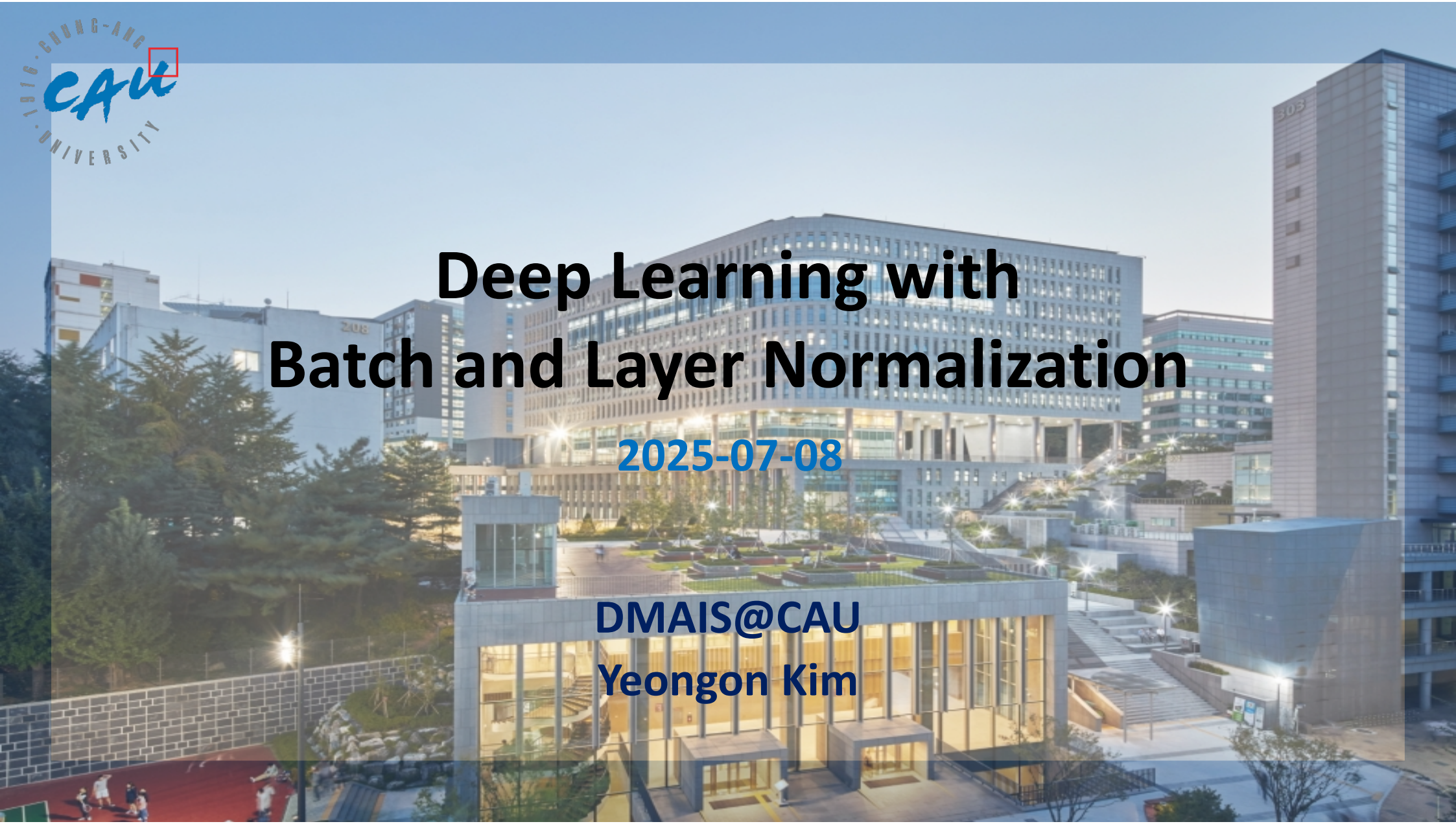


Deep Learning with Batch and Layer Normalization

2025-07-08

DMAIS@CAU

Yeongon Kim

INDEX

- Introduction to Deep Learning
- Fundamentals of Deep Learning
- Convolutional Neural Networks
- Recurrent Neural Network
- Batch and Layer Normalization

Papers

- “Deep Learning.”, Nature 2015, LeCun, Y., Bengio, Y. & Hinton, G.
- “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015, Ioffe, S. & Szegedy, C.
- "Layer Normalization”, Preprint 2016, Ba, J. L., Kiros, J. R. & Hinton, G. E.

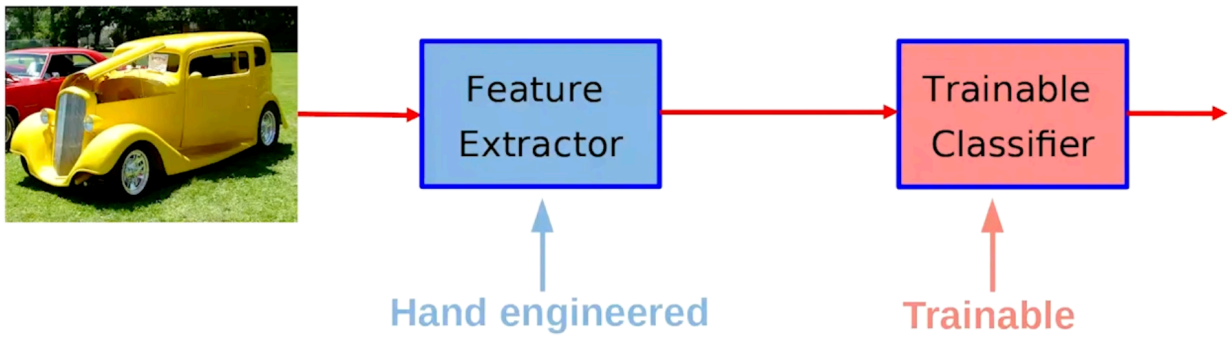
Introduction to Deep Learning

What is Machine Learning?

- Study of algorithms that learns from data and perform tasks without explicit instructions

Limitations of Conventional Machine Learning

- Requires careful engineering and high domain expertise to design a feature extractor
- Selectivity-Invariance Dilemma



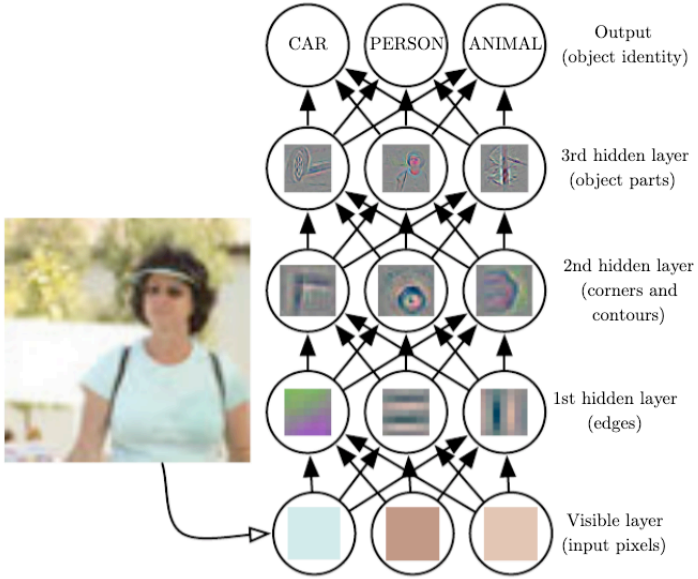
Introduction to Deep Learning

- **Selectivity-Invariance Dilemma**
  - Selectivity: the aspects of the image that are important for discrimination
  - Invariance: invariant to irrelevant aspects such as the pose of the animal
  - If a feature extractor is too sensitive, you gain selectivity but lose invariance
    if it is too insensitive, you gain invariance but sacrifice selectivity

➡ **Deep learning handles it with multilayer stack of simple modules**

Introduction to Deep Learning

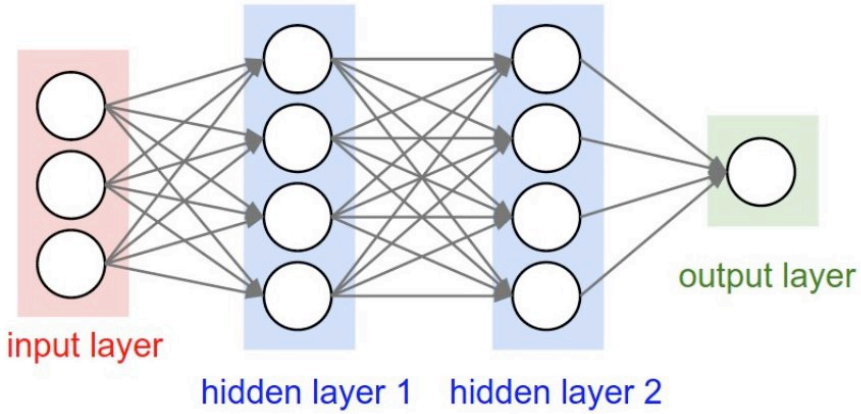
- **How does stacking layers solve this dilemma?**
  - **Natural signal has compositional hierarchies**
  - **Local combinations of edges form motifs, motifs assemble into parts, and parts to objects**
  - **Lower layers: small receptive fields detect edges & corners → selectivity**
  - **Higher layers: large receptive fields combine & abstract features → invariance**



Fundamentals of Deep Learning

❖ Deep Learning

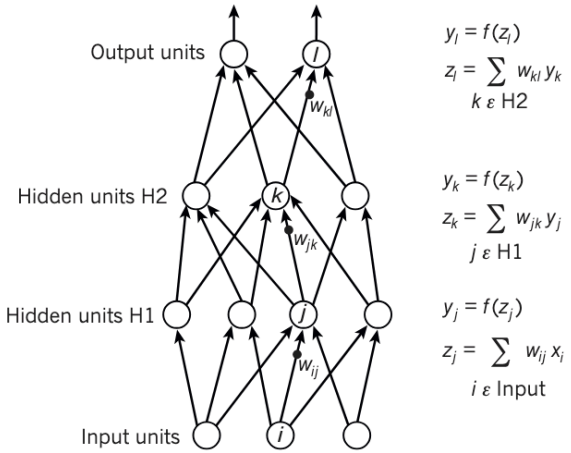
- Automatically learn complex patterns from data by general-purpose learning procedure
- Each layer abstracts representations, from low-level features to high-level concept
- Key components of deep learning
  - Forward Propagation, Activation, Back-Propagation, Optimization



Fundamentals of Deep Learning

❖ Forward Propagation

- Set of units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear function
- Weighted sum filters and combines input signals, enabling higher layers to learn abstract patterns



$y_l = f(z_l)$

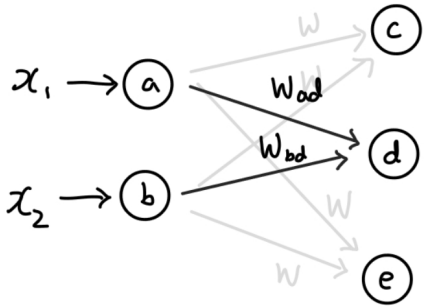
$z_l = \sum_{k \,\varepsilon\, H2} w_{kl}\, y_k$

$y_k = f(z_k)$

$z_k = \sum_{j \,\varepsilon\, H1} w_{jk}\, y_j$

$y_j = f(z_j)$

$z_j = \sum_{i \,\varepsilon\, Input} w_{ij}\, x_i$



$$Z_d = W_{ad}x_1 + b_a + W_{bd}x_2 + b_b$$

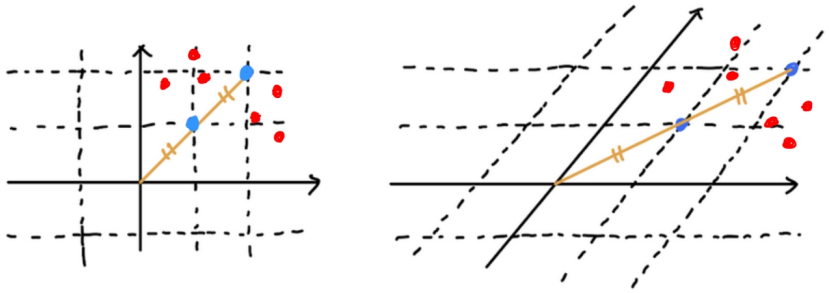
$$y_d = f(Z_d)$$

Fundamentals of Deep Learning

- **Why is non-linear activation function needed?**
  - **No matter how many times linear transformation is applied, they remain linear**
  - **Cannot learn complex patterns**

$$\mathbf{y} = W_4^T\left(W_3^T\left(W_2^T\left(W_1^T\,\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3\right) + \mathbf{b}_4$$

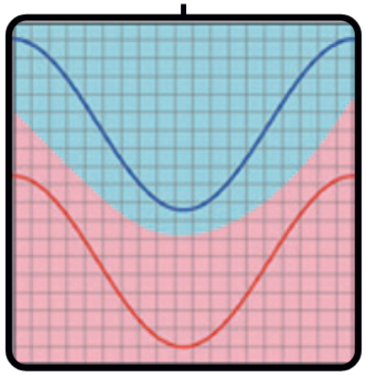
$$\mathbf{y} = W^T\left(\mathbf{x} + \mathbf{b}\right)$$

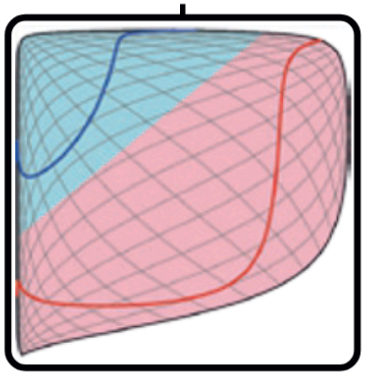


Fundamentals of Deep Learning

❖ Activation Functions

- Adds non-linearity to neural networks
- Non-linearity enables neural networks to learn more complex patterns
- Through repeated application, it enables linear separability



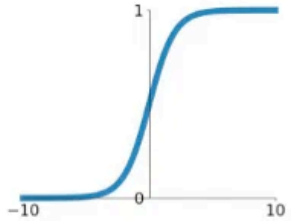


Fundamentals of Deep Learning

- **Common activation functions**
  - **Sigmoid Function**
    - **Output range between 0 and 1**
    - **Outputs are directly interpretable as probabilities**
    - **Vanishing gradient problem: hard to update early layers**

Sigmoid

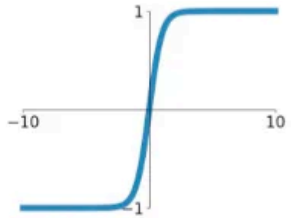
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



  - **Hyperbolic Tangent Function**
    - **Output range between -1 and 1**
    - **Outputs centered around zero ables faster convergence**
    - **Stronger gradient but still has vanishing gradient problem**

tanh

$$\tanh(x)$$



Fundamentals of Deep Learning

- **Vanishing gradient problem**
  - **By the time they reach early layers, gradients become so tiny they're practically zero**
  - **Range of gradient in Sigmoid is (0, 0.25), Range of gradient in tanh is (0, 1)**

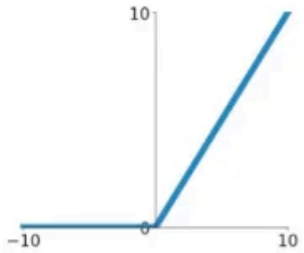
$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y}\underbrace{\frac{\partial y}{\partial z_2}}_{\sigma}\underbrace{\frac{\partial z_2}{\partial W_2}}_{a_1}, \qquad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y}\underbrace{\frac{\partial y}{\partial z_2}}_{\sigma}\underbrace{\frac{\partial z_2}{\partial a_1}}_{W_2}\underbrace{\frac{\partial a_1}{\partial z_1}}_{\sigma}\underbrace{\frac{\partial z_1}{\partial W_1}}_{x\,(\text{input})}$$

Fundamentals of Deep Learning

- **Common activation functions**
  - **Rectified Linear Unit Function**
    - **Maps all negative values in the input with zero**
    - **Fast and simple calculation, Free from the vanishing gradient problem**
    - **Dying ReLU problem: neurons die when the input is negative**

ReLU

$$\max(0, x)$$



Fundamentals of Deep Learning

❖ Dying ReLU problem

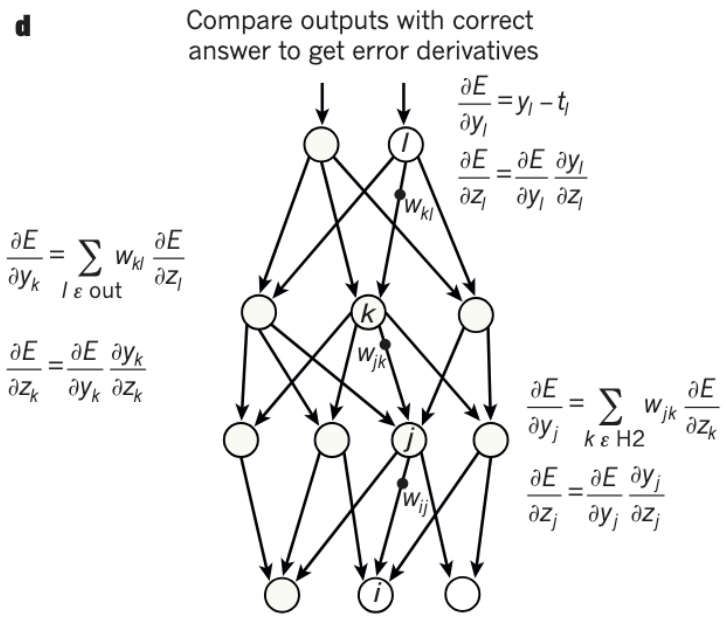
- ReLU's gradient is zero when the input is negative,
  so the gradients cannot be back-propagated to those neurons
- Sigmoid and tanh have gradients greater than zero, so they affect the entire model,
  but ReLU's gradient is zero, so it affects only specific neurons

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y}\underbrace{\frac{\partial y}{\partial z_2}}_{\sigma}\underbrace{\frac{\partial z_2}{\partial W_2}}_{a_1}, \qquad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y}\underbrace{\frac{\partial y}{\partial z_2}}_{\sigma}\underbrace{\frac{\partial z_2}{\partial a_1}}_{W_2}\underbrace{\frac{\partial a_1}{\partial z_1}}_{\sigma}\underbrace{\frac{\partial z_1}{\partial W_1}}_{x\,(\text{input})}$$

Fundamentals of Deep Learning

❖ Back-Propagation

- Feed the final output into a loss function to compute the error(loss)
- Propagate the gradient that minimizes this loss backward to every parameter, and the parameters are updated using that gradient



Fundamentals of Deep Learning

❖ Back-Propagation

- Using chain rule to compute the parameter's gradient from the gradient of the loss function

$$y = \sigma\big(W_2\,\sigma(W_1 x + b_1) + b_2\big)$$

$$z_1 = W_1 x + b_1, \quad a_1 = \sigma(z_1), \quad z_2 = W_2 a_1 + b_2, \quad y = \sigma(z_2)$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y}\underbrace{\frac{\partial y}{\partial z_2}}_{\text{Relu}}\underbrace{\frac{\partial z_2}{\partial W_2}}_{a_1}, \qquad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y}\underbrace{\frac{\partial y}{\partial z_2}}_{\text{Relu}}\underbrace{\frac{\partial z_2}{\partial a_1}}_{W_2}\underbrace{\frac{\partial a_1}{\partial z_1}}_{\text{Relu}}\underbrace{\frac{\partial z_1}{\partial W_1}}_{x\,(\text{input})}$$

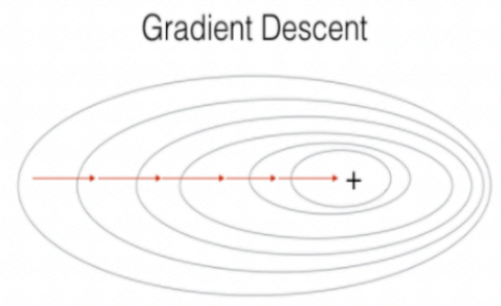
Fundamentals of Deep Learning

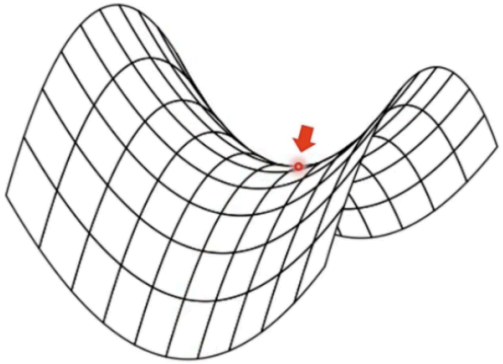
❖ Optimization

- Optimize the trainable parameters of neural networks using the gradient descent algorithm

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

- Full-Batch Gradient Descent
  - Updates parameters by computing the exact gradient over the entire dataset
  - Becomes very slow as the dataset size increases
  - Cannot escape saddle points because the gradient is zero there

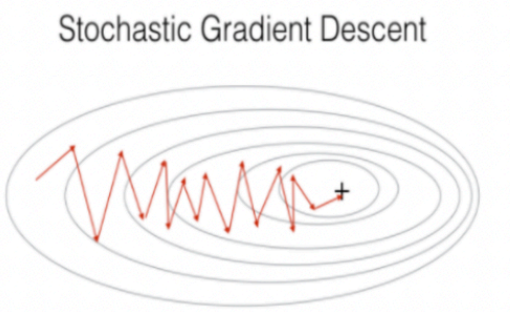


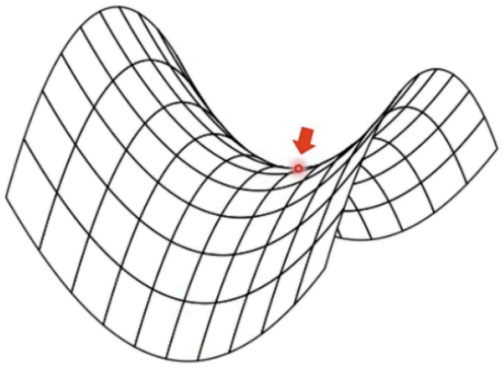


Fundamentals of Deep Learning

❖ Stochastic Gradient Descent

- Updates parameters by computing the approximate gradient from a subset of data
- Gradient is estimated using mini-batches, introducing a small amount of noise that helps escape saddle points and local minima and improving generalization performance





Fundamentals of Deep Learning

- **Is this Fully Connected Neural Network effective on all types of data?**
  - **Images**
    - **Processing images requires a lot of weight parameters**
    - **Images are not arbitrary vectors of pixels, they exhibit locality**
  - **Sequential Data**
    - **Not suitable for variable length data**
    - **“I love you” ≠ “love you I”**

➡ **To overcome these limitations, variant models such as CNNs and RNNs were developed**

Convolutional Neural Networks

What is Convolutional Neural Networks?

- Structure designed to process data that come in the form of multiple arrays like images

Characteristics of Images

- Locality: Images are not arbitrary vectors of pixels
- Translation invariance: Same patterns or features can appear anywhere in an image
- Compositional Hierarchies
  - Edges form motifs, motifs assemble into parts, and parts to objects

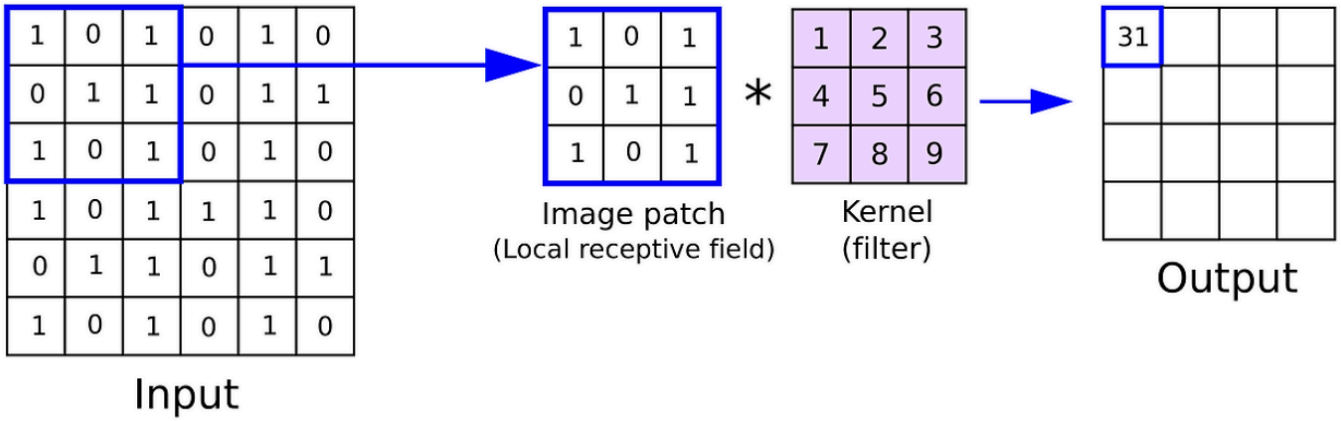
➡ How can Convolutional Neural Networks capture all these characteristics?

Convolutional Neural Networks

❖ Convolution Layer

□ Convolution in CNN

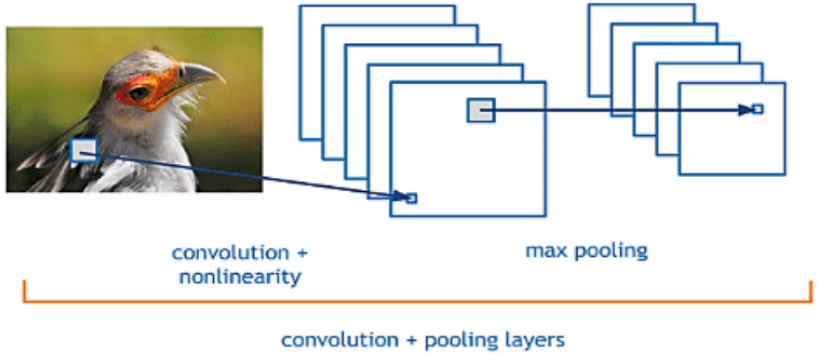
- Multiplication and summing operation used in image processing
- Applying a filter to a local patch resulting local weighted sum

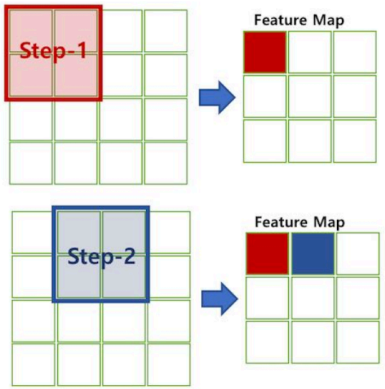


Convolutional Neural Networks

❖ Convolution Layer

- □ **Shared Weight**
  - **One filter slides across the whole image**
  - **Also reduces number of parameters and can be processed in parallel**
  - **Provides translation invariance to the model**

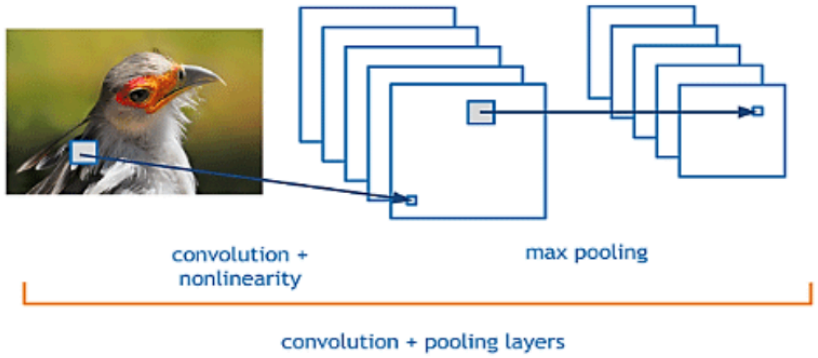


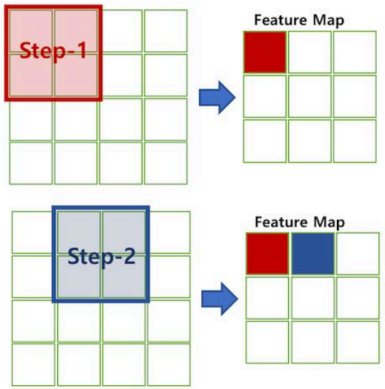


Convolutional Neural Networks

❖ Convolution Layer

- □ **Sparsity of Connections**
  - **Each neuron sees only a small local patch in the feature maps of the previous layer**
  - **Provides locality to the model**

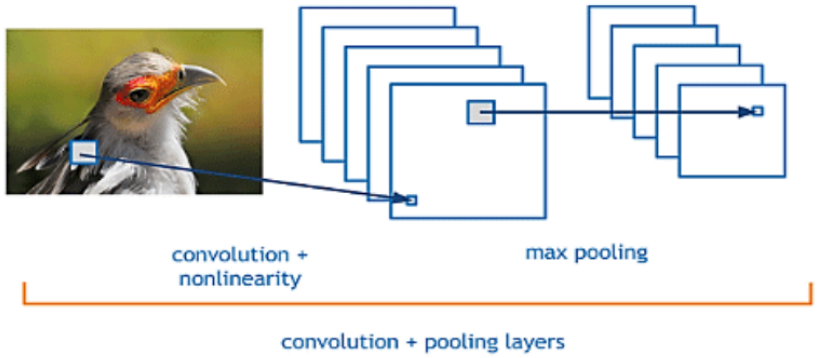


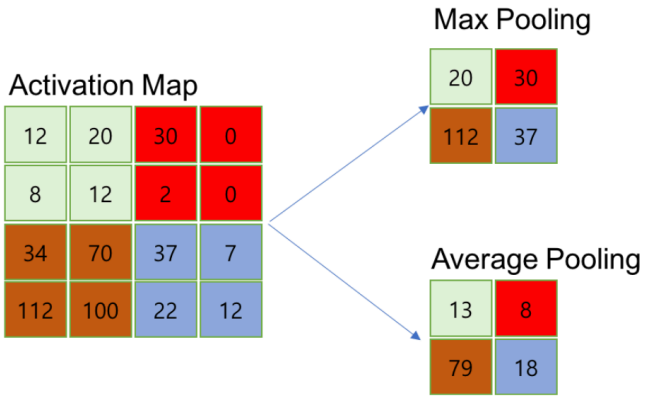


Convolutional Neural Networks

❖ Pooling Layer

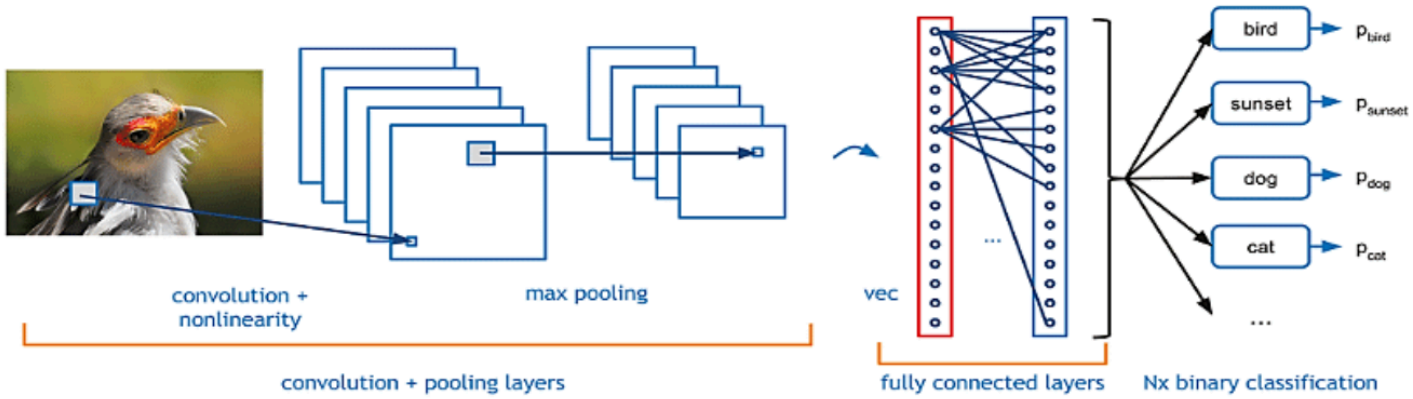
- Merges semantically similar features into one
- Tells if a feature was present in the previous layer, but not precisely where
- Provides translation invariance to the model





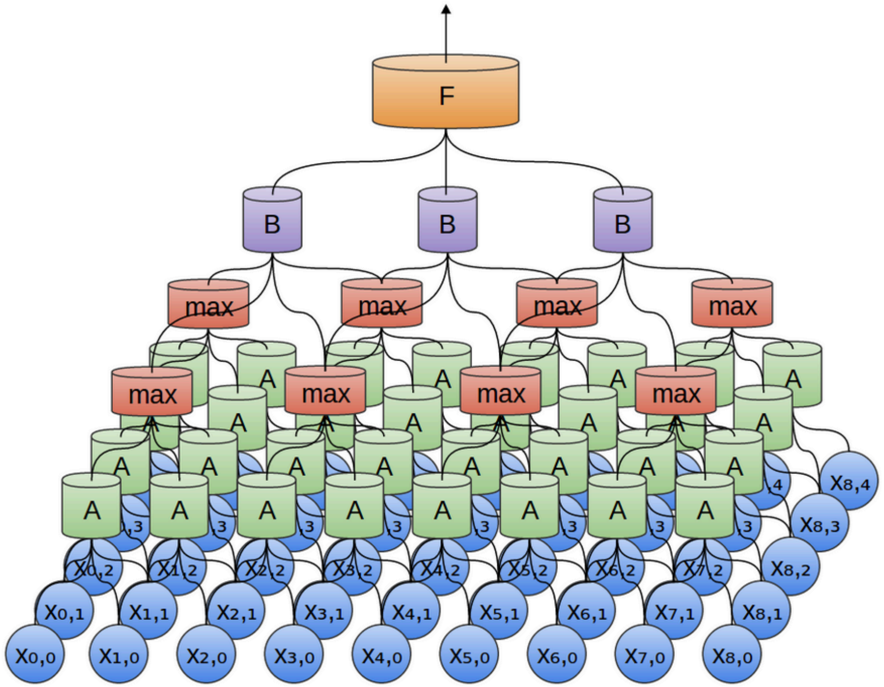
Convolutional Neural Networks

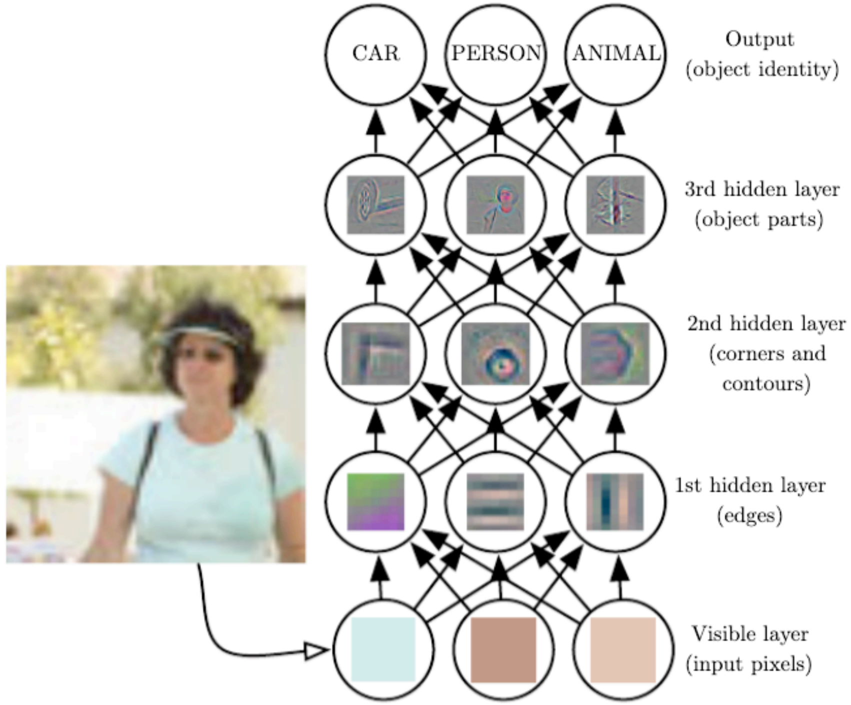
- **Stacking Convolution and Pooling Layers**
  - Stacking layers naturally composes features from simple features to complex features
  - Lower layers: small receptive fields detect edges & corners
  - Higher layers: large receptive fields combine & abstract features



Convolutional Neural Networks

❖ Structure of Convolutional Neural Networks

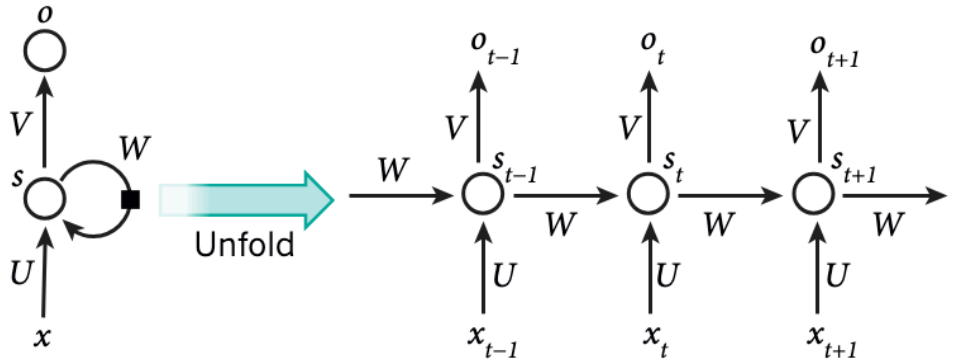




Recurrent Neural Network

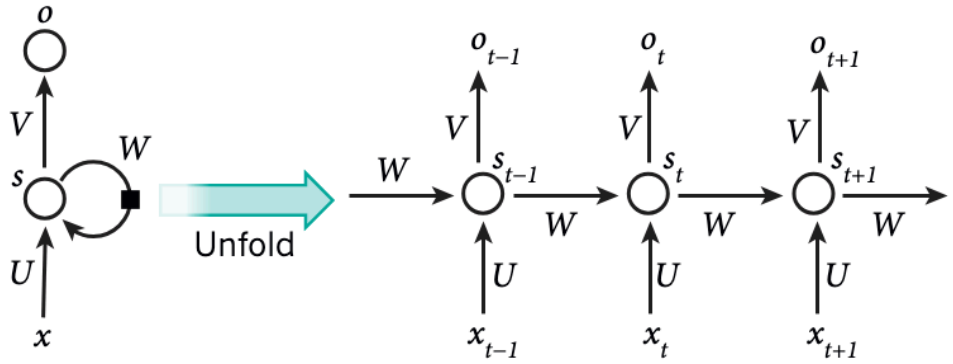
What is Recurrent Neural Network?

- A neural network that can handle various size of sequential data
- Reuses the same parameters at every time step
  - Like a CNN filter scanning every location,
    shared weights let RNNs spot patterns at any time step.
- Has vanishing/exploding gradient problem



Recurrent Neural Network

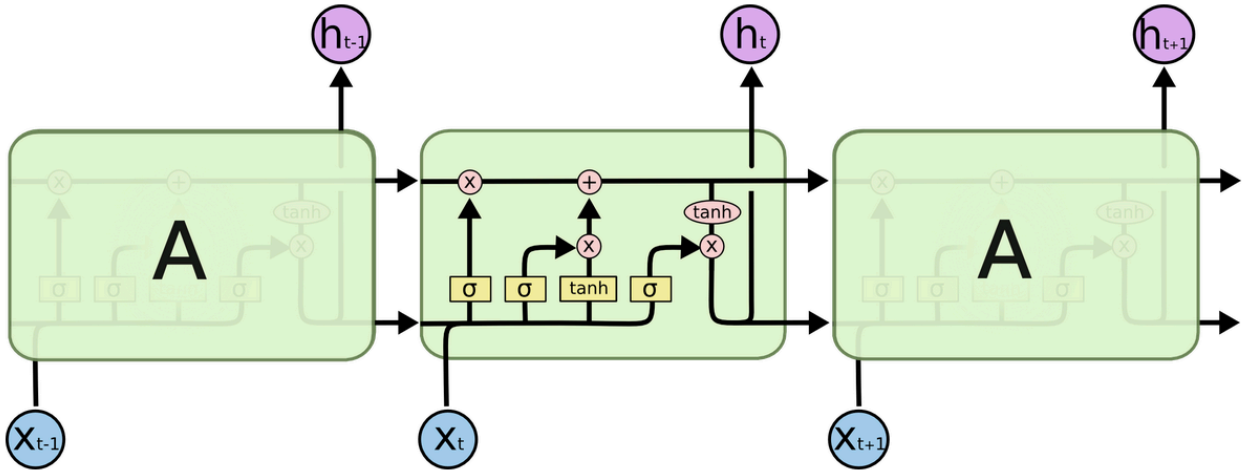
- **Vanishing/Exploding gradient problem in RNN**
  - **In backpropagation, the weight matrix W is multiplied repeatedly**
    - **if |W|>1, gradients explode; i|W|<1, they vanish**
  - **Difficult to learn long-term dependencies**



Recurrent Neural Network

❖ LSTM

- Variant of RNN model designed to learn long-term dependencies by using gates
- The cell state c_t follows an additive path that lets gradients flow directly
- Gates merely control what information enters or leaves it



Recurrent Neural Network

- **Benefits of LSTM**
  - Less gradient vanishing → stable training even on long sequences

- **Drawbacks of LSTM**
  - Information is added through gated sigmoids, whose slight attenuation still limits the network to learn extremely long-term dependencies

Batch and Layer Normalization

- More ways to handle vanishing/exploding gradient
  - Gradient stabilization seen earlier stabilize the values that are multiplied repeatedly
  - How about stabilizing the input of the each layer?

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial W_2} \qquad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial W_1}$$

Relu a_1 Relu W_2 Relu x(input)

Batch and Layer Normalization

- **Covariate Shift**
  - Change in distribution of inputs entering a layer
  - When the input distribution shifts, parameters have to readjust for new distribution
  - External Covariate Shift
    - Input distribution differs between the training phase and the prediction (testing) phase

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial W_2} \qquad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial W_1}$$

Relu a_1 Relu W_2 Relu x(input)

Batch and Layer Normalization

❖ Internal Covariate Shift

- Distribution of inputs to each hidden layer due to the change in network parameters
- Require lower learning rates and careful parameter initialization

➡ **Stabilize input distribution by Batch and Layer Normalization**

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial W_2} \qquad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial W_1}$$

Relu a_1 Relu W_2 Relu x(input)

Features of Batch Normalization

❖ Batch Normalization

- Fix the means and variances of layer inputs using mini-batch
- Two necessary simplifications
  - Normalize each scalar feature independently

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

  - Simply normalizing each input of layer may change what the layer can represent
    - ➡ Learnable parameters are inserted in each activation

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Features of Batch Normalization

- **Batch Normalization**
  - Fix the means and variances of layer inputs using mini-batch
  - Two necessary simplifications
    - Each mini-batch produces estimates of the mean and variance of each activation

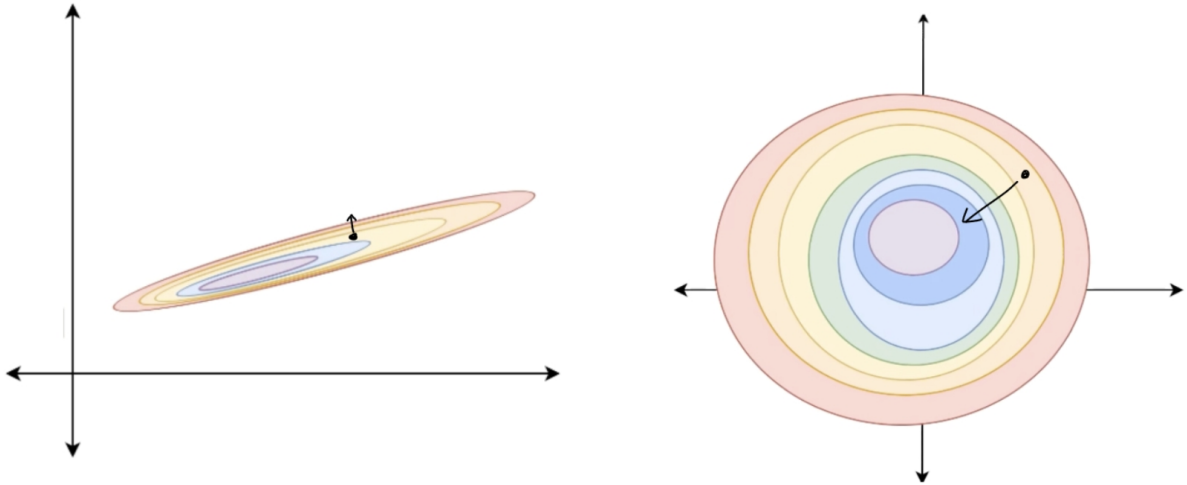
Features of Batch Normalization

- **Batch Normalization after training**
  - **In the inference phase, batch normalization uses the running mean/variance computed during training that estimates the statistics of the entire dataset**
  - **Enables reliable normalization even with small batch sizes**

Features of Batch Normalization

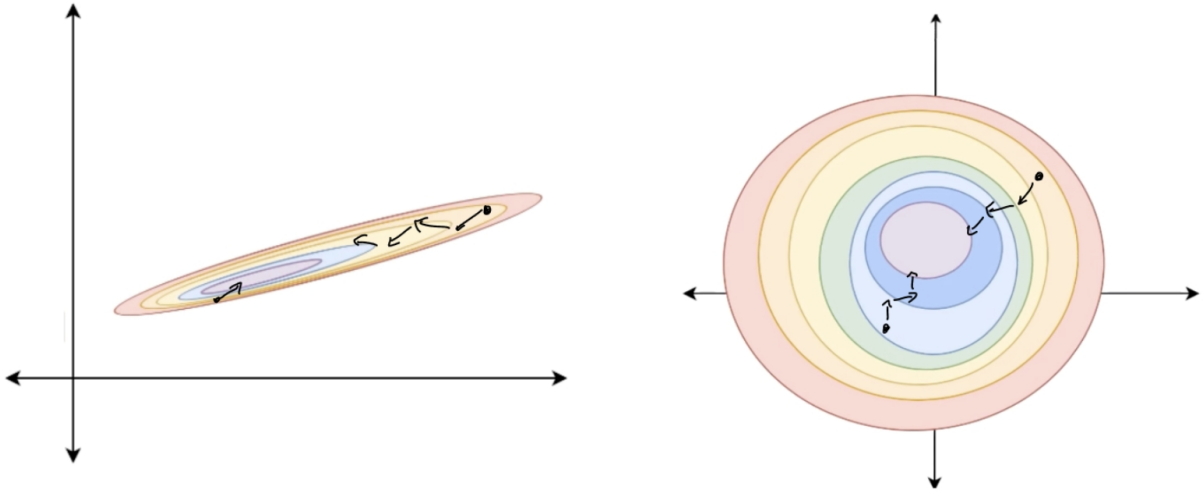
❖ Benefits of Batch Normalization

- Speeds up training
  - Optimization landscape significantly smoother
  - Smoothness induces a more predictive and stable behavior of the gradients, allowing for larger learning rate



Features of Batch Normalization

- **Benefits of Batch Normalization**
  - **Reduces sensitivity to initialization**

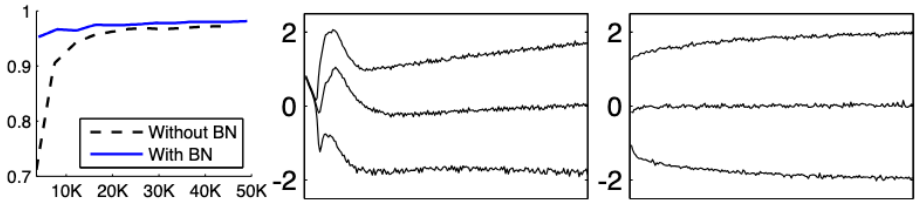


Features of Batch Normalization

- **Benefits of Batch Normalization**
  - **Regularizes the model**
    - **Non-deterministic outputs can little yield different results for the same sample**
    - **Slight randomness helps prevent overfitting and improves generalization**

Experiments on Batch Normalization

- **MNIST dataset with MLP**



(a) (b) Without BN (c) With BN

Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*

Experiments on Batch Normalization

- **ImageNet classification with CNN**

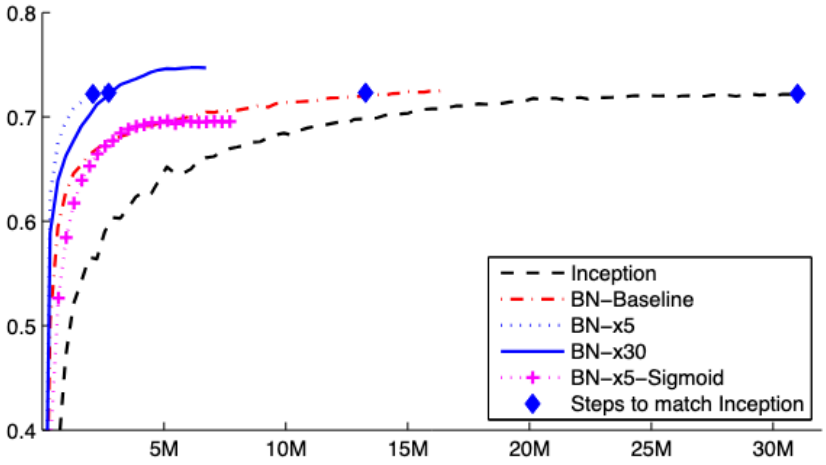


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Figure 3: *For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.*

Features of Batch Normalization

❖ Drawbacks of Batch Normalization

- Dependent on the mini-batch size

- Hard to apply it to recurrent neural networks
  - Distribution of the hidden state varies at each time step, needs separate parameters for every step

- Procedures used during training and inference differ
  - External covariate shift can occur

Features of Layer Normalization

❖ Layer Normalization

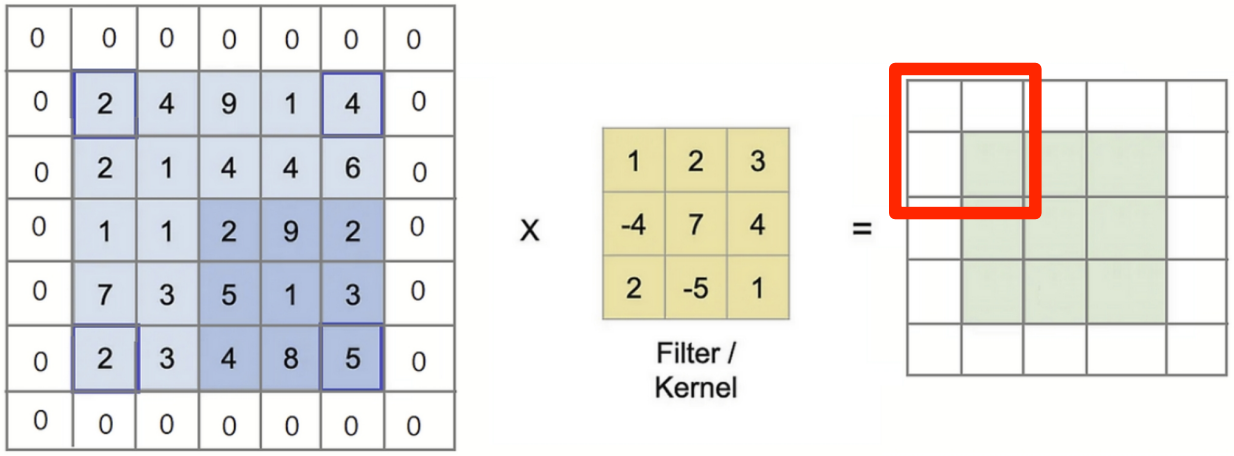
- Similar to batch normalization: each neuron has its own adaptive bias and gain
- Computes normalization statistics from inputs

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$

- No dependency on batch size
- Requires computation of mean and variance in each input, adding extra computation

Batch Normalization vs. Layer Normalization

- **Why Layer Normalization is bad in CNNs**
  - **Padding makes edge pixels behave differently, distorting the layer's statistics**



Batch Normalization vs. Layer Normalization

- **Why Batch Normalization is bad in RNNs**
  - **Distribution in the sequence varies at each time step**
  - **When size of sequence varies, padding introduces distributional distortion**

1025	1105	1185	1265		
1505	1585	1665	1745	1825	1905
1985	2065	2145	2225	2305	

Experiments on Layer Normalization

- **Comparison with Batch normalization in lstm**

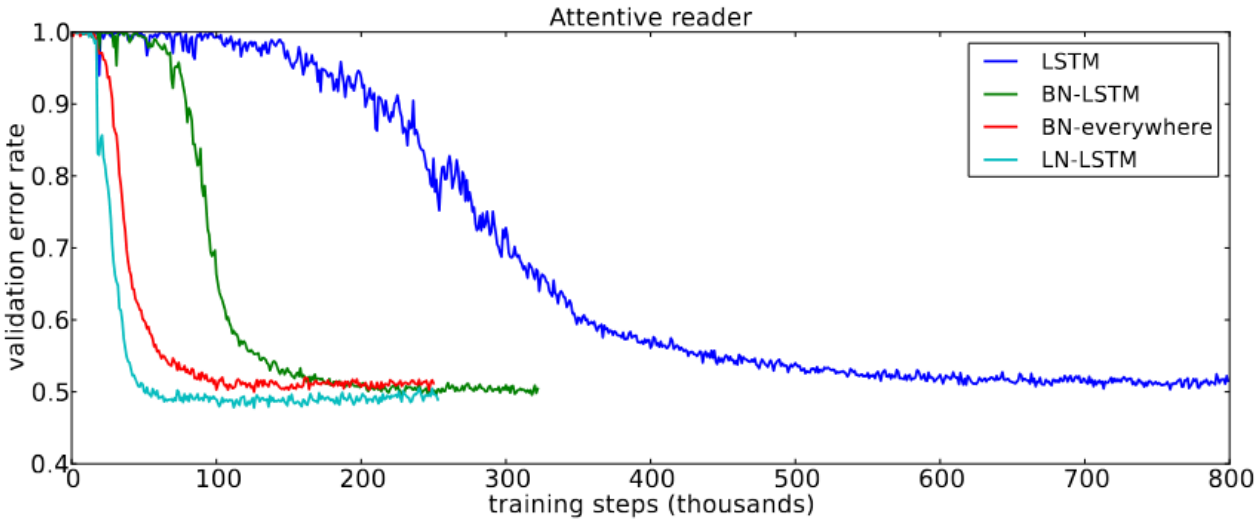


Figure 2: Validation curves for the attentive reader model. BN results are taken from [Cooijmans et al., 2016].

Thank you!