Deep Learning for Computer Vision (CS231n)

Note: Yunlong Song

Among competing hypotheses, the simplest is the best. — Occam's Razor

Introduction

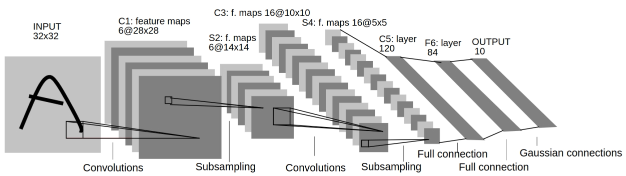


Figure 1: LeNet-5.

Image Classification with Linear Classifiers

Nearest Neighbor Classifier

- Memorize all training data.
- Predict the label of the most similar training image. Distance: L_1 (Manhattan)/L_2 (Euclidean).

K-Nearest Neighbor Classifier

- Predict the majority label of the k most similar training images.
- Hyperparameter: k.

Linear Classifier

- $f(x, W) = Wx + b$.
- W: weights, b: bias.
- $W \in \mathbb{R}^{D \times C}$, $b \in \mathbb{R}^C$.
- D: dimension of input, C: number of classes.

Multiclass SVM Loss: Given an example (x_i, y_i) and using the score $s = f(x_i, W)$, the loss is:

$$
\begin{aligned}
L_i &= \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases} \\
&= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\end{aligned}
$$

Softmax Loss:

$$
\begin{aligned}
L_i &= -\log P(Y = y_i | X = x_i) \\
&= -\log \left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right)
\end{aligned}
$$

Neural Networks and Backpropagation

Neural Networks

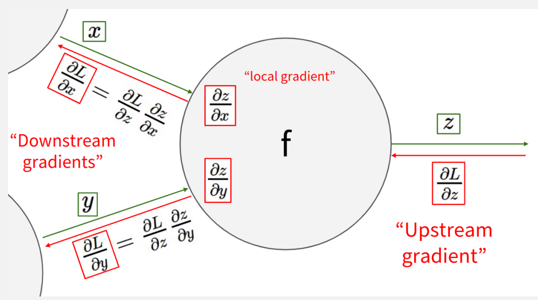
- Multiple layers of neurons.
- Convolutional Neural Networks (CNNs).
- Recurrent Neural Networks (RNNs).
- Long Short-Term Memory (LSTM).
- Gated Recurrent Unit (GRU).
- Transformer.

Activation Functions

- **Sigmoid**: $\sigma(x) = \frac{1}{1+e^{-x}}$.
- **Tanh**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
- **ReLU**: $\text{ReLU}(x) = \max(0, x)$.
- **Leaky ReLU**: $\text{LeakyReLU}(x) = \max(0.01x, x)$.
- **ELU**: $\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$.

Note: ReLU is the most popular activation function.

Backpropagation



Chain Rule:

$$
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \frac{\partial y}{\partial x}
$$

A simple example:

$$
\begin{aligned}
&f(x, y, z) = (x + y)z \\
&\frac{\partial f}{\partial x} = z, \frac{\partial f}{\partial y} = z, \frac{\partial f}{\partial z} = x + y
\end{aligned}
$$

Convolutional Neural Networks

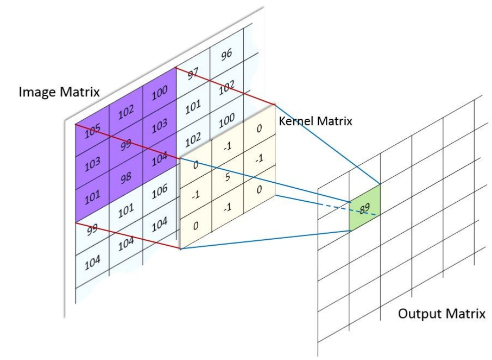


Figure 2: Kernel Convolution.

1D Convolution

- $f(x) * g(x) = \sum_n f(n) g(x - n)$.

2D Convolution

- $f(x, y) * g(x, y) = \sum_m \sum_n f(m, n) g(x - m, y - n)$.

3D Convolution

- $f(x, y, z) * g(x, y, z) = \sum_m \sum_n \sum_p f(m, n, p) g(x - m, y - n, z - p)$.

Convolutional Layer

- **Input**: $W_1 \times H_1 \times D_1$.
- **Output**: $W_2 \times H_2 \times D_2$.
- **Filter**: $F \times F \times D_1$.
- **Stride**: S.
- **Padding**: P.
- **Output Size**: $W_2 = \frac{W_1 - F + 2P}{S} + 1$.
- **Output Size**: $H_2 = \frac{H_1 - F + 2P}{S} + 1$.
- **Output Depth**: $D_2 =$ Number of filters.

Pooling Layer

- **Max Pooling**: $\max(x, y)$.
- **Average Pooling**: $\frac{x+y}{2}$.

Training Neural Networks

Activation Function

- **Sigmoid**: Vanishing gradient, not zero-centered.
- **Tanh**: Vanishing gradient, zero-centered.
- **ReLU**: Dying ReLU.
- **Leaky ReLU**: Solves dying ReLU,
- **ELU**: Solves dying ReLU

Note: Use ReLU as default, may try Leaky ReLU or ELU. Don't use sigmoid.

Data Preprocessing

- **Zero-centered**: Subtract the mean.
- **Normalization**: Divide by the standard deviation.
- **PCA/Whitening**: Decorrelate features.

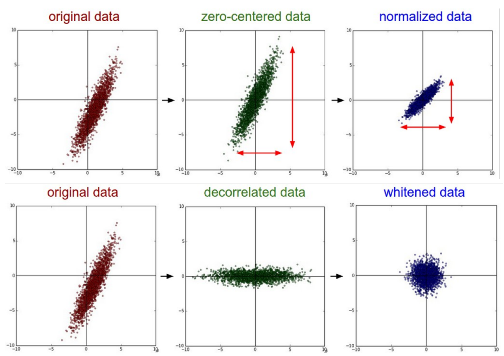


Figure 3: Data Preprocessing.

Weight Initialization

- **Small random numbers**:
  $W = 0.01 \times \text{randn}(\text{size}(W))$.
- **Xavier initialization**
- **He initialization**

Batch Normalization Motivation: "you want to normalize the input to each layer so that it has a nice distribution of values, e.g., Gaussian with zero mean and unit variance."

$$\hat{x} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}] + \epsilon}} \tag{1}$$

Batch Normalization

Input: Values of x over a mini-batch: $\{x_1, \ldots, x_m\}$.
Parameters to learn: γ, β.
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$.

$$\mu = \frac{1}{N}\sum_i x_i$$

$$\sigma^2 = \frac{1}{N}\sum_i (x_i - \mu)^2$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

Baby Sitting the Learning Process

- **Learning Rate**: Start with a small learning rate.
- **Hyperparameters**: Tune hyperparameters.
- **Monitor Loss**: Plot loss over time.
- **Overfitting**: Regularization, dropout, data augmentation.
- **Visualize**: Visualize weights, activations, gradients.

Regularization

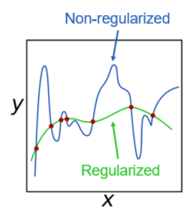


Figure 4: Regularization.

- $L =$ data loss + regularization loss.
- $L = L_i + \lambda R(W)$.
- $R(W) = \sum_k \sum_l W_{k,l}^2$ (L2 regularization).

Dropout

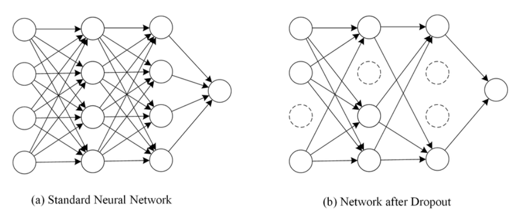


Figure 5: Dropout.

Data Augmentation

- **Random Cropping**: Randomly crop the image.
- **Random Flipping**: Randomly flip the image.
- **Color Jittering**: Randomly change the color.
- **Rotation**: Randomly rotate the image.

Optimization

- **Random Search**:
  $W = \text{randn}(\text{size}(W))$.
- **Gradient Descent**:
  $W = W - \alpha \nabla_W L$.
- **Stochastic Gradient Descent (SGD)**:
  $W = W - \alpha \nabla_W L_i$.
- **Mini-batch Gradient Descent**:
  $W = W - \alpha \nabla_W L_{\text{batch}}$.

SGD with Momentum

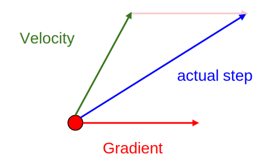


Figure 6: SGD with Momentum.

SGD with momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

- $v = \gamma v - \alpha \nabla_W L$.
- $W = W + v$.

AdaGrad AdaGrad is an adaptive learning rate method. The key idea is to adapt the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.

- $G = G + g^2$.
- $W = W - \alpha \frac{g}{\sqrt{G+\epsilon}}$.

RMSProp RMSProp is an unpublished, adaptive learning rate method proposed by Geoff Hinton. The key idea is to divide the learning rate by an exponentially decaying average of squared gradients.

- $E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g^2$.
- $W = W - \alpha \frac{g}{\sqrt{E[g^2]+\epsilon}}$.

Adam

Adam looks a bit like RMSProp with momentum. The key idea is to use the first and second moments of the gradients to adaptively adjust the learning rate for each parameter.

- $m = \beta_1 m + (1 - \beta_1) g$.
- $v = \beta_2 v + (1 - \beta_2) g^2$.
- $W = W - \alpha \frac{m}{\sqrt{v} + \epsilon}$.

Deep Learning Software

Figure 7: PyTorch

CNN Architectures

AlexNet [2012, Alex Krizhevsky] The first deep learning model to win the ImageNet Large Scale Visual Recognition Challenge.

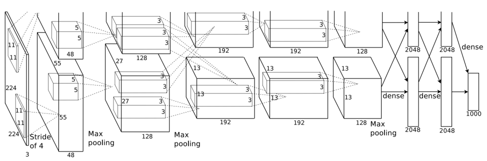


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure 8: AlexNet.

- 8 layers: 5 convolutional and 3 fully connected.
- ReLU activation.
- Dropout.
- Data augmentation.
- SGD with momentum.
- Total parameters: 60 million.

VGGNet [2014, Karen Simonyan and Andrew Zisserman]

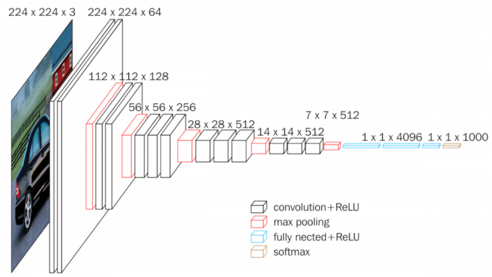


Figure 9: VGGNet.

- 19 layers: 16 convolutional and 3 fully connected.
- Small filters: 3×3.
- Max pooling.
- ReLU activation.
- Dropout.
- SGD with momentum.
- Total parameters: 138 million.

GoogLeNet [2014, Szegedy et al.]

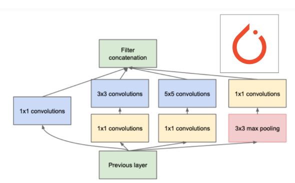


Figure 10: GoogLeNet.

- 22 layers.
- Inception module.
- Global average pooling.
- SGD with momentum.
- Total parameters: 5 million.

ResNet [2015, Kaiming He et al.]

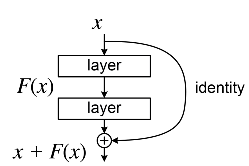
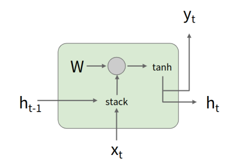


Figure 11: ResNet.

- 152 layers.
- Skip connections.
  $y = F(x, \{W_i\}) + x$.
- Batch normalization.
- SGD with momentum.
- Total parameters: 60 million.

Recurrent Neural Networks

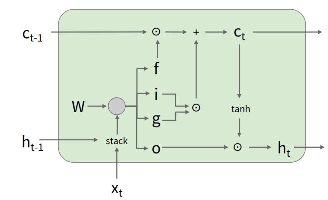
Recurrent Neural Networks (RNN)



$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t) = \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

Hidden state h_t. Output y_t.

Long Short-Term Memory (LSTM)



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

- Input gate i_t.
- Forget gate f_t.
- Output gate o_t.
- Gate gate (candidate) g_t.
- Cell state c_t.
- Hidden state h_t.

Gated Recurrent Unit (GRU)

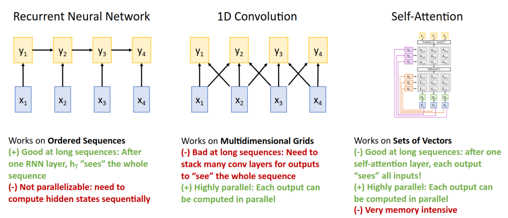
- Update gate:
  $z_t = \sigma(W_x z x_t + W_h z h_{t-1} + b_z)$.
- Reset gate:
  $r_t = \sigma(W_x r x_t + W_h r h_{t-1} + b_r)$.
- Candidate hidden state:
  $\tilde{h}_t = \tanh(W_x h x_t + W_h h(r_t \odot h_{t-1}) + b_h)$.
- Hidden state:
  $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$.

Sequence-to-Sequence Models

The best YouTube Video on Sequence-to-Sequence Models, including self-attention and transformers. Link: https://youtu.be/YAgjfMR9R_M?si=4Y5vQMbW5fqKONsX. I am too lazy to type the notes; thus, I did many screenshots of the slides. They are great if you know the context.

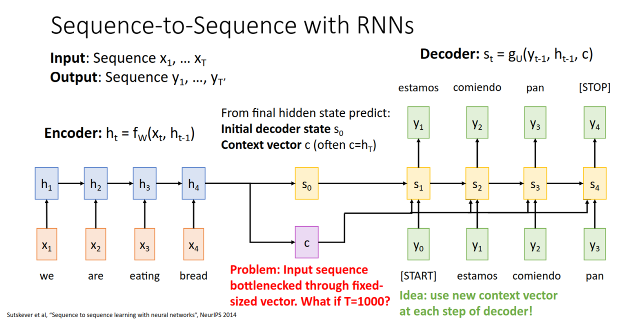
Three ways of processing sequences:

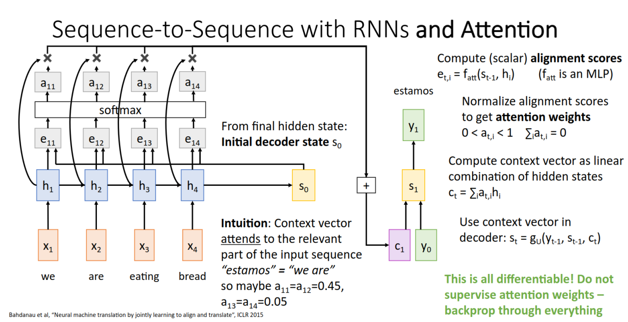
- Recurrent Neural Networks (RNNs).
- 1D Convolutional Neural Networks.
- Self-Attention Networks.

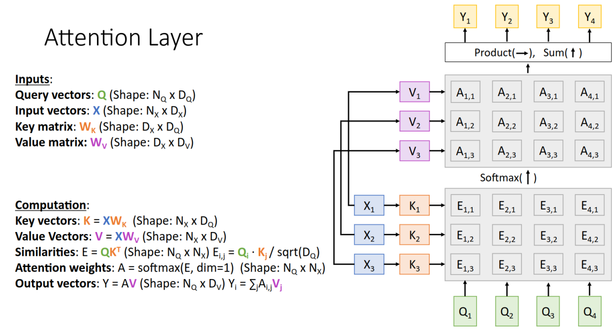


Attention Mechanism

Sequence-to-Sequence with RNNs







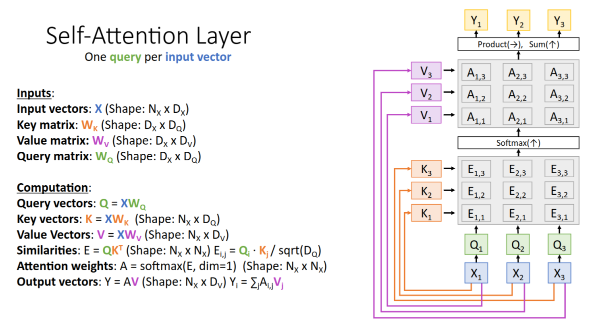


Figure 12: Self-Attention Layer (The self-attention layer is **Permutation Invariant/Equivariant**.).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

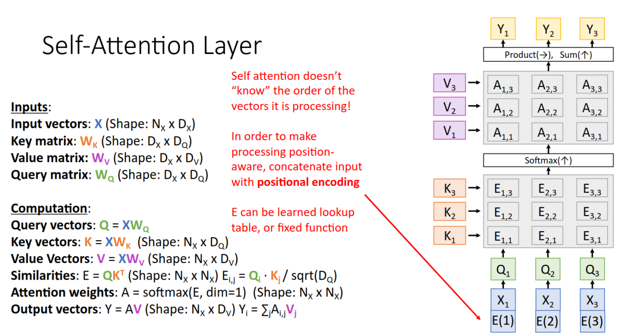


Figure 13: Self-Attention Layer with Positional Encoding.

Transformer

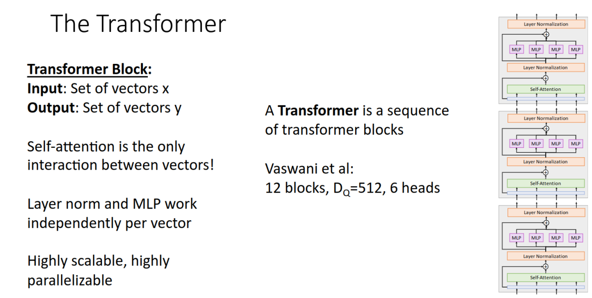


Figure 14: Transformer.

Object Detection

Loss Function

- **Bounding Box Regression**: $L_{\text{reg}} = \sum_i \sum_j \sum_k \sum_l \text{smooth}_{L1}(t_{ij}^k - t_{ij}^{k*})$.
- **Classification**: $L_{\text{cls}} = -\sum_i \log p_{ij}^{c*}$.
- **Total Loss**: $L = L_{\text{reg}} + \lambda L_{\text{cls}}$.
- **Sliding Window Detection**
- **Region Proposals**: Selective Search.
- **Region-based CNNs (R-CNN)**: Selective Search.
- **Fast R-CNN.**: Region of Interest (RoI) Pooling.
- **Faster R-CNN**: Region Proposal Network (RPN).
- **YOLO**: You Only Look Once.
- **SSD**: Single Shot MultiBox Detector.

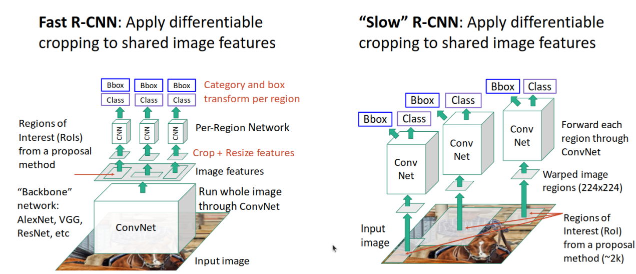


Figure 15: R-CNN

Intersection over Union (IoU) How can we compare the predicted bounding box with the ground truth bounding box?

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Non-Maximum Suppression (NMS) How can we remove redundant bounding boxes?

- Sort the bounding boxes by confidence.
- Pick the bounding box with the highest confidence.
- Remove all bounding boxes with IoU > threshold.
- Repeat until no more bounding boxes.

Mean Average Precision (mAP) How can we evaluate object detection?

- Precision: $\frac{\text{True Positives}}{\text{True Positives+False Positives}}$.
- Recall: $\frac{\text{True Positives}}{\text{True Positives+False Negatives}}$.

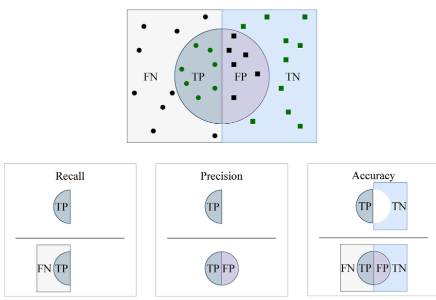


Figure 16: Mean Average Precision.

Semantic Segmentation

The goal of semantic segmentation is to label each pixel in an image with a corresponding class of what is being represented. The loss function (per pixel cross entropy) is:

$$L = \sum_i \sum_j \text{cross\_entropy}(p_{ij}, p^*_{ij}) = -\sum_i \sum_j p^*_{ij} \log p_{ij}$$

- **Mask R-CNN**: Faster R-CNN + FCN.
- **Segment Anything**

3D Vision

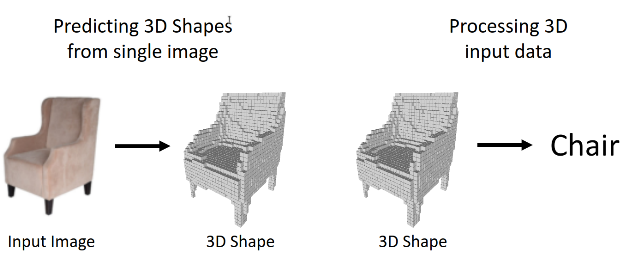


Figure 17: 3D Vision Problems

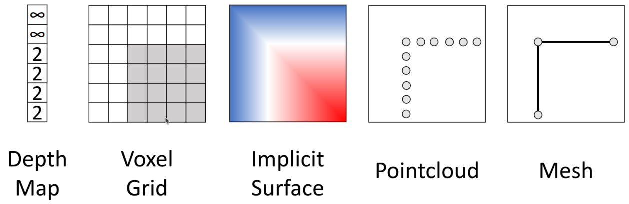


Figure 18: 3D Representation.

- **3D Convolutional Neural Networks**.
- **Volumetric CNNs**.
- **PointNet**.
- **PointNet++**.

Core Problems

- **Depth Estimation**
- **3D Reconstruction**
- **Pose Estimation**
- **Scene Understanding**

Video

Core Problems

- **Action Recognition**
- **Video Classification**
- **Video Segmentation**
- **Video Captioning**

Video Models

- **Single Frame CNN**
- **Late Fusion**
- **Early Fusion**
- **3D CNN/C3D**
- **Two-Stream Networks**
- **CNN + RNN**
- **Convolutional RNN**
- **Spatio-Temporal self-attention**
- **SlowFast Networks**

Generative Models

x: data, y: label.

- **Discriminative Models**: $P(y|x)$
  Assign a label to an input.
  Feature Learning (supervised learning).
- **Generative Models**: $P(x)$
  Generate new data.
  Feature Learning (unsupervised learning).
  Detect outliers.
- **Conditional Generative Models**: $P(x|y)$
  Generate data given a label.
  Assign laels to data, while rejecting outliers.

Autoregressive Models (Explicit Density Models)

Goal: Write down an explicit function for $p(x) = f(x, W)$. Given dataset $\{x_1, \dots, x_N\}$, maximize the likelihood:

$$\begin{aligned} W^* &= \max_W \prod_i p(x_i) \\ &= \max_W \prod_i f(x_i, W) \\ &= \max_W \sum_i \log f(x_i, W) \\ &= \min_W \sum_i -\log f(x_i, W) \end{aligned}$$

Example: PixelRNN and PixelCNN explicity parameterize density function with a neural network, so we can train to maximize likelihood of training data.

$$p_\theta(x) = \prod_i^n p_\theta(x_i | x_1, \dots, x_{i-1})$$

Variational Autoencoders (Implicit Density Models) VAE define an intractable density function that we cannot explicitly compute or optimize. But we will be able to directly optimize a **lower bound** on the likelihood of the data (density).

VAE jointly train an encoder $q_\phi(z|x)$ and a decoder $p_\theta(x|z)$ to maximize the variational lower bound on the log-likelihood of the data.

$$\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) || p(z))$$

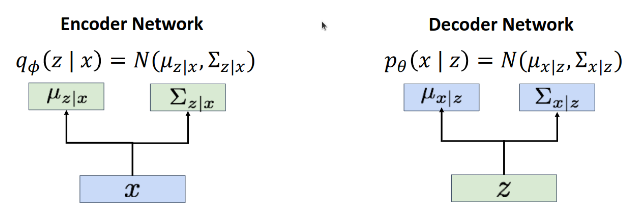


Figure 19: Variational Autoencoder.

Detailed Derivation of VAE

$$
\begin{aligned}
\log p_\theta(x) &= \log \frac{p_\theta(x|z)p(z)}{p_\theta(z|x)} \\
&= \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)} \\
&= \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \\
&= E_z[\log p_\theta(x|z)] - E_z[\log \frac{q_\phi(z|x)}{p(z)}] \\
&+ E_z[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}] \\
&= \underbrace{E_z[\log p_\theta(x|z)]}_{\text{Reconstruction Loss}} \underbrace{- \mathrm{KL}(q_\phi(z|x)||p(z))}_{\textbf{KL Divergence}} \\
&+ \underbrace{\mathrm{KL}(q_\phi(z|x)||p_\theta(z|x))}_{\textbf{cannot compute}}
\end{aligned}
$$

$$
\log p_\theta(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z))
$$

Reparameterization Trick

Motivation: During training, we need to compute the gradient of the loss function with respect to the parameters of the encoder. However, the direct sampling process $z \sim \mathcal{N}(\mu, \sigma^2)$ is not differentiable.

$$
\begin{aligned}
&z = \mu + \sigma \odot \epsilon \\
&\epsilon \sim \mathcal{N}(0, 1)
\end{aligned}
$$

Generative Adversarial Networks (Implicit Density Models) GANs are a framework for estimating generative models via an adversarial process. Different from VAE, GANs do not explicitly model the density function $p(x)$, but instead learn a generator function $G(z)$ that can generate samples.

GAN

$$
\begin{aligned}
\min_G \max_D L(D, G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] \\
+ \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]
\end{aligned}
$$