CSE4261: Neural Network and Deep Learning

Lecture: 19.06.2025

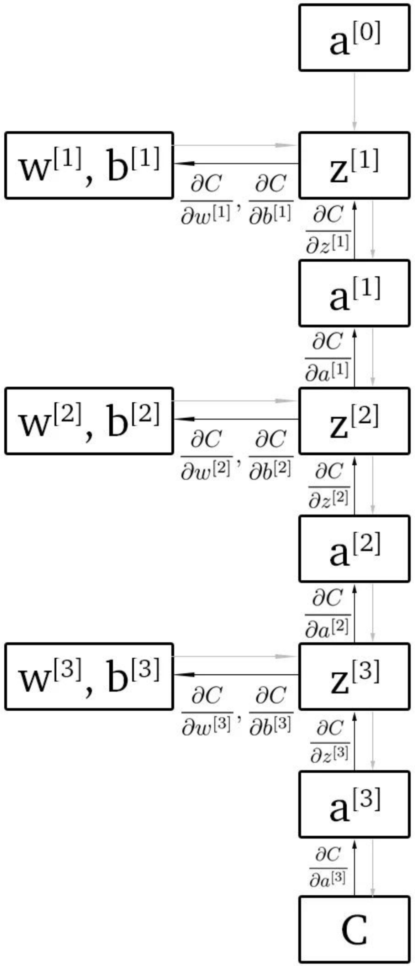
Sangeeta Biswas, Ph.D.
Associate Professor,
University of Rajshahi, Rajshahi-6205, Bangladesh

Backpropagation Computation Graph

- for calculating the partial derivatives of C with respect to $w[l]$, $b[l]$, we need to calculate:

$$\frac{\partial C}{\partial z^{[L]}}, \frac{\partial C}{\partial z^{[l]}}, \frac{\partial z^{[l]}}{\partial w^{[l]}}, \frac{\partial z^{[l]}}{\partial b^{[l]}}$$

- error information is propagated backward through the network layers for updating the weights



Jacobean Matrix

y = f(x)

Types of Differentiation:

- Derivative:
  - independent variable, **x**, is 1D
  - dependent variable, **y**, is a scalar value
  - df/dx
- Partial Derivative
  - x is multi-dimensional
  - *δf/δx*

Results of Differentiation:

- Gradient
  - A vector of partial derivatives of scalar valued **y**
- Jacobean Matrix
  - A matrix of partial derivatives of vector valued **y**

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

Gradient

$$\boldsymbol{J} f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & & \frac{\partial f_2}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Jacobian Matrix

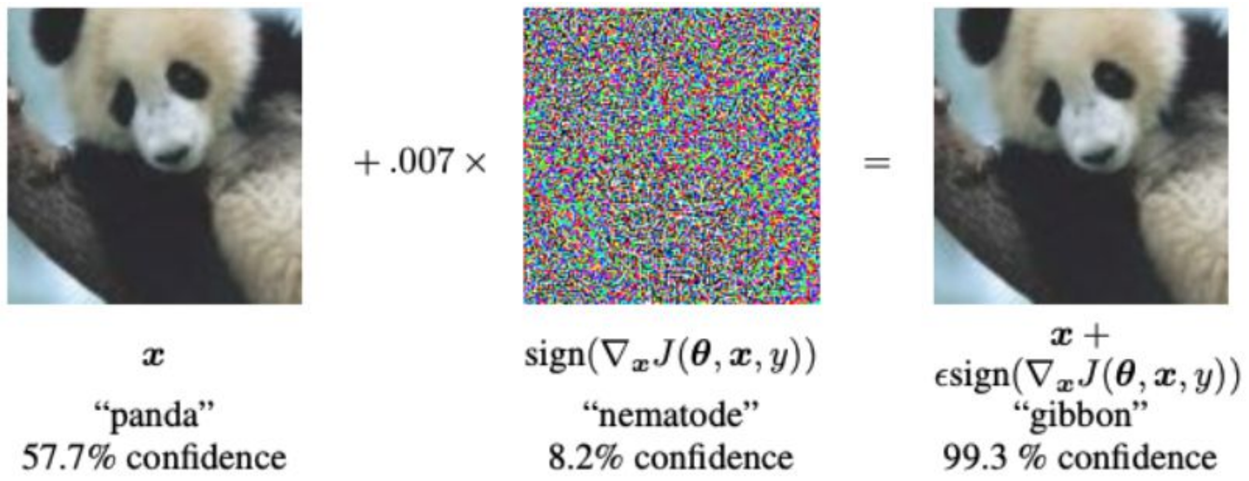
tf.GradientTape()

- It records operations for automatic differentiation
- https://www.tensorflow.org/api_docs/python/tf/GradientTape
- Example:

```
x = tf.constant(3.0)
with tf.GradientTape() as g:
    g.watch(x)
    y = x * x
dy_dx = g.gradient(y, x)
```

Adversarial Attack: Fast Gradient Signed Method (FGSM)

- It uses the gradients of the loss with respect to the input image to create a new image that maximises the loss.
- The new image is called the adversarial image. $adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$



https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

Usage of tf.GradientTape()

```
def create_adversarial_pattern(input_image, input_label):
    with tf.GradientTape() as tape:
        tape.watch(input_image)
        prediction = pretrained_model(input_image)
        loss = loss_object(input_label, prediction)
    # Get the gradients of the loss w.r.t to the input image.
    gradient = tape.gradient(loss, input_image)
    # Get the sign of the gradients to create the perturbation
    signed_grad = tf.sign(gradient)
    return signed_grad
```