More about Neural Networks

PHYS591000 2022.03.30

Outline

- A few more details about how NN work:
  - Back propagation
- More about Regularization:
  - Dropout
  - Early Stopping

Warming up

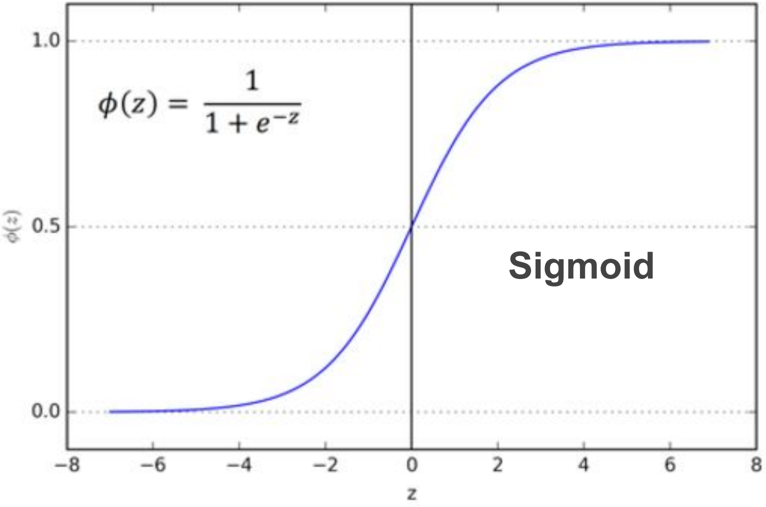
- As usual, take 3 mins to introduce yourself to your teammate for this week!
  – “What do you remember about neural networks from last week?”
  – “We’re halfway through! Any plans for Spring break?”

Review: Activation Function

- Recall that output of each neuron is obtained by feeding an weighted input into an activation function such as **ReLU** or **sigmoid**

$$z = \sum_i w_i x_i + b,$$

$$\text{output} = \sigma(z) = \frac{1}{1 + \exp(-z)}$$



Review: Loss Function

- Recall the goal of training is to find optimal parameters (weights (w_i) and biases (b_j)) which minimize the **loss function** L.

- Typically need to optimize a lot of parameters → Method of Gradient Descent

How to minimize the loss function

- Method of gradient descent: The next step is proportional to the negative of the local gradient

Gradient of Loss (average over the input data) $$\nabla L = \frac{1}{n}\sum_{x}^{n} \nabla L_x$$

$$w_i \rightarrow w_i' = w_i - \eta \frac{\partial L}{\partial w_i}$$

$$b_j \rightarrow b_j' = b_j - \eta \frac{\partial L}{\partial b_j}$$

η: learning rate

a tunable hyperparameter

Courtesy of Prof. Kai-Feng Chen (NTU)

Optimizer for minimizing loss function

- When the input data size is large it will take a lot of time calculating the gradients, and make the NN too slow. There are several **optimizer** to speed up the learning process.

- Most optimizers are based on stochastic gradient descent (SGD), which calculates the gradients using subsets of input data (‘batches’).

Backpropagation

- So it all boils down to calculating the gradient the loss function with respect to many, many parameters.

- In general this means we have to do this numerical calculations many times, and thus make the NN very slow….

- However, the gradients can be calculated in an effective way called **back propagation**.

Backpropagation

- Backpropagation sounds fancy but it's just the chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \sigma(z_l)} \frac{\partial \sigma(z_l)}{\partial w_i} = \frac{\partial L}{\partial \sigma(z_l)} \frac{\partial \sigma(z_l)}{\partial z_l} \frac{\partial z_l}{\partial w_i}$$

$$= \frac{\partial L}{\partial \sigma(z_l)} \sigma'(z_l) \frac{\partial z_l}{\partial w_i} = \frac{\partial L}{\partial z_l} \frac{\partial z_l}{\partial w_i}$$

where z_l is the weighted input in the l-th layer, $\sigma(z_l)$ is the activation function.

Backpropagation

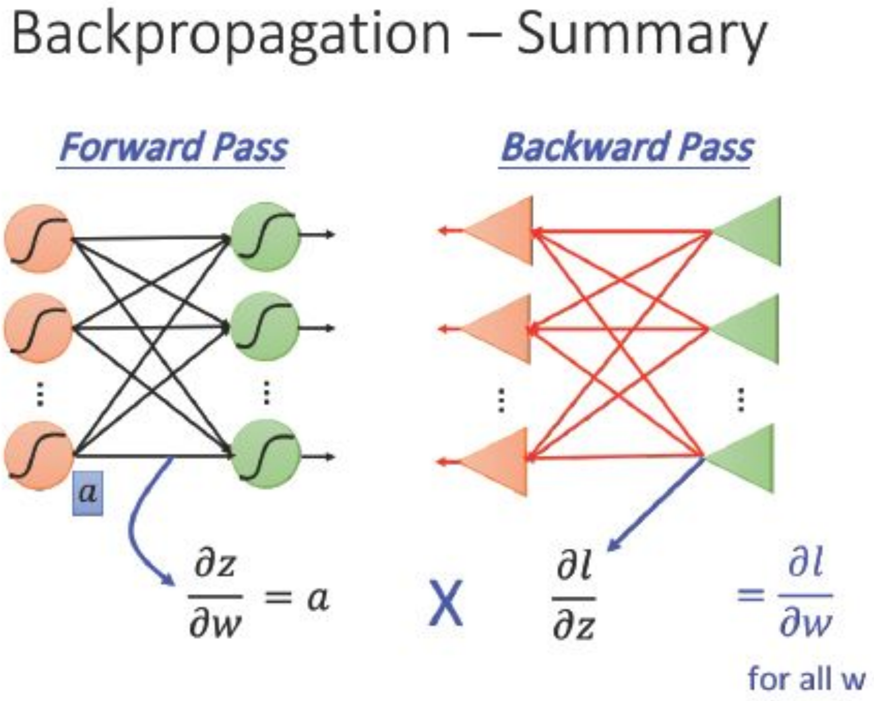
- A few observations:
  - In general it depends on the first derivative of σ(z).
  - $\frac{\partial L}{\partial z_l}$ is *independent* of the weights used for inputs to previous layers (changing weights in this layer won't affect what happened in previous layers). On the other hand, it depends on all weights/gradients applied in next layers.

Backpropagation

- In order to calculate the gradients, we first initialize the weights randomly, and
  - Perform a feedforward calculations to get all $\frac{\partial z_l}{\partial w_i}$
  - Calculate $\frac{\partial L}{\partial z_l}$ from the output (ending) layer, **back propagating** to the second last layer, and to the third last layer,... until we obtained $\frac{\partial L}{\partial z_l}$ for all layers.

Backpropagation

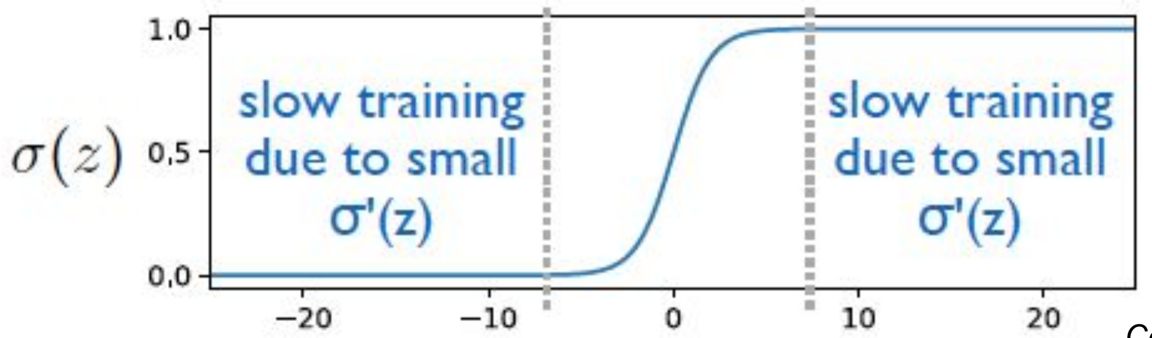
$$\frac{\partial l}{\partial w} = \frac{\partial z}{\partial w}\Big|_{forward\ pass} \cdot \frac{\partial l}{\partial z}\Big|_{backward\ pass}$$



Reference: Slides and lecture video
from Prof. Hung-yi Lee (NTU)

Choice of Loss Function

- This is why we say using mean square error (MSE) as loss function can be slow since one may fall into regions with tiny slope (very small σ'(z)).



Courtesy of Prof. Kai-Feng Chen (NTU)

Choice of Loss Function

- And that's why the **cross entropy function** (combined with Softmax output) is another popular choice for loss function:

$$L = -\sum_i t_j \ln(y_j) \quad j\text{: classes}$$

$$y_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)} \quad k\text{: classes}$$

Courtesy of Prof. Kai-Feng Chen (NTU)

Choice of Loss Function

$$\boxed{\frac{\partial L}{\partial z_i}} = -t_i \frac{1}{y_i}\frac{\partial y_i}{\partial z_i} - \sum_{j \neq i} t_j \frac{1}{y_j}\frac{\partial y_j}{\partial z_i}$$

$$= -t_i(1-y_i) + \sum_{j \neq i} t_j y_i = -t_i + t_i y_i + \sum_{j \neq i} t_j y_i$$

$$= -t_i + y_i \left(t_i + \sum_{j \neq i} t_j \right) = \boxed{y_i - t_i}$$

t_j = target value for class j
by definition $\Sigma t_j = 1$

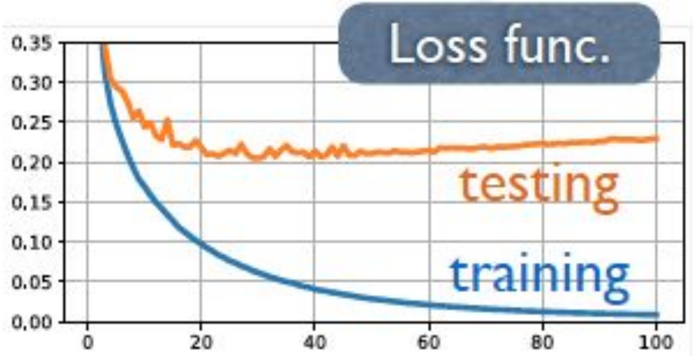
It ends up with the same results as before and no dependency on $\sigma'(z)$!

Courtesy of Prof. Kai-Feng Chen (NTU)

Regularization

- Last week we introduced L1/L2 regularization: Add extra term $\lambda\Sigma i|\mathbf{Wi}|$ (L1) or $\lambda\Sigma i\mathbf{Wi}^2$ (L2) to the loss function

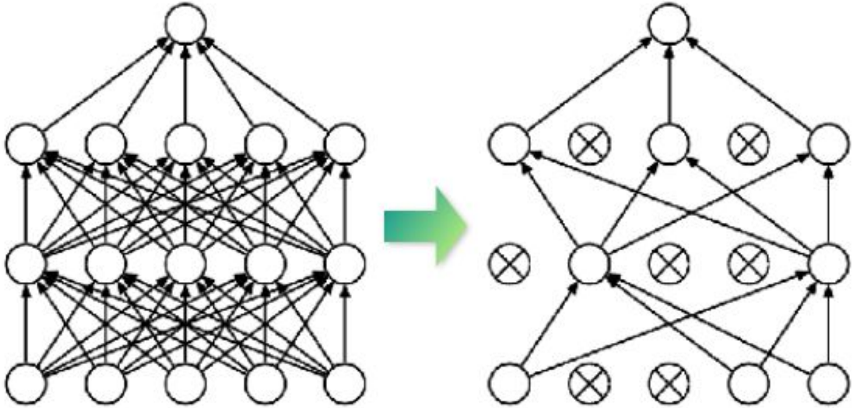
- This week we will work with two other methods: Dropout and Early Stopping.



Number of epochs

Dropout

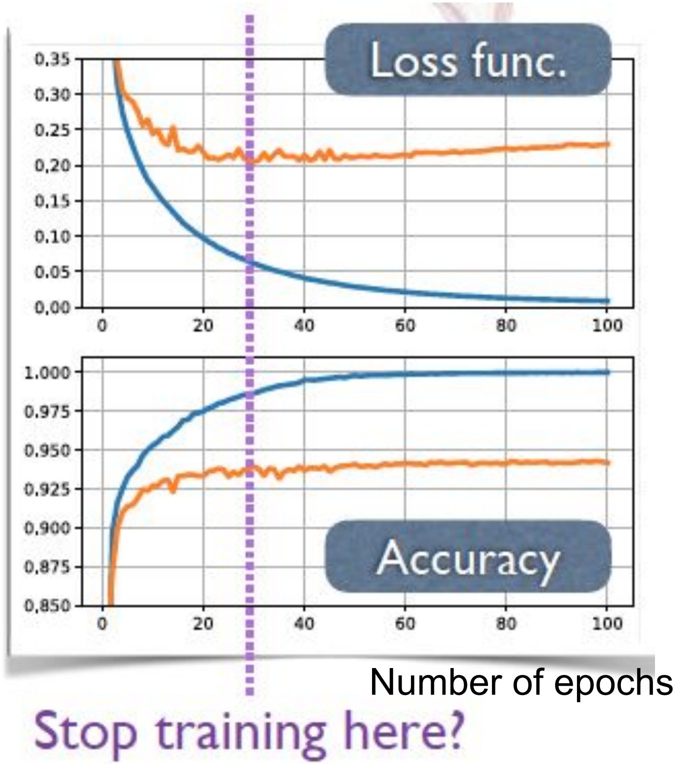
- Dropout method means to randomly disconnect some of the inputs of a specific layer/neurons at each training cycle, and thus less sensitive to noises and be more general.



Courtesy of Prof. Kai-Feng Chen (NTU)

Early Stopping

- Stop training when the model stops improving after certain number of consecutive iterations on a **validation** sample (independent from the train *and* the test data).



Lab for this week

- No in-class exercise this week.

- For the Lab this week, we continue with the same W/Z v.s. QCD jet dataset, and we'll play with

  - Dropout and Early Stopping

  - *KerasTuner* for hyperparameter optimization

Backup