Deep Learning

CS444-Deep Learning for Computer Vision
CS446-Machine Learning

Qi Long

Contents

0 Preliminaries

0.1 Matrix Derivative

0.1.1 First Order

$$\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a$$

$$\frac{\partial a^T X b}{\partial X} = ab^T$$

0.1.2 Second Order

$$\frac{\partial x^T B x}{\partial x} = (B + B^T)x$$

$$\frac{\partial b^T X^T X c}{\partial X} = X(bc^T + cb^T)$$

0.2 Dual Program

0.2.1 Primal Program

The goal is to Find the optimum of a function given a restriction of another function, for example,

$$\max_{x,y} f(x, y)$$

given restriction

$$g(x, y) = c$$

0.2.2 Lagrangian

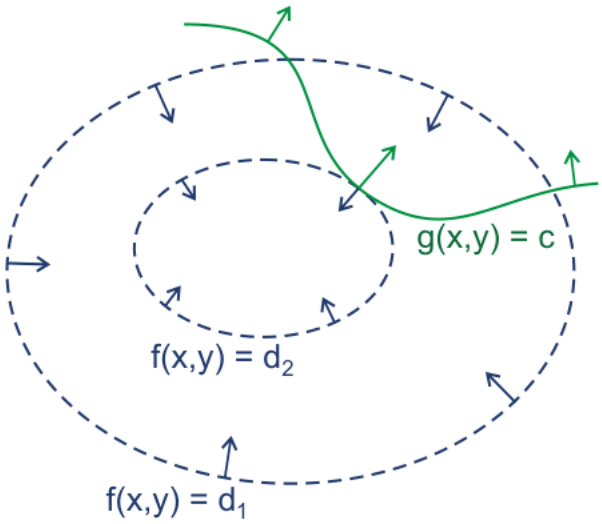
$f(x, y) = d_n$ can be regarded as contours with **adjustable** values d_n.
$g(x, y) = c$ can be regarded as a fixed curve crossing contours.
Then the optimum is when $f(x, y) = d_n$ and $g(x, y) = c$ have same derivative.
Import λ to represent adjustable d_n,

$$\nabla \frac{1}{\lambda} f(x, y) = \nabla(g(x, y) - c)$$

$$\nabla[f(x, y) - \lambda(g(x, y) - c)] = 0 \tag{1}$$

The corresponding Lagrangian Program is

$$L(x, y, \lambda) = f(x, y) - \lambda(g(x, y) - c)$$



0.2.3 Dual Form

Solves equation (1) to represent primal parameters (x, y) using Lagrangian Multiplier λ, getting the dual form equation with only one parameter λ.
After solving λ, plug λ back to get optimal x^* and y^*.

0.3 Vector Norms

0.3.1 p-norm

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \quad p \geq 1$$

0.3.2 1-norm

Property of encouraging sparsity: even differences among all magnitudes.

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

0.3.3 2-norm

Analog to Euclidean distance: straight-line distance.

$$||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

0.3.4 max-norm

Largest one dimension.

$$||x||_\infty = \max_i |x_i|$$

1 Perceptron

1.1 Objective

Learn a hyperplane $\mathbf{w}$ that separating data points into classes.
Prediction:

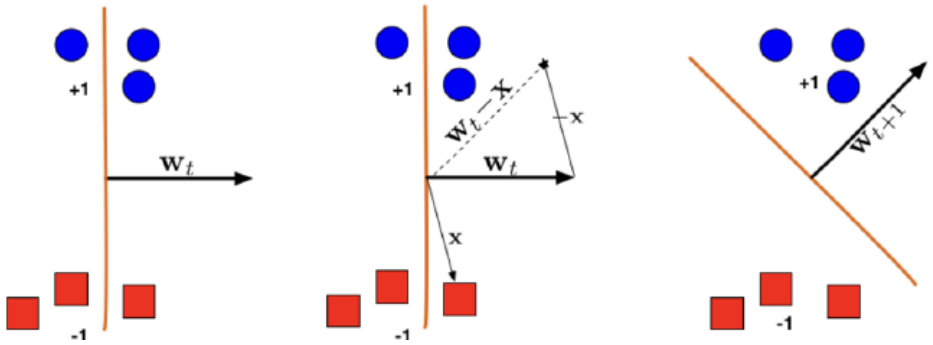
$$\hat{y} = \text{sign}(\mathbf{w^T x})$$

where $\mathbf{x}$ includes biased term, i.e., $\mathbf{x} \leftarrow [\mathbf{x}, 1]$.
Objective: $\mathbf{x_i}$ is classified correctly if

$$y_i(\mathbf{w^T x_i}) > 0$$

Interpretation: geometrically, $\vec{\mathbf{w}}$ is perpendicular to decision boundary, pointing towards one class. Therefore, $\mathbf{w^T x}$ measures similarity between class vector and data point.

1.2 Model Update



Algorithm: if $y(\mathbf{w}^\mathbf{T}\mathbf{x}) \leq 0$,

$$\mathbf{w}_{new} \leftarrow \mathbf{w} + y\mathbf{x} \tag{2}$$

Derivation:

$$y(\mathbf{w}^\mathbf{T}{}_{new}\mathbf{x}) = y(\mathbf{w} + y\mathbf{x})^T\mathbf{x} = y(\mathbf{w}^\mathbf{T}\mathbf{x}) + y^2||\mathbf{x}||^2 > y(\mathbf{w}^\mathbf{T}\mathbf{x})$$

Interpretation: when a data point is misclassified, add it to or subtract it from the class arrow $\mathbf{w}$, so that class arrow is pointing closer or farther from the data point.
Convergence: Assume $\mathbf{w}$ is current misclassified hyperplane, $\mathbf{w}^*$ is a separating hyperplane, all data points are scaled down to unit circle, exists a separating hyperplane $||\mathbf{w}^*|| = 1$, then at most $\frac{1}{\gamma^2}$ updates.
Proof:

$$y(\mathbf{x}^\mathbf{T}\mathbf{w}) \leq 0 \qquad y(\mathbf{x}^\mathbf{T}\mathbf{w}^*) \geq 0$$

by update rule (2), each update $\mathbf{w}^\mathbf{T}\mathbf{w}^*$ grows by at least γ ($\mathbf{w}$ gets closer to $\mathbf{w}^*$),

$$\mathbf{w}^\mathbf{T}{}_{new}\mathbf{w}^* = (\mathbf{w} + y\mathbf{x})^T\mathbf{w}^* = \mathbf{w}^\mathbf{T}\mathbf{w}^* + y(\mathbf{x}^\mathbf{T}\mathbf{w}^*) \geq \mathbf{w}^\mathbf{T}\mathbf{w}^* + \gamma$$

each update $\mathbf{w}^\mathbf{T}\mathbf{w}$ grows by at most 1 (magnitude of $\mathbf{w}$ increases within range)

$$\mathbf{w}^\mathbf{T}{}_{new}\mathbf{w}_{new} = (\mathbf{w} + y\mathbf{x})^T(\mathbf{w} + y\mathbf{x}) = \mathbf{w}^\mathbf{T}\mathbf{w} + 2y(\mathbf{w}^\mathbf{T}\mathbf{x}) + y^2(\mathbf{x}^\mathbf{T}\mathbf{x}) \leq \mathbf{w}^\mathbf{T}\mathbf{w} + 1$$

Since initially $\mathbf{w} = 0$, $\mathbf{w}^\mathbf{T}\mathbf{w} = \mathbf{w}^\mathbf{T}\mathbf{w}^* = 0$, after M updates,

$$\mathbf{w}^\mathbf{T}\mathbf{w}^* \geq M\gamma \qquad \mathbf{w}^\mathbf{T}\mathbf{w} \leq M$$

$$M\gamma \leq \mathbf{w}^\mathbf{T}\mathbf{w}^* = ||\mathbf{w}||||\mathbf{w}^*||\cos(\theta) \leq ||\mathbf{w}|| \cdot 1 = \sqrt{\mathbf{w}^\mathbf{T}\mathbf{w}} \leq \sqrt{M} \rightarrow M \leq \frac{1}{\gamma^2}$$

(See Github:CS440/ECE448-MP4 for more details)

(See Github:CS444-MP1 for more details)

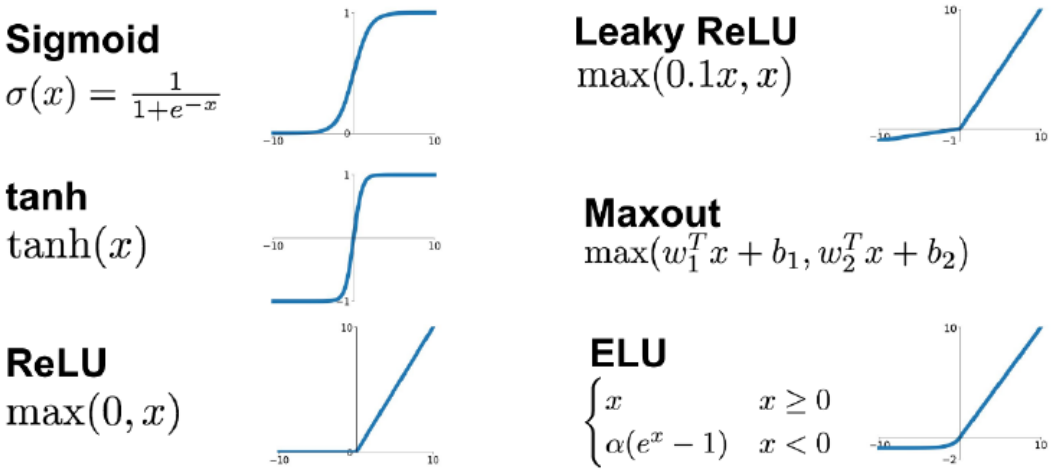
1.3 Neural Network

Prediction:

$$\hat{y} = \mathbf{w}^\mathbf{T}\phi(\mathbf{x}), \qquad \phi(\mathbf{x}) = \sigma(\mathbf{A}\phi'(\mathbf{x}) + \mathbf{b})$$

where σ is activation function.
Objective:

$$\underset{\mathbf{w}\in\mathbb{R}^m, \mathbf{A}\in\mathbb{R}^{m\times d}, \mathbf{b}\in\mathbb{R}^m}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} l(y_i\mathbf{w}^\mathbf{T}\sigma(\mathbf{A}\phi'(\mathbf{x}) + \mathbf{b}))$$



1.3.1 Nonlinearity

1.3.2 Multi-class Classification

Softmax activation: last non-linear layer, used for multiclass output.

$$f(\mathbf{z}) = \sum_{i=1}^{k} \frac{\exp(\mathbf{z_i})\mathbf{e_i}}{\sum_{j=1}^{k} \exp(\mathbf{z_j})}$$

Cross-entropy Loss: given one hot label $\mathbf{y} \in \{\mathbf{e}_1, \cdots, \mathbf{e}_k\}$ and probability vector $\hat{y} \in \mathbb{R}^k$,

$$l(y, \hat{y}) = -\sum_{i=1}^{k} \mathbf{y}_i \ln \hat{\mathbf{y}}$$

Combine CELoss with Softmax $\hat{\mathbf{y}} \propto \exp(\mathbf{z})$,

$$l(y, \hat{y}) = -\sum_{i=1}^{k} \mathbf{y}_i \ln \left(\frac{\exp(\mathbf{z_i})}{\sum_{j=1}^{k} \exp(\mathbf{z_j})} \right) = -\sum_{i=1}^{k} \mathbf{y}_i \mathbf{z}_i + \ln \left(\sum_{i=1}^{k} \exp(\mathbf{z}_i) \right)$$

1.3.3 Computer Vision Interpretation

Learnt parameters $\mathbf{w}$ are bank of templates to match similarity with images. Then each dimension of $\mathbf{w^T x}$ is computing the similarity between image and a feature template.

Last layer is a recombination of all templates for each class.

(See Github:CS440/ECE448-MP5 for more details)

1.4 Multi-layer Perceptrons

1.4.1 Back Propagation

Update parameters from end layers to the front by chain rule.
Computational Graph:
Common BPs:

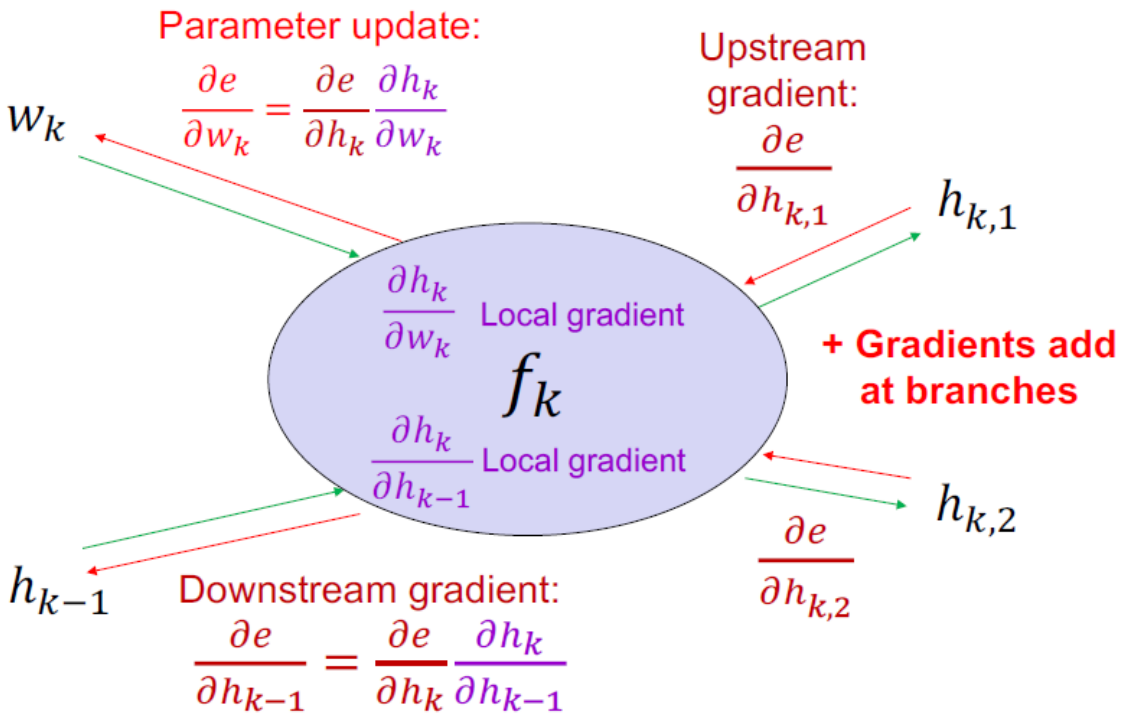
- Add Gate (Gradient Distributor): passing through addition, replicate gradient for each branches.
- Multiply Gate (Gradient Switcher): passing through multiplication, gradient is multiplied by the other branch's output.
- Max Gate (Gradient Router): no gradient for non-max branches.

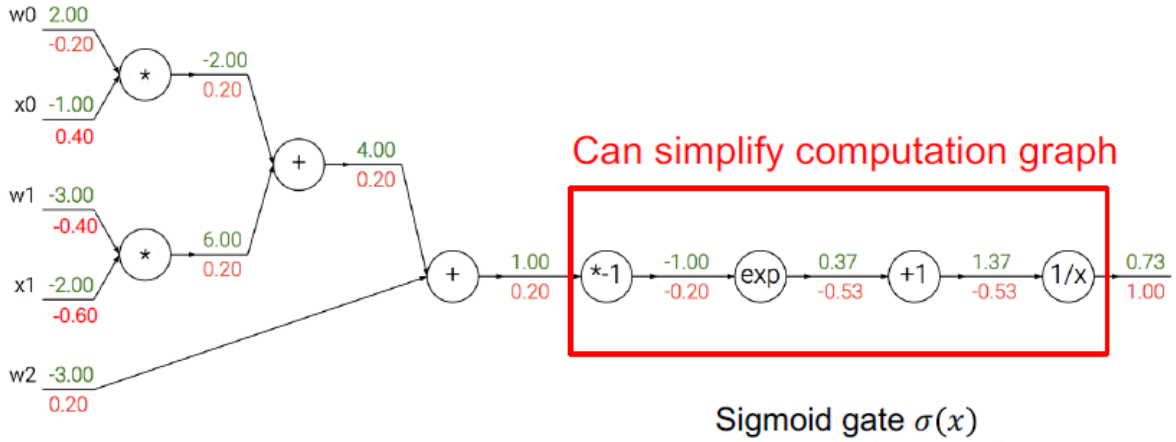
Matrix version:

1.4.2 Optimization

Gradient descent:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L(\mathbf{W})$$

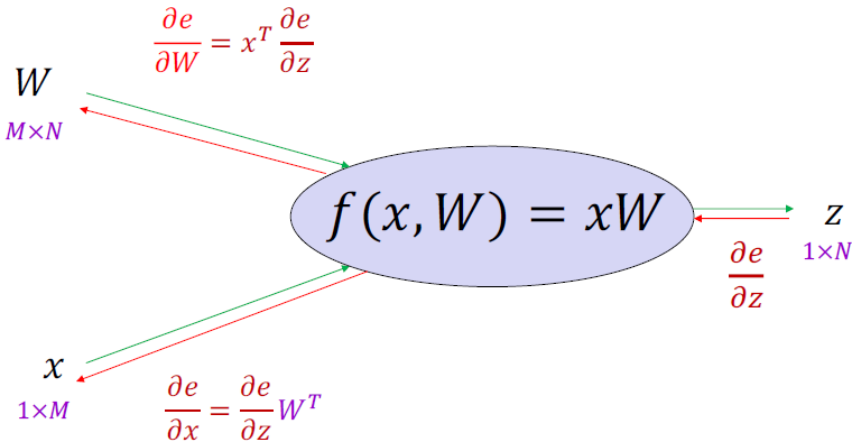




Sigmoid gate $\sigma(x)$

$$\sigma'(x) = \sigma(x)\big(1-\sigma(x)\big)$$

$$\sigma(1)\big(1-\sigma(1)\big) = 0.73 * (1-0.73) = 0.20$$



Mini-batch / SGD($b = 1$): update based on subset of data each time, has issue of being trapped in local minima and poor conditioning.

$$\nabla_{\mathbf{W}} L = \frac{1}{b} \sum_{i=1}^{b} \nabla_{\mathbf{W}} l(\mathbf{W}, x_i, y_i)$$

1.4.3 Hyperparameters

- Regularization constant λ: tradeoff between margin and classification errors.
- Number of layers / units per layer.
- Type of nonlinearity.
- Type of loss function.
- SGD settings: learning rate, number of epochs, minibatch size.

(See Github:CS444-MP2 for more details)

2 CNN

2.1 Convolution

Motivation: take local relations into account for Computer Vision.
Interpretation: compare similarity between a pattern (filter) with a sub-region.

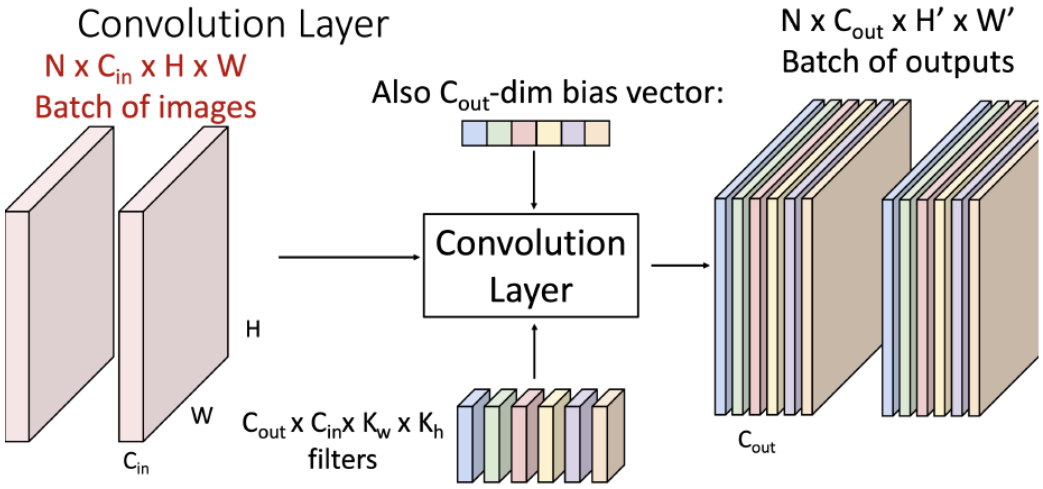
Correlation: filter $h[i,j]$ and sub-region $x[i,j]$

$$y[k,l] = \sum_i \sum_j h[i-k, j-l]x[i,j]$$

Convolution: filter $h[i,j]$ and sub-region $x[i,j]$

$$y[k,l] = \sum_i \sum_j h[k-i, l-j]x[i,j] = h[k,l] * x[k,l]$$

Implementation: Each filter of shape $(C_{in} \times K_w \times K_h)$ is applied to each image $(C_{in} \times W \times H)$,



sum up over all C_{in} results in one $(W^{'} \times H^{'})$. Then after C_{out} filters, the output is concat as $(C_{out} \times W^{'} \times H^{'})$.

2.1.1 Computation

- Output size: N is total pixels in one dimension, F is pixels in one dimension of a filter.

$$(N - F + pad \times 2)/stride + 1$$

- Trainable parameters: C_{out} is also number of filters, $+1$ stands for bias term.

$$C_{out} \times (C_{in} \times F^2 + 1)$$

  filter sliding over whole image reduce the trainable parameters heavily compared to fully connected layers.

2.1.2 Bottleneck Module

Since applying each filter results in a one-channel output and so the number of filters determine number of output channels, a 1×1 Conv Layer can be used to reduce weights and operations. For example:

- 3×3 Conv Layer with $C_{in} = C_{out} = 256$, weights and operations per location:

$$256 \times 256 \times 3 \times 3 = 600,000$$

- 1×1 Conv Layer with $C_{in} = 256$, $C_{out} = 64$, then 3×3 Conv Layer with $C_{in} = C_{out} = 64$, then 1×1 Conv Layer with $C_{in} = 64$, weights and operations per location:

$$256 \times 64 \times 1 \times 1 + 64 \times 64 \times 3 \times 3 + 64 \times 256 \times 1 \times 1 \approx 70,000$$

2.1.3 Back Propagation

$$\frac{\partial L}{\partial h[i,j]} = \sum_k \sum_l \frac{\partial L}{\partial y[k,l]} \frac{\partial y[k,l]}{\partial h[i,j]} = \sum_k \sum_l \frac{\partial L}{\partial y[k,l]} x[k-i, l-j]$$

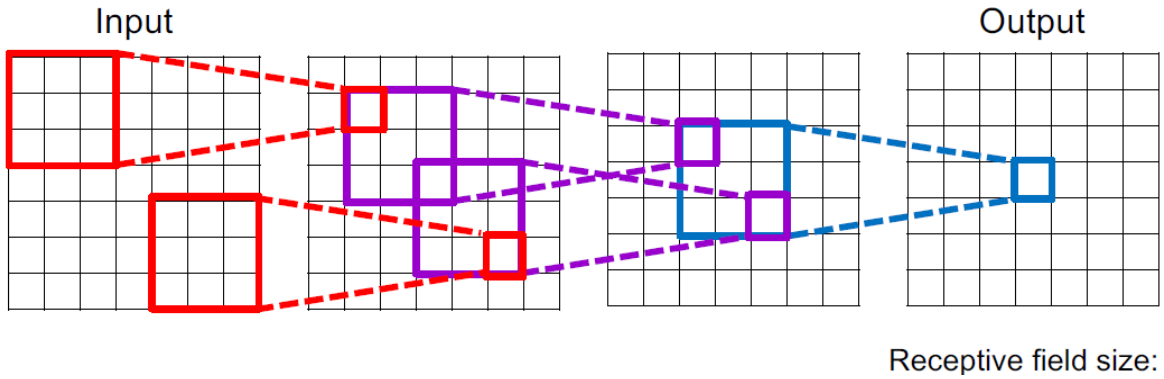
2.1.4 Transposed Convolution

$$\frac{\partial e}{\partial x_i} = w_3 \frac{\partial e}{\partial z_{i-1}} + w_2 \frac{\partial e}{\partial z_i} + w_1 \frac{\partial e}{\partial z_{i+1}}$$

as one x_i can contribute to multiple pixels in a filter as filter slides.

2.2 Pooling

Motivation: larger receptive field with less layers. Max Pooling:



$$z[m,n] = \max_{(m-1)p+1 \le k \le mp, (n-1)p+1 \le k \le np} y[k,l]$$

No trainable parameters.
Back Propagation:

$$\frac{\partial L}{\partial y[k,l]} = \frac{\partial L}{\partial z[m,n]} \frac{\partial z[m,n]}{\partial y[k,l]} = \frac{\partial L}{\partial z[m,n]}, \quad \text{if} \quad y[k,l] = \max_{(m-1)p+1 \le i \le mp, (n-1)p+1 \le j \le np} y[i,j]$$

(See Github:CS446/ECE449-MP3 for more details)

2.3 Architectures

2.3.1 AlexNet (2012-2013)

Successor of LeNet-5 with following improvements.

- Add max pooling, ReLU nonlinearity.
- Dropout Regularization.
- More data and bigger model: 7 hidden layers, 650K units, 60M params.
- GPU implementation: trained on 2 GPUs for a week.

Sub optima:

- 11×11 Conv Layers, too many weights.
- 5×5 Pooling Layers, too large receptive field.

2.3.2 VGGNet (2014)

Improvements:

- Deeper Network.
- Replace large receptive fields with successive layers of 3×3 Conv + ReLU.
- Max pooling layers are unified to 2×2 stride 2, followed by doubling of number of channels ($W^{'}$, $H^{'}$ decrease while C_{out} increase, to keep cost of conv the same).

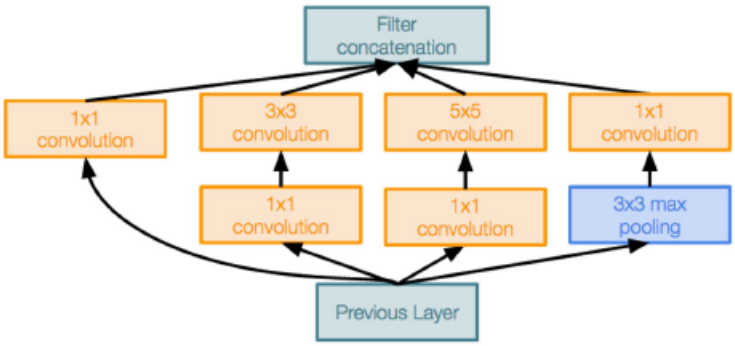
Sub optima:

- FC layers are parameter-heavy.

2.3.3 GoogLeNet (2014)

Improvements:

- Start with Stem Network: aggressively down-sampling input.
- **Inception Module**: parallel paths with different receptive field sizes and operations to capture sparse patterns of correlations in the stack of feature maps (machine learn a balance between different sizes of conv). Besides, use bottleneck Module for weight reduction.



- Replace FC layers with global average pooling (each of 1024 channels remain one average) to collapse spatial dimensions at the end.
- Auxiliary Classifier: add prediction heads to middle layers of the network, since network is too deep to back propagate cleanly from the end.

Sub optima:

- Auxiliary Classifier is not the best solution.

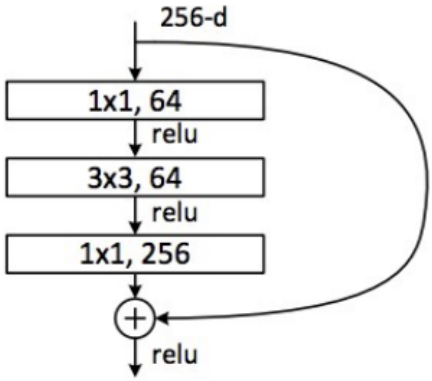
2.3.4 ResNet (2015-)

Improvements:

- Deeper **Residual Module**: skip or shortcut connections between layers. Make it easy for network layers to represent identity mapping (Allow layers partially keep original data throughout the network so that deeper model is practical). Use bottleneck trick to reduce weights.

Onward extentions:

- Wide ResNet: reduce number of residual blocks while increase number of feature maps in each block.



- ResNeXt: propose cardinality apart from depth and width (separate channels into groups independent to each other).
- DenseNets: Residual Module can cross several layers.

2.3.5 Usage

Transfer Learning: Regard architectures as pre-trained network, keep backbones frozen or fine-tune backbones, replace last layers (prediction heads) to new prediction layers.
(See Github:CS444-MP3.1 for more details)

2.4 Object Detection

2.4.1 Objective

Given an image, predict the locations that contain objects.
Evaluation:

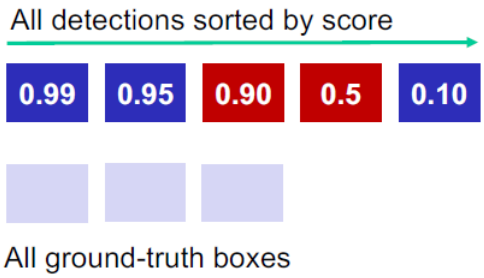
- Get all detections sorted by IoU score.

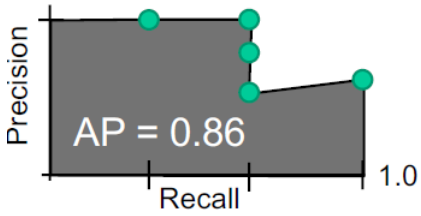
$$\text{IoU} = \frac{Area(intersection)}{Area(union)}$$

- Starting from the highest box, match its class prediction to ground-truth boxes. If match with IoU > 0.5, mark as positive and eliminate that matched ground-truth box (cannot be matched again), otherwise negative.
- Plot on PR curve after matching each box.

$$\text{Precision} = \frac{true - positive - detections}{total - detections - so - far} \qquad \text{Recall} = \frac{true - positive - detections}{true - positive - test - instances}$$

- The size of region is AP score.





2.4.2 R-CNN

A break through in Object Detection field, can be adapted to any deep network.

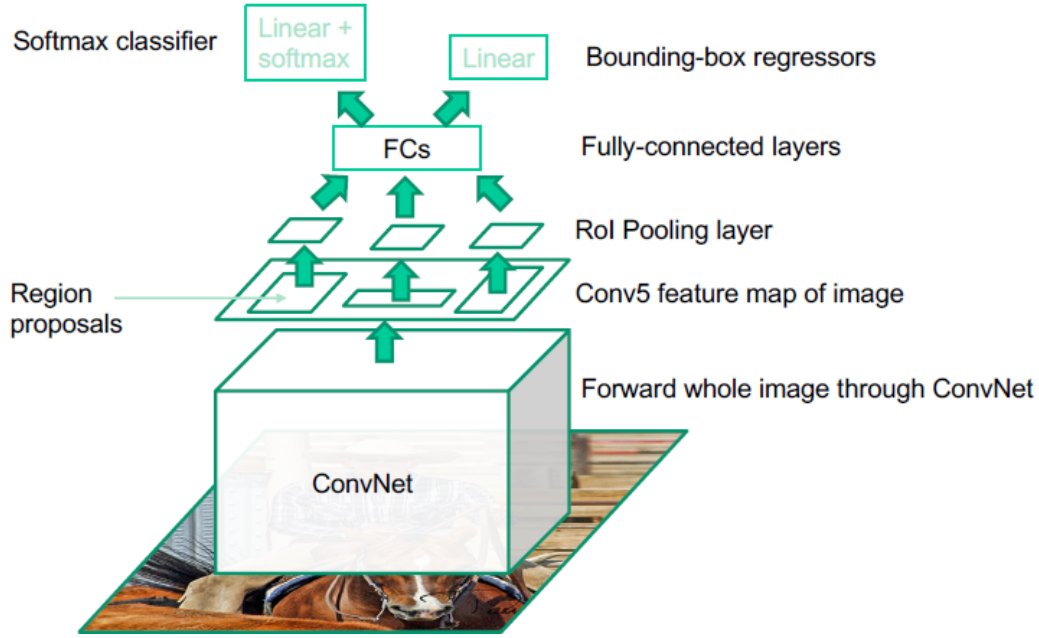
- Selective Search: non-CNN methods (segmentation) to select regions from image as proposal regions.
- CNN: feed wrapped selected regions to CNN for classification. (AlexNet)
- Bounding Box Regression: refine box locations.

Limitations: not a single end-to-end system, slow.

2.4.3 Faster R-CNN

Improve on R-CNN to an end-to-end system.

- CNN: feature extraction on whole image.
- Region Proposal: select regions on convoluted feature layer (non-deep-learning methods).
- RoI Pooling: match bounding boxes into sub-regions then max-pooling within each subregion (Back propagation similar to max pooling, can have multiple subregions back propagate to one block).



- FC: FC + Softmax for class prediction, FC + regressor for bbox regression (given region proposal, predict bbox offset).
- Loss: Log loss on classification + smooth L1 loss on bbox regression.

$$L(y, P, b, \hat{b}) = -\log P(y) + \lambda \mathbb{I}[y \geq 1] \sum_{i=\{x,y,w,h\}} \text{smooth}_{L_1}(b_i - \hat{b}_i)$$

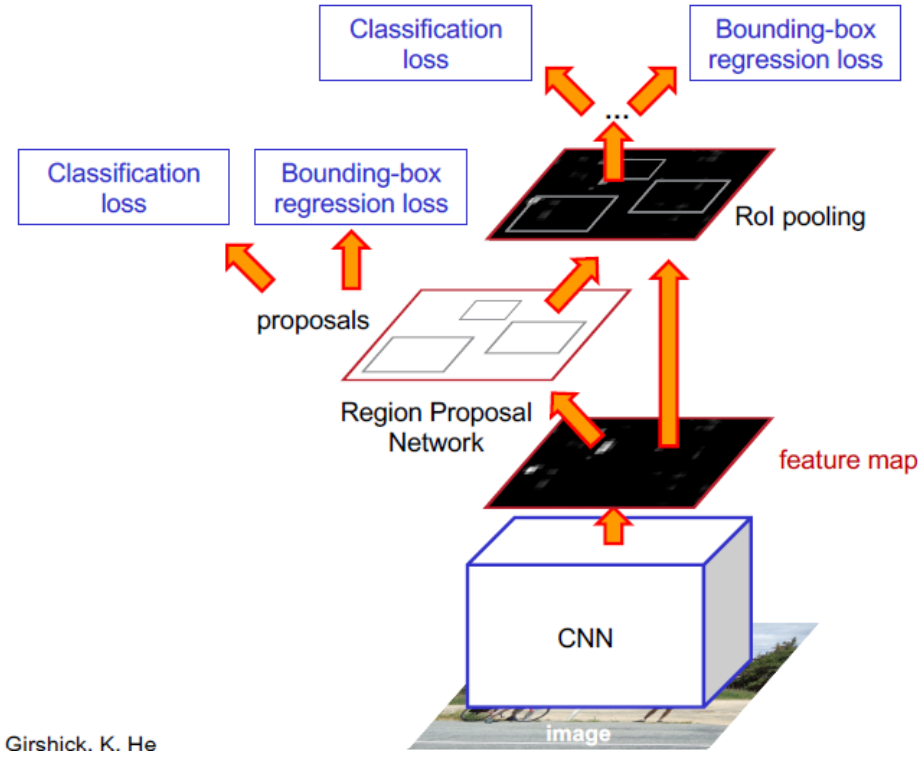
$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2, & |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}$$

Limitations: region proposal is complex.

2.4.4 Faster R-CNN

Improve on Fast R-CNN by replacing RP + RoI with RPN.

- CNN: feature extraction on whole image.
- Anchor Boxes: tile image with "anchor boxes" of a set size.
- Region Proposal Network (RPN) predict how likely the anchor boxes contain an object. (RPN directly compute loss by regressing predicted anchor boxes with ground truth anchor boxes)



- RoI Pooling & prediction.
- Loss: classification loss and bbox regression loss from RPN and final prediction.

2.4.5 YOLO

Key idea: divide image into a coarse grid and directly predict class label and a few candidate boxes for each grid cell. Loss:

- regression between ground truth bbox's center point and predicted bbox's center point.
- regression between the ground truth bbox size and predicted bbox size.
- confidence for a grid block having object.
- confidence for a grid block having no object.
- class probability prediction error.

(See Github:CS444-MP3.2 for more details)

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$
$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Regression

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$
$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Object/no object confidence

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Class prediction

2.4.6 Others

RetinaNet (feature pyramid network): improve predictive power of lower-level feature maps by adding contextual information from higher-level feature maps, predict different sizes of bboxes from different levels of pyramid.
FCOS: anchor free.
CenterNet: use additional center point to verify preditions.
DETR: CNN with Transformers.

2.5 Dense Prediction

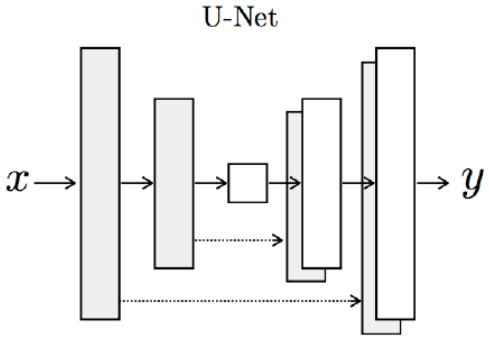
Input: image.
Output: segmented image of same size.
Architecture: first down sampling then up sampling (since input and output are of same size).

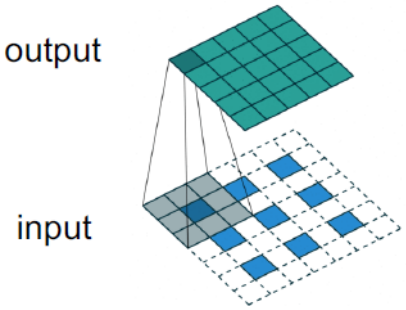
2.5.1 U-Net

Adding feature maps from down-sampling to up-sampling process.
Feature map up-sampling: insert pixels (average value) to between existing points. (Formally insert rows and columns of zero and apply conv with up-sampling filter)

2.5.2 DeconvNet

First convoluted to row vector and then deconvoluted to original size.





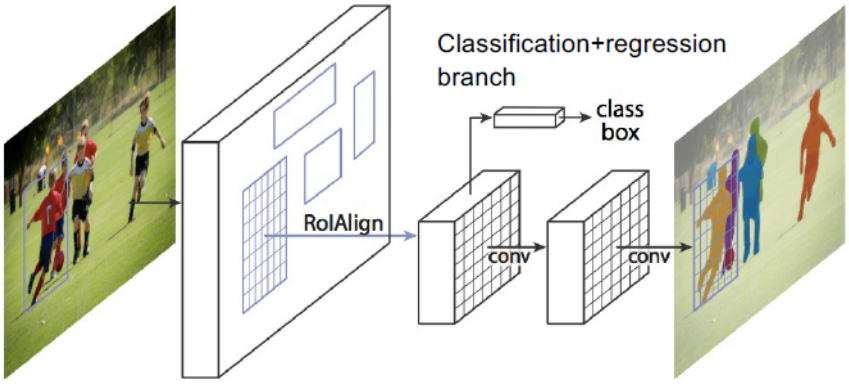
Upsampling filter:

$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
$\frac{1}{2}$	1	$\frac{1}{2}$
$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$

2.5.3 SegNet

A CNN Encoder-decoder architecture.

2.5.4 Mask R-CNN



Faster R-CNN + FCN on RoIs.

3 Neural Network Training

3.1 Optimization

- Mini-batch SGD.
- Learning rate decay: reduce learning rate by a constant factor every few epochs.
- Manual tuning: watch validation error and reduce lr whenever it stops improving.
- Warmup: train with a low lr or linearly increase lr for first few epochs.
- Early stop: when validation loss reach minimum (as a type of regularization).
- Adaptive Optimizer:
  - SGD with momentum: add "friction" coefficient β so that it moves fast in directions with consistent gradient, avoid oscillating.

$$m \leftarrow \beta m - \eta \nabla L \qquad\qquad w \leftarrow w + m$$

  - Adagrad: store all history gradients v_k, scale lr based on history (parameters with small gradients get large updates). Has issue of history accumulation leading to quick lr decay (As v_k gets larger, update magnitude to w_k becomes smaller).

$$v_k \leftarrow v_k + \left(\frac{\partial L}{\partial w_k}\right)^2 \qquad\qquad w_k \leftarrow w_k - \frac{\eta}{\sqrt{v_k} + \epsilon}\frac{\partial L}{\partial w_k}$$

    where power of 2 and $\frac{1}{2}$ are used for storing history magnitude.
  - RMSProp: improve based on Adagrad, introduce decay factor β to downweigh past history exponentially.

$$v_k \leftarrow \beta v_k + (1-\beta)\left(\frac{\partial L}{\partial w_k}\right)^2 \qquad\qquad w_k \leftarrow w_k - \frac{\eta}{\sqrt{v_k} + \epsilon}\frac{\partial L}{\partial w_k}$$

  - Adam: combine RMSProp with momentum.

$$m_k \leftarrow \beta_1 m_k - (1-\beta_1)\nabla L$$

$$v_k \leftarrow \beta_2 v_k + (1-\beta_2)\left(\frac{\partial L}{\partial w_k}\right)^2$$

$$w_k \leftarrow w_k - \frac{\eta}{\sqrt{v_k} + \epsilon} m_k$$

Besides, full algorithm includes bias correction to avoid m_k, v_k close to zero when k is small.

$$\hat{m} = \frac{m}{1-\beta_1^t} \qquad \hat{v} = \frac{v}{1-\beta_2^t}$$

where t is time step, computed as power.
Default parameters:

$$\beta_1 = 0.9 \qquad \beta_2 = 0.999 \qquad \eta = 10^{-3} \qquad \epsilon = 10^{-8}$$

(See Github:CS444-MP2 for more details)

3.2 Training Data

- Augmentation:
  - Geometric: flipping, rotation, shearing, multiple crops.
  - Photometric: color transformations.
  - Others: add noise, compression artifacts, lens distortions.
- Preprocessing: zero centering (subtract mean), rescaling (divide by standard deviation). (Need same transformation at both train and test time)

3.3 Numerical Tricks in Network

- Weight Initialization: weights random sampling from Gaussian $N(0, \sigma^2)$, bias set to 0.
  - Xavier init: $\sigma^2 = \frac{1}{n_{in}}$ or $\sigma^2 = \frac{2}{n_{in}+n_{out}}$, where n_{in} and n_{out} are number of inputs and outputs to a layer.
  - Kaiming init: $\sigma^2 = \frac{2}{n_{in}}$.
- Batch Normalization: network learn how to normalize data by differentiate with shifting and rescaling (γ, β are trainable).
  During test time, use training set's mean and variance instead of mini-batch. Usually add between Conv and ReLU Layers.
  Benefits: prevent exploding and vanishing gradients, keep most activations away from saturation regions of non-linearities, accelerate convergence, make training more robust.
  Pitfalls: require pattern matching between training and testing regime, do not work for small mini-batch sizes, limited to CNN.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

3.4 Regularization

- Loss with L1 / L2 regularization.
- Adding noise to inputs / weights.
- Label smoothing: replace hard $\{0, 1\}$ targets to soft $\{1 - \epsilon, \frac{\epsilon}{C-1}\}$.
- Dropout: at training time, in each forward pass, turn off some neurons with probability p (cut off all paths from and to the node); at test time, multiply output of neuron by p. This can prevent "co-adaptation" of units, increase robustness to noise and train implicit ensembles.

3.5 Others

- Ensembles: train multiple independent models and average their predictions during test time and take majority vote.
- Distillation: train a teacher network on initial labeled dataset, save softmax outputs of teacher network for each training sample, then train a student network with cross-entropy loss using softmax outputs of the teacher network as targets. (Compressing large models, copying black-box teacher model, extending a network to additional tasks without forgettiing old tasks)

4 RNN

4.1 Recurrent

Original RNN: Elman network

- sequential hidden layer:

$$h_t = f(h_{t-1}, x_t, w) = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + w_{hb})$$

$$\frac{\partial e}{\partial W_{hh}} = \frac{\partial e}{\partial h_t} \cdot \left(1 - \tanh^2(W_{hx}x_t + W_{hh}h_{t-1})\right) h_{t-1}^T$$

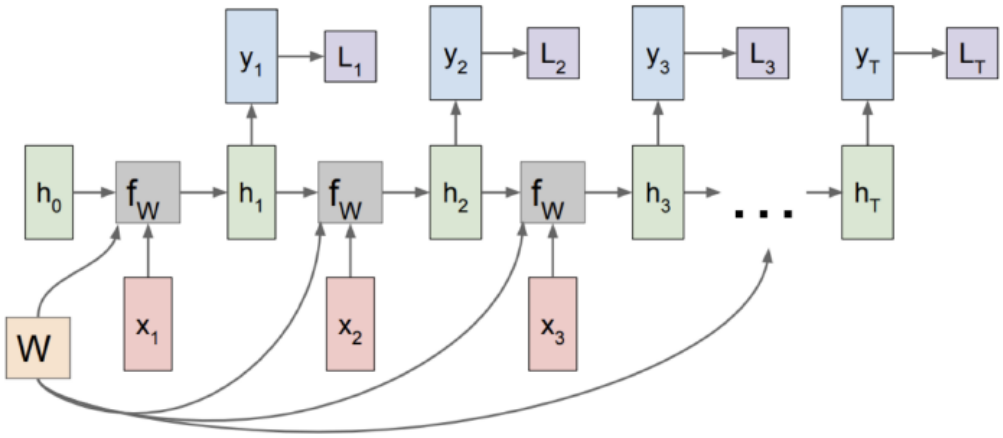
same for W_x.

$$\frac{\partial e}{\partial h_{t-1}} = W_{hh}^T \left(1 - \tanh^2(W_{hx}x_t + W_{hh}h_{t-1})\right) \cdot \frac{\partial e}{\partial h_t}$$

- output from hidden layer:

$$y_t = g(h_t) = \text{sigmoid}(W_{yh}h_t + w_{yb})$$

f and g are independent of time.
BPTT: Since f is applied to all steps, every step can back propagate through hidden layers or directly to w.

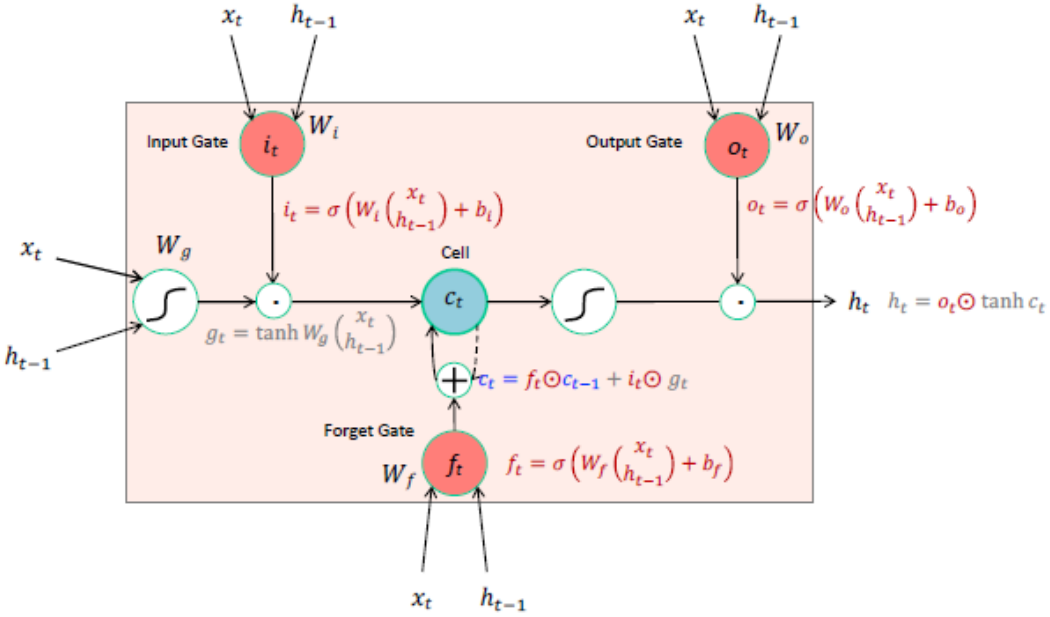


$$\frac{\partial L}{\partial w} = \sum_{t=1}^{n} \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial w}$$

Problem: vanishing gradient and long term dependency.
TPTT: To ameliorate the problems, truncate whole sequence into pieces and back propagate only within pieces.

4.2 LSTM

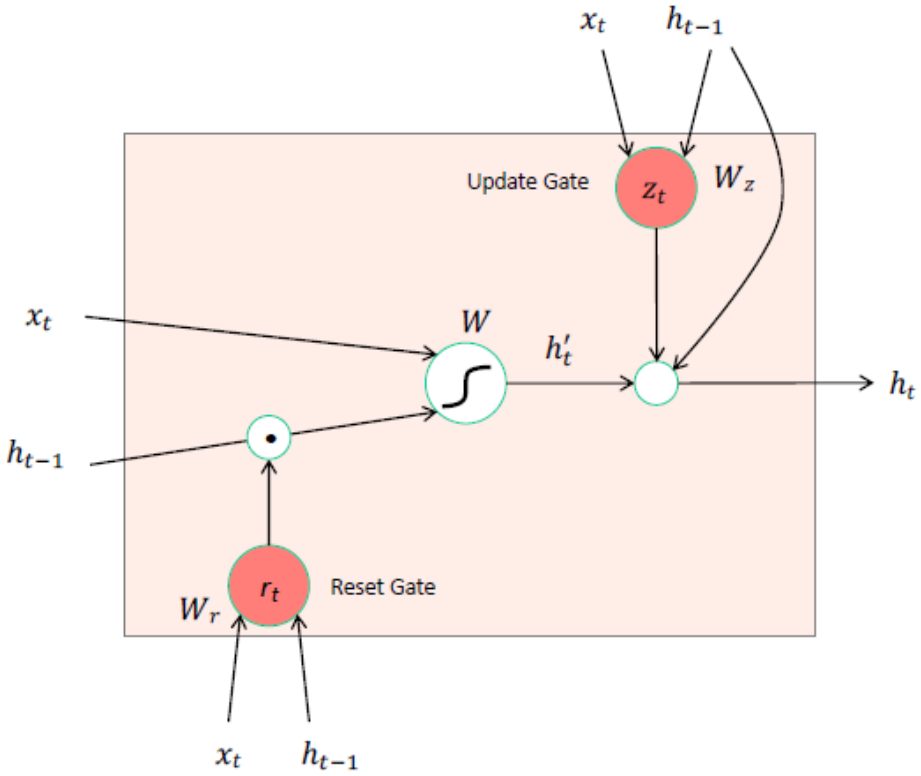
Compared to original RNN, LSTM better captures long-term dependencies and addressed the vanishing gradient problem.
(See Github:CS444-MP5 for more details)



$$
\begin{aligned}
i^{(t)} &= \sigma_i(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + w_{bi}) && \text{Input gate} \\
f^{(t)} &= \sigma_f(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + w_{bf}) && \text{Forget gate} \\
o^{(t)} &= \sigma_o(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + w_{bo}) && \text{Output/Exposure gate} \\
\tilde{c}^{(t)} &= \sigma_c(W_{cx}x^{(t)} + W_{ch}h^{(t-1)} + w_{bc}) && \text{New memory cell} \\
c^{(t)} &= f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)} && \text{Final memory cell} \\
h^{(t)} &= o^{(t)} \circ \sigma_h(c^{(t)})
\end{aligned}
$$

4.3 GRU

Reduce the parameters of LSTM.

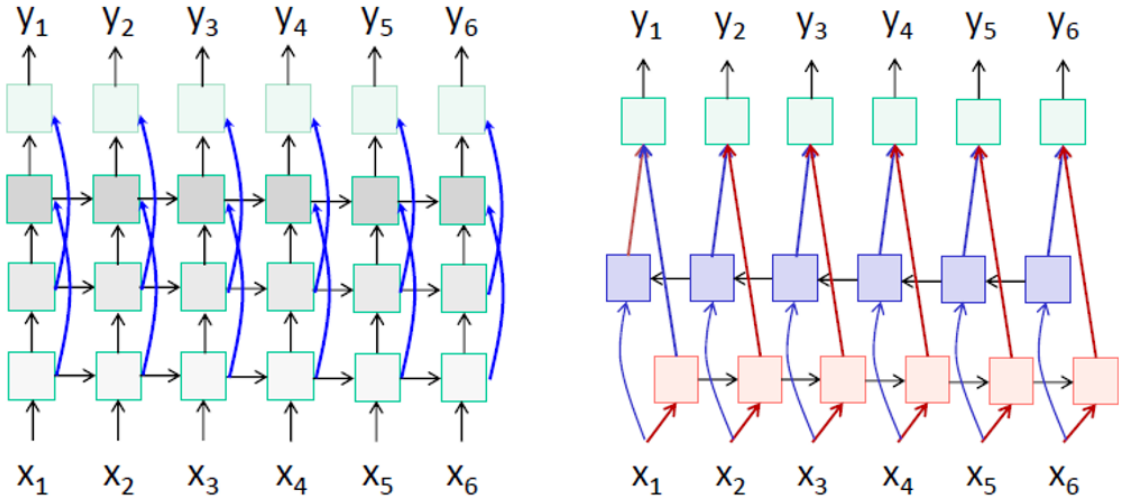


$$
\begin{aligned}
z^{(t)} &= \sigma_z(W_{zx}x^{(t)} + W_{zh}h^{(t-1)} + w_{bz}) && \text{Update gate} \\
r^{(t)} &= \sigma_r(W_{rx}x^{(t)} + W_{rh}h^{(t-1)} + w_{br}) && \text{Reset gate} \\
\tilde{h}^{(t)} &= \sigma_h(W_{hx}x^{(t)} + W_{rwh}(r^{(t)} \circ h^{(t-1)}) + w_{bh}) && \text{New memory cell} \\
h^{(t)} &= (1 - z^{(t)}) \circ \tilde{h}^{(t)} + z^{(t)} \circ h^{(t-1)} && \text{Hidden state}
\end{aligned}
$$

4.4 Architectures

4.4.1 Multi-layer RNNs

Multiple hidden layers.



4.4.2 Skip Connection

Hidden values depend on units other than nearby units. Both between hidden layers and between steps.

4.4.3 Bi-directional RNNs

Independent hidden layers in two directions.

4.5 Applications

4.5.1 NLP: Translation

4.5.2 CV: Image Caption Generation

Set image as initial hidden state h_0. Output from hidden state at each step is prediction of words, followed by Beam Search (See HMM) to generate sentences.

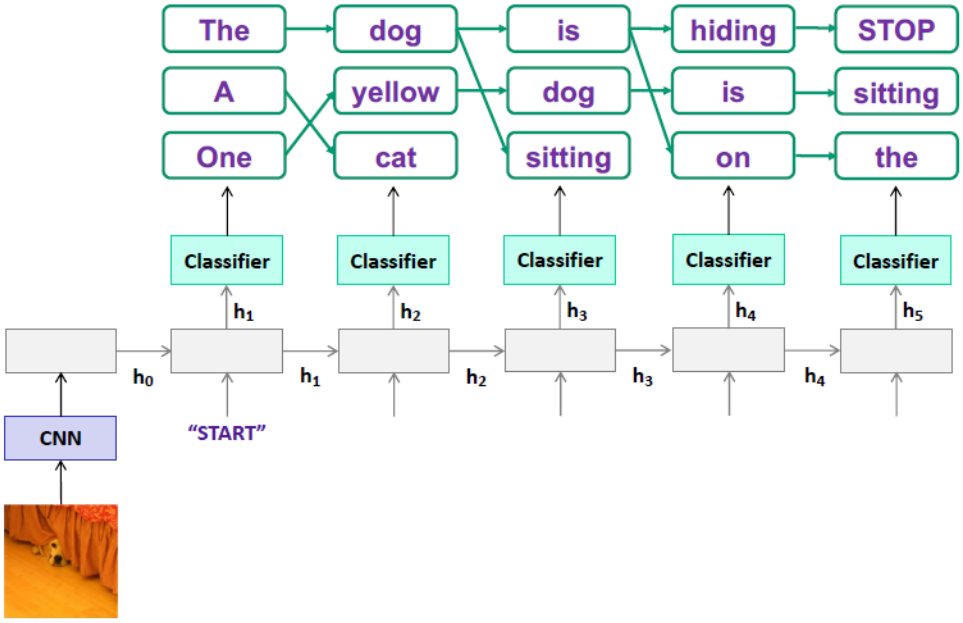
5 Transformers

5.1 Structure

5.1.1 Attention

Input vectors: query (Q), input (X). Parameters: key matrix (W_k), value matrix (W_v).

$$K = XW_k \qquad V = XW_v$$



Scores: D_Q is the dimension of query Q.

$$E = \frac{QK^T}{\sqrt{D_Q}}$$

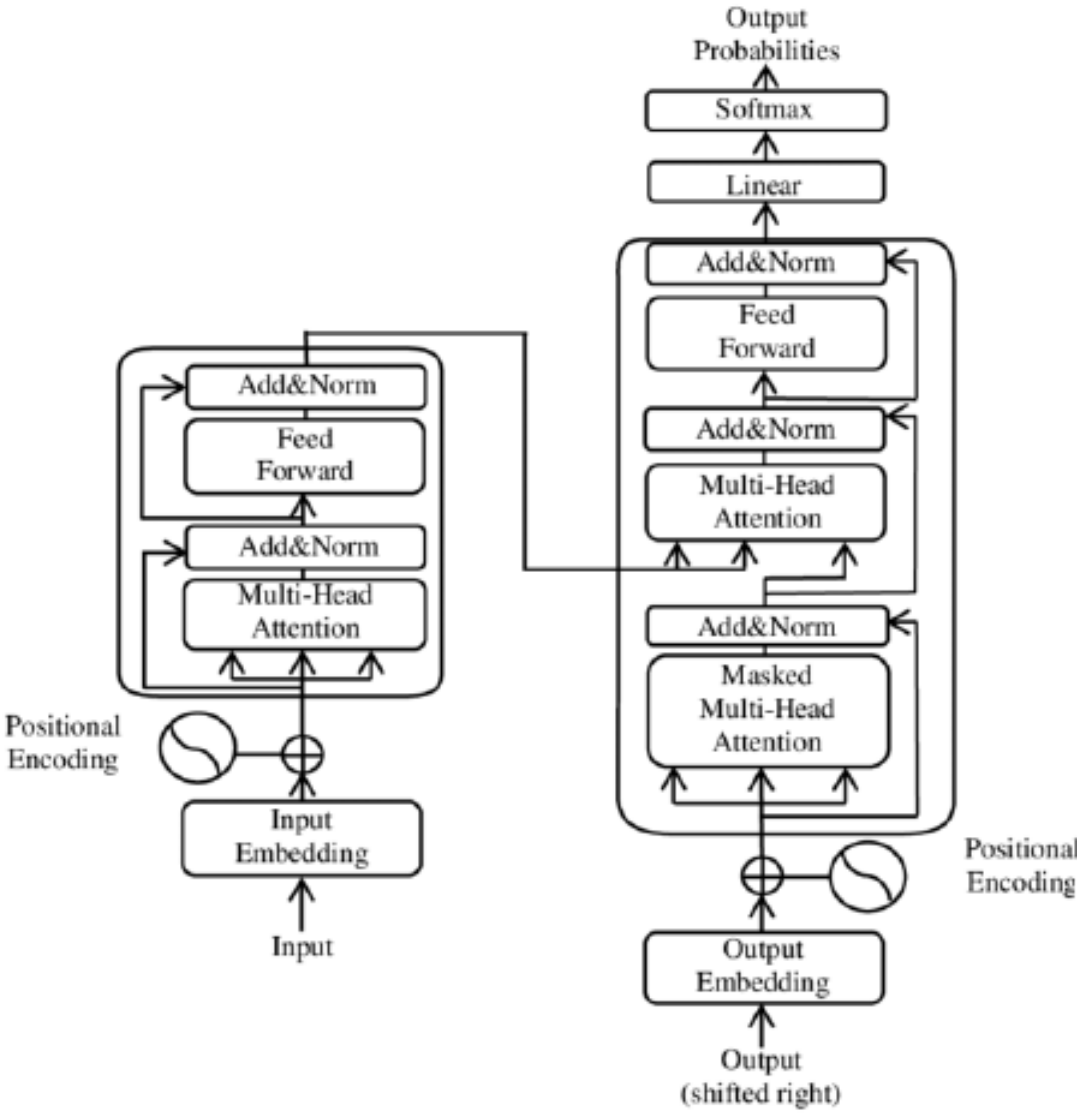
Attention: $A = \text{softmax}(E, \text{dim} = 1)$
Representation: $R = AV$
Self-Attention: replace query Q with trainable parameter W_Q, $Q = XW_Q$.
Multi-headed-Attention: each head uses a separate W_Q, W_K, W_V. After applying to input X, concat on second dimension.
Cross-Attention: used in decoder, Q depends on preceding output, K, V depend on input.
Masked-Attention: Set future output positions to $-\infty$ before softmax.

$$s(q_i, k_t) = f(x) = \begin{cases} q_i^T k_t, & t < i \\ -\infty, & t \geq i \end{cases} \qquad A = \begin{cases} \text{softmax}(q_i^T k_t), & t < i \\ 0, & t \geq i \end{cases}$$

5.1.2 Add & Norm

After attention layer, still need to maintain input x's information.

$$Y = \frac{X + R - E[X + R]}{\sqrt{Var(X + R)}}$$


Output Probabilities
Softmax
Linear
Add&Norm
Feed Forward
Add&Norm
Multi-Head Attention
Add&Norm
Masked Multi-Head Attention
Positional Encoding
Output Embedding
Output (shifted right)
Add&Norm
Feed Forward
Add&Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Input

5.1.3 Feed-Forward

Take Y from Attention and Add & Norm Layers,

$$Z = \text{LayerNorm}\left(Y + \text{DropOut}\left(\text{FFN}(Y)\right)\right)$$

$$\text{FFN}(Y) = \max(0, YW_1 + b_1)W_2 + b_2$$

5.1.4 Positional Embedding

Embed the relative location of tokens in a sequence by Fourier basis:

$$e_{t-d} = f(x) = \begin{bmatrix} \cos\left(\frac{\pi d}{T}\right) \sin\left(\frac{\pi d}{T}\right) & \cdots \\ -\sin\left(\frac{\pi d}{T}\right) \cos\left(\frac{\pi d}{T}\right) & \cdots \\ \vdots\vdots & \vdots \end{bmatrix}$$

Then add it into embeddings: $x_t = [x_t, e_t]$ or $x_t = x_t + e_t$.

5.2 Implementation

5.2.1 Teacher-forcing Training

During training time, given paired source and target sequences, we feed the source sequence into the encoder, we feed the **entire target sequence** as the decoder's input, masked out target sequence in a trapezoidal manner and we predict the left-shifted target sequence.

5.2.2 Auto-regressive Inference

During inference time, decoder must work in a **step-by-step** fashion, conditioning the prediction for time step $T + 1$ on the predicted tokens from $t = 1, 2, \cdots, T$.
Due to auto-regressive nature, in self-attention layer, Q (query) is from single current token, K, V (key & value) are from **all previously generated tokens**.
To avoid repeatedly computing K, V (key & value) at every decoding step (generating each token), after last feed-forward layer in decoder, we store current generated tokens' output value to **cache** and use them directly when future Q comes.
(See Github:CS440/ECE448-MP9 for more details)