Deep Learning Study Lecture Note

Lecture 1 : What is Deep Learning?

Kim Tae Young

December 26, 2019

1.1 Introduction

This lecture note is for deep learning study in mathematical problem solving club in KAIST. The purpose of this study is giving mathematical view for deep learning and the ability to implement neural net and conduct experiments with deep learning. I wish everyone attending this study will able to 1. read and summarize deep learning papers, 2. implement model that you want, train, and use it, and 3. at least know what deep learning is and when we can apply it.

For prerequisites, you should know vector calculus and some simple ways of analyzing random variables, and bayes theorem.

1.2 What is AI/ML/DL?

Actually, definition of AI, ML, DL varies from person to person. So this definition is one of my opinion.

First, let's see what wikipedia says.

In computer science, artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans.

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead.

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on artificial neural networks. Learning can be supervised, semi-supervised or unsupervised.

Actually, my definition slightly differs, AI is simply some automated system that makes decision. For example, AlphaGo is one famous AI, but automatic door is also one example of AI, though many would not agree. It would be better if we add that this automated system should be 'nontrivial', making AI intelligent enough. It varies on your choice.

ML is quite different, now we have some rigorous definition. We say some system is ML if it contains parameters that can be changed by automated system. The parameters here are, for example, weights and bias of regression, conditions for decision trees, or weights of neural network.

Finally DL is what contains 'deep neural network'. Though deep is not defined well, I will just say anything larger than 1 is deep.

We have one relation between AI, ML, and DL.

$$DL \subset ML \subset AI$$

Here, all inclusions are strict. For example, SVM or k-NN are simple examples of machine learning that are not deep learning. SAT-solver is one example that is not machine learning, but is AI(even in the wikipedia sense).

Here, we will focus on deep learning, so many concepts in machine learning will be omitted. If you are interested, I recommend book elements of statis-

tical learning. It is quite tough, but it will gives you sure understanding for mathematical background for machine learning.

So deep learning is one part of machine learning, where parameters are weights and bias for neural network. Our problem is to approximate nontrivial function(usually we call it semantic in the sense of that the function is rather real life related than mathematical). For optimization, assuming that the functions are differentiable, we can calculate gradient of loss function w.r.t. each weight and bias, so we can obtain gradient descent. By repeating gradient descent, we get trained neural network which well approximates the function.

In experiment, we have two functions, loss function and accuracy function. Loss function is what we use in gradient descent, so it needs to be differentiable. Instead, it doesn't need to be meaningful to us. It would be better if the loss function is convex, but usually it isn't. Accuracy function doesn't need to be differentiable, but it needs to be meaningful to us. We will see how well our neural network works with accuracy function.

Also, we have two datasets, one called training set, and the other called validation set. We train the Neural Net(meaning we apply gradient descent) with training set, and compute loss and accuracy for validation set, to see whether our neural network works well.

The problem is when neural networks learns the training set too much, therefore learning noise in the training set. This is called overfitting, and treating overfitting is the main concern in training. We can visualize overfitting by looking at difference between train loss and validation loss, or train accuracy and validation accuracy.

There are three narratives understanding deep learning, or neural network. The first narrative is neuroscientific way, viewing artificial neural net as realization of real neural net. The second narrative is representation narrative, which views neural net as transformation of data with the manifold hypothesis. Finally, there is probabilistic narrative, where we view neural net as finding correct value of latent variable.

Theorem 1.2.1. *The manifold hypothesis is that natural data forms lower-dimensional manifolds in its embedding space. There are both theoretical and experimental reasons to believe this to be true. If you believe this, then the task of a classification algorithm is fundamentally to separate a bunch of tangled manifolds.*

Definition 1.2.2. *Given some probability distribution $X \backsim p(X;\theta)$, latent random variable is some ϕ satisfying*

$$p(X;\theta) = \int p(X,\phi;\theta)d\phi$$

1.3 What will we study?

We will study from basic to moderate deep learning. Since we will cover both theoretical basis and practical implementation, you need both backgrounds on

simple linear algebra, simple calculus, and python.

In this first lecture, we will first see what problems we treat in deep learning. Then we will see how deep learning works, and some mathematical background for deep learning. Finally, we will see how python library torch helps us in deep learning.

In the second lecture, we will see what is MLP (Multi Layer Perceptron) and how it is used. By studying MLP, we will see basic structure for neural network. Also, we will see examples of activation function, which makes the nonlinearity of neural network. Then we will implement MLP using torch with dataset MNIST. Finally, we will see some advanced MLP, using Batch Normalization and Dropout, also looking at their mathematical meaning.

In the third lecture, we will focus on optimization, gradient descent. Though stochastic gradient descent works well, we need more 'well optimizing' methods, since we lack time and resource. Momentum ˜Adam are advanced gradient descent methods, using adaptive descent or momentum method. L-FBGS is quite different, it uses quasi Newton method, which is faster but a lot more time consuming. (Here, faster means faster with respect to epoch). Then finally we will see lr scheduler, which is one way of adjusting learning rate during experiment. If time allows, we will see some non gradient descent optimizers.

The fourth lecture will focus on how to experiment. First, we will focus on regularization methods, like cross validation and lasso/ridge regularization. Next, we will see hyperparameter tuning, with grid tuning and bayesian optimization. Also we will treat ensemble method. Then for further research, I will give various way of collecting dataset. Finally we will treat tensorboard, which is tool in tensorflow helping you to monitor the experiment.

The fifth lecture will go deeper to computer vision, where our topic is CNN (Convolutional Neural Network). We will see most basic structure for CNN, and treat some datasets like MNIST, CIFAR10, CIFAR100 with CNN, showing CNN is superior then MLP in problems of images. Then we will study advanced CNN, like GoogleNet and ResNet, where we use reception and residual.

The sixth lecture will treat problems in time series, with our topic of RNN (Recurrent Neural Network). Similar to CNN, we will see basic structure for RNN, treating simple dataset like PTB, TIMIT, then some advanced RNN like LSTM (Long Short Term Memory), GRU (Gated Recurrent Unit), and attention method.

The seventh lecture will see other networks. Since MLP/CNN/RNN are mostly used, other networks are not that popular. So I will just simply describe each neural network. We will treat GCNN (Graph Convolutional Neural Network), MANN (Memory Augmented Neural Network), and Modular Neural Network, Spiking Neural Network. Also, we will practice how to create user defined layer or loss.

The eighth lecture will be similar to seventh one, but now we will focus on other methods. First, we will see unsupervised learning in deep learning, with semi supervised learning also. Then we will see GAN (Generative Adversarial Network) which is mostly used when generating certain types of data. Finally, I will focus on deep reinforced learning. Since all 3 topics that worth one session

each are in one session, this will be light overview.

Finally, individual presentation is our endpoint for this study. I'd like you to choose one (famous) paper related to deep learning, and introduce it. It is okay to choose one we already treated in this study, but I would recommend something new.

1.4 Problems in Deep Learning

We usually classify problems by what data we are given.

	Supervised	Unsupervised
Continuous	Regression	Dimensionality Reduction
Discrete	Classification	Clustering

Figure 1.1: Problems in Deep Learning

Here supervised and unsupervised means whether we have information of the target data. If we have data $(X = (x_1, \ldots, x_n), Y = (y_1, \ldots, y_n))$ and our target is function $f(x) = y$, it is supervised learning. However if we have data $X = (x_1, \ldots, x_n)$ and our target is function $f(x) = y$ such that y well represents the structure of X and has lower dimension, it is unsupervised learning.

Discrete and Continuity means that codomain of function f is discrete/continuous (real or euclidean).

The other kind of problem is called reinforcement learning, where we have reward function for strategy, and our target is to optimize strategy to maximize this reward function. We will discuss reinforcement learning later.

Other classification of problems in deep learning is by domain of the problem. Our problems are usually related to computer vision or natural language processing.

Computer vision means that our problems are related to images. The most simple example is image classification, where we are given image and classifies whether this image is animal, car, etc. The other problem is object localization, which is finding location of object from an image. This is well solved by YOLO (You Only Look Once). There are also other problems like image captioning (writing comment for image) or image generation/style conversion. For this, see StyleGAN.

Natural Language Processing means that we treat natural language like Korean, English. Auto translation is major problem in here. As in problems in computer vision, there are also classification of text, for example filtering hate tweets in twitter.

There are other domains like sound processing (either human voice or music), or video processing. I will not explain details for these, since these are harder to treat than the preceding two examples.

Reinforcement learning is also a major domain for deep learning, where AlphaGo is a famous case. Many games are target of deep reinforcement learning.

Also problems like movement of robot are treated as reinforcement learning. Recently, there are some researches that view mathematical theorem proving as reinforcement learning.

1.5 Feed Forwarding and Back Propagation

Main two algorithm in neural network is feed forward and back propagation. Now suppose our neural network comes as following.

$$x \xrightarrow{W^{(1)}} x^{(1)} \xrightarrow{W^{(2)}} x^{(2)} \xrightarrow{W^{(2)}} \dots \xrightarrow{W^{(n-1)}} x^{(n-1)} \xrightarrow{W^{(n)}} y$$

Figure 1.2: Simple Neural Network

Then forwarding algorithm is simple, we compute

$$\begin{aligned} y &= W^{(n)}(x^{(n-1)}) \\ &= W^{(n)}(W^{(n-1)}(x^{(n-2)})) \\ &\vdots \\ &= W^{(n)}(W^{(n-1)}(\dots W^{(1)}(x))) \end{aligned}$$

And finally, we have following problem, where $\hat{y}$ is given data, $\mathcal{L}$ is loss function. So forward algorithm is computing the loss, given weights $W^{(1)}, \dots, W^{(n)}$.

$$\min_{W^{(1)},\dots,W^{(n)}} \mathcal{L}(\hat{y}, y)$$

Now backward algorithm is where we compute gradients. By chain rule, we obtain

$$\frac{\partial \mathcal{L}}{\partial x^{(k-1)}} = \frac{\partial \mathcal{L}}{\partial x^{(k)}} \frac{\partial x^{(k)}}{\partial x^{(k-1)}}$$

Then with $\dfrac{\partial x^{(k)}}{\partial x^{(k-1)}} = W^{(k-1)}$ we can recursively calculate $\dfrac{\partial \mathcal{L}}{\partial x^{(k)}}$. Then with

$$\frac{\partial \mathcal{L}}{\partial W^{(k)}} = \frac{\partial \mathcal{L}}{\partial x^{(k)}} \frac{\partial x^{(k)}}{\partial W^{(k)}}$$

we obtain gradient of each function.

1.6 Underlying Theory for Deep Learning

Theorem 1.6.1 (Universal Approximation Theorem)**.** *For any continuous function $f(x)$ on a compact subset of $\mathbb{R}^n$ and any $\epsilon \geq 0$, there exists a Neural Network $g(x)$ with a single hidden layer (with reasonable activation function) such that $||f - g||_\infty < \epsilon$. In other words, the neural networks are dense in the space of continuous functions.*

Actually Universal Approximation Theorem does not tell us a lot. It only assures existences, but do not give approximation for number of terms needed. I will not explain details of this theorem, since proof is public in paper 'Approximation by Superpositions of a Sigmoidal Function'. The paper does not only treat continuous case which we stated before, but also indicator function for some set, which is approximated by other continuous function with help of Lusin's theorem.

1.7 How torch works

So our final topic in this session is how torch performs forward algorithm and backward algorithm. Following is simple example of neural network.

```
import torch.nn
class LinearNet(nn.Module):
    def __init__(self):
        super(LinearNet, self).__init__()
        self.linear = nn.Linear(in=1, out=1, bias=True)
        # This is w * x + b

    def forward(self, x):
        return self.linear(x)

net = LinearNet()
optimizer = torch.optim.SGD(net.parameters(), lr=1)
data = ([1, 2, 3, 4], [1.9, 2.1, 2.9, 4.1])
for epoch in range(10):
    x, bar_y = data
    x = x.view(-1, 1)
    y = net(y)
    loss = MLEloss(y, bar_y)
    loss.backward()
    optimizer.step()
```

The basic object of torch is torch.Tensor. As defined in other subjects, tensor is generalization of matrix where matrix is 2 dimensional tensor, vector is 1 dimensional tensor. And what tensor stores is not only its value, but it also contains what 'expression' it was created. And depending on its field requires_grad, it also stores its gradient. If requires_grad is True, it means that this tensor needs gradient in back propagation, meaning this is a parameter and will be optimized. Otherwise, it means this tensor is constant. Weights in neural networks are requires_grad=True, where input and label data are requires_grad=False.

If we see the code now, we can see net(input) line. This line performs forwarding algorithm based on our definition of forward function in class LinearNet. During this computation, variable output stores that it is created from net(input), or more basically, net.linear(input). Then computing MLEloss is

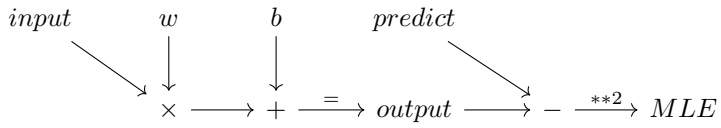

Figure 1.3: Computation path

identical. Loss will contain the info that it is created from MLEloss(output, predict).

Now we call loss.backward(), then reversing its computation path, we store gradient info for each tensor. This is where our info of creation of each tensor and backward algorithm is used.

Finally when we call optimizer.step(), optimizer iterates every elements of tensors that was stored to this optimizer: net.parameters(), and performs gradient descent looking at each gradient value stored in tensor.

1.8 Appendix

1.8.1 Riesz Representation Theorem

You may remember one theorem in linear algebra, that every linear functional $f : F^n \to F$ is in form $f(x) = \langle y, x \rangle$ for some y in F^n, where $< -, - >$ is inner product. Riesz representation theorem is generalization of this statement to general Hilbert Space.

Theorem 1.8.1. *Let H be a Hilbert space with inner product $\langle -, - \rangle$, and H^* be its dual space, consisting of all continuous linear functionals from H into $\mathbb{R}$ or $\mathbb{C}$. For every $\pi \in H^*$, there exists $f \in H$ such that for every $x \in H$, $\pi(x) = \langle f, x \rangle$. Moreover, $||f||_H = ||\pi||_{H^*}$.*

This theorem is used as the main lemma for Universal Approximation Theorem.

1.8.2 Representation Theory

This theory views neural network as embedding from natural data(which is high dimensional euclidean space, like MNIST is $28 \times 28 = 576$ dimension, to low dimensional manifold. More precisely, each layer pulls or pushes the space so that each data can be classified by one line.

For more information, see colah or deepreview.

1.8.3 Automated System

The term automated system I used is really inexplicit, but I don't want to mention deep computability theory to explain this concept. Instead, I'd like to introduce SAT-solver here. SAT problem is most famous NP-complete problem,

and SAT-solver is algorithm that tries to solve this problem as fast it can. There are a lot of approaches for SAT-solver, like walk-SAT which is probabilistic method, and DPLL algorithm, that is look ahead algorithm that works really well in most case. SAT-solver is basis of many automated theorem provers, like for SMT solver. If you are interested in this topic, I recommend CS402 Introduction to Computational Logic.

1.8.4 Mathematical Background

Though in this study we will only consider simple calculus, linear algebra and simple probability theory, the theory of deep learning also contains functional analysis, manifold theory, and probability theory. Though they are not mainstream, but these concepts are well used in sense of proving or finding reason of some methods works well, like batch normalization.

1.8.5 Back Propagation

Here, I will explain back propagation for simple network. Here, assume f is activation function which is differentiable, monotone increasing, and nonlinear.

These are variables that we will use.

- w_{jk}^l : Weight to connect from k-th node in $(l-1)$-th layer to j-th node in l-th layer.
- b_j^l : Bias of the j-th node in the l-th layer.
- y_j^l : Output of j-th node in the l-th layer.
- x_j^l : Input of j-th node in the l-th layer.
- δ_j^l : Error of j-th node in the l-th layer.

The definition of each variables are following.

$$
\begin{aligned}
x_j^l &= \sum_k w_{jk}^l y_k^{l-1} + b_j^l \\
y_j^l &= f(x_j^l) = f(\sum_k w_{jk}^l y_k^{l-1} + b_j^l) \\
\delta_j^l &= \frac{\partial \mathcal{L}}{\partial x_j^l}
\end{aligned}
$$

Since $\mathcal{L}$ is function on y_j^L and given data y (which are constant), I'll assume $\dfrac{\partial \mathcal{L}}{\partial y_j^L}$ is already computed. Then we can compute

$$
\delta_j^L = \frac{\partial \mathcal{L}}{\partial x_j^L} = \sum_k \frac{\partial \mathcal{L}}{\partial y_k^L} \frac{\partial y_k^L}{\partial x_j^L} = \frac{\partial \mathcal{L}}{\partial y_j^L} \frac{\partial y_j^L}{x_j^L} = \frac{\partial \mathcal{L}}{\partial y_j^L} f'(x_j^L)
$$

Then given

$$x_k^{l+1} = \sum_j w_{kj}^{l+1} y_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f(x_j^l) + b_k^{l+1}$$
$$\frac{\partial x_k^{l+1}}{\partial x_k^l} = w_{kj}^{l+1} f'(x_j^l)$$

We can recursively compute

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial x_j^l} = \sum_k \frac{\partial \mathcal{L}}{\partial x_k^{l+1}} \frac{\partial x_k^{l+1}}{\partial x_k^l} = \sum_k \delta_k^{l+1} \frac{\partial x_k^{l+1}}{\partial x_k^l}$$

With this error and following equation

$$x_k^l = \sum_j w_{kj}^l y_j^{l-1} + b_k^l$$
$$\frac{\partial x_k^l}{\partial w_{kj}^l} = y_j^{l-1}$$
$$\frac{\partial x_k^l}{\partial b_k^l} = 1$$

we obtain gradient of weight

$$\frac{\partial \mathcal{L}}{\partial w_{kj}^l} = \frac{\partial \mathcal{L}}{\partial x_k^l} \frac{\partial x_k^l}{\partial w_{kj}^l} = \delta_k^l \frac{\partial x_k^l}{\partial w_{kj}^l} = \delta_k^l y_j^{l-1}$$

and

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \sum_k \frac{\partial \mathcal{L}}{\partial x_k^l} \frac{\partial x_k^l}{\partial b_j^l} = \sum_k \delta_k^l \frac{\partial x_k^l}{\partial b_j^l} = \delta_j^l$$

1.9 Gradient descent

The normal gradient descent computes every loss for each data, then average it, then apply gradient descent. However, this has major problem that this approach will make optimizer stops in saddle point. So we instead use stochastic gradient descent (or mini-batch gradient descent), where we does not average every loss for each data, instead we average mini-batch, which is small subset of whole data set. This has effect of escaping saddle point, also higher speed of computation.