Deep Learning

Tianhang Zheng

https://tianzheng4.github.io

Deep Learning

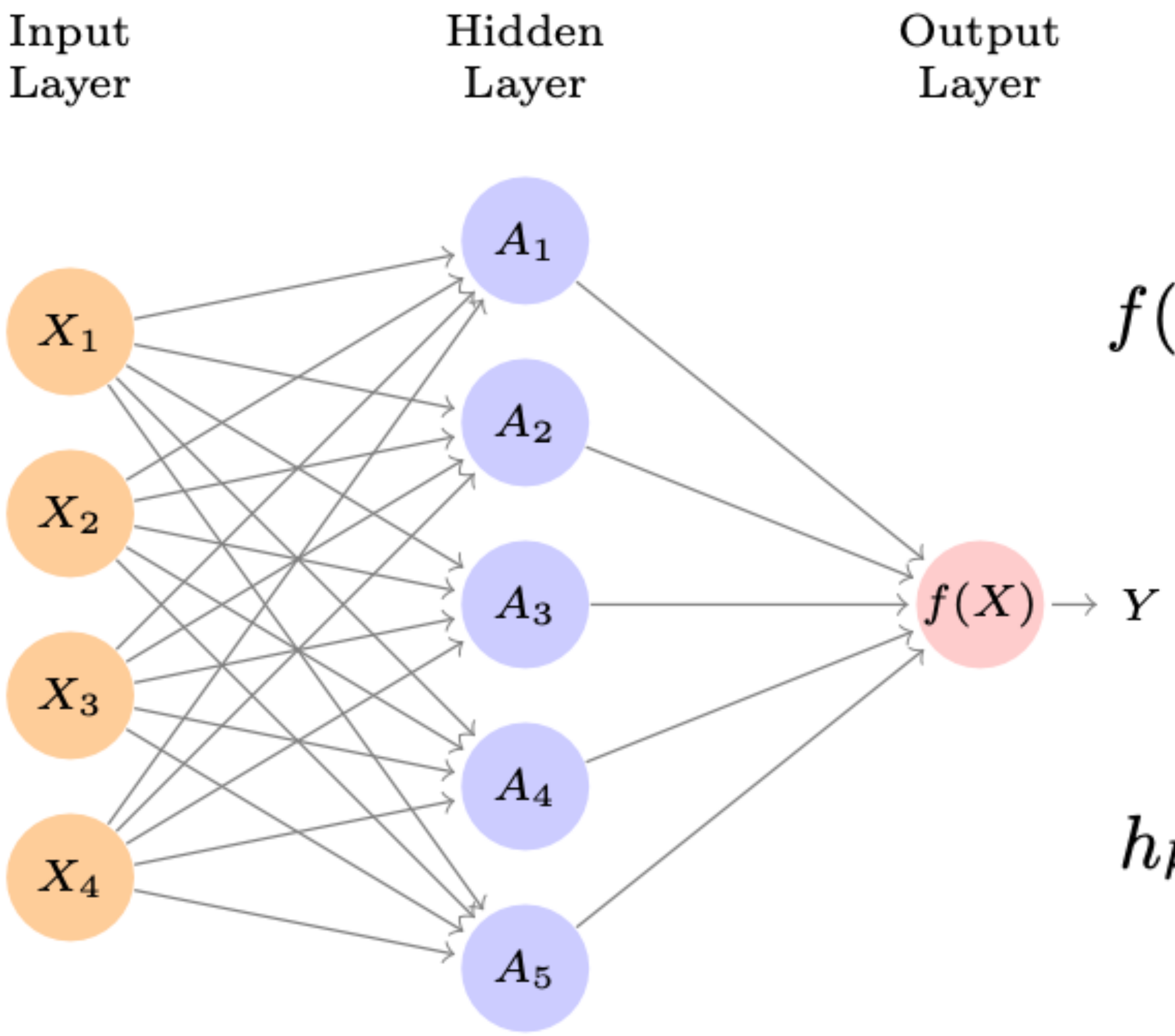
People begin to study neural networks starting from 1980

Deep Learning

Deep learning becomes very popular starting from 2012

ImageNet classification with deep convolutional neural networks (NeurIPS 2012) shows that deep neural networks can outperform traditional machine learning techniques by a large margin!

Single Layer Neural Network



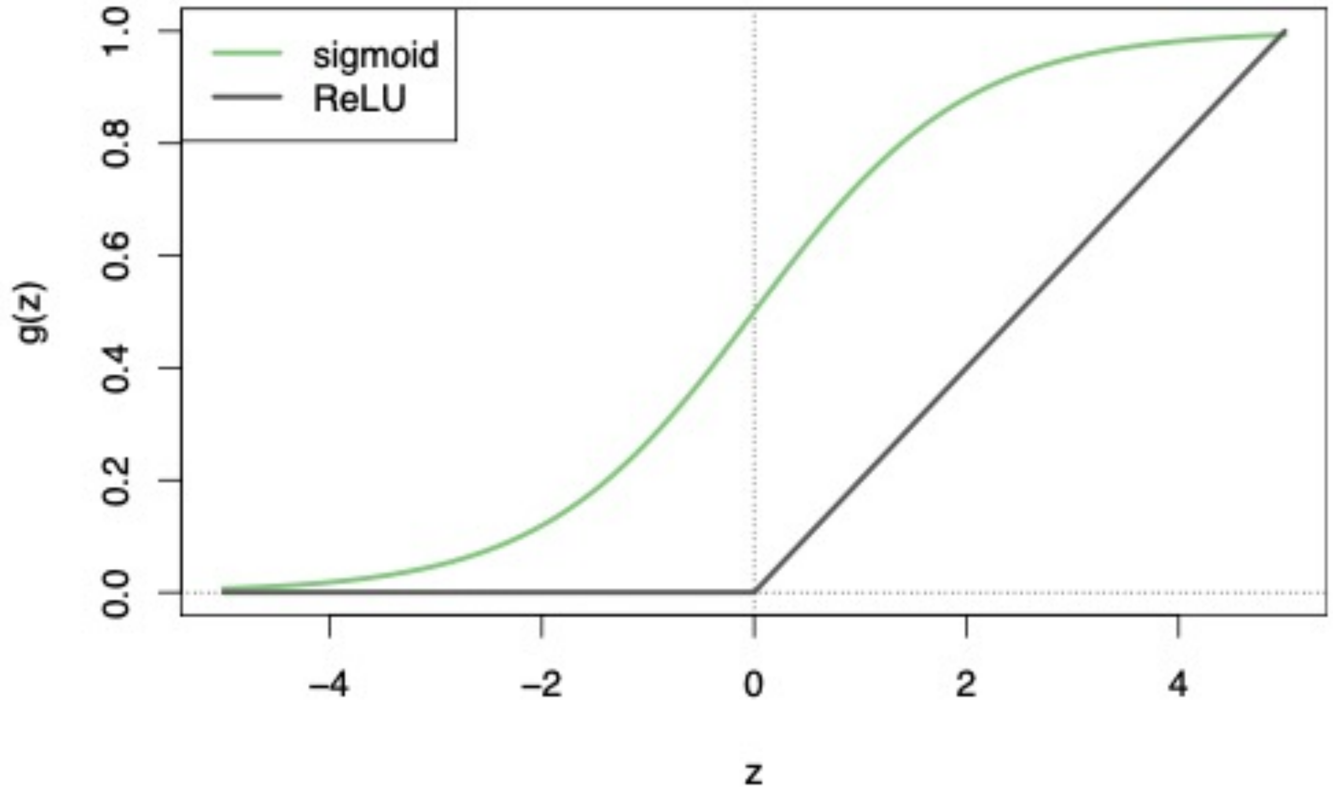
$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

$$h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

Activation Function

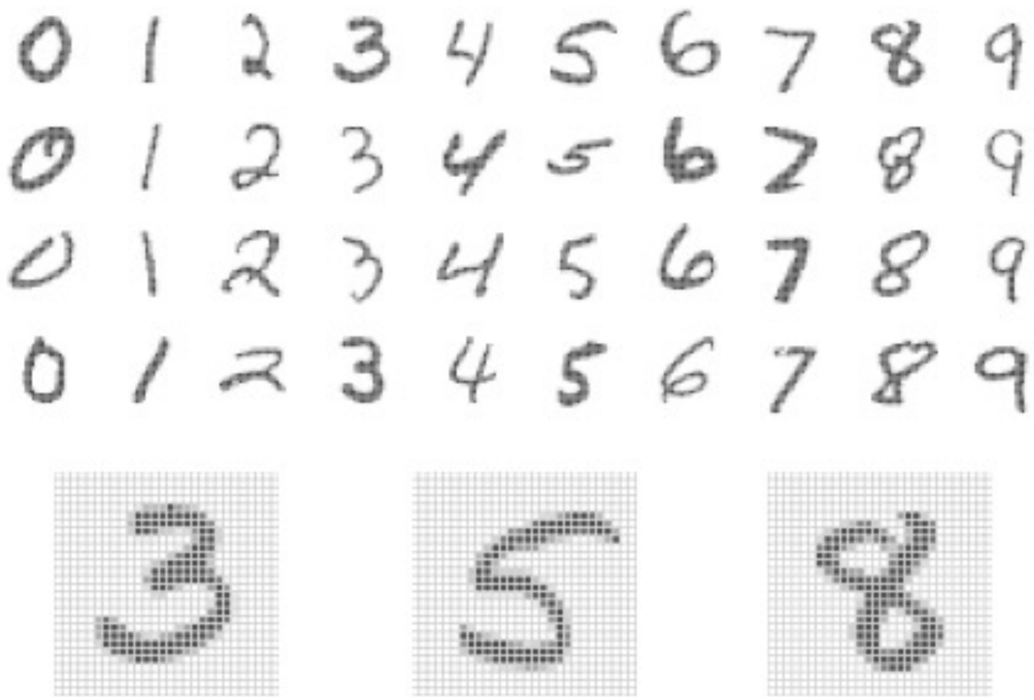
$$g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

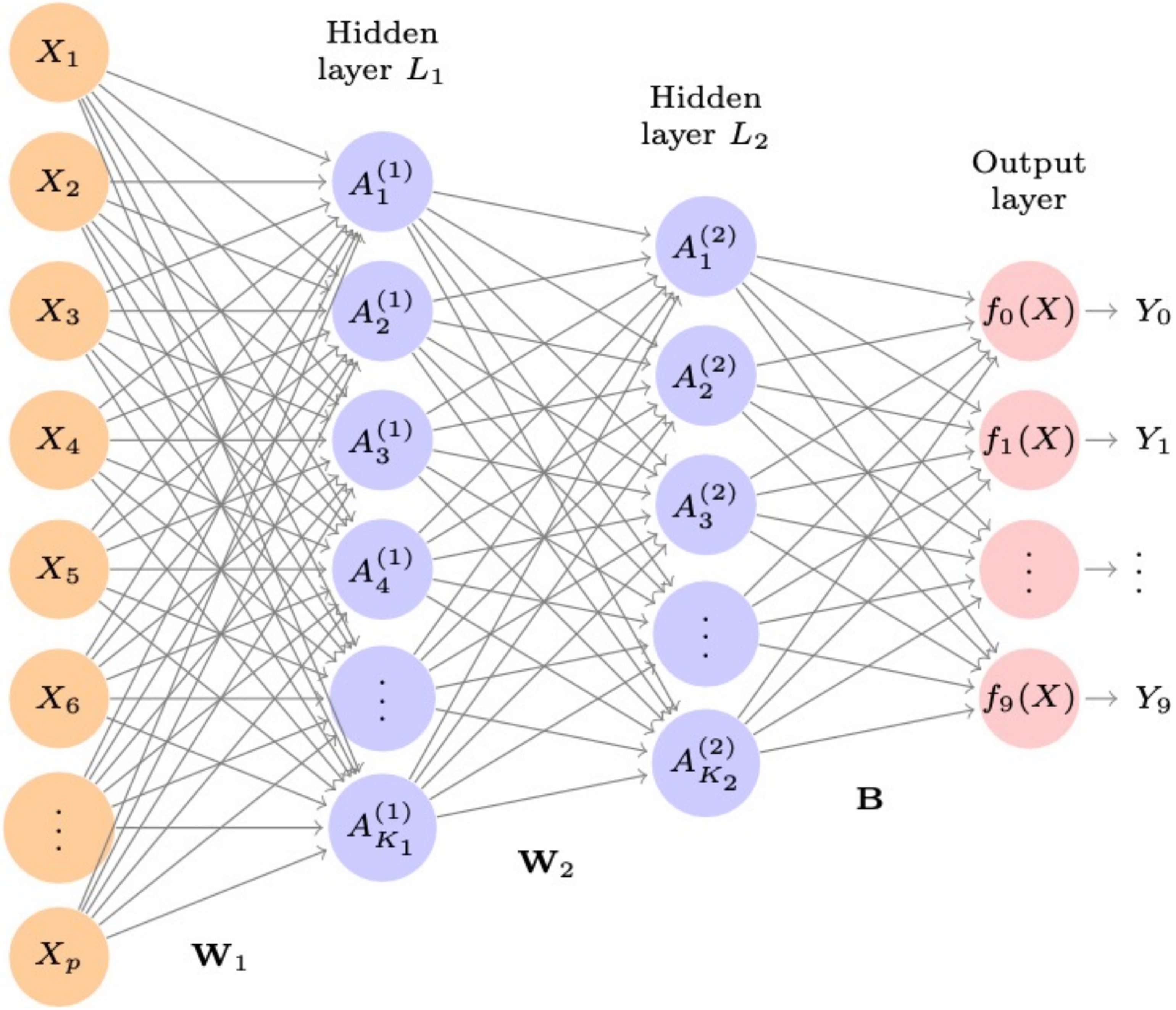
Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.



Example: MNIST Digits

Build a two-layer network with 256 units at first layer, 128 units at second layer, and 10 units at output layer.





Hidden layer L_1
Hidden layer L_2
Output layer
X_1
X_2
X_3
X_4
X_5
X_6
$\vdots$
X_p
$A_1^{(1)}$
$A_2^{(1)}$
$A_3^{(1)}$
$A_4^{(1)}$
$\vdots$
$A_{K_1}^{(1)}$
$A_1^{(2)}$
$A_2^{(2)}$
$A_3^{(2)}$
$\vdots$
$A_{K_2}^{(2)}$
$f_0(X) \to Y_0$
$f_1(X) \to Y_1$
$\vdots \to \vdots$
$f_9(X) \to Y_9$
$\mathbf{W}_1$
$\mathbf{W}_2$
$\mathbf{B}$

Softmax Output

Let $Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}$, $m = 0, 1, \ldots, 9$ be 10 linear combinations of activations at second layer.

Softmax output:

$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}.$$

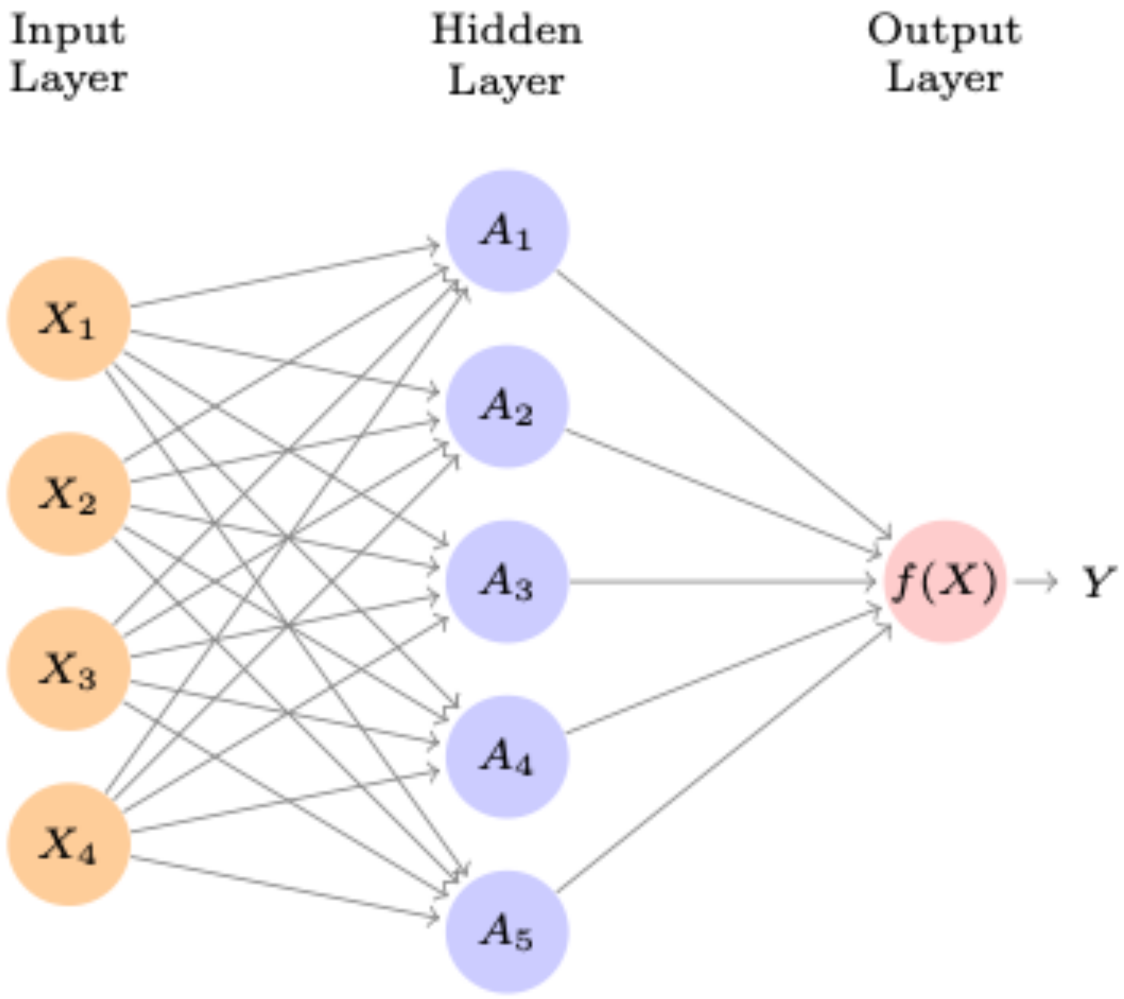
Cross-Entropy Loss

Negative multinomial log-likelihood:

$$-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log(f_m(x_i))$$

y_{im} is 1 if true class for observation i is m, else 0

Train Neural Networks



$$\underset{\{w_k\}_1^K,\ \beta}{\text{minimize}} \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

The objective is non-convex

$$f(x_i) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\right)$$

Train Neural Networks

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

Start with a guess θ^0 for all the parameters in θ, and set $t = 0$.

Iterate until the objective $R(\theta)$ fails to decrease:

(a) Find a vector δ that reflects a small change in θ, such that $\theta^{t+1} = \theta^t + \delta$ *reduces* the objective; i.e. $R(\theta^{t+1}) < R(\theta^t)$.

(b) Set $t \leftarrow t + 1$.

Train Neural Networks

How to find a direction δ that points downhill?

$$\nabla R(\theta^t) = \frac{\partial R(\theta)}{\partial \theta}\Big|_{\theta=\theta^t}$$

The gradient points uphill, so our update is $\delta = -\rho \nabla R(\theta^t)$

$$\theta^{t+1} \leftarrow \theta^t - \rho \nabla R(\theta^t)$$

Gradients and Backpropagation

Backpropagation uses the chain rule for differentiation:

$$
\begin{aligned}
\frac{\partial R_i(\theta)}{\partial \beta_k} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k} \\
&= -(y_i - f_\theta(x_i)) \cdot g(z_{ik}). \\
\frac{\partial R_i(\theta)}{\partial w_{kj}} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}} \\
&= -(y_i - f_\theta(x_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}.
\end{aligned}
$$

Optimizers

Stochastic Gradient Descent (SGD)

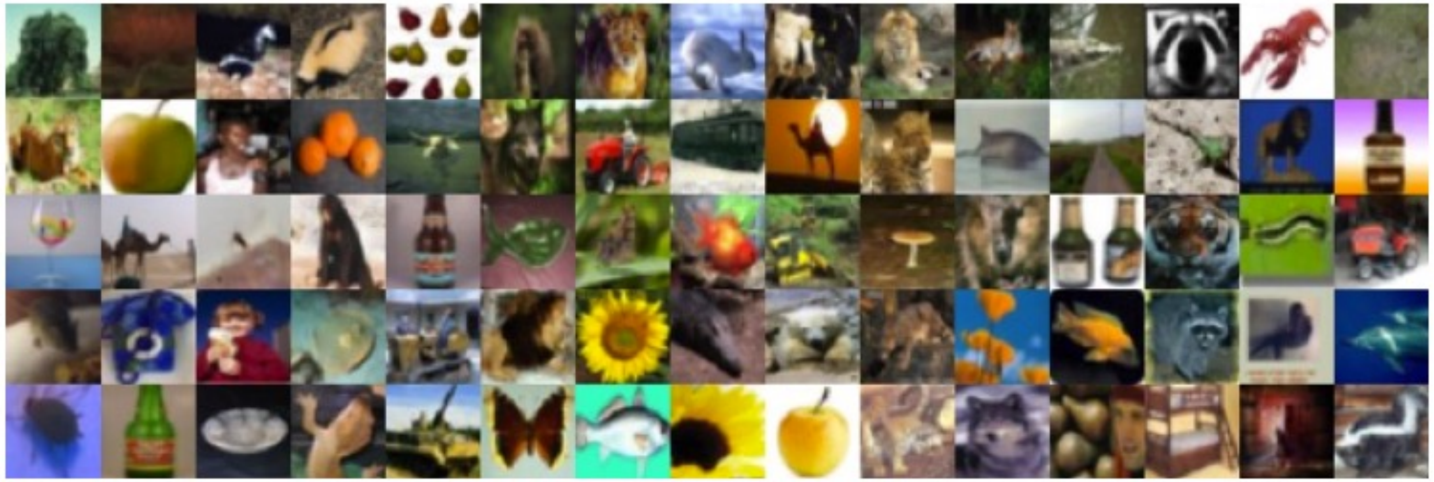
Stochastic Gradient Descent with Momentum

Adam Optimizer

Why Neural Network?

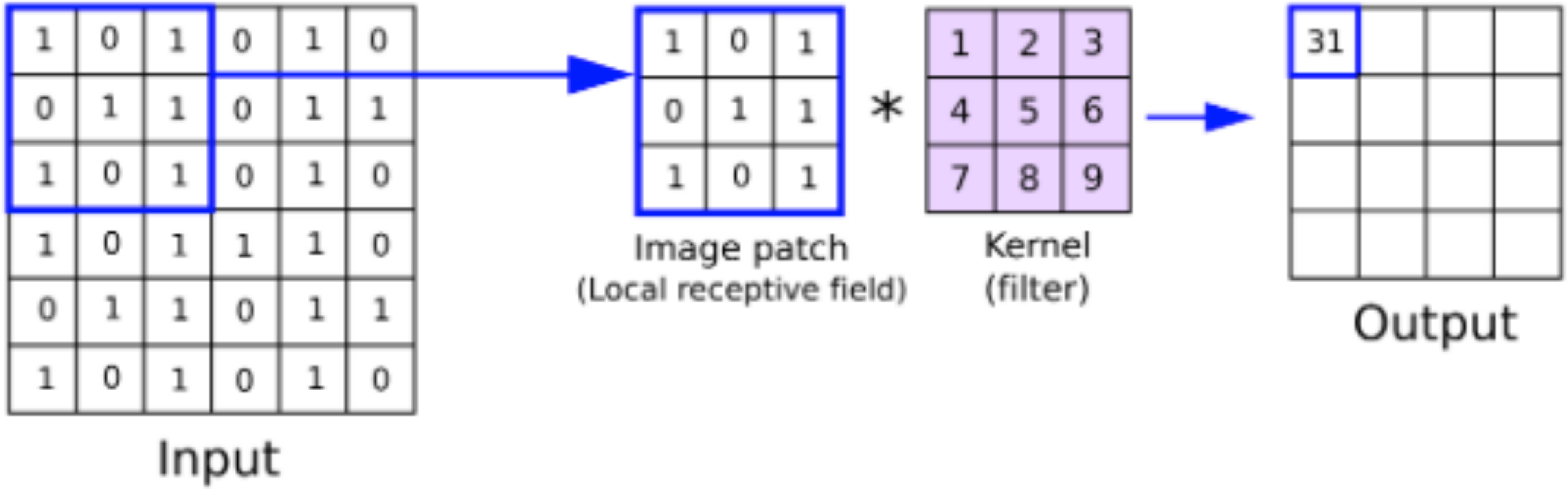
Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

Convolutional Neural Network



A commonly used network architecture for classifying images

Convolutional Kernel



Convolutional Kernel

Identity

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Convolutional Kernel

Sharpen

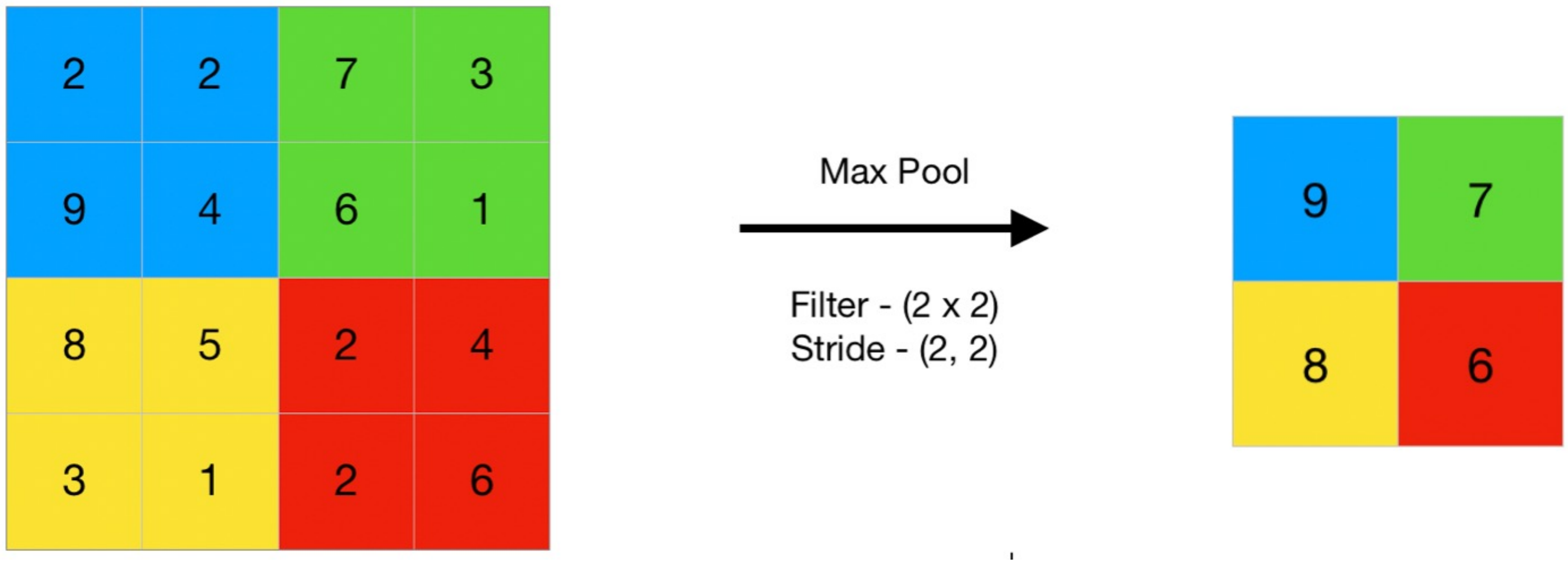
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Box Blur

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

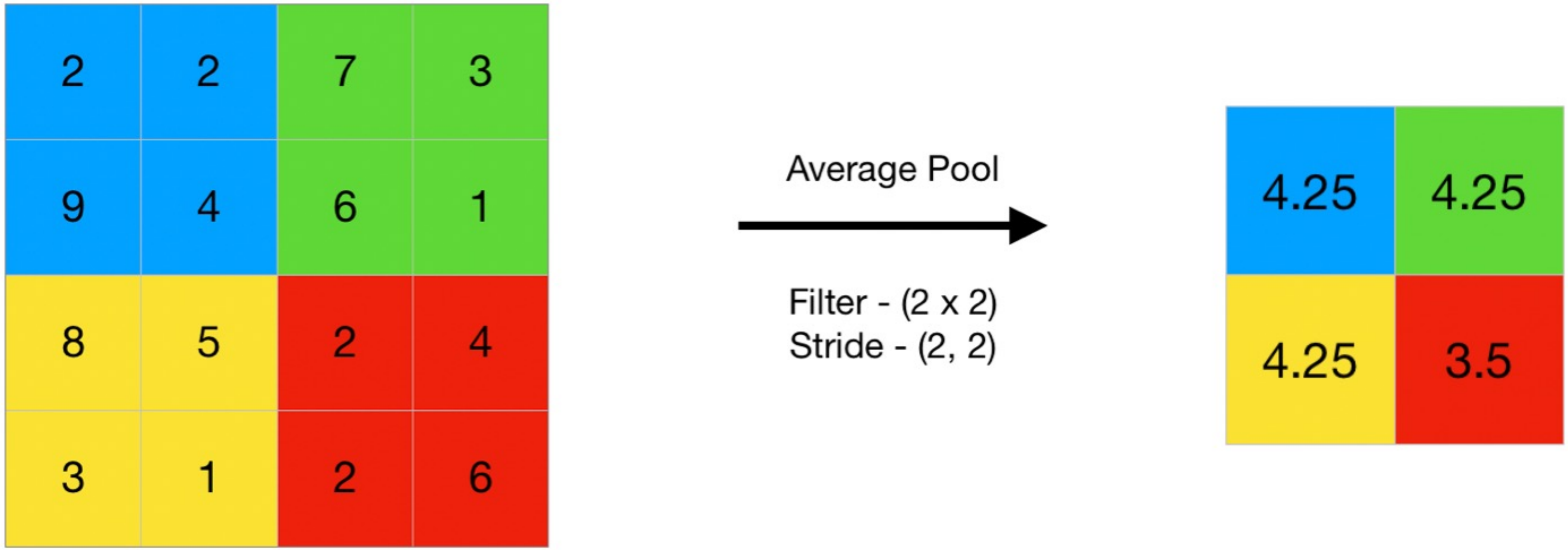
Max Pooling

Max pooling mainly helps in extracting sharp features, and reduce model variance and computation cost

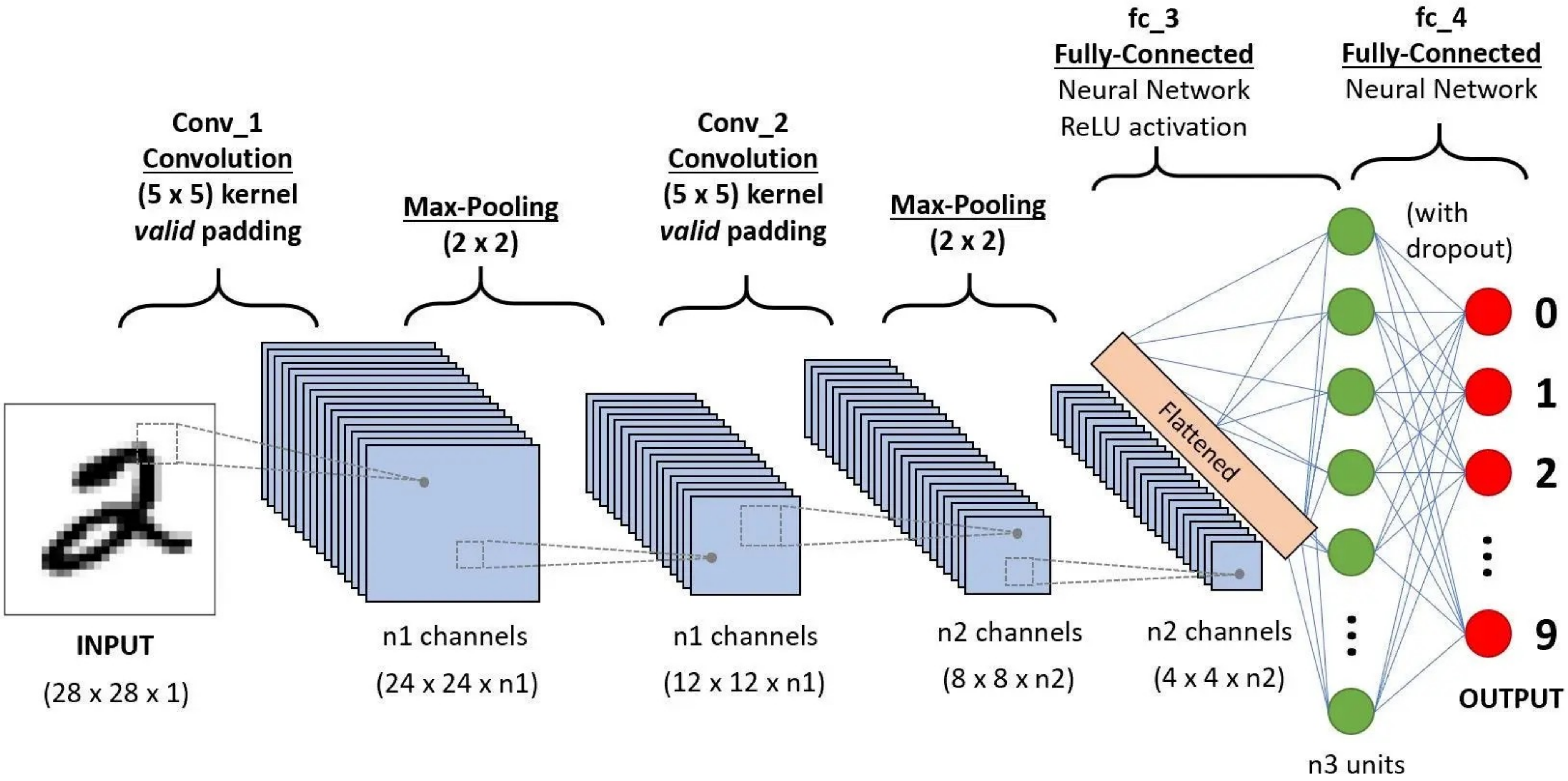


Avg Pooling

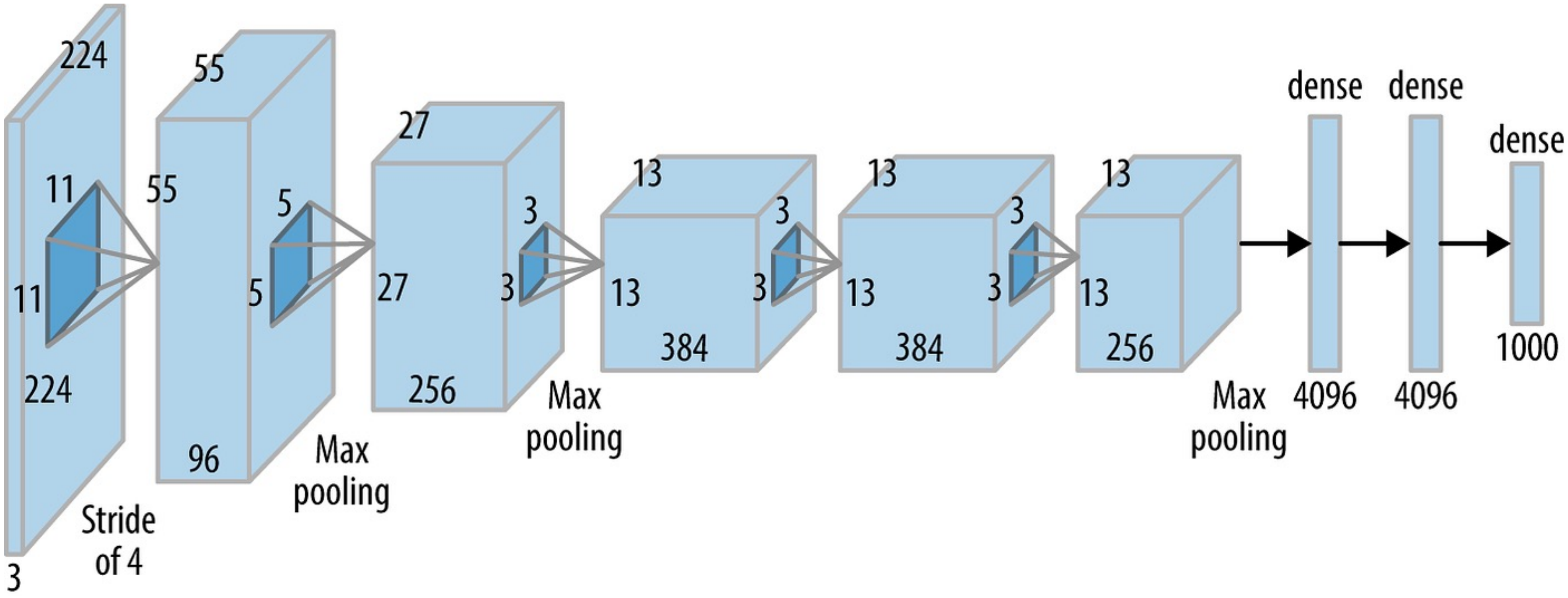
Avg pooling mainly helps in extracting smooth features, and reduce model variance and computation cost



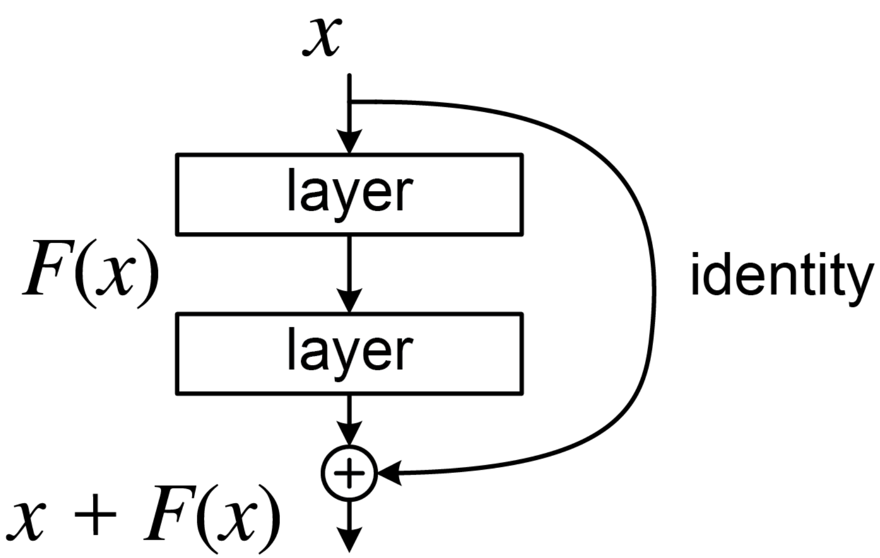
Convolutional Neural Network



Deep Convolutional Neural Network



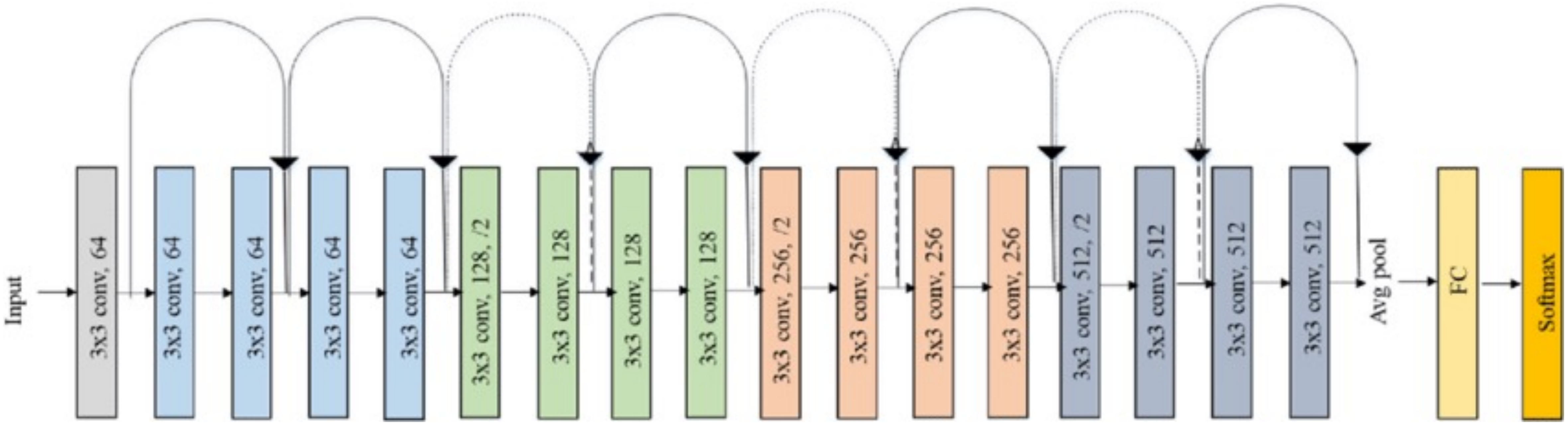
Residual Neural Networks (ResNet)



1. Address the issue of gradient vanishing

2. Easy to model identity function

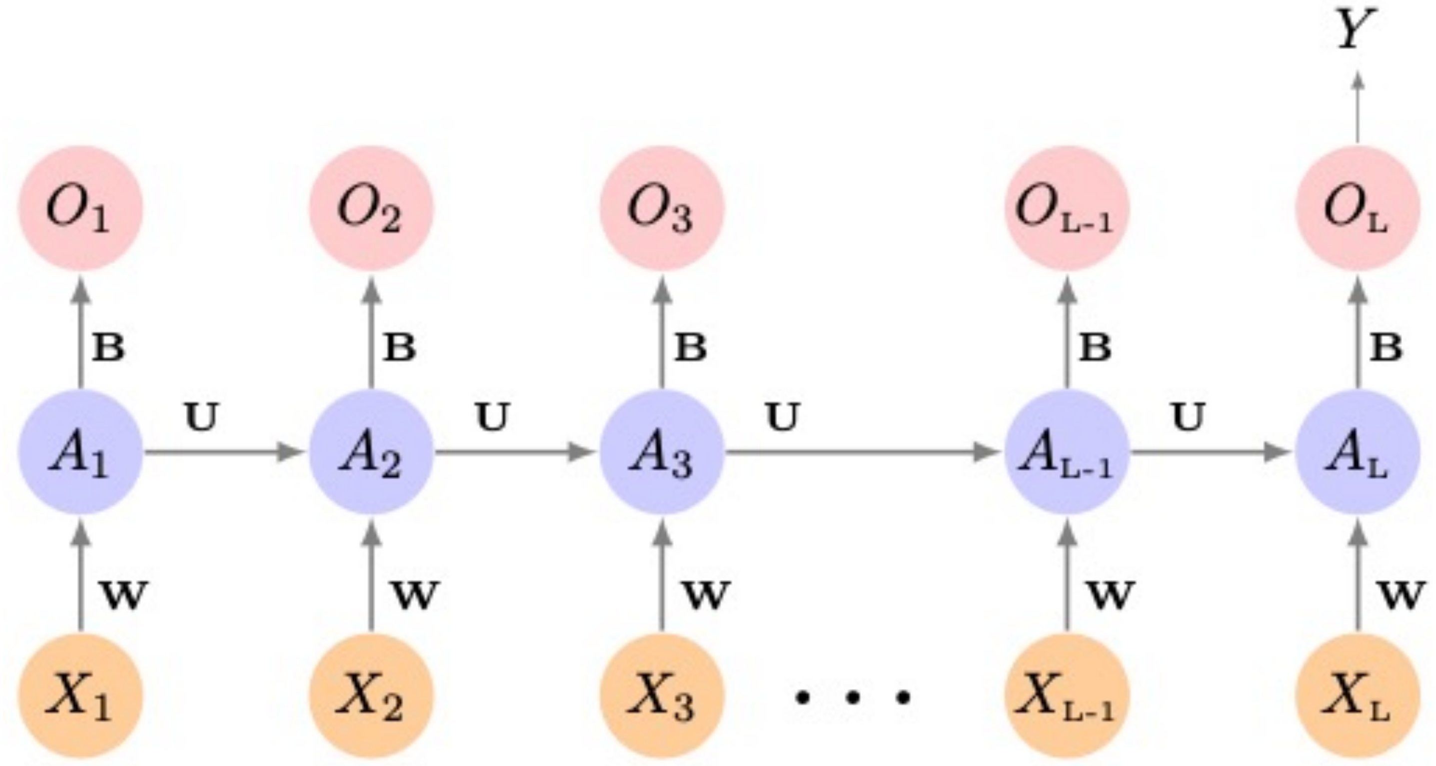
Residual Neural Networks (ResNet)



ResNet-18

Recurrent Neural Networks

The hidden layer is a sequence of vectors A_ℓ, receiving as input X_ℓ as well as $A_{\ell-1}$. A_ℓ produces an output O_ℓ.



Recurrent Neural Networks

Suppose $X_\ell = (X_{\ell 1}, X_{\ell 2}, \ldots, X_{\ell p})$ has p components, and $A_\ell = (A_{\ell 1}, A_{\ell 2}, \ldots, A_{\ell K})$ has K components. Then the computation at the kth components of hidden unit A_ℓ is

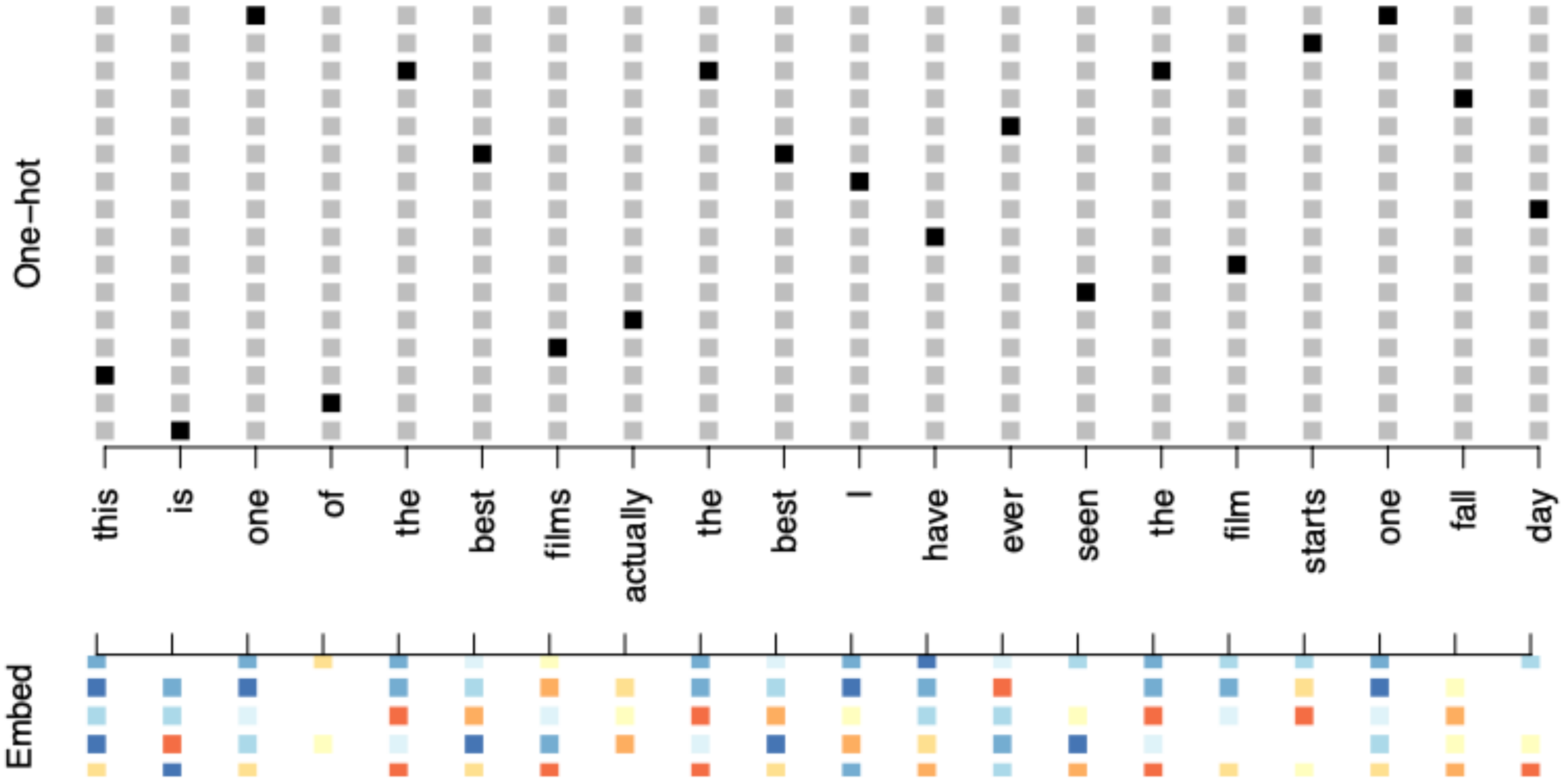
$$
\begin{aligned}
A_{\ell k} &= g\Big(w_{k0} + \sum_{j=1}^{p} w_{kj} X_{\ell j} + \sum_{s=1}^{K} u_{ks} A_{\ell-1,s}\Big) \\
O_\ell &= \beta_0 + \sum_{k=1}^{K} \beta_k A_{\ell k}
\end{aligned}
$$

Recurrent Neural Network Loss

Often we are concerned only with the prediction O_L at the last unit. For squared error loss, and n sequence/response pairs, we would minimize

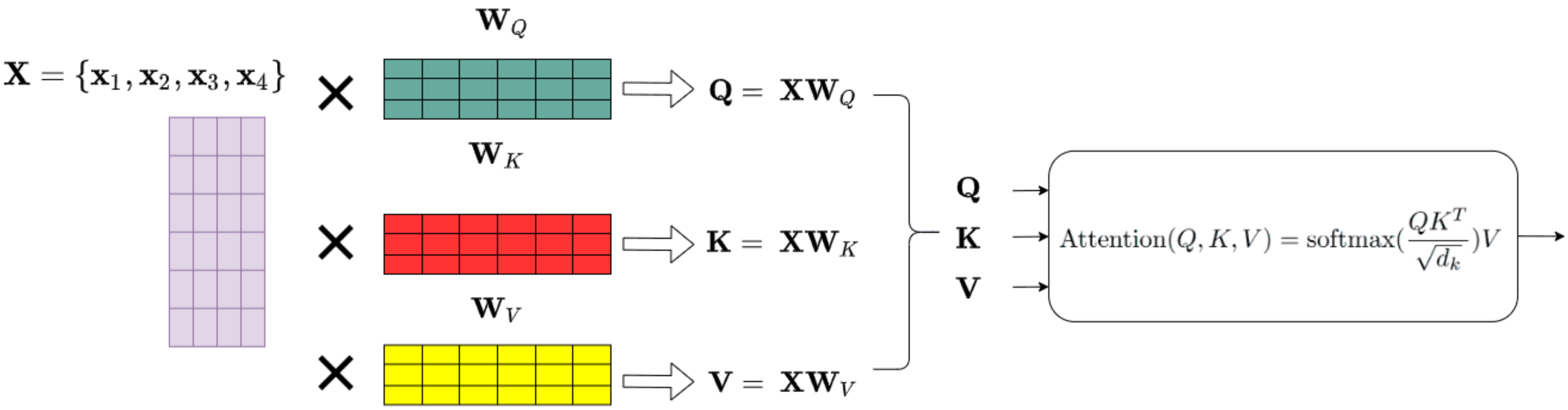
$$\sum_{i=1}^{n}(y_i - o_{iL})^2 = \sum_{i=1}^{n}\Big(y_i - \big(\beta_0 + \sum_{k=1}^{K}\beta_k g\big(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{iLj} + \sum_{s=1}^{K} u_{ks}a_{i,L-1,s}\big)\big)\Big)^2$$

Word Embedding



this is one of the best films actually the best I have ever seen the film starts one fall day · · · .

Transformers



Tricks of Deep Learning

Dropout: At each SGD update, randomly remove units with probability

Regularization: Minimize L2 norm of model parameters

Learning Scheduler: Decay or Periodic

Data Augmentation: Randomly crop the images