Lecture 01:
Introduction, Overview and DNN Basics

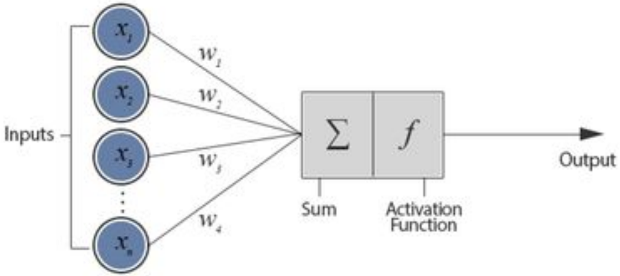
ECE-GY 9483/CSCI-GA 3033
Special Topics in Electrical Engineering EFFICIENT AI AND HARDWARE ACCELERATOR DESIGN

Basics of Deep Neural Networks

- Multi-layer Perceptrons (MLPs)
    - Fully-connected layers
    - Activation functions
    - Loss function
    - Backpropagation
- How forward and backward propagation is performed?
- How to compute the gradient?
- How to update the weight?
- How to initialize the weight before training?

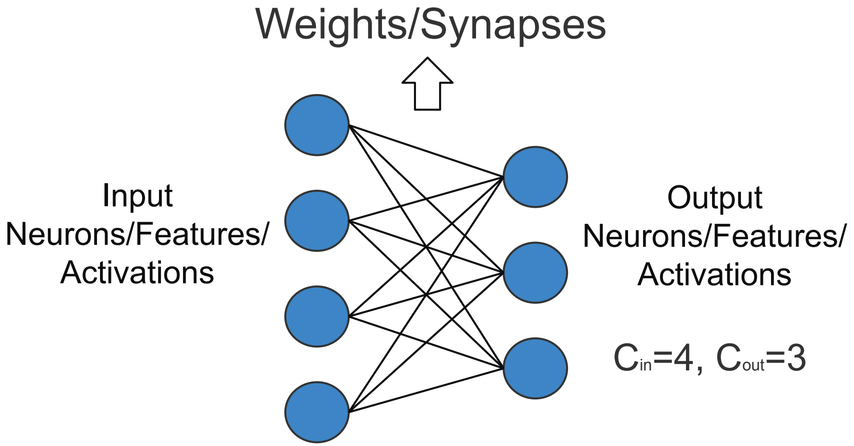
Multi-layer Perceptrons

- Usually consists of fully-connected layers with nonlinear activation functions.



- A neural network consists of interconnected nodes, called neurons, organized into layers.
- Each neuron receives input signals (activations), performs a computation on them, and produces an output signal that may be passed to other neurons in the network.

Fully-connected layers (Linear layers)



Weights/Synapses

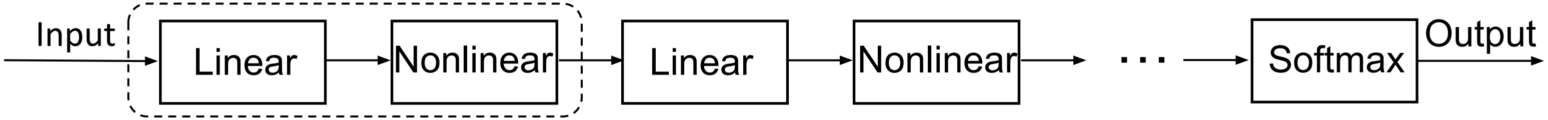
Input Neurons/Features/ Activations

Output Neurons/Features/ Activations

$C_{in}=4$, $C_{out}=3$

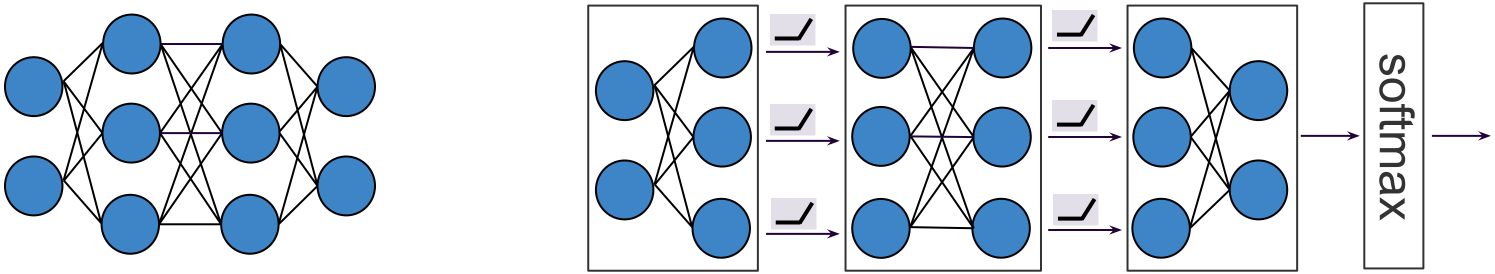
$$Y = XW + b$$

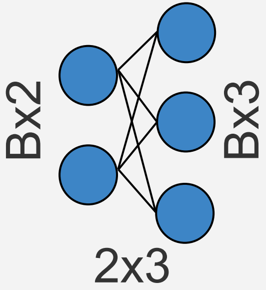
- X (input activations): $B \times C_{in}$
- Y (output activations): $B \times C_{out}$
- W (weights): $C_{in} \times C_{out}$
- b (bias): $1 \times C_{out}$
- C_{in}: Number of input activations
- C_{out}: Number of output activations
- B: batch size



Input → Linear → Nonlinear → Linear → Nonlinear → · · · → Softmax → Output

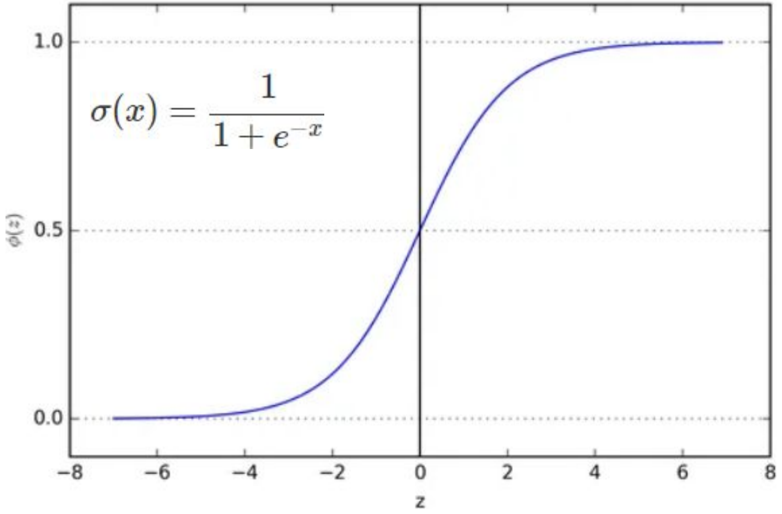
Computational Cost for MLP





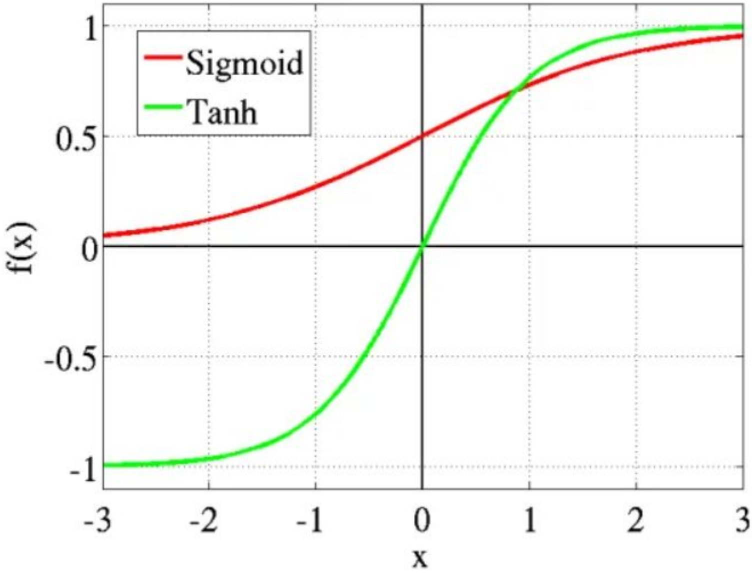
- Number of MACs:
  - Bx2x3 = 6B
- Storage cost:
  - 6 x 32 = 192 bits (Weights)
  - (2B + 3B) x 32 bits (Activation)

Sigmoid



- Function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Domain: $(-\infty, \infty)$
- Range: [0,1]
- Differentiable everywhere
- Derivative: δ(x)(1-δ(x))

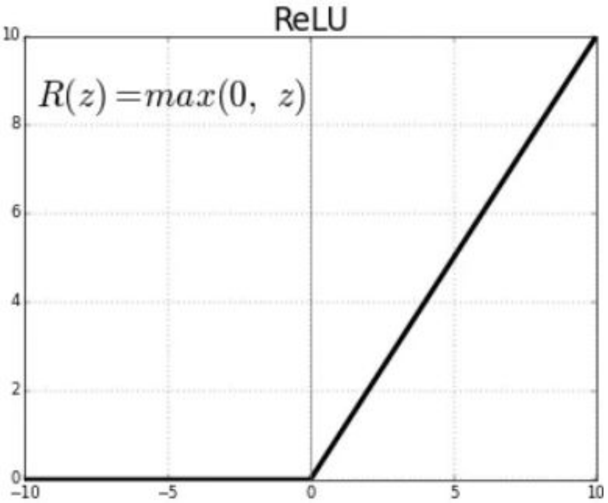
Tanh



- Function: $tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$
- Domain: $(-\infty, \infty)$
- Range: [-1,1]
- Differentiable everywhere
- Derivative: $1 - tanh^2(x)$

NYU SAI LAB

ReLU

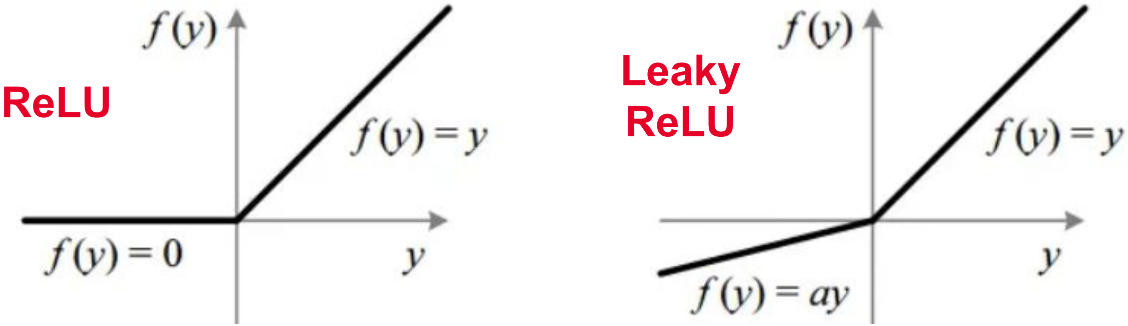


$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

- Domain: $(-\infty, \infty)$
- Range: $[0, \infty]$
- Differentiable everywhere $\begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases}$

NYU SAI LAB

Leaky ReLU



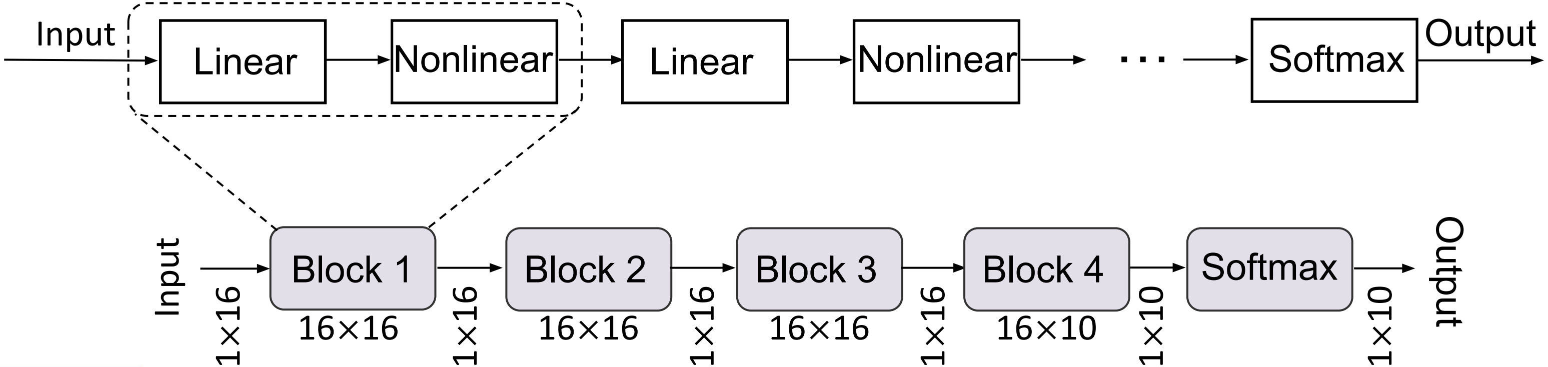
$$Leaky\_ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$$

- Domain: $(-\infty, \infty)$
- Range: $(-\infty, \infty)$

Softmax

$$s_i = \frac{e^{z_i}}{\Sigma_{j=0}^{N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

- Domain: $[-\infty\ \infty]^N$
- Range: $[0,1]^N$
- It is a multivariate function



Loss Functions

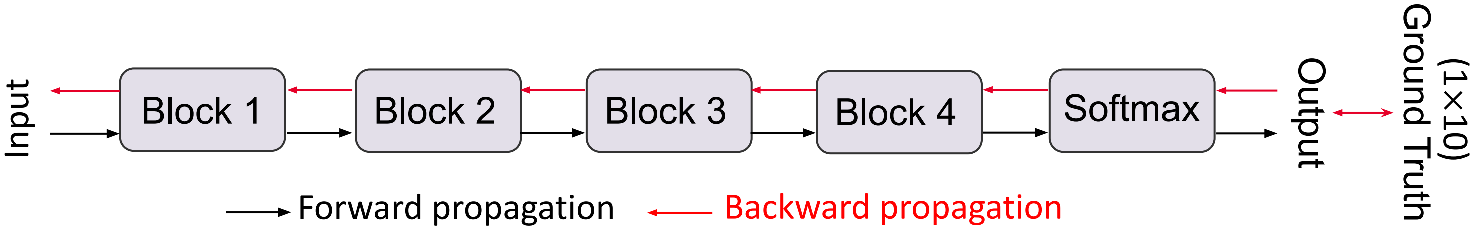
- Loss functions quantify the difference between the DNN output and the ground truth output in the training dataset.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

L2 loss

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

Cross-entropy loss

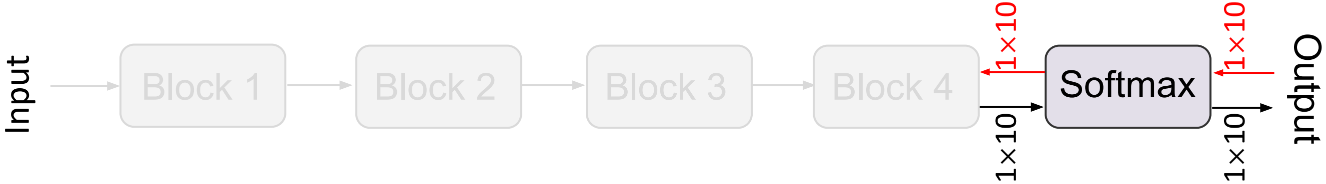


NYU SAI LAB

Softmax

$$s_i = \frac{e^{z_i}}{\Sigma_{j=0}^{N-1} e^{z_j}} \quad For\ i = 1, 2, \cdots, N$$

- Domain: $(-\infty, \infty)$
- Range: [0,1]



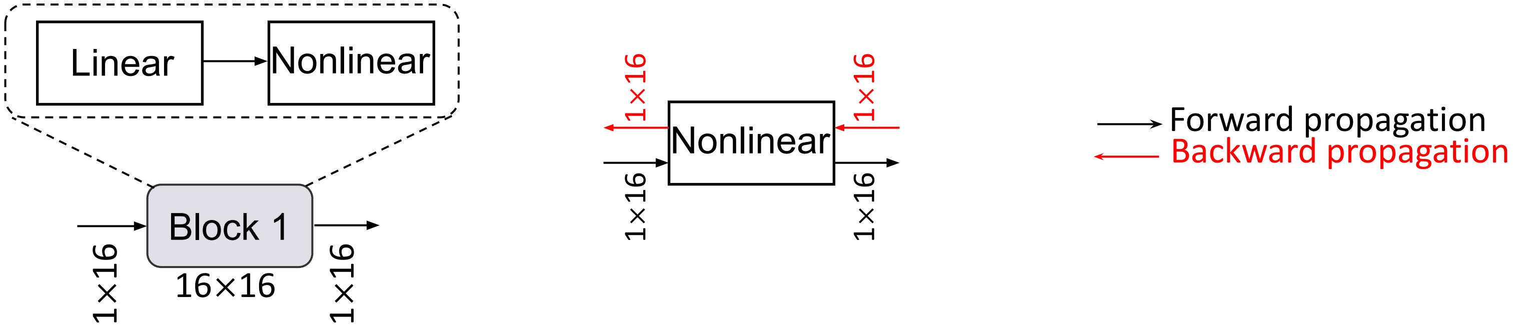
$$\frac{\mathrm{ds}}{\mathrm{dz}} = diag(s) - ss^T$$

When s has a dimension of 3

$$\frac{\mathrm{ds}}{\mathrm{dz}} = \begin{bmatrix} s_1 - s_1^2 & -s_1 \cdot s_2 & -s_1 \cdot s_3 \\ -s_2 \cdot s_1 & s_2 - s_2^2 & -s_2 \cdot s_3 \\ -s_3 \cdot s_1 & -s_3 \cdot s_2 & s_3 - s_3^2 \end{bmatrix}$$

NYU SAI LAB

Backpropagation for Nonlinear Layers



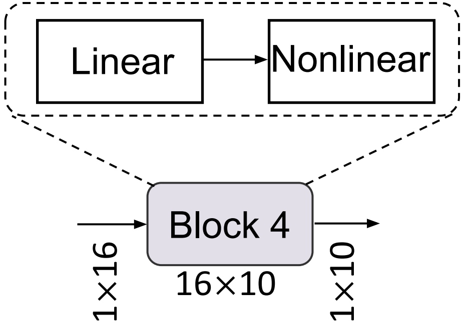
- Due to the elementwise nature, usually the nonlinear layer does not change the input and output shape during both forward and backward passes.

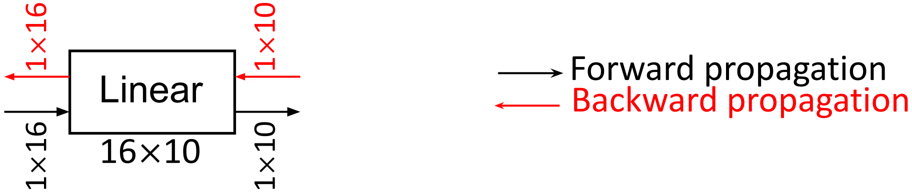
Backpropagation for Nonlinear Layers

- Tanh: $tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$ $1 - tanh^2(x)$
- ReLU: $ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$ $\frac{dReLU(x)}{dx} = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$
- Leaky_ReLU: $Leaky\_ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases}$ $\frac{dLeaky\_ReLU(x)}{dx} = \begin{cases} 1, & \text{if } x \geq 0 \\ a, & \text{otherwise} \end{cases}$
- Sigmoid: $\sigma(x) = \frac{1}{1 + e^{-x}}$ $\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$

NYU SAI LAB

Backpropagation for Nonlinear Layers



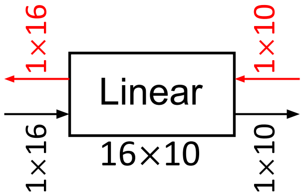


- Due to the elementwise nature, usually the nonlinear layer does not change the input and output shape during both forward and backward passes.

Fully-connected layers (Linear layers)

Y = XW + b

- X (input activations): B×C$_{in}$
- Y (output activations): B×C$_{out}$
- W (weights): C$_{in}$×C$_{out}$
- b (bias): 1×C$_{out}$



$$\overset{1\times16}{\frac{dL}{dX}} = \frac{dL}{dY}\frac{dY^{\top}}{dX} = \overset{1\times10}{\frac{dL}{dY}}\overset{10\times16}{W^{\top}}$$

Derivative wrt data

$$\frac{dL}{db} = \frac{dL}{dY}$$

Derivative wrt bias

$$\underset{16\times10}{\frac{dL}{dW}} = \underset{16\times1}{X^{\top}}\underset{1\times10}{\frac{dL}{dY}}$$

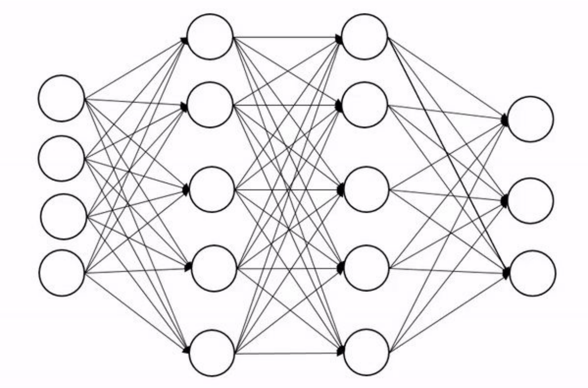
Derivative wrt weight

Weight Decay and Dropout

- The loss function is usually attached with a weight decay loss to penalize the complexity of the function and prevent the overfitting.

$$L = L + \lambda||W||^2$$

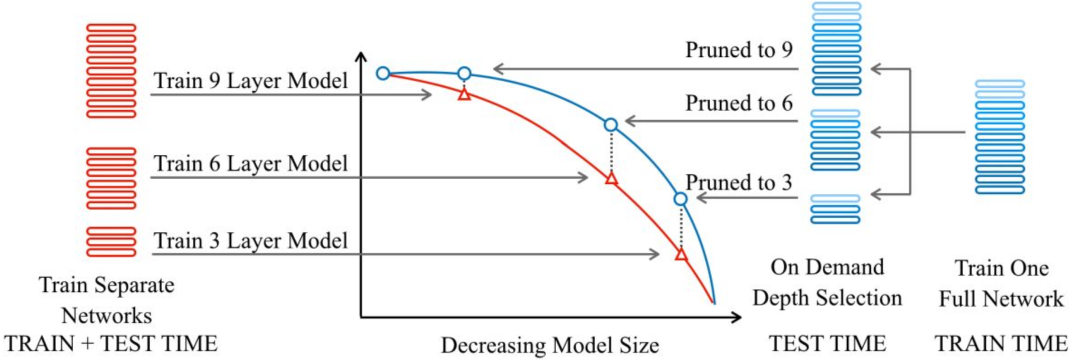
- Dropout refers to the practice of disregarding certain nodes in a layer at random during training.
- All the nodes will be there during inference.
- Can be used to prevent overfitting and reduce the dependency on any one of a single node.



Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15.1 (2014): 1929-1958.

Layer Dropout

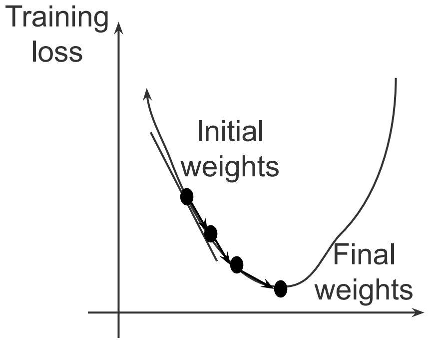
- LayerDrop, a form of structured dropout, which has a regularization effect during training and allows for efficient skipping at inference time.
- It is possible to select sub-networks of any depth from one large network without having to finetune them and with limited impact on performance.
- Usually used in transformer.



Fan, Angela, Edouard Grave, and Armand Joulin. "Reducing transformer depth on demand with structured dropout." *arXiv preprint arXiv:1909.11556* (2019).

DNN Training Process

- An optimizer is a crucial element that adjusts DNN parameters during training. Its primary role is to minimize the training loss defined by the loss function.
    - Epoch: The number of times the algorithm runs on the whole training dataset.
    - Batch: The size of block of dataset that is used to update the model weights.
    - Iteration: total_training_data_size/Batch
    - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.



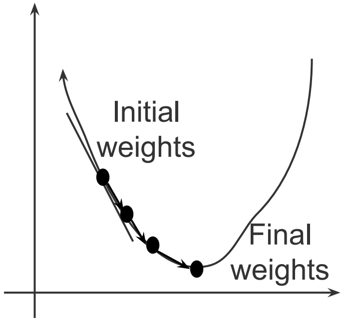
```
Initialized W for each layer.
For each epoch:
        Shuffle the training data.
        For each batch in training datasize:
                Perform the forward propagation
                Compute the loss and weight gradient via backward propagation.
                Update the weights
        Update the learning rate (if necessary)
```

Batch, Iteration and Epoch

- A data batch refers to a subset of the entire training dataset used to train the network.
- Iteration refers to a single update of the model's parameters.
- An epoch represents one complete pass through the entire training dataset. Here's what typically happens during an epoch:
    - For example, if you have 1,000 training examples and you use a batch size of 100, it would take 10 iterations to complete one epoch.
- The composition of minibatches typically changes after every epoch during the training of a DNN.

DNN Training Process

- An optimizer is a crucial element that fine-tunes DNN parameters during training. Its primary role is to minimize the model’s error or loss function, enhancing performance.
    - Epoch: The number of times the algorithm runs on the whole training dataset.
    - Batch: The size of block of dataset that is used to update the model weights dataset.
    - Iteration: total_trainingdata_size/Batch
    - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.

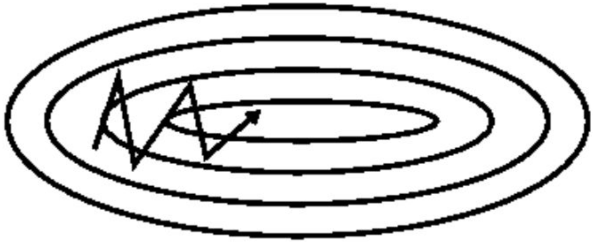


```
Initialized W for each layer.
For each epoch:
        Shuffle the training data.
        For each batch in training datasize:
                Perform the forward propagation
                Compute the loss and weight gradient via backward propagation.
                Update the weights
        Update the learning rate (if necessary)
```

Stochastic Gradient Descent (with Momentum)

- $W' = W - \eta\, dL/dW$
- Due to the significant noise introduced during the SGD process, it is beneficial to stabilize the process.
- $W' = W - \eta g_t$ $\quad g_t \rightarrow s g_{t-1} + (1-s) dL/dW$, s is a hyperparameter between 0 and 1, close to 1.

SGD without momentum



SGD with momentum

RMSProp

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

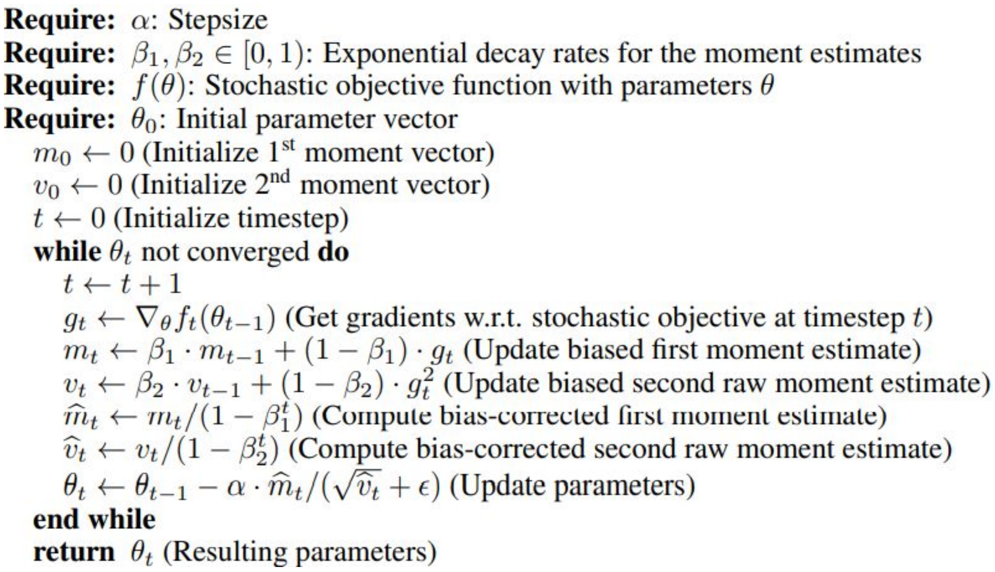
g = [0.02, -0.04, 1.6, -0.01]

1.6 will be scaled down with RMSProp

- All operations are elementwise operations.
- When the variance of gradients is high, we scale down the gradient as we want to be more conservative and vice versa.

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Adam Optimizer



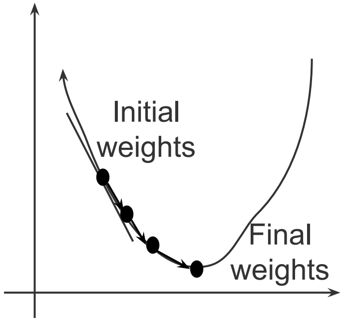
```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

- Combine RMSProp with Momentum SGD.
- By adapting the learning rate during training, Adam converges much more quickly than SGD.

Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

DNN Training Process

- An optimizer is a crucial element that fine-tunes DNN parameters during training. Its primary role is to minimize the model's error or loss function, enhancing performance.
  - Epoch: The number of times the algorithm runs on the whole training dataset.
  - Batch: The size of block of dataset that is used to update the model weights dataset.
  - Iteration: total_trainingdata_size/Batch
  - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.



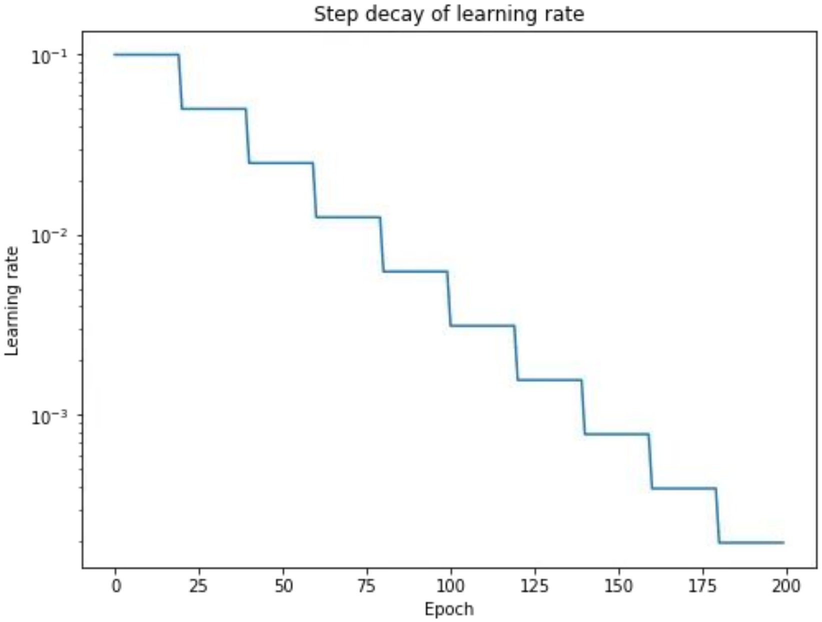
```
Initialized W for each layer.
For each epoch:
    Shuffle the training data.
    For each batch in training datasize:
        Perform the forward propagation
        Compute the loss and weight gradient via backward propagation.
        Update the weights
    Update the learning rate (if necessary)
```

Learning Rate Scheduler

- Learning rate η is an important hyperparameter for training the DNNs.
- A large learning rate can help the algorithm to converge quickly. But it can also cause the algorithm to bounce around the minimum without reaching it or even jumping over it if it is too large.
- If the learning rate is too small, the optimizer may take too long to converge or get stuck in a plateau if it is too small.

$$W' = W - \eta\, g_t$$

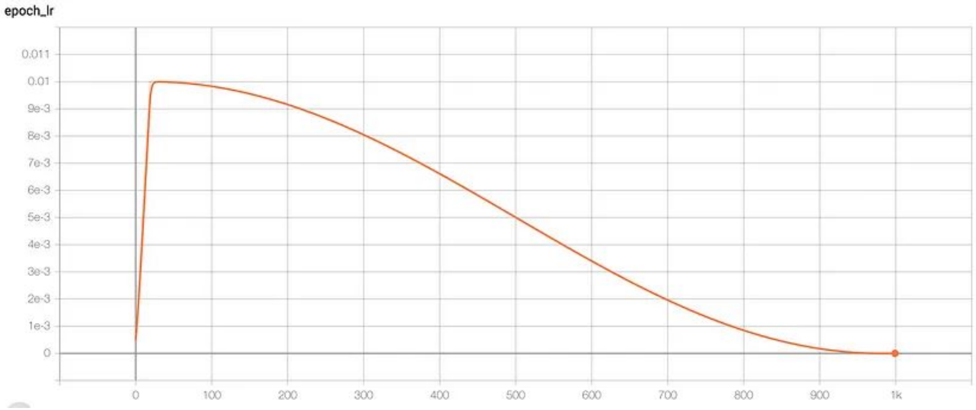
Multistage Learning Rate



- The learning rate is reduced by a fixed amount after every T epochs.
- Typically, the learning rate is reduced to 10% of its value after every T epochs.
- Widely used in image classification task.

NYU SAI LAB

Cosine Learning Rate

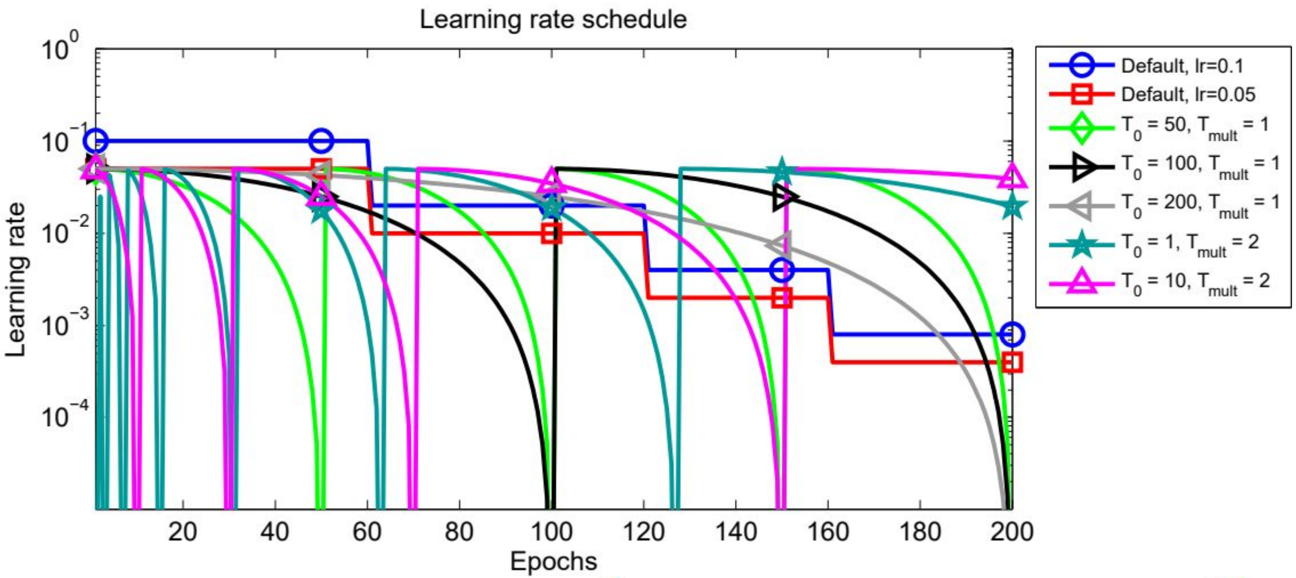


- We propose to periodically simulate warm restarts of SGD, where in each restart the learning rate is initialized to some value and is scheduled to decrease.
- Periodic restart can effectively avoid local minima and saddle points during the training.

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

Loshchilov, Ilya, and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." *arXiv preprint arXiv:1608.03983* (2016).

Cosine Learning Rate

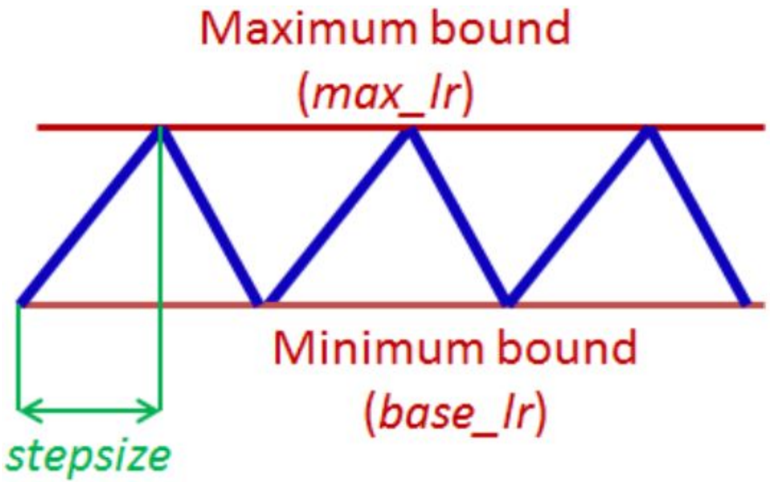


- Tcur accounts for how many iterations have been performed since the last restart.
- Tcur is updated at each iteration t.
- The SGD is restarted once Ti epochs are performed, where i is the index of the run.
- Ti may increase with i.

$$\eta_t = \eta^i_{min} + \frac{1}{2}(\eta^i_{max} - \eta^i_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right),$$

Loshchilov, Ilya, and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." *arXiv preprint arXiv:1608.03983* (2016).

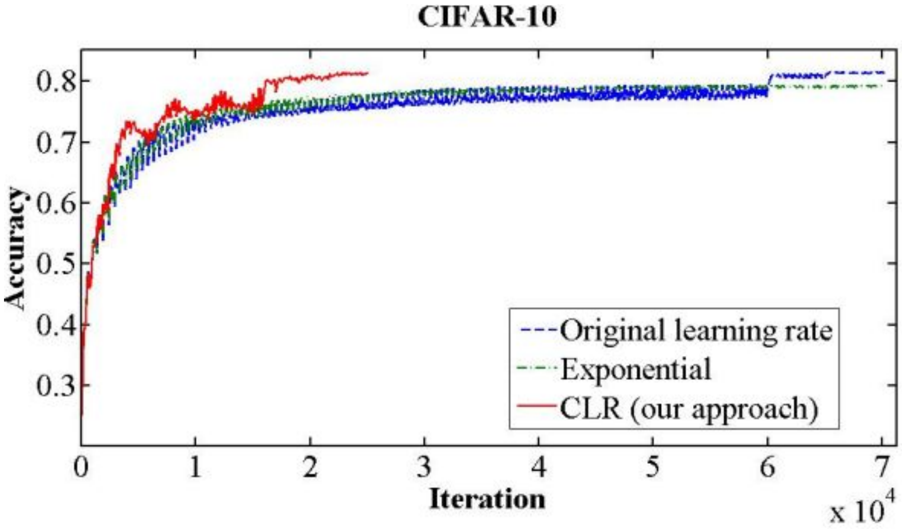
Cyclical Learning Rate



- Increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect.

Smith, Leslie N. "Cyclical learning rates for training neural networks." *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017.

Cyclical Learning Rate

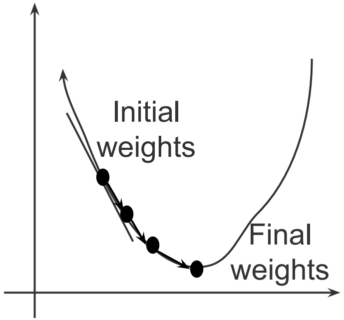


- The red curve shows the result of training with cyclical learning rate achieves the shortest convergence time.

Smith, Leslie N. "Cyclical learning rates for training neural networks." *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2017.

DNN Training Process

- An optimizer is a crucial element that fine-tunes DNN parameters during training. Its primary role is to minimize the model’s error or loss function, enhancing performance.
    - Epoch: The number of times the algorithm runs on the whole training dataset.
    - Batch: The size of block of dataset that is used to update the model weights dataset.
    - Iteration: total_trainingdata_size/Batch
    - Learning rate: It is a parameter that provides the model a scale of how much model weights should be updated.

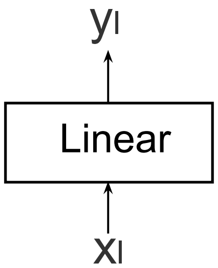


```
Initialized W for each layer.
For each epoch:
    Shuffle the training data.
    For each batch in training datasize:
        Perform the forward propagation
        Compute the loss and weight gradient via backward propagation.
        Update the weights
    Update the learning rate (if necessary)
```

DNN Initialization: Kaiming Initialization

- Kaiming initialization is designed for modern DNN that uses ReLU.

$$W \sim \mathcal{N}\left(0, \frac{2}{n^l}\right)$$



- Target: ensure the activation variance is the same across different layers.
- Assumption:
  - ReLU activation.
  - Weight is normally distributed with mean of zero.
  - Weight and activations are independent.

NYU SAI LAB

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." *Proceedings of the IEEE international conference on computer vision*. 2015.

Derivation

$$\mathbf{y}_l = \mathrm{W}_l \mathbf{x}_l + \mathbf{b}_l$$

Assume W_l has a shape of m_l by n_l, and x_l has a size of $n_l \times 1$, then y_l has a size of $m_l \times 1$.

For each element $y_{l,i}$ of y_l, its variance $var(y_{l,i}) = var(\sum_{j=1}^{n_l} W_{l,i,j} x_{l,j}) = n_l var(W_{l,i,j} x_{l,j})$

Assume each pair of $W_{l,i,j}$ and x_j are independent random variable, then we have:

$$var(W_{l,i,j} x_{l,j}) = E(W_{l,i,j}^2 x_{l,j}^2) - E^2(W_{l,i,j} x_{l,j}) = E(W_{l,i,j}^2) E(x_{l,j}^2) - E^2(W_{l,i,j}) E^2(x_{l,j})$$

Assume $W_{l,i,j}$ follows a normal distribution with mean of 0, that is $E(W_{l,i,j}) = 0$, then:

$$var(W_{l,i,j} x_{l,j}) = var(W_{l,i,j}) E(x_{l,j}^2)$$

$$var(y_{l,i}) = n_l var(W_{l,i,j} x_{l,j}) = n_l var(W_{l,i,j}) E(x_{l,j}^2)$$

NYU SAI LAB

Derivation

Let see how $E(x_{l,j}^2)$ is related to the variance of $y_{l-1,j}$, where $x_{l,j}=ReLU(y_{l-1,j})$

$$E(x_{l,j}^2) = E(ReLU^2(y_{l-1,j}))$$

Then we have: $E(ReLU(y_{l-1,j})^2)$

$$= E(ReLU(y_{l-1,j})^2 | y_{l-1,j} > 0)P(y_{l-1,j} > 0) + E(ReLU(y_{l-1,j})^2 | y_{l-1,j} < 0)P(y_{l-1,j} < 0)$$

$$= E(ReLU(y_{l-1,j})^2 | y_{l-1,j} > 0)P(y_{l-1,j} > 0) = 0.5E(y_{l-1,j}^2) = 0.5var(y_{l-1,j})$$

Therefore, we have: $E(x_{l,j}^2) = 0.5var(y_{l-1,j})$

Given this, we have:

$$var(y_{l,i}) = n_l var(W_{l,i,j})E(x_{l,j}^2) = 0.5n_l var(W_{l,i,j})var(y_{l-1,j})$$

NYU SAI LAB

$E(ReLU(y_{l-1,j})^2) = E(ReLU(y_{l-1,j})^2 | y_{l-1,j} > 0)P(y_{l-1,j} > 0) + E(ReLU(y_{l-1,j})^2 | y_{l-1,j} < 0)P(y_{l-1,j} < 0)$

Derivation

$$var(y_{l,i}) = \left(\prod_{s=2}^{l} 0.5 n_s var(W_{s,i,j})\right) var(y_{1,j})$$

In order to ensure the variance of y does not change, we have to make sure:

$$var(W_{s,i,j}) = \frac{2}{n_s} \quad W_{s,i,j} \sim \mathcal{N}(0, \sqrt{\frac{2}{n_s}})$$

$$E(ReLU(y_{l-1,j})^2) = E(ReLU(y_{l-1,j})^2 | y_{l-1,j} > 0) P(y_{l-1,j} > 0) + E(ReLU(y_{l-1,j})^2 | y_{l-1,j} < 0) P(y_{l-1,j} < 0)$$