# Problem Solving with AI Techniques
# Convolutional/Recurrent Neural Networks

Paul Weng

UM-SJTU Joint Institute

VE593, Fall 2018

# Motivations

- MLP has a simple architecture, may become hard to train when the number of layers becomes large

- Issue: MLP doesn't really exploit a priori knowledge we may have
- Example: Classification tasks in computer vision enjoy e.g., locality and translation invariance
- Convolutional NNs exploit such properties

- Issue: MLP requires a fixed-sized input
- Example: In NLP tasks, inputs have varying lengths
- Recurrent NN accept sequential inputs

# Training Many-Layers ANN

Recall backpropagation:

$$\delta_j^L = \frac{\partial R}{\partial a_j^L} f^{l\prime}(z_j^L)$$

$$\boldsymbol{\delta}^l = \left(\boldsymbol{w}^{l+1\intercal} \boldsymbol{\delta}^{l+1}\right) \otimes f^{l\prime}(\boldsymbol{z}^l)$$

$$\frac{\partial R}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial R}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

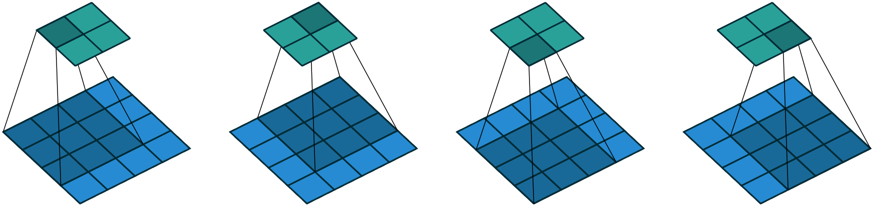
Issue: Unstable gradient:

- Vanishing gradient
- Exploding gradient

# Convolution Neural Network (CNN)

- CNNs are feedforward ANNs and is a variation of MLP
- MLPs only contain fully-connected layers
- CNN also contains:
    - Convolutional layers
    - Pooling layers

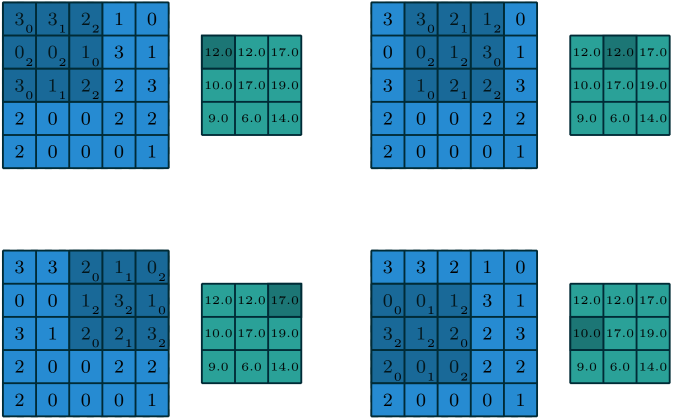
# Convolutional Layer: Introduction



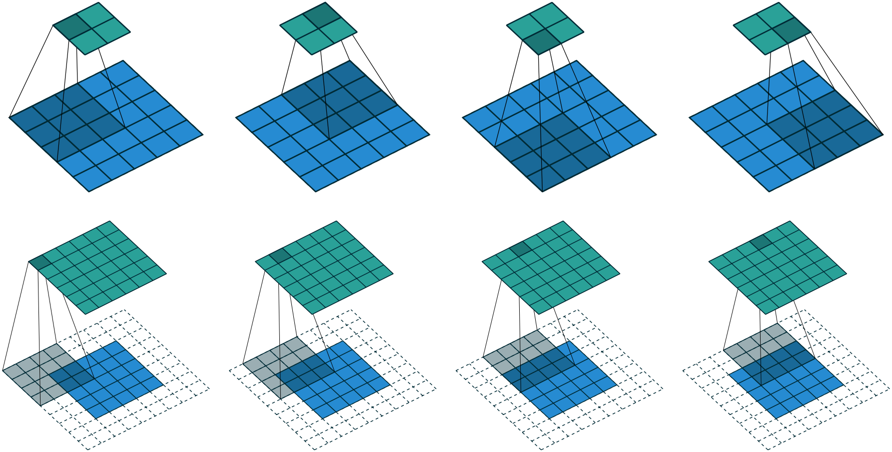
- A node is connected to a small square area
- All nodes share the same weights, which are learned
- This set of weights defines a filter (or kernel)
- A filter defines a feature map

# Convolutional Layer: Numerical Example

Assuming a filter with weights $\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$, bias/activation fcn ignored



# Convolutional Layer: Strides and Zero-Padding



# Convolutional Layer: Hyperparameters

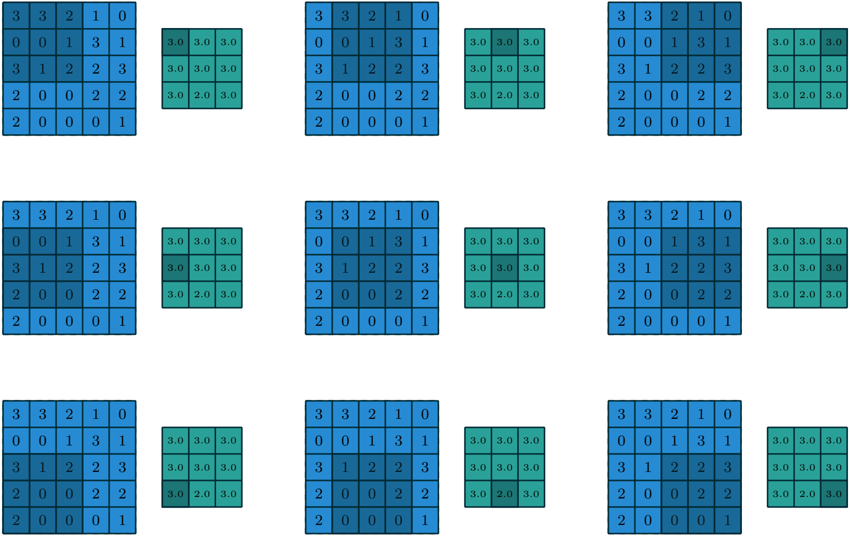
- Depth: # of filters
- Filter size
- Stride
- Pading

- Output size for 1 filter: (input size - filter size + 2 x padding) / stride + 1

# Pooling Layer: Introduction

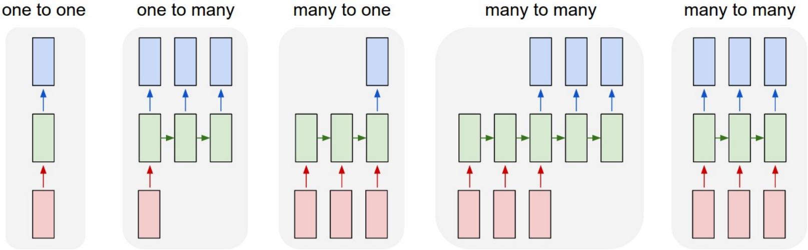
- Feature maps may be of high dimension
- Pooling layer summarizes the information from previous layer
- No learning at pooling layer

# Pooling Layer: Max



# Recurrent Neural Network (RNN)

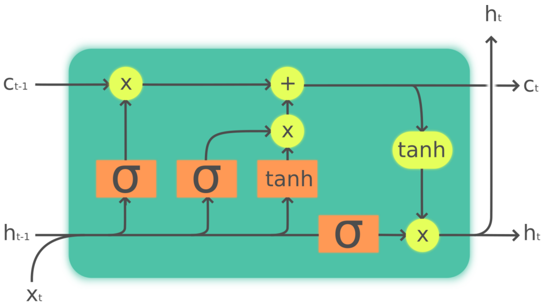
- RNNs are defined by graphs that contain cycles
- Cycles endow RNNs with a "memory"




from Karpathy

- Issue: Managing long-term dependencies
- Long Short Term Memory (LSTM) network

# Long Short Term Memory (LSTM) network




from Wikipedia

- Forget gate: $f_t = \sigma\big(W_f(h_{t-1}, x_t) + b_f\big)$
- Input gate: $i_t = \sigma\big(W_i(h_{t-1}, x_t) + b_i\big)$
- Cell update: $c_t = f_t \otimes c_{t-1} + i_t \otimes \tanh\big(W_c(h_{t-1}, x_t) + b_c\big)$
- Output gate: $o_t = \sigma\big(W_o(h_{t-1}, x_t) + b_o\big)$
- Output: $h_t = o_t \otimes \tanh\big(c_t\big)$

# Heuristics to Train ANNs

- Different initialization techniques
  - $w_{ij}^l \sim \mathcal{N}(0, \varepsilon)$
  - $w_{ij}^l \sim \mathcal{N}(0, \sqrt{\frac{2}{n^{l-1}}})$
  - all weights are zero, except for a fixed number of connections per node

- Batch normalization
  - Idea: Input is normalized, why not the next layers?
  - Each activation is normalized for a minibatch

# ML Pipeline

**Standard ML pipeline**

- Data acquisition
- Data preprocessing
- Feature engineering
- Model building/training/selection
- Model deployment

**Deep learning pipeline**

- Data acquisition
- Data preprocessing
- Model building/training/selection
- Model deployment

Data acquisition: Data collection, aggregation, consolidation

Data preprocessing: Data cleaning, missing data imputation, normalization

Feature engineering: Feature construction/selection

Model building, training and selection: Training on training dataset, hyperparameter tuning on validation dataset, model selection on validation dataset, cross-validation, evaluation on test dataset

Model deployment: Profit!

# Why do we need to divide the data into different datasets?

- Training dataset for training models
- Validation dataset used for hyperparameter tuning/model selection
- Testing dataset used for evaluating the final model

- An estimation is biased if it is computed on dataset that was used to optimize parameters or hyperparameters
- Risk of performance overestimation

# Cross-validation

- Issue: One estimation may have a high variance
- Idea: Use several estimations!
- Principle of $k$-fold cross-validation
  - Divide dataset in $k$ parts (called folds)
  - For each fold, train model on $k-1$ remaining folds and evaluate model on it
  - Average evaluations

# Hyperparameter Optmization

- Grid search
- Random search
- Reinforcement Learning

# Learning Rate Decay

Learning rate schedule:

- Step decay $\alpha_t = \alpha_0 \beta^{\lfloor t/h \rfloor}$ where $\beta \in (0, 1)$ and $h \in \mathbb{N}$
- Exponential decay $\alpha_t = \alpha_0 e^{-\beta t}$ where $\beta > 0$
- Time-based decay $\alpha_t = \frac{\alpha_0}{1+\beta t}$ where $\beta > 0$

Accelerated (stochastic) gradient descent:

- Accelerated first-order methods: Momentum and Nesterov acceleration
- Adagrad (Adaptive gradient algorithm)
- RMSprop (root mean square propagation)
- Adam (adaptive moment estimation)