

Tutorial 3: More Networks and More Areas

1. From Redshift to Classification: A Simple Change
2. Deepening the Redshift Net
3. Applying the Redshift Net to Gravitational Wave

Yu Wang

ICRANet / INAF / University of Rome

Nov 4th, 2021

Online

TUTORIAL 3: MORE NETWORKS AND MORE AREAS

Deep Learning Procedure

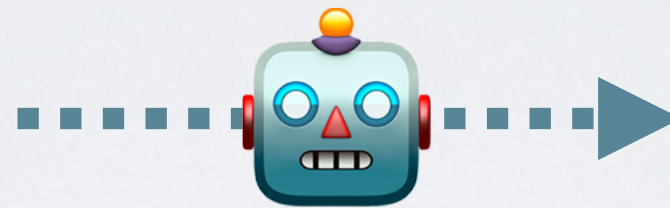
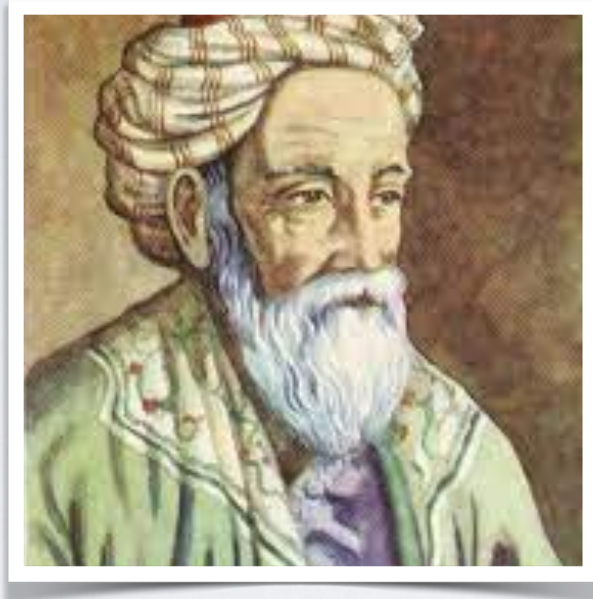
DEEP LEARNING

What can machine do ?

Input

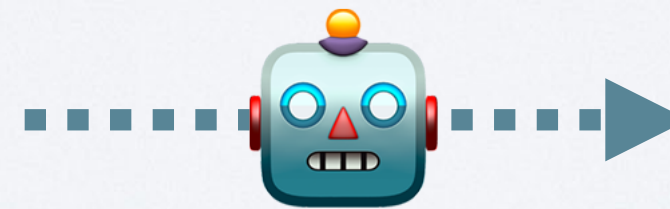
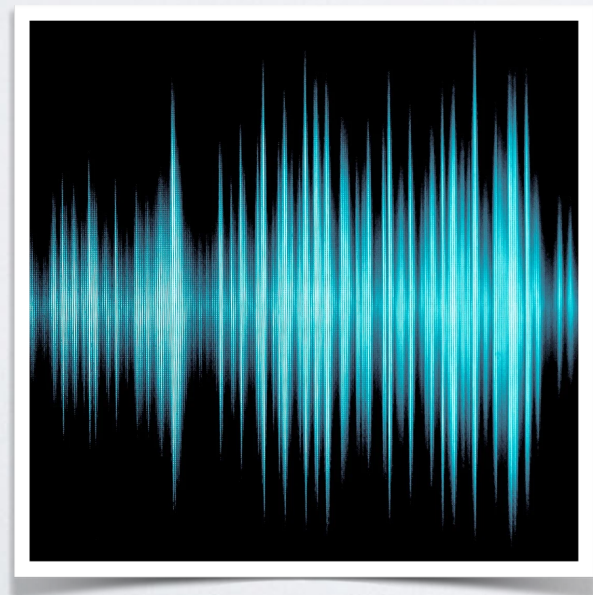
Output

Image Recognition



Omar Khayyam

Voice Recognition



Isfahan is half the world

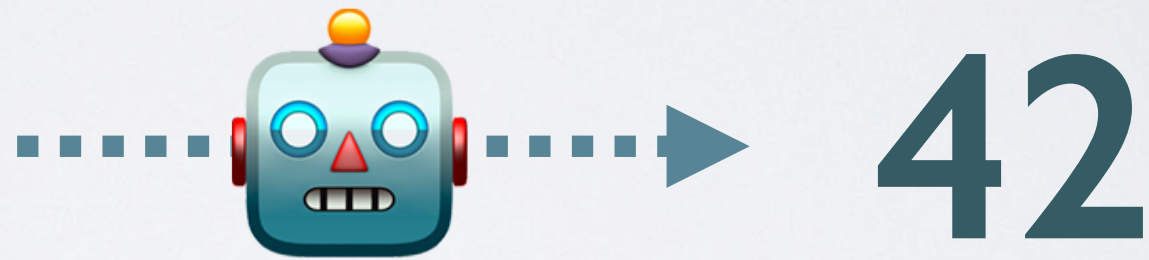
DEEP LEARNING

An answer from a given dataset

Data

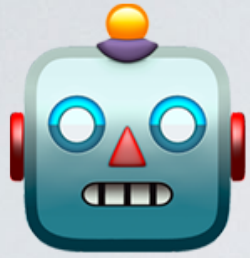


Answer



DEEP LEARNING

Machine as a *map* action.



$f: data \mapsto answer$

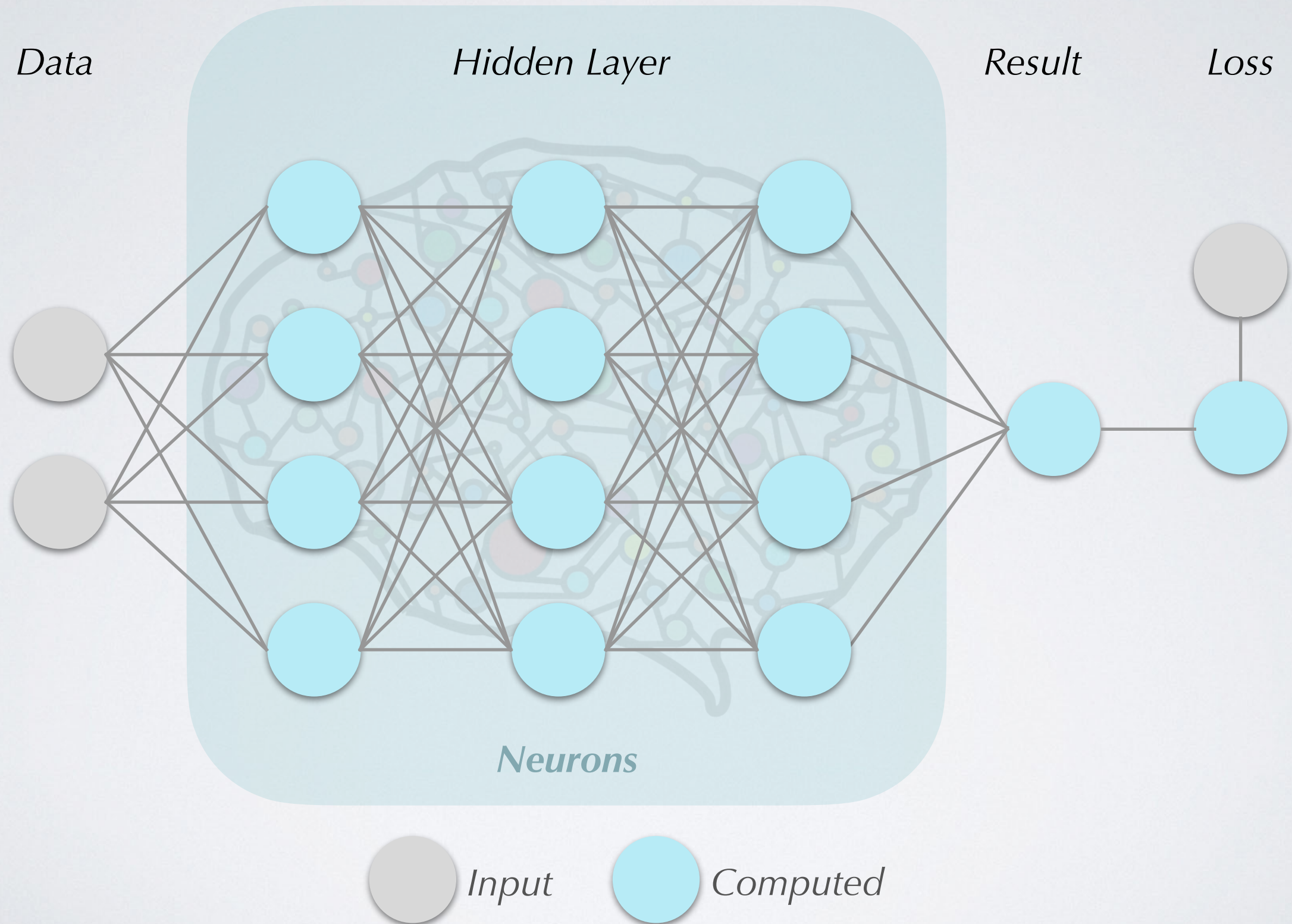
Deep learning:

f is constructed by many layers of neurons



DEEP LEARNING

A Simple Example - Fully Connected Net



DEEP LEARNING

A Simple Example - Fully Connected Net



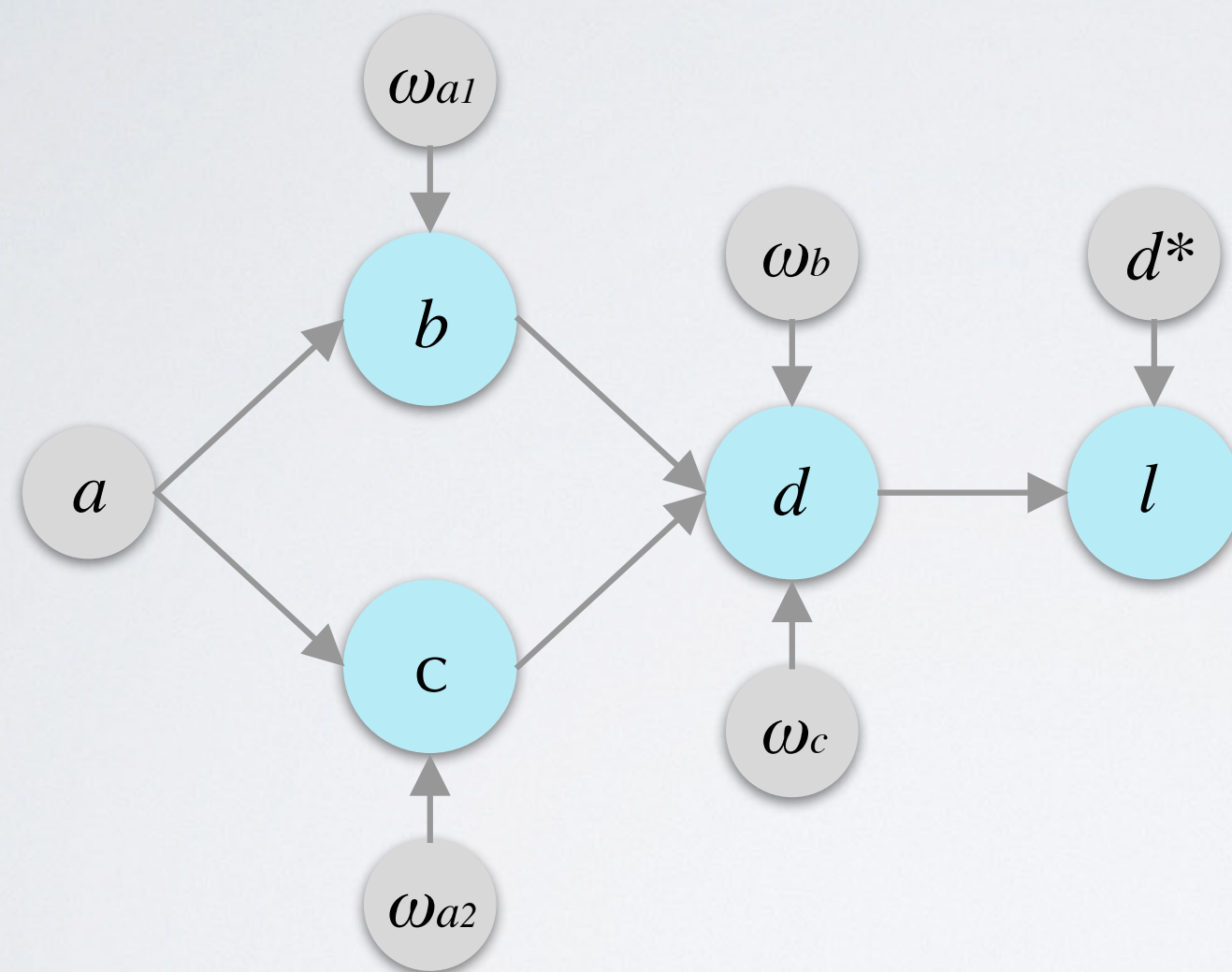
$$\vec{y} = A(\omega \cdot \vec{x} + b)$$

Output *Activation function* *Weight* *Input* *Offset*

DEEP LEARNING

A simple Net

Input Hidden Layer Output Loss



Forward Propagation

$$x = A(x^T \omega)$$
$$e = \text{loss}(d, d^*)$$

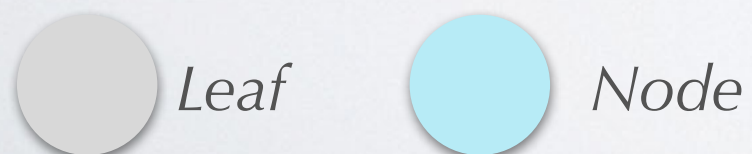


$$b = \omega_{a1} a$$

$$c = \omega_{a2} a$$

$$d = \omega_b b + \omega_c c$$

$$e = d^* - d$$



* For simplicity, the activation function (A) is ignored.

DEEP LEARNING

Optimization

For a give set of ω^* , the loss reaches 0:

$$\text{loss}(d, d^*) = 0$$

When $\omega = \omega^*$

To find ω^* , a simple and common way:

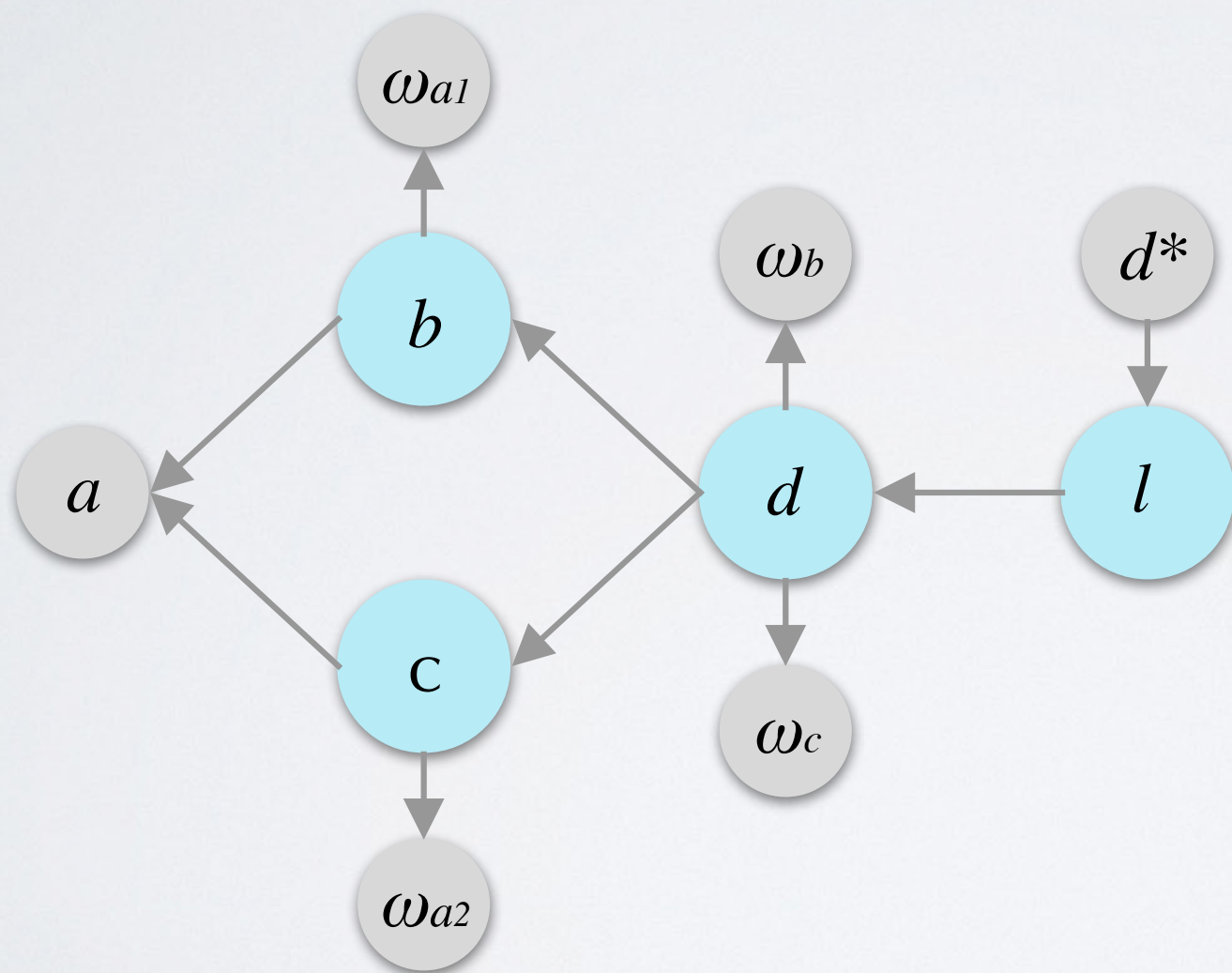
Gradient descent

repeat until convergence

$$\omega_{t+1} = \omega_t - \eta \frac{\partial \text{loss}}{\partial \omega_t}$$

Chain Rule of Derivatives

Input Hidden Layer Output Loss



$$\frac{\partial l}{\partial \omega_d} = \frac{\partial l}{\partial d} \frac{\partial d}{\partial \omega_d}$$

$$\frac{\partial l}{\partial \omega_c} = \frac{\partial l}{\partial d} \frac{\partial d}{\partial \omega_c}$$

$$\frac{\partial l}{\partial \omega_{a2}} = \frac{\partial l}{\partial d} \frac{\partial d}{\partial c} \frac{\partial c}{\partial \omega_{a2}}$$

$$\frac{\partial l}{\partial \omega_{a1}} = \frac{\partial l}{\partial d} \frac{\partial d}{\partial b} \frac{\partial b}{\partial \omega_{a1}}$$



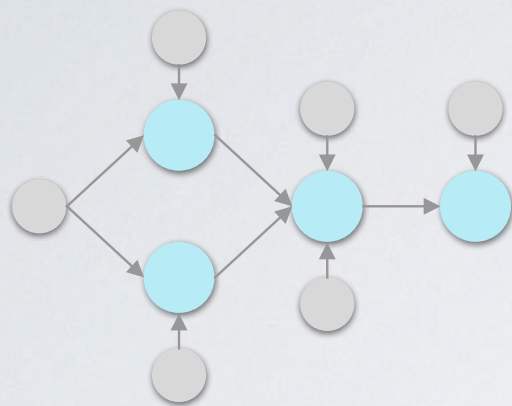
$$\omega_{t+1} = \omega_t - \eta \frac{\partial l}{\partial \omega_t}$$

DEEP LEARNING

Procedure of Training a Net

$$\text{normal}(\text{gain} \times \sqrt{\frac{6}{\text{fan_in} + \text{fan_out}}})$$

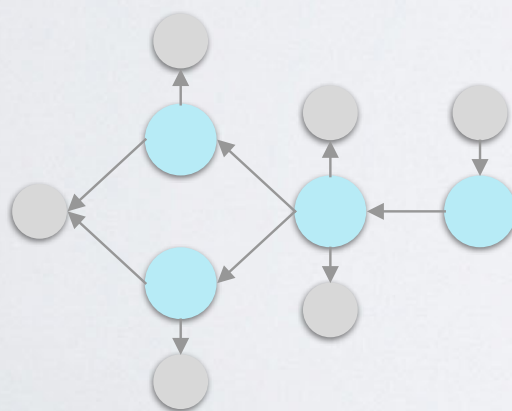
$$\text{uniform}(\text{gain} \times \sqrt{\frac{3}{\text{fan_mode}}})$$



$$\text{MSE}(x) = E((x - x^*)^2)$$

$$H(p, q) = - \sum p(x) \log q(x)$$

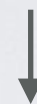
$$D_{KL}(p || q) = \sum p(x) \ln \frac{Q(x)}{P(x)}$$



$$\frac{\partial \text{loss}}{\partial \omega} \rightarrow 0$$

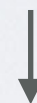
0. Initialization

Initialize the weight and offset by random or specific methods, for e.g., Xavier normal, He uniform.



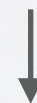
1. Forward Propagation

The input features go throughout all the layers with lots of matrix computation and non-linear activation functions, and finally make a prediction.



2. Compute the loss

Loss is the distance defined between the predicted label and the true label, for e.g., mean squared error, cross entropy, Kullback-Leibler divergence.



3. Back Propagation

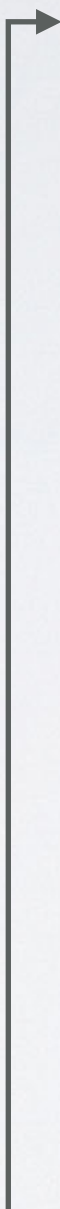
Loss back propagates all the net, and optimizes parameters by methods of, for e.g., stochastic gradient descent, RMSProp, Adam.



4. Convergence

The loss function reaches a global minimal or a local minimal.

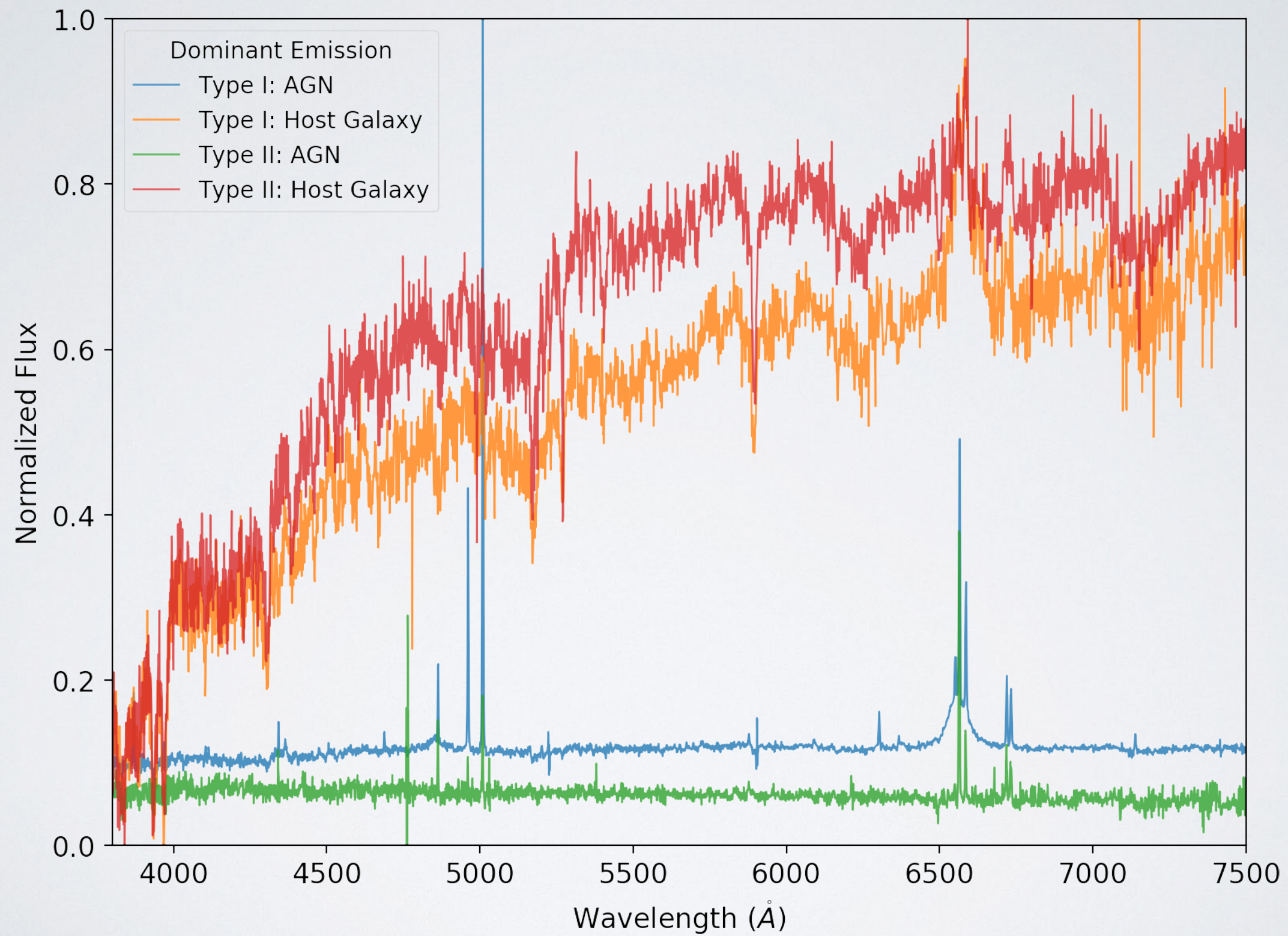
repeat until convergence



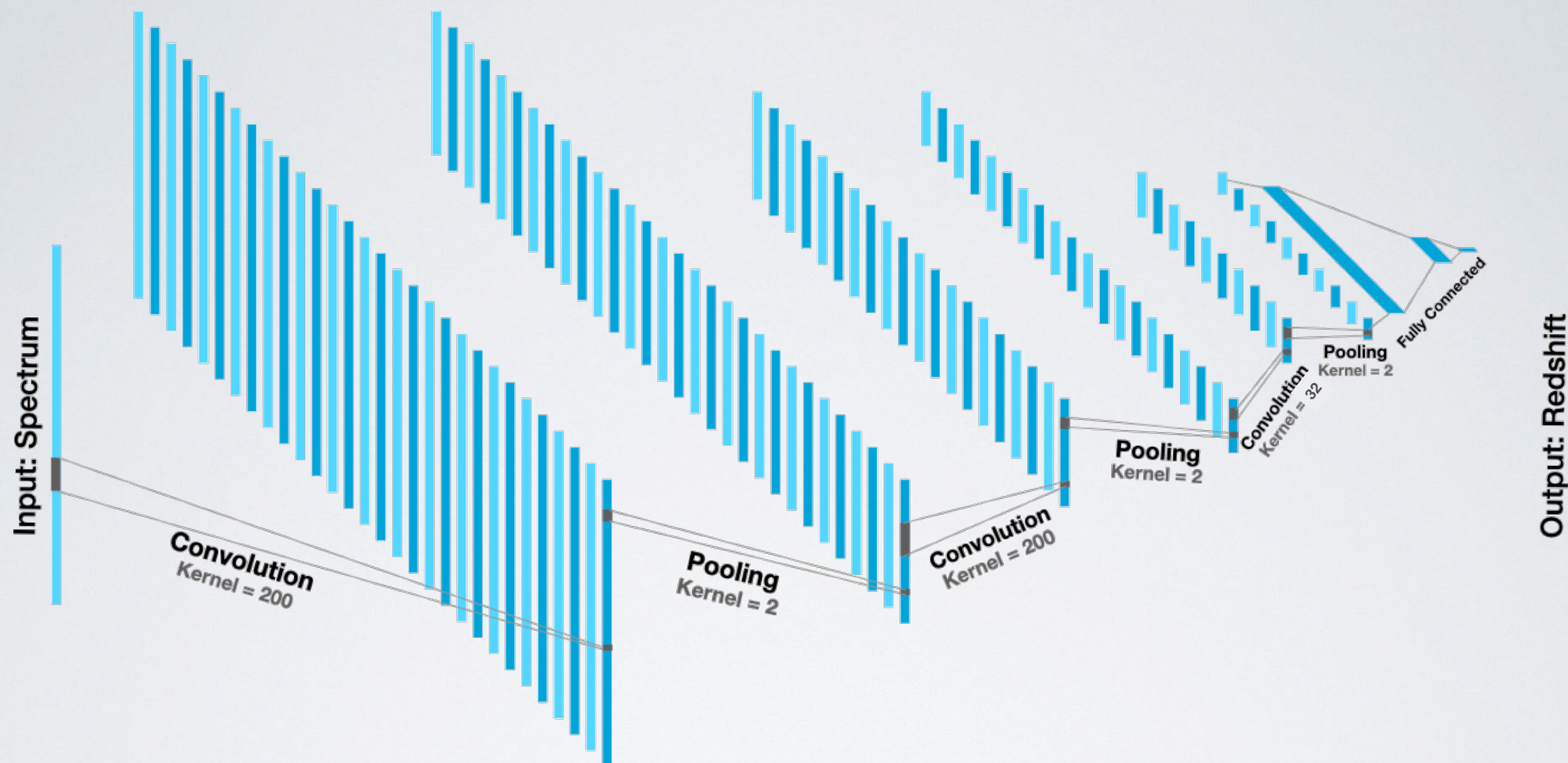
TUTORIAL 3: MORE NETWORKS AND MORE AREAS

From Redshift to Classification: A Simple Change

SDSS Data - One Dimensional



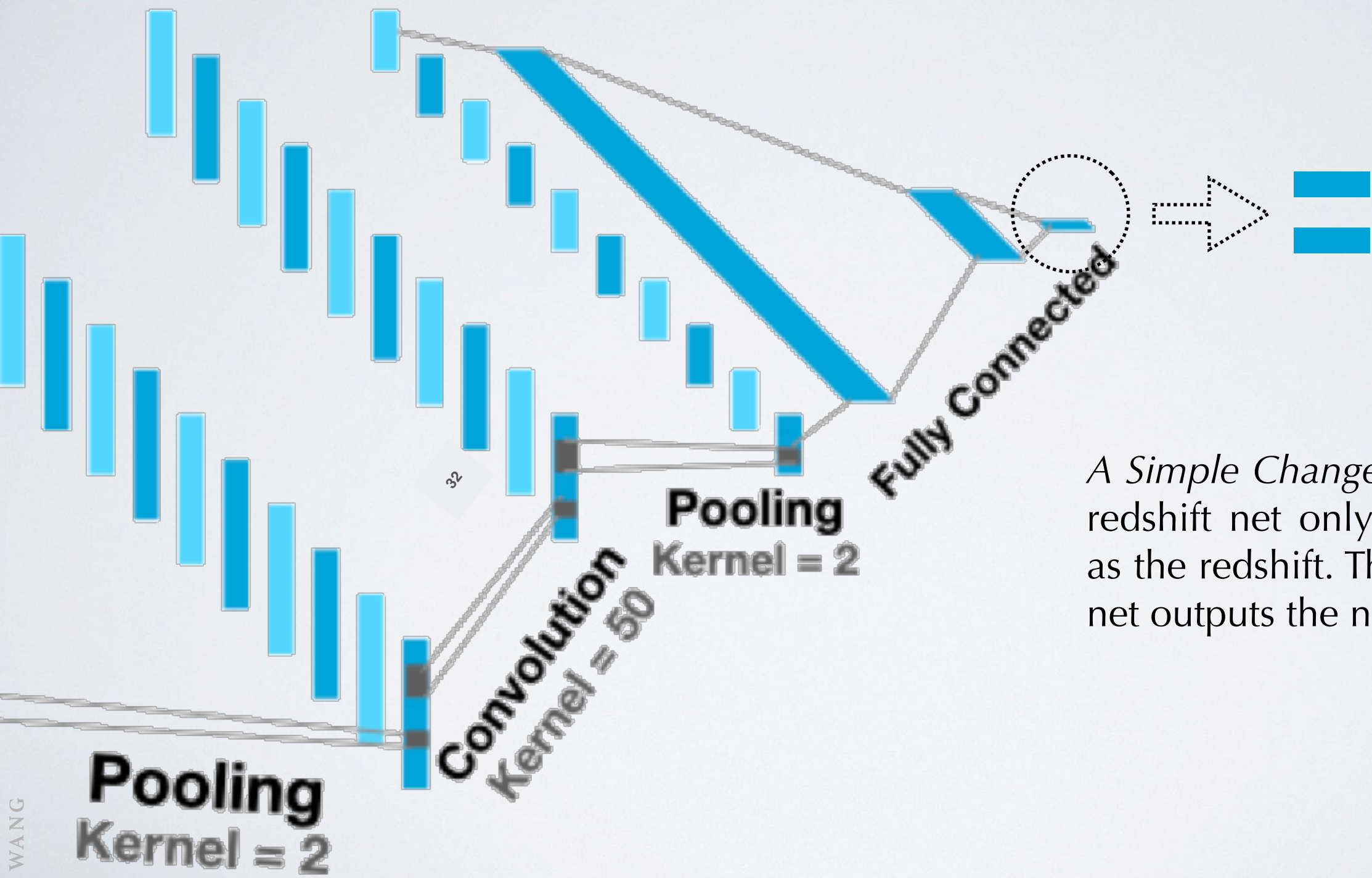
One Dimensional Convolutional Neural Network For Inferring Redshift



Structure of one dimensional CNN. The spectrum of quasar is input as a one-dimensional array, which goes through the convolutional layer of kernel size = 200, 200, 32 respectively in order to search for the global and local pattern. The fully connected layers output the redshift.

FROM REDSHIFT TO CLASSIFICATION: A SIMPLE CHANGE

One Dimensional Convolutional Neural Network For Classification



A Simple Change: The output of redshift net only has one value as the redshift. The classification net outputs the number.

One Dimensional Convolutional Neural Network For Classification

To standardize the output, that all values are between 0 and 1, and the sum of all equals to one. Done by passing to a LogSoftmax function

$$\text{LogSoftmax: } p(x_i) = \log \left(\frac{\exp(x_i)}{\sum_j \exp(x_j)} \right),$$

And we need to change the loss function. The result $q(x_i)$ (predicted classification) will be adopted and together with the labels $p(x_i)$ (real classification) to compute the cross entropy as the loss

$$\text{CrossEntropy: } H(p, q) = \sum_i p(x_i) \log q(x_i)$$

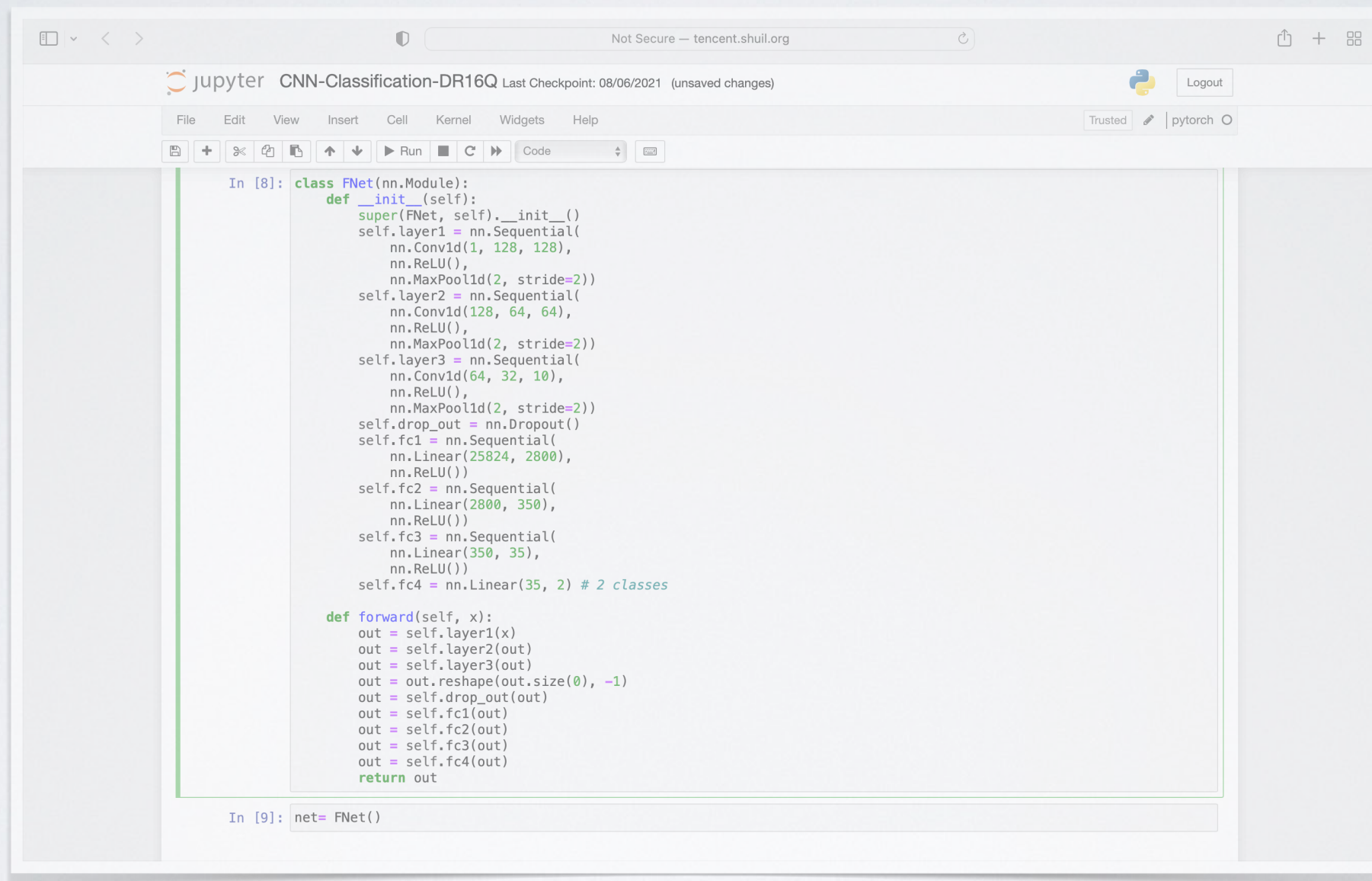
One Dimensional Convolutional Neural Network For Classification



The sum of all outputs equals to 1, the biggest value corresponds to the predicted class.

FROM REDSHIFT TO CLASSIFICATION: A SIMPLE CHANGE

Demonstration in Jupyter notebook



The screenshot shows a Jupyter Notebook interface with a browser window at the top displaying 'Not Secure - tencent.shuil.org'. The notebook title is 'CNN-Classification-DR16Q' with a last checkpoint of '08/06/2021' and '(unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations and execution. The main area contains two code cells. The first cell, labeled 'In [8]:', defines a class 'FNet' that inherits from 'nn.Module'. The class has an '.__init__' method that initializes several layers: 'layer1' (Conv1d, MaxPool1d), 'layer2' (Conv1d, MaxPool1d), 'layer3' (Conv1d, MaxPool1d), 'drop_out' (Dropout), and three fully connected layers ('fc1', 'fc2', 'fc3', 'fc4'). The 'forward' method processes the input 'x' through these layers and returns the output. The second cell, labeled 'In [9]:', instantiates the class with 'net = FNet()'.

```
In [8]: class FNet(nn.Module):
def __init__(self):
super(FNet, self).__init__()
self.layer1 = nn.Sequential(
nn.Conv1d(1, 128, 128),
nn.ReLU(),
nn.MaxPool1d(2, stride=2))
self.layer2 = nn.Sequential(
nn.Conv1d(128, 64, 64),
nn.ReLU(),
nn.MaxPool1d(2, stride=2))
self.layer3 = nn.Sequential(
nn.Conv1d(64, 32, 10),
nn.ReLU(),
nn.MaxPool1d(2, stride=2))
self.drop_out = nn.Dropout()
self.fc1 = nn.Sequential(
nn.Linear(25824, 2800),
nn.ReLU())
self.fc2 = nn.Sequential(
nn.Linear(2800, 350),
nn.ReLU())
self.fc3 = nn.Sequential(
nn.Linear(350, 35),
nn.ReLU())
self.fc4 = nn.Linear(35, 2) # 2 classes

def forward(self, x):
out = self.layer1(x)
out = self.layer2(out)
out = self.layer3(out)
out = out.reshape(out.size(0), -1)
out = self.drop_out(out)
out = self.fc1(out)
out = self.fc2(out)
out = self.fc3(out)
out = self.fc4(out)
return out

In [9]: net = FNet()
```

<https://github.com/YWangScience/Isfahan-workshop-2021/tree/main/code>

TUTORIAL 3: MORE NETWORKS AND MORE AREAS

Deepening the Redshift Net

If we count the latest neural network structures and refer to the winning networks of some recent machine learning competitions, we can see that most of the networks use the classical ResNet as a basis. This is especially true for networks targeting one-dimensional data, such as the gravitational wave prediction competition held at Kaggle just last month, where the top 3 networks all involved ResNet and did not make complex changes. So in this tutorial, we are going to demonstrate the ResNet.

ResNet (Arxiv: 1512.03385)

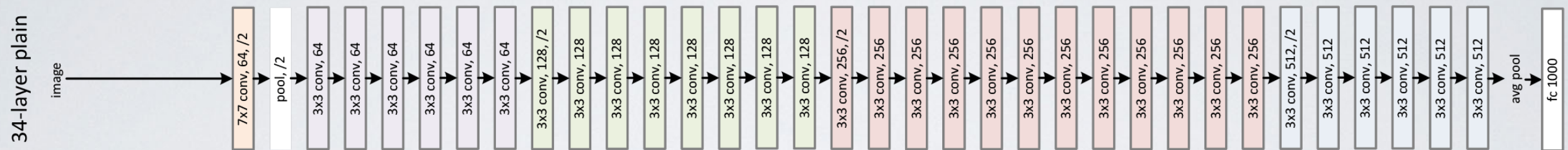
Deep Residual Learning for Image Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

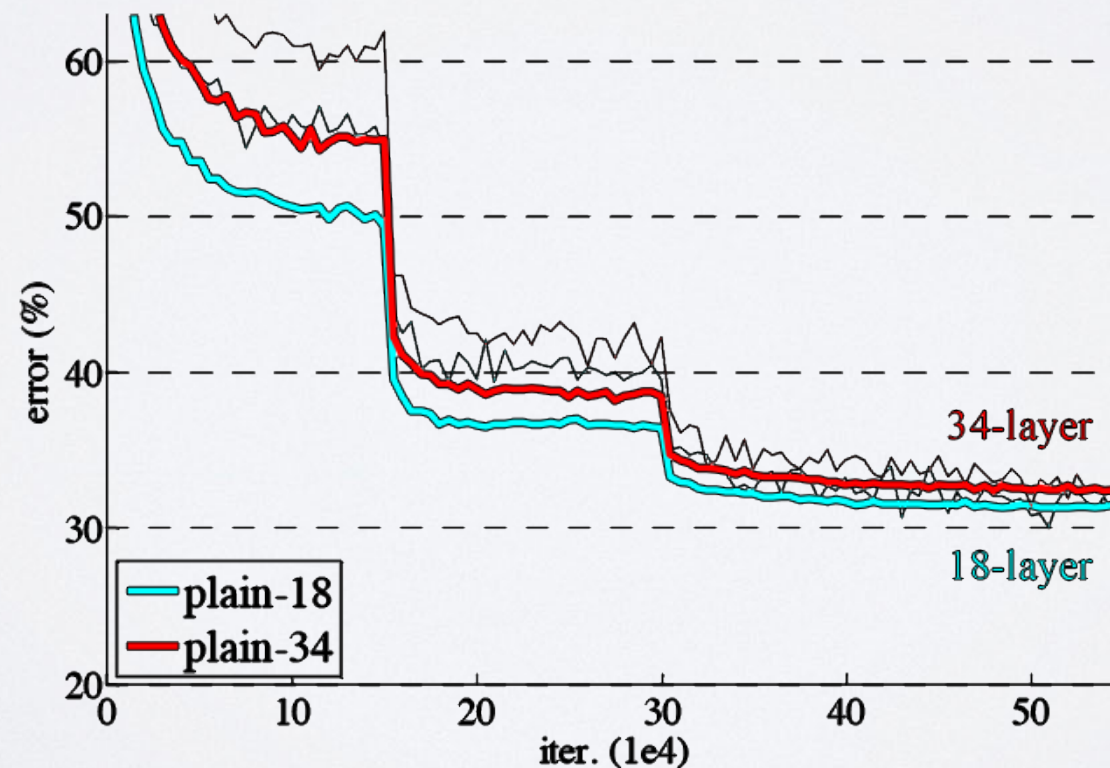
DEEPENING THE REDSHIFT NET

More Layers

The most straightforward idea to get more accurate predictions is to make the network deeper and wider to include more parameters and non-linear structures.



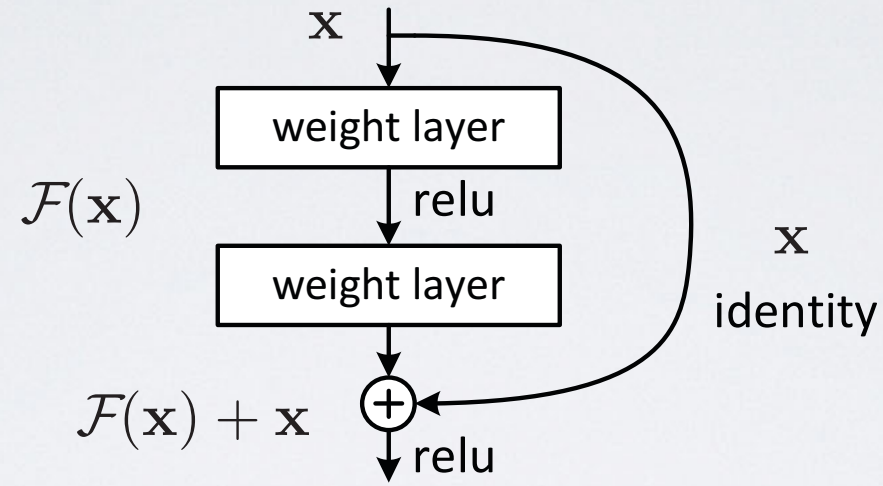
*As the network depth increasing, accuracy gets saturated and then degrades rapidly, called the **degradation** problem.*



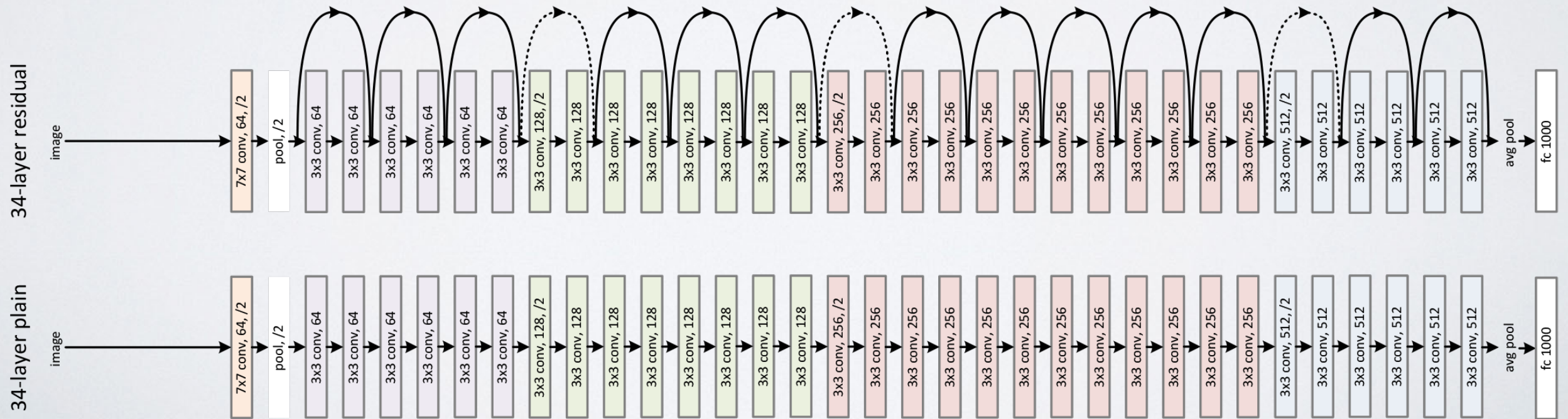
DEEPENING THE REDSHIFT NET

Shortcuts

The shortcut connections simply perform **identity** mapping, and their outputs are added to the outputs of the stacked layers

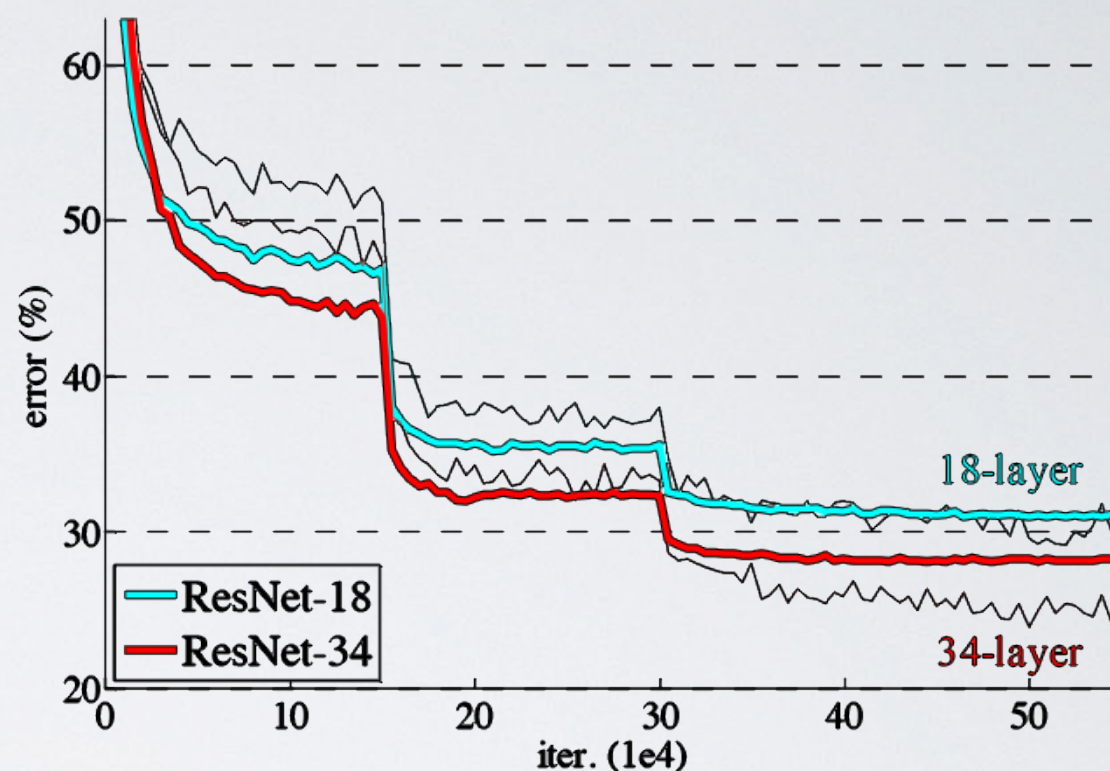
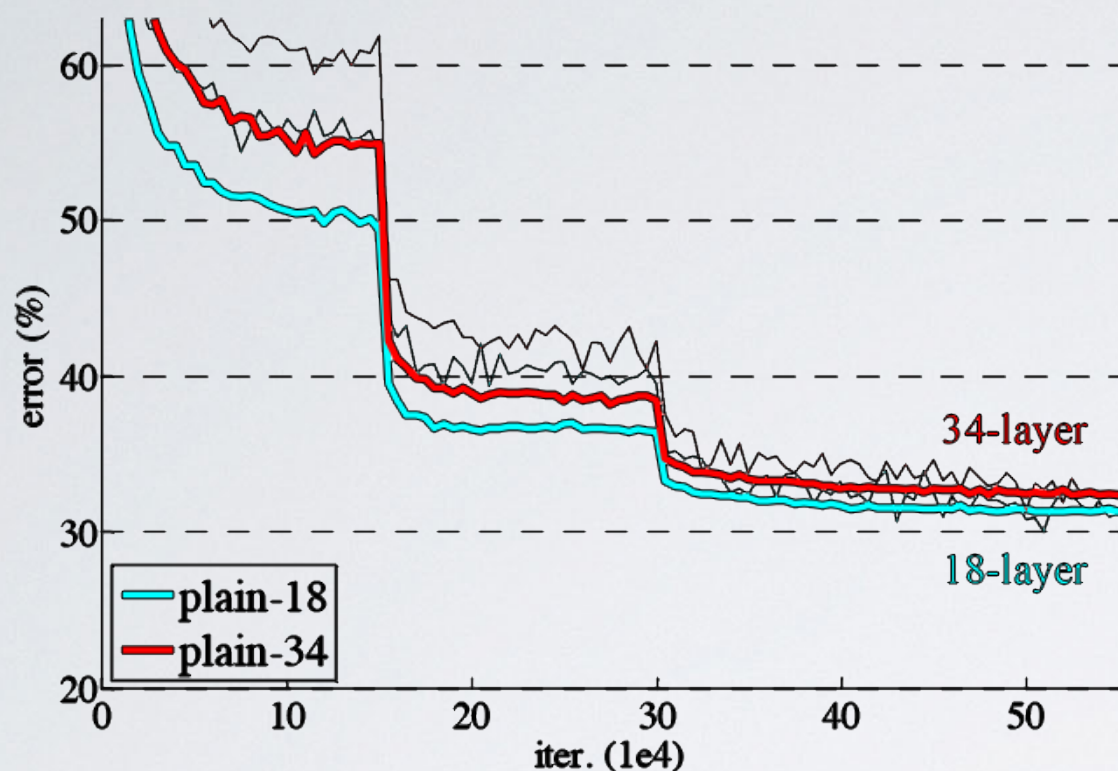


Example of Residual network, shortcuts added.



DEEPENING THE REDSHIFT NET

Solving the degradation problem



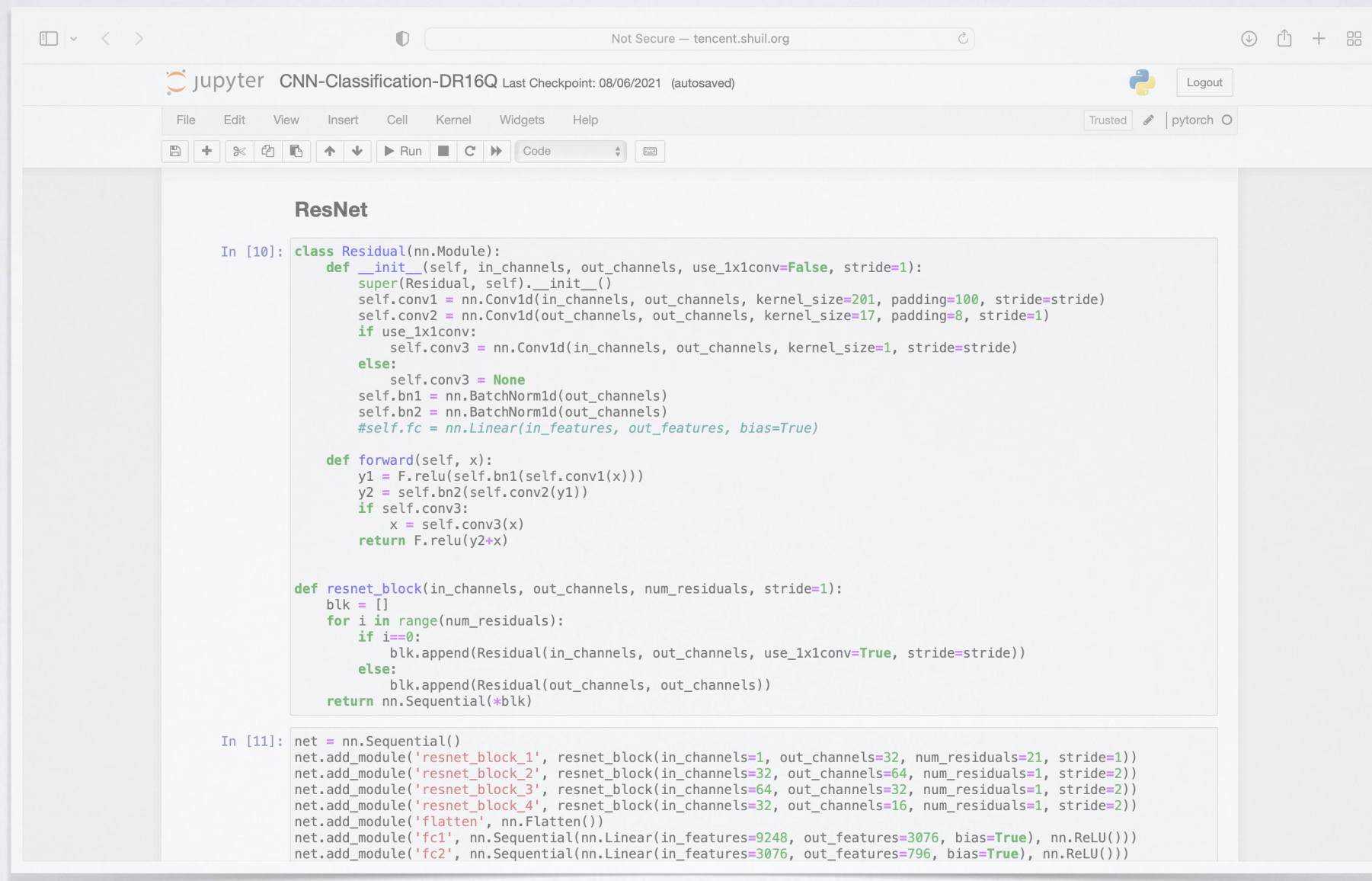
The error of 34 layers ResNet becomes smaller than the 18 layers network.

**Why residual structure works? This is an unanswered question, many papers proposed the explanation, we will discuss if we have time.*

** Also pay attention to the efficient net, which studies model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Arxiv: 1905.11946*

DEEPENING THE REDSHIFT NET

Demonstration in Jupyter notebook



```
ResNet

In [10]: class Residual(nn.Module):
def __init__(self, in_channels, out_channels, use_1x1conv=False, stride=1):
    super(Residual, self).__init__()
    self.conv1 = nn.Conv1d(in_channels, out_channels, kernel_size=201, padding=100, stride=stride)
    self.conv2 = nn.Conv1d(out_channels, out_channels, kernel_size=17, padding=8, stride=1)
    if use_1x1conv:
        self.conv3 = nn.Conv1d(in_channels, out_channels, kernel_size=1, stride=stride)
    else:
        self.conv3 = None
    self.bn1 = nn.BatchNorm1d(out_channels)
    self.bn2 = nn.BatchNorm1d(out_channels)
    #self.fc = nn.Linear(in_features, out_features, bias=True)

def forward(self, x):
    y1 = F.relu(self.bn1(self.conv1(x)))
    y2 = self.bn2(self.conv2(y1))
    if self.conv3:
        x = self.conv3(x)
    return F.relu(y2+x)

def resnet_block(in_channels, out_channels, num_residuals, stride=1):
    blk = []
    for i in range(num_residuals):
        if i==0:
            blk.append(Residual(in_channels, out_channels, use_1x1conv=True, stride=stride))
        else:
            blk.append(Residual(out_channels, out_channels))
    return nn.Sequential(*blk)

In [11]: net = nn.Sequential()
net.add_module('resnet_block_1', resnet_block(in_channels=1, out_channels=32, num_residuals=21, stride=1))
net.add_module('resnet_block_2', resnet_block(in_channels=32, out_channels=64, num_residuals=1, stride=2))
net.add_module('resnet_block_3', resnet_block(in_channels=64, out_channels=32, num_residuals=1, stride=2))
net.add_module('resnet_block_4', resnet_block(in_channels=32, out_channels=16, num_residuals=1, stride=2))
net.add_module('flatten', nn.Flatten())
net.add_module('fc1', nn.Sequential(nn.Linear(in_features=9248, out_features=3076, bias=True), nn.ReLU()))
net.add_module('fc2', nn.Sequential(nn.Linear(in_features=3076, out_features=796, bias=True), nn.ReLU()))
```

<https://github.com/YWangScience/Isfahan-workshop-2021/tree/main/code>

TUTORIAL 3: MORE NETWORKS AND MORE AREAS

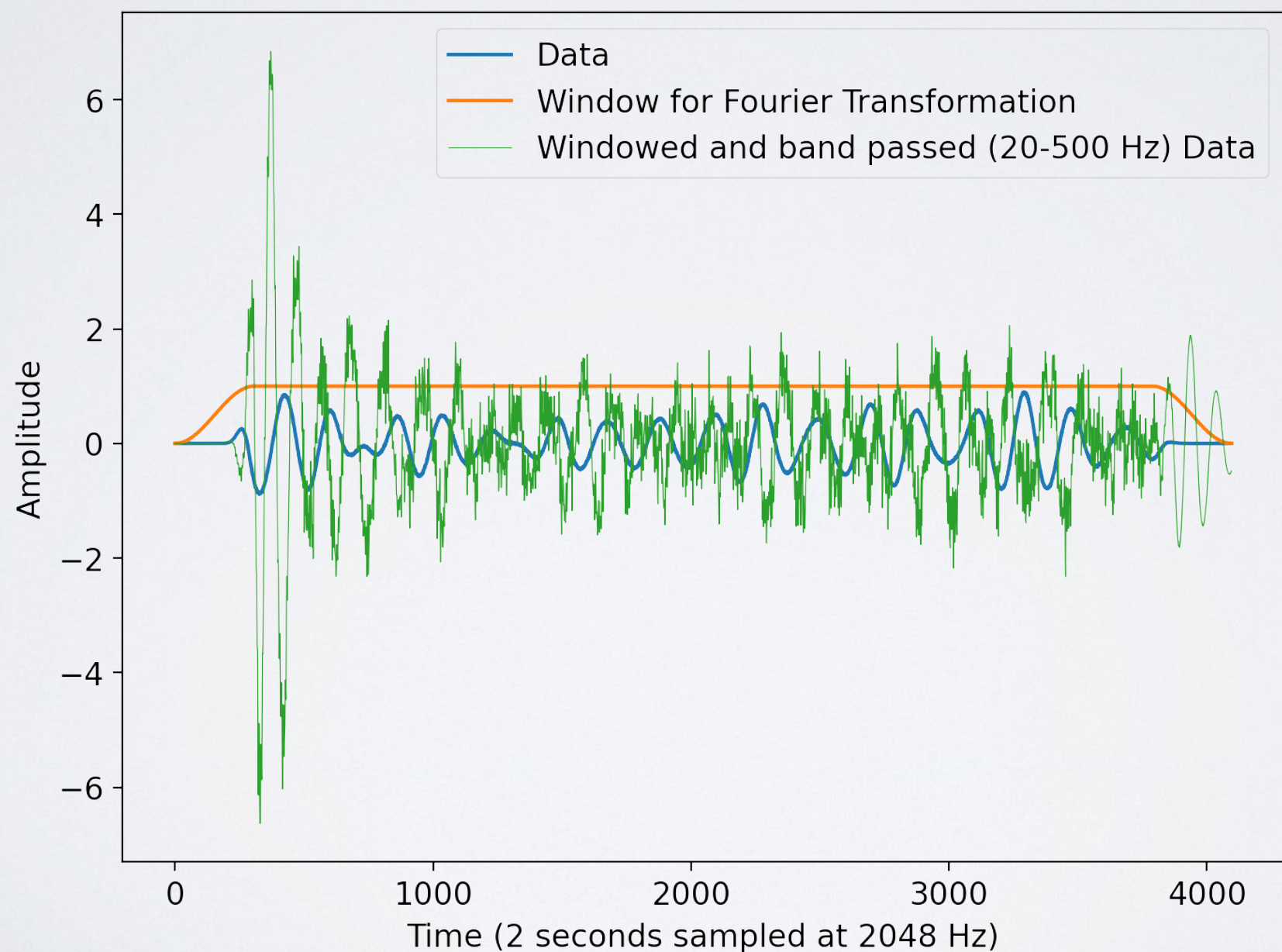
Applying the Redshift Net to Gravitational Wave

APPLYING THE REDSHIFT NET TO GRAVITATIONAL WAVE

Gravitational Wave Data - One Dimensional

Taking the competition of G2Net Gravitational Wave Detection hosted by European Gravitational Observatory as an example.

Details: <https://www.kaggle.com/c/g2net-gravitational-wave-detection/>



APPLYING THE REDSHIFT NET TO GRAVITATIONAL WAVE

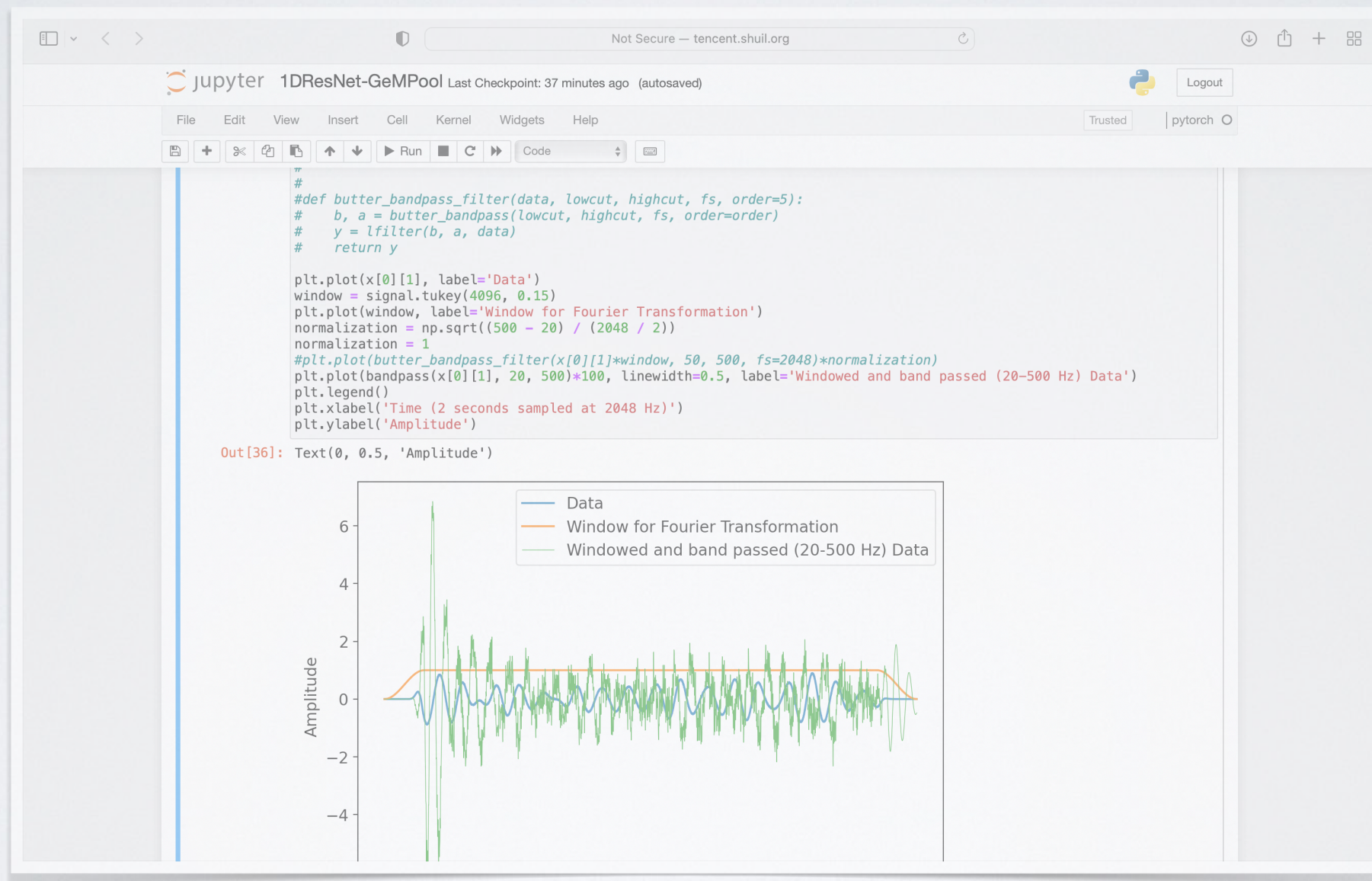
Gravitational Wave Data - One Dimensional

	Input	Output
Redshift	one dimensional data	redshift value
Gravitational Wave	one dimensional data	Is true signal?

We can adopt the same network for redshift detection, simply change the size of input length, and the loss function.

APPLYING THE REDSHIFT NET TO GRAVITATIONAL WAVE

Demonstration in Jupyter notebook



<https://github.com/YWangScience/Isfahan-workshop-2021/tree/main/code>

THANKS YOU

JOIN THE DISCUSSION CHANNEL ON SLACK

<http://shorturl.at/tvyL1>