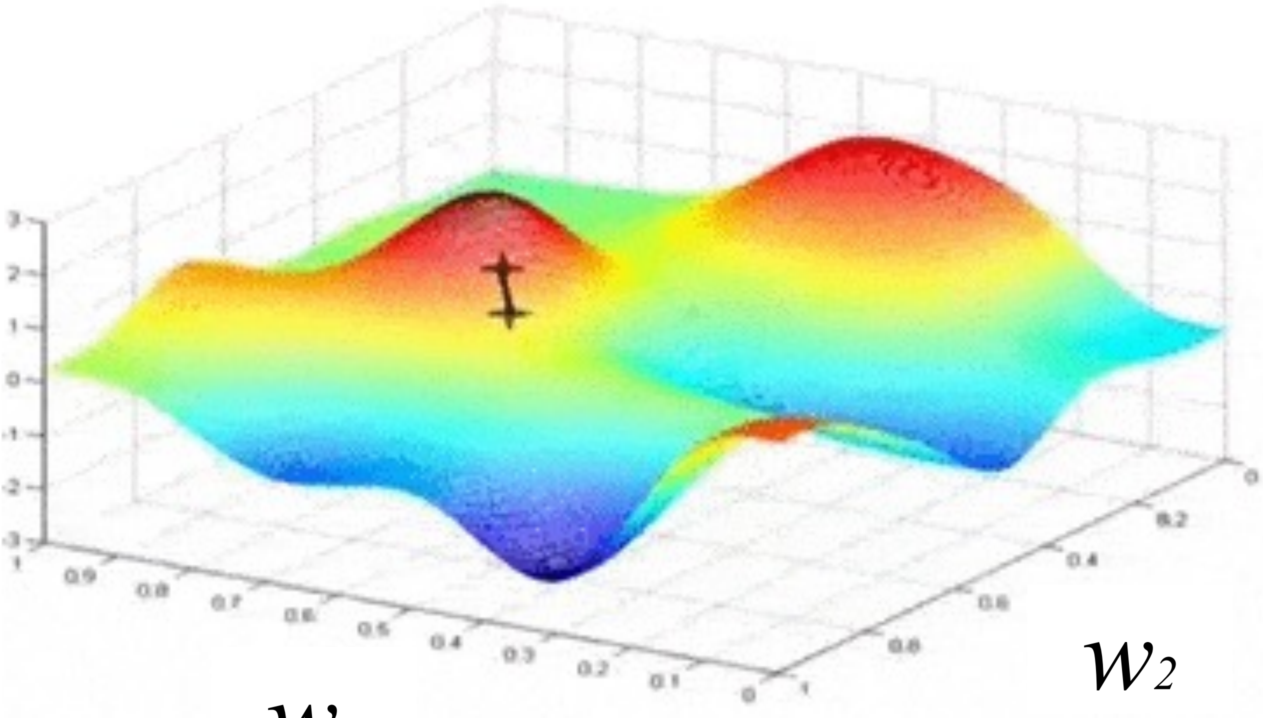
Fully connected layers



Non-Fully connected layers

Learning rate

$J(w)$



w_1 w_2

Gradient descent:

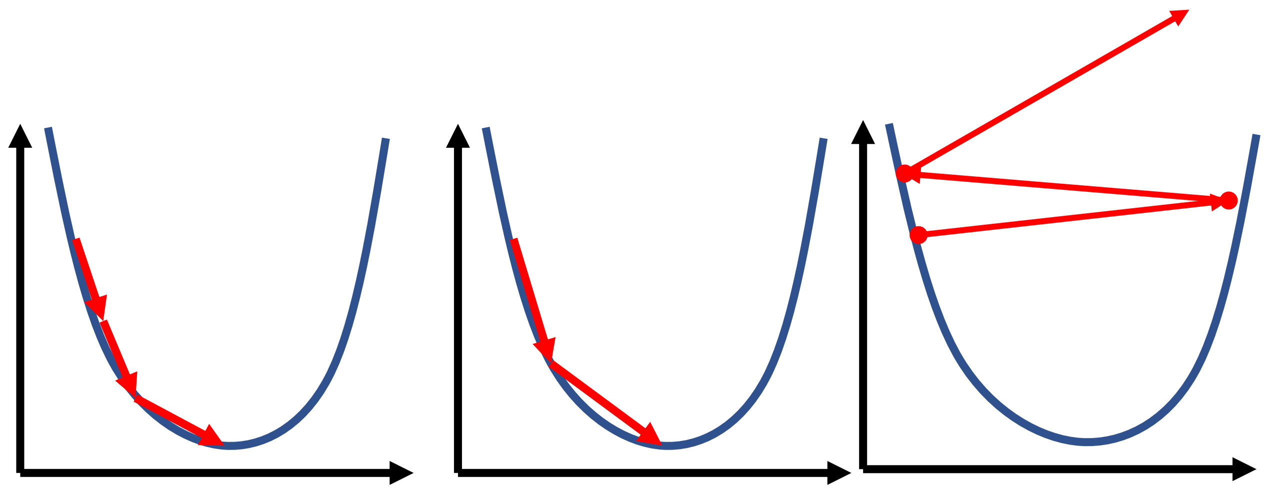
1. Compute the slope (gradient) at the current step $\frac{\partial J}{\partial w}$

2. Make a move in the direction opposite to the slope

The move = $-\eta \frac{\partial J}{\partial w}$

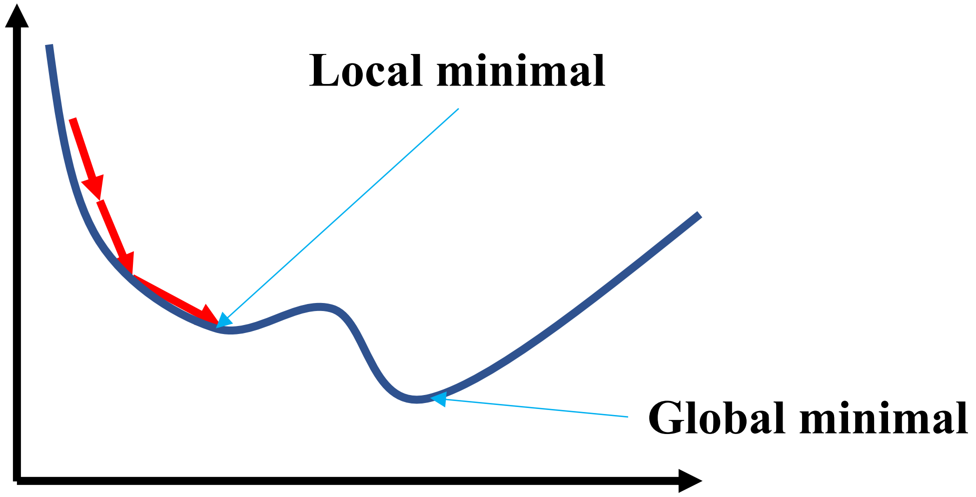
Learning rate

Learning rate



Too small Good Overshoot

Learning rate



It’s difficult to find a right learning rate.

We need advanced optimization algorithms (optimizers)

where learning rate can change adaptively during the learning process.

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

class SGD: Gradient descent (with momentum) optimizer.

class Adadelta: Optimizer that implements the Adadelta algorithm.
class Adagrad: Optimizer that implements the Adagrad algorithm.
class Adam: Optimizer that implements the Adam algorithm.
class RMSprop: Optimizer that implements the RMSprop algorithm.
class Adamax: Optimizer that implements the Adamax algorithm.
class Nadam: Optimizer that implements the NAdam algorithm.

class Ftrl: Optimizer that implements the FTRL algorithm.

Feature Engineering: One-hot Encoding of Categorical Output

For classification models (the output of the model is a categorical quantity)

	Label	Label Encoding	Label Encoding	One-hot Encoding	
	Cat	0	1	[1, 0, 0]	3 dummy variables
	Dog	1	2	[0, 1, 0]	
	Otter	2	3	[0, 0, 1]	

Feature Engineering: One-hot Encoding of Categorical Output

One-hot encoding vs dummy encoding

	Label	One-hot Encoding	N dummy variables	One-hot Encoding	N-1 dummy variables
	Cat	[1, 0, 0]		[1, 0]	
	Dog	[0, 1, 0]		[0, 1]	
	Otter	[0, 0, 1]		[0, 0]	

Feature Engineering: One-hot Encoding of Categorical Output

Encoding of features: Challenges of One-Hot Encoding - Dummy Variable Trap

Means: Dummy variables can be correlated (Multicollinearity)

One of the common ways to check for multicollinearity is the Variance Inflation Factor (VIF):

- VIF=1, Very Less Multicollinearity
- VIF<5, Moderate Multicollinearity
- VIF>5, Extreme Multicollinearity (This is what we have to avoid)

You can drop columns with high VIF

Feature Engineering: One-hot Encoding of Categorical Output

When to use a Label Encoding vs. One Hot Encoding
This question generally depends on your **dataset** and the **model** which you wish to apply. But still, a few points to note before choosing the right encoding technique for your model:

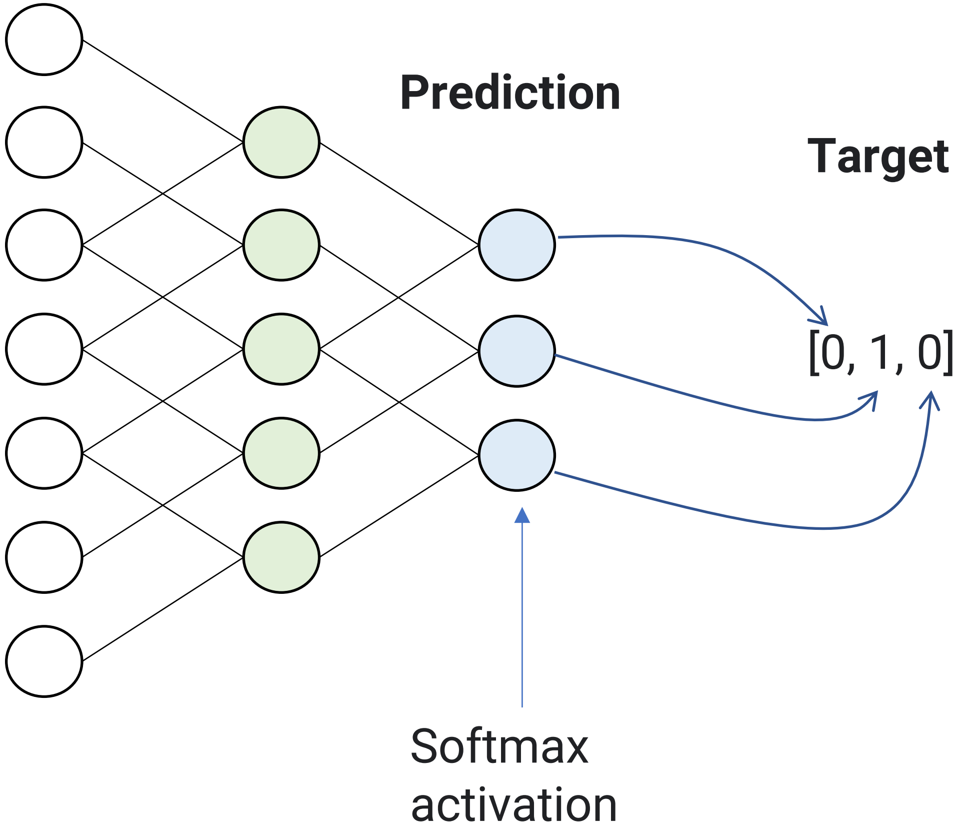
We apply One-Hot Encoding when:
1.The categorical feature is **not ordinal** (like the countries above)
2.The number of categorical features is less so one-hot encoding can be effectively applied

We apply Label Encoding when:
1.The categorical feature is **ordinal** (like Jr. kg, Sr. kg, Primary school, high school)
2. The number of categories is quite large as one-hot encoding can lead to high memory consumption

Question:
What if not ordinal with a large number of categories?

Feature Engineering: One-hot Encoding of Categorical Output

For classification models (the output of the model is a categorical quantity)



Feature Engineering: One-hot Encoding of Categorical Output

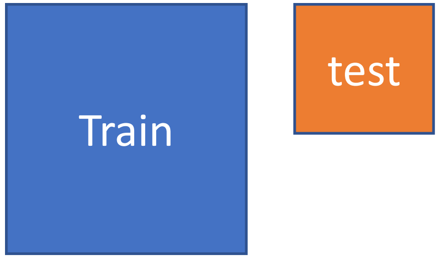
Softmax

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

$$\begin{bmatrix}1.2\\0.9\\0.4\end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix}0.46\\0.34\\0.20\end{bmatrix} \begin{array}{l}\text{P (class 1)}\\ \text{P (class 2)}\\ \text{P (class 3)}\end{array}$$

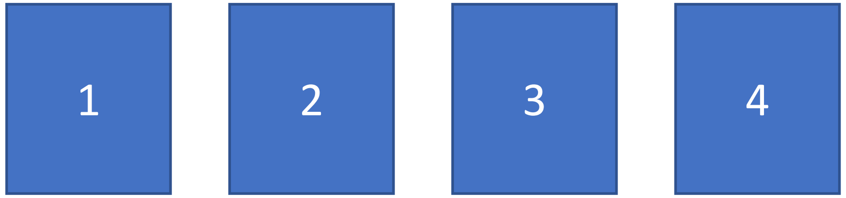
Probabilities, sum is 1.0

A single train/test split



Could be biased

k-Fold Cross Validation



Train: 1+2+3, Test: 4 -> Model 1
Train: 1+2+4, Test: 3 -> Model 2
Train: 1+3+4, Test: 2 -> Model 3
Train: 2+3+4, Test: 1 -> Model 4

Check if the data/model is biased