STAT 526

# Topic X
# First Look of Deep Learning

Dr. Qifan Song

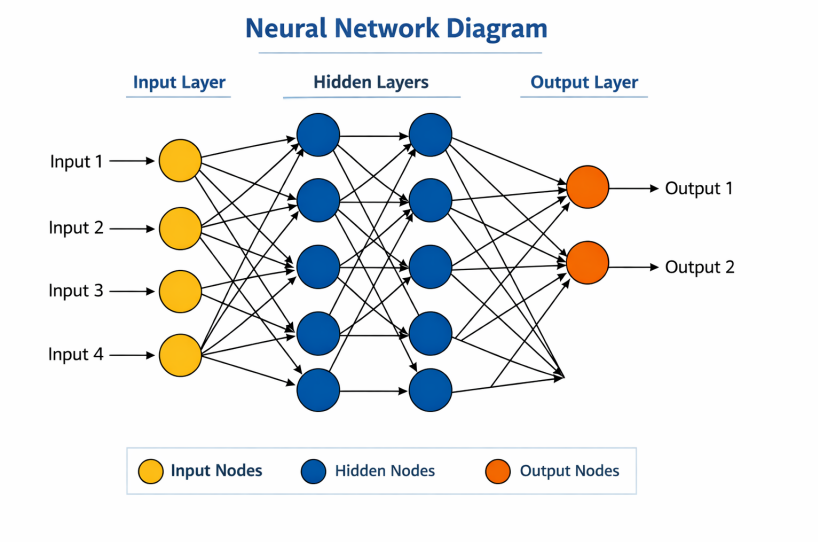
# Artificial Neural Network

- Date back to 1958, Perceptron model: 3 layer network with random connection, partially untrained weight
- Multi-Layer Perceptron ($L+1$ layer structure)

$$y = \sigma_1 \circ W_1 \circ \sigma_2 \ldots\ldots \circ \sigma_L \circ W_L x$$

where $W_l \in \mathbb{R}^{d_l \times d_{l+1}}$ denotes linear mapping with this matrix, $d_1 = d_y$, $d_{L+1} = d_x$. $\sigma_l$ denotes (entrywise) activation.

- Can add a "bias" node (intercept of the linear mapping); or one can absorb the bias in the activation function



# MLP

- The activation is nonlinear (unless in theoretical studies). It grants the model the capacity to fit complex relationship between input $x$ and output $y$.
- Common choice of activation function in modern DNNs.

  Sigmoid/Tanh: $\sigma(x) = e^x/(1+e^x)$, or $\tanh(x)$

  ReLu: $\sigma(x) = \max(0, x)$

  Leaky ReLu: $\sigma(x) = \max(\alpha x, x)$ (resolve the issue of no gradient information from unactivated neuron)

  ELU: $\sigma(x) = x1(x > 0) + \alpha(e^x - 1)1(x \leq 0)$

  GELU: $\sigma(x) = x\Phi(x)$

  Soft-max: $\sigma(x_i) = e^{x_i}/\sum_j e^{x_j}$ converts any vector to a probability PMF.

# Universal Approximation Theory

- The theoretical results associated with the approximation capabilities of NNs.
- Applied mathematical results
- A general statement: for any continous/$L_p$ integrable function $f$ and $\epsilon$, there exist a NN $f_{NN}$, such that the difference between $f_{NN}$ and $f$ (under $L_p$ or $L\infty$ norm) is $< \epsilon$.
- As $\epsilon \to 0$, it usually requires an increasing depth or width.
- Proof technique varies
  - Construction-based: Design a small NN that can approximate some basic function, then stack many small NNs together to approximate any arbitrary function
  - Functional analysis (Hahn-Banach): Assume the network class is not dense in $\mathcal{C}$, Then by the Hahn–Banach theorem, there exists a nonzero continuous linear functional $L \in \mathcal{C}^*$ such that: $L(f_{NN}) = 0$ for all neural network functions, which is impossible.

# Universal Approximation Theory

- Other type of UAT studies
  - Minimum width requirement
  - Sparse connection, i.e., allows many zero in $W$'s
  - Shared weights, entries in $W$ can only take some specific values
  - Bounded depth and bounded width, but arbitaray activation function
- Special result of ReLu: any MLP ReLu network is a continuous piecewise linear function. Any continuous piecewise linear function can be represented by a MLP ReLu network (`https://arxiv.org/pdf/1611.01491`)
- Compared with nonparametric regression, NNs are more flexible and doesn't require tuning on smoothness parameter

# Supervised Task with Deep Neural Network

- Regression Task: Fit model $E(y) = f_{NN}(x)$ with Least Square Error objective

$$l = \sum_i \|y_i - f_{NN}(x_i)\|^2$$

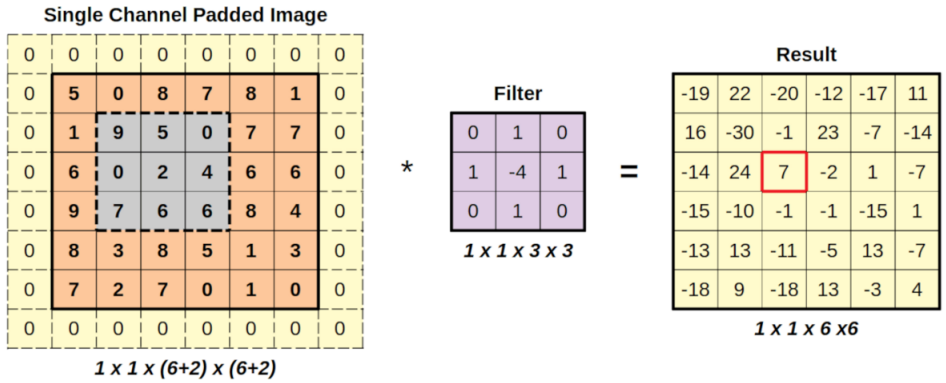
- Classification Task: Fit model $\{P(y = j)\}_{j=1}^m = f_{NN}(x)$ with $\sigma_1$ being the softmax function. Cross Entropy loss (negative log-likelihood) is used

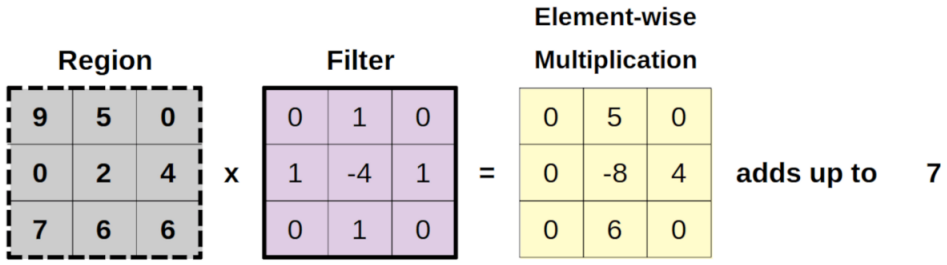
$$l = -\sum_i y_i^T \log(f_{NN}(x)).$$

  In the above equation, $y_i$ represents a one-hot vector of the true label (e.g., $y = (0, 0, 1, 0, 0)$), log applies entrywisely to $f_{NN}(x)$

# Convolutional Neural Network (CNN)

- Feedforward network that learns features via filter (or kernel).
- Convolutional Layer (concepts of kernel, padding, stride)
- Below: 1 kernel/filter, padding = 1, stride = 1




credit: Wikipedia.org

# CNN

- Convolutional Layer

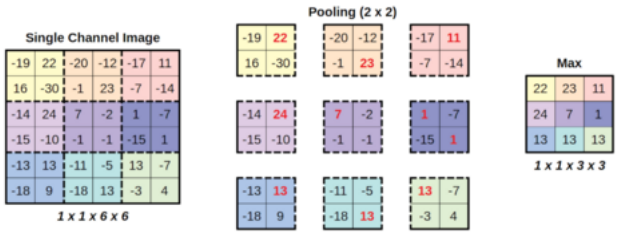
  Learns the local spatial relationship; good for spatial data

  Can add a bias for the kernel;

  Both input and kernel can be high dimensional tensor. For colored image(s), the dimension is 3 for one image or 4 for a batch of image; a kernel with 2 dimension will apply to each channel of the input, a kernel with 3 dimension means it will aggregrate multiple channel in the input.

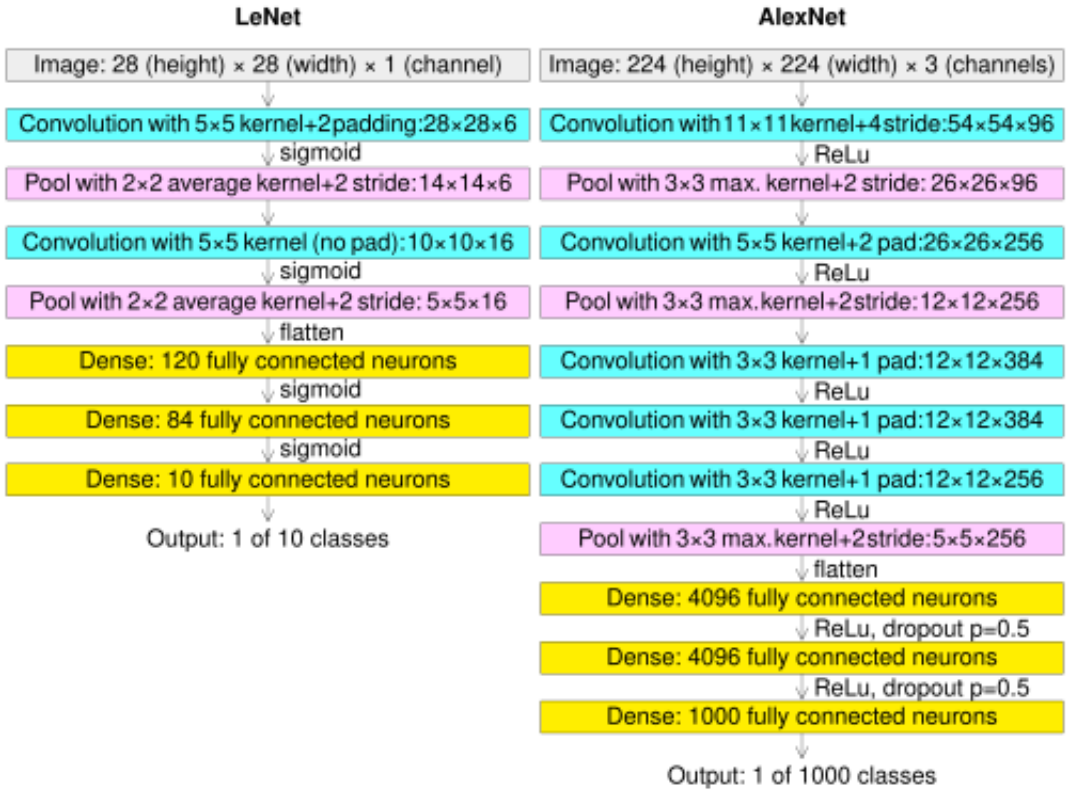
  Can have multiple kernels.

- Pooling Layer: downsize the feature by making max or average over a fixed window. $2 \times 2$ Max pooling, stride 2



# CNN

- CNN networks usually consist of block series of (convolutional Layers + Activation + Pooling Layer), following by MLP.
- Convolutional layer is feed forward layer with sparse connection and shared weights.
- 3-d tensor with depth inferred from input dim (`nn.LazyConv2d`)

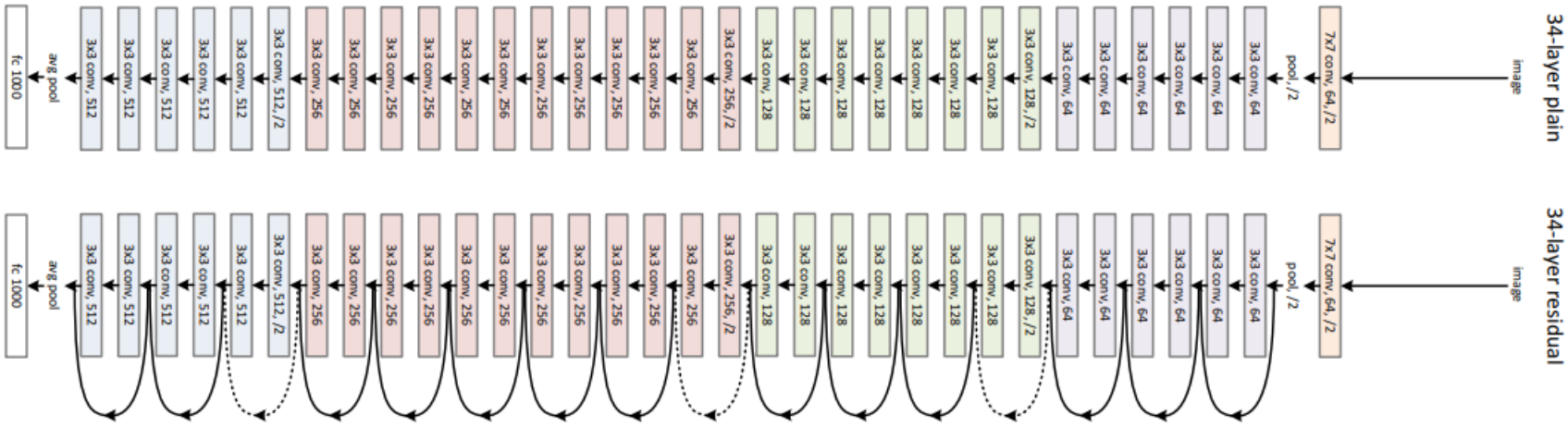


# Residual Neural Network (ResNet)

- A plain deeper network suffers from perfromance degradation (not overfitting, but worse training performance)
- But even adding identical mapping can make the NN deeper with identical performance
- It suggest that in a deep NN, even identity mapping cannot be well approximation
- A ResNet contains additional skip connections in its architecture, i.e., $x \to H(x) = x + F(x)$ where $f$ denotes a 2-/3-layer network block module
- Instead of learning $H$, we do "residual learning" for $H(x) - x$

# Residual Neural Network (ResNet)

- If $F(x)$ and $x$ have different dimension, we can either adding padding to $x$ (increase dim) or apply a conv. kernel to $x$ (decrease dim)
- A Common understanding is the ResNet help the training and optimization on deep networks
- A plain deep network suffers from vanishing/exploding gradient issue, skip connection establish a short cut in backprop (Later lecture will discussion optimization about deep learning).

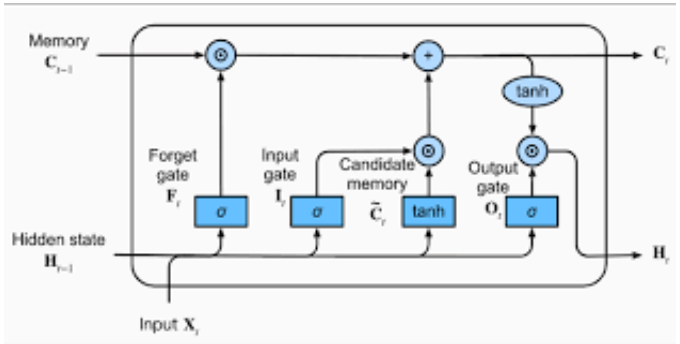


# Recurrent neural network (RNN)

- Recurrent NNs are designed to process sequential data in a sequential manner (e.g., text with indefinite length);
- When RNN processes the current data piece, it can use the output of previous processing process.
- A typical (many-to-many) RNN: $h_t = f_{NN}(h_{t-1}, x_t)$, $y_t = f'_{NN}(h_t)$, where $h_t$ is the hidden state (summarizing past information), and $h_0$ is random initialized. Input data: $x_1, \ldots, x_t, \ldots$. Output data: $y_1, \ldots, y_t, \ldots$
- Many-to-one RNN: $h_t = f_{NN}(h_{t-1}, x_t)$, $y_T = f'_{NN}(h_T)$. Input: $x_1, \ldots, x_T$. Output data: $y_T$
- One-to-many RNN: $h_1 = f_{NN}(h_0, x_1)$, $(y_t, h_t) = f'_{NN}(h_t, y_{t-1})$, Input: $x_1$. Output data: $y_1, \ldots, y_t, \ldots$

# Long short-term memory (LSTM)

- Similar to deep network, a RNN with a larger time sequence suffers from gradient vanishing/exploding, resulting unstable training.
- LSTM models can store short-term memory for a long time
- Besides hidden state $h_t$, an additional memory state $c_h$
- 3 "gates" : forget gate – should the model forget the previous memory; input gate – should the model add information from current step to the memory; output gate – how the memory output information to hidden state.



# Bidirectional recurrent neural networks

- An extension of RNNs; processes sequential data in both forward and backward directions
- The model utilizes both past and future context when making predictions
- $h_t = f_{NN}(h_{t-1}, x_t)$, $\tilde{h}_t = f'_{NN}(\tilde{h}_{t+1}, x_t)$, $y_t = f''_{NN}(h_t, \tilde{h}_t)$, where $h_t$ and $\tilde{h}_t$ are forward/backward hidden states.
- Need to run the sequence twice, one in forward and one in backward direction.

# Graph neural network

- Graph neural networks (GNNs) are design to learn graph-related task ($\alpha$-fold).
- A graph: $G = (V, E)$, $V$ is the vertex set including all nodes, $E$ is the edge set including all edges (e.g., $e = (u, v)$)
- $G$ can be represented by its adjacency matrix $A \in \{0, 1\}^{|V| \times |V|}$. $a_{ij} = 0$ means an edge from the node $i$ to node $j$.
- All vertices and edges associated with a feature vector, i.e., $x_u$ for $u \in V$ and $e_{uv}$ for $e = (u, v) \in E$. We can present the features can be organized as a matrix (i.e., each row of $X$ means a node).
- A GNN takes adj matrix and feature matrix as input. For every layer of a GNN, graph in, graph out. When the graph becomes one node, the output is a vector. Follows by MLP.
- A GNN must be permutation invariant (average, max etc).

# Graph neural network

- Message passing layers: for each node $u$, it output $\tilde{x}_u = f_{NN}(x_u, g_{v \in N_u}[\phi_{NN}(x_u, x_v, e_{uv})])$ where $N_v$ is the neighborhood set of $u$, determined by $A$, $g$ is permutation invariant operation

  MPNN updates node representations by aggregating feature information from local neighborhoods. $\phi_{NN}$ computes the message, $g$ aggregates all messages, $f_{NN}$ updates node embedding.

- Pooling layer: reduce graph size by selecting the most informative
  - Top-k pooling: keep the top-k vertices by $Xp$, where $p$ is learnable parameter.
  - global pooling: reduce $X$ to one row by max/average etc

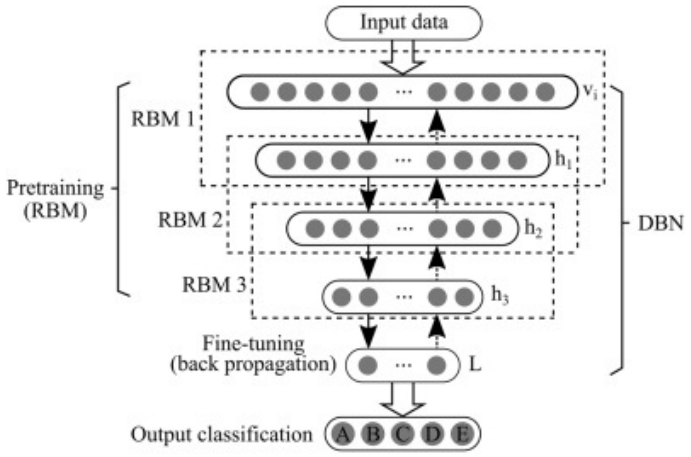
- A visualization at `https://visual-intelligence-umn.github.io/GNN-101/`

# Restricted Boltzmann machine

- Restricted Boltzmann machine (RBM): a layer of visible nodes (observed r.v. $v$) and a layer of hidden nodes (hidden r.v. $h$).
- The joint distribution of $h$ and $v$ is proportional to $-\exp\{a^T v + b^T h + v^T W h\}$ for binary $v$ and $h$.
- MLE; require special training strategy (Contrastive Divergence) since the marginal distribution of $v$ is not tractable
- $v|h$ and $h|v$ follows logistic regression (HW). Reconstruction $v \to h \to v$ uses the same weight matrix $W$

# Deep Belief Network

- A deep belief network (DBN) consists of multiple layers of latent variables. Can be viewed as stacked RBMs
- Unsupervised/pre-training model (for a generative model and data reconstruction, feature extraction purpose): Bi-directed network; Layer-wise Contrastive Divergence Training
- Supervised/fine-tuning model (for classification/regression purpose): Add MLP on the deepest hidden layer; using latent representation for supervised tasks



# Auto-Encoder

- An encoder NN $E$ that maps input $x$ to low dimensional feature $h$
- An decoder NN $D$ that maps feature $h$ to sample space $x$
- Trains model by minimizing the reconstruction error $d(x, D(E(x)))$
- Applications
  - Compression (e.g. video codecs)
  - Data generation
  - Feature extraction, dimension reduction
  - Outlier/anomaly detection
  - Denoising

# seq2seq model

- A seq2seq consists of a encoder RNN and a decoder RNN; applications: translation, Q&A, etc
- The encoder RNN is many-to-one, i.e., input sequence to a hidden summary; the decoder RNN is one-to-many, i.e., a hidden summary to sequence of output
- A seq2seq model defines a probability law of $\{y_1, \ldots, y_t, \ldots\}$ given input sequence $x_1, \ldots, x_T$. The training loss is the neg loglikelihood of $\{y_1, \ldots, y_{T'}\}$ conditional on $x_1, \ldots, x_T$, for data set $\{x_1, \ldots, x_T, y_1, \ldots, y_{T'}\}$
- Modern language models uses transformer/attention head to build RNN blocks (to be taught in later lecture).

# Normalization

Normalization is a de facto standard in deep learning applications. Theoretical understand is limited, but empirically proven to be very effective.

- Input normalization: normalize $x \in \mathbb{R}^p \Rightarrow [0,1]^P$ or $[-,1]^P$. Many theories also require a compact space.

- Batch normalization layer: Normalize input by subtracting population mean and dividing populuation s.d.. Since Deep Learning are always executed in batchs, batch mean and s.d. are used instead.

  $$\tilde{x}_i = (x_i - \hat{\mu}_i)/\sqrt{\hat{\sigma}_i^2 + \epsilon},$$

  for all $i = 1, \ldots, p$ and the $\epsilon$ is to avoid numerical instability.

  BatchNorm can be interpreted as removing the purely linear effect in previous layers, so that later layers focus solely on modelling the nonlinear aspects of data

# Normalization

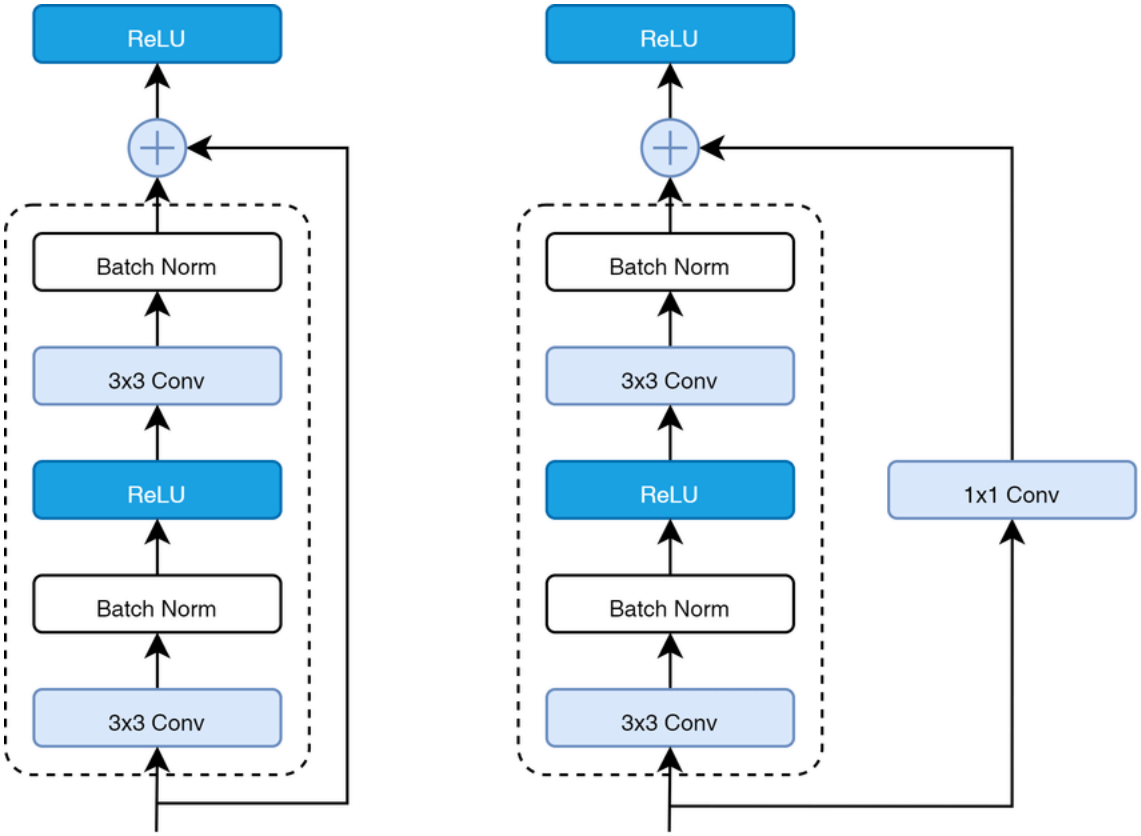
During training, for both forward path (compute model output output) and backward path (compute gradient), the computation is no longer independent across samples

During testing (inference stage), a fixed global mean and variance, typically calculated as a moving average of the statistics observed during the entire training phase, is used

- BatchNorm normalizes across the batch for each dimension. Layer Normalization normalizes across all the dimension within a single data sample (e.g., a 3-D tensor in image task).
- Root mean square layer normalization (RMSNorm) normalizes each single data into a unit $L_2$ vector
- For both Batch/Layer Normalization, additional linear transform is conducted after normalization, $\widehat{x}_i = \tilde{x}_i * \gamma_i + \beta_i$, where $\gamma_i$ and $\beta_i$ are learnable.

# Normalization

- BatchNorm and LayerNorm stabilize neural network training by adjusting data distributions
- BatchNorm is preferred for CNNs (computer vision) and large batches, while LayerNorm is more popular for Transformers and RNNs (NLP task) due to its batch-size independence.



# Regularization

- Generalization thoery claims that generalization MAY be big given a very complex model, e.g., a deep NN.
- Regularization is a common techique. It regularizes the training/optimization of the mode, such that only a subspace of the parameter space is explored during the optimization.
- Regularization usually brings in a bias, but stablizes the results
  - $L_1$ regularization: add a $L_1$ penalty (LASSO); encourage model sparsity
  - $L_2$ regularization: add a $L_2$ penalty (ridge regression); equivalent to weight decay for gradient descent optmizer
  - Earlier Stopping: avoid overfitting/data memorization
  - Implicit regularization due to gradient descent (to be discussed later)

# Dropout

- Reduce model size by randomly dropping out hidden/input nodes. Avoid overfitting.
- During each iteration of training, hidden (input) nodes are discarded from network (i.e., output from them are fixed to be 0) with probability $p \approx 0.5$ (or smaller). Hence model updates will not use these node.
- During the testing, all nodes are retained, but their output is multiplied by $p$. This can be viewed as a surrogate of model average of sparse networks.
- Originally for MLP; Some modification is need for CNN layer (pooling dropout/cutout) and RNN (variational dropout: drop the same network units at each time step)

# Lottery Ticket Hypothesis

*A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

- Winning ticketS: sparse, randomly initialized subnetworks that, when trained alone, achieve comparable accuracy to the full, large network.
- Identifies these tickets through iterative magnitude pruning, which removes low-magnitude weights, resets the remaining weights to their original initialization, and trains the smaller, sparse network.
- Training success depends on identifying these sparse, well-initialized subnetworks
- Inefficient training the whole huge network may be unnecessary

# Strong LTH

*A sufficiently over-parameterized random network contains a sub-network that achieves high accuracy without any training.*

- Compare to UAT, (S)LTH claims the existence of an approximation within an random dense network.
- In practice, usually, 90% or more reduction in parameters without significant loss in accuracy
- Finding winning tickets typically requires iterative pruning and retraining

# Manifold Hypothesis of modern data set

- High-dimensional data (such as images, text, or audio) actually resides on or around a lower-dimensional, structured manifold (or the union of multiple low dimensional manifold) embedded within the high-dimensional ambient space.
- It justifies the usefulness of nonlinear dimensional reduction technique
- Manifold learning algorithm