Introduction to Deep Learning

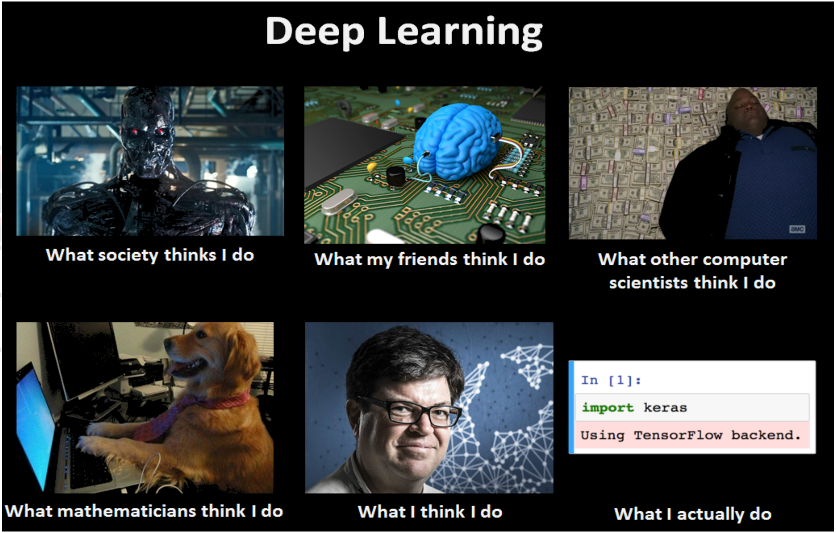
Texas Tech University REU Program

Speaker: Wenhan Gao

Stony Brook University

June 2021

What is Supervised Deep Learning?

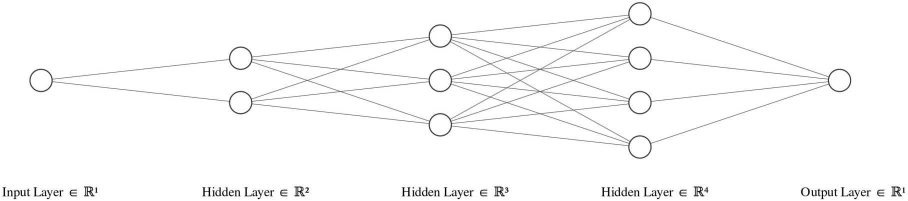


Supervised Deep Learning

- Supervised deep learning is the task of learning a function(mapping) that maps an input to an output based on labeled data.
- For example, we have a dataset containing some cat images and non-cat images, and our task is to learn a mapping from an input image to an output recognizing whether or not the input image is a cat image..
- This mapping can be constructed and trained through a deep neural network(DNN)

Neural Network

- Example of a Deep Neural Network



How does it learn? A Simple Example on Binary Image Classification

- Notation:
- x_i denotes an input training example, y_i denotes its corresponding value.
- In our example, x_i is an image, and y_i labels if it's a cat image or not(0 for non-cat, 1 for cat)
- ϕ denotes the neural network.
- Then $\phi(x_i, w_1, w_2, ..., b_1, b_2, ...)$, which takes an input image x_i, will output a value that predicts y_i, the ground truth label.
- Our Goal: to make the prediction accurate. In other words, to make $\phi(x_i, ...) = \hat{y}_i \approx y_i$
- That is, if our input image is a cat, then we want the output to be close to 1, and close to 0 otherwise.

How can we achieve our goal?

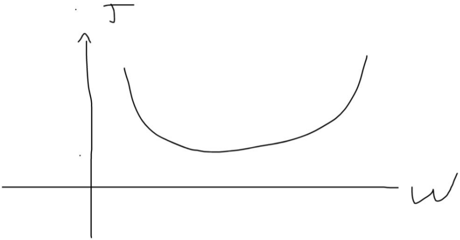
- We will construct a Cost Function that measures how good our neural network is.
- Terminology: Cost Function is with respect to the entire trainning set(the whole image set), and Loss Function is with respect to one training example(a single image).
- An example Loss Function(Binary Cross Entropy Loss): $\mathcal{L}(\hat{y}_i, y_i) = -y_i \log(\hat{y}_i) - ((1 - y_i) \log(1 - \hat{y}_i))$
- This measures how good the neural network is on a single training example, an image, when the prediction value $\hat{y}_i$, is close to the label value y_i, the loss will be small.

Cost Function

- The Cost Function $J = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}_i$, where m is the number of training examples. This measures how good the neural network is on the entire training set.
- Now, our goal is to minimize this cost function. First, what are the parameters in this cost function?
- J $= -\frac{1}{m}\sum_{i=1}^{m}(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))$, the only term that has parameters is $\hat{y}_i$, and parameters are $w_1, w_2, ...b_1, b_2$
- Write J as $J(w_1, w_2, ..., b_1, b_2)$
- Now, the question becomes: How can we reduce, potentially minimize, the cost function, J?

Gradient Descent

- An answer to our question would be gradient descent(GD).
- Illustration of gradient descent with 1D convex function:



- In each iteration, we update $W := W - \alpha \frac{\partial J}{\partial W}$, so that J is moving towards the minima(global minima in this 1D convex case).

Making Prediction

- After a number of iterations, the cost function will approach a minima(local or global), and then just wandering around the minima.
- Now we have the weights and biases that minimize the cost function, we can pass an input, an image, thorough the neural network to make predictions.

NN Architecture: Cat Recognition

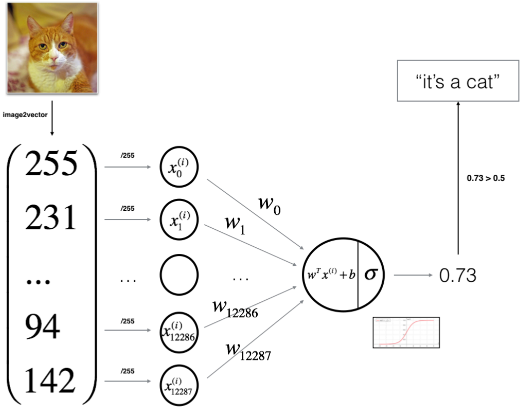


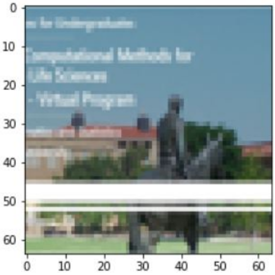
Figure 1: Image adopted from: Andrew Ng, Younes Bensouda Mourri, Kian Katanforoosh.Deep Learning [MOOC]. Coursera.

Cat Recognition: Sample Results

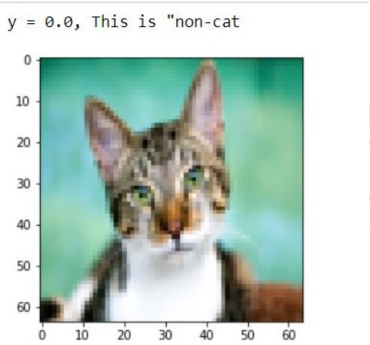
In [54]:
```
my_image = "t1.JPG"
#preprocess the image to fit the matrix size
fname = "images/" + my_image
image = np.array(Image.open(fname).resize((num_px, num_px)))
plt.imshow(image)
image = image / 255.
image = image.reshape((1, num_px * num_px * 3)).T
my_predicted_image = predict(logistic_regression_model["w"], logistic_regr

print("y = " + str(np.squeeze(my_predicted_image)) + ", This is \"" + clas
```

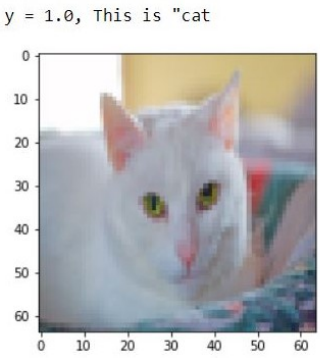
```
y = 0.0, This is "non-cat
```



Cat Recognition: Sample Results



example2



example3

Why is it not making correct prediction

- In example 2, the program predicts "0" when it's a cat image. After testing, it's concluded that this neural networks has 99 percent accuracy on the training set but only 70 percent accuracy on a testing set that comes from the same distribution
- This neural network has high variance problem(overfitting); it's doing much better on the training set than on the testing set.

How to improve?

- More training data
- Include regularization
- Train a Deep Neural Network(DNN) that has more hidden layers, which usually does a better job.
- Apply Conventional Neural Network(CNN) architecture, which does a better job on computer vision. There are also other NN architectures, such, Recurrent Neural Network(RNN) that is widely used in speech recognition and natural language processing.

Residual Model

- We can apply deep learning techniques into approximating solutions for PDEs, and this is actually unsupervised learning because we don't need a labeled dataset, meaning that we can solve PDEs that we don't have solution(s) or access to another solver.

A DNN $\phi(\boldsymbol{x};\boldsymbol{\theta}^*)$ is constructed to approximate the solution $u(\boldsymbol{x})$ via

$$
\begin{aligned}
\boldsymbol{\theta}^* &= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\
&:= \operatorname*{argmin}_{\boldsymbol{\theta}} \|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|_2^2 + \lambda\|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|_2^2 \\
&= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\in\Omega}\left[\left|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\right|^2\right] + \lambda\mathbb{E}_{\boldsymbol{x}\in\partial\Omega}\left[\left|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\right|^2\right] \\
&\approx \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N_1}\sum_{i=1}^{N_1}\left|\mathcal{D}\phi(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)\right|^2 + \lambda\frac{1}{N_2}\sum_{j=1}^{N_2}\left|\mathcal{B}\phi(\boldsymbol{x}_j;\boldsymbol{\theta}) - g(\boldsymbol{x}_j)\right|^2
\end{aligned}
$$

Acknowledgement

- The following content and equations come from the paper published by Dr.Gu, Dr.Yang, and Dr.Zhou.
- *Ref*: Yiqi Gu, Haizhao Yang, Chao Zhou,SelectNet: Self-paced learning for high-dimensional partial differential equations, Journal of Computational Physics.

SelectNet

- The question is, how do we select these collocation points, x_i, in the interior of the domain and on the boundary?
- Are there points that our model will be better than if we trained on other points? If there are, how can we find them?
- To answer this question, a self-paced learning framework, SelectNet, is introduced in their paper to adaptively choose training samples in the above model. It chooses a part of the training samples for actual training over time.
- "The philosophy of self-paced learning is to simulate the learning style of human beings, which tends to learn easier aspects of a learning task first and deal with more complicated samples later."

How SelectNet works

- A Selection Network is introduced and to be trained simultaneously with the residual model.
- This selection network adaptively weighting the training samples to help deciding whether a sample point x_i is of higher importance for the residual model to be trained on.
- How is the level of importance is defined?

Optimization Problem in SelectNet

$$\begin{aligned}(3.11)\quad \min_{\theta}\max_{\theta'_{\mathrm{s}},\theta''_{\mathrm{s}}} \mathbb{E}_{x\in Q}\left[\phi'_{\mathrm{s}}(\boldsymbol{x};\theta'_{\mathrm{s}})|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right] \\ +\lambda\mathbb{E}_{x\in\Gamma}\left[\phi''_{\mathrm{s}}(\boldsymbol{x};\theta''_{\mathrm{s}})|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right] \\ -\varepsilon^{-1}\left[\left(\frac{1}{|Q|}\int_Q\phi'_{\mathrm{s}}(\boldsymbol{x};\theta'_{\mathrm{s}})d\boldsymbol{x}-1\right)^2+\left(\frac{1}{|\Gamma|}\int_\Gamma\phi''_{\mathrm{s}}(\boldsymbol{x};\theta''_{\mathrm{s}})d\boldsymbol{x}-1\right)^2\right],\end{aligned}$$

- where ϕ'_s is the Selection Network(SN) that gives weights to points in the interior of the domain, and ϕ''_s does the same for the boundary.
- First of all, we have an maximization problem, namely: $\max\limits_{\theta'_s,\theta''_s}$.

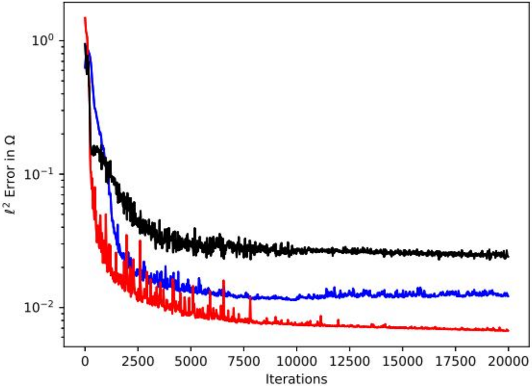
  The intuition is that these two SNs give higher weights to points with larger residual errors.

Optimization Problem in SelectNet

$$
\begin{aligned}
(3.11)\quad \min_{\theta}\max_{\theta_s',\theta_s''} & \mathbb{E}_{x\in Q}\left[\phi_s'(\boldsymbol{x};\theta_s')|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right] \\
& + \lambda\mathbb{E}_{x\in\Gamma}\left[\phi_s''(\boldsymbol{x};\theta_s'')|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right] \\
& - \varepsilon^{-1}\left[\left(\frac{1}{|Q|}\int_Q \phi_s'(\boldsymbol{x};\theta_s')d\boldsymbol{x}-1\right)^2+\left(\frac{1}{|\Gamma|}\int_\Gamma \phi_s''(\boldsymbol{x};\theta_s'')d\boldsymbol{x}-1\right)^2\right],
\end{aligned}
$$

- Also, we randomly initialize SNs with zero bias and random weights with $\mu = 0$ and $\sigma^2 \to 0$. So initially, the SNs are random functions close to a constant. This basically means that SNs have no bias in weighting samples(do nothing) in the early stages of training.
- The last term is a penalty term that penalizes the maximum constraint if ϕ_s' and ϕ_s'' are far apart from satisfying these normalization conditions.

Result



- Red: SelectNet Model; Blue:Residual Model

- Image on page 4:
  https://www.pinterest.com/bjohnston24_gmail_dot_com/machine-learning-meme
- Image on page 13 and 15:
  Andrew Ng, Younes Bensouda Mourri, Kian Katanforoosh. Deep Learning [MOOC]. Coursera.