Convolution:

$S(t) = \int x(a)\,w(t-a)\,da$

(feature map) $= (x * w)(t)$

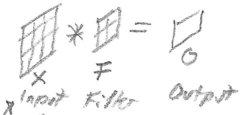
"Input" "kernel"

(Commutative Property)

$(A*B)(t) \equiv (B*A)(t)$

$$S(i,j) = (K*I)(i,j) = \sum_m \sum_n I(i+m, j+n)\,K(m,n)$$

CROSS-CORRELATION (for machine learning libraries)



(Convolution of $X * F$)

CHAIN RULE OF DIFFERENTIATION

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \times \frac{\partial q}{\partial x}$$

2D Discrete Convolution:

$$S(i,j) = (I*K)(i,j) = \sum_m \sum_n I(m,n)\,K(i-m, j-n)$$

$$\equiv (K*I)(i,j) = \sum_m \sum_n I(i-m, j-n)\,K(m,n)$$

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \mathrm{Conv}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

(Output)

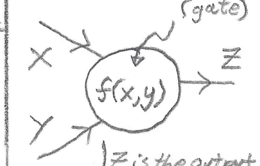
where $\begin{cases} X = \text{Input} \\ F = \text{Filter} \end{cases}$

& $O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$

CONVOLUTION FILTER UPDATE

$$F_{updated} = F - \alpha \frac{\partial L}{\partial F}$$

COMPUTATIONAL GRAPH



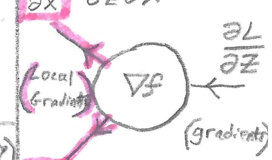
Z is the output

X, Y are inputs

FORWARD PASS

$$\boxed{\frac{\partial L}{\partial x}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

(L is the Loss fn)



$$\boxed{\frac{\partial L}{\partial y}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$$

BACKWARD PASS

$\frac{\partial L}{\partial z}$ = "Loss gradient from previous layer"

$\frac{\partial z}{\partial y}$ = "Local Gradients"

$\frac{\partial L}{\partial y}$ = "Gradient to update the filter with"

Full Convolution ← zero-padding

$$\frac{\partial L}{\partial F} = \mathrm{Convolution}\left(X, \frac{\partial L}{\partial O}\right)$$

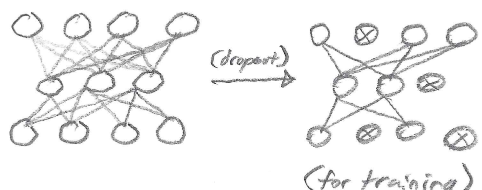
(Loss gradient)

$$\frac{\partial L}{\partial X} = \mathrm{FullConvolution}\left(\begin{matrix}F \text{ Rotated} \\ \text{by } 180°\end{matrix}, \frac{\partial L}{\partial O}\right)$$

(Input Image)

Typical Layout: Input → Convolution → ReLU → Pooling → FC

Dropout (with probability p that unit is present during training)



$p \sim \begin{cases} 0.5 \text{ for hidden layer} \\ 0.8 \text{ for input layer} \end{cases}$

⇒ PREVENTS OVERFITTING

* Bernoulli Distribution

$r_j^{l} \sim \mathrm{Bernoulli}(p) = \begin{cases} 1 \text{ for keep} \\ 0 \text{ for drop} \end{cases}$

$\tilde{y}^{(l)} = r^{(l)} * y^{(l)}$ (~ indicates dropout)

$z_i^{(l+1)} = w_i^{(l+1)}\tilde{y}^{(l)} + b_i^{(l+1)}$

* when testing, all units remain & weights ω are multiplied by prob. p

*Element-Wise Multiplication

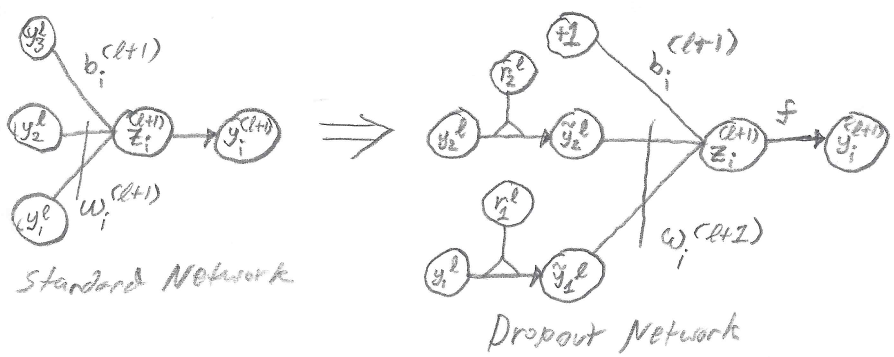
$A \odot B$ or $A \otimes B$

⇒ $p \rightarrow p\omega$

(Training) (Testing)

Dropout VS Ridge Regression (L2)

$x' = \frac{x}{|x|}$



Ridge Regression (L2 Regularization):

$$\text{minimize}\left\{ \left| y - X\omega \right|^2 + \lambda \left| \omega \right|^2 \right\}$$

(target)

* Smooths out behaviour of results

(forces ω norms to be small)

for dropout the result to minimize is

$\{|y - pX\omega|^2 + p(1-p)|\Gamma\omega|^2\}$

where $\Gamma = \mathrm{diag}(X^tX)$

⇒ SIMILAR RESULTS

Convolution Layer:

$m \times m \times r$

(size) (channel)

filter map: $n \times n \times q$ (kernel size) (filter count, representing different colors)

feature map: $m - n + 1$

⇒ Input convolved w/ filter results in a feature map

BACKPROPAGATION WITH CONVOLUTION LAYERS

$$\delta^{(l)} = W^{(l)t}\delta^{(l+1)} \cdot f'(z^{(l)})$$

(transpose)

(for dense connections)

(l = Convolutional layer)

Cost Function: $J(\omega, b, x, y)$ (training data)

Error Term: $\delta^{(l+1)}$ (of the $(l-1)$-th layer)

* If conv layer is sub-sampling layer too (i.e., mean or max pooling)

$$\delta_k^{(l)} = \mathrm{upsample}\left(W_k^{(l)t}\delta_k^{(l+1)}\right) \cdot f'(z_k^{(l)})$$

(where k is the filter index)

P = zero-padding

S = stride

OUTPUT SIZE = $(W - F + 2P)/S + 1$

maintain spatial size by $\begin{cases} P = \frac{1}{2}(F-1) \\ S = 1 \end{cases}$

Partial Derivatives

of a matrix O (output) wrt matrix F (filter)

(Previous Layer) (Local gradient)

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k}\frac{\partial O_k}{\partial F_i}$$

(via chain rule)

Bernoulli Random Variable (x)

$x = \begin{cases} 1: \text{Success} \\ 0: \text{Failure} \end{cases}$ & PMF follows:

$S_x = \{0,1\}$

$P_x(1) = p, \; P_x(0) = 1-p$

$\mu_x = E[X] = p, \; \sigma_x^2 = Var[X] = p(1-p)$

KL Divergence (Kullback-Leibler)

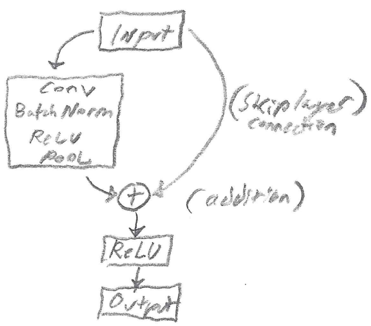
$\sum_{j=1}^{S_2} KL(\rho, \hat{\rho}_j)$ } Divergence btwn a Bernoulli RV with means ρ & $\hat{\rho}_j$

(S_2 = # of hidden nodes) (if similar, $\approx \emptyset$)

*Note that the "sparsity" of convolutional layers is due to the parameter sharing of weights

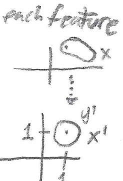
*Directed Acyclic Graph (DAG) Networks

*ResNet is a type of DAG with a residual connection (shortcut) that bypasses main network layers



*Good way to normalize data is to subtract the mean & divide by the standard deviation along each feature (higher accuracy)

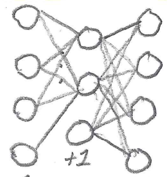
$$x' \rightarrow \frac{x - \mu_x}{\sigma_x} \Rightarrow$$



*DONE PER FEATURE (BatchNorm)

Autoencoders

(for data compression, denoising, feature maps, classification, sparse representation)



*Aim is for output to equal input

*Consider hidden node: a_j^2 (activation of hidden unit j)

*Given the input X: $a_j^2(x^{(i)})$ is the current activation of the hidden unit j after receiving input X (corresponding to example) in the training set

FEATURE EXTRACTION

Batch Normalization

(done in minibatch of # of vectors)

$$y = \left(\frac{X - E[X]}{\sqrt{Var[X] + \text{Constant}}} \right) \gamma + \beta$$

(mean) (vectors)

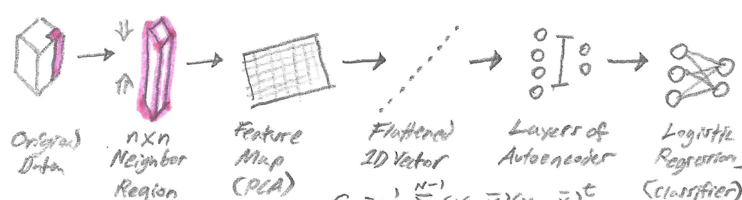
(to avoid numerical instability)

Faster Convergence
Regularization
Decreased Importance of weight initialization

*Standard Normalization (most common): γ is unity & $\beta = \emptyset$

Principal Component Analysis (PCA):

"Converting input vector X into feature vector Y"



(Keep k components)

} Spectral–Spatial Representation Extraction (compression along spectral dimension)

$$C_x = \frac{1}{N-1} \sum^{N-1} (x_i - \bar{x})(x_i - \bar{x})^t$$

DKLT Theory: Transformation T to diagonalize the covariance matrix of a random signal in the discrete time domain for signal compression

$\Rightarrow$ PCA Approximation through uncorrelated elements of Y (keeping M components)

SVM – Kernel Functions (e.g., RBFNs)

Kernel function (usually non linear) } $f(|x - \omega|^2)$ (distance^2) (data) (params)

Example: Gaussian Kernel Function:

$$f \sim e^{-|x-\omega|^2 / 2\sigma^2}$$

$\phi \leftarrow f$ (for kernel functions)

$\Rightarrow \phi$ returns inner product of two points in a feature space

Polynomial Kernel: (Image Processing)

$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$

Gaussian Kernel: (NO PRIOR KNOWLEDGE)

$K(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$

Gaussian Radial Basis Function: (NO PRIOR KNOWLEDGE)

$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$

Kernel Functions (SVMs) take input vectors in original space & returns their dot product in the feature space

$K(x, x') = \langle \phi(x), \phi(x') \rangle$

*Mapping from input space to a dual space ("Gram Matrix")

Sparsity Parameter ρ (constraint)

(typically $\rho \approx 0.05$)

*ENFORCE APPROXIMATION CONSTRAINT

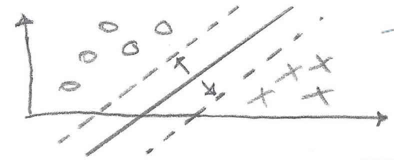
$\hat{\rho}_j = \rho$ (requires an extra penalty term in optimization)

where $\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} a_j^2(x^{(i)})$

Average activation for hidden unit j over the entire training data set (containing m examples)

*Support Vector Machines (SVMs):

For linearly seperable decision boundaries defined as a border with as much space between vector classes as possible



(if k = N → NO LOSS OF INFORMATION)

Eigenvalues with large variance are the most important components to keep (i.e., keep λ if large)

*for real signals: $\rho \approx 0.9$ (speech modeling)

$$\lambda_k = \frac{1 - \rho^2}{1 - 2\rho\cos(w_k) + \rho^2}$$

Discrete Cosine Transform (DCT) is a good approximation of KLT

Toeplitz Matrix ($0 < \rho < 1$)

Foregets anything except 1 element

$$C_x = \begin{bmatrix} 1 & \rho & \rho^2 \\ \rho & 1 & \rho \\ \rho^2 & \rho & 1 \end{bmatrix}$$

= { Covariance Matrix for a zero-mean, first-order Markov Sequence (of N elements)

$X_n \leftarrow \rho X_{n-1} + \epsilon$