11. Neural Networks

오영민

Introduction

- **PPR: Projection Pursuit Regression**
- **Neural Networks**
- **Other layers: CNN, RNN, Attention mechanism**
- **Approximation Theory: Does Neural network converge to the global minimizer?**

PPR: Projection Pursuit Regression

- PPR has the form: $f(X) = \sum_{m=1}^{M} g_m(w_m^T X),\ X \in \mathbb{R}^p$
- PPR is an additive model, but in the derived features $V_m = w_m^T X$ rather than the inputs X.
- **Projection Pursuit:** $V_m = w_m^T X$ is the projection of X onto w_m, and we seek w_m so that the model fits well.
- $g_m(w_m^T \cdot)$ is called a ridge function in $\mathbb{R}^p$. It varies only in the direction defined by w_m.

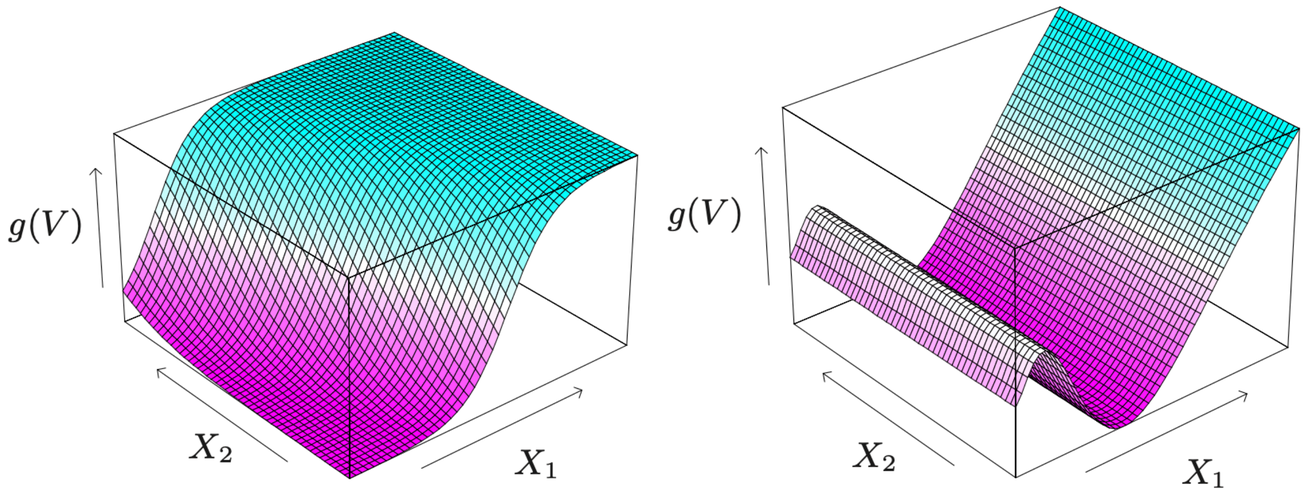


FIGURE 11.1. *Perspective plots of two ridge functions.*
(Left:) $g(V) = 1/[1 + \exp(-5(V - 0.5))]$, *where* $V = (X_1 + X_2)/\sqrt{2}$.
(Right:) $g(V) = (V + 0.1)\sin(1/(V/3 + 0.1))$, *where* $V = X_1$.

PPR: Projection Pursuit Regression

- **Example.** Suppose $f(X) = X_1 X_2$ where $X = (X_1, X_2)$. Then we can represent PPR by $f(X) = g_1(w_1^T X) + g_2(w_2^T X)$ where $g_1(t) = t^2/4$, $g_2(t) = -t^2/4$, $w_1^T = (1,1)$, $w_2^T = (1,-1)$.

- **Optimization.** Given training data (x_i, y_i), $i = 1,...,N$,

$$\min_{g_1,...,g_M,w_1,...,w_M} \sum_{i=1}^{N} \left[y_i - \sum_{m=1}^{M} g_m(w_m^T x_i) \right]^2$$

- **Optimization procedure. Forward stage-wise procedure**

For $m = 1,...,M$, adding a pair (w_m, g_m). (M is determined by AIC,BIC,… or Cross-validation.)

1. Choose w_m arbitrary and fitting g_m (scatter plot smoother) e.g. local regression, smoothing splines.

2. Fitting w_m s.t. minimize above squared error using Gauss-Newton search. e.g. quasi-Newton method.

3. Repeat 1,2 until converge.

Neural Networks

- **Kernel based model vs. Neural Network:**

- Kernel SVM and neural network were the models which could handle non-linear data.

- Kernel SVM transforms nonlinear data to RKHS to make the pattern of data linear hopefully and then applies linear SVM on it. Neural Network is different because the model itself is nonlinear.

- The success of kernel SVM + vanishing gradient problem in neural networks resulted in the winter of neural network.

- However, the problem of kernel SVM:

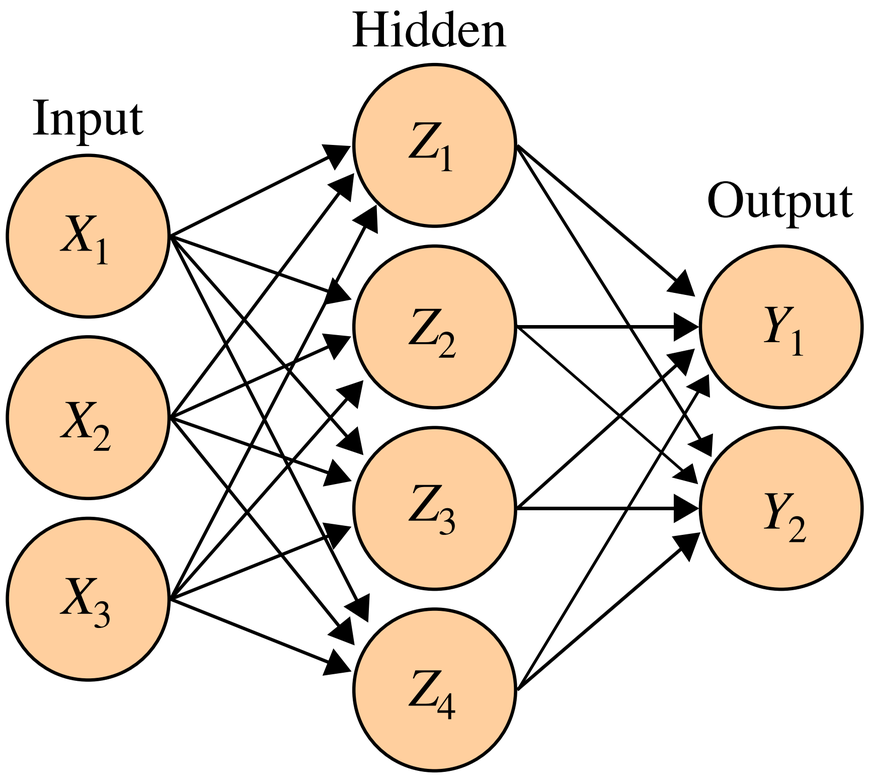
    1. Not knowing the suitable kernel type for various learning problem. But neural network is end-to-end and robust to hyper parameter.

    2. Kernel SVM could not handling big data.

- Due to vast improvements in computing power, larger training sets, and software: Tensorflow and PyTorch, neural networks take over kernel SVM gradually.

Neural Networks

- **Vanilla Neural Net** (single hidden layer back-propagation network, or single layer perceptron)



$$f(X) = g(\beta_0 + \sum_{m=1}^{M} \beta_m \sigma(\alpha_{0m} + \alpha_m^T X)),\ X \in \mathbb{R}^p$$

Parameters $(\beta_0, \beta_m, \alpha_{0m}, \alpha_m) : (M+1) + M(p+1)$-parameters.

$\sigma(\alpha_{0m} + \alpha_m^T X)$ are the **hidden unit** which is similar to **derived features** in PPR denoted as Z_m and M is the number of hidden units.

$\sigma(\cdot)$ is called an **activation function**. Populars are the sigmoids and ReLU. σ is typically non-linear, otherwise the model collapse to a linear model.

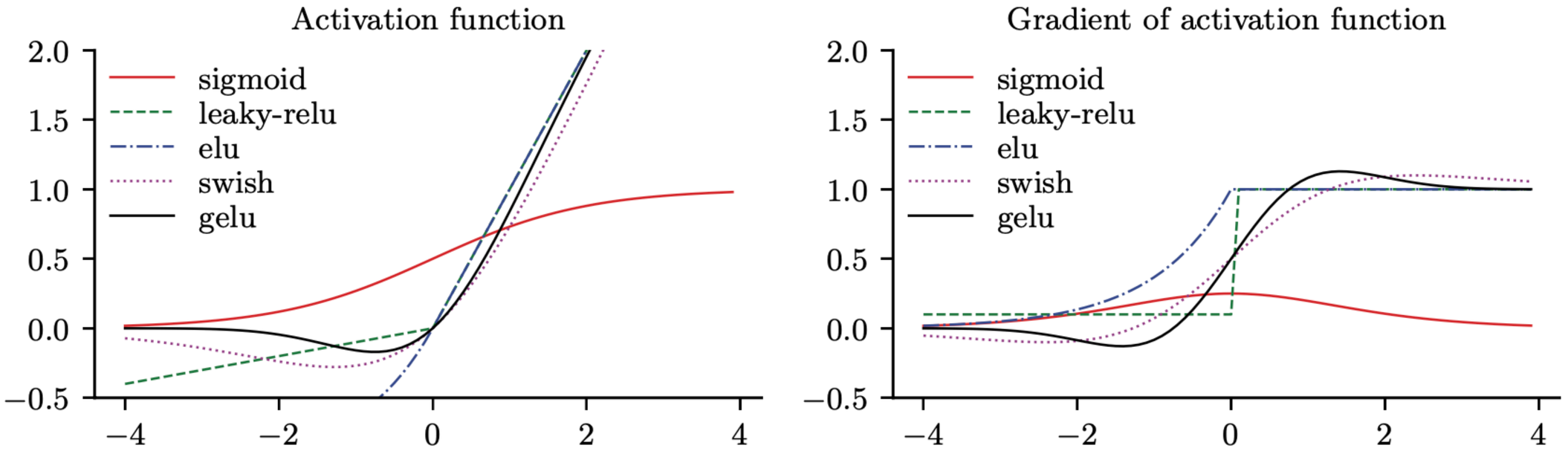
$g(x)$ is called an output function. In regression task, $g(x) = x$ and for classification task, g is softmax function.

Fit by minimizing squared error loss for regression, cross-entropy for classification using back-propagation.

Neural Networks

- **List of popular activation function.**

Name	Definition	Range
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$[0, 1]$
Hyperbolic tangent	$\tanh(a) = 2\sigma(2a) - 1$	$[-1, 1]$
Softplus	$\sigma_+(a) = \log(1 + e^a)$	$[0, \infty]$
Rectified linear unit	$\text{ReLU}(a) = \max(a, 0)$	$[0, \infty]$
Leaky ReLU	$\max(a, 0) + \alpha \min(a, 0)$	$[-\infty, \infty]$
Exponential linear unit	$\max(a, 0) + \min(\alpha(e^a - 1), 0)$	$[-\infty, \infty]$
Swish	$a\sigma(a)$	$[-\infty, \infty]$
GELU	$a\Phi(a)$	$[-\infty, \infty]$
Sine	$\sin(a)$	$[-1, 1]$



Neural Networks

- **Single hidden layer network regression vs. PPR:**

$$f(x) = \sum_m a_m \sigma(w_m^T x) \text{ vs. } f(x) = \sum_m g_m(w_m^T x)$$

- Now we only consider for regression problem with no intercept for notational simplicity.

- The specific Network architecture $(L, \mathbf{p})$ consists of L, called the number of hidden layers and width vector- $\mathbf{p} = (p_0, \ldots, p_{L+1}) \in \mathbb{N}^{L+2}$ recording the number of nodes in each layer.

- A L-hidden layer neural network $\tilde{f}$, is then any function of the form for input $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{p_0}$:

$$\tilde{f} : \mathcal{X} \to \mathbb{R},\ \mathbf{x} \to f(\mathbf{x}) = W_L \sigma W_{L-1} \ldots \sigma W_1 \mathbf{x},\ W_i \in \mathbb{R}^{p_{i+1} \times p_i}$$

Some Issues in Training Neural Networks.

- **Starting values:** Note that if the weights are near 0, then the operative part of the sigmoid is roughly linear, and hence the neural network collapses into an approximately linear model. Thus, starting values for weights are chosen to be random values near zero.

(Variance of the response in each layer) Let n be the number of connections of a response.

$$Var[y] = Var[\sum_i w_i x_i] = \sum_i Var[w_i x_i] = \sum_i Var[E[w_i x_i \,|\, x_i]] + E[Var[w_i x_i \,|\, x_i]]$$

$$= \sum_i Var[x_i E[w_i \,|\, x_i]] + E[x_i^2 Var[w_i \,|\, x_i]] = \sum_i Var[x_i E[w_i]] + E[x_i^2 Var[w_i]]$$

$$= \sum_i E[w_i]^2 Var[x_i] + Var[w_i] E[x_i^2] = \sum_i E[w_i]^2 Var[x_i] + Var[w_i]\big(Var[x_i] + E[x_i]^2\big) = \sum_i Var[x_i] Var[w_i] = \big(nVar(w)\big)\, Var(x)$$

This term isn't 0 for ReLU activation. If we use Batch Normalization, then we can guarantee that the above term is 0.

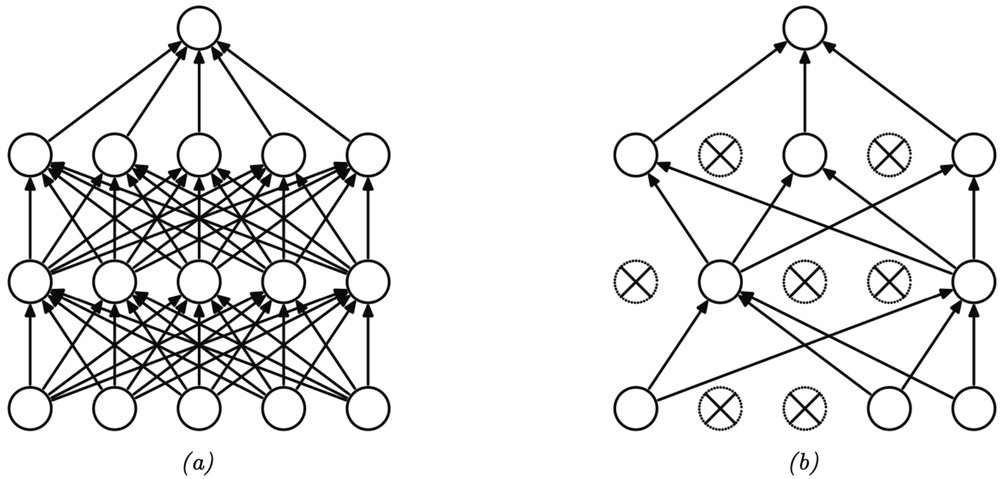
The weight must be divided by the $\sqrt{n}$

- **He initialization** (He at el 2015). $w \sim \mathcal{N}(0, \frac{2}{N_{in}})$ where N_{in} is # of input nodes. Usually use He initialization with ReLU.

Some Issues in Training Neural Networks.

- **Overfitting:** Often neural networks have too many weights and will overfit the data at the global minimum.
  - **Early stopping**: Training until the validation errors begin to increase.
  - **Weight decay**: Analogous to ridge regression, we add a penalty to the error function $R(w) + \lambda ||w||^2$. Typically the tuning-parameter $\lambda \geq 0$ is estimated by cross validation.
  - **Dropout**: In training time, multiply the W_{ijl} by $\epsilon_{il} \sim Bernoulli(1-p)$ where p is the drop probability on each step. The result is an **ensemble** of networks, each with slightly different sparse graph structures. At test time, we usually turn the dropout noise off. Thus, we should multiply the weights by $(1-p)$.

.



(a) (b)

Dropout layers

Some Issues in Training Neural Networks.

- **Number of Hidden Units and Layers:** With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data; with too many hidden units, the extra weights can be **shrunk toward 0** if appropriate regularization is used.
  - Since we choose λ by cross validation, Use cross validation to estimate the optimal number seems unnecessary.
  - Choice of the number of hidden layers is guided by background knowledge and experimentation.
  - **Shortcut connection (skip connection).** Skip connection allow the signal to skip one or more layers, which prevents it from being modified.

.

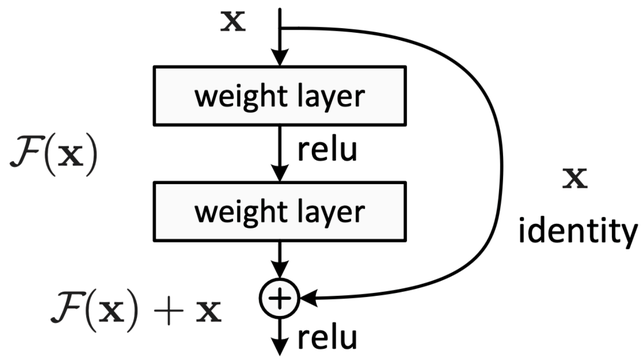


Figure 2. Residual learning: a building block.

Deep Residual Learning for Image Recognition (He at el 2015.)

$$y = \underbrace{\mathcal{F}(X, \{W_i\})}_{\text{Hidden units}} + \underbrace{W_s X}_{\text{Projection matrix which matches the dimensions of inputs and hidden units.}}$$

CNN: Convolution Neural Networks

- When dealing with image data, we can **apply the same weight matrix to each local patch** of the image, in order to reduce the number of parameters.

- If we "**slide**" this weight matrix over the image and add up the results, we get a technique known as convolution; in this case the weight matrix is often called a "kernel" or "filter".

- Note that the spatial size of the outputs may be smaller than inputs, due to boundary effects, although this can be solved by using **padding**.

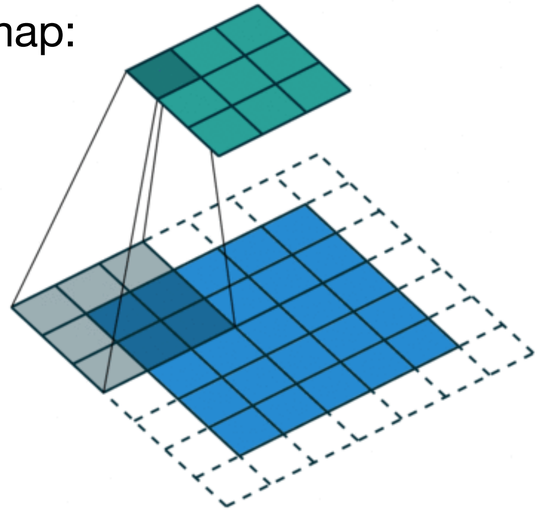
- We have (H, W, C) input image with D kernels, each size (h, w, C). Then the d-th output feature map:

$$z_{i,j,d} = \sum_{u=0}^{h-1} \sum_{v=0}^{w-1} \sum_{c=0}^{C-1} x_{i+u,j+v,c} w_{u,v,c,d} \quad \text{where } i = 0,1,...,(H-h+2P)/S,$$

$$j = 0,1,...,(W-w+2P)/S,$$

$$d = 0,1,...,D\ ,$$

P is the amount of padding, S is stride

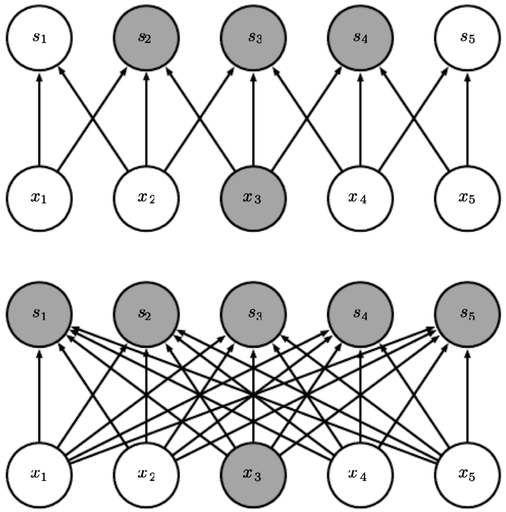


Convolution layers from https://github.com/vdumoulin/conv_arithmetic

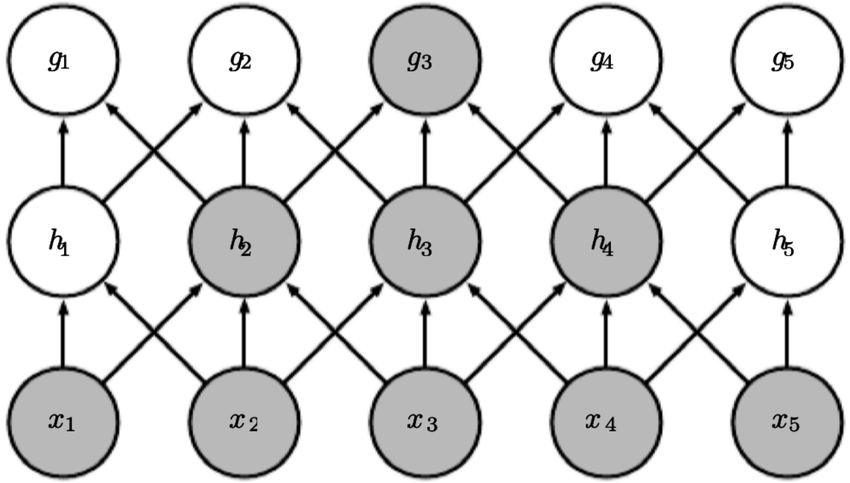
Advantages of CNN

- **Sparse interactions and parameter sharing.** Even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image. And each member of the kernel is used at every position of the input.

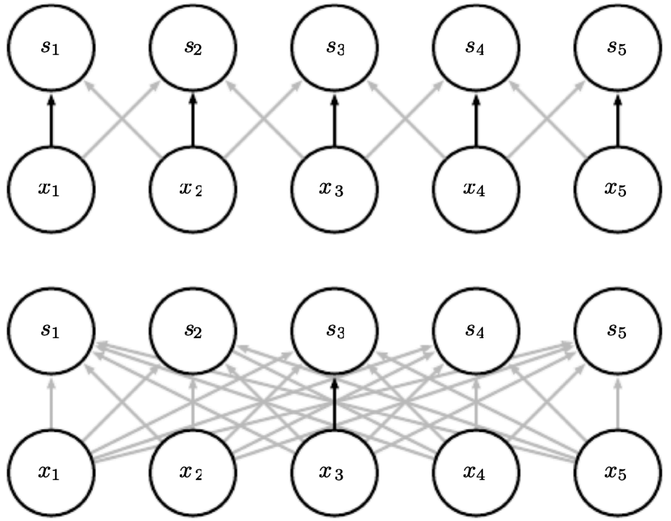
  ➡ Statistical efficiency and regularization effect.



Convolution later and Fully connected layer.



Sparse connectivity, viewed from below.

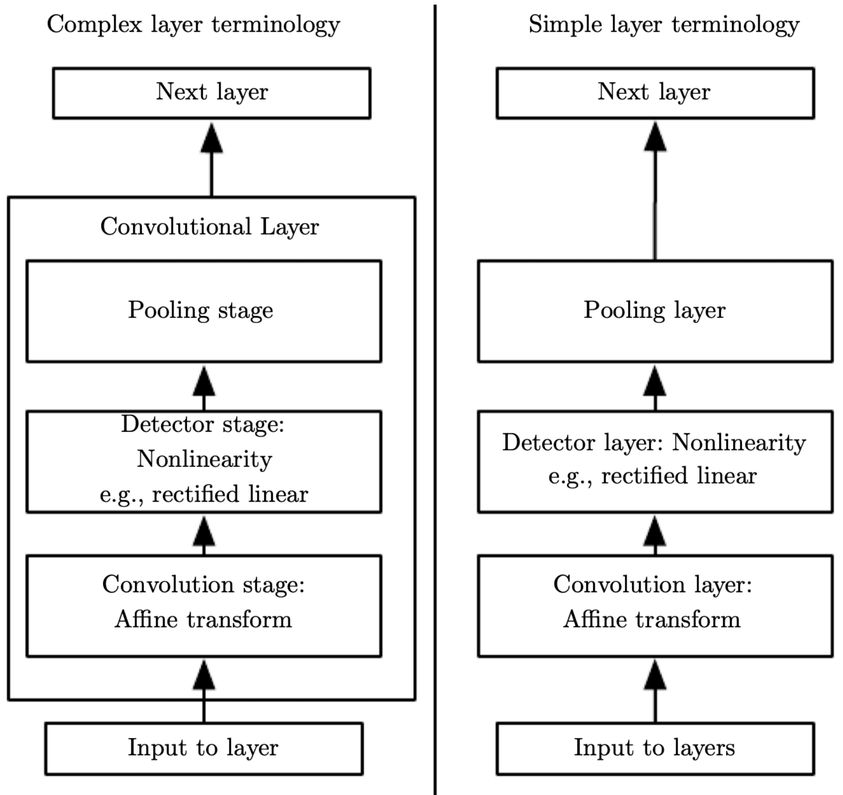


Parameter sharing. From Deep Learning , Ian Goodfellow

Advantages of CNN

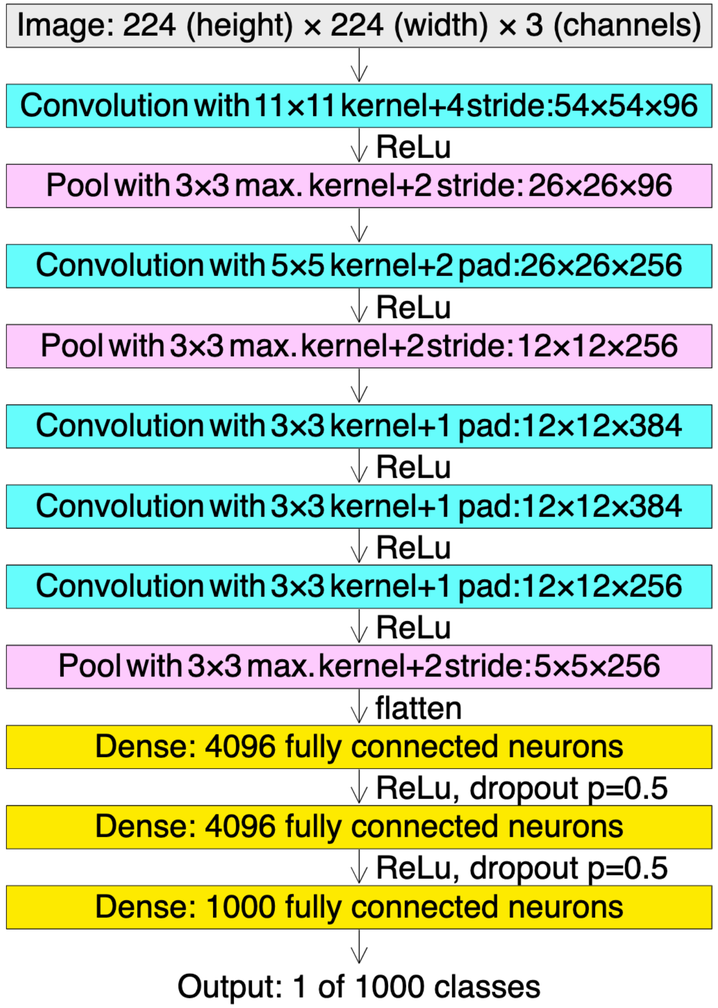
- **Shift equivalence.** Due to parameter sharing, if a pattern in the input shifts locations, the corresponding output activation will also shift.

- **Shift invariance.** If we translate the input by a small amount, the values of most of the pooled outputs do not change. This can be a very useful property if we care more about whether some feature is present than exactly where it is. Sift invariance can be obtained by using a **pooling layer**, which computes the maximum or average value in each local patch of the input.



Example of CNN

AlexNet

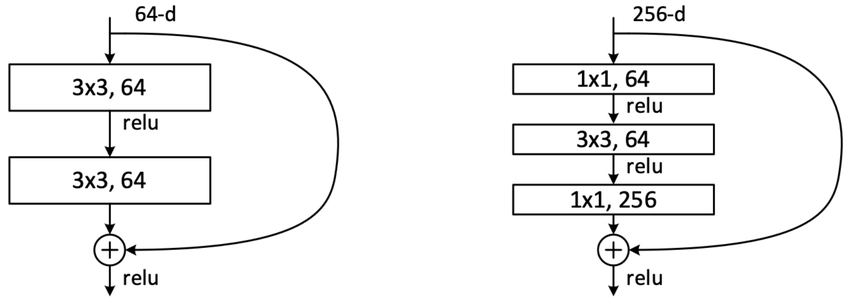


AlexNet(Krizhevsky et al, 2012) won the 2012 ImageNet competition

ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet(He et al., 2015) won ILSVRC 2015.



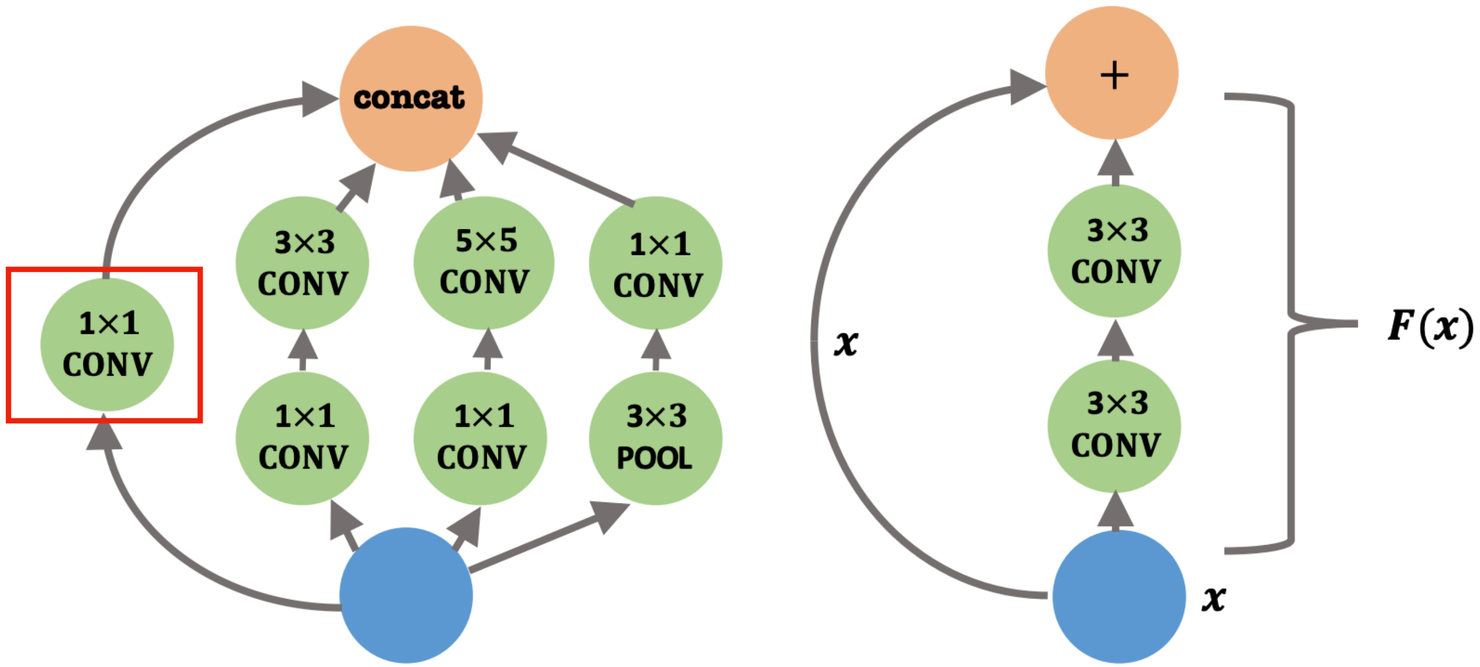
Building block of ResNet

Example of CNN

- **Inception module in GoogleNet(Szegedy et al., 2015):** The concatenation of information from filters with different sizes give the model great flexibility to capture spatial information.

- **Skip connections in ResNet(He et al., 2015).** Shortcut connection to preserve the gradient flow or Identity connection -> Alleviate the training problem. **Note.** However, ResNet-1000 has similar accuracy as ResNet-101 -> stochastic depth..

1x1 filter reduces # of parameters and speed-up the computation



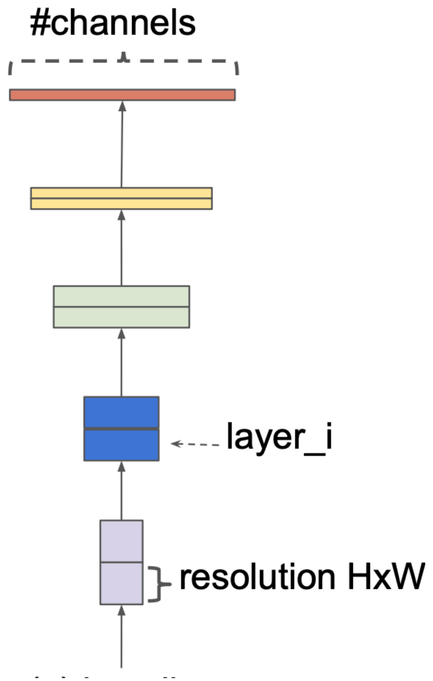
(a) "Inception" module (b) Skip connections

The building block from GoogleNet and ResNet

Example of CNN

- CNNs are often partitioned into multiple stages and all layers in each stage share the same architecture.

- Layer i can be defined as $Y_i = \mathcal{F}_i(X_i)$ where Y_i is output tensor, X_i is input tensor with (H_i, W_i, C_i). Then CNN $\mathcal{N}$ can be represented by $\mathcal{N} = \bigodot_{i=1,\ldots,s} \mathcal{F}_i^{L_i}(X)$ where $\mathcal{F}_i^{L_i}$ denotes layer $\mathcal{F}_i$ is repeated L_i times in stage i.



CNN architecture from EfficientNet(Tan et al., 2019)

How to choose $H_i, W_i, C_i, L_i, \mathcal{F}_i$?

1. **Hand-craft:** AlexNet(Krizhevsky et al., 2012), GoogleNet(Szegedy et al., 2015), ResNet(He et al., 2015),….
2. **Neural Architecture Search:** NasNet(Zoph et al, 2017), MnasNet(Tan et al., 2019), EfficientNet(Tan et al., 2019),…

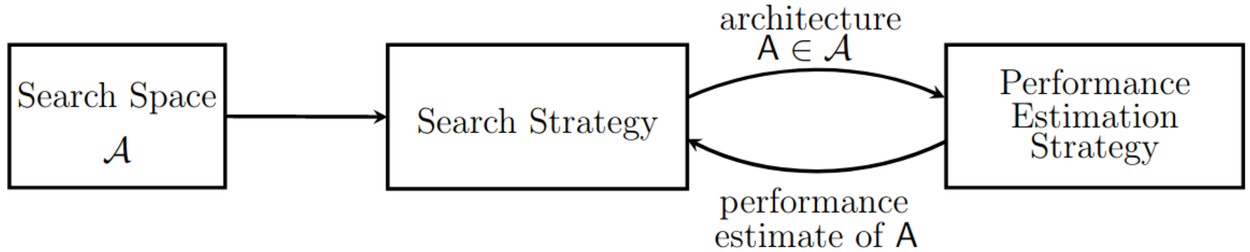


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

Neural Architecture Search: A Survey (Elsken et al., 2018)

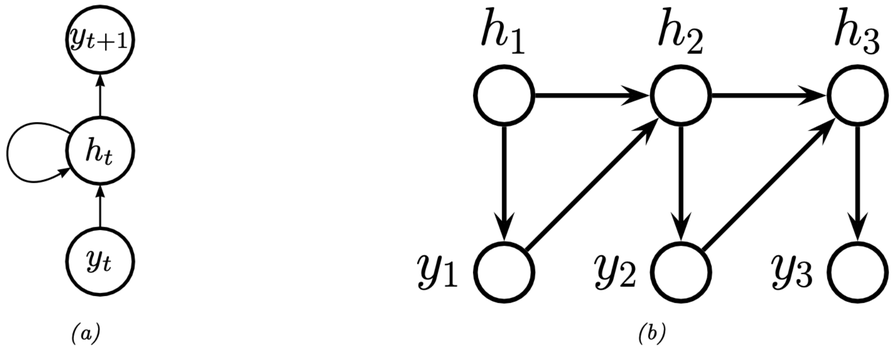
RNN: Recurrent Neural Networks

- We can make the model be stateful by augmenting the input x with the current state s_t, and then computing the output and the new state using some kind of function: $(y, s_t) = f(x, s_t)$;

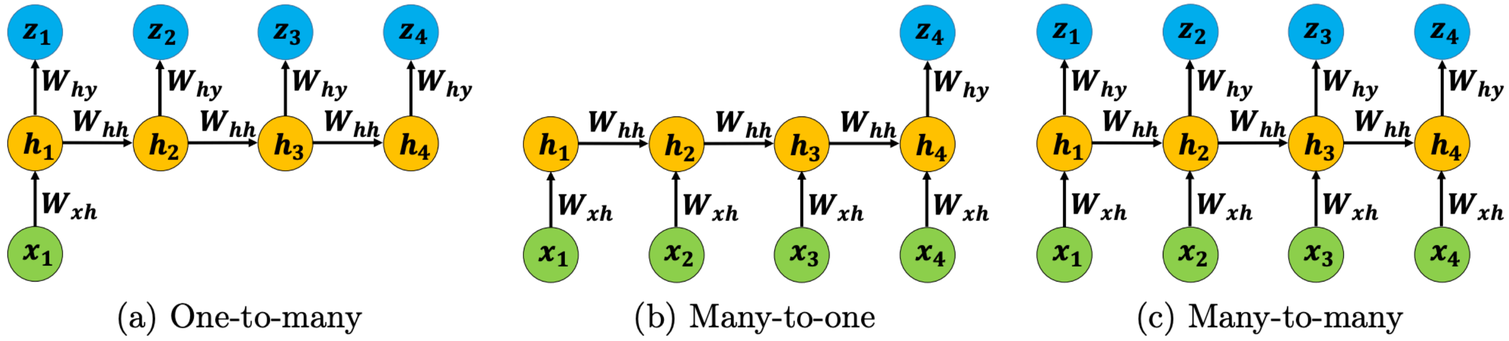
- In vanilla RNN, f is a simple MLP: $y_t = W^{hy} h_t$ where $h_t = sigm(W^{hx} x_t + W^{hh} h_{t-1})$.

Note. RNN also shares parameters. This makes it possible to extend and apply the model to examples of different forms and generalize across them. For example, consider the two sentences “I went to Nepal in 2009” and “In 2009, I went to Nepal.”

- For example, **One-to-One**: image captioning, **Many-to-One**: text sentiment classification, **Many-to-Many**: translation task.



(a) With self-loop. (b) Unrolled in time.



Vanilla RNNs with different inputs/outputs setting.

RNN: Recurrent Neural Networks

- **(Long term gradients)** Update the weights W^{hh} by getting the derivative of the loss at the very last time step L_T:

$$\frac{\partial L_T}{\partial W^{hh}} = \frac{\partial L_T}{\partial h_T} \cdot \frac{\partial h_T}{\partial h_{T-1}} \cdot \ldots \cdot \frac{\partial h_1}{\partial W^{hh}} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W^{hh}}$$

Since $$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial}{\partial h_{t-1}} sigm(W^{hx} x_t + W^{hh} h_{t-1}) = \underbrace{sigm(W^{hx} x_t + W^{hh} h_{t-1})(1 - sigm(W^{hx} x_t + W^{hh} h_{t-1}))}_{\text{Derivate of the sigmoid function is less then 1.}} W^{hh},$$

if T gets larger (i.e. longer time steps), the gradient will decrease in value and get close to 0.

➡ **Vanishing gradients problem**

Attention mechanism

- In fully connected layer, the hidden activations are followed by $\mathbf{Z} = \sigma(\mathbf{X}W)$ where $\mathbf{X} \in \mathbb{R}^{N \times p_0}$, $W \in \mathbb{R}^{p_0 \times p_1}$.

- Now, we can imagine a more flexible model in which the weights depend on the inputs $\mathbf{Z} = \sigma(\mathbf{X}W(X))$; This kind of multiplicative interaction is called **attention**.

- How to construct $W(X)$?

  - In non-parametric kernel based prediction: Given input $x \in \mathbb{R}^p$,

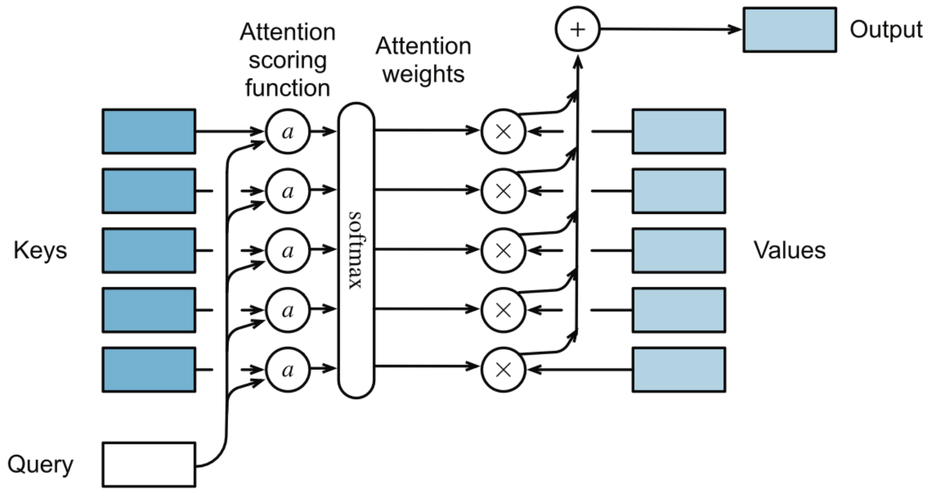
    1. we use a kernel to get a vector similarity scores $\alpha(x) = \{\alpha_1, \ldots, \alpha_N\} = \{K(x, x_1), \ldots, K(x, x_N)\}$

    2. Retrieve a weighted combination to compute the predicted output: $\hat{y} = \sum_{i=1}^{N} \alpha_i y_i$

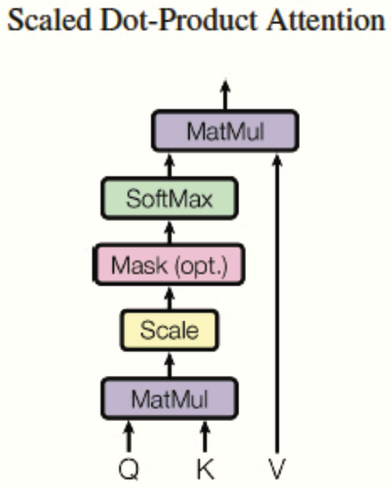
Attention mechanism

- By analogy to kernel based prediction, we can make a differentiable and parametric version of this as follow:
  - By using embedding matrix $W_k \in \mathbb{R}^{p_0 \times p_k}$, $W_v \in \mathbb{R}^{p_{L+1} \times p_v}$, $W_q \in \mathbb{R}^{p_k \times p_0}$, we can construct stored keys $\mathbf{K} = \mathbf{X}W_k \in \mathbb{R}^{N \times p_k}$, stored values $\mathbf{V} = \mathbf{y}W_v \in \mathbb{R}^{N \times p_v}$, and for given input $x \in \mathbb{R}^p$ to create query $q = W_q x \in \mathbb{R}^{p_k}$. The parameters to be learned are the three embedding matrices.

  Then we define the weighted output for query q to be $Attn(q, (k_{1:N}, v_{1:N})) = \sum_{i=1}^{N} \alpha_i(q, k_{1:N}) v_i$.



Attention layer



Matrix form

$\alpha_i(q, k_{1:N})$ is called the i-th **attention weight**; which satisfy $\alpha_i(q, k_{1:N}) \in [0,1]$ and $\sum_{i=1}^{N} \alpha_i(q, k_{1:N}) = 1$.

The attention weights can be computed from an **attention score** function $a(q, k_i) \in \mathbb{R}$ e.g. $a(q, k) = q^T k / \sqrt{p_k}$

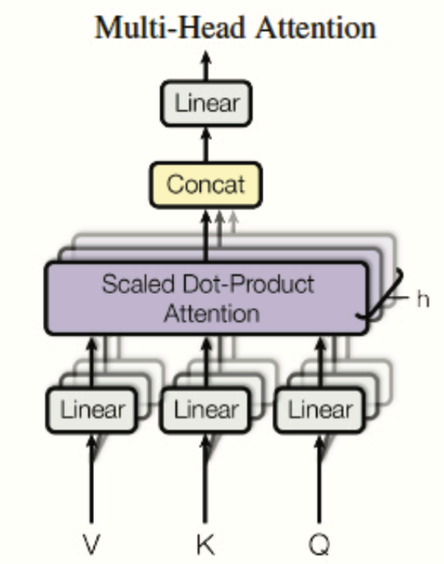
For input queries $\mathbf{Q} \in \mathbb{R}^{N \times p_k}$,

$$\mathbf{Z} = Attn(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{p_k}})\mathbf{V} \in \mathbb{R}^{N \times p_v}$$

Attention mechanism

- **Masked Attention.** For example, we might want to pad sequences to a **fixed length** (for efficient mini batching), in which case we should "**mask out**" the padded locations. This can be done by setting the attention score for the masked entries to a large negative number, such as -10^6, so that the corresponding softmax weight will be 0.

- **Multi-Head Attention**. To increase the flexibility of the model, we can use the pairs of projection matrices $\{(W_i^Q, W_i^K, W_i^V)\}_1^h$.



Let the i-th head be $\mathbf{h}_i = Attn(\mathbf{Q}W_i^Q, \mathbf{K}W_i^K, \mathbf{V}W_i^V) \in \mathbb{R}^{N \times p_v}$ where $W_i^Q \in \mathbb{R}^{p_k \times p_0}$, $W_i^K \in \mathbb{R}^{p_k \times p_0}$, $W_i^V \in \mathbb{R}^{p_0 \times p_v}$

Then the output of the Multi-Head Attention layer to be:

$$\mathbf{Z} = MHA(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(\mathbf{h}_1, \ldots, \mathbf{h}_h)W_o \in \mathbb{R}^{N \times p_0} \text{ where } W_o \in \mathbb{R}^{hp_v \times p_0}$$

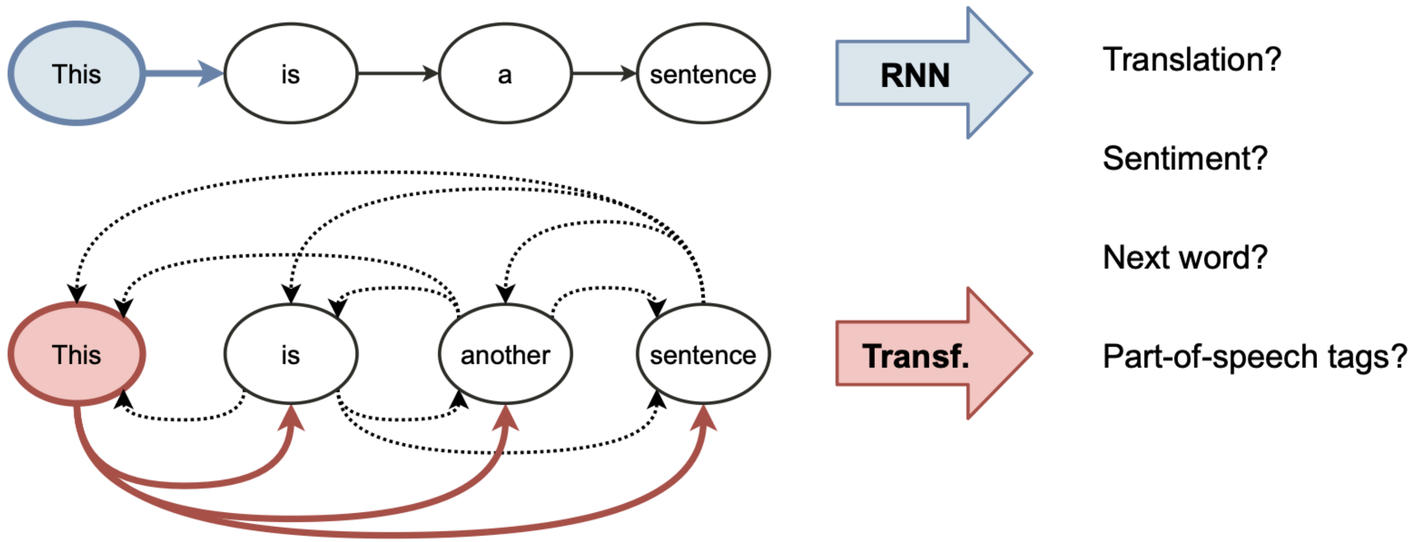
- When the output of one attention layer is used as input to another, the method is called **self attention**.

Transformer

- RNNs process one token at a time, so the embedding of the word at location t, z_t ,depends on the hidden state of the network, s_t, which may be a lossy summary of all the previously seen words.

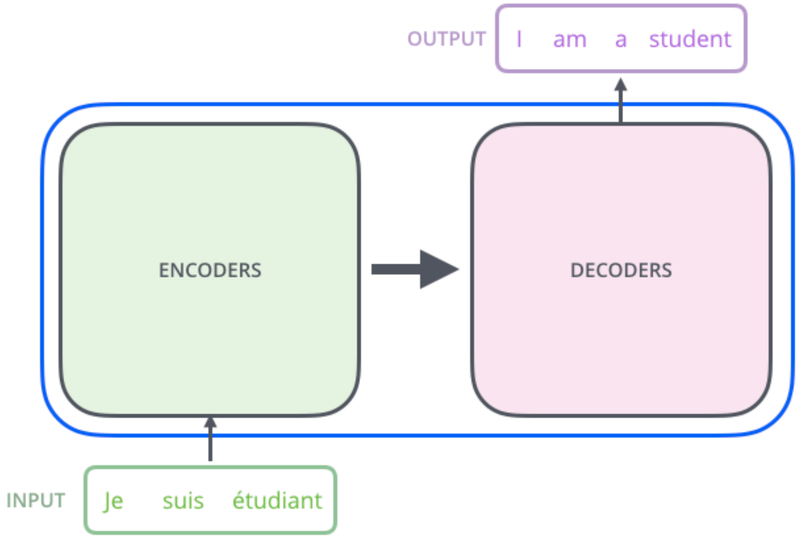
- We can create bidirectional RNNs so that future words can also affect the embedding of z_t, but this dependence is still mediated via the hidden state.

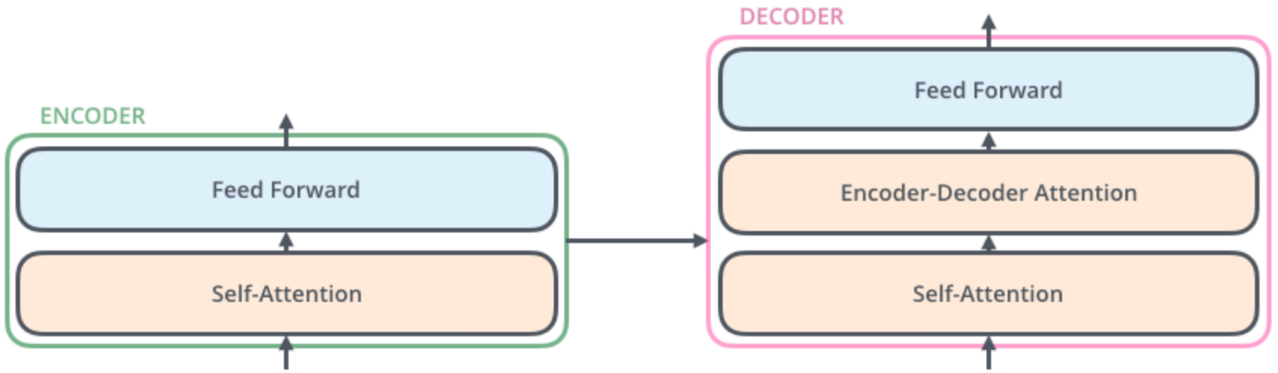
- **Transformer:** To compute z_t as a direct function of all the other words in the sentence, by using the **attention operator** rather than using hidden state. e.g. BERT(Bidirectional Encoder Representations from Transformers, Devlin et al, 2019), GPT(Generative pre-trained Transformers, Radford et al., 2017))



Visualizing the difference between an RNN and a transformer

Transformer





High level structure of the transformer.

- **Encoder. The encoder is composed of a stack $N = 6$ identical layers. Each layer has the form:**

1. (Self-Attention) The embedded input token(sentence) $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, ($\mathbf{x}_i \in \mathbb{R}^d$ is sub-vector or word) pass through an attention layer (typically multi-headed), and the output $\mathbf{Z}$ is added to the input $\mathbf{X}$ using a residual connection:

$$\mathbf{Z} = MHA(Q, K, V) + \mathbf{X}$$

2. (Feed-Forward) Above output is passed into layer normalization - MLP layer with skip connection and normalization:

$$\mathbf{Z} = W_2\sigma(W_1\mathbf{Z}) + \mathbf{Z}$$

Repeat N - times to get an encoding $\mathbf{H}_x$ of the output.

Transformer

- **Decoder. The encoder is also composed of a stack $N = 6$ identical layers. Each layer has the form:**

  1. (Self-Attention) Input tokes are all previously generated tokes, $\mathbf{y}_{1:t-1}$ and computes the encoding $\mathbf{H}_y$ using mask attention.

  2. (Encoder-Decoder Attention) Combine input $\mathbf{H}_x, \mathbf{H}_y$ to compute $\mathbf{Z} = Attn(\mathbf{Q} = \mathbf{H}_y, \mathbf{K} = \mathbf{H}_x, \mathbf{V} = \mathbf{H}_x)$, which compares the output to the input.

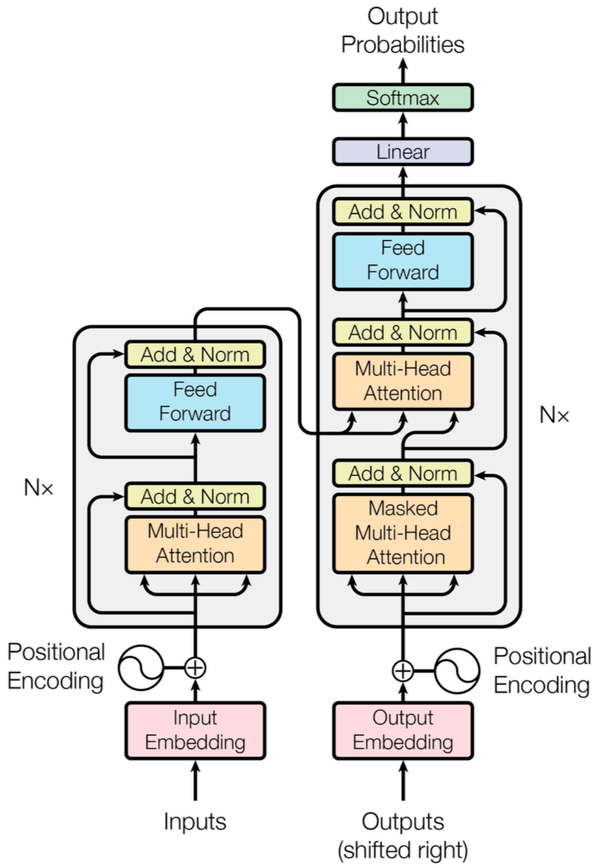
  3. (Feed Forward) Pass though the MLP layer.

  Repeat N - times.

- **Positional Encoding.**

Note. Attention operation pools information across all locations, so the transformer is **invariant to the ordering** of the inputs

Add positional encoding vectors $\mathbf{u}$ to input token $\mathbf{x}$: $\mathbf{x} + \mathbf{u}$



The Transformer - model architecture.

Approximation Theory view point

- **Metrics for complexity of the networks.**

  1. Maximum width: $p_{max} = \max_i(p_0, \ldots, p_{L+1})$, **2. Depth**: L, **3. The number of non-zero parameters** $\mathcal{N}$.

- The form of neural network we consider:

$$\mathcal{F}(L, \mathbf{p}, \mathcal{N}) = \left\{ \tilde{f}(\mathbf{x}) = W_L \sigma W_{L-1} \ldots \sigma W_1 \mathbf{x} : \sum_{j=1}^{L} \|W_j\|_0 \leq \mathcal{N} \right\}$$

- The capacity or expressive power of neural network is often characterized by $(L, \mathbf{p}_{max}, \mathcal{N})$.

- Define an **approximation error** $\epsilon_{Approx} = \sup_{f_\rho \in \mathcal{G}} \inf_{f \in \mathcal{F}(L, \mathbf{p}_{max}, \mathcal{N})} \|f - f_\rho\|_{L^p}$ where $\|f\|_{L^p}^p = \int_X |f|^p$

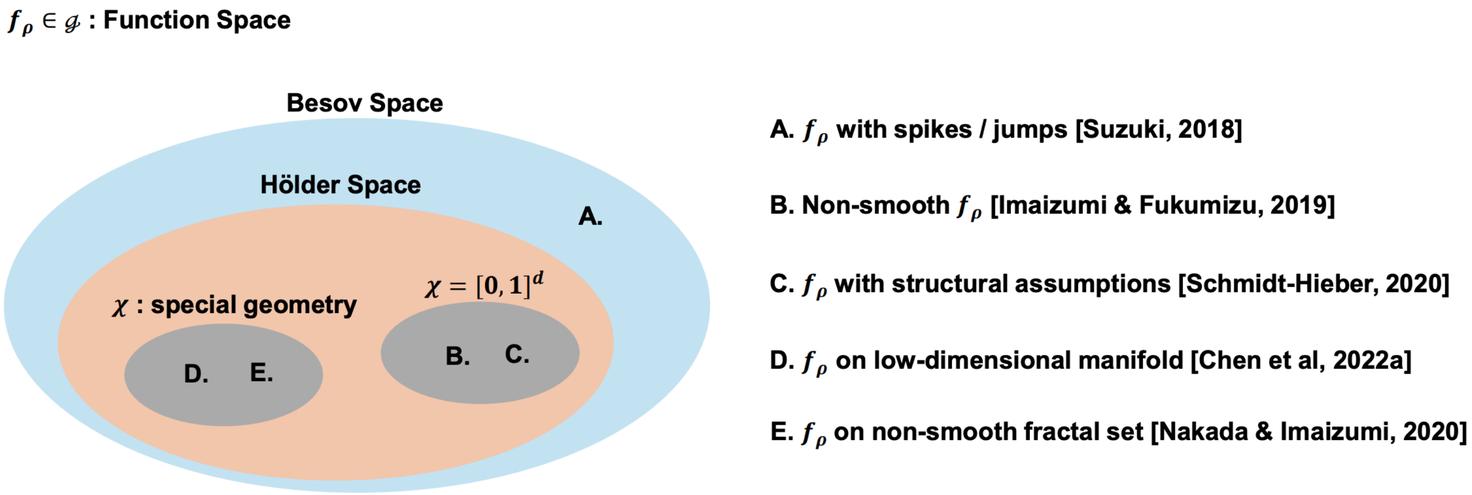
Q. How does the network architecture $(L, \mathbf{p}_{max}, \mathcal{N})$ scale in terms of ϵ_{Approx}?

Approximation Theory view point

- **(Universal approximation theorem)** Any continuous functions (i.e. $\mathscr{G} = \{\text{Continuous function on } \mathbb{R}^p\}$) can be approximated by a **single hidden network** with sigmoid activation at an arbitrary accuracy.

- However, achieving a good approximation may require an **extremely large number of hidden nodes,** which significantly increases the capacity of $\mathscr{F}$.

- If we restrict $\mathscr{G}$ by limiting capacity measured by the integrability of their Fourier transform, the approximation result is not affected by p i.e. neural network is very effective in dealing with high-dimensional data (Baron, 1994).

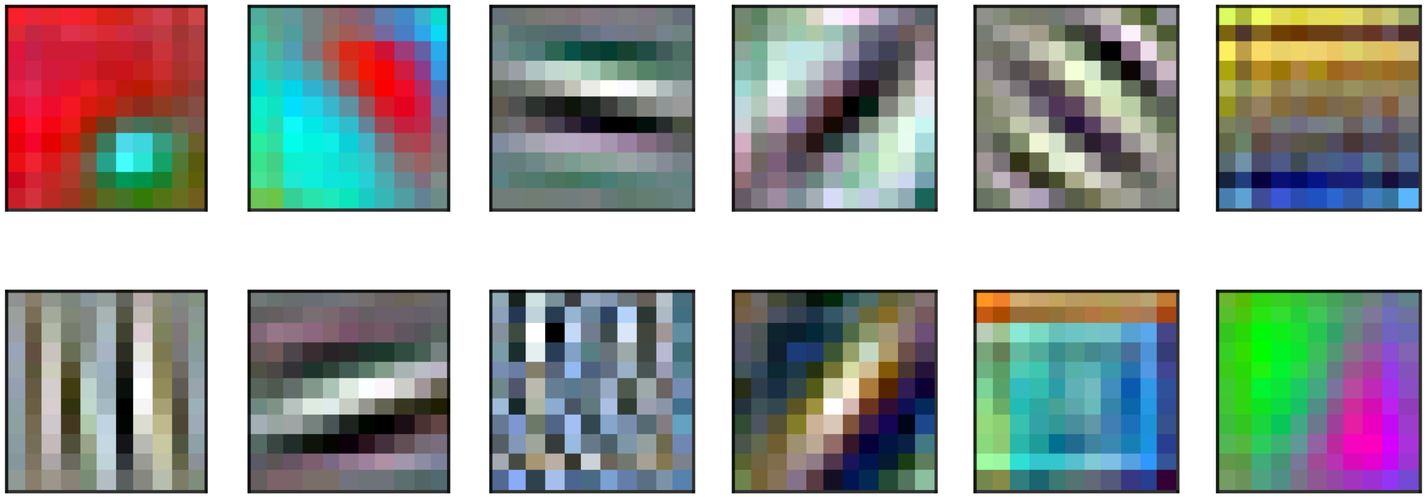


The different settings of function classes where neural networks exhibit superior adaptabilities over the classical estimators.

Approximation Theory view point

- **(Role of depth)** DNN expresses complicated nonlinearity through composing many nonlinear functions. This is the main difference from traditional statistical methods such as PPR.

  Intuition. In many real-world datasets such as images, there are different levels of features and lower-level features are building blocks of higher-level ones.



Visualization of trained filters in the first layer of AlexNet.

It has been shown that the **expressive power of DNN grows exponentially against** L. Under the same # of nodes, deep networks can represent exponentially more piece-wise functions than their single-layer counterparts can do.(Pascanu et al., 2013)

Approximation Theory view point

- **Q.** How are these approximation results used in obtaining statistical guarantees of neural networks under **“noisy-observations”?**

- Let $(X, y) \sim \rho$ and ρ_X be the marginal distribution of X. In regression task, we assume that the noisy dataset $\{x_i, y_i\}_1^N$ are generate from the non-parametric regression model: $y_i = f_\rho(x_i) + \epsilon_i,\ \forall i$ where $\mathbb{E}(\epsilon_i \,|\, x_i) = 0$.

- Under the squared error loss, we know that the minimizer of the population point-wise : $f_\rho(x) = \mathbb{E}(y \,|\, X = x)$.

- The minimizer of the population in our neural network class: $f = \underset{f \in \mathcal{F}(L, \mathbf{p}, \mathcal{N})}{\arg\min}\ \mathbb{E}[(y_i - f(x_i))^2]$

- However, since ρ is unknown, we solve the empirical minimizer: $\hat{f}_n = \underset{f \in \mathcal{F}(L, \mathbf{p}, \mathcal{N})}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2$

Approximation Theory view point

- Now we assume that $\hat{f}_n$ is obtainable, ignoring the optimization process. Then, the estimation error can be decomposed as:

$$\mathbb{E}_{X\sim\rho_X}\left[(\hat{f}_n(X) - f_\rho(X))^2\right] \leq \underbrace{\frac{\text{Compexity Measuer of } \mathscr{F}}{n}}_{\text{e.g. VC-dimension, pseudo-dimension, ...}} + \underbrace{\|f - f_\rho\|_\infty^2}_{\text{The approximation error which determined by } \mathscr{F}.}$$

- **The trade-off between the approximation and the complexity measure.** The richer $\mathscr{F}$ becomes the finer the approximation result we get. Nonetheless, the arbitrary increase in $\mathscr{F}$ eventually leads to the increase of the bound in the estimation error.