CPEN 455: Deep Learning

Lecture 3: Multilayer Perceptron

Renjie Liao

University of British Columbia

Winter, Term 2, 2024

Outline

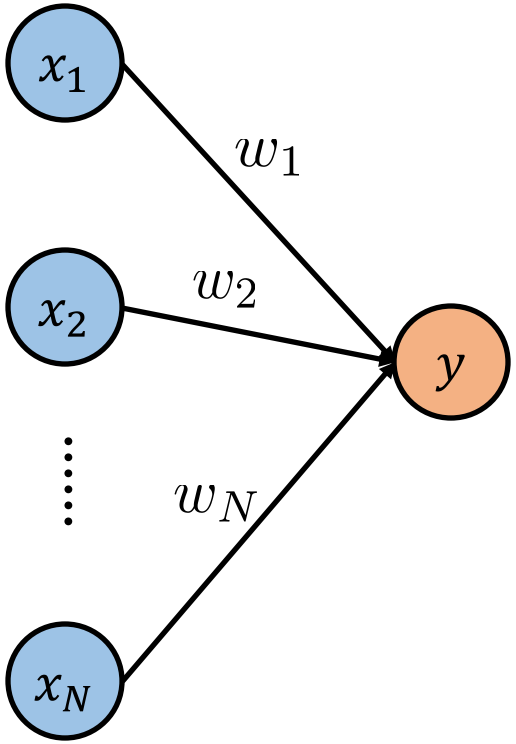
- Multilayer Perceptron (MLP)
  - Linear Layer
  - Nonlinear Activation
  - Batch Normalization
  - Dropout
  - Build a Deep MLP

Outline

- Multilayer Perceptron (MLP)
  - **Linear Layer**
  - Nonlinear Activation
  - Batch Normalization
  - Dropout
  - Build a Deep MLP

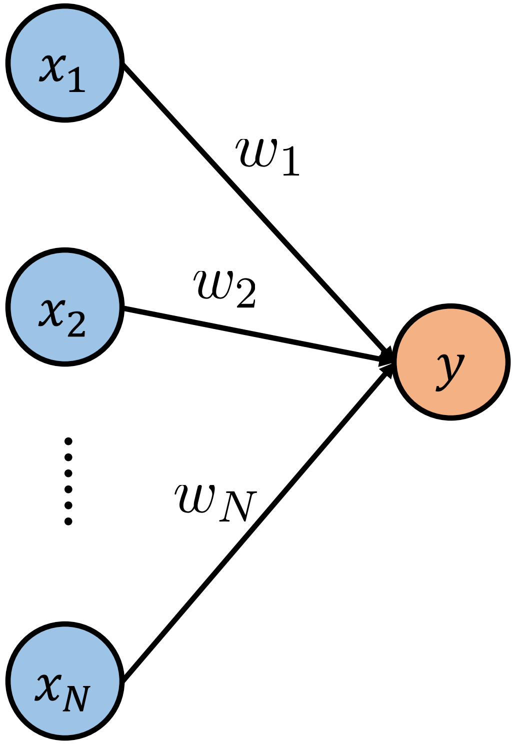
Linear Layer

Consider the following linear layer of a single output unit:



Linear Layer

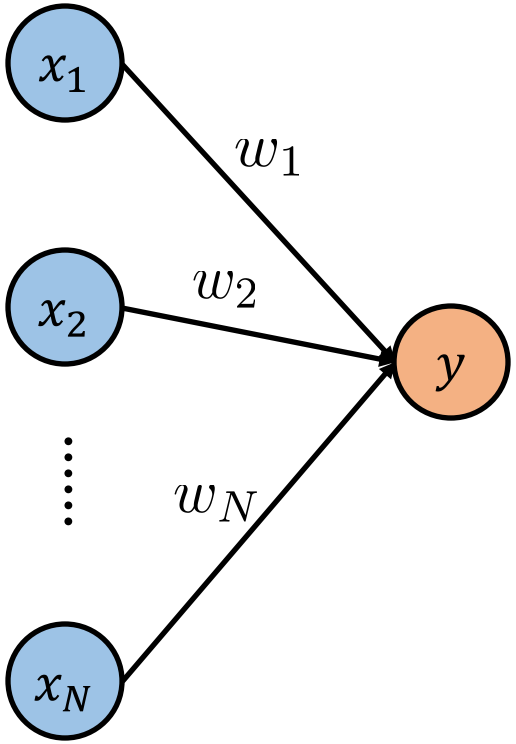
Consider the following linear layer of a single output unit:



$$y = \sum_{n=1}^{N} w_n x_n$$

Linear Layer

Consider the following linear layer of a single output unit:

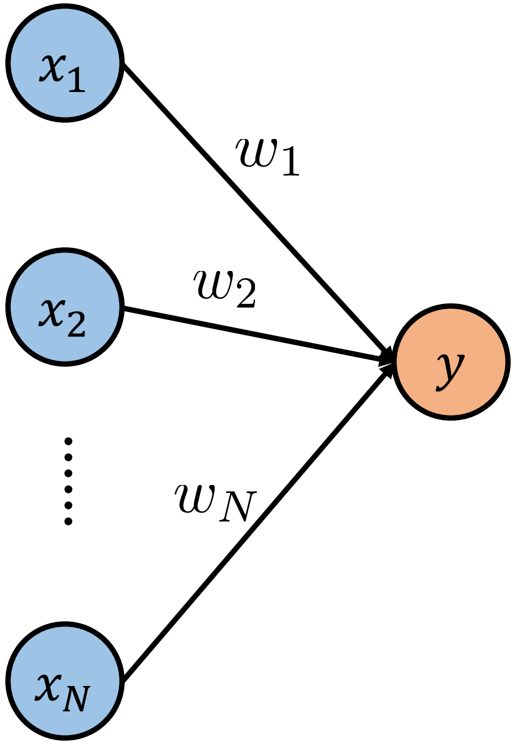


$$y = \sum_{n=1}^{N} w_n x_n$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Linear Layer

Consider the following linear layer of a single output unit:



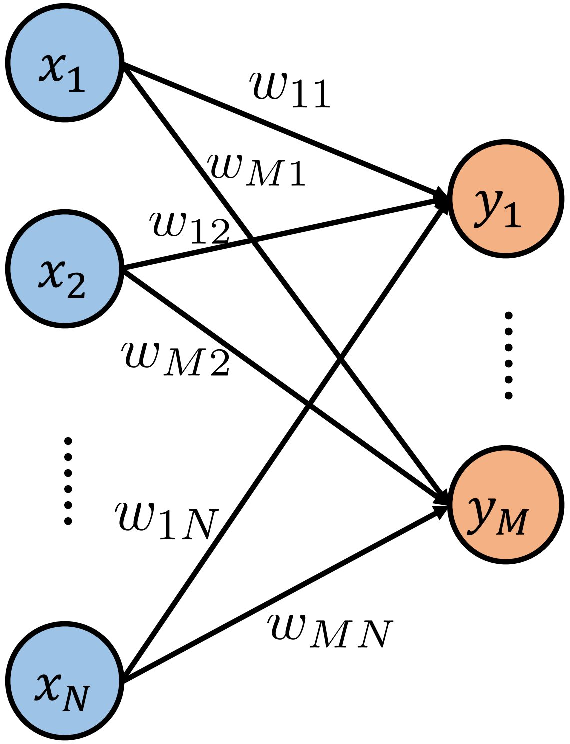
$$y = \sum_{n=1}^{N} w_n x_n = \mathbf{w}^\top \mathbf{x}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Linear Layer

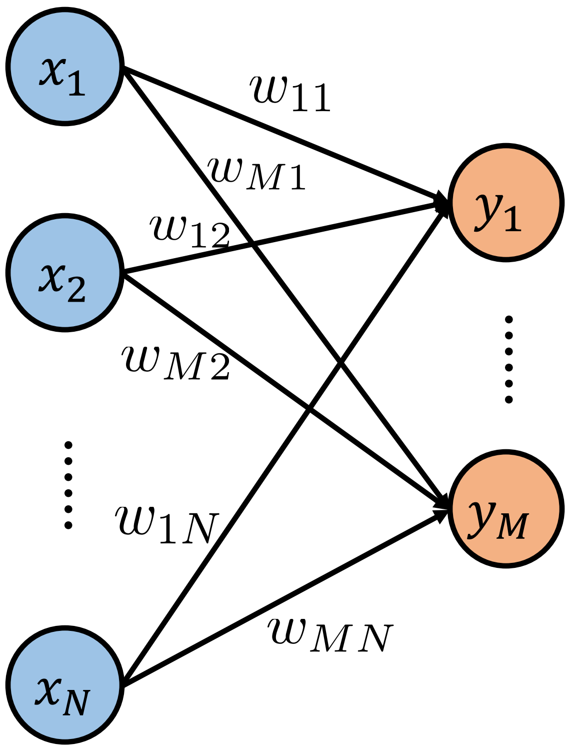
What if we have multiple output units?

$$y_m = \sum_{n=1}^{N} w_{mn} x_n$$



Linear Layer

What if we have multiple output units?

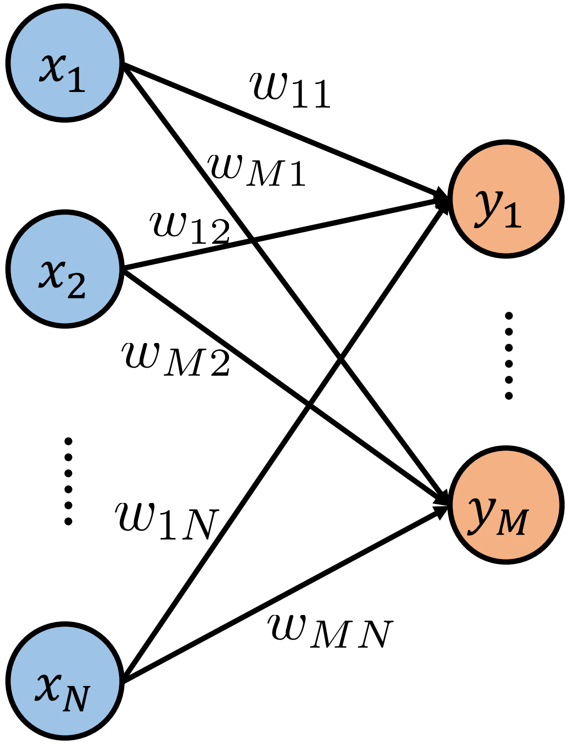


$$y_m = \sum_{n=1}^{N} w_{mn} x_n$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} \qquad W = \begin{bmatrix} w_{11}, & w_{12}, & \cdots, & w_{1N} \\ w_{21}, & w_{22}, & \cdots, & w_{2N} \\ \vdots, & \vdots, & \vdots, & \vdots \\ w_{M1}, & w_{M2}, & \cdots, & w_{MN} \end{bmatrix}$$

Linear Layer

What if we have multiple output units?



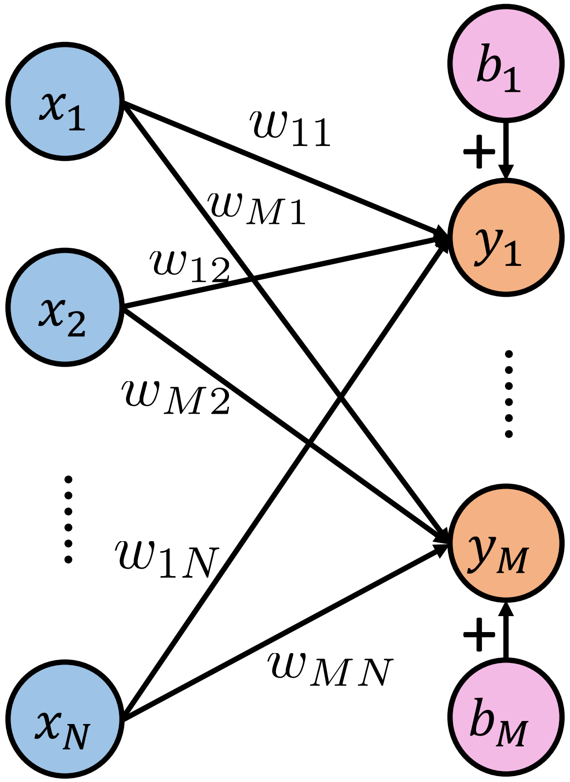
$$y_m = \sum_{n=1}^{N} w_{mn} x_n$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} \qquad W = \begin{bmatrix} w_{11}, & w_{12}, & \cdots, & w_{1N} \\ w_{21}, & w_{22}, & \cdots, & w_{2N} \\ \vdots, & \vdots, & \vdots, & \vdots \\ w_{M1}, & w_{M2}, & \cdots, & w_{MN} \end{bmatrix}$$

$$\mathbf{y} = W\mathbf{x}$$

Linear Layer

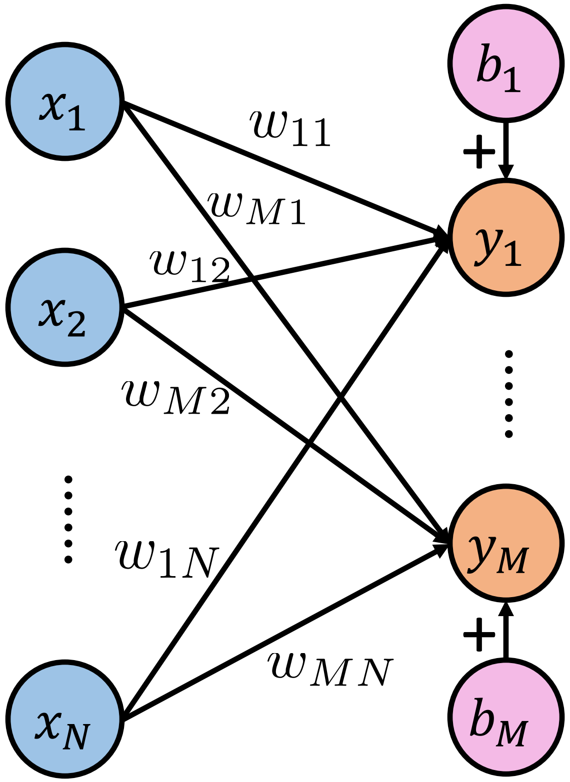
What if we have multiple output units? What about bias?



$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}$$

Linear Layer

What if we have multiple output units? What about bias?



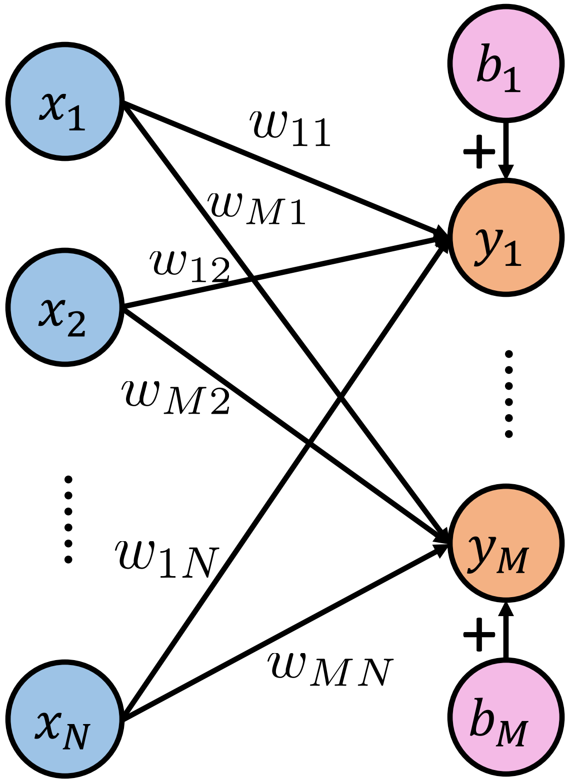
$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}$$

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

Linear Layer

What if we have multiple output units? What about bias?

We can compactly rewrite it via homogeneous coordinates.

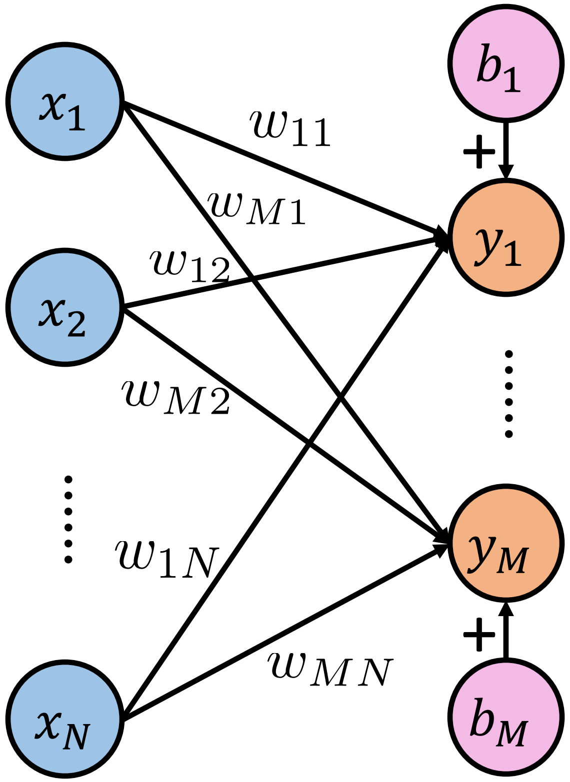


$$\tilde{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ 1 \end{bmatrix}$$

Linear Layer

What if we have multiple output units? What about bias?

We can compactly rewrite it via homogeneous coordinates.

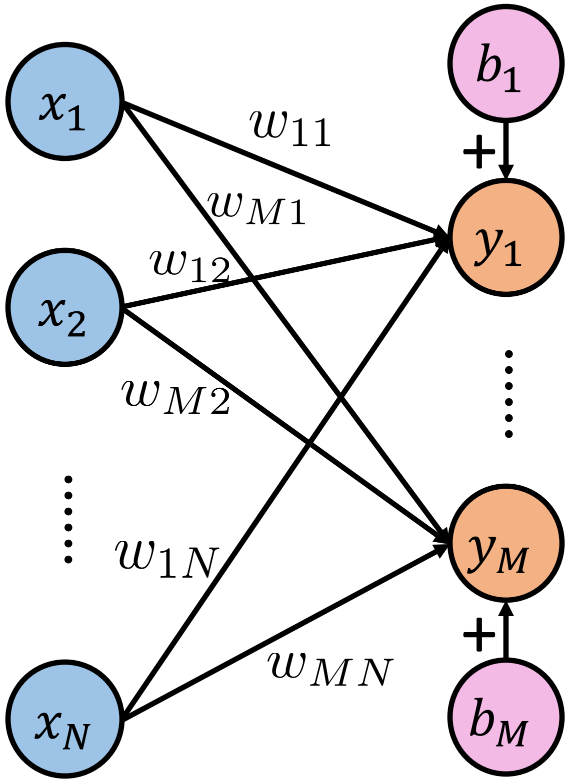


$$\tilde{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ 1 \end{bmatrix} \quad \tilde{W} = \begin{bmatrix} w_{11}, & w_{12}, & \cdots, & w_{1N} & b_1 \\ w_{21}, & w_{22}, & \cdots, & w_{2N} & b_2 \\ \vdots, & \vdots, & \vdots, & \vdots & \vdots \\ w_{M1}, & w_{M2}, & \cdots, & w_{MN} & b_M \end{bmatrix}$$

Linear Layer

What if we have multiple output units? What about bias?

We can compactly rewrite it via homogeneous coordinates.



$$\tilde{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ 1 \end{bmatrix} \quad \tilde{W} = \begin{bmatrix} w_{11}, & w_{12}, & \cdots, & w_{1N} & b_1 \\ w_{21}, & w_{22}, & \cdots, & w_{2N} & b_2 \\ \vdots, & \vdots, & \vdots, & \vdots & \vdots \\ w_{M1}, & w_{M2}, & \cdots, & w_{MN} & b_M \end{bmatrix}$$

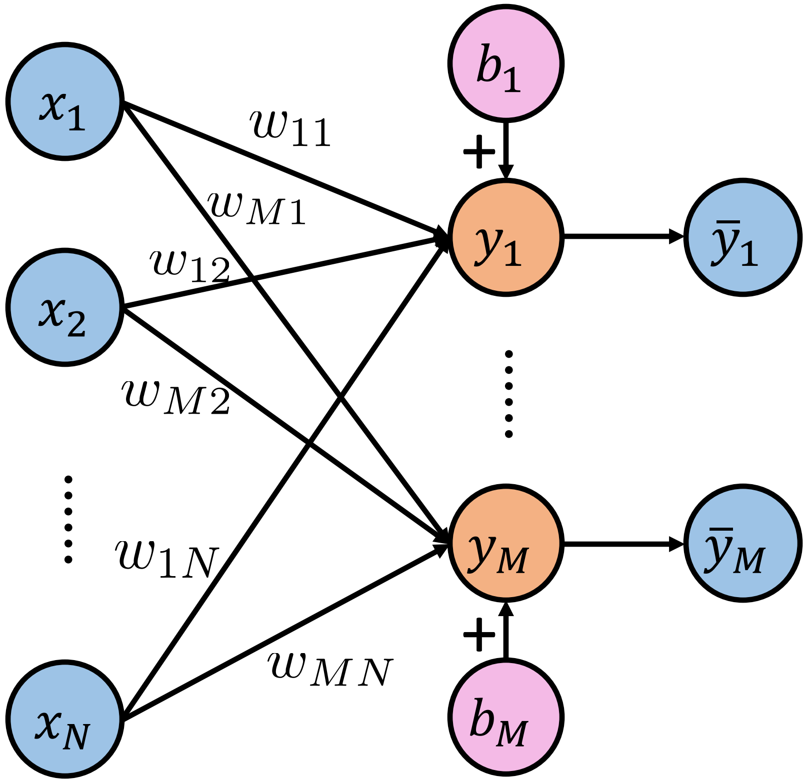
$$\mathbf{y} = W\mathbf{x} + \mathbf{b} = \tilde{W}\tilde{\mathbf{x}}$$

Outline

- Multilayer Perceptron (MLP)
  - Linear Layer
  - **Nonlinear Activation**
  - Batch Normalization
  - Dropout
  - Build a Deep MLP

Nonlinear Activation

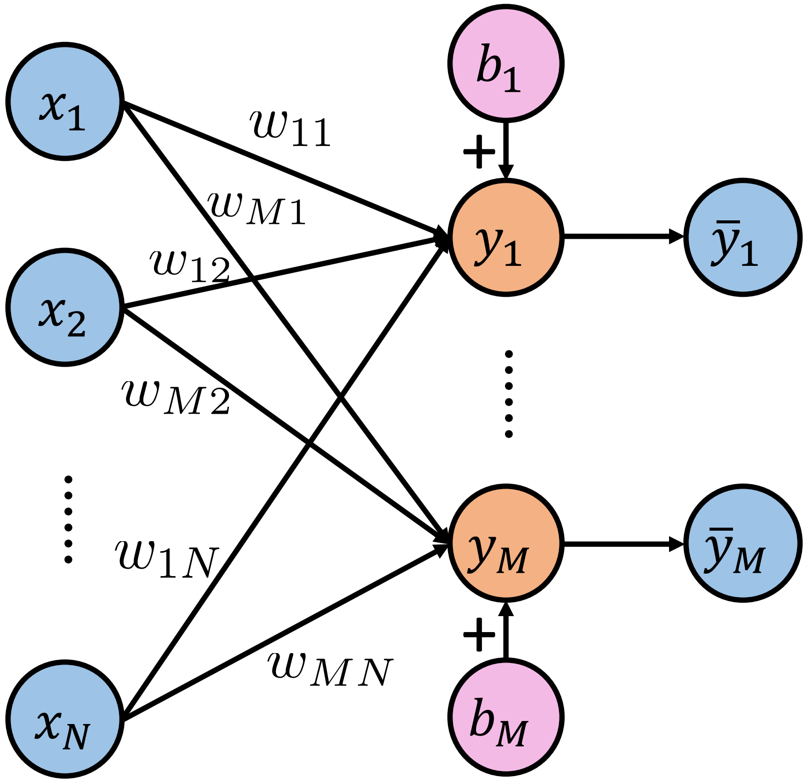
To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.



$$\bar{y}_i = f(y_i)$$

Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.



$$\bar{y}_i = f(y_i)$$

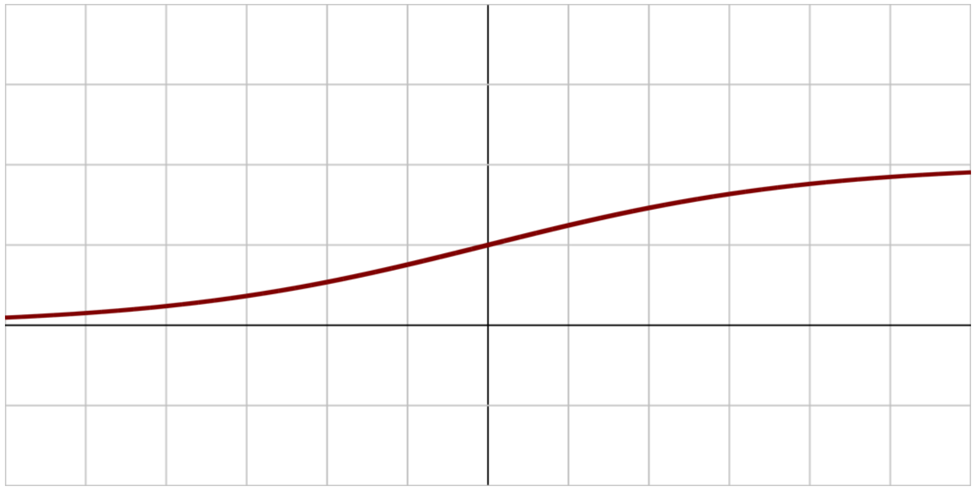
$$\bar{\mathbf{y}} = f(\mathbf{y})$$

People would clarify it if the nonlinear activation is not element-wise.

Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.

- **Sigmoid** $$f(x) = \frac{1}{1 + e^{-x}}$$

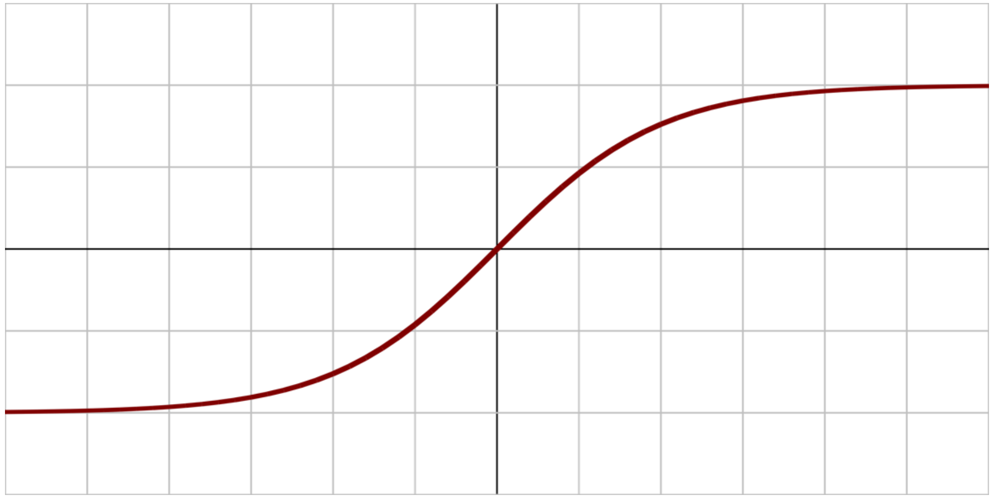


Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.

- Sigmoid
- **Tanh**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

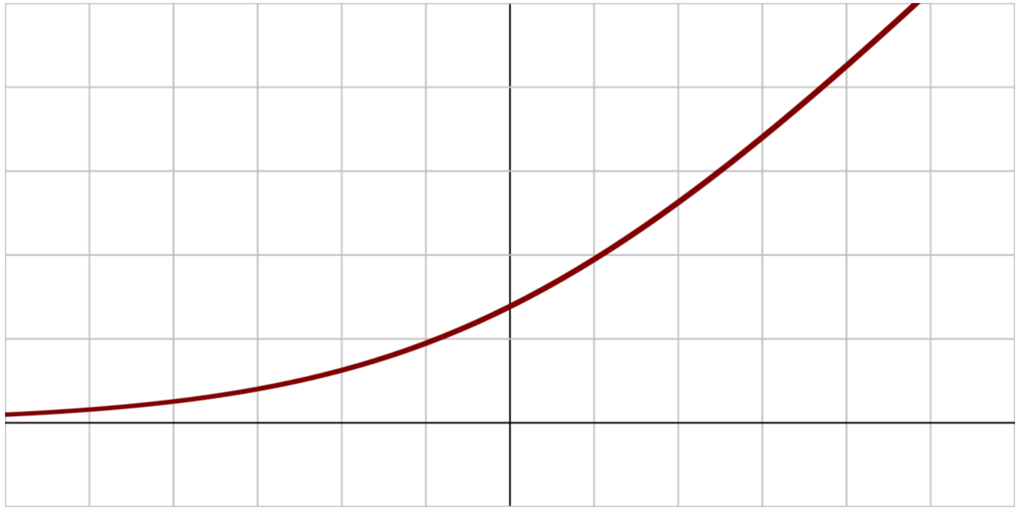


Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.

- Sigmoid
- Tanh
- **Softplus [1]**

$$f(x) = \ln(1 + e^x)$$

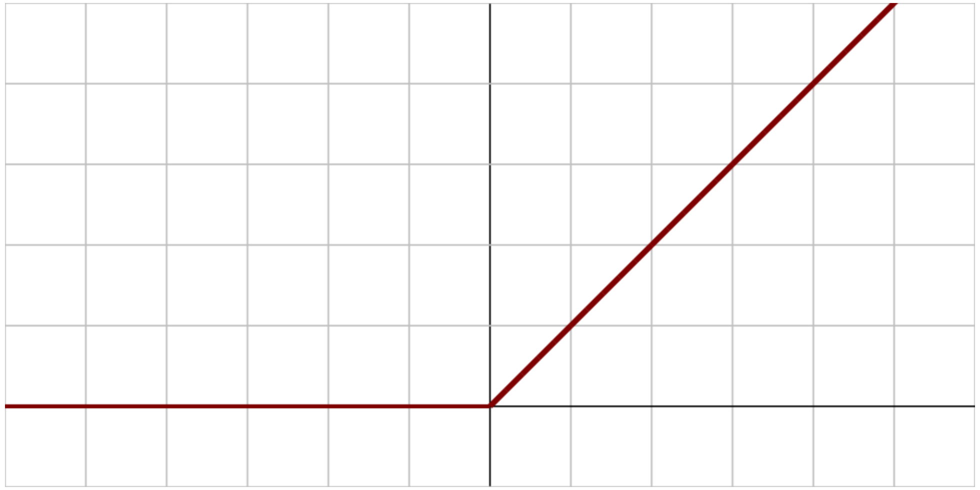


Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.

- Sigmoid
- Tanh
- Softplus [1]
- **Rectified Linear Units (ReLU) [2]**

$$f(x) = \max(x, 0)$$

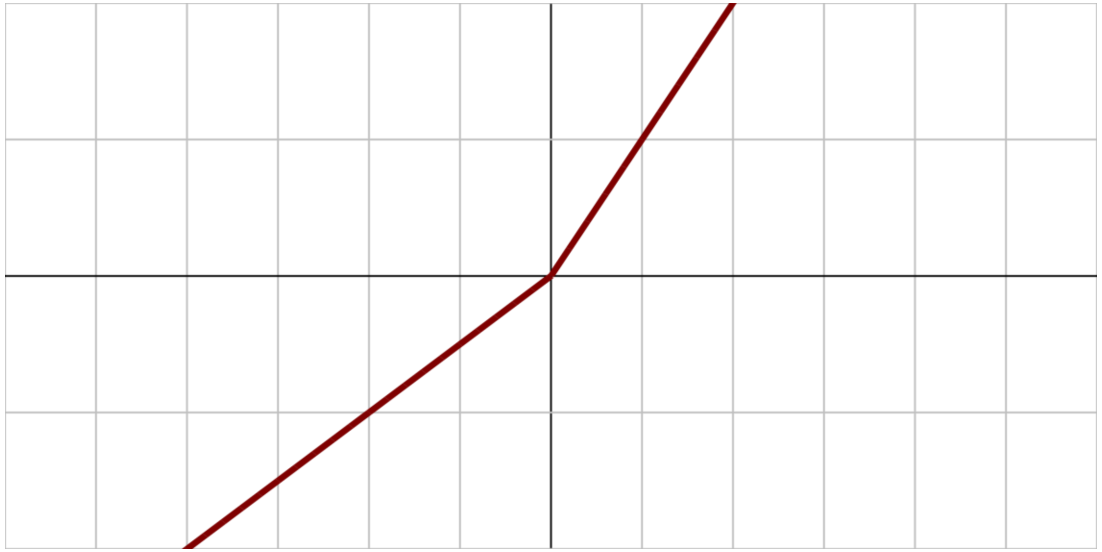


Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.

- Sigmoid
- Tanh
- Softplus [1]
- Rectified Linear Units (ReLU) [2]
- **Parametric rectified linear unit (PReLU) [3]**

$$f(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

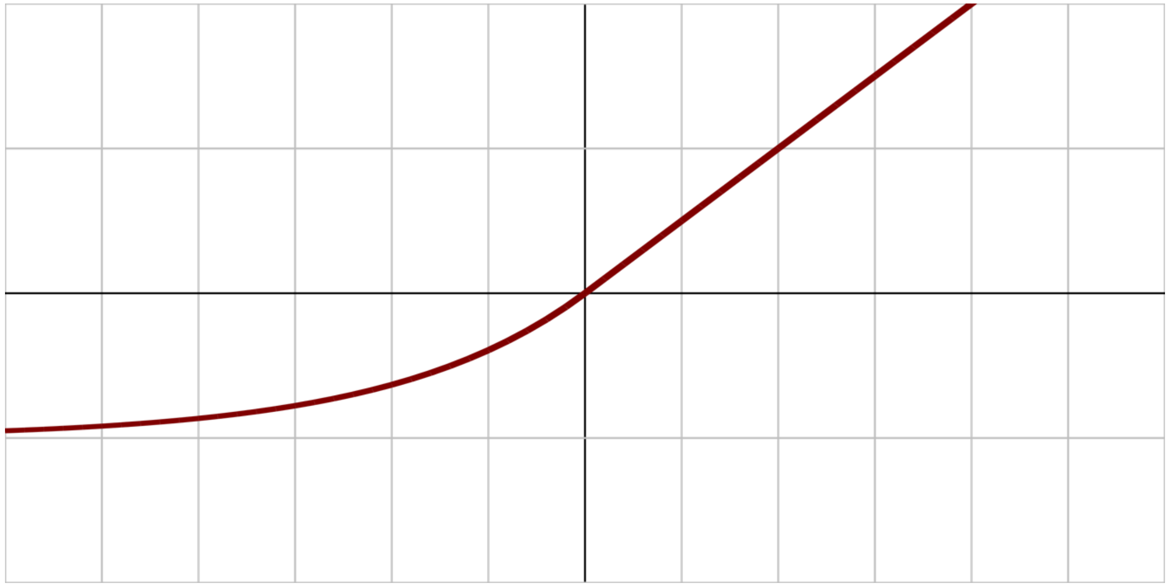


Nonlinear Activation

To make neural networks become nonlinear models, we often apply **element-wise** nonlinear activation functions.

$$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

- Sigmoid
- Tanh
- Softplus [1]
- Rectified Linear Units (ReLU) [2]
- Parametric rectified linear unit (PReLU) [3]
- **Exponential linear unit (ELU) [4]**



Nonlinear Activation

There also exists **non-element-wise** nonlinear activation functions.

Nonlinear Activation

There also exists **non-element-wise** nonlinear activation functions.

- Softmax

$$f(\mathbf{x}) = \left[\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, \frac{\exp(\mathbf{x}_K)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}\right]$$

Nonlinear Activation

There also exists **non-element-wise** nonlinear activation functions.

- Softmax

$$f(\mathbf{x}) = \left[\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{K} \exp(\mathbf{x}_k)}, \ldots, \frac{\exp(\mathbf{x}_K)}{\sum_{k=1}^{K} \exp(\mathbf{x}_k)} \right]$$

- Maxout [5]

$$f(\mathbf{x}) = \max_i \quad \mathbf{x}_i$$

Nonlinear Activation

There also exists **non-element-wise** nonlinear activation functions.

- Softmax

$$f(\mathbf{x}) = \left[\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, \frac{\exp(\mathbf{x}_K)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}\right]$$

- Maxout [5]

$$f(\mathbf{x}) = \max_i \quad \mathbf{x}_i$$

- Cummax [6]

$$f(\mathbf{x}) = \left[\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, \frac{\sum_{i=1}^{j}\exp(\mathbf{x}_i)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, 1\right]$$

Nonlinear Activation

There also exists **non-element-wise** nonlinear activation functions.

- Softmax

$$f(\mathbf{x}) = \left[\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, \frac{\exp(\mathbf{x}_K)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}\right]$$

- Maxout [5]

$$f(\mathbf{x}) = \max_i \quad \mathbf{x}_i$$

- Cummax [6]

$$f(\mathbf{x}) = \left[\frac{\exp(\mathbf{x}_1)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, \frac{\sum_{i=1}^{j}\exp(\mathbf{x}_i)}{\sum_{k=1}^{K}\exp(\mathbf{x}_k)}, \ldots, 1\right]$$

Softmax followed by cumulative sum

Outline

- Multilayer Perceptron (MLP)
    - Linear Layer
    - Nonlinear Activation
    - **Batch Normalization**
    - Dropout
    - Build a Deep MLP

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Reduce Internal Covariate Shift $\Rightarrow$ Improve the training (e.g., converge faster, generalize better)

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Reduce Internal Covariate Shift $\Rightarrow$ Improve the training (e.g., converge faster, generalize better)

Batch Normalization (BN) is a technique to achieve the goal!

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Reduce Internal Covariate Shift $\Rightarrow$ Improve the training (e.g., converge faster, generalize better)

Batch Normalization (BN) is a technique to achieve the goal!

Intuition:

1) If the internal activations are properly normalized, the neural network tends to be more stable

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Reduce Internal Covariate Shift $\Rightarrow$ Improve the training (e.g., converge faster, generalize better)

Batch Normalization (BN) is a technique to achieve the goal!

Intuition:

1) If the internal activations are properly normalized, the neural network tends to be more stable

Think about a sequence of linear layers where activations could easily blow up or vanish

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Reduce Internal Covariate Shift $\Rightarrow$ Improve the training (e.g., converge faster, generalize better)

Batch Normalization (BN) is a technique to achieve the goal!

Intuition:

1) If the internal activations are properly normalized, the neural network tends to be more stable

Think about a sequence of linear layers where activations could easily blow up or vanish

2) Normalizing the activations tends to better leverage the nonlinearity

Batch Normalization

Internal Covariate Shift [7]: *the change in the distribution of network activations due to the change in network parameters during training*

Reduce Internal Covariate Shift $\Rightarrow$ Improve the training (e.g., converge faster, generalize better)

Batch Normalization (BN) is a technique to achieve the goal!

Intuition:

1) If the internal activations are properly normalized, the neural network tends to be more stable

Think about a sequence of linear layers where activations could easily blow up or vanish

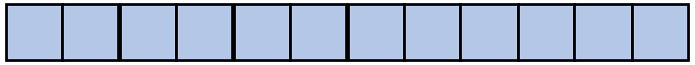
2) Normalizing the activations tends to better leverage the nonlinearity

Normalizing pre-activations helps avoid the saturation region of nonlinear activation function

Batch Normalization

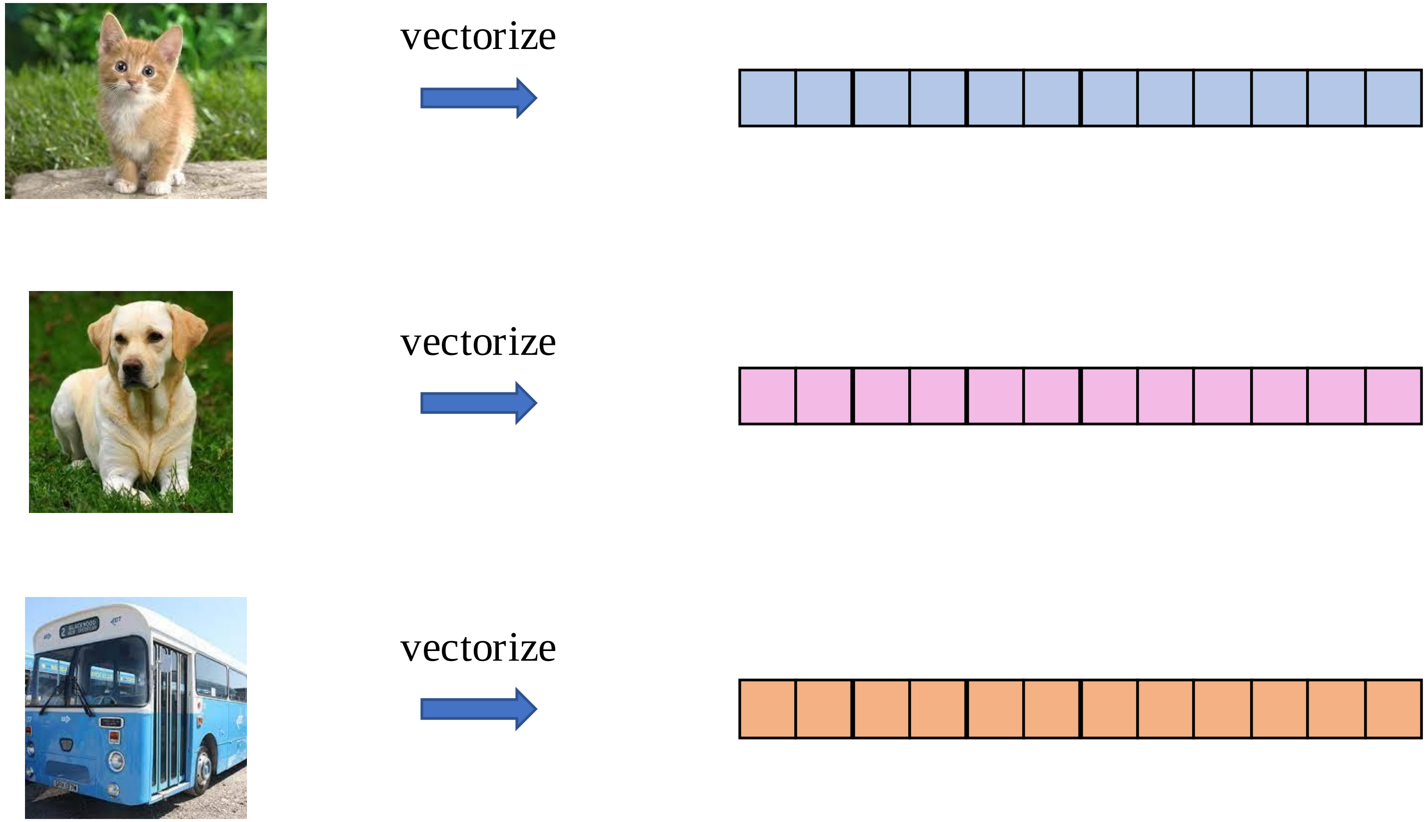
Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

vectorize



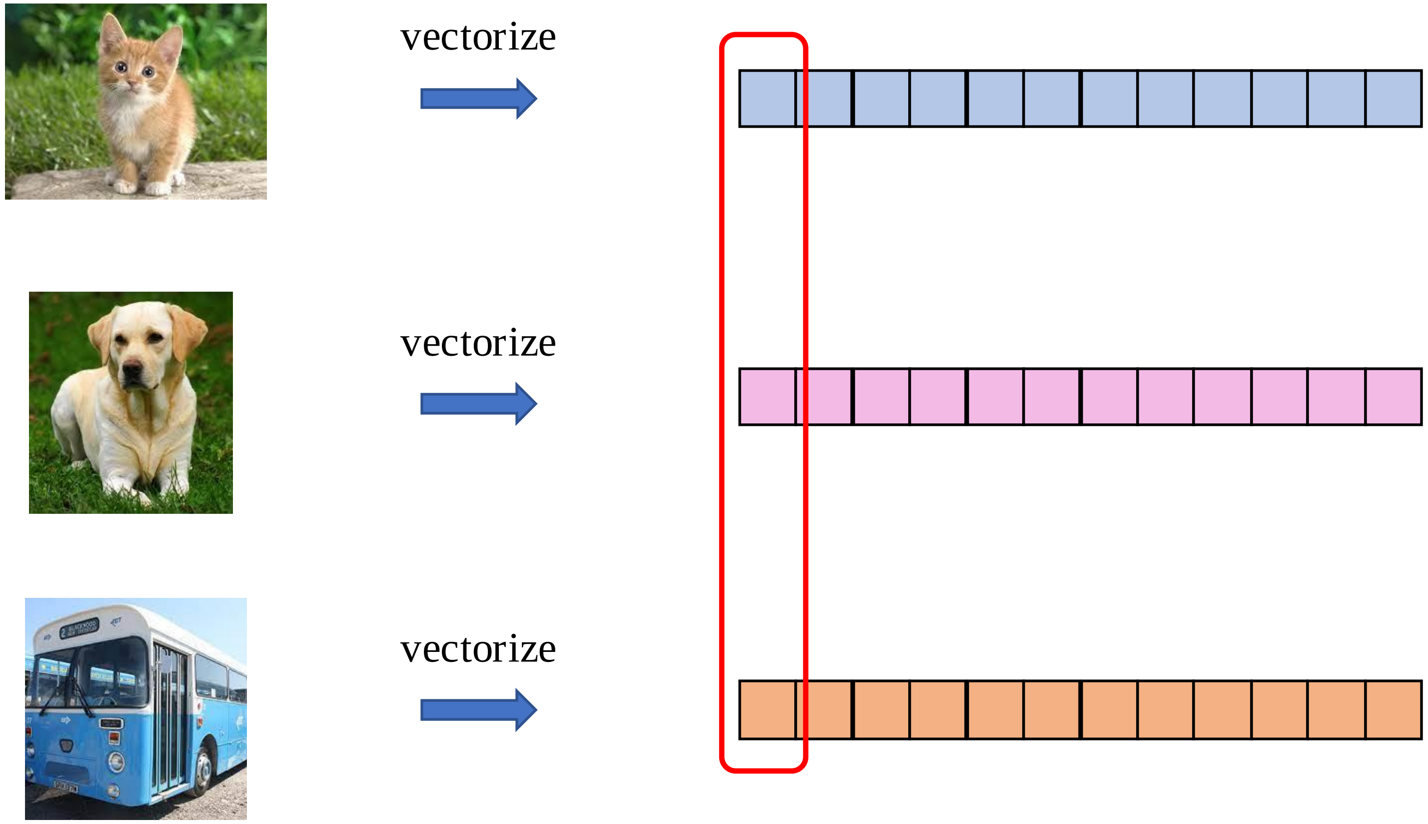
Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$



Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$



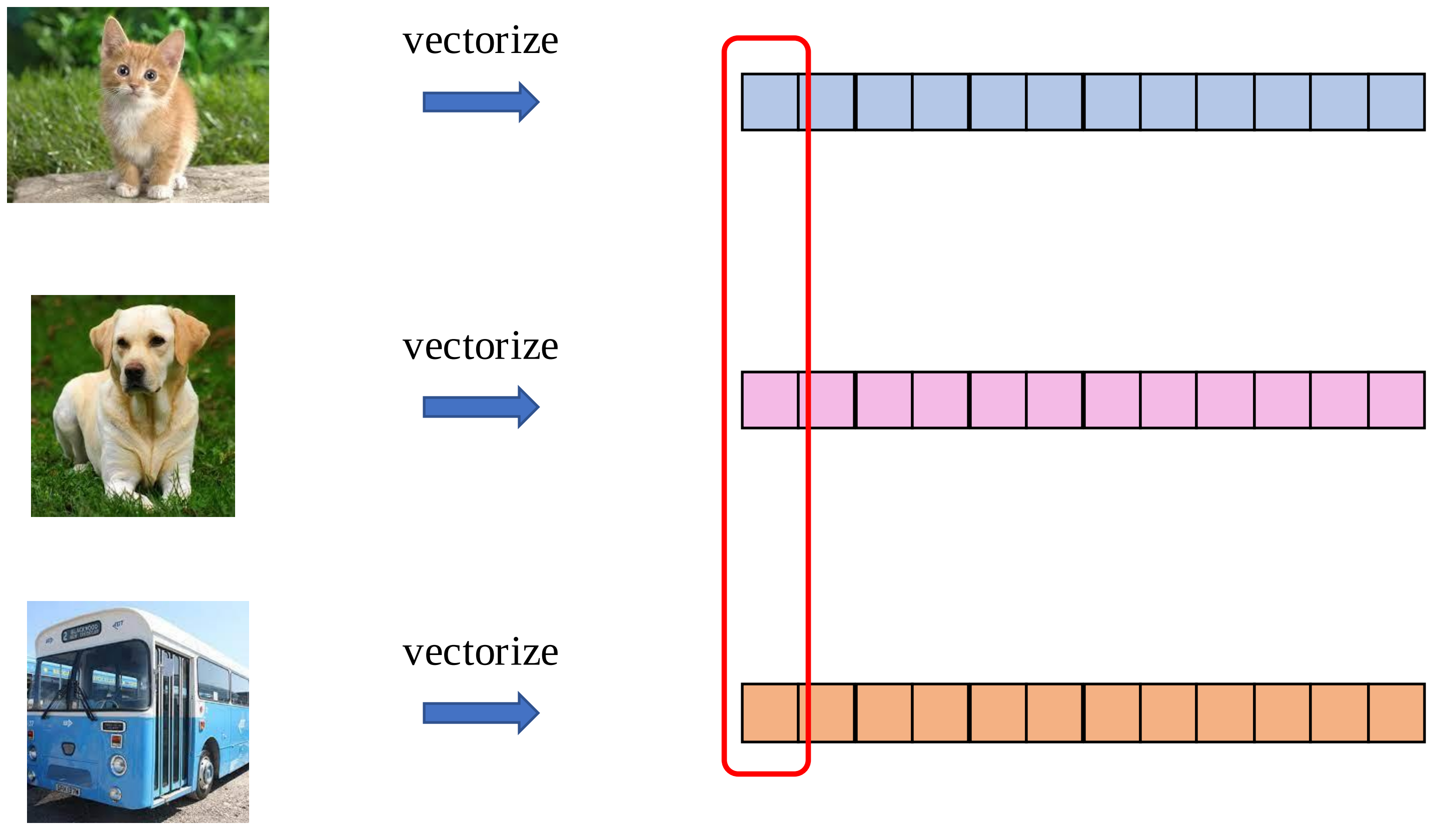
$$\mathbf{m}[j] = \frac{1}{B} \sum_{i=1}^{B} X[i,j]$$

$$\Downarrow$$

$$\mathbf{m} = \frac{1}{B} \sum_{i=1}^{B} X[i,:]$$

Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

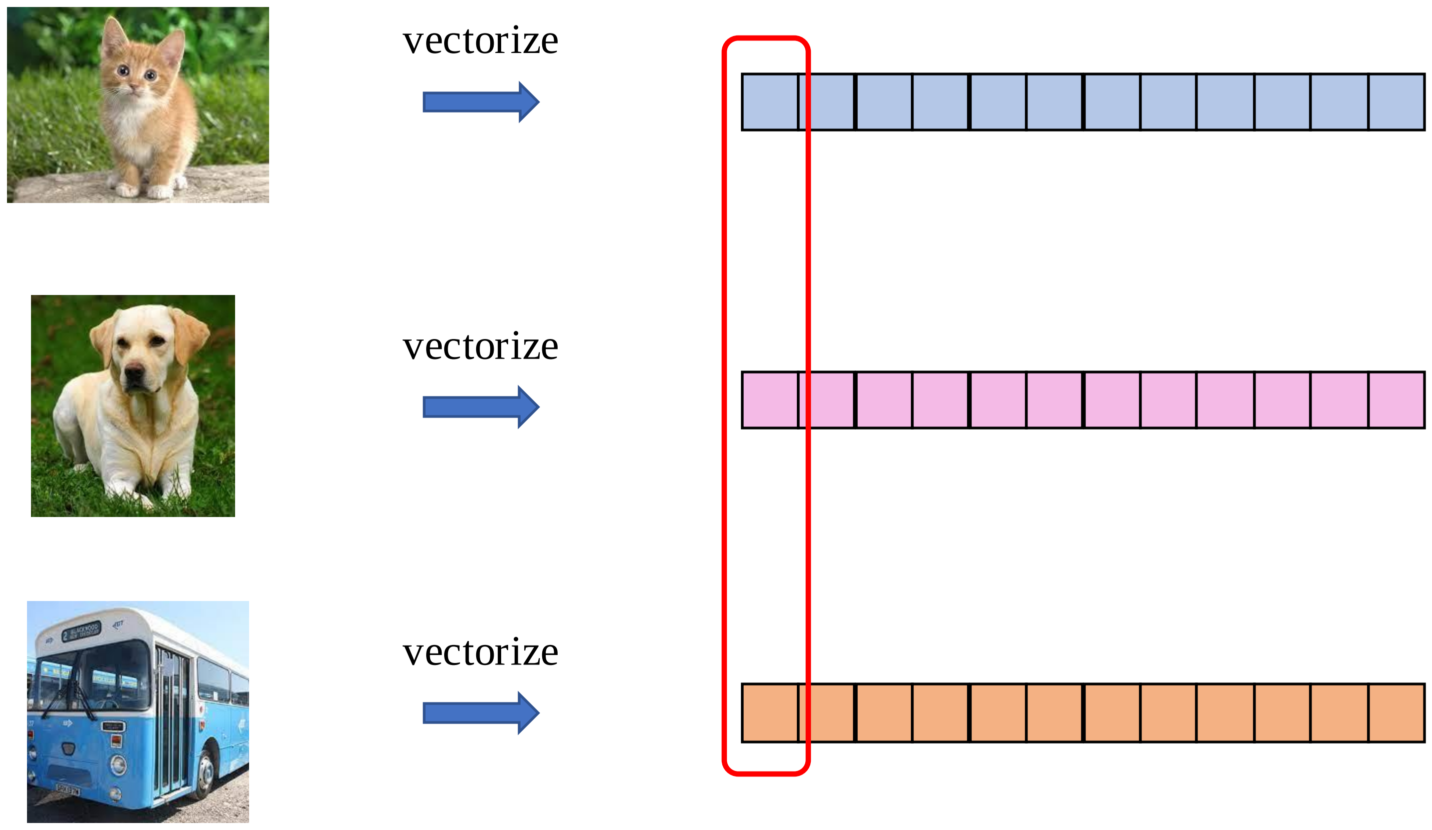


$$\mathbf{m}[j] = \frac{1}{B} \sum_{i=1}^{B} X[i, j]$$

$$\mathbf{v}[j] = \frac{1}{B} \sum_{i=1}^{B} (X[i, j] - \mathbf{m}[j])^2$$

Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$



$$\mathbf{m}[j] = \frac{1}{B} \sum_{i=1}^{B} X[i,j]$$

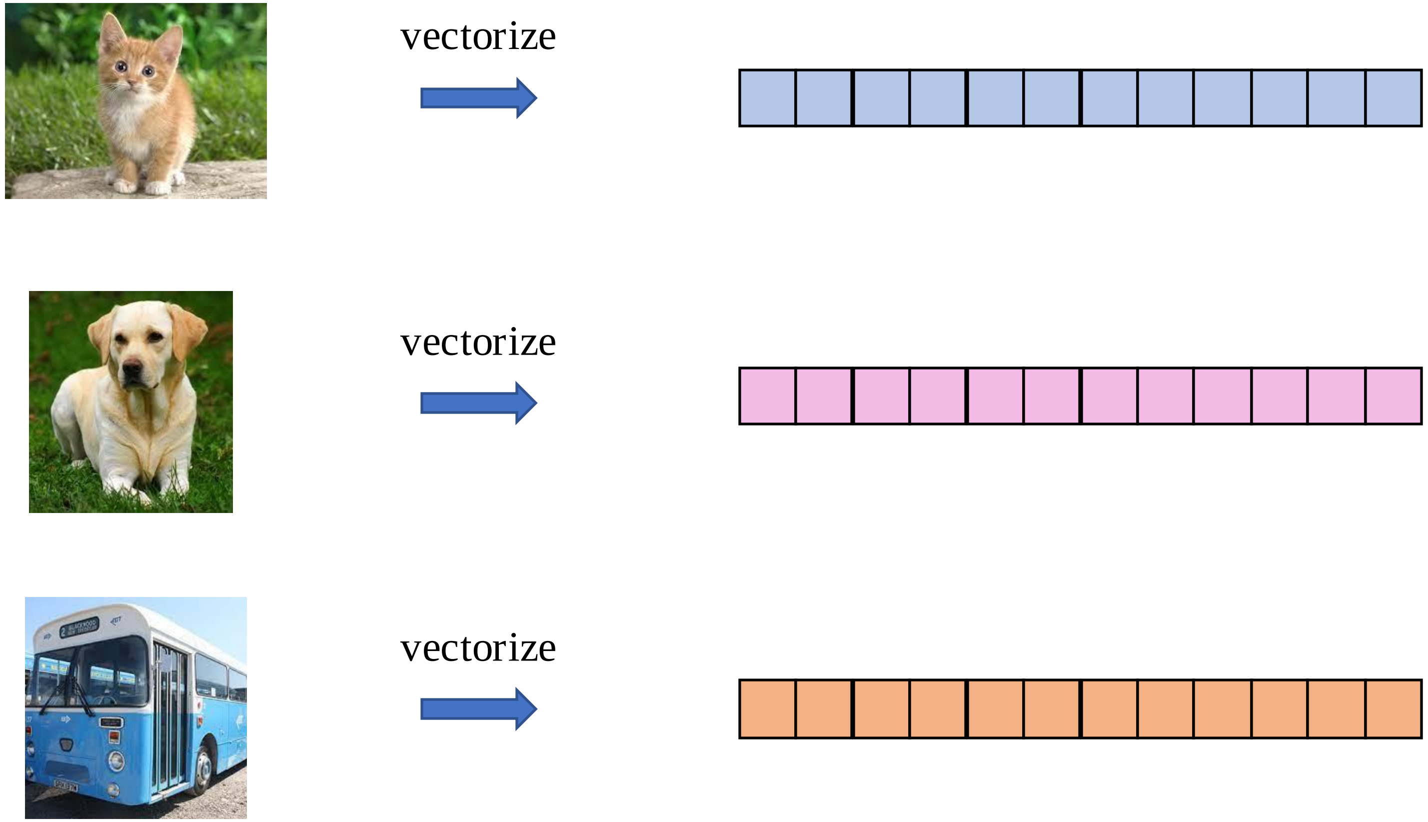
$$\mathbf{v}[j] = \frac{1}{B} \sum_{i=1}^{B} (X[i,j] - \mathbf{m}[j])^2$$

normalized activations:

$$\hat{X}[i,j] = \boldsymbol{\gamma}[j] \frac{X[i,j] - \mathbf{m}[j]}{\sqrt{\mathbf{v}[j] + \epsilon}} + \boldsymbol{\beta}[j]$$

Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$



$$\mathbf{m}[j] = \frac{1}{B}\sum_{i=1}^{B} X[i,j]$$

$$\mathbf{v}[j] = \frac{1}{B}\sum_{i=1}^{B} (X[i,j] - \mathbf{m}[j])^2$$

normalized activations:

$$\hat{X}[i,j] = \boxed{\boldsymbol{\gamma}[j]}\frac{X[i,j] - \mathbf{m}[j]}{\sqrt{\mathbf{v}[j] + \epsilon}} + \boxed{\boldsymbol{\beta}[j]}$$

Learnable Parameters

Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

$$\mathbf{m}[j] = \frac{1}{B} \sum_{i=1}^{B} X[i,j]$$

$$\mathbf{v}[j] = \frac{1}{B} \sum_{i=1}^{B} (X[i,j] - \mathbf{m}[j])^2$$

$$\hat{X}[i,j] = \boldsymbol{\gamma}[j] \frac{X[i,j] - \mathbf{m}[j]}{\sqrt{\mathbf{v}[j] + \epsilon}} + \boldsymbol{\beta}[j]$$

Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

$$\mathbf{m}[j] = \frac{1}{B}\sum_{i=1}^{B} X[i,j]$$

$$\mathbf{v}[j] = \frac{1}{B}\sum_{i=1}^{B}(X[i,j] - \mathbf{m}[j])^2$$

$$\hat{X}[i,j] = \boldsymbol{\gamma}[j]\frac{X[i,j] - \mathbf{m}[j]}{\sqrt{\mathbf{v}[j] + \epsilon}} + \boldsymbol{\beta}[j]$$

In practice, to account for the dynamically changing weights and stochastic data, we use running mean and variance:

$$\mathbf{m}^t[j] = (1-\alpha)\mathbf{m}^{t-1}[j] + \alpha\mathbf{m}[j]$$
$$\mathbf{v}^t[j] = (1-\alpha)\mathbf{v}^{t-1}[j] + \alpha\mathbf{v}[j]$$

$\alpha \in [0,1]$ is a hyperparameter

Batch Normalization

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

$$\mathbf{m}[j] = \frac{1}{B} \sum_{i=1}^{B} X[i,j]$$

$$\mathbf{v}[j] = \frac{1}{B} \sum_{i=1}^{B} (X[i,j] - \mathbf{m}[j])^2$$

$$\hat{X}[i,j] = \boldsymbol{\gamma}[j] \frac{X[i,j] - \mathbf{m}[j]}{\sqrt{\mathbf{v}[j] + \epsilon}} + \boldsymbol{\beta}[j]$$

In practice, to account for the dynamically changing weights and stochastic data, we use running mean and variance:

$$\mathbf{m}^t[j] = (1-\alpha)\mathbf{m}^{t-1}[j] + \alpha \mathbf{m}[j]$$
$$\mathbf{v}^t[j] = (1-\alpha)\mathbf{v}^{t-1}[j] + \alpha \mathbf{v}[j]$$

$\alpha \in [0,1]$ is a hyperparameter

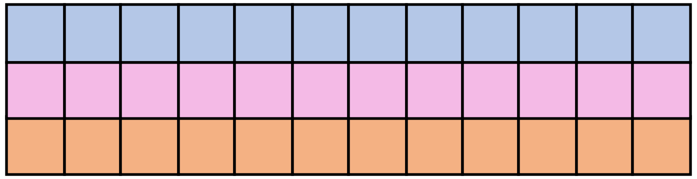
BN can be generalized to convolutional neural networks (CNNs). We will cover that in the future.

Outline

- Multilayer Perceptron (MLP)
  - Linear Layer
  - Nonlinear Activation
  - Batch Normalization
  - **Dropout**
  - Build a Deep MLP

Dropout

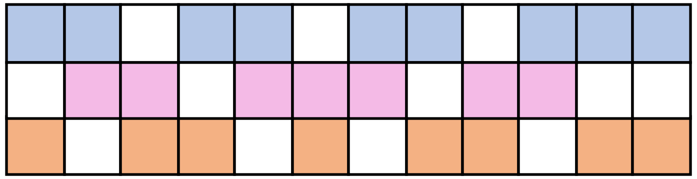
Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$



Dropout

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

We can make them stochastic by randomly turning some units off

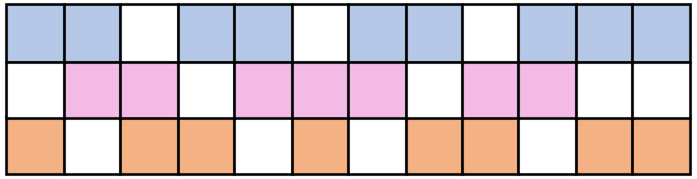


Dropout

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

We can make them stochastic by randomly turning some units off

Mathematically, we create a matrix mask $M \in \mathbb{R}^{B \times D}$ as follows



$$M[i, j] \sim \mathrm{Bernoulli}(1 - p)$$

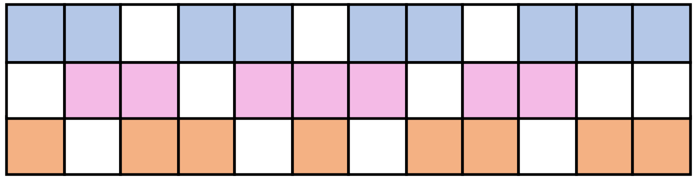
$$\mathbb{P}(M[i, j] = 1) = 1 - p$$
$$\mathbb{P}(M[i, j] = 0) = p$$

Dropout

Suppose we have a batch of activations $X \in \mathbb{R}^{B \times D}$

We can make them stochastic by randomly turning some units off

Mathematically, we create a matrix mask $M \in \mathbb{R}^{B \times D}$ as follows



$$M[i,j] \sim \mathrm{Bernoulli}(1-p)$$

$$\mathbb{P}(M[i,j]=1) = 1-p$$
$$\mathbb{P}(M[i,j]=0) = p$$

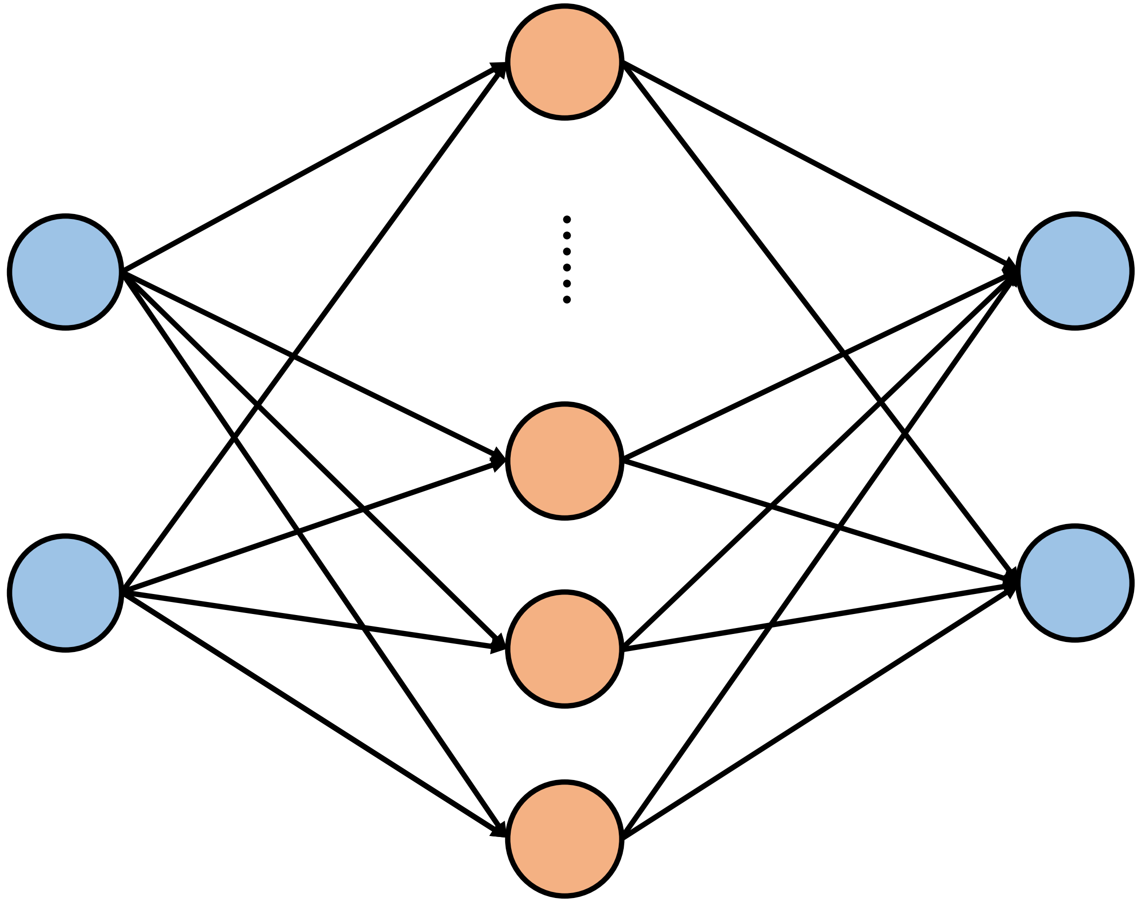
Then we perform element-wise product (a.k.a. Hadamard product)

$$\hat{X} = M \odot X$$

$$\hat{X}[i,j] = M[i,j]X[i,j]$$

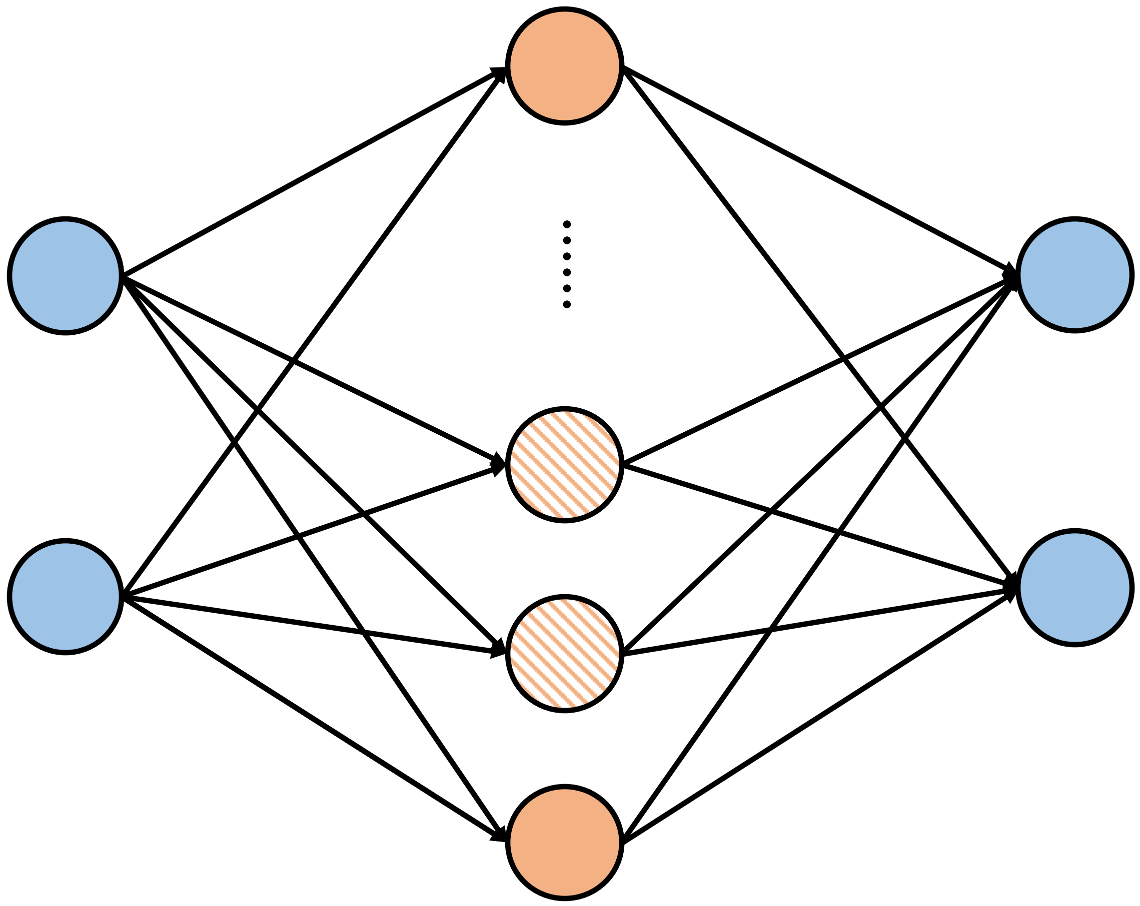
Dropout

What does Dropout [8] do to the neural networks?



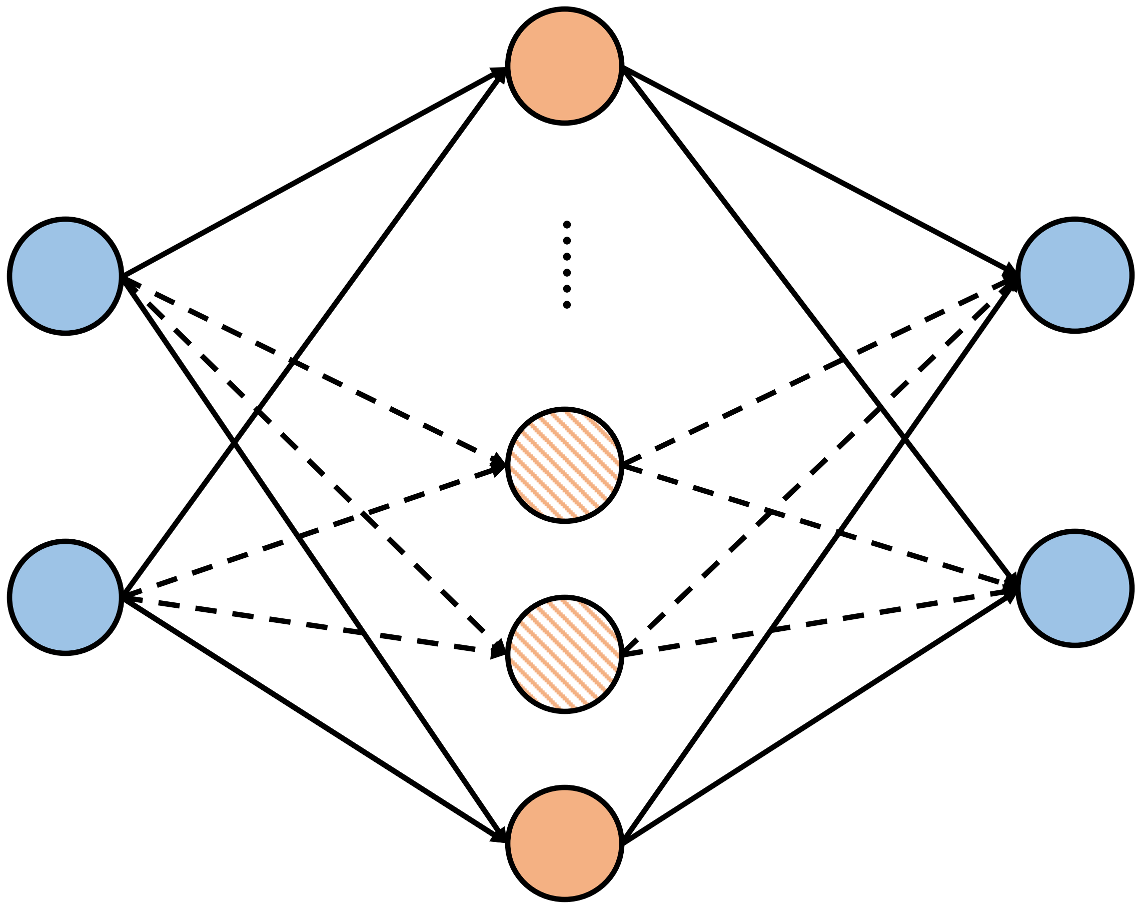
Dropout

What does Dropout [8] do to the neural networks?



Dropout

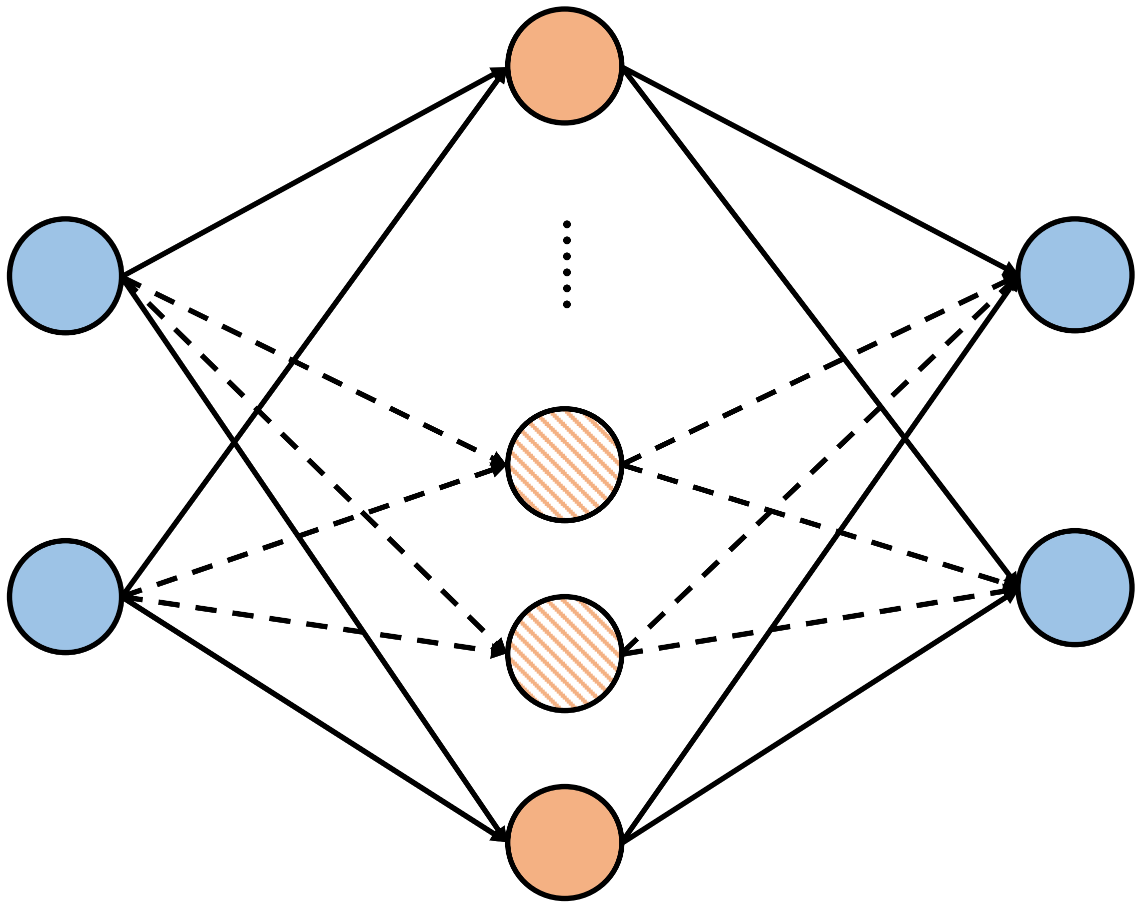
What does Dropout [8] do to the neural networks?



Dropout

What does Dropout [8] do to the neural networks?

It turns off a random subset of units, thus blocking a random subset of paths!

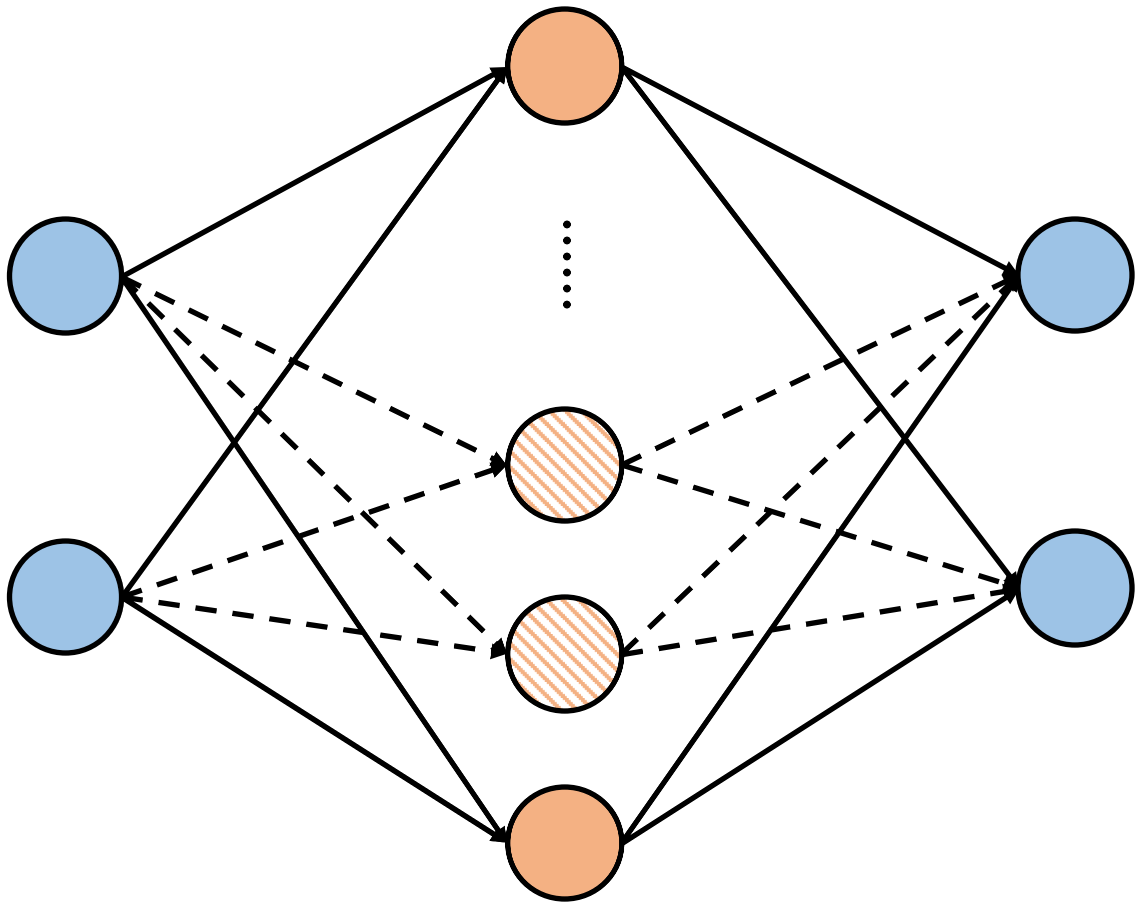


Dropout

What does Dropout [8] do to the neural networks?

It turns off a random subset of units, thus blocking a random subset of paths!

Every sample gets its own sub-network, thus being less likely to overfit.



Dropout

How can we perform testing with Dropout?

Dropout

How can we perform testing with Dropout?

We can compute the expected output for each unit!

Dropout

How can we perform testing with Dropout?

We can compute the expected output for each unit!

Recall

$$\hat{X}[i,j] = M[i,j]X[i,j]$$
$$\mathbb{P}(M[i,j] = 1) = 1 - p$$
$$\mathbb{P}(M[i,j] = 0) = p$$

Dropout

How can we perform testing with Dropout?

We can compute the expected output for each unit!

Recall

$$\hat{X}[i,j] = M[i,j]X[i,j]$$
$$\mathbb{P}(M[i,j]=1) = 1-p$$
$$\mathbb{P}(M[i,j]=0) = p$$

We can compute the expectation as

$$\begin{aligned}\mathbb{E}\left[\hat{X}[i,j]\right] &= \mathbb{E}\left[M[i,j]X[i,j]\right] \\ &= \mathbb{E}\left[M[i,j]\right]X[i,j] \\ &= \left(1 \times \mathbb{P}(M[i,j]=1) + 0 \times \mathbb{P}(M[i,j]=0)\right)X[i,j] \\ &= (1-p)X[i,j]\end{aligned}$$

Outline

- Multilayer Perceptron (MLP)
    - Linear Layer
    - Nonlinear Activation
    - Batch Normalization
    - Dropout
    - **Build a Deep MLP**

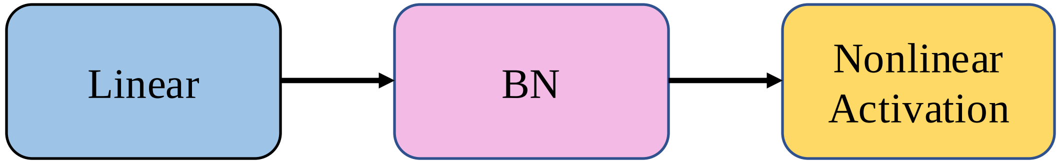
Putting Pieces Together

Now we learn linear layers, nonlinear activations, batch normalization (BN), dropout, how can we build a deep MLP?

Putting Pieces Together

Now we learn linear layers, nonlinear activations, batch normalization (BN), dropout, how can we build a deep MLP?

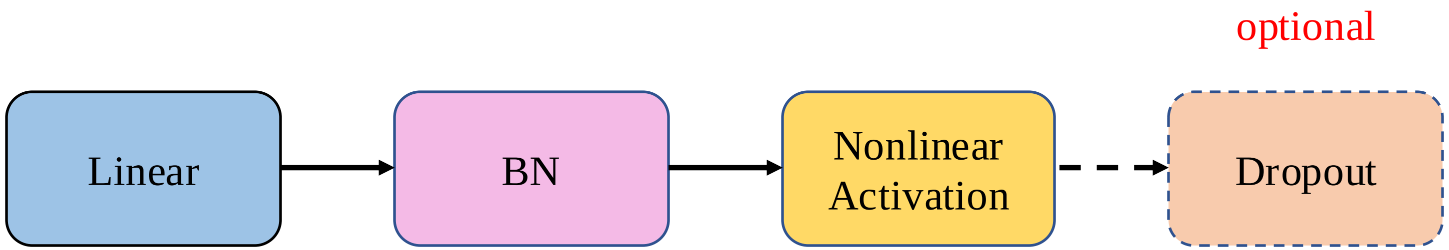
Let us first build a block:



Putting Pieces Together

Now we learn linear layers, nonlinear activations, batch normalization (BN), dropout, how can we build a deep MLP?

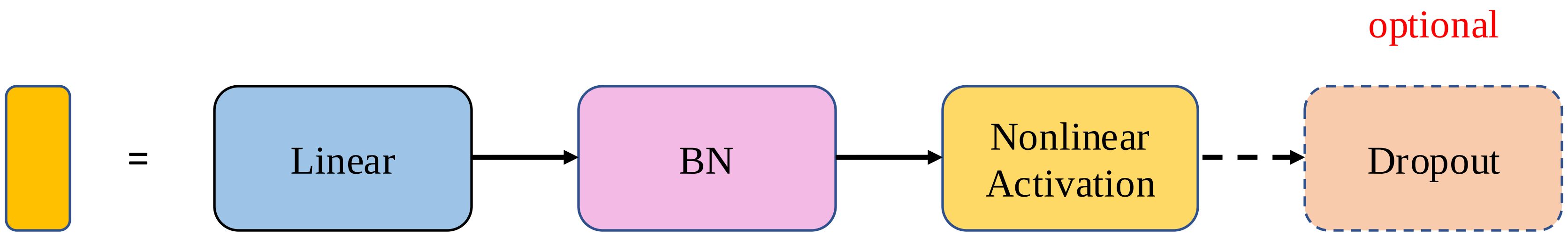
Let us first build a block:



Putting Pieces Together

Now we learn linear layers, nonlinear activations, batch normalization (BN), dropout, how can we build a deep MLP?

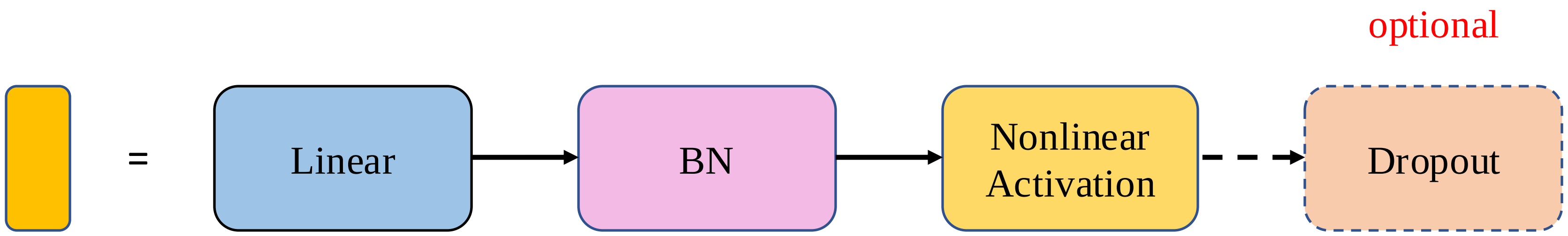
Let us first build a block:



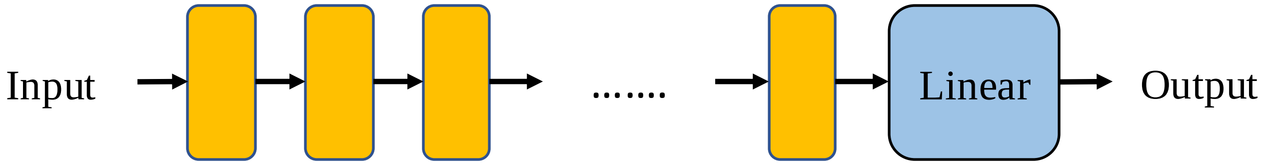
Putting Pieces Together

Now we learn linear layers, nonlinear activations, batch normalization (BN), dropout, how can we build a deep MLP?

Let us first build a block:



Then we can build a deep MLP:



References

[1] Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2000). Incorporating second-order functional knowledge for better option pricing. Advances in neural information processing systems , 13 .

[2] Nair, V., & Hinton, G. E. (2010, January). Rectified linear units improve restricted boltzmann machines. In ICML.

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision (pp. 1026-1034).

[4] Clevert, D. A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289.

[5] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013, May). Maxout networks. In International conference on machine learning (pp. 1319-1327). PMLR.

[6] Shen, Y., Tan, S., Sordoni, A., & Courville, A. (2018). Ordered neurons: Integrating tree structures into recurrent neural networks. arXiv preprint arXiv:1810.09536.

[7] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

[8] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

Questions?