Linear regression

Cost function, Gradient descent

The objective of linear regression is to minimize the cost function

$$J = \frac{1}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)^2$$

$$\hat{y} = b + Wx \;=\; W_0 + W_1 x$$

- *J : cost function*
- $\hat{y}$: *predicted target value*
- y: *actual target value*
- x: *input value*
- *b: bias*
- *W: weight*

Linear regression

(Batch) Gradient Descent

$$W_j := W_j - \alpha \frac{\partial J}{\partial W}$$

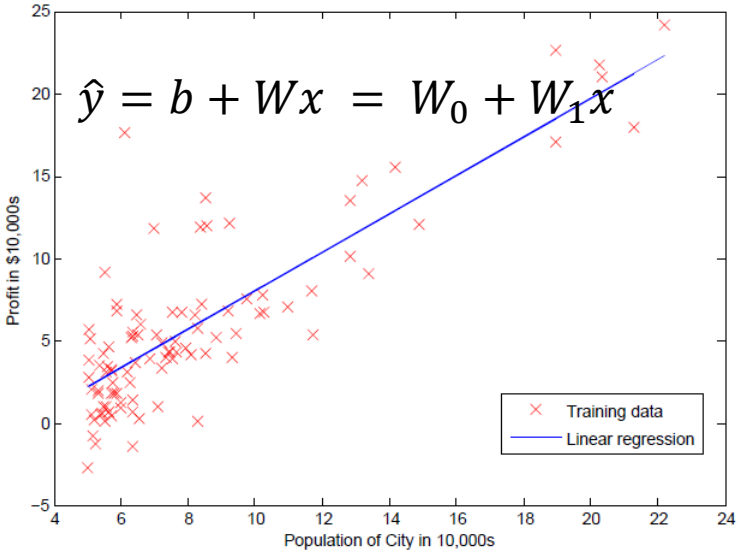
- α: $learning\ rate$
- $\frac{\partial J}{\partial W}$: gradient

$$\frac{\partial J}{\partial W_0} = \frac{2}{m} \sum_{i=1}^{m} (\widehat{y^i} - y^i)$$

$$\frac{\partial J}{\partial W_1} = \frac{2}{m} \sum_{i=1}^{m} (\widehat{y^i} - y^i) x$$

Linear regression

Linear regression fit



<Machine Learning, Andrew Ng>

- Weights
  $W_0, W_1, \ldots$
- Performance measure
  - Cost

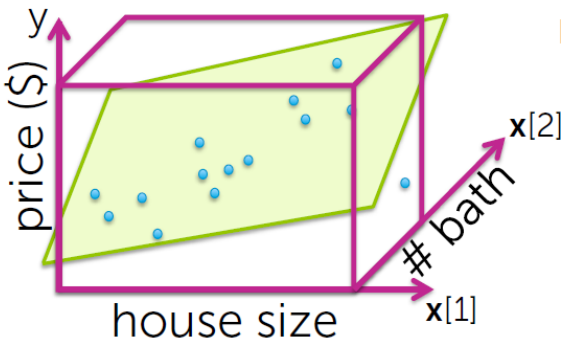
$$J = \frac{1}{m}\sum_{i=1}^{m}(\hat{y^i} - y^i)^2$$

  - Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y^i} - y^i)^2}$$

Linear regression

With multiple variables



<Machine Learning, Emily Fox & Carlos Guestrin>

$$\hat{y} = W_0 + W_1 x_1 + W_2 x_2 + \ldots$$

$$\frac{\partial J}{\partial W_0} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)$$

$$\frac{\partial J}{\partial W_1} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_1$$

$$\frac{\partial J}{\partial W_2} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_2$$

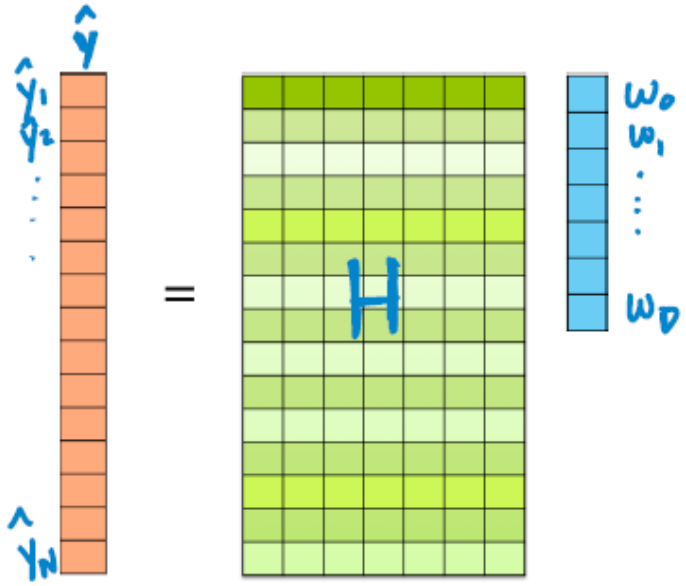
Linear regression

Matrix calculation

$$J = \frac{1}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)^2$$

$$J = \frac{1}{m}(\hat{Y} - Y)^T(\hat{Y} - Y)$$

$$J = \frac{1}{m}(W \cdot X - Y)^T(W \cdot X - Y)$$



<Machine Learning, Emily Fox & Carlos Guestrin>

Linear regression

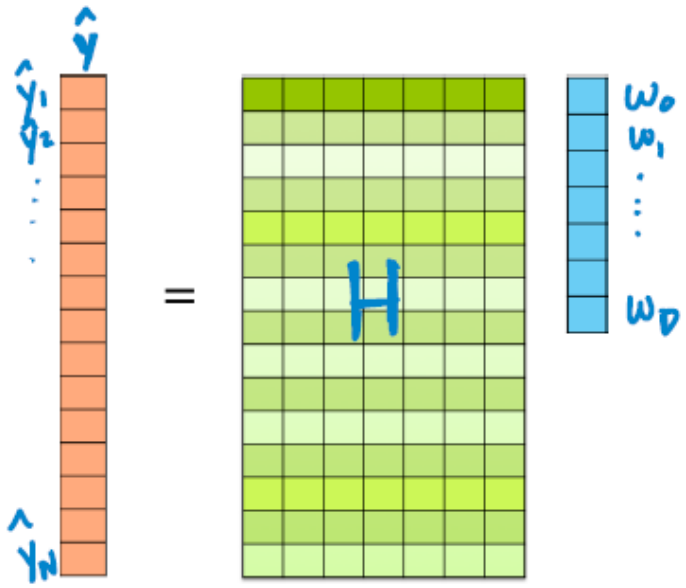
Matrix calculation for gradient

$$\frac{\partial J}{\partial W_j} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_j$$

$$\frac{\partial J}{\partial W} = \frac{2}{m}(\hat{Y} - Y)^T \cdot X$$

$$\frac{\partial J}{\partial W} = \frac{2}{m}(X \cdot W - Y)^T \cdot X$$

$$\frac{\partial J}{\partial W} = \frac{2}{m}(X^T \cdot X \cdot W - X^T \cdot Y)$$



<Machine Learning, Emily Fox & Carlos Guestrin>

Linear regression

Normal equation

To find the value of θ that minimizes the cost function, there is a *closed-form solution*

—in other words, a mathematical equation that gives the result directly. This is called the *Normal Equation.*

$$W = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$$

Normal equation

$$W = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$$

Gradient descent

- Need to choose learning rate α
- Need to feature scaling
- Need to many iterations
- Works well even large number of features

Normal equation

- No Need to choose α
- No Need to feature scaling
- No need to iterations
- Needs to compute inverse matrix

$(X^T \cdot X)^{-1}$

Feature Scaling

Feature normalization

- Min-Max scaling: (X – min.value) / (max.value – min.value)
  - shifted and rescaled. 0~1

- Standardization: (X – mean.value) / (std.value)
  - zero mean and unit standard deviation
  - does not bound a specific range.
  - less affected by outliers

Feature normalization for test data

(X – mean.value) / (std.value)

In order to normalize validation / test data set, you need to use mean value and standard deviation value for training data set.

Avoiding Overfitting through Regularization

- **Sum of squares**

Ridge : L2 norm

$W_0^2 + W_1^2 + W_2^2 + \cdots = \sum_{j=1}^{n} W_j^2 = \|W\|_2^2$

Cost function $+\lambda\|W\|_2^2$

- **Sum of absolute value**

Lasso: L1 norm

$|w_0| + |w_1| + |w_2| + \cdots = \sum_{j=1}^{n} |w_j| = \|W\|_1$

Cost function $+\lambda\|W\|_1$

Regularized linear regression

Ridge regression (L2 penalty)

Cost function

$$J = \frac{1}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)^2 + \frac{\lambda}{m}\sum_{j=1}^{n} W_j^2$$

Regularized linear regression

Ridge regression (L2 penalty)

Gradient

$$\frac{\partial J}{\partial W_0} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_0^i$$ when j=0

$$\frac{\partial J}{\partial W_j} = \frac{2}{m}\sum_{i=1}^{m}(\widehat{y^i} - y^i)x_j^i + \frac{2\lambda}{m}W_j$$ when j>=1

Regularized linear regression

Ridge regression (L2 penalty)

Gradient descant

Repeat {

$$W_0 = W_0 - \alpha \frac{2}{m} \sum_{i=1}^{m} (\widehat{y^i} - y^i) x_0^i \qquad \text{when j=0}$$

$$W_j = W_j - \alpha \left[\frac{2}{m} \sum_{i=1}^{m} (\widehat{y^i} - y^i) x_j^i + \frac{2\lambda}{m} W_j \right] \qquad \text{when j>=1}$$

}

Regularized linear regression

Ridge regression (L2 penalty)

Matrix form

Cost function

$$J = \frac{1}{m}[(W \cdot X - Y)^T (W \cdot X - Y) + \lambda W^T \cdot W]$$

Gradient Descent

$$\frac{\partial J}{\partial W} = \frac{2}{m}[(X \cdot W - Y)^T \cdot X + \lambda W]$$

$$\frac{\partial J}{\partial W} = \frac{2}{m}[(X \cdot W - Y)^T \cdot X + \lambda I \cdot W]$$

I: identity matrix

Regularized linear regression

Ridge regression (L2 penalty)

Normalization equation for ridge regression

$$W = (X^T \cdot X + \lambda I)^{-1} \cdot X^T \cdot Y$$

when λ=0, $W = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$

When λ= infinite, W=0

Regularized linear regression

Lasso regression (L1 penalty)

Matrix form

Cost function

$$J = \frac{1}{m}[(W \cdot X - Y)^T (W \cdot X - Y) + \lambda |W|]$$

Gradient Descent

Coordinate descent method

<Machine Learning, Emily Fox & Carlos Guestrin>

$$A = \pi r^2$$

Bias / Variance Trade off

Large λ: high bias, low variance
Small λ: low bias, high variance

Bias

This part of the generalization error is due to wrong assumptions, such as assuming that the data is linear when it is actually quadratic. A high-bias model is most likely to underfit the training data.

Variance

This part is due to the model's excessive sensitivity to small variations in the training data. A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance, and thus to overfit the training data.

<Hands-On ML, Aurelien Geron>

Logistic Regression

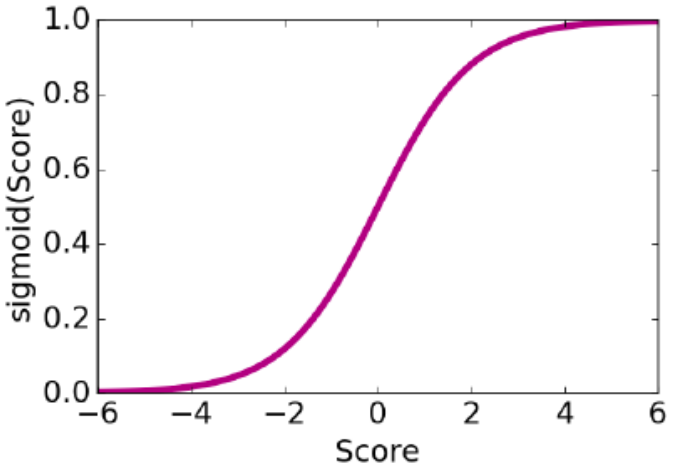
Given x, the probability if y = 1

$$\hat{y} = P(y = 1|x)$$

$$P = \sigma(W^T x + b)$$

Logistic function (sigmoid)

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



<Machine Learning, Emily Fox & Carlos Guestrin>

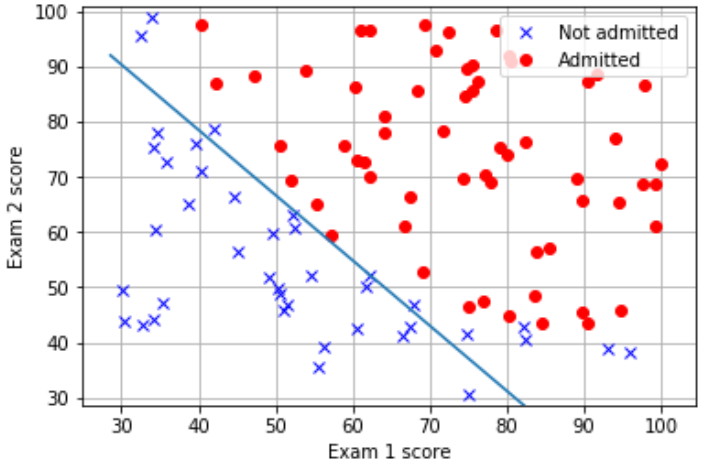
Logistic regression

Model Prediction

$$P(y = 1|x)$$

$$= \frac{1}{1+\exp(-W^T x - b)}$$

$$\hat{y} = \begin{cases} 1 \; if \; P \geq 0.5 \\ 0 \; if \; P < 0.5 \end{cases}$$



Logistic Regression

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})]$$

If y = 1 $$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i})]$$

If y = 0 $$J = -\frac{1}{m}\sum_{i=1}^{m}[\log(1 - \widehat{y^i})]$$

<Machine Learning, Emily Fox & Carlos Guestrin>

Logistic Regression

Regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1-y^i)\log(1-\widehat{y^i})] + \frac{\lambda}{m}\sum_{j=1}^{n} W_j^2$$

gradient

$$\frac{\partial J}{\partial W_0} = -\frac{1}{m}\Sigma_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] \qquad \text{for j=0}$$

$$\frac{\partial J}{\partial W_j} = -\frac{1}{m}\Sigma_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] + \frac{2\lambda}{m}W_j \qquad \text{for j} \geq 1$$

Logistic Regression

Gradient (or derivative of cost funciton)

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\left(\widehat{y^i} - y^i\right)x_j^i]$$

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\left(P - y^i\right)x_j^i]$$

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\left(\sigma(W^T x + b) - y^i\right)x_j^i]$$

Logistic Regression

Regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1-y^i)\log(1-\widehat{y^i})] + \frac{2\lambda}{m}\sum_{j=1}^{n} W_j^2$$

gradient

$$\frac{\partial J}{\partial W_0} = -\frac{1}{m}\sum_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] \qquad \text{for j=0}$$

$$\frac{\partial J}{\partial W_j} = -\frac{1}{m}\sum_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] + \frac{\lambda}{m}W_j \qquad \text{for j} \geq 1$$

Logistic Regression

$$\hat{y} = \sigma(W^T x + b)$$

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})]$$

<Machine Learning, Emily Fox & Carlos Guestrin>

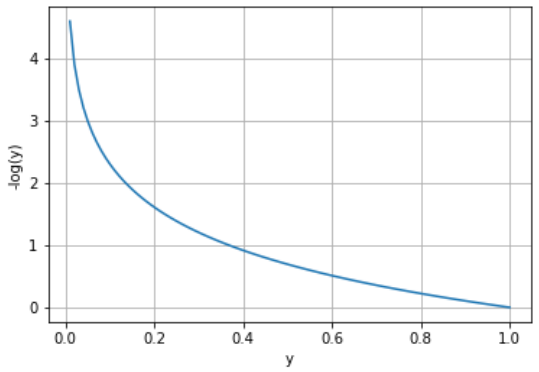
Logistic Regression

Loss function

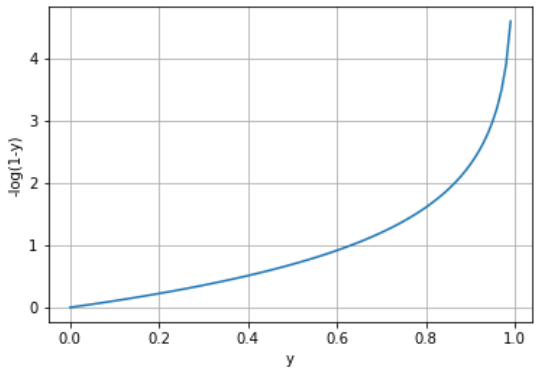
Given x, want $\widehat{y^i} \approx y^i$

$$L = -[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})]$$

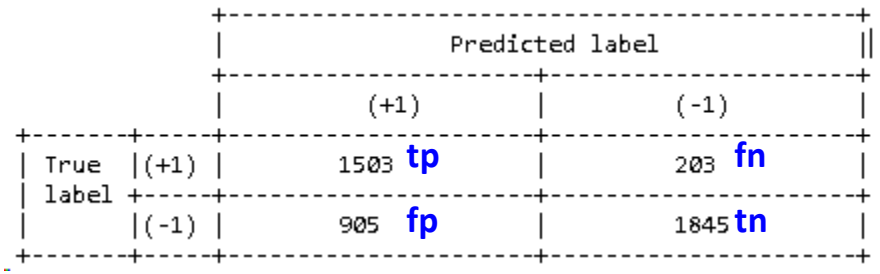
If y = 1, $L = -\log(\widehat{y^i})$



If y = 0, $L = -\log(1 - \widehat{y^i})]$



Confusion matrix



Accuracy : (tp + tn) / (tp + fp + fn + tn) * 100

Precision : tp / predict positive = tp / (tp + fp) * 100

Recall (true positive rate) : tp / actual positive = tp / (tp + fn) * 100

True negative rate : tn / actual negative = tn / (fp + tn) * 100

Precision

Fraction of positive predictions that are correct.

$$\text{precision} = \frac{\text{\# true positives}}{\text{\# true positives + \# false positives}}$$

<Machine Learning, Emily Fox & Carlos Guestrin>

Recall

Fraction of positive data points correctly classified

$$\text{Recall} = \frac{\text{\# true positives}}{\text{\# true positives} + \text{\# false negatives}}$$

<Machine Learning, Emily Fox & Carlos Guestrin>

Mini-batch Gradient Descent

- Batch Gradient Descent

Computes the gradients based on full training set

Ex) Offline learning

- Stochastic Gradient Descent

Computes just one instance

Ex) Online learning

- Mini-batch Gradient Descent

Computes the gradients on small random sets of instances

<Hands-On ML, Aurelien Geron>

Stochastic gradient descent

Batch Gradient descent

$$\frac{\partial J}{\partial W} = -\frac{1}{m}\sum_{i=1}^{m}[\,(\widehat{y^i} - y^i)x_j^i]$$

(Sum over data points)

Stochastic Gradient descent

$$\frac{\partial J}{\partial W} = -(\widehat{y^i} - y^i)x_j^i$$

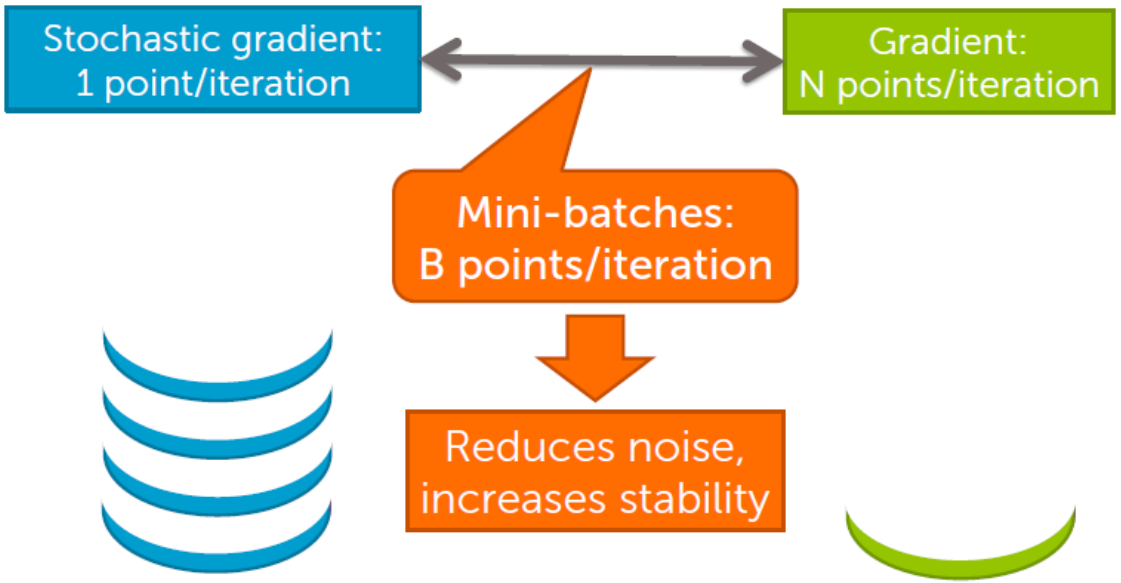
(Each time, pick different data point)

Batch vs. Stochastic gradient

Algorithm	Time per iteration	Total time to convergence for large data: In theory	Total time to convergence for large data: In practice	Sensitivity to parameters
Batch Gradient	Slow for large data	Slower	Often slower	Moderate
Stochastic gradient	Always fast	Faster	Often faster	Very high

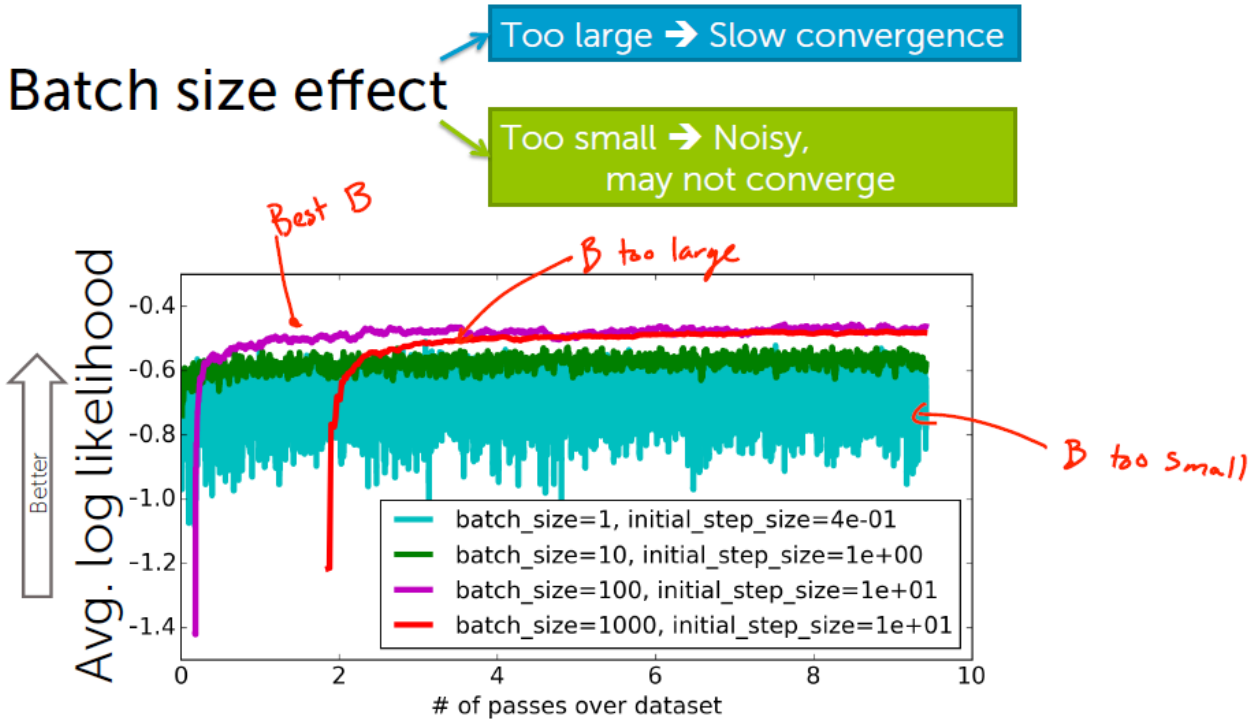
<Machine Learning, Emily Fox & Carlos Guestrin>

Batch / stochastic: two extremes



<Machine Learning, Emily Fox & Carlos Guestrin>

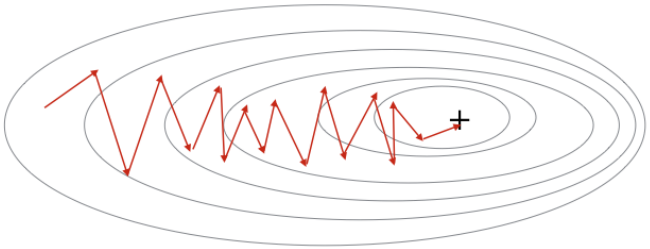
Batch size effect in mini-batch



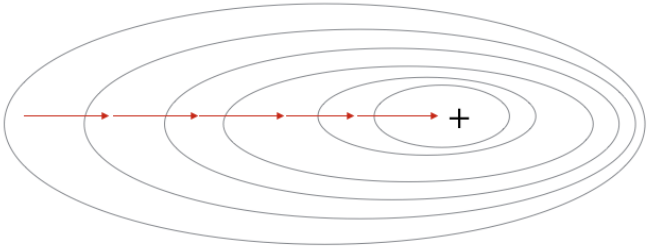
<Machine Learning, Emily Fox & Carlos Guestrin>

Converge

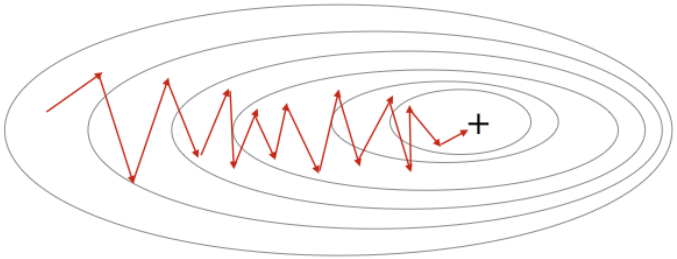
Stochastic Gradient Descent



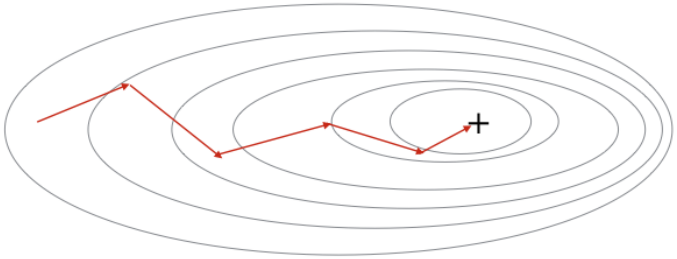
Gradient Descent



Stochastic Gradient Descent



Mini-Batch Gradient Descent



Softmax function

Softmax score for class k

$$s_k(x) = W^k \cdot x$$

Softmax function

$$p_k = \frac{\exp(s_k(x))}{\sum_{j=1}^{K} \exp(s_j(x))}$$

- K is the number of classes.
- **s**(**x**) is a vector containing the scores of each class for the instance **x**.
- p_k is the estimated probability that the instance **x** belongs to class k given the scores of each class for that instance.

Softmax classifier

- Multiclass classifier
- Provides normalized class probabilities

$$p_k = \frac{\exp(s_k(x))}{\sum_{j=1}^{K} \exp(s_j(x))}$$

$$\hat{y} = \underset{k}{\operatorname{argmax}}\, p_k$$

It predicts the class with the highest estimated probability.

Softmax in matrix

- for $x \in \mathbb{R}^{1\times n}$, $softmax(x) = softmax(\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}) = \begin{bmatrix} \frac{e^{x_1}}{\sum_j e^{x_j}} & \frac{e^{x_2}}{\sum_j e^{x_j}} & \cdots & \frac{e^{x_n}}{\sum_j e^{x_j}} \end{bmatrix}$
- for a matrix $x \in \mathbb{R}^{m\times n}$, x_{ij} maps to the element in the i^{th} row and j^{th} column of x, thus we have:

$$softmax(x) = softmax\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix} = \begin{bmatrix} \frac{e^{x_{11}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{12}}}{\sum_j e^{x_{1j}}} & \frac{e^{x_{13}}}{\sum_j e^{x_{1j}}} & \cdots & \frac{e^{x_{1n}}}{\sum_j e^{x_{1j}}} \\ \frac{e^{x_{21}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{22}}}{\sum_j e^{x_{2j}}} & \frac{e^{x_{23}}}{\sum_j e^{x_{2j}}} & \cdots & \frac{e^{x_{2n}}}{\sum_j e^{x_{2j}}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{e^{x_{m1}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m2}}}{\sum_j e^{x_{mj}}} & \frac{e^{x_{m3}}}{\sum_j e^{x_{mj}}} & \cdots & \frac{e^{x_{mn}}}{\sum_j e^{x_{mj}}} \end{bmatrix} = \begin{pmatrix} softmax(\text{first row of x}) \\ softmax(\text{second row of x}) \\ \cdots \\ softmax(\text{last row of x}) \end{pmatrix}$$

<Deep Learning, Andrew Ng>

Backpropagation

Forward pass

$f(x) = 5x$

$x(u) = 1 - u$

$u(t) = t^2$

$f(t) = 5(1 - t^2) = 5 - 5t^2$

$\frac{df}{dt} = -10t$

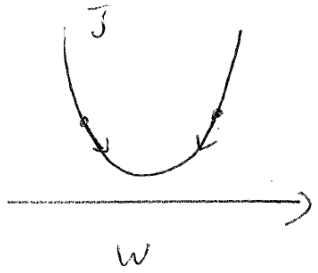
Backward pass

$\frac{df}{dt} = \frac{df}{dx} \frac{dx}{du} \frac{du}{dt}$

$\frac{df}{dt} = (5)(-1)(2t)$

$= -10t$

Linear regression & chain rule



$$W := W - \alpha \frac{\partial J}{\partial W}$$

$$J = \frac{1}{m} \sum (\hat{y} - y)^2$$

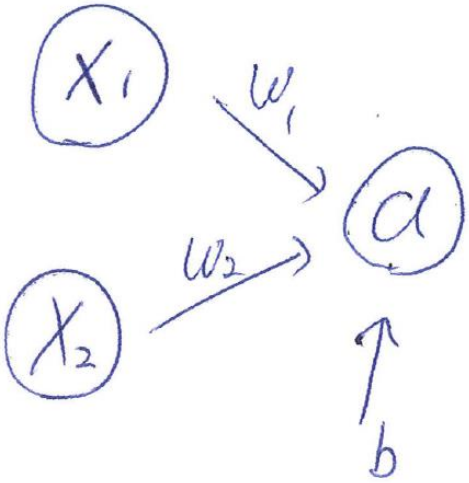
$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W}$$

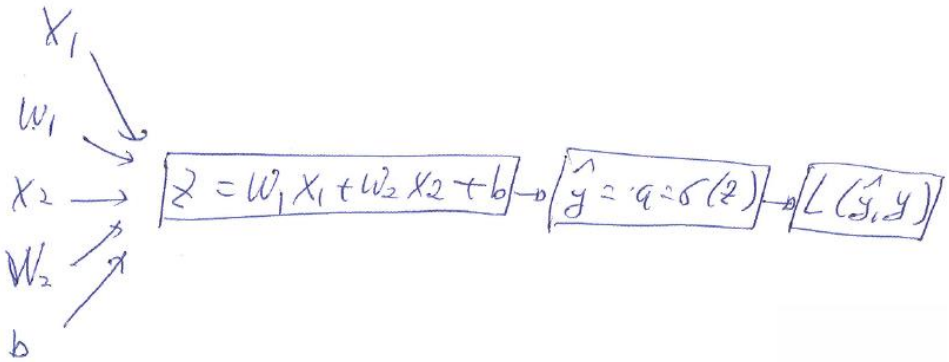
$$\frac{\partial J}{\partial \hat{y}} = \frac{2}{m} \sum (\hat{y} - y)$$

$$\frac{\partial \hat{y}}{\partial W} = x \quad \text{since } \hat{y} = W \cdot x$$

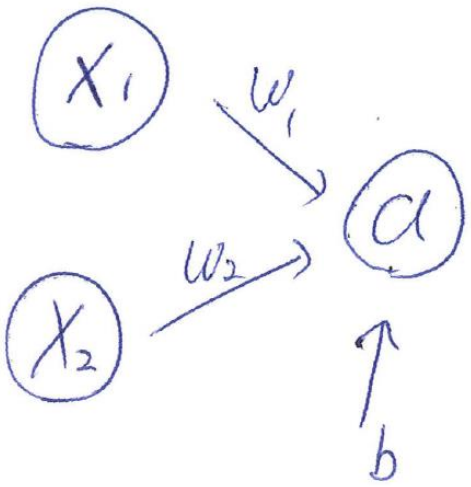
$$\therefore \frac{\partial J}{\partial W} = \frac{2}{m} \sum (\hat{y} - y) \cdot x$$

Logistic regression





Logistic regression



$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z) = \frac{1}{1+\exp(-z)}$$

$$L(\hat{y}, y) = -[y \log(a) + (1-y) \log(1-a)]$$

Two neurons (Logistic regression)

$$a^{[1]} = \sigma(w \cdot a^{[0]} + b)$$

$$L = (a^{[1]} - y)^2$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial b}$$

$$z^{[1]} = w \cdot a^{[0]} + b$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b}$$

Derivative of da/dz

$$a = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial a}{\partial z} = \frac{d}{dz}\left[(1+e^{-z})^{-1}\right]$$

$$= e^{-z}(1+e^{-z})^{-2}$$

$$= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}}$$

$$= \frac{1}{1+e^{-z}} \cdot \left[\frac{1+e^{-z}-1}{1+e^{-z}}\right]$$

$$= \frac{1}{1+e^{-z}} \cdot \left[1 - \frac{1}{1+e^{-z}}\right]$$

$$= a\,[1-a]$$

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z) = \frac{1}{1+\exp(-z)}$$

$$L(\hat{y}, y) = -[y \log(a) + (1-y)\log(1-a)]$$

Derivative of dL/dz

$$\frac{\partial L}{\partial a} = "da" = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$"da" = \frac{\partial L(a,y)}{\partial a}$$

$$\frac{\partial L}{\partial z} = "dz" = \frac{\partial L(a,y)}{\partial z}$$

$$dz = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}$$

$$= \left[-\frac{y}{a} + \frac{1-y}{1-a}\right][a(1-a)]$$

$$= a - y$$

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z) = \frac{1}{1+\exp(-z)}$$

$$L(\hat{y}, y) = -[y\log(a) + (1-y)\log(1-a)]$$

Derivative of dL/dW

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z) = \frac{1}{1+\exp(-z)}$$

$$L(\hat{y}, y) = -[y \log(a) + (1-y) \log(1-a)]$$

$$\frac{\partial L}{\partial w_1} = "dw_1"$$

$$= \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w_1}$$

$$= dz \cdot \frac{\partial z}{\partial w_1}$$

$$= (a-y) \cdot x_1$$

$$dw_2 = \frac{\partial L}{\partial w_2}$$

$$= (a-y) \cdot x_2$$

$$db = \frac{\partial L}{\partial b}$$

$$= (a-y) \cdot 1$$

$$= a - y$$

Gradient Descent on m observations

$$J(w,b) = \frac{1}{m} \sum L(a^{(i)}, y)$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} L(a^{(i)}, y)}_{dw_1^{(i)}}$$

Gradient Descent on m observations

Loop to calculate derivatives

For $i = 1$ to m:

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log (1-a^{(i)})]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 \mathrel{+}= dz^{(i)} \cdot x_1^{(i)}$$

$$dw_2 \mathrel{+}= dz^{(i)} \cdot x_2^{(i)}$$

$$db \mathrel{+}= dz^{(i)}$$

After the loop

$$J = J/m$$

$$dw_1 = dw_1/m$$

$$dw_2 = dw_2/m$$

$$db = db/m$$

Gradient Descent

After the loop

$J = J/m$

$dw_1 = dw_1/m$

$dw_2 = dw_2/m$

$db = db/m$

For iteration in range (1000):

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

Gradient Descent using matrix form

For iter in range(1000):

$Z = np.dot(X, W) + b$: (m, 1)

$A = \sigma(Z)$: (m, 1)

$dZ = A - Y$: (m, 1)

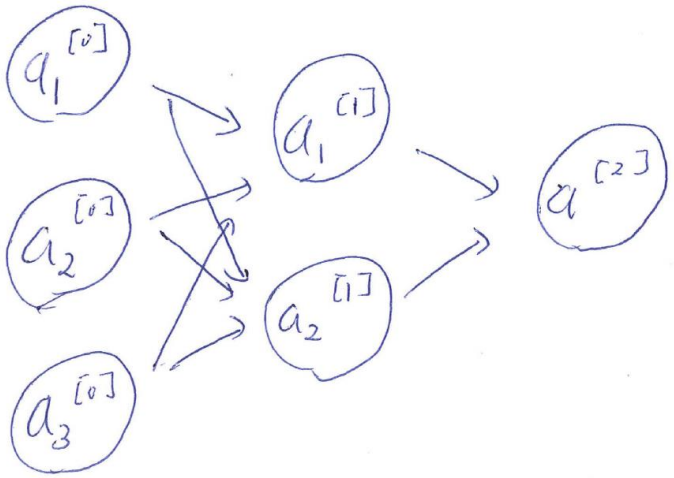
$dW = \frac{1}{m} np.dot(X.T, dZ)$: (n, 1)

$db = \frac{1}{m} np.sum(dZ)$

$W := W - \alpha dW$

$b := b - \alpha db$

Neural Network: 1 hidden layer



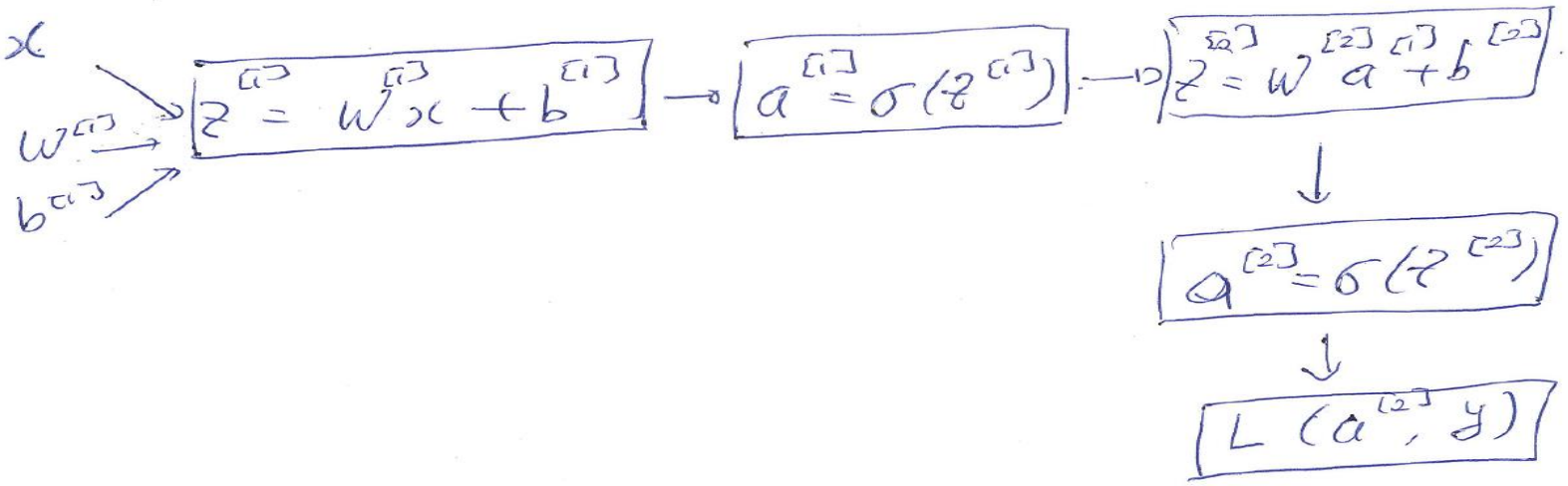
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$

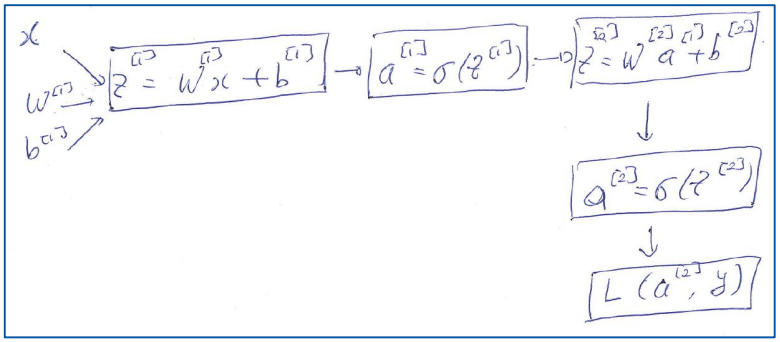
$$= \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$

$\hat{y} = a^{[2]}$

Neural Network: 1 hidden layer



Derivative of $dL/dW^{[2]}$



$$L(\hat{y}, y) = -[y \log(a^{[2]}) + (1-y)\log(1-a^{[2]})]$$

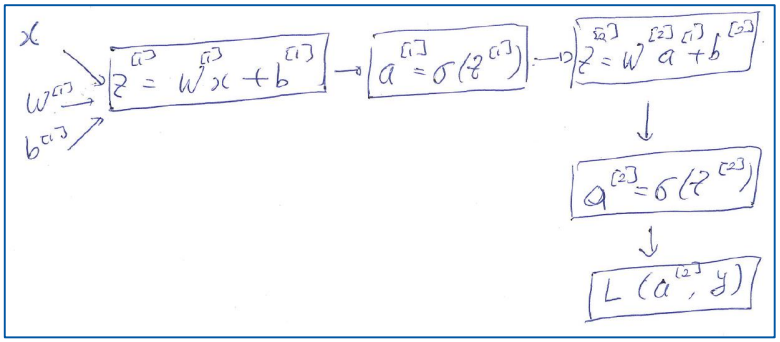
$$da^{[2]} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$$

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = (a^{[2]} - y)\, a^{[1]} = dz^{[2]} \cdot a^{[1]}$$

$$db^{[2]} = dz^{[2]} = a^{[2]} - y$$

Derivative of $dL/dZ^{[1]}$



$$da^{[1]} = \frac{\partial L}{\partial a^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}}$$

$$= dz^{[2]} \cdot w^{[2]}$$

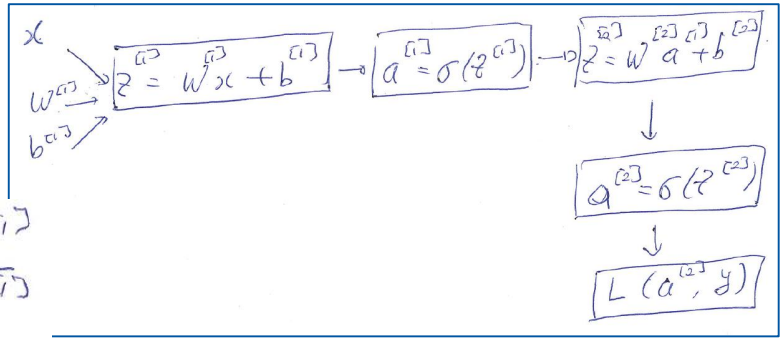
$$= (a^{[2]} - y) \cdot w^{[2]}$$

$$dz^{[1]} = \frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

$$= da^{[1]} \cdot a^{[1]}(1 - a^{[1]})$$

$$= (a^{[2]} - y) \cdot w^{[2]} \cdot a^{[1]} \cdot (1 - a^{[1]})$$

Derivative of dL/dW[1]



$$dW^{[1]} = \frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

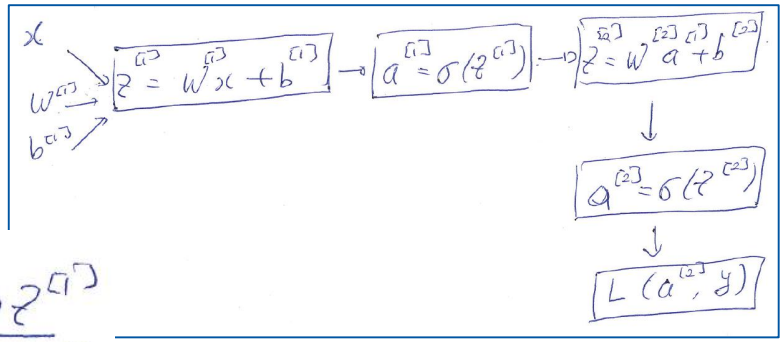
$$= dz^{[1]} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

$$= (a^{[2]} - y) \cdot W^{[2]} \cdot a^{[1]} \cdot (1 - a^{[1]}) \cdot x$$

$$= dz^{[1]} \cdot x$$

$$= dz^{[1]} \cdot a^{[0]}$$

Derivative of $dL/db^{[1]}$



$$db^{[1]} = \frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

$$= dz^{[1]} \cdot \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

$$= dz^{[1]}$$

Forward propagation

$$z^{[1]} = w^{[1]} \cdot x + b^{[1]}$$

$$= w^{[1]} \cdot A^{[0]} + b^{[1]}$$

$$A^{[1]} = g^{[1]}(z^{[1]})$$

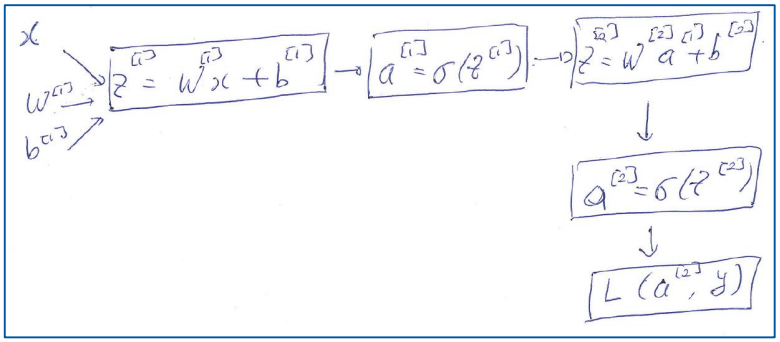
$$= \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]} \cdot A^{[1]} + b^{[2]}$$

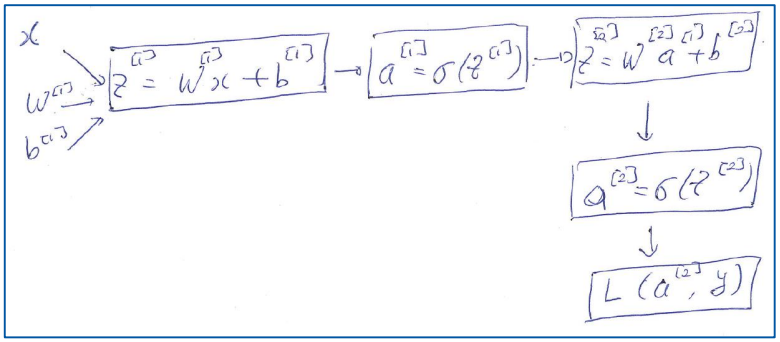
$$A^{[2]} = g^{[2]}(z^{[2]})$$

$$= \sigma(z^{[2]})$$

$$= \hat{Y}$$



Backward propagation



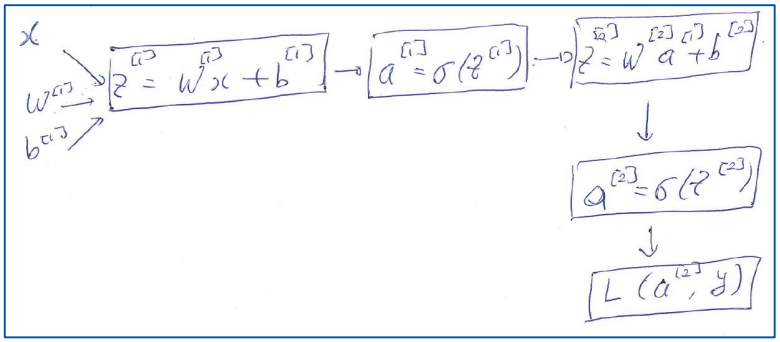
$$dA^{[2]} = -\frac{Y}{A^{[2]}} + \frac{1-Y}{1-A^{[2]}} \rightarrow \text{skip}$$

$$dz^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]}$$

$$db^{[2]} = \frac{1}{m} \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims}=\text{True})$$

Backward propagation



$$dA^{[1]} = dz^{[2]} \cdot W^{[2]}$$

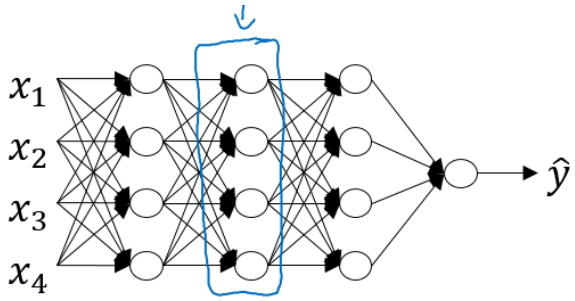
$$dz^{[1]} = dA^{[1]} A^{[1]}(1 - A^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dz^{[1]} \cdot A^{[0]}$$

$$= \frac{1}{m} dz^{[1]} \cdot X$$

$$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims = True)$$

Forward and backward functions



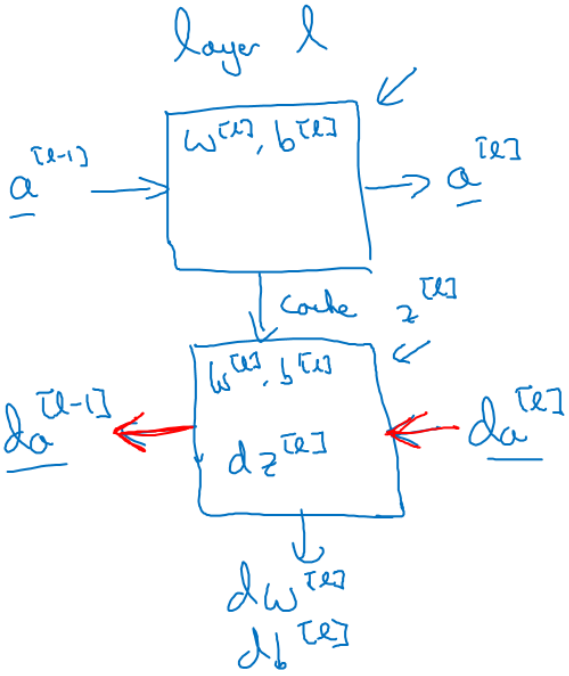
Layer l: $W^{[l]}, b^{[l]}$

→ Forward: Input $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

→ Backward: Input $da^{[l]}$, cache($z^{[l]}$), output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$



<deep learning, Andrew Ng>

Forward and backward propagation

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]^T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]^T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

<deep learning, Andrew Ng>

Initialize weights

Neural network with hidden layers

Weights should be initialized randomly rather than to all zeros.

If weights are initialized to be zero, then, each neuron in the first hidden layer will perform the same computation. So even after multiple iterations of gradient descent, each neuron in the layer will be computing the same thing as other neurons.

<deep learning, Andrew Ng>

Initialize weights

Pitfall: all zero initialization

Assume that approximately half of the weights will be positive and half of them will be negative.

Set all the initial weights to zero.

If every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates.

In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.

<cs231n, Stanford university>

Initialize weights

Small random numbers

Want the weights to be very close to zero, but not identically zero.

Assume that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network.

Initialize the weights of the neurons to small numbers and refer to doing so as symmetry breaking.

W = 0.01* np.random.randn(D,H)

randn samples from a zero mean, unit standard deviation Gaussian.

<cs231n, Stanford university>

Initialize weights

Logistic regression

Logistic regression does not have a hidden layer.

If you initialize the weights to zeros, the first example x fed in the logistic regression will output zero but the derivatives of the logistic regression depend on the input x (because there is no hidden layer) which is not zero.

So at the second iteration, the weights values follow x's distribution and are different from each other if x is not a constant vector.

<deep learning, Andrew Ng>

Xavier initialization

When using sigmoid activation function

Multiplies weights by

$$\frac{\sqrt{2}}{\sqrt{n_{inputs} + n_{outputs}}}$$

<Hands-On ML, Aurelien Geron>

He initialization

When using ReLU activation function (and its variants, including the ELU activation)

Multiplies weights by

$$\sqrt{2}\frac{\sqrt{2}}{\sqrt{n_{inputs}+n_{outputs}}}$$

<Hands-On ML, Aurelien Geron>

Initialization for Hyperbolic tangent

When using tanh activation function

Multiplies weights by

$$4\frac{\sqrt{2}}{\sqrt{n_{inputs}+n_{outputs}}}$$

Weights initialization

Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

<Hands-On ML, Aurelien Geron>

L2 Regularization

Most common form of regularization

Penalize the squared magnitude of all parameters directly.

Heavily penalize peaky weight vectors and prefer diffuse weight vectors.

Network to use all of its inputs a little rather that some of its inputs a lot.

During gradient descent parameter update, every weight is decayed

Linearly towards zero.

<cs231n, Stanford university>

Logistic regression with L2 Regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1-y^i)\log(1-\widehat{y^i})] + \frac{\lambda}{m}\sum_{j=1}^{n} W_j^2$$

Gradient

$$\frac{\partial J}{\partial W_0} = -\frac{1}{m}\sum_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] \quad \text{for j=0}$$

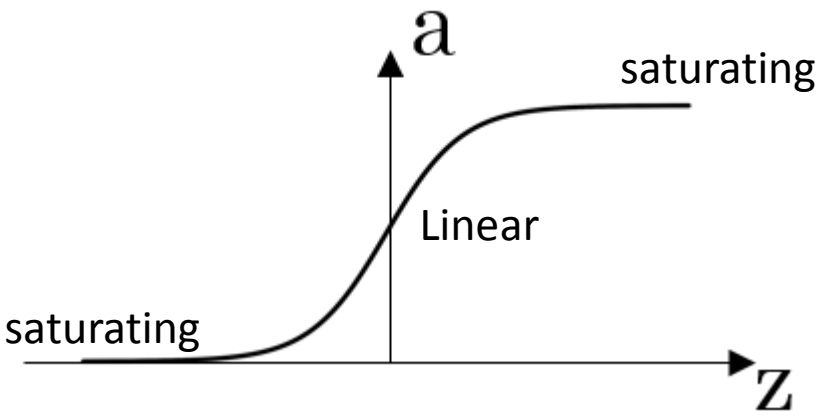
$$\frac{\partial J}{\partial W_j} = -\frac{1}{m}\sum_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] + \frac{2\lambda}{m}W_j \quad \text{for j} \geq 1$$

Neural network with L2 regularization

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)$$

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

Sigmoid

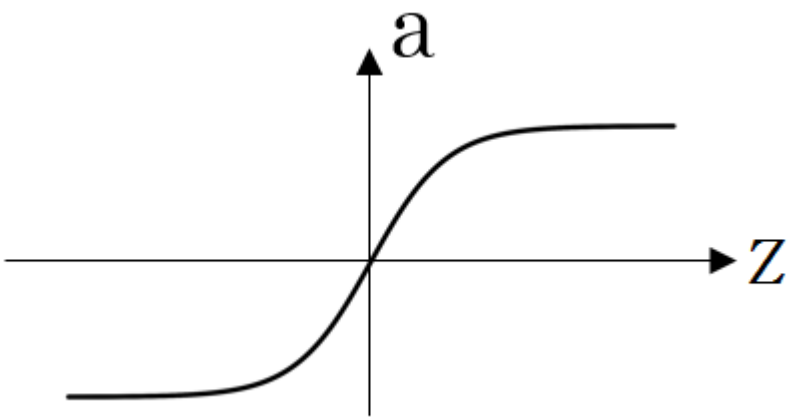


<Deep learning, Andrew Ng>

$$a = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- It can be saturated(0 or 1) when z is large
- Gradients will be almost zero.
- The output is not zero-centered.

tanh



<Deep learning, Andrew Ng>

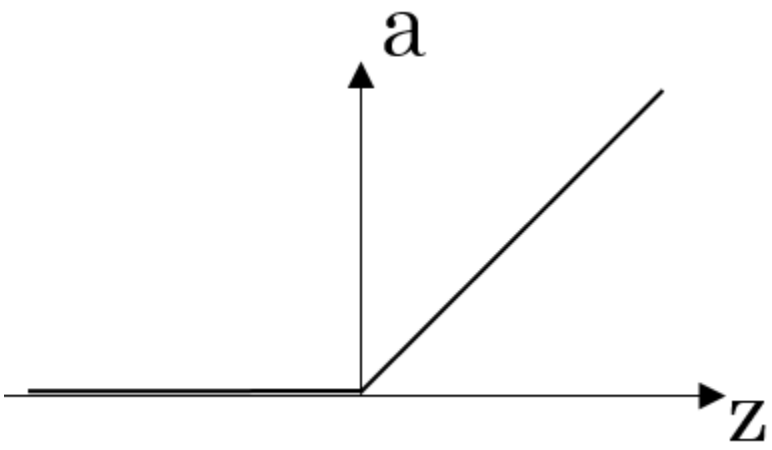
$$a = tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

- The output is zero-centered.
- It is simply a scaled sigmoid function.

$$tanh(z) = 2\sigma(2z) - 1$$

ReLU

Rectified Linear Unit

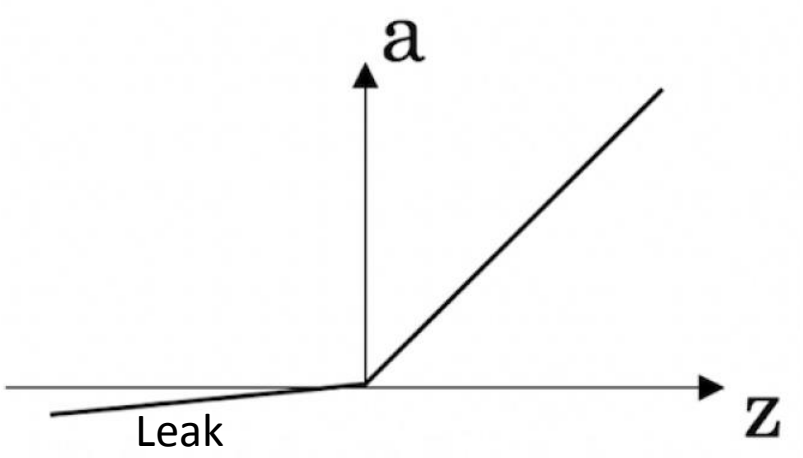


<Deep learning, Andrew Ng>

$$a = ReLU(z) = \max(0, z)$$

- It is zero when z is <=0
- It is linear with slope 1 when z is >0.
- It can learn fast
- It is most commonly used in CNN.
- Some network can be dead since neuron will not activate across entire dataset.

Leaky ReLU



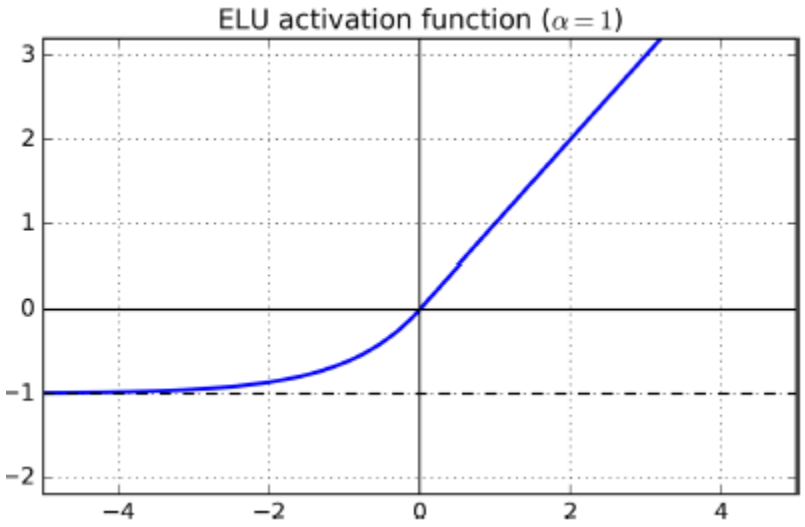
<Deep learning, Andrew Ng>

$$a = Leaky\ ReLU(z) = \max(0.01z, z)$$

- It is linear with small negative slope when z is <=0
- It is linear with slope 1 when z is >0.

ELU

Exponential Linear Unit



<Hands-On ML, Aurelien Geron>

$$a = ELU(z) = \begin{cases} \alpha(\exp(z) - 1) & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$$

- Negative values when z<0 → average output closer to 0
- Nonzero gradient for z<0 → avoid dying units issue
- It is smooth everywhere including z=0

Derivatives of activation functions

Sigmoid function

$$g(z) = a = \frac{1}{1 + \exp(-z)}$$

$$\frac{d}{dz} g(z) = \frac{1}{1 + \exp(-z)} \left(1 - \frac{1}{1 + \exp(-z)}\right)$$

$$g'(z) = g(z)(1 - g(z)$$

$$g'(z) = a(1 - a)$$

Derivatives of activation functions

Tanh function

$$g(z) = a = tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

$$\frac{d}{dz} g(z) = 1 - (\tanh(z))^2$$

$$g'(z) = 1 - g(z)^{\,2}$$

$$g'(z) = 1 - a^2$$

Derivatives of activation functions

ReLU and Leaky ReLU

ReLU

$$a = g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

Leaky ReLU

$$a = g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

Derivatives of activation functions

ELU

$$a = ELU(z) = \begin{cases} \alpha(\exp(z) - 1) & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$$

$$g'(z) = \begin{cases} \alpha \exp(z) & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

L2 regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(a^{[L](i)}) + (1-y^i)\log(1-a^{[L](i)})] + \frac{\lambda}{2m}\sum_{l=1}^{L}\|W^{[l]}\|^2$$

Gradient

$$dW^{[l]} = (from\ backpropagation) + \frac{\lambda}{m}W^{[l]}$$

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$

$$W^{[l]} = W^{[l]} - \alpha[(from\ backprop) + \frac{\lambda}{m}W^{[l]}]$$

$$W^{[l]} = W^{[l]} - \frac{\alpha\lambda}{m}W^{[l]} - \alpha(from\ backprop)$$

Per-layer regularization:
It is not common to regularize different layers to different amount.

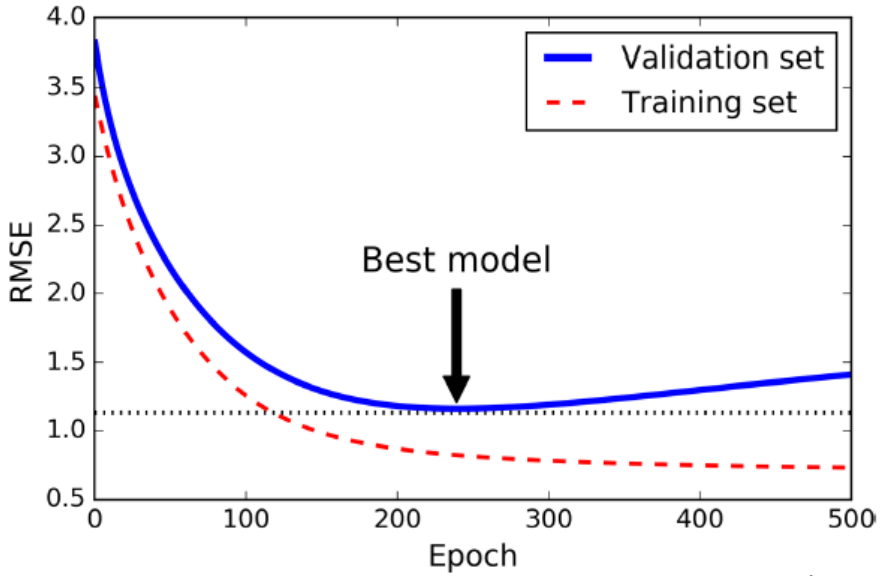
Elastic net regularization

Combine L1 regularization with L2 regularization

$$\lambda_1|W| + \lambda_2 W^2$$

Early Stopping

Stop training as the validation error reaches a minimum.



<Hands-On ML, Aurelien Geron>

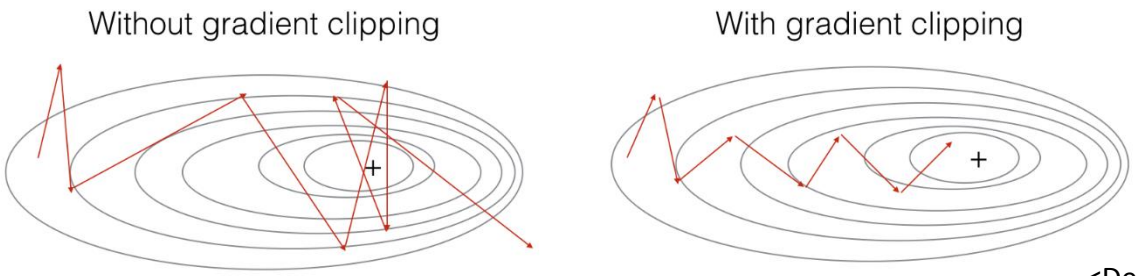
Max norm constraints

Enforce an absolute upper bound on the magnitude of the weight vector for every neuron.

$$\|W\|_2 < c$$

Gradient is clipping to prevent exploding gradients.

Every element of the gradient vector is clipped to lie between some range [-N,N].

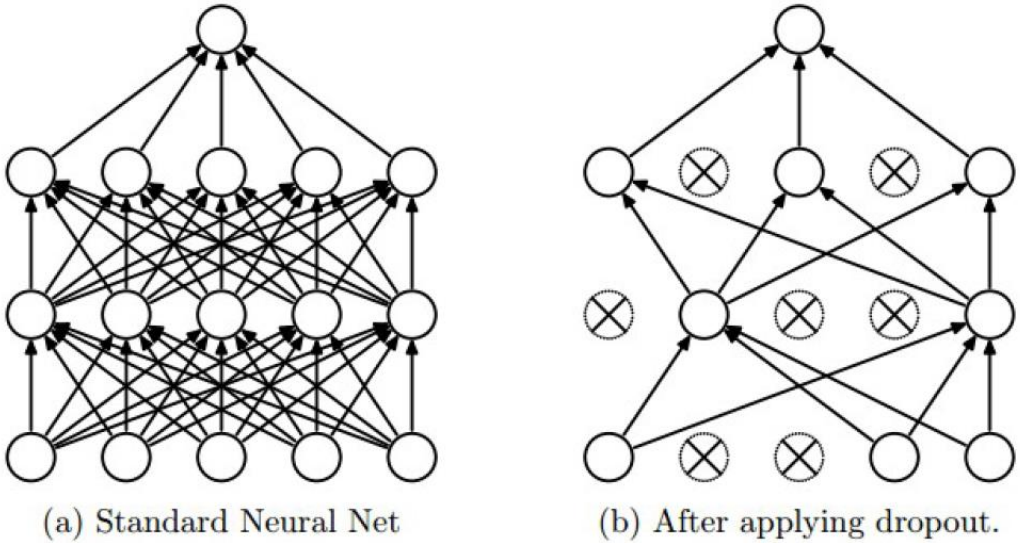


<Deep learning, Andrew Ng>

Drop out

$$px + (1 - p)0$$

Keep a neuron active with some probability p and set to zero others



(a) Standard Neural Net

(b) After applying dropout.

<cs231n, Stanford university>

Forward propagation with Drop out

1. Initialize matrix D1
2. Convert entries of D1 to 0 or 1(using keep prob as threshold)
3. Shut down some neurons of A1
4. Scale the value of neurons that haven't been shut down

```
keep_prob = 0.5
D1 = np.random.rand(A1.shape[0],A1.shape[1])
D1 = np.where(D1<keep_prob,1,0)
A1 = np.multiply(D1,A1)
A1 = A1/keep_prob
```

Backward propagation with dropout

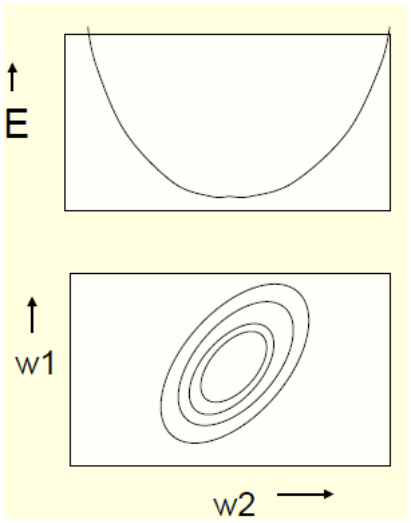
1. Apply mask D1 to shut down the same neurons as during the forward propagation
2. Scale the value of neurons that haven't been shut down

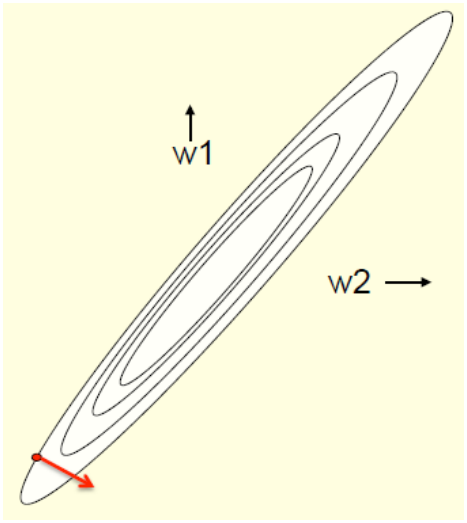
```
dA1 = np.multiply(D1,dA1)
dA1 = dA1/keep_prob
dZ1 = np.multiply(dA1, np.int64(A1 > 0))
```

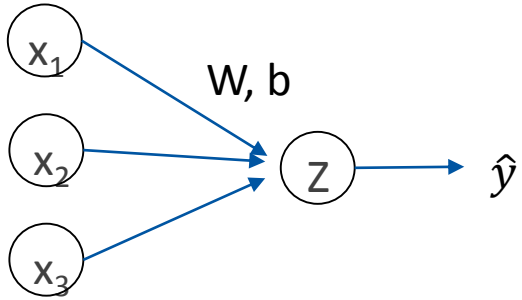
Normalizing inputs to speed up learning

In linear and logistic regression, we normalize input X to speed up learning.

$X = (X - \mu)/\sigma$





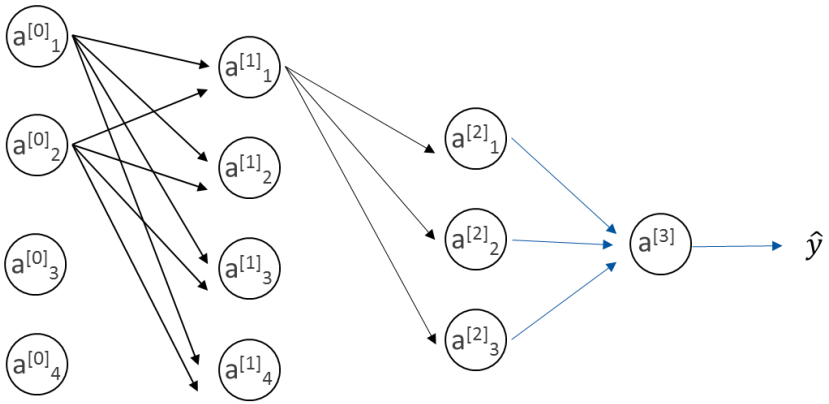


<Neural Networks, Geoffrey Hinton>

Normalizing activations in a network

In Neural Network, we may normalize $a^{[l-1]}$ to train $W^{[l]}$,$b^{[l]}$ faster.

In actual, we normalize $Z^{[l-1]}$ instead of $a^{[l-1]}$.



Batch Normalization

Address the vanishing / exploding gradients problems

Address the problem that the distribution of each layer's inputs changes during training.

Adding an operation in the model just before the activation function of each layer, zero-centering and normalizing the inputs, then scaling and shifting the result using two new parameters per layer

(one for scaling, the other for shifting)

Implementing Batch Norm

Given some intermediate values in NN,

$$\mu = \frac{1}{m}\sum_i Z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (Z^{(i)} - \mu)^2$$

If $\gamma = \sqrt{\sigma^2 + \epsilon}$

$\beta = \mu$

then

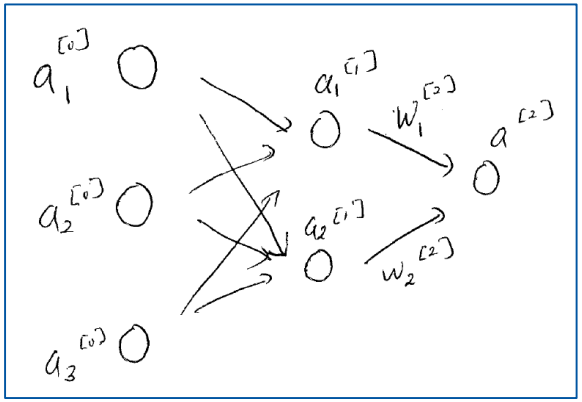
$\tilde{Z}^{(i)} = Z^{(i)}_{norm}$

$$Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{Z}^{(i)} = \gamma Z^{(i)}_{norm} + \beta$$

ε is a smoothing term to avoid division by zero, a tiny number
γ is scaling parameter for a layer
β is shifting parameter for a layer
γ and β *are learnable parameters for model*

Adding Batch Norm to a network



$a^{[0]} \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[\text{Batch Norm}]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]}$

Parameters: $\{ W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \ldots, W^{[L]}, b^{[L]} \}$

$\{ \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \ldots, \beta^{[L]}, \gamma^{[L]} \}$

$\beta^{[l]} := \beta^{[l]} - \alpha d\beta^{[l]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

$z_1^{[1]} = w_{11}^{[1]} \cdot a_1^{[0]} + w_{12}^{[1]} \cdot a_2^{[0]} + w_{13}^{[1]} \cdot a_3^{[0]} + b_1^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$

$z_2^{[1]} = w_{21}^{[1]} \cdot a_1^{[0]} + w_{22}^{[1]} \cdot a_2^{[0]} + w_{23}^{[1]} \cdot a_3^{[0]} + b_2^{[1]}$

$a^{[2]} = \sigma(z^{[2]})$

$z^{[2]} = w_1^{[2]} \cdot a_1^{[1]} + w_2^{[2]} \cdot a_2^{[1]} + b^{[2]}$

Pros & Cons for Batch Normalization

Vanishing gradients problems are strongly reduced

Less sensitive to the weight initialization

Be able to use much larger learning rates

Play as a regularization

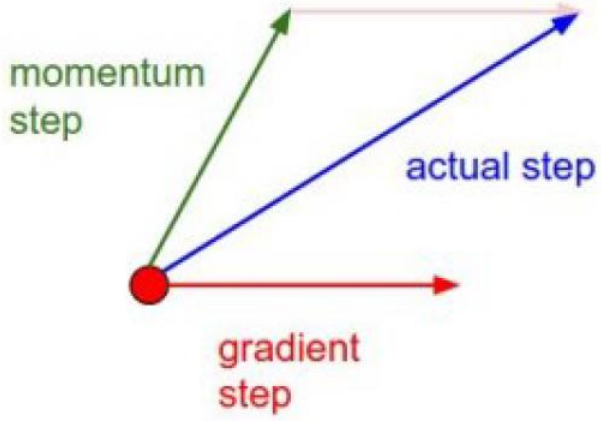
Add complexity to the model

Require Extra computation

Momentum update

Gradient descent update

$x := x - \text{learning rate} * dx$



Momentum update

$v = \beta * v - \text{learning rate} * dx$

$x := x + v$

v : velocity

β : momentum

Adagrad

Adaptive learning rate method

v := v + dx**2

x := x – learning_rate *dx / (sqrt(v) + eps)

v : sum of squared gradients

Normalized the parameter update step, element-wise.

Weights with high gradients will have effective learning rate reduces,

Weights with low gradients will have effective learning rate increases.

Gradient descent with momentum

$V_{dW} = 0, V_{db} = 0$

For every iteration t:

Compute V_{dW}, V_{db} using dW and db

$$V_{dW} := \beta_1 \cdot V_{dW} + (1 - \beta_1) \cdot dW$$

$$V_{db} := \beta_1 \cdot V_{db} + (1 - \beta_1) \cdot db$$

$$W := W - \alpha \cdot V_{dW}$$

$$b := b - \alpha \cdot V_{db}$$

RMS prop

Root Mean square prop

$$S_{dW} \neq 0, S_{db} \neq 0, \epsilon = 10^{-8}$$

For every iteration t:

Compute S_{dW}, S_{db} using dW and db

$$S_{dW} := \beta_2 \cdot S_{dW} + (1 - \beta_2) \cdot dW^2$$

$$S_{db} := \beta_2 \cdot S_{db} + (1 - \beta_2) \cdot db^2$$

$$W := W - \alpha \cdot \frac{dW}{\sqrt{S_{dW} + \varepsilon}}$$

$$b := \mathrm{b} - \alpha \cdot \frac{db}{\sqrt{S_{db} + \varepsilon}}$$

Adam Optimization

Adaption moment estimation

$$V_{dW} = 0, V_{db} = 0, S_{dW} = 0, S_{db} = 0, \epsilon = 10^{-8}$$

For every iteration t:

Compute V_{dW}, V_{db} , S_{dW}, S_{db} using dW and db

$$V_{dW} := \beta_1 \cdot V_{dW} + (1 - \beta_1) \cdot dW$$

$$V_{db} := \beta_1 \cdot V_{db} + (1 - \beta_1) \cdot db$$

$$S_{dW} := \beta_2 \cdot S_{dW} + (1 - \beta_2) \cdot dW^2$$

$$S_{db} := \beta_2 \cdot S_{db} + (1 - \beta_2) \cdot db^2$$

$$W := W - \alpha \cdot V_{dW}$$

$$b := b - \alpha \cdot V_{db}$$

Adam Optimization

Bias correction

$$V_{dW}^{cor} = \frac{V_{dW}}{1 - \beta_1^t}$$

$$V_{db}^{cor} = \frac{V_{db}}{1 - \beta_1^t}$$

$$S_{dW}^{cor} = \frac{S_{dW}}{1 - \beta_2^t}$$

$$S_{db}^{cor} = \frac{V_{db}}{1 - \beta_2^t}$$

$$W := W - \alpha \cdot \frac{V_{dW}^{cor}}{\sqrt{S_{dW}^{cor} + \varepsilon}}$$

$$b := \mathrm{b} - \alpha \cdot \frac{V_{db}^{cor}}{\sqrt{S_{db}^{cor} + \varepsilon}}$$

Adam optimization

Hyper parameters choices

α : need to be tune

β_1 : 0.9 (momentum)

β_2 : 0.999 (RMS prop)

ε : 10^{-8}

Learning rate decay

Slowing decreasing α

If learning rate is large, parameters decay too aggressively and the parameters are unable to reach the local minimum.

1 epoch : 1 pass through data set, iteration number

- 1/t decay
- Exponentially decay
- Step decay: discrete staircase
- Manual decay

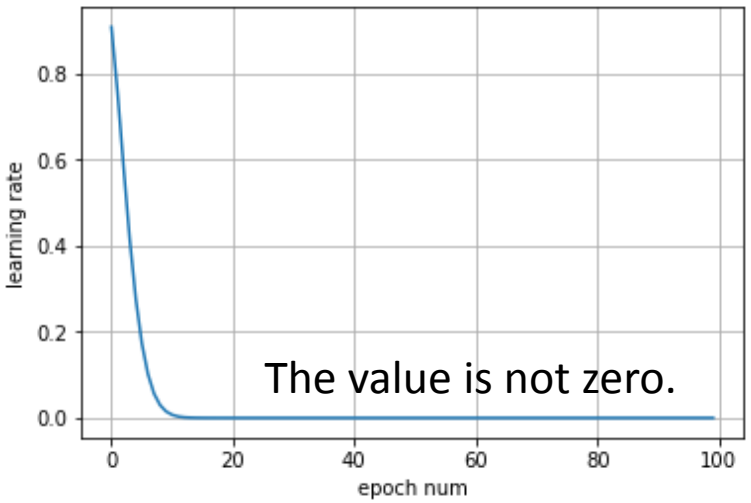
1/t decay

$$\alpha = \frac{\alpha_0}{1 + decay - rate \times epoch - num}$$

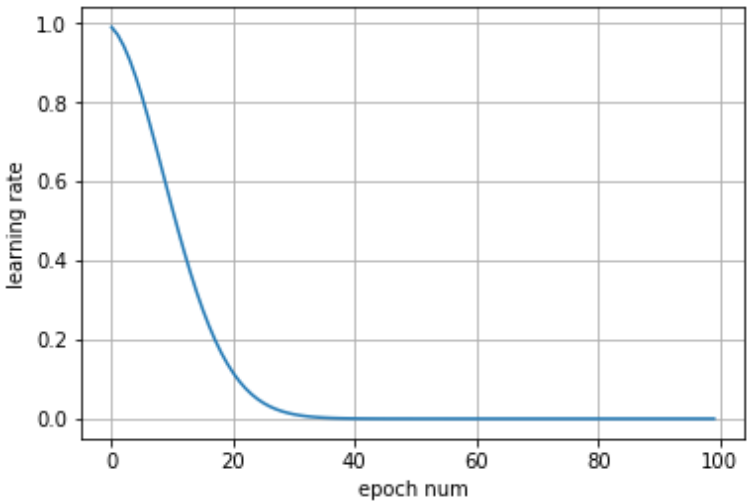
$$\alpha = \frac{\alpha_0}{1 + kt}$$

$\alpha_0 = 1$

decay rate = 0.1



decay rate = 0.01

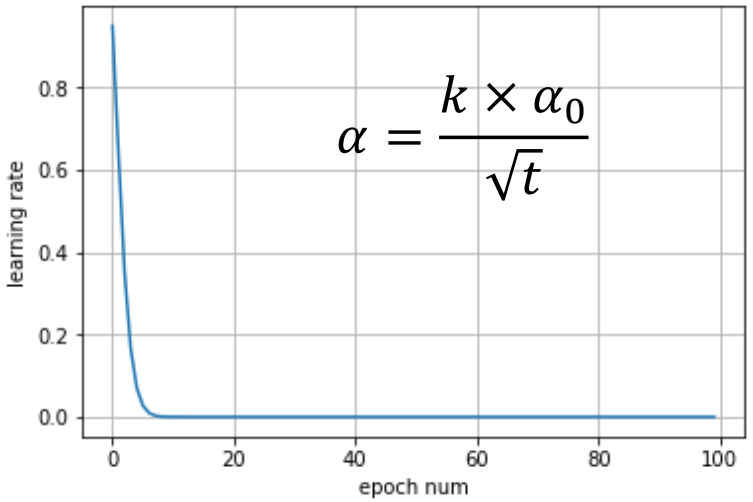


1/t decay

$$\alpha = \frac{k \times \alpha_0}{\sqrt{epoch - num}}$$

$\alpha_0 = 1$

k= 0.95

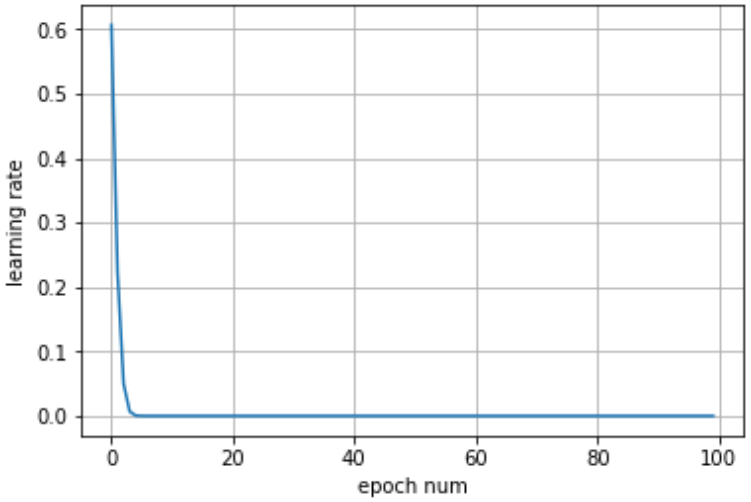


Exponential decay

$$\alpha = \alpha_0 \cdot \exp(-kt)$$

$\alpha_0 = 1$

k= 0.5

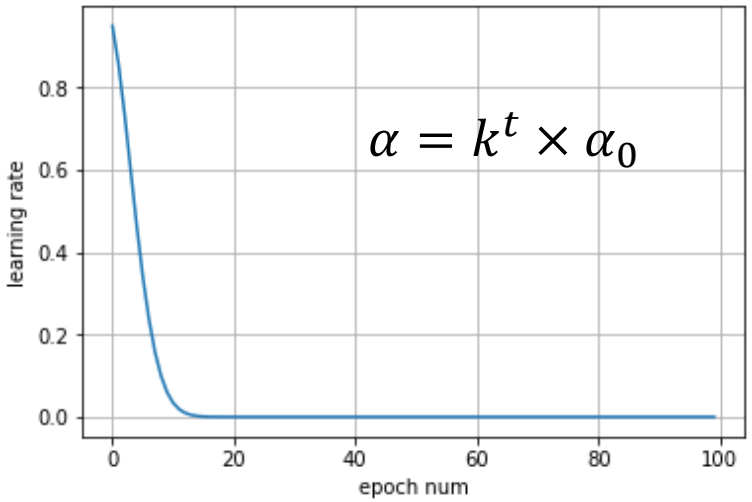


Exponentially decay

$$\alpha = k^{epoch\_num} \times \alpha_0$$

$\alpha_0 = 1$

k= 0.95



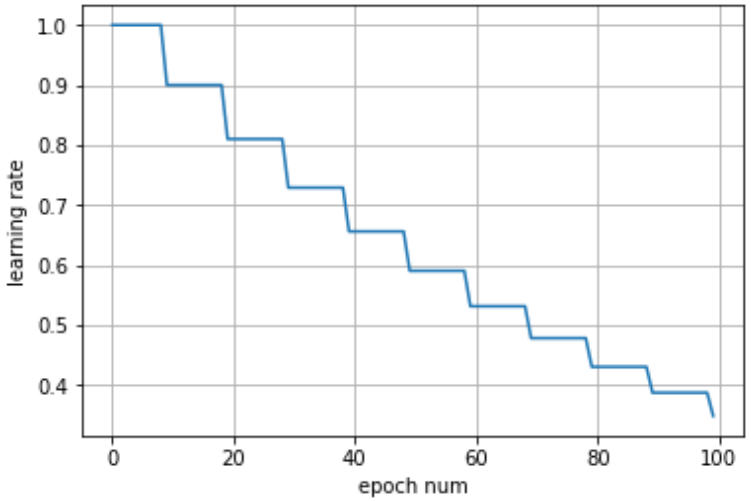
Discrete staircase

$\alpha = k \times \alpha$ with every few epochs

$\alpha_0 = 1$

k= 0.9

every 10 epochs



Learning rate decay

Annealing learning rate

Hyper parameter k: [0.01, 0.1, 0.5, 0.9, 0.95, 0.99]

$$\alpha = \frac{\alpha_0}{1 + kt} \qquad \alpha = \frac{k \times \alpha_0}{\sqrt{t}}$$

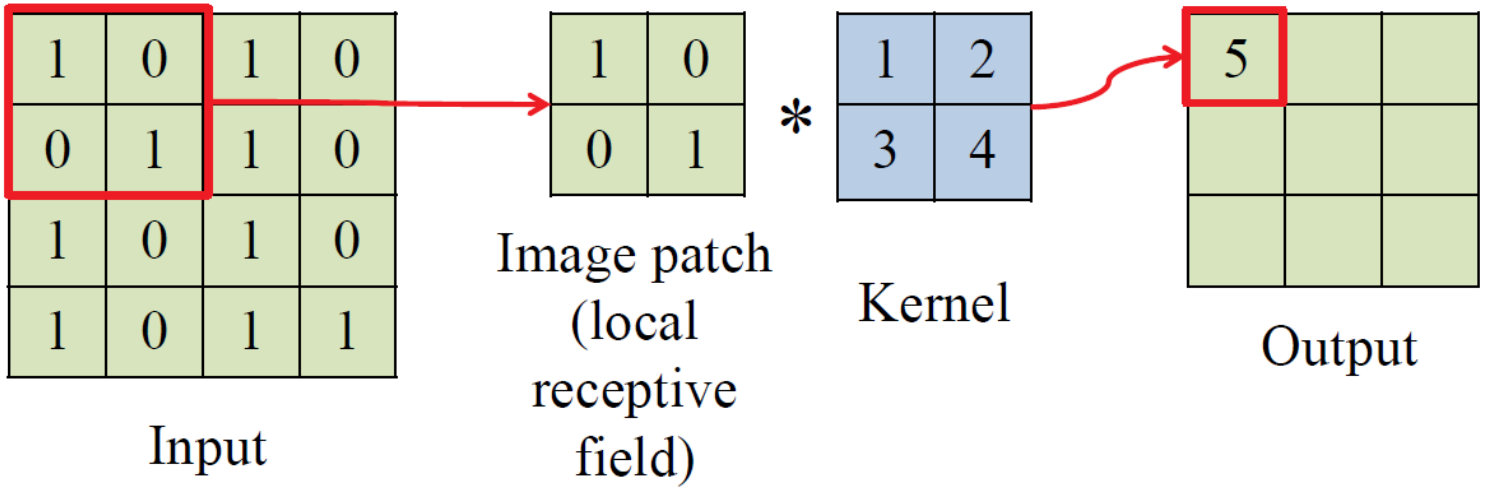
$$\alpha = \alpha_0 \cdot \exp(-kt) \qquad \alpha = k^t \times \alpha_0$$

$\alpha = k \times \alpha$ with every t

Convolutions

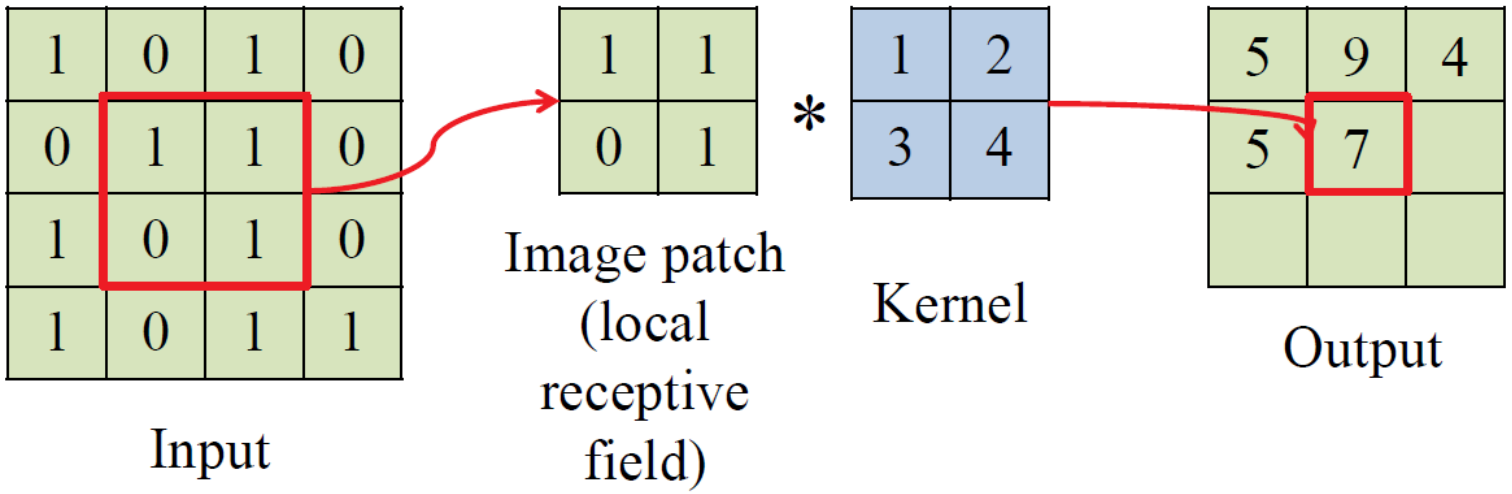
Convolution is a dot product of a **kernel** (or filter)

and a patch of an image (**local receptive field**) of the same size



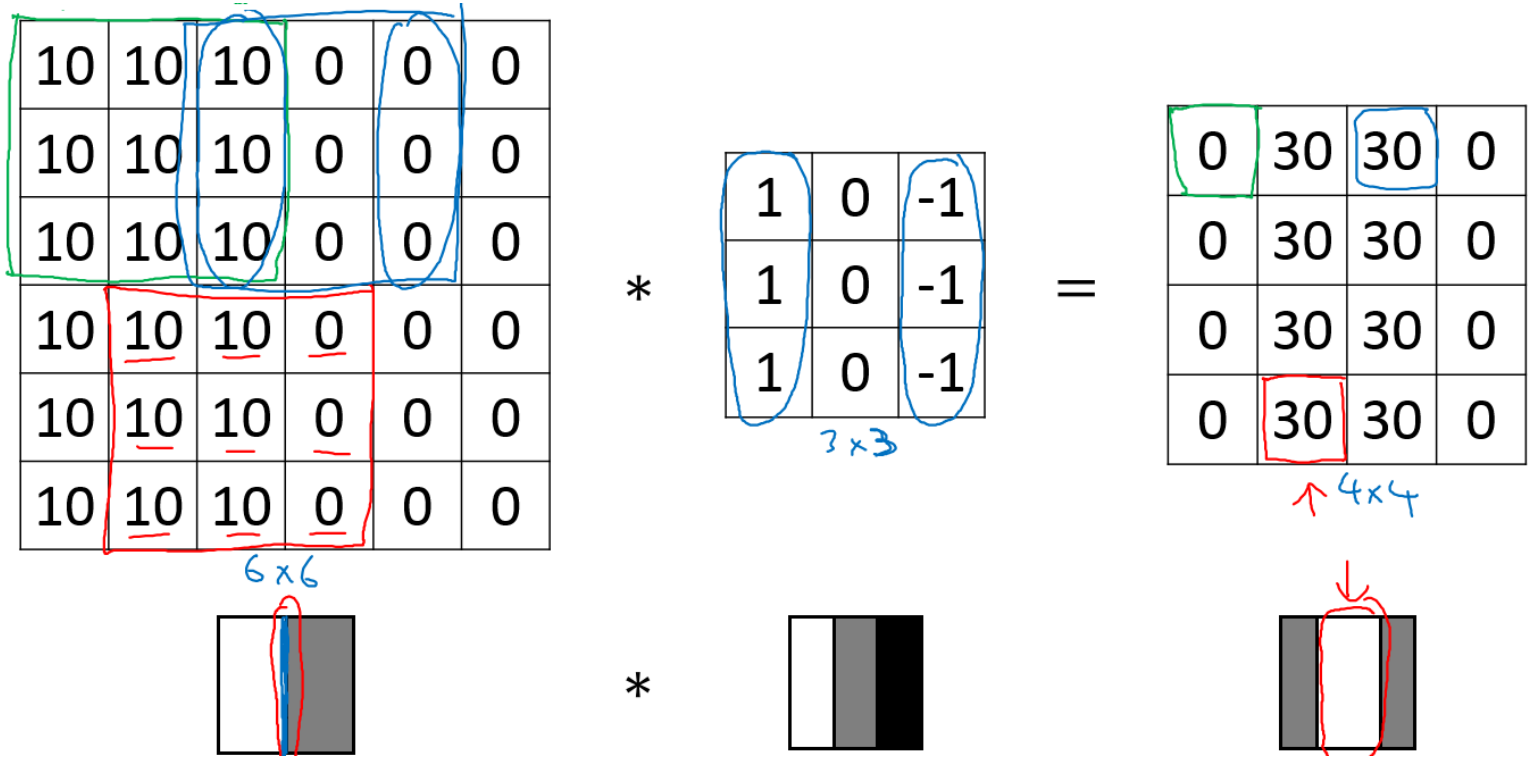
<Intro to deep learning, national research university>

Convolutions



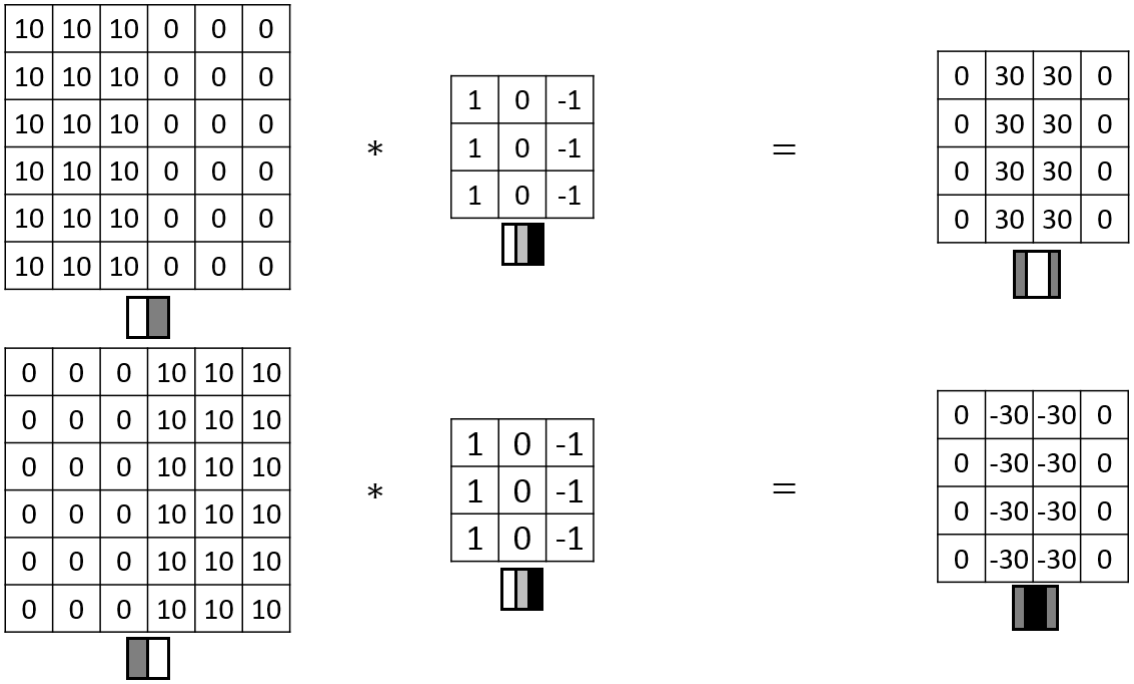
<Intro to deep learning, national research university>

Vertical edge detection



<Deep Neural Network, Andrew Ng>

Vertical edge detection



<Deep Neural Network, Andrew Ng>

Horizontal edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

6 x 6

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

<Deep Neural Network, Andrew Ng>

Edge detection

Kernel

-1	-1	-1
-1	8	-1
-1	-1	-1

$*$ $=$

Edge detection

Original image

Sums up to 0 (black color) when the patch is a solid fill

<Intro to deep learning, national research university>

Sharpening

Kernel

-1	-1	-1
-1	8	-1
-1	-1	-1

* = 

Edge detection

Original image

0	-1	0
-1	5	-1
0	-1	0

* = 

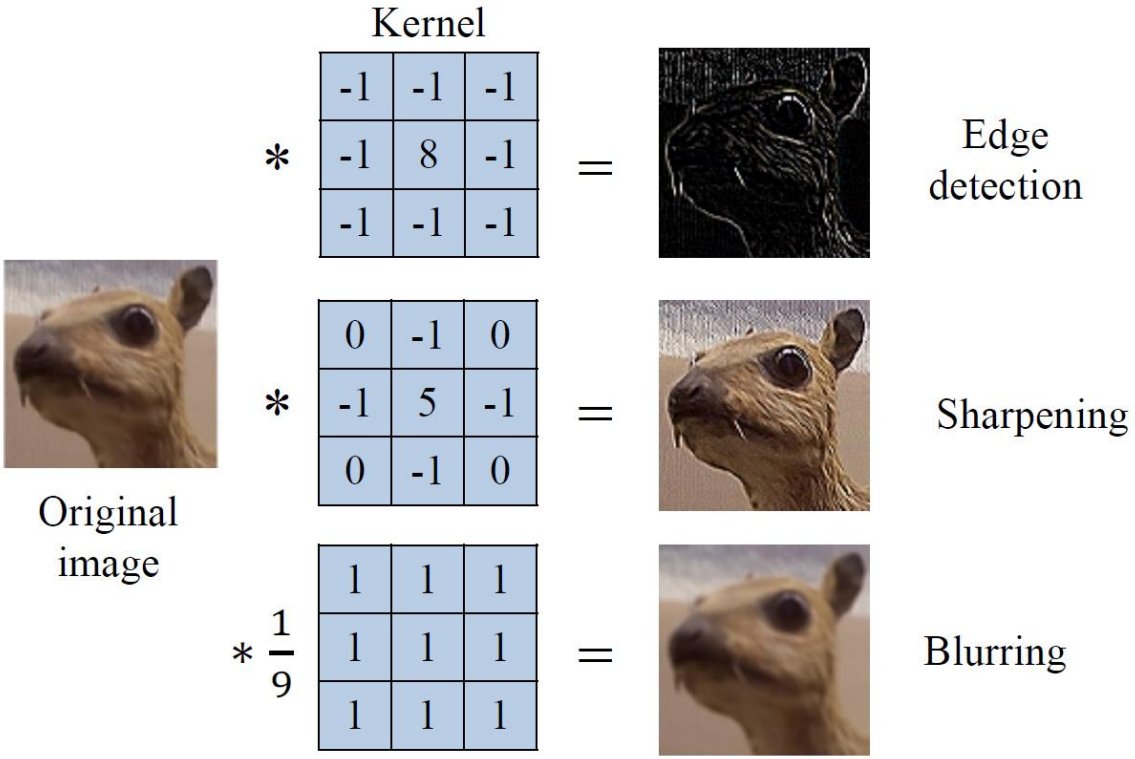
Sharpening

Doesn't change an image for solid fills

Adds a little intensity on the edges

<Intro to deep learning, national research university>

Blurring



<Intro to deep learning, national research university>

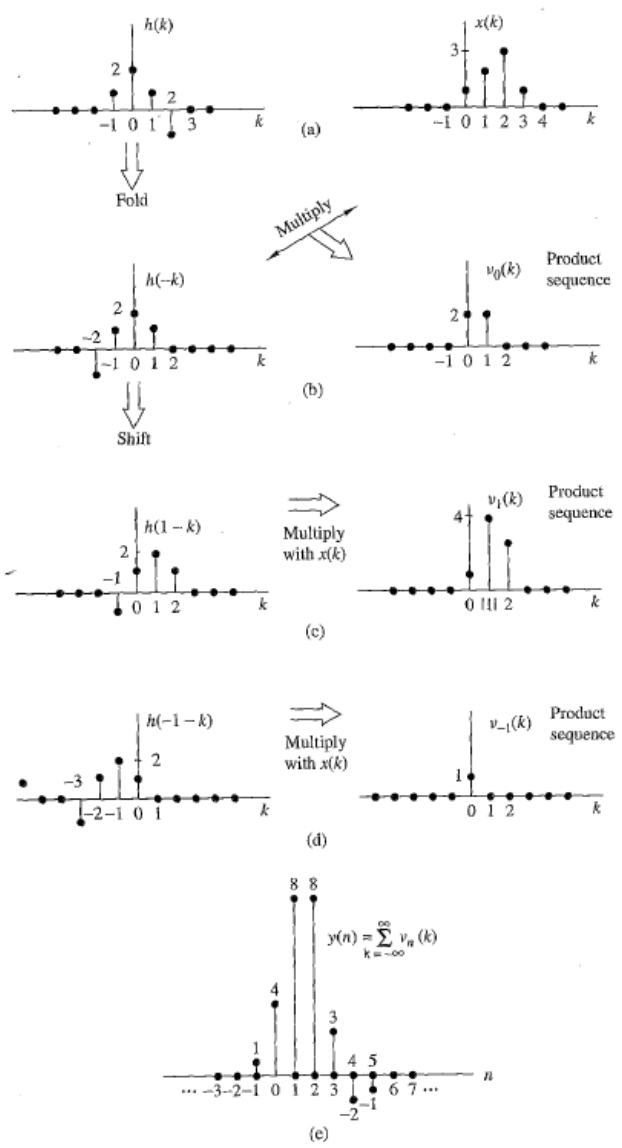
Convolution

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

<Digital signal processing, J.G., Proakis et al.>

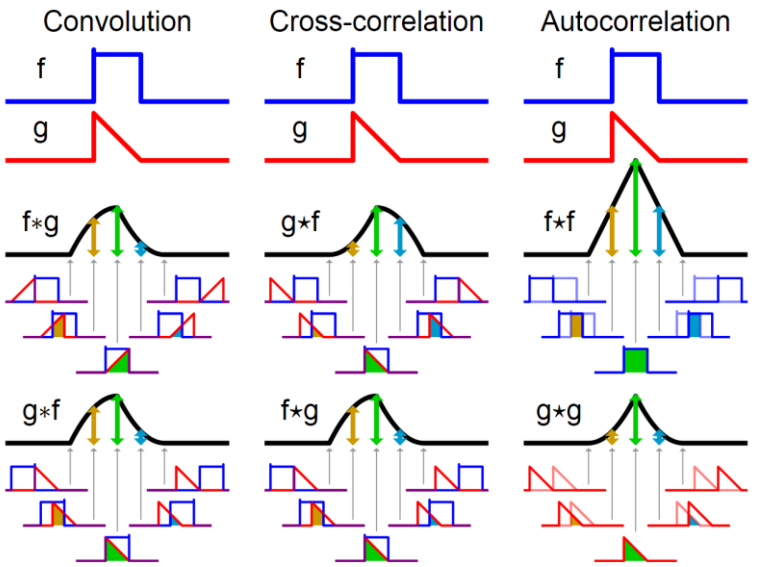
Convolution is a mathematical operation that slides one function over another and measures the integral of their pointwise multiplication. It is heavily used in signal processing.

<Hands-On ML, A. Geron>



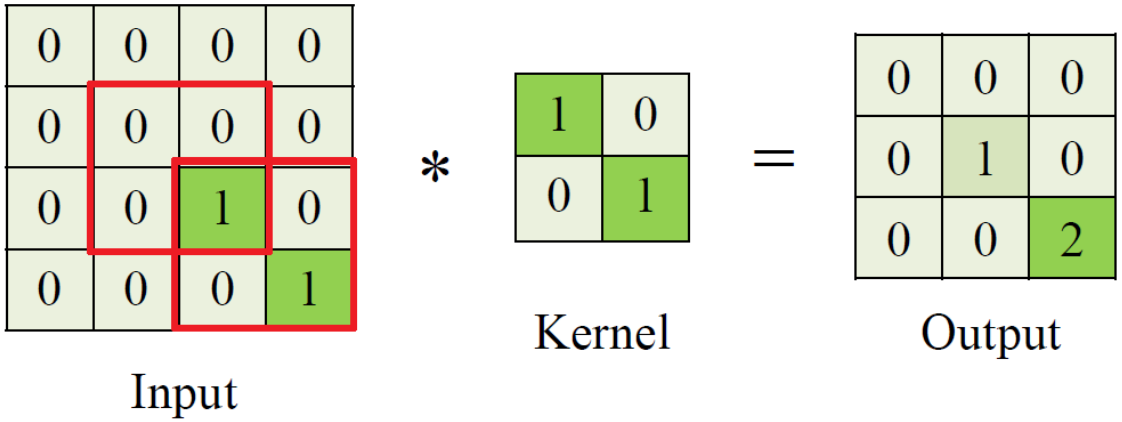
Convolution

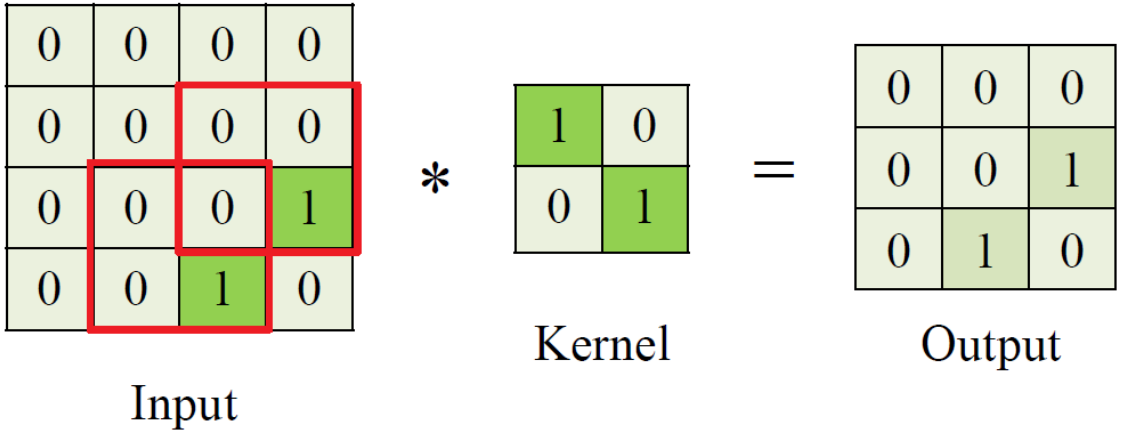
Convolutional layers actually use cross-correlations. <Hands-On ML, A. Geron>



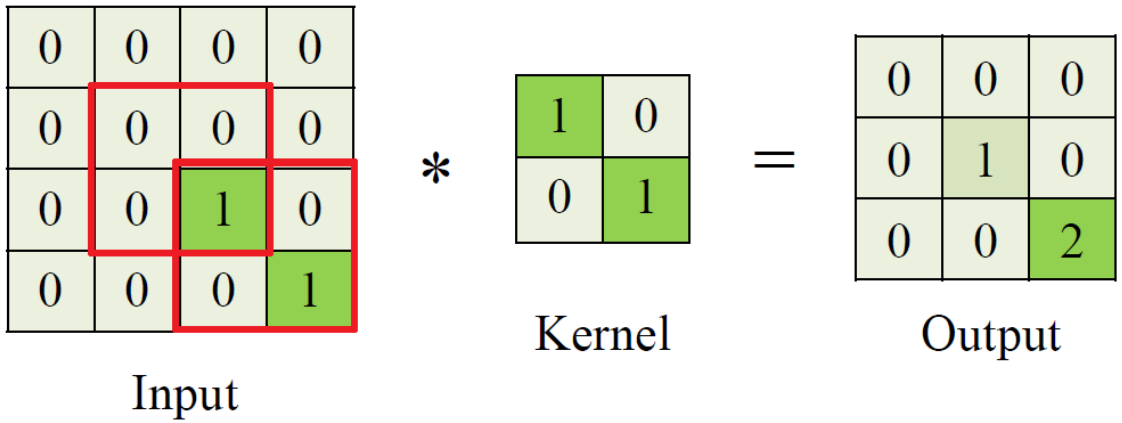
<Wikipedia>

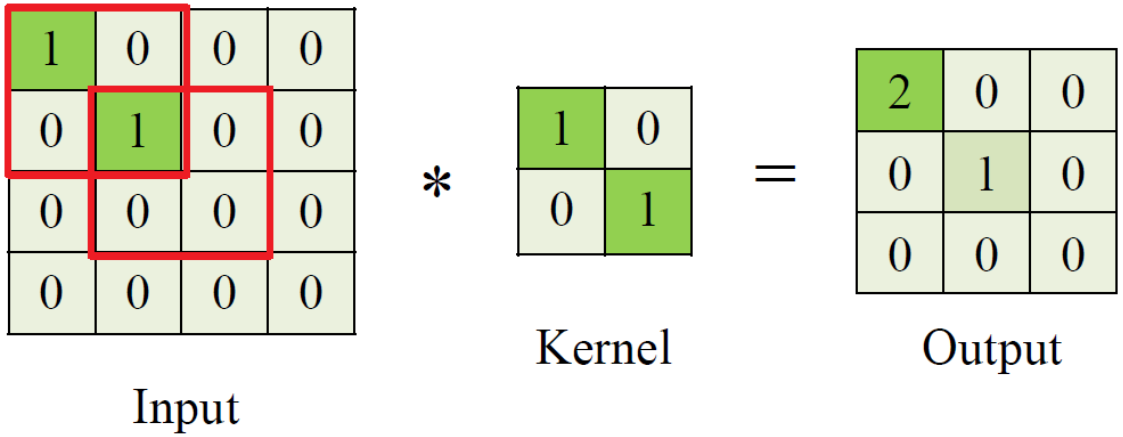
Convolution as correlation



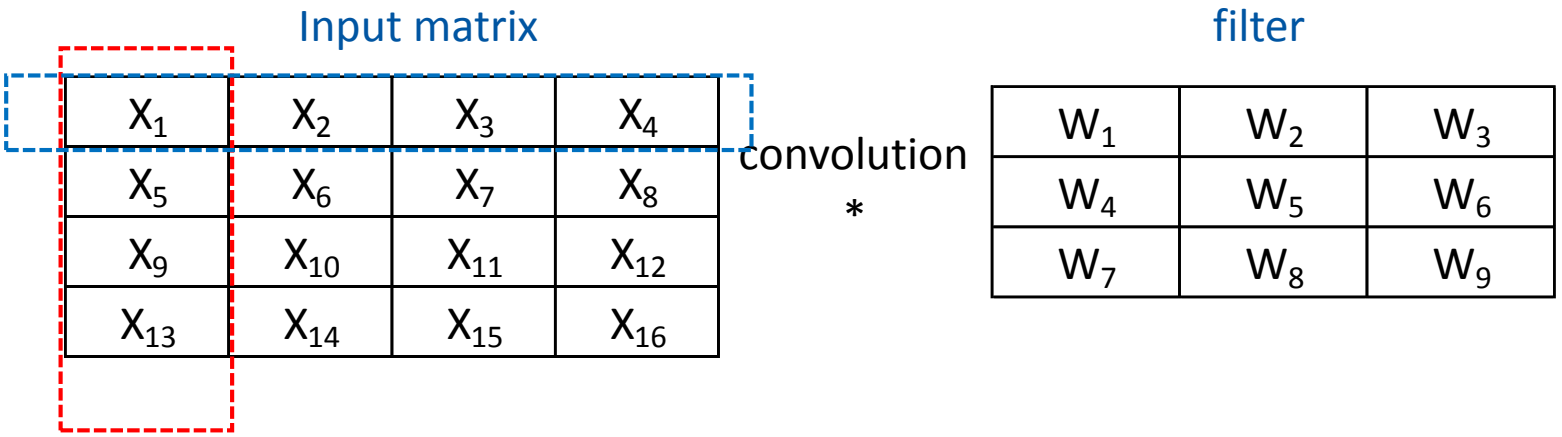


Convolution as translation





Edge pixels?



What about information from edge pixels?

Padding

0	0	0	0	0	0
0	X_1	X_2	X_3	X_4	0
0	X_5	X_6	X_7	X_8	0
0	X_9	X_{10}	X_{11}	X_{12}	0
0	X_{13}	X_{14}	X_{15}	X_{16}	0
0	0	0	0	0	0

*

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

=

Z_1	Z_2	Z_3	Z_4
Z_5	Z_6	Z_7	Z_8
Z_9	Z_{10}	Z_{11}	Z_{12}
Z_{13}	Z_{14}	Z_{15}	Z_{16}

Padding

To detect the information of edge or border

To prevent shrinking(compression) of size of next layer

- Valid convolution: without padding

(n x n) * (f x f) → (n-f+1) x (n-f+1)

(6 x 6) * (3x3) → (4 x 4)

- Same convolution: with zero padding, output size is the same as the input size

(n x n) * (f x f) → (n+2p-f+1) x (n+2p-f+1)

(6 x 6) * (3x3) → (6 x 6)

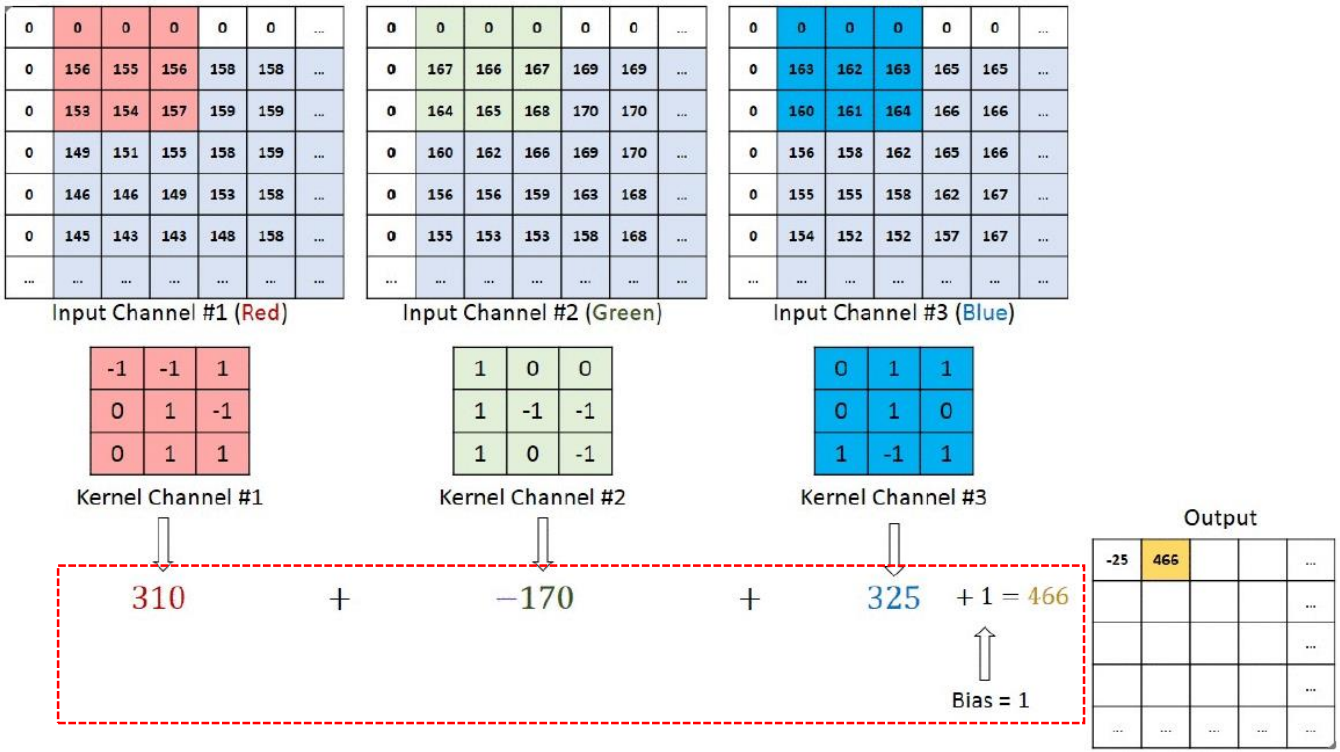
Summary of convolutions

- n x n image
- f x f filter
- padding p
- stride s

Output size:

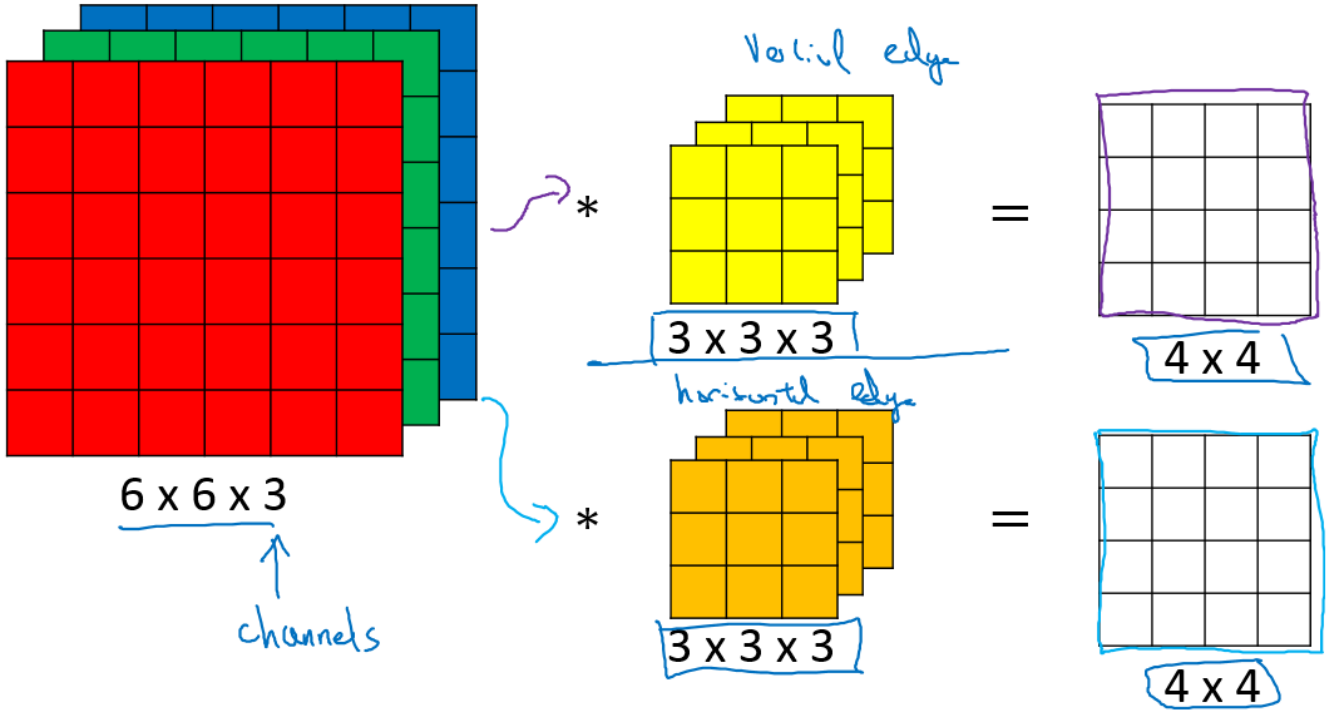
$$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$$

Convolutions on RGB images



<understanding convolutional layers in convolutional neural networks>
Machinelearninguru.com/computer_vision/basics/convolution/convolution_layer.html

Multiple filters (channels)

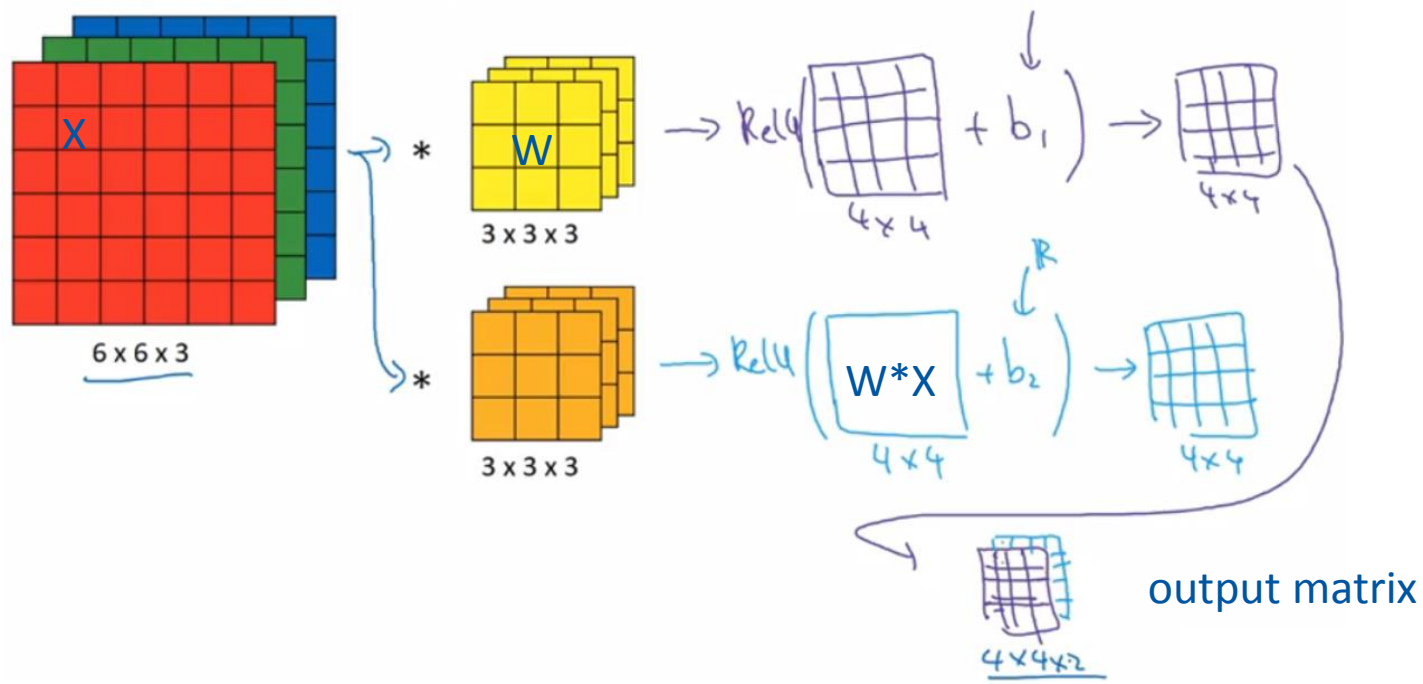


<deep learning, Andrew Ng>

Multiple filters (channels)

Z=W*X + b

a=ReLU(Z)



<deep learning, Andrew Ng>

Number of parameters in one layer

If there are 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?

For 1 filter:

3 x 3 x 3 = 27 for W

1 for bias

For 10 filters:

Then, 28 * 10 = 280

Summary of size in matrix

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

Input size: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output size: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

$$n_W^{[l]} = \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $1 \times 1 \times 1 \times n_c^{[l]}$

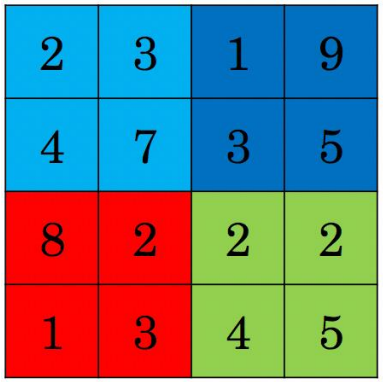
Pooling layers

No parameters to learn

- To subsample (shrink) the input image in order to reduce the computational load, the memory usage, the number of parameters (reduce overfitting)

- Max Pooling : more commonly use
- Average Pooling

Pooling

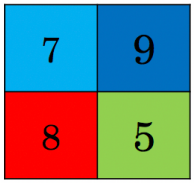


Hyperparameters

f=2

s=2

Max:



Average:

<Deep Learning, Andrew Ng>

Size of Pooling

- Hyperparameters
  - f: filter size
  - s : stride
  - p : padding (usually p=0)

- Max or average pooling
- No parameters to learn

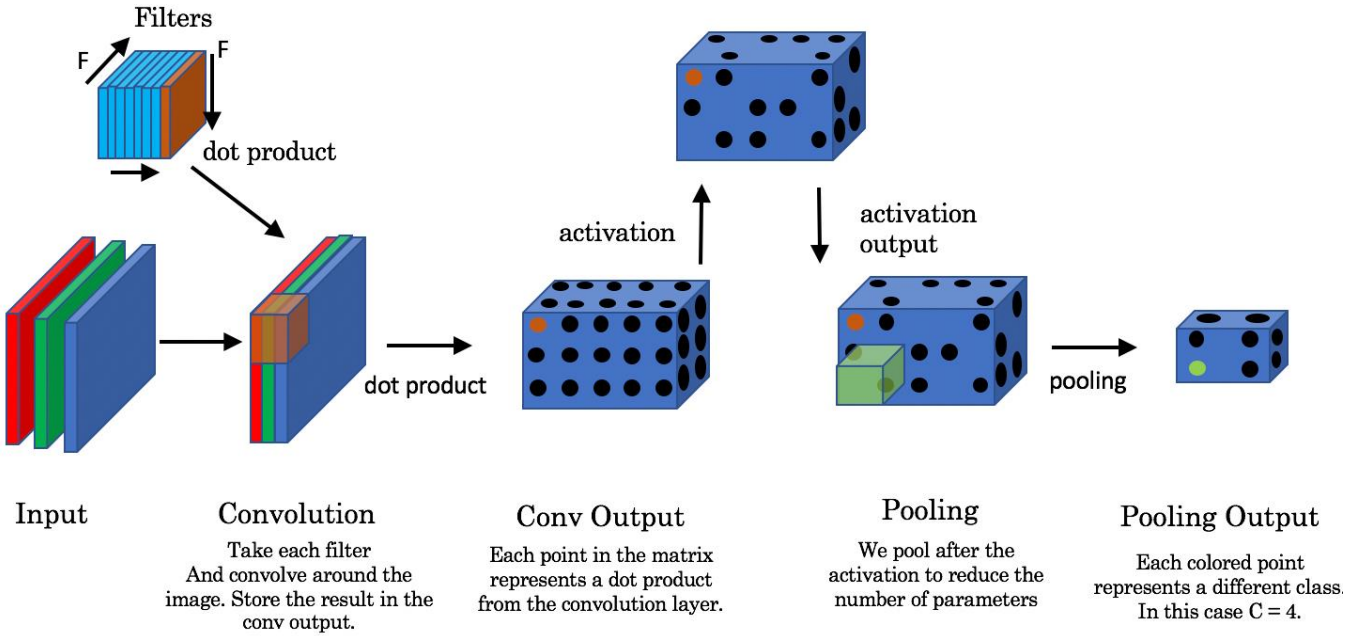
Input size: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output size: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_H^{[l]} = \frac{n_H^{[l-1]} - f^{[l]}}{s^{[l]}} + 1$$

$$n_W^{[l]} = \frac{n_W^{[l-1]} - f^{[l]}}{s^{[l]}} + 1$$

Convolution and Pooling



<Deep Learning, Andrew Ng>

Summary of Pooling

- Hyperparameters
  - f: filter size
  - s : stride
  - p : padding (usually p=0)

- Max or average pooling
- No parameters to learn

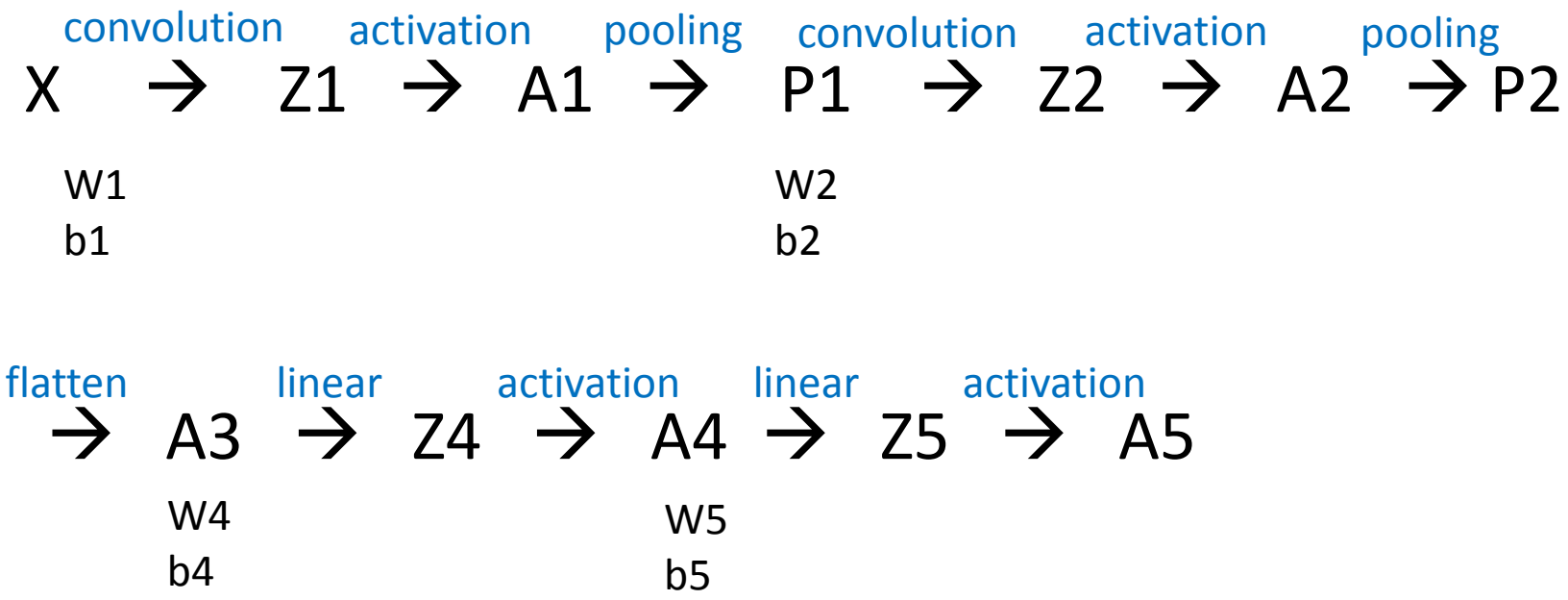
Input size: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output size: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

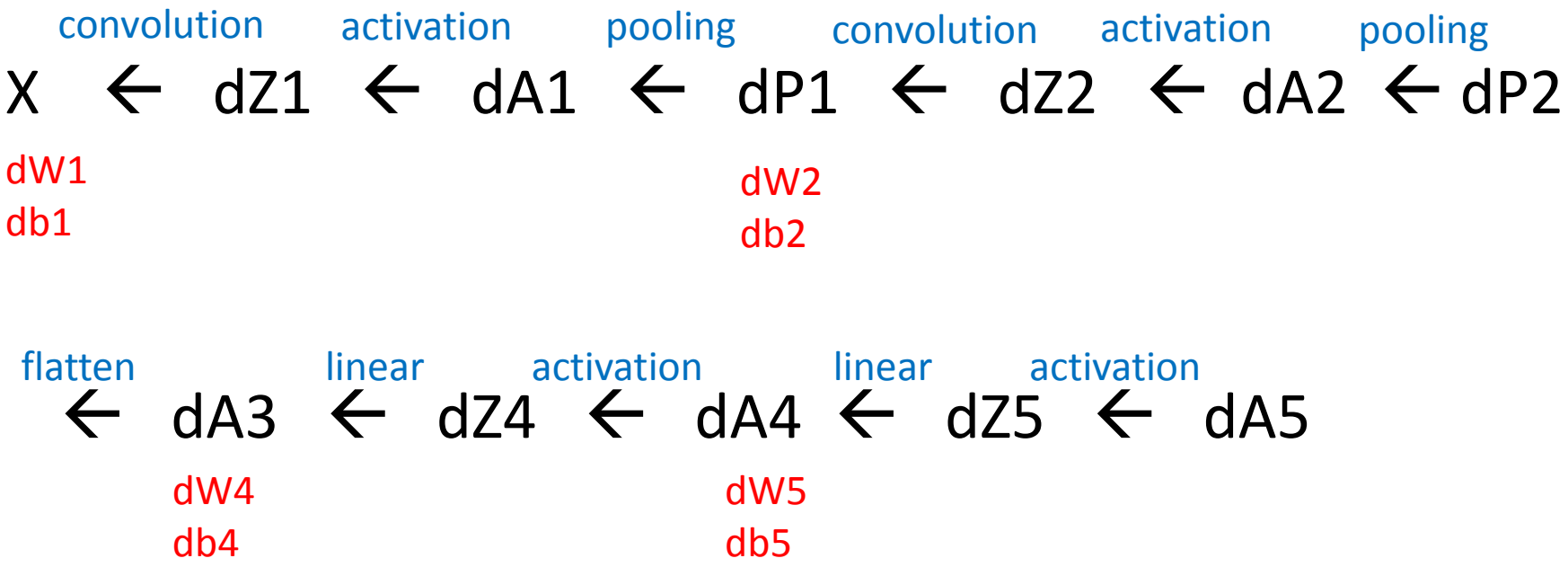
$$n_H^{[l]} = \frac{n_H^{[l-1]} - f^{[l]}}{s^{[l]}} + 1$$

$$n_W^{[l]} = \frac{n_W^{[l-1]} - f^{[l]}}{s^{[l]}} + 1$$

Forward propagation step



Backward propagation step



Pooling backward

average

dA := dP / (distribute_value) * np.ones of the shape <Deep Learning, Andrew Ng>

dP = 1 (scalar)

shape : (n_H, n_W) of the output matrix

distribute_value = n_H*n_W

```
In [15]: dP = 1

In [24]: n_H, n_W = (2,2)

In [25]: distribute_value = n_H*n_W

In [26]: dA = dP*np.ones((n_H,n_W))/float(distribute_value)

In [27]: dA

Out[27]: array([[ 0.25,  0.25],
                [ 0.25,  0.25]])
```

Pooling backward

max : dA := mask*dP <Deep Learning, Andrew Ng>

```
In [28]: A = np.arange(4).reshape((2, 2))
         A
Out[28]: array([[0, 1],
                [2, 3]])
```

Pooling forward →

```
In [31]: P = np.max(A)
         P
Out[31]: 3
```

```
In [29]: mask = (A==np.max(A))
         mask
Out[29]: array([[False, False],
                [False,  True]], dtype=bool)
```

dP=1 → Pooling backward →

```
In [33]: dA = mask*dP
         dA
Out[33]: array([[0, 0],
                [0, 1]])
```

Convolution layer backward

dA

<Deep Learning, Andrew Ng>

$$dA+ = \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} W_c \times dZ_{hw}$$

```
da [vert_start:vert_end, horiz_start:horiz_end, :] += W[:,:,:,c] * dZ[i, h, w, c]
```

dW

$$dW_c+ = \sum_{h=0}^{n_H} \sum_{w=0}^{n_W} a_{slice} \times dZ_{hw}$$

```
dW[:,:,:,c] += a_slice * dZ[i, h, w, c]
```

Convolution layer backward

db

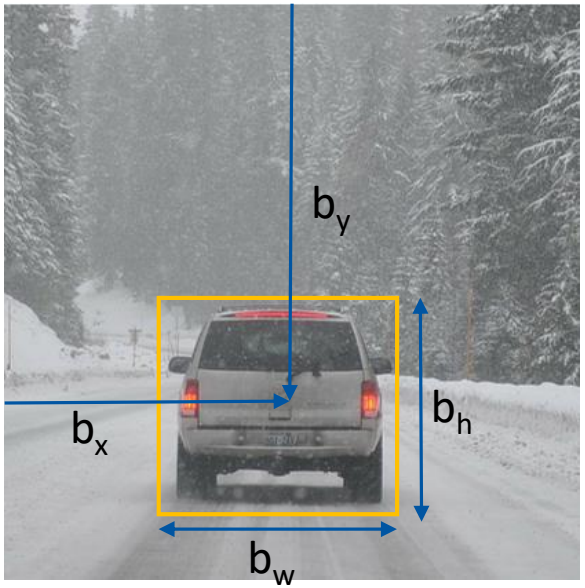
<Deep Learning, Andrew Ng>

$$db = \sum_h \sum_w dZ_{hw}$$

```
db[:,:,:,c] += dZ[i, h, w, c]
```

Classification with localization

(0,0)



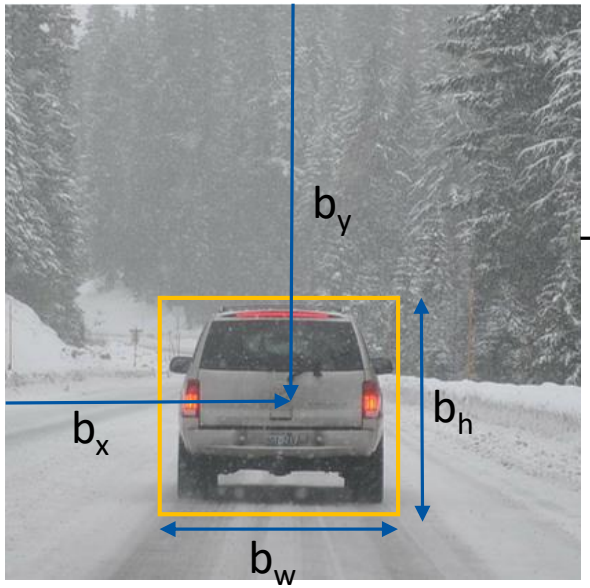
(1,1)

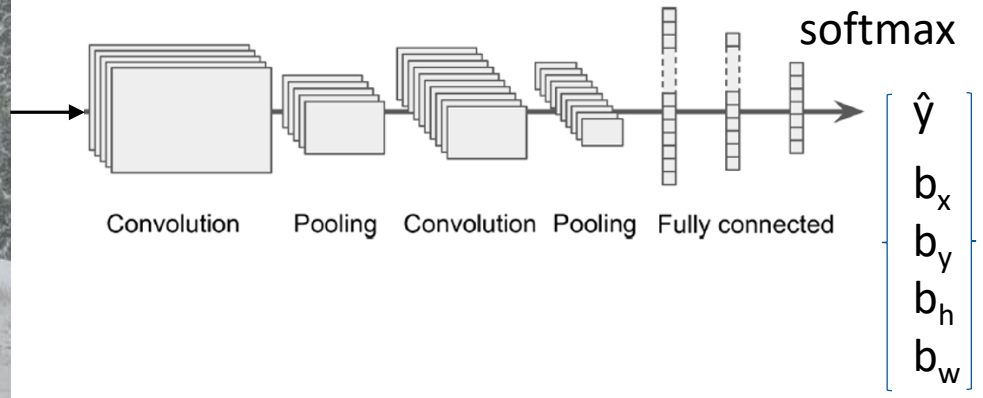
Bounding box
$b_x = 0.5$
$b_y = 0.7$
$b_h = 0.3$
$b_w = 0.4$

<deep learning, Andrew Ng>

Classification with localization

(0,0)

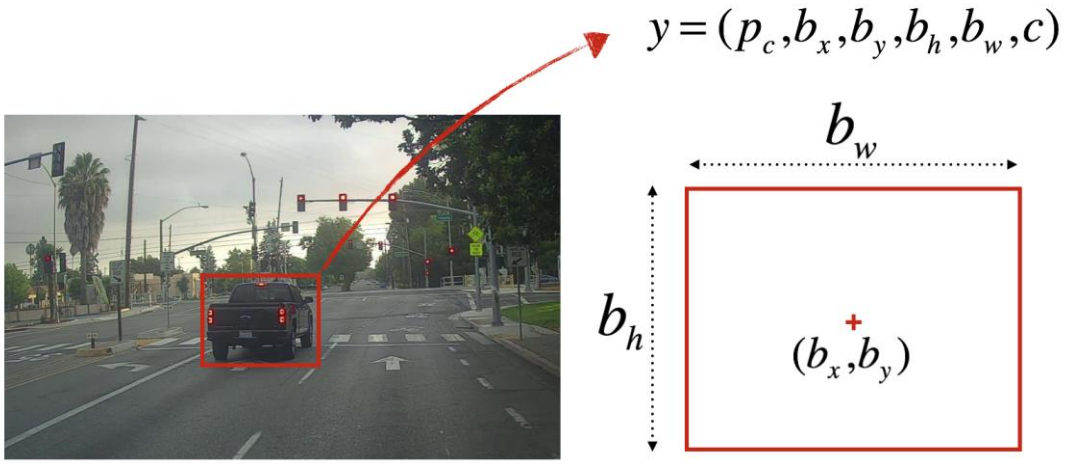




(1,1)

<deep learning, Andrew Ng>

Example of bounding box



$p_c = 1$: confidence of an object being present in the bounding box

$c = 3$: class of the object being detected (here 3 for "car")

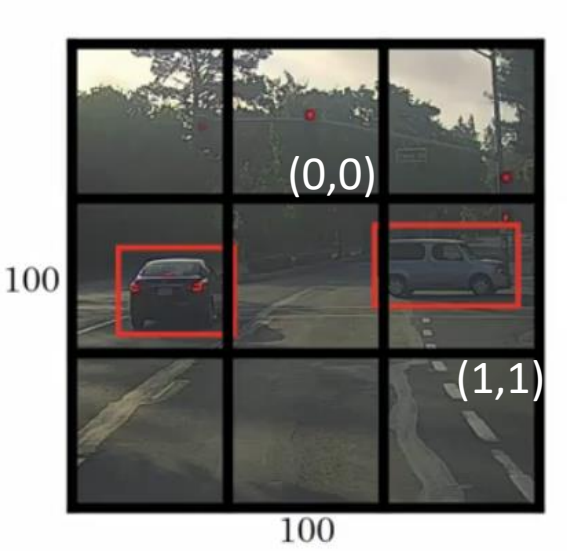
<deep learning, Andrew Ng>

Sliding windows detection

Instead of one size,
Many different size of windows (from small one to larger one) are needed to apply.

<Deep Learning, Andrew Ng>

Specify bounding boxes



$$\begin{bmatrix} \hat{y} \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

$0 < b_x, b_y < 1$

$0 < b_h, b_w$

<Deep Learning, Andrew Ng>

Redmon et al., 2015, You Only Look Once: Unified real-time object detection

YOLO (you only look once)

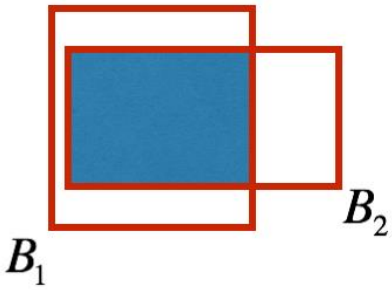
For each grid cell:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

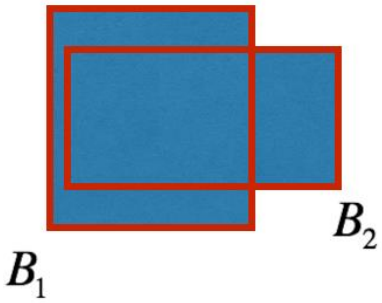
Redmon et al., 2015, You Only Look Once: Unified real-time object detection

Intersection over Union (IoU)

Intersection

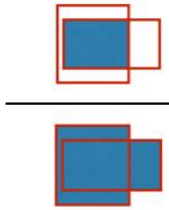


Union



Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} =$$



$=P_c$

"Correct" if IoU $\geq$ 0.5

More generally, IoU is a measure of the overlap between two bounding boxes.

<Deep Learning, Andrew Ng>

Non-max suppression

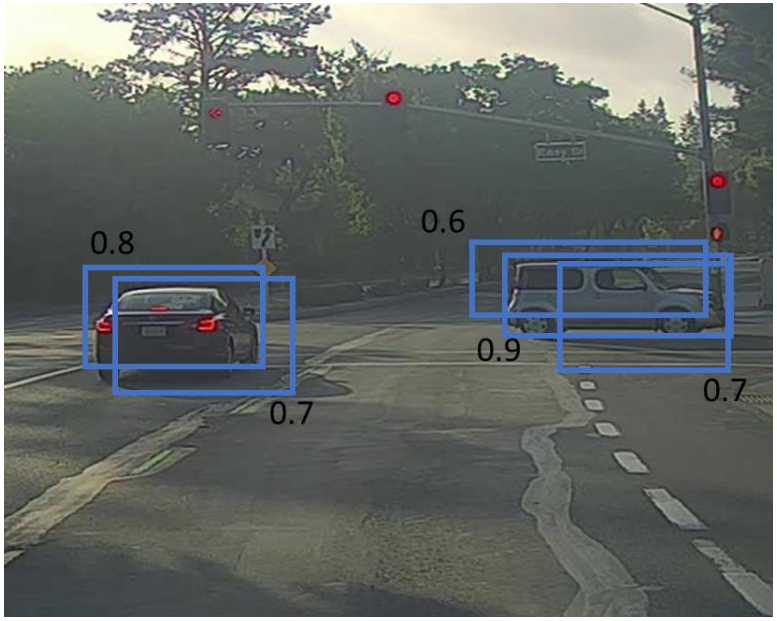
Each output prediction is:

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$

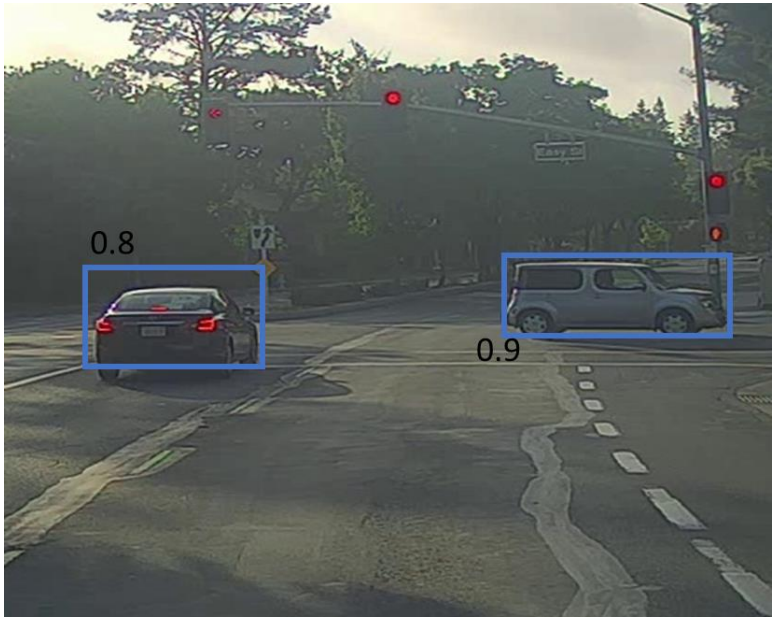
Discard all boxes with $p_c \leq 0.6$
Among remaining boxes,
Pick the box with the largest p_c.
Output that as a prediction.

Non-max suppression

Before non-max suppression



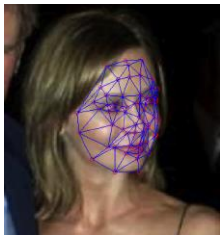
After non-max suppression



<Deep Learning, Andrew Ng>

Face recognition

Input image :



- Has a database of K persons
- Get an input image
- Output identity (name or id) if the image is any of the K persons (or “not recognized”)
- OneShotLearning

<Deep Learning, Andrew Ng>

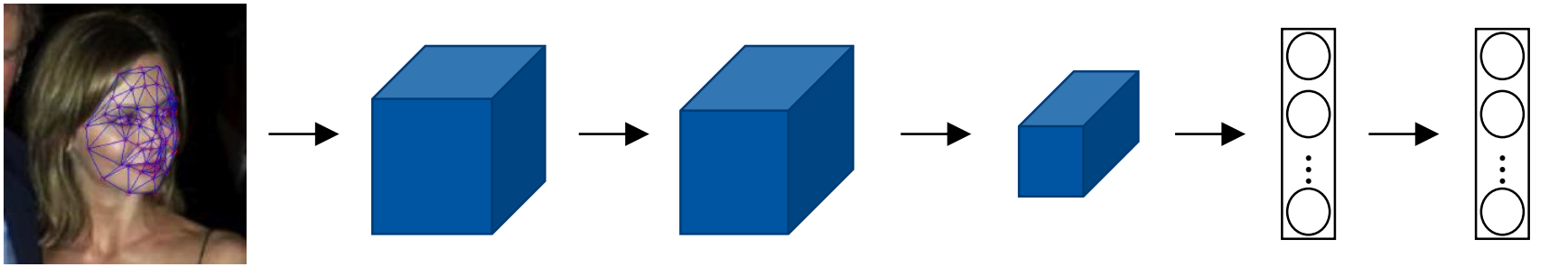
Distance between images

- d(img1,img2) = degree of difference between images

- Image similarity
  - d(img1,img2) $\leq \tau$ → same
  - d(img1,img2) $> \tau$ → different

<Deep Learning, Andrew Ng>

Deep face feature vector



If $x^{(i)}, x^{(j)}$ are same person, $d\left(f(x^{(i)}) - f(x^{(j)})\right)$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $d\left(f(x^{(i)}) - f(x^{(j)})\right)$ is large.

Taigman et. al., 2014. DeepFace closing the gap to human level performance

<Deep Learning, Andrew Ng>

Distance between two feature vectors

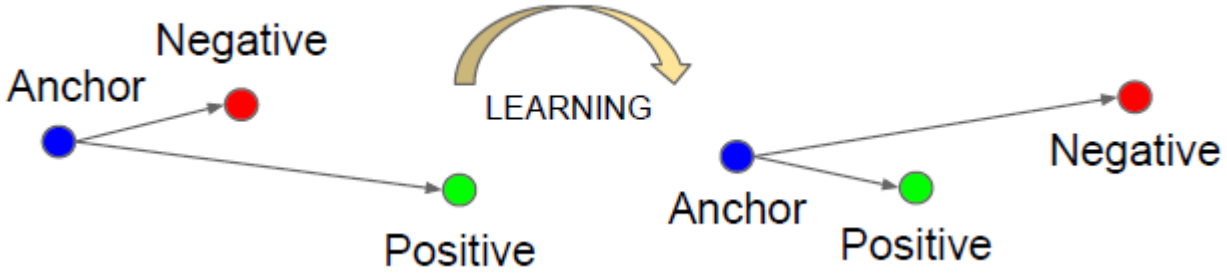
- χ^2 distance $= \frac{\left(f\left(x^{(i)}\right)-f\left(x^{(j)}\right)\right)^2}{f\left(x^{(i)}\right)+f\left(x^{(j)}\right)}$

- Siamese network: $d\left(x^{(i)} - x^{(j)}\right) = \left\|f\left(x^{(i)}\right) - f\left(x^{(j)}\right)\right\|_2^2$

Same as L2 norm

Taigman et. al., 2014. DeepFace closing the gap to human level performance

Triplet Loss



$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$$

Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering

Triplet Loss

Loss is minimized:

$$L = \sum_{i}^{N} \left[\| f(x_i^a) - f(x_i^p) \|_2^2 - \| f(x_i^a) - f(x_i^n) \|_2^2 + \alpha \right]$$

Schroff et al.,2015, FaceNet: A unified embedding for face recognition and clustering

Triplet Selection

For fast convergence,

Given x_i^a,

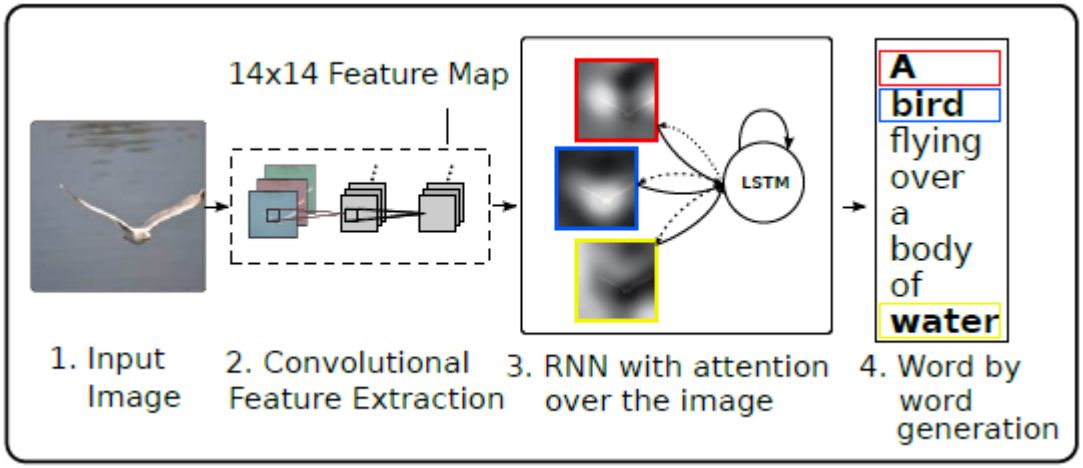
Select hard positive x_i^p, to make $\left\|f(x_i^a) - f(x_i^p)\right\|_2^2$ maximum.

Select hard negative x_i^n, to make $\|f(x_i^a) - f(x_i^n)\|_2^2$ minimum.

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2$$

Neural Image Caption Generation with Visual Attention

Visualized attentional maps



Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, Kelvin Xu et al., 2016

How to separate words from a sentence?

Tokenization

Tokenization is a process that splits an input sequence into tokens.

We can split token by space, punctuation, a set of rule.

Tokenization

In Düsseldorf I took my hat off. But I can't put it back on.

WhitespaceTokenizer

1. In | Düsseldorf | I | took | my | hat | off.
2. But | I | can't | put | it | back | on.

WordPunctTokenizer

1. In | Düsseldorf | I | took | my | hat | off | .
2. But | I | can | ' | t | put | it | back | on | .

TreebankWordTokenizer

1. In | Düsseldorf | I | took | my | hat | off | .
2. But | I | ca | n't | put | it | back | on | .

http://text-processing.com/demo/tokenize/

Token normalization

Same token for different forms of words

- Examples
  - wolf, wolves → wolf
  - talk, talks → talk
- Stemming
  - removes and replaces suffixes to get to the root form of a word, which is called as stem.
- Lemmatization
  - returns the base or dictionary form of a word, which is known as lemma.

Coursera: Natural Language Processing, National Research University Higher School of Economics

Transforming tokens into features

Bag of words (BOW)

For each token, we have a feature column, which is called text vectorization.

good movie
not a good movie
did not like

good	movie	not	a	did	like
1	1	0	0	0	0
1	1	1	1	0	0
0	0	1	0	1	1

Coursera: Natural Language Processing, National Research University Higher School of Economics

Preserve some ordering

N-grams: Token pairs, triplets, etc.

good movie
not a good movie
did not like

good movie	movie	did not	a	…
1	1	0	0	…
1	1	0	1	…
0	0	1	0	…

Coursera: Natural Language Processing, National Research University Higher School of Economics

Remove some n-grams

- High frequency n-grams
  - Articles, prepositions, etc. (example: and, a, the)
  - They are called stop-words. They do not help to discriminate texts.
- Low frequency n-grams
  - Typos, rare words

Coursera: Natural Language Processing, National Research University Higher School of Economics

Word Embedding

- Convert texts into numbers
- Map a word to a vector using a dictionary
- Applications
  - Sentiment analysis of reviews (amazon, movie review)
  - Document or news classification or clustering (google)

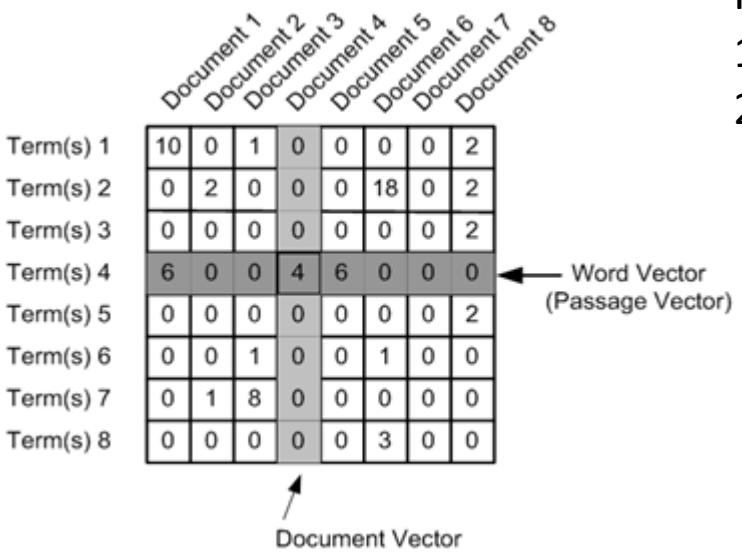
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Frequency based Embedding

- Count Vector
  - Frequency : Number of times a word has appeared in the document
  - Presence : Has the word appeared in the document?
- TF-IDF Vector
- Co-occurrence Vector

https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Word Matrix



How to make Term (word) features

1. All words in a dictionary
2. Unique words in corpus(all documents)

https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

TF-IDF

- N-gram with smaller frequency can be more discriminating because it can capture a specific issue in the text
- Term frequency (TF)
  - Frequency for term (or n-gram) t in document d

term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$

- Inverse document frequency (IDF)
  - N = D: total number of documents in corpus
  - $\{d \in D : t \in d\}$: Number of documents where the term t appears

$$\text{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Coursera: Natural Language Processing, National Research University Higher School of Economics

TF-IDF

Tfidf(t,d,D) = tf(t,d) x idf(t,D)

A high weight in TF-IDF means a high term frequency (in a given document) and a low document frequency of the term in a all collection of documents

good movie
not a good movie
did not like

good movie	movie	did not	…
0.17	**0.17**	0	…
0.17	**0.17**	0	…
0	0	**0.47**	…

Coursera: Natural Language Processing, National Research University Higher School of Economics

Term frequency & Inverse document frequency

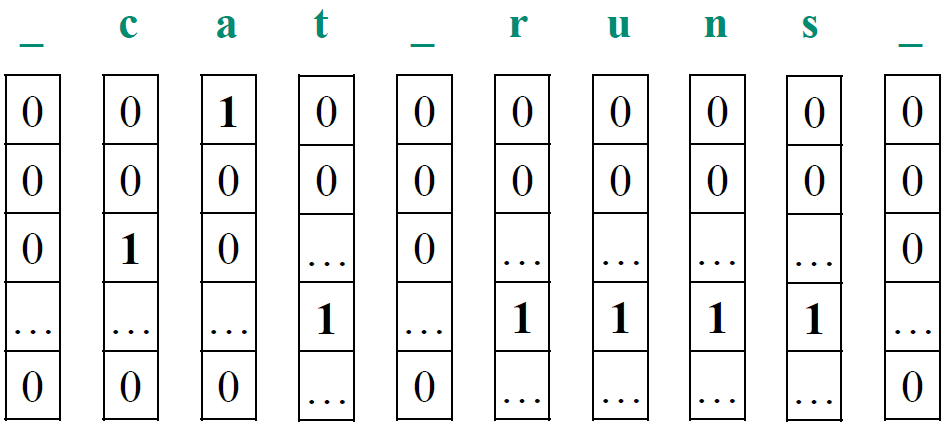
1-gram

					term frequency					
text	good	movie	not	a	did	not	like	I	it	one
good movie										
not a good movie										
did not like										
I like it										
good one										

					inverse document frequency					
text	good	movie	not	a	did	not	like	I	it	one
good movie										
not a good movie										
did not like										
I like it										
good one										

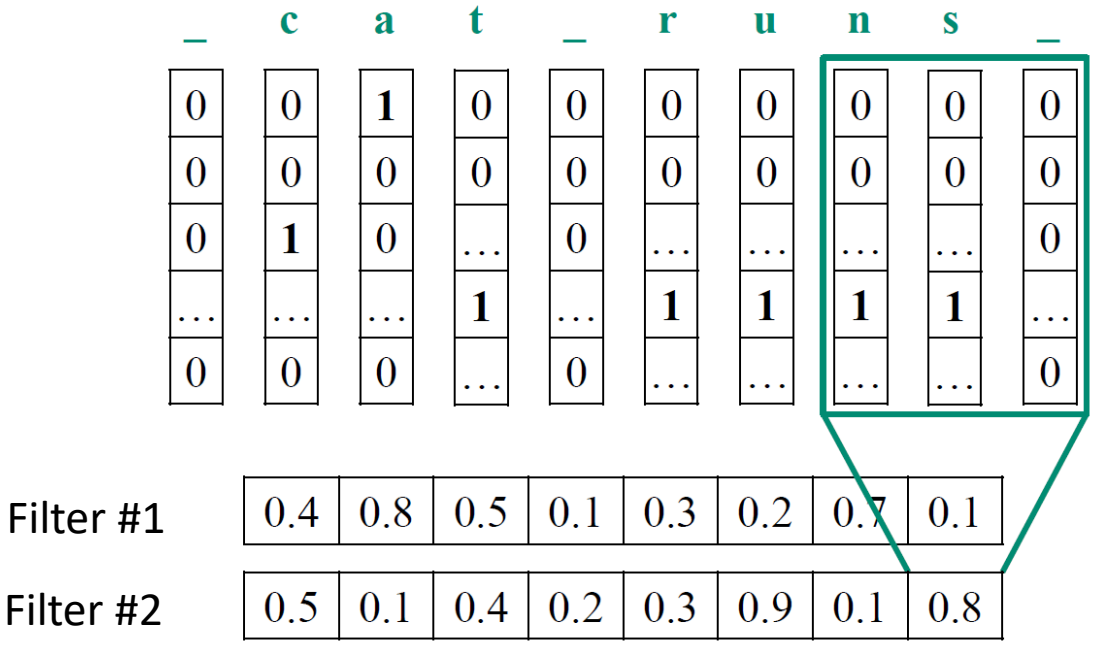
Text as a sequence of characters

One-hot encoding characters, length ~ 70



Coursera: Natural Language Processing, National Research University Higher School of Economics

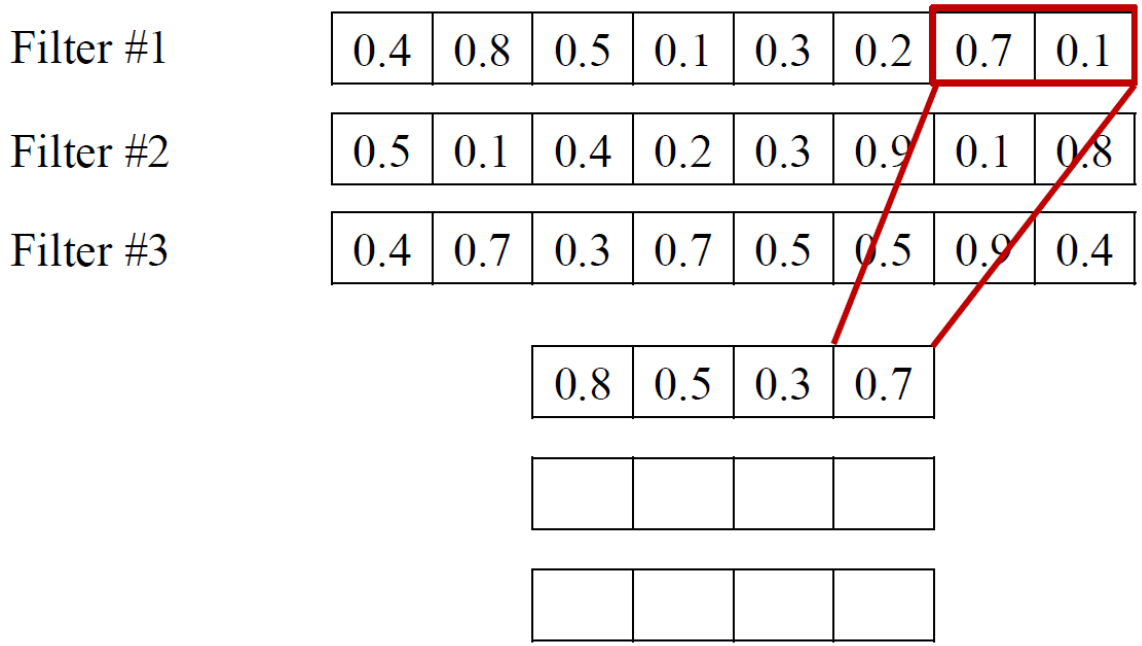
1D convolutions on characters



Coursera: Natural Language Processing, National Research University Higher School of Economics

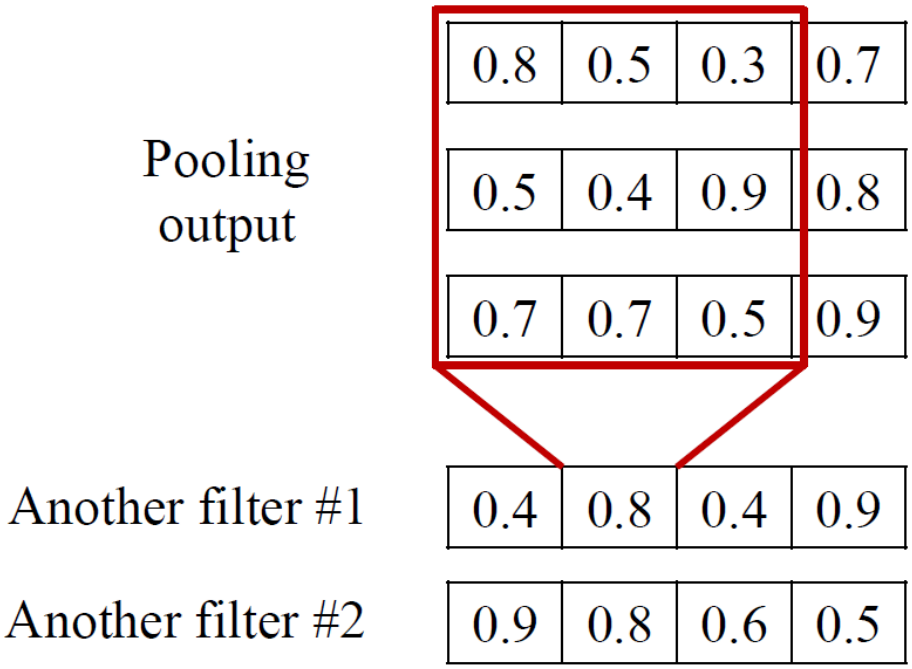
Max pooling

Provides position invariance for character n-gram



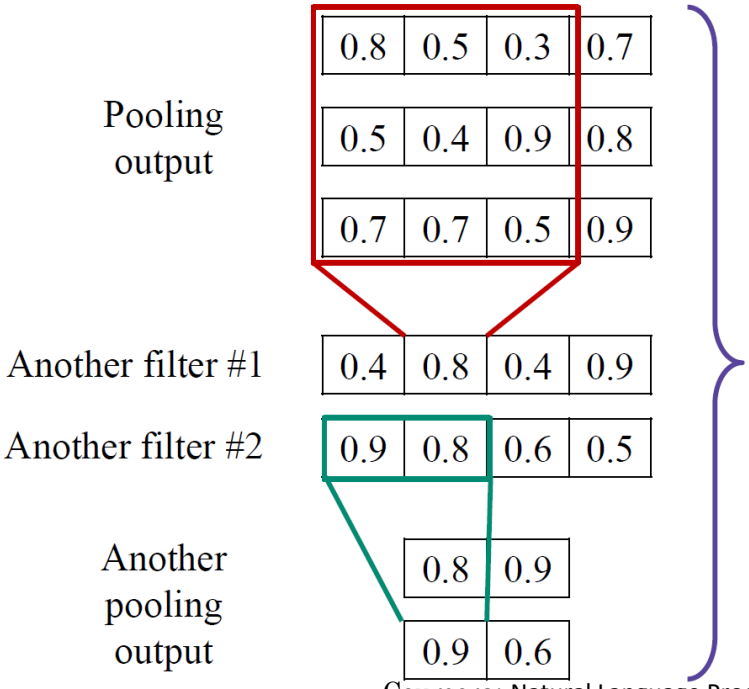
Coursera: Natural Language Processing, National Research University Higher School of Economics

Repeat 1D convolutions + pooling



Coursera: Natural Language Processing, National Research University Higher School of Economics

Repeat 1D convolutions + pooling



Coursera: Natural Language Processing, National Research University Higher School of Economics

Example of 1D convolution architecture for chacacters

- Characters of text : 1014
- Apply 1D convolution + max pooling 6 times
- Kernels widths : 7,7,3,3,3,3
- # of Filters at each step : 1024
- Output: 1024 x 34 matrix of features
- Apply multiclass classification for sentiment analysis

Coursera: Natural Language Processing, National Research University Higher School of Economics

Problem of one-hot representation

- It treats each word as a thing unto itself, and it doesn't allow an algorithm to easily generalize the cross words.
- Example
  - Sally Johnson is an orange farmer.
  - Robert Lin is an apple farmer.
  - Robert Lin is a durian cultivator.
- product of any two word vector is zero.

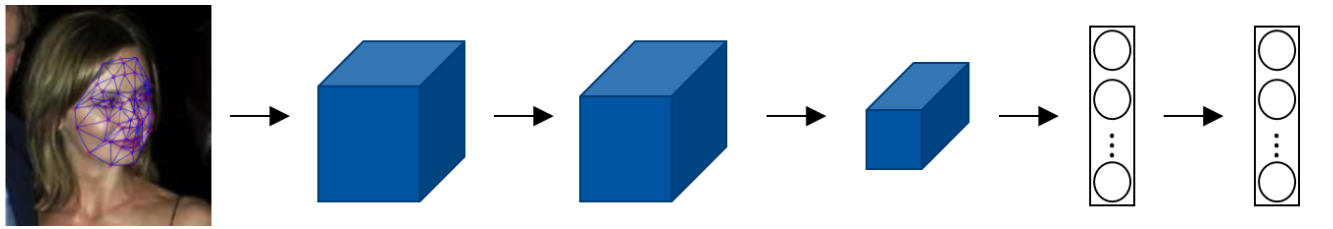
Coursera: Deep learning Specialization, Andrew Ng

Featured representation: word embedding

	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
Size						
Cost						
alive						
verb						

Word embedding vector

Deep face feature vector



If $x^{(i)}, x^{(j)}$ are same person, $d\left(f(x^{(i)}) - f(x^{(j)})\right)$ is small.

If $x^{(i)}, x^{(j)}$ are different persons, $d\left(f(x^{(i)}) - f(x^{(j)})\right)$ is large.

Taigman et. al., 2014. DeepFace closing the gap to human level performance

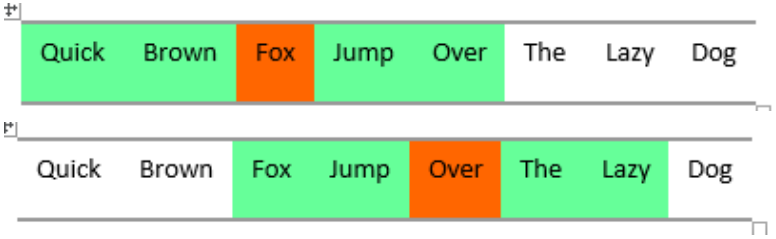
Transfer learning and word embeddings

- Learn word embeddings from large text corpus. (1-100B words)
  (Or download pre-trained embedding online.)
- Transfer embedding to new task with smaller training set. (say, 100k words)
- Continue to fine-tune the word embeddings with new data.

Coursera: Deep learning Specialization, Andrew Ng

Co-occurrence Matrix

- Hypothesis: Similar words tend to occur together and will have similar context.
- Example
  - Apple is a fruit. Mango is a fruit.
- Co-occurrence
  - For a given corpus, the co-occurrence of a pair of words is the number of times they have appeared together in a context window
- Context window
  - Context window is specified by a number and the direction



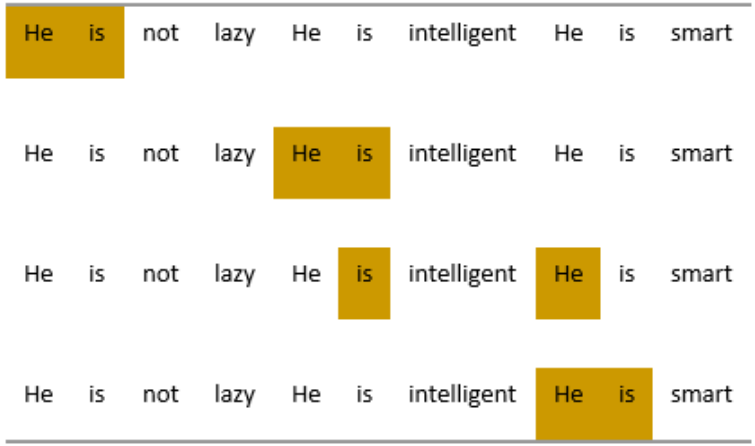
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Co-occurrence Matrix

Corpus = He is not lazy. He is intelligent. He is smart.

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

Context windows = 2



https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Co-occurrence Matrix

Matrix size

- V x V
  - Not practical
- V x N
  - N is a subset of V and can be obtained by removing irrelevant words like stop words
- V x k
  - k is k principal components out of V using PCA

PCA to decompose Co-occurrence matrix

$X = U \cdot S \cdot V^T$ (Singular value decomposition)

U and S represent word vector

V presents word context

U is principal component.

$$\underset{m \times n}{\hat{X}} \approx \underset{m \times r}{U} \; \underset{r \times r}{S} \; \underset{r \times n}{V^T}$$

m x m m x n n x n

https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Frequency based Embedding

- Count Vector
  - Frequency : Number of times a word has appeared in the document
  - Presence : Count if the word appeared in the document
- TF-IDF Vector
- Co-occurrence Vector

https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Analogies

	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

Man → Woman vs. King → ? (Queen)

$e_{Man} - e_{Woman} \approx e_{King} - e_{Queen}$

Linguistic regularities in continuous space word representations, Mikolov et. al., 2013,

Coursera: Deep learning Specialization, Andrew Ng

Analogies using word vectors

Man → Woman vs. King → ? (Queen)

$e_{Man} - e_{Woman} \approx e_{King} - e_{Queen}$

$e_{Man} - e_{Woman} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $e_{King} - e_{Queen} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$e_{Man} - e_{Woman} \approx e_{King} - e_{w}$

Find word w : arg max (e_w , $e_{King} - e_{Man} + e_{Woman}$)

Man:Woman as Boy:Girl
Ottawa:Canada as Nairobi:Kenya
Big:Bigger as Tall:Taller
Yen:Japan as Ruble:Russia

Coursera: Deep learning Specialization, Andrew Ng

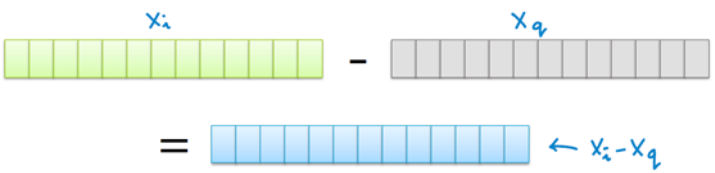
Distance between two word vectors

(non-scaled) Euclidean distance

Defined in terms of inner product

$$\text{distance}(\mathbf{x}_i, \mathbf{x}_q) = \sqrt{(\mathbf{x}_i - \mathbf{x}_q)^T(\mathbf{x}_i - \mathbf{x}_q)}$$

$$= \sqrt{(\mathbf{x}_i[1] - \mathbf{x}_q[1])^2 + \ldots + (\mathbf{x}_i[d] - \mathbf{x}_q[d])^2}$$



Cosine similarity – normalize

$$\text{Similarity} = \frac{\sum_{j=1}^{d} \mathbf{x}_i[j]\, \mathbf{x}_q[j]}{\sqrt{\sum_{j=1}^{d} (\mathbf{x}_i[j])^2} \sqrt{\sum_{j=1}^{d} (\mathbf{x}_q[j])^2}}$$

$$\mathbf{x}_i^T \mathbf{x}_q = \cos(\theta)$$

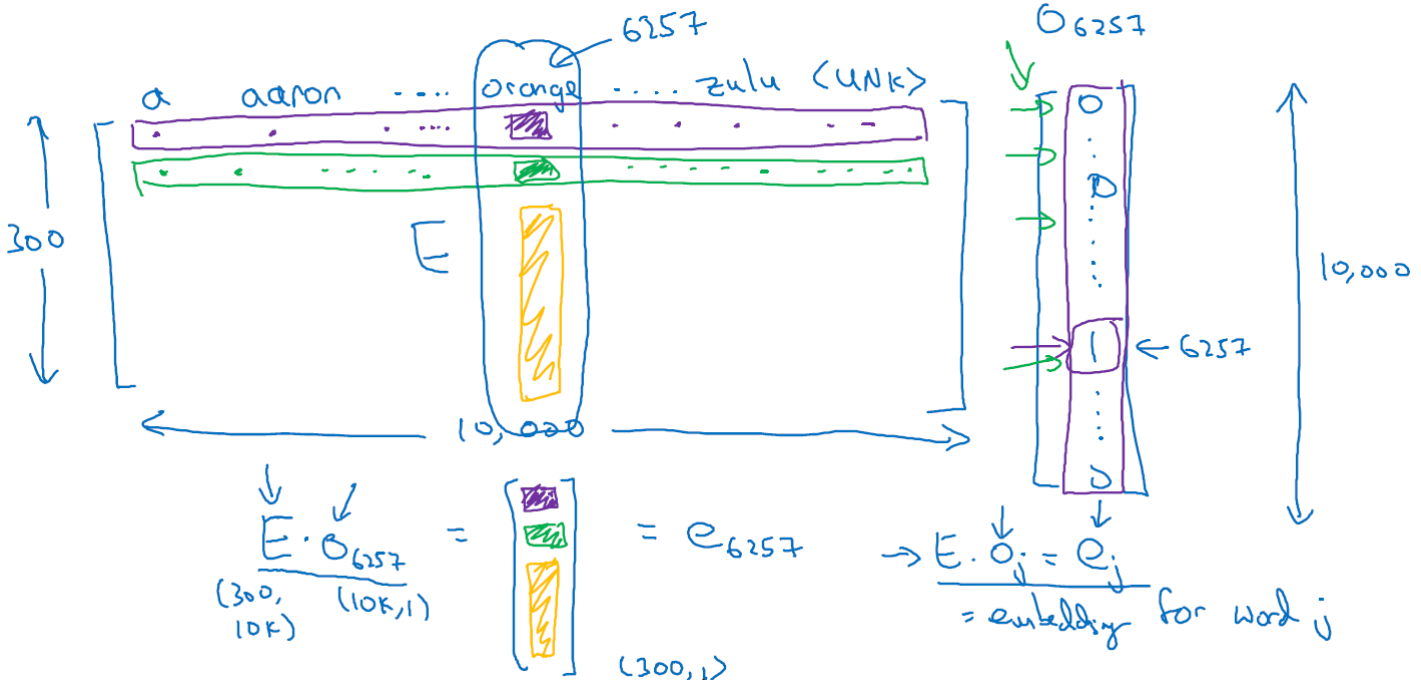
$$\frac{\mathbf{x}_i^T \mathbf{x}_q}{\|\mathbf{x}_i\| \|\mathbf{x}_q\|} = \left(\frac{\mathbf{x}_i}{\|\mathbf{x}_i\|}\right)^T \left(\frac{\mathbf{x}_q}{\|\mathbf{x}_q\|}\right)$$

first normalize

$a^T b = \|a\| \|b\| \cos(\theta)$

Coursera: Machine Learning, Emily Fox & Carlos Guestrin

Pull out word vector from Embedding matrix



Coursera: Deep learning Specialization, Andrew Ng

Language model using word vector

I	want	a	glass	of	orange	_____.
4343	9665	1	3852	6163	6257	

$e_{4343} = E\, o_{4343}$

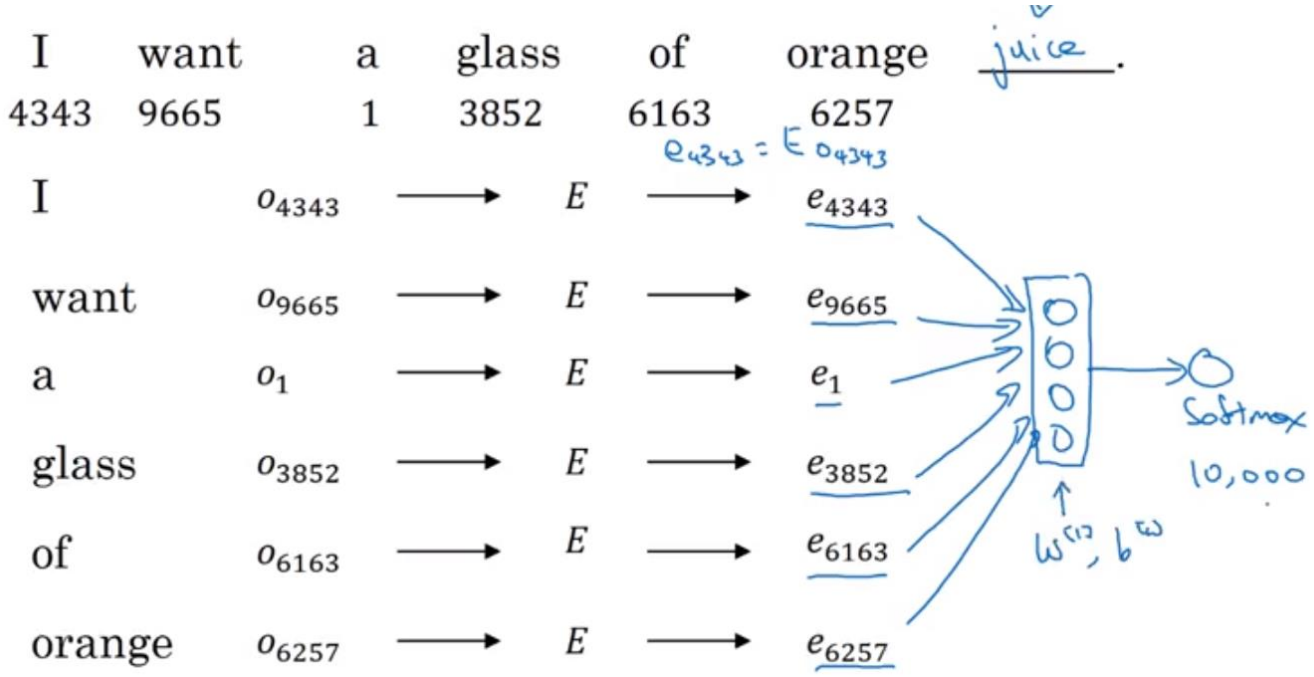
I	o_{4343}	$\longrightarrow$ E $\longrightarrow$	e_{4343}
want	o_{9665}	$\longrightarrow$ E $\longrightarrow$	e_{9665}
a	o_1	$\longrightarrow$ E $\longrightarrow$	e_1
glass	o_{3852}	$\longrightarrow$ E $\longrightarrow$	e_{3852}
of	o_{6163}	$\longrightarrow$ E $\longrightarrow$	e_{6163}
orange	o_{6257}	$\longrightarrow$ E $\longrightarrow$	e_{6257}

Coursera: Deep learning Specialization, Andrew Ng

Language model using word vector

A neural probabilistic language model , Bengio et. al., 2003

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_1^{t-1})$$

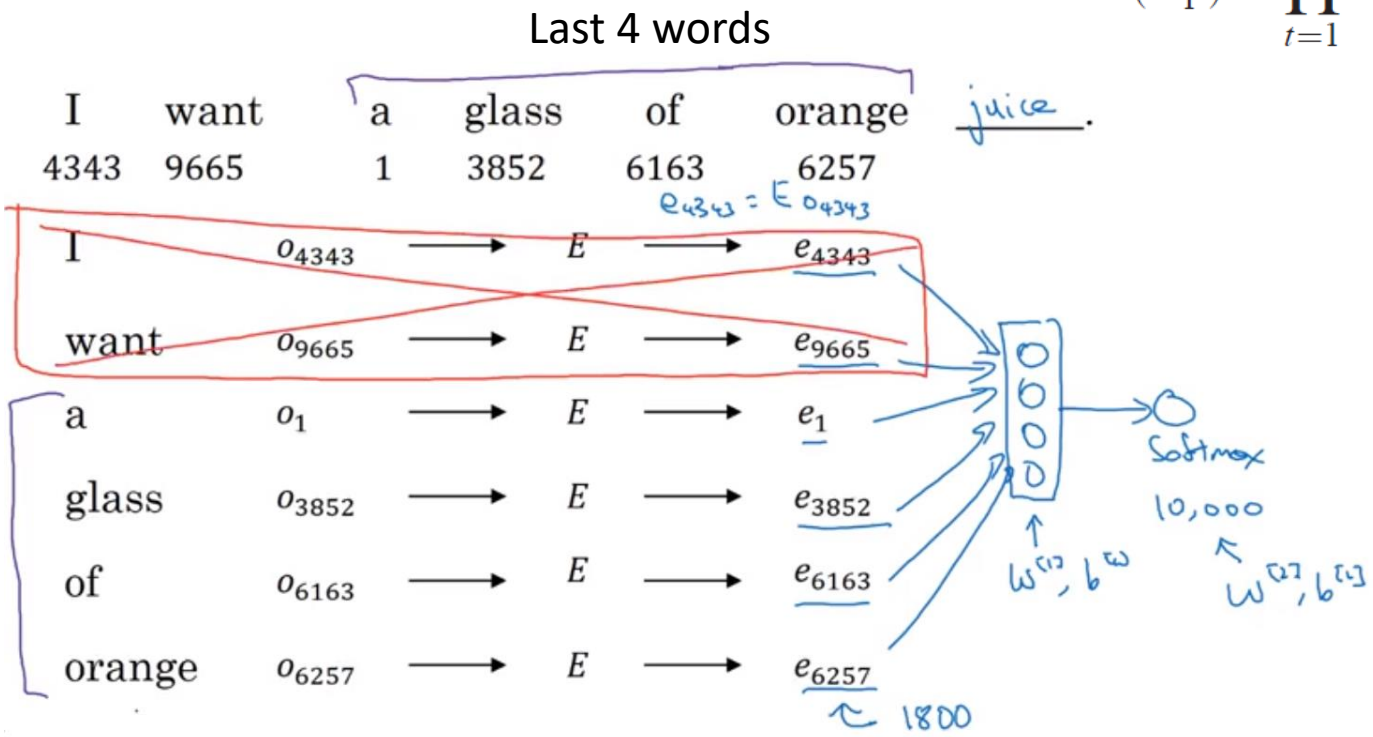


Coursera: Deep learning Specialization, Andrew Ng

Language model using word vector

A neural probabilistic language model , Bengio et. al., 2003

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_1^{t-1})$$



Coursera: Deep learning Specialization, Andrew Ng

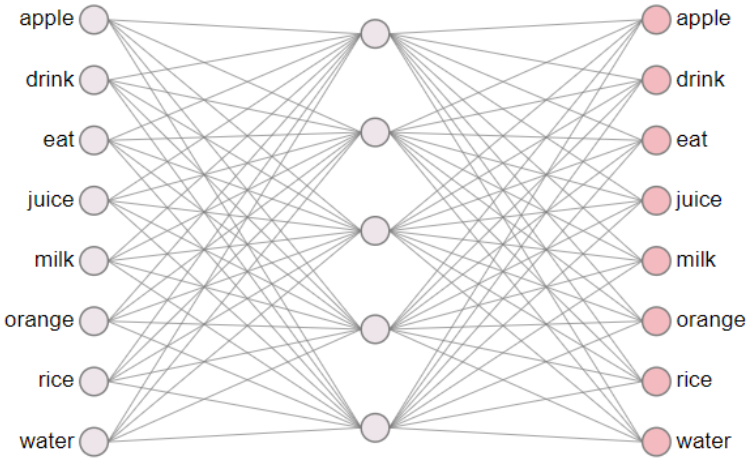
Word2vec : Prediction based on Embedding

- CBOW (Continuous Bag of words)
  - Predicts the current word based on the context.
- Skip-gram
  - Predicts surrounding words given the current word.

Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov et al.,2013

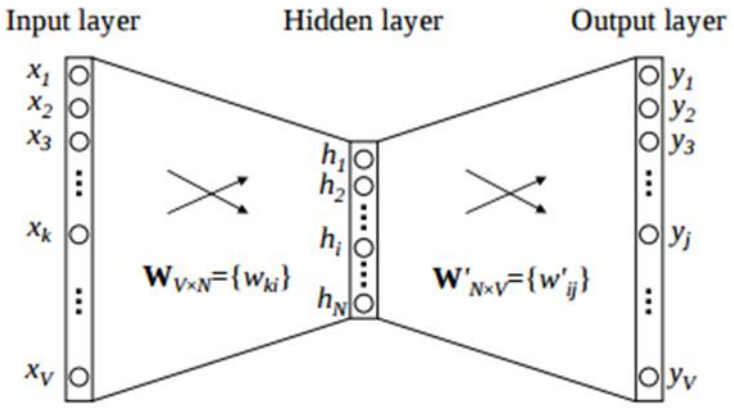
CBOW(Continuous Bag of words)

Predicts the current word based on the context



https://ronxin.github.io/wevi/

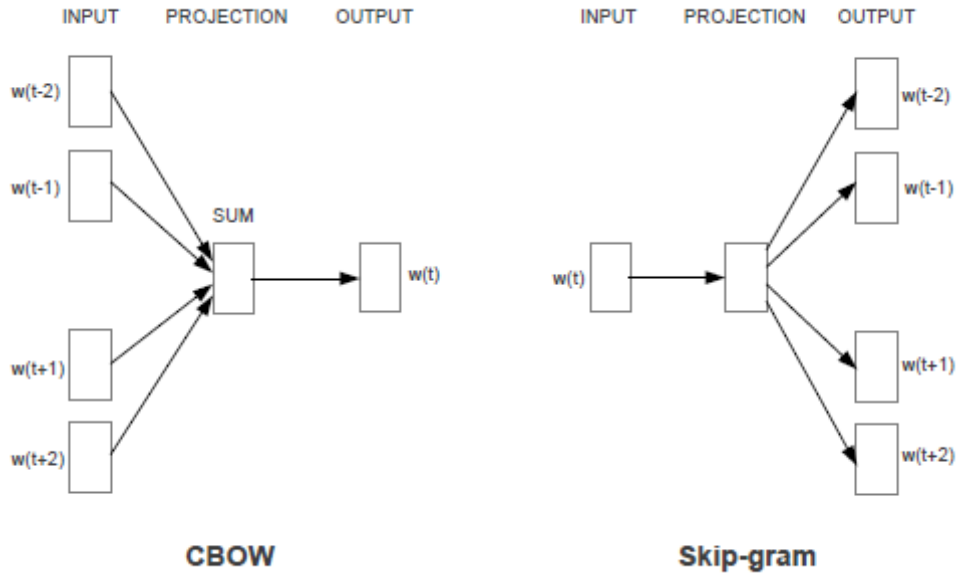
word vector :
the weight between the hidden layer and the output layer



Linear activation (no activation function between any layers)

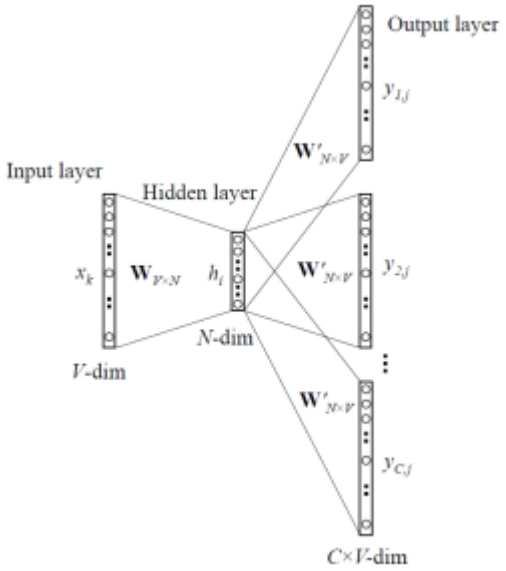
https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Word2vec : Prediction based on Embedding



Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov et al.,2013

Skip-gram



Predicts surrounding words given the current word

https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

Context/target pair

I want a glass of orange juice to go along with my cereal.

target

Context:
Last 4 words : a glass of orange
4 words on left & right : a glass of orange, to go along with
Last 1 word : orange
Nearby 1 word → skip-gram

Coursera: Deep learning Specialization, Andrew Ng

Skip-gram

I want a glass of orange juice to go along with my cereal.

Content	Target	
Orange	Juice	Next 1 word
Orange	Glass	Left/right 2 windows
Orange	to	

Coursera: Deep learning Specialization, Andrew Ng

Skip-gram

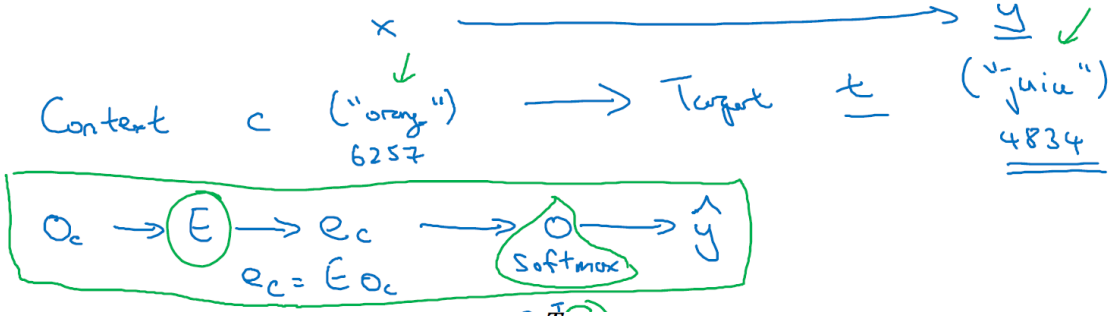
How to sample content word?

- Randomly select → there is high chance to pick up most of stop-words (a, the, and, etc)
- Select from less-frequent words

Coursera: Deep learning Specialization, Andrew Ng

Softmax-classification

Vocab size = 10,000k



Softmax: $$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

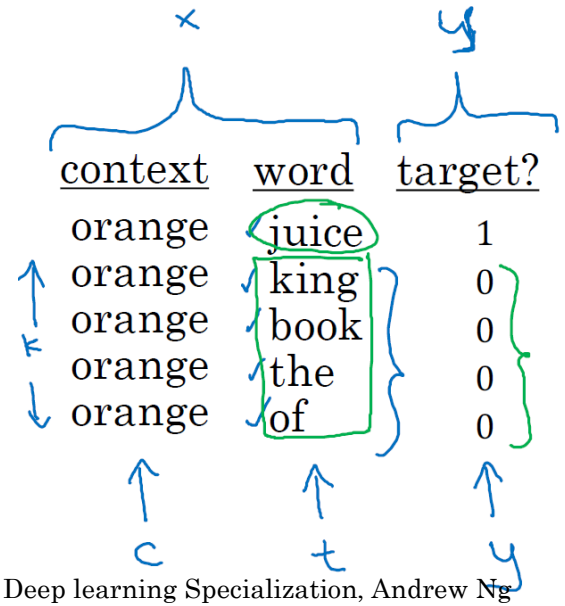
Θ_t : parameter associated with target t

Cost: $$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

Coursera: Deep learning Specialization, Andrew Ng

Negative sampling

I want a glass of orange juice to go along with my cereal.



Binary classification

$$P(y=1 \mid c,t) = \sigma(\theta_t^T e_c)$$

Sigmoid function

Randomly pick up from dictionary

or

$$P(\omega_i) = \frac{f(\omega_i)^{3/4}}{\sum_{j=1}^{10,000} f(\omega_j)^{3/4}}$$

Coursera: Deep learning Specialization, Andrew Ng

Word2vec

vocabulary_size=7 and *embedding_size=3*

anarchism	0.5	0.1	−0.1
originated	−0.5	0.3	0.9
as	0.3	−0.5	−0.3
a	0.7	0.2	−0.3
term	0.8	0.1	−0.1
of	0.4	−0.6	−0.1
abuse	0.7	0.1	−0.4

	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/

GloVe: Global Vectors for Word Representation

GloVe: Global Vectors for Word Representation, Jeffrey Pennington et al.,2014

I want a glass of orange juice to go along with my cereal.

X_{ij} = # times j appears in content of i

content target target content

Shallow Window-based methods (making predictions within local context windows)

GloVe: Model

GloVe: Global Vectors for Word Representation, Jeffrey Pennington et al.,2014

Loss function $$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Minimize $$\sum_{i=1}^{10{,}000} \sum_{j=1}^{10{,}000} f(X_{ij})\left(\theta_i^T e_j + b_i + b_j' - \log X_{ij}\right)^2$$

1. f (0) = 0. If f is viewed as a continuous function, it should vanish as x ! 0 fast enough that the limx!0 f (x) log2 x is finite.
2. f (x) should be non-decreasing so that rare co-occurrences are not overweighted.
3. f (x) should be relatively small for large values of x, so that frequent co-occurrences are not overweighted.

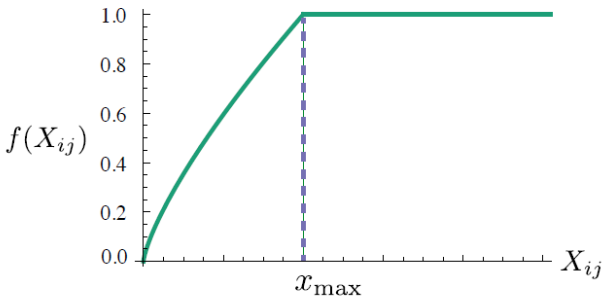


Figure 1: Weighting function f with $\alpha = 3/4$.

GloVe: Global Vectors for Word Representation

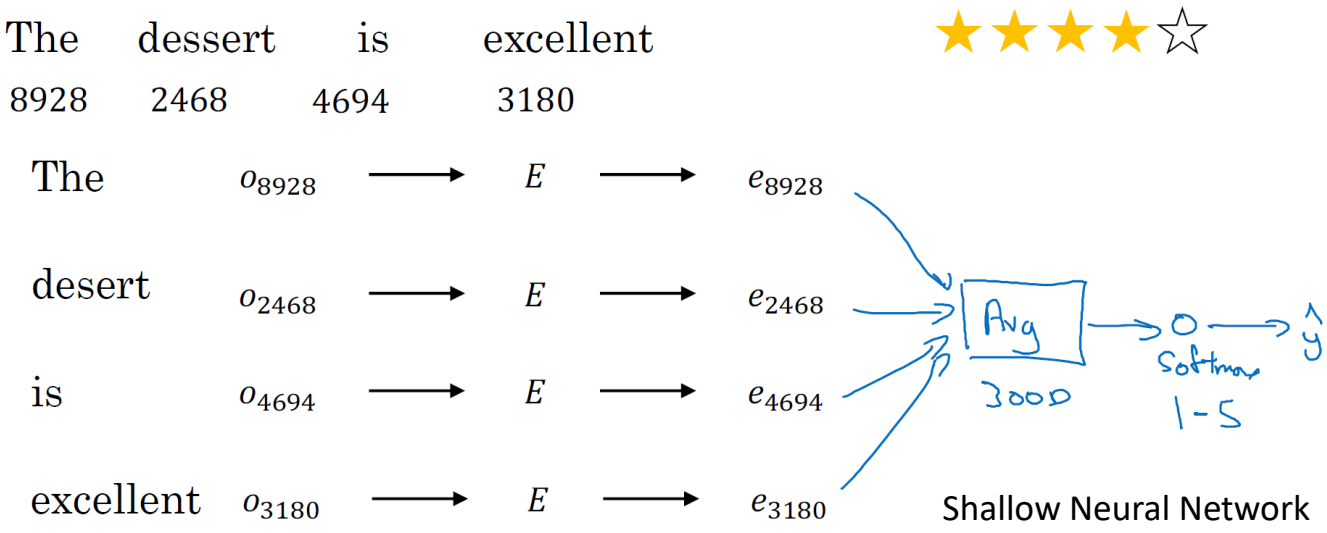
GloVe: Global Vectors for Word Representation, Jeffrey Pennington et al.,2014

- Unsupervised word representation
- Combined
  - Count-based method
  - Prediction-based method
- Outperforms
  - Word analogies
  - Word similarity
  - Named entity recognition

Sentiment classification problem

x	y
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

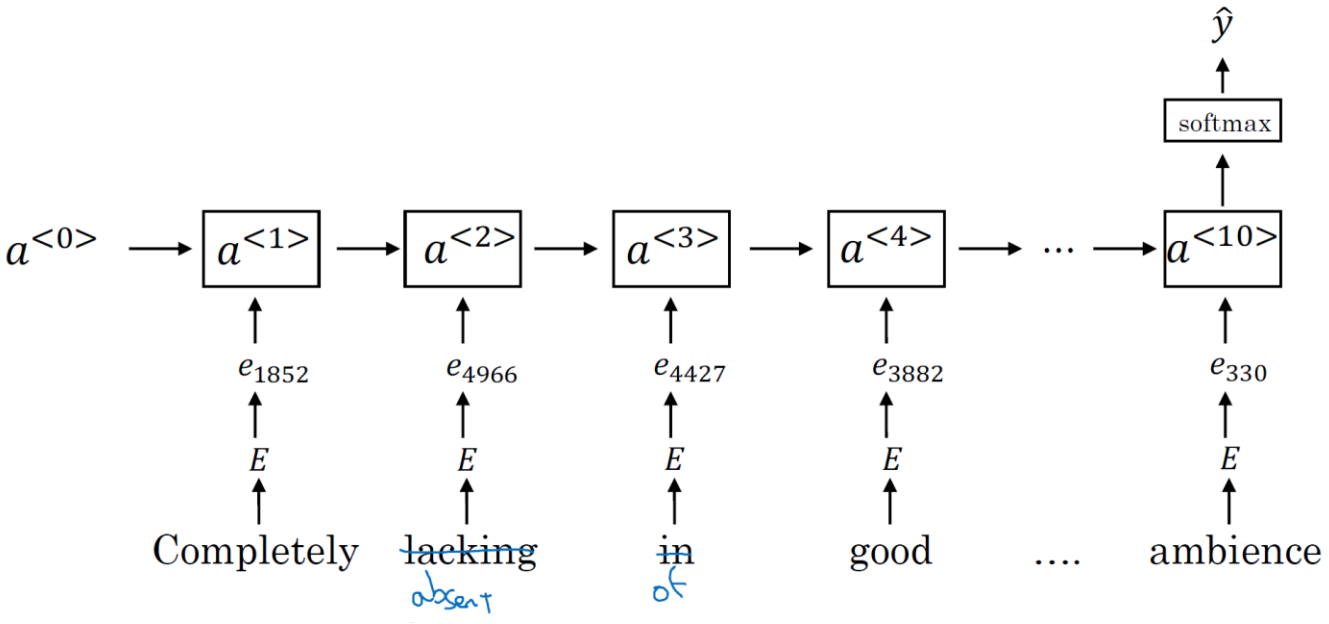
Simple sentiment classification model



Problem in Simple sentiment classification model

Completely lacking in good taste, good service, and good ambience.

RNN for sentiment classification



Character-level language model

Vocabulary = [a, b, c, …, z, [ ], ., ‘, ; , … , A, …, Z]

- Advantage:
  - Do not worry about unknown token.
- Disadvantage:
  - Much longer sequence
  - Is not as good as word level language models at capturing long range dependencies between how
  - the earlier parts of the sentence also affect the later part of the sentence.
  - more computationally expensive to train

Coursera: Deep learning Specialization, Andrew Ng

Example of Name entity recognition

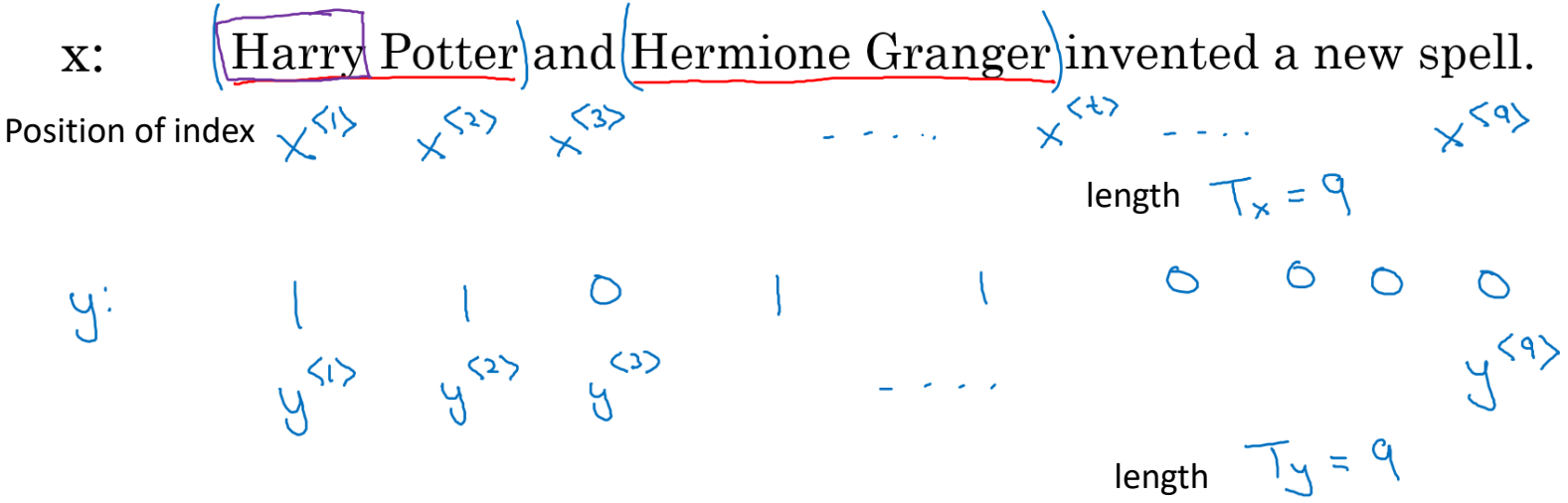
Name entity recognition can be used to find people's names, companies names, times, locations, countries names, currency names, and so on.

x: Harry Potter and Hermione Granger invented a new spell.

y: 1 1 0 1 1 0 0 0 0

Coursera: Deep learning Specialization, Andrew Ng

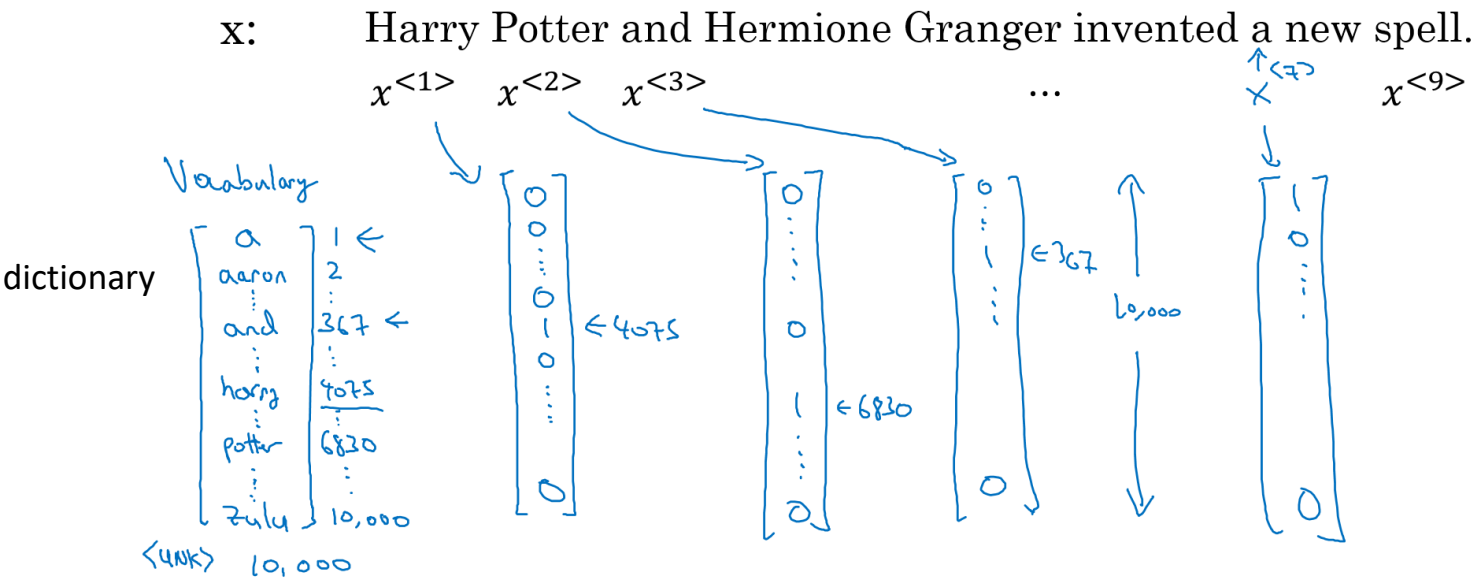
Name entity recognition



Coursera: Deep learning Specialization, Andrew Ng

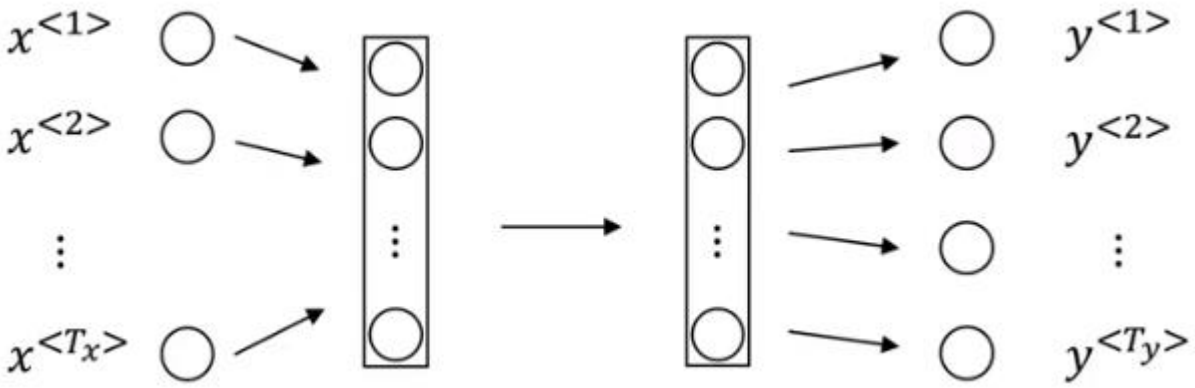
One-hot vector to represent words

Map: $x^{<t>} \rightarrow y^{<t>}$



Coursera: Deep learning Specialization, Andrew Ng

Why not a Neural Network?



Problems:

- Arbitrary sequence length : Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.
- Large number of parameters

Coursera: Deep learning Specialization, Andrew Ng

Language model

Cats sleep average 15 hours per a day. <EOS>

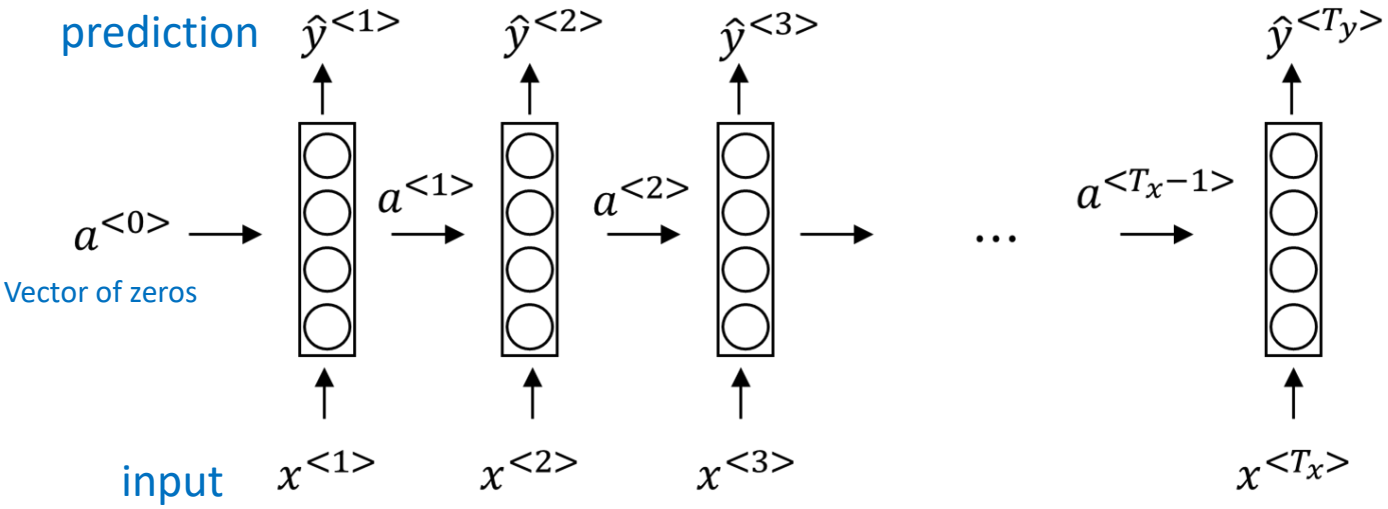
$$P(text) = P(x_0, \ldots, x_n) =$$
$$= P(x_0)P(x_1|x_0)P(x_2|x_0, x_1)\ldots P(x_n|\ldots)$$

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t|w_1^{t-1})$$

A neural probabilistic language model , Bengio et. al., 2003

Coursera: Natural Language Processing, National Research University Higher School of Economics

Recurrent Neural Networks

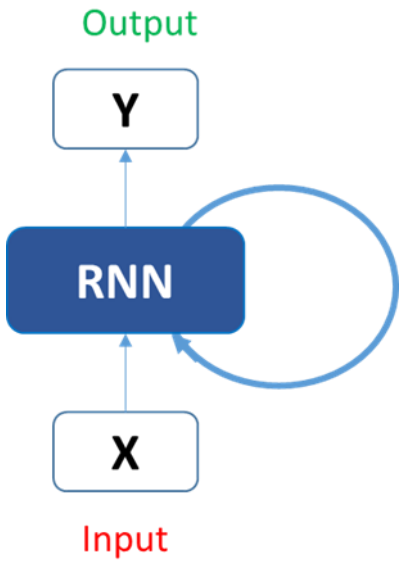


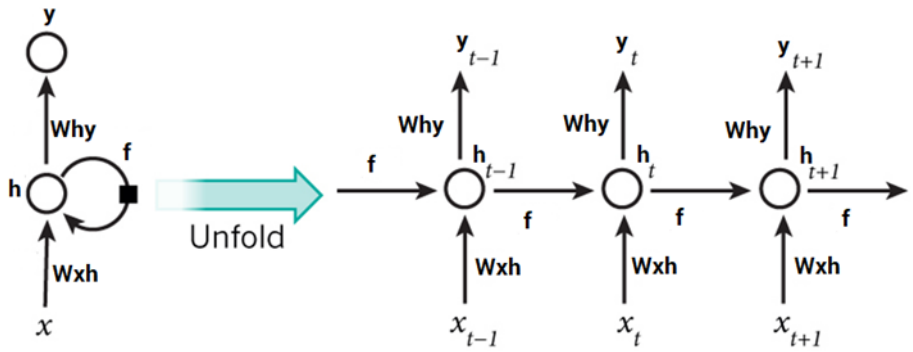
Fixed number of inputs at each time step
→ Solve the arbitrary sequence length

$x^{<t>}$: current input (word embedding)
$a^{<t-1>}$: activation value in previous hidden state
$a^{<t>}$: activation value in current hidden state
$y^{<t>}$: prediction (probability distribution for the next word)

Coursera: Deep learning Specialization, Andrew Ng

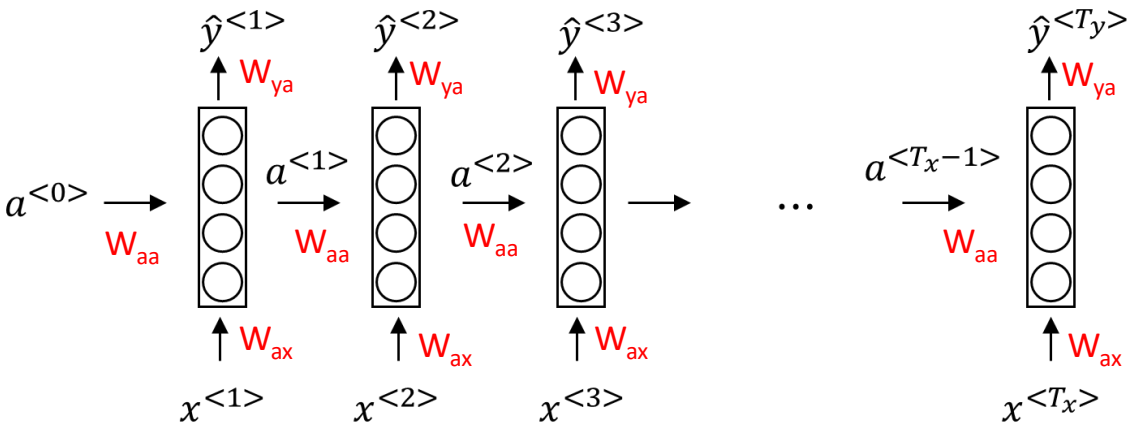
Recurrent Neural Networks





Fundamentals of Deep Learning – Introduction to Recurrent Neural Networks
DISHASHREE GUPTA,2017

Forward Propagation



$x^{<t>}$: current input
$a^{<t-1>}$: activation in previous hidden state
$a^{<t>}$: activation in current hidden state
$y^{<t>}$: prediction

g_1, g_2 : activation function
g_1 : tanh, ReLU
g_2 : sigmoid, softmax

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$a^{<1>} = g_1(W_{aa}a^{<0>} + W_{ax}x^{<1>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

$$\hat{y}^{<1>} = g_2(W_{ya}a^{<1>} + b_y)$$

All the parameters are shared across the different time steps → solve large number of parameters

Coursera: Deep learning Specialization, Andrew Ng

Simplified RNN notation

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$

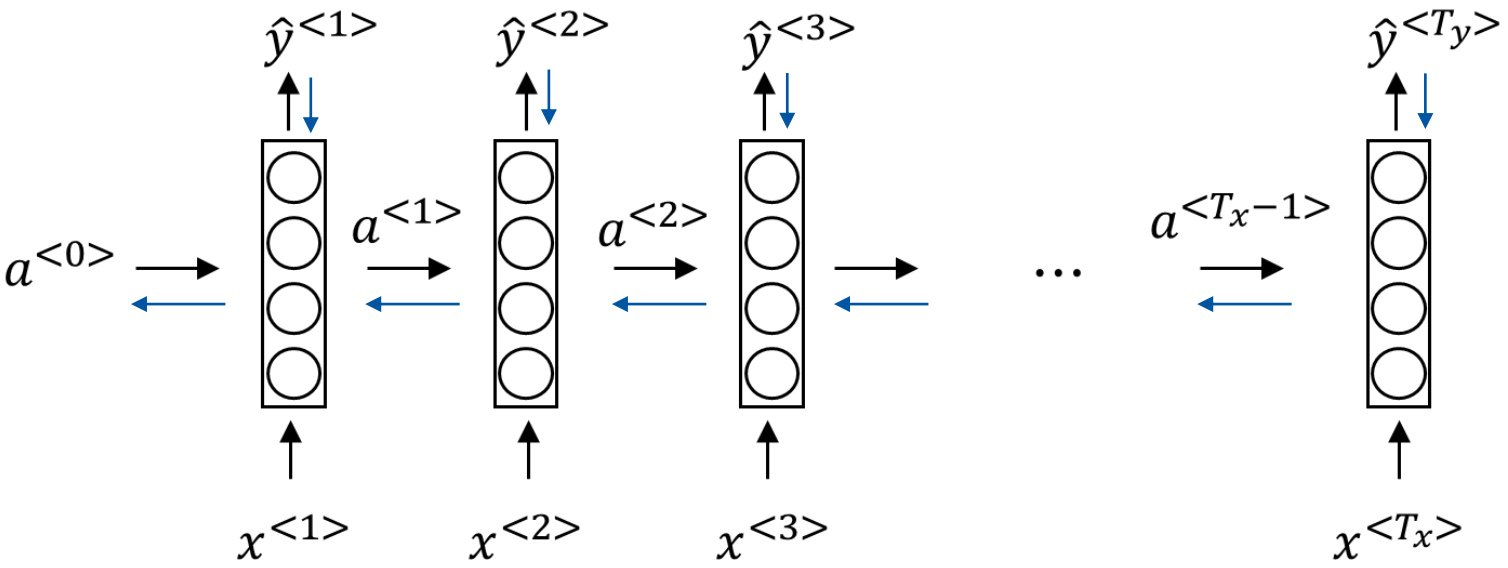
$$[W_{aa}; W_{ax}] = W_a$$

$$[W_{aa}; W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$$

$x^{<t>}$: current input
$a^{<t-1>}$: activation in previous hidden state
$a^{<t>}$: activation in current hidden state
$y^{<t>}$: prediction

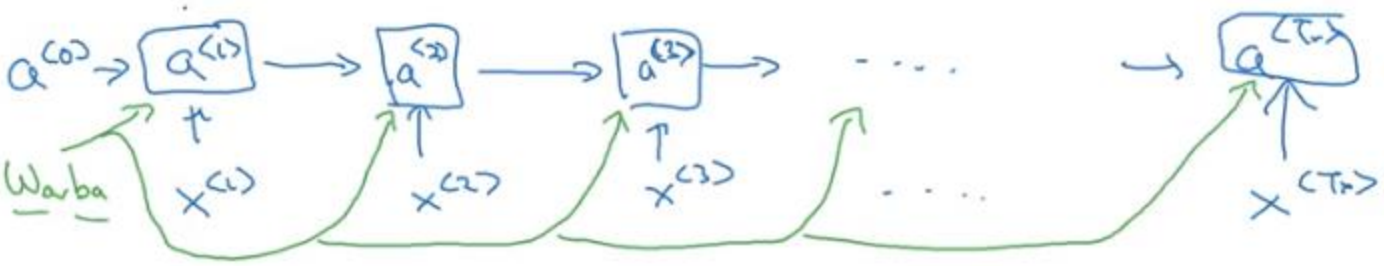
Coursera: Deep learning Specialization, Andrew Ng

Forward propagation and backpropagation



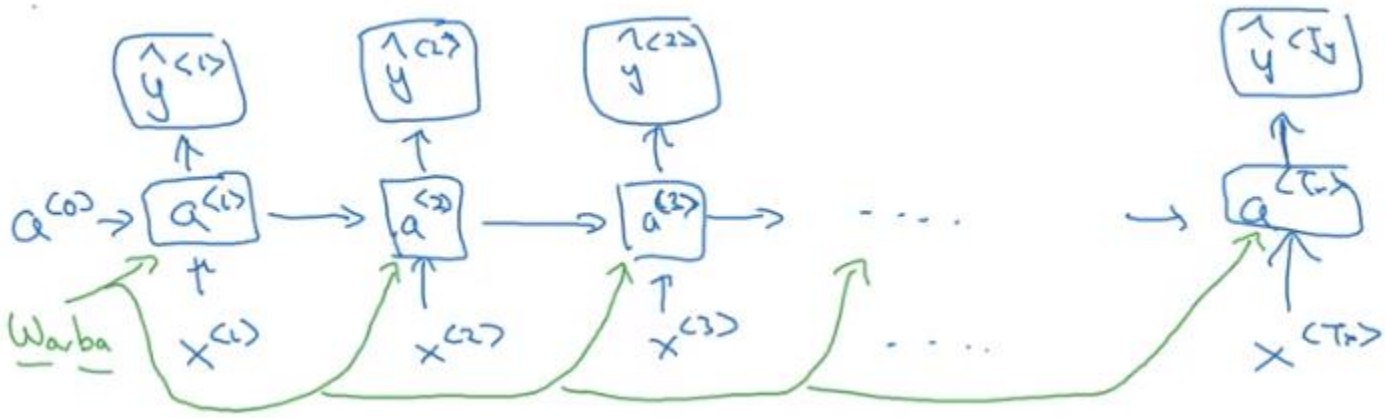
Coursera: Deep learning Specialization, Andrew Ng

Backpropagation through time



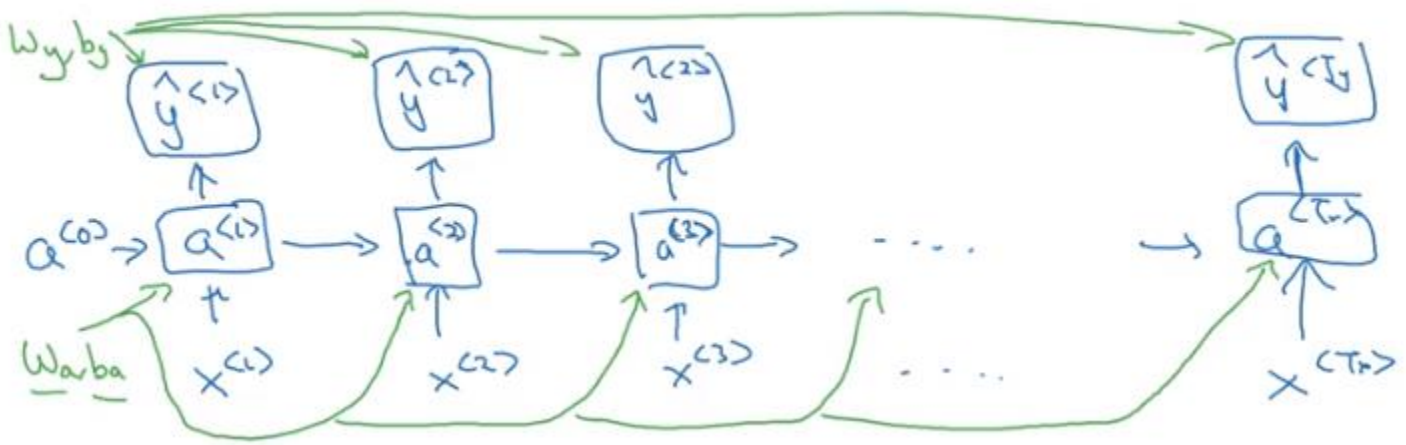
Coursera: Deep learning Specialization, Andrew Ng

Backpropagation through time



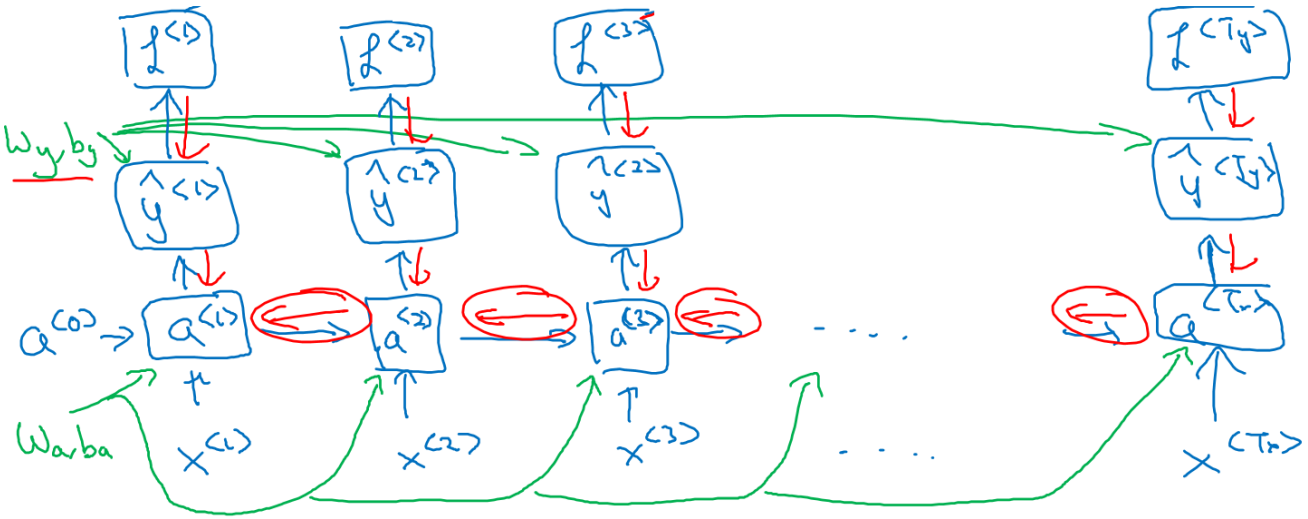
Coursera: Deep learning Specialization, Andrew Ng

Backpropagation through time



Coursera: Deep learning Specialization, Andrew Ng

Backpropagation through time(BPTT)

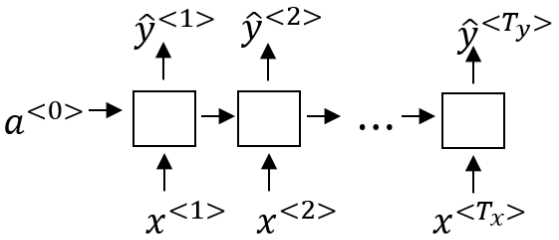


Cross-entropy loss $L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}\log(\hat{y}^{<t>}) - (1 - y^{<t>})\log(1 - \hat{y}^{<t>})$

Total loss $L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$

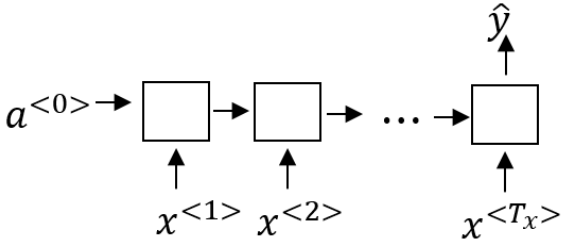
Coursera: Deep learning Specialization, Andrew Ng

RNN types



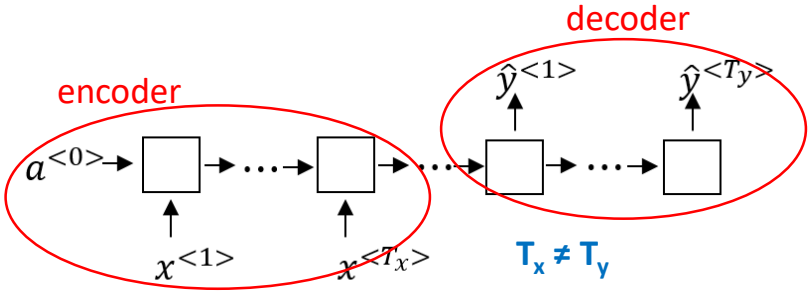
Many to many $T_x = T_y$

Name entity recognition



Many to one

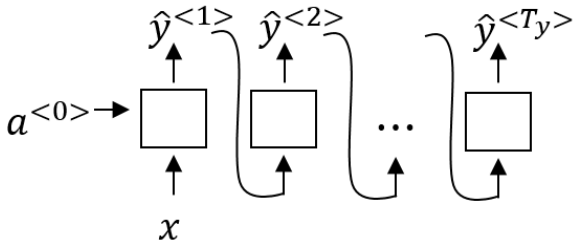
Sentiment classification



$T_x \neq T_y$

Many to many

Machine translation

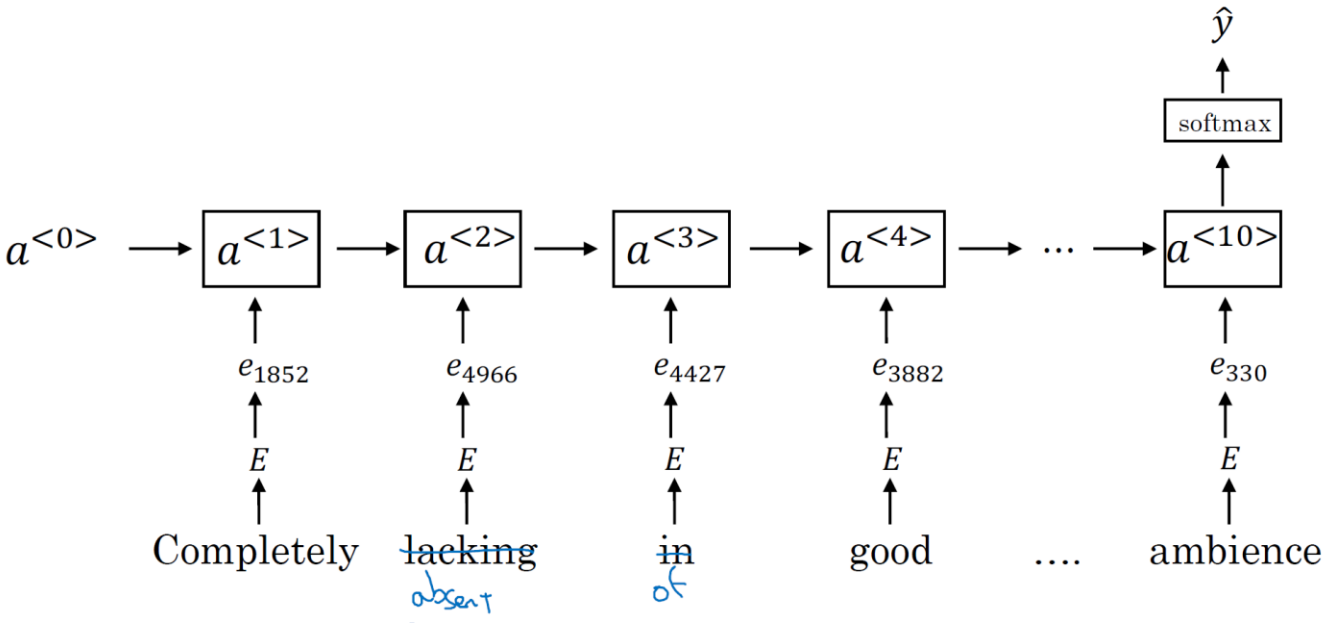


One to many

Music generation

Coursera: Deep learning Specialization, Andrew Ng

RNN for sentiment classification



Coursera: Deep learning Specialization, Andrew Ng

Long range of dependence

The cat, which already ate …, was full.

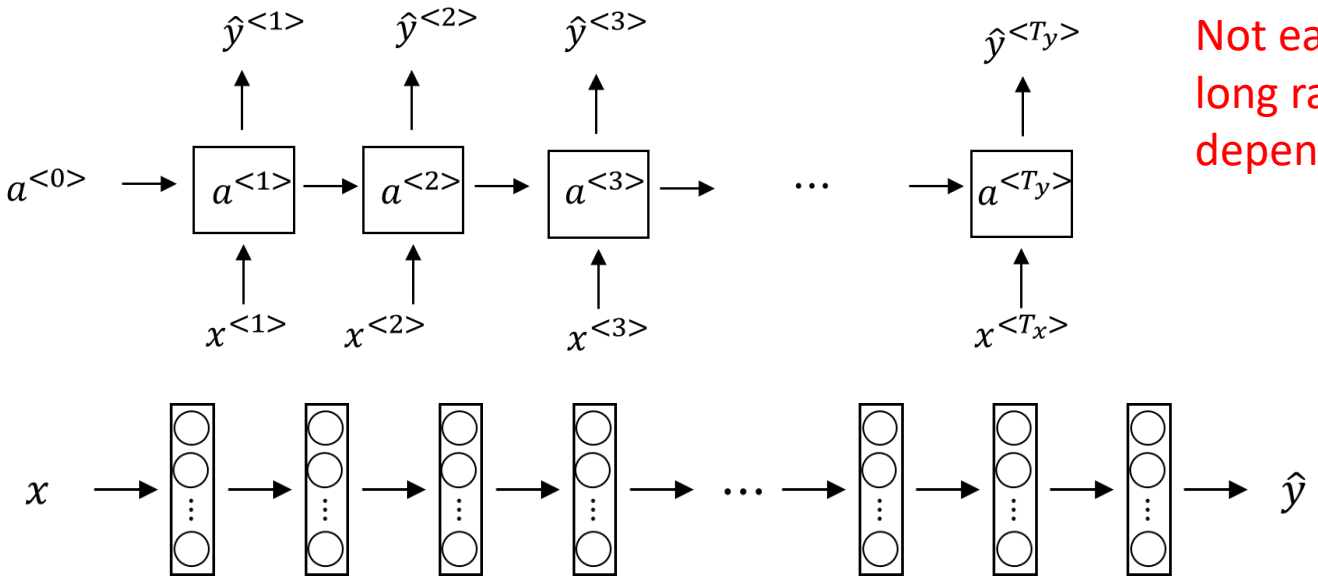
The cats, which already ate …, were full.

long range dependencies is how the earlier parts of the sentence affects the later part of the sentence.

Vanishing gradients with RNNs

The cat, which already ate …, was full.

The cats, which already ate …, were full.



Not easy to capture long range of dependence with RNN.

Coursera: Deep learning Specialization, Andrew Ng

Gradients with RNNs

$$\left|\frac{\partial h_i}{\partial h_{i-1}}\right| < 1$$

- Gradients is vanishing after 3-4 time steps.
  - Not easy to capture long range of dependence between the earlier parts of the sentence and the later part of the sentence.

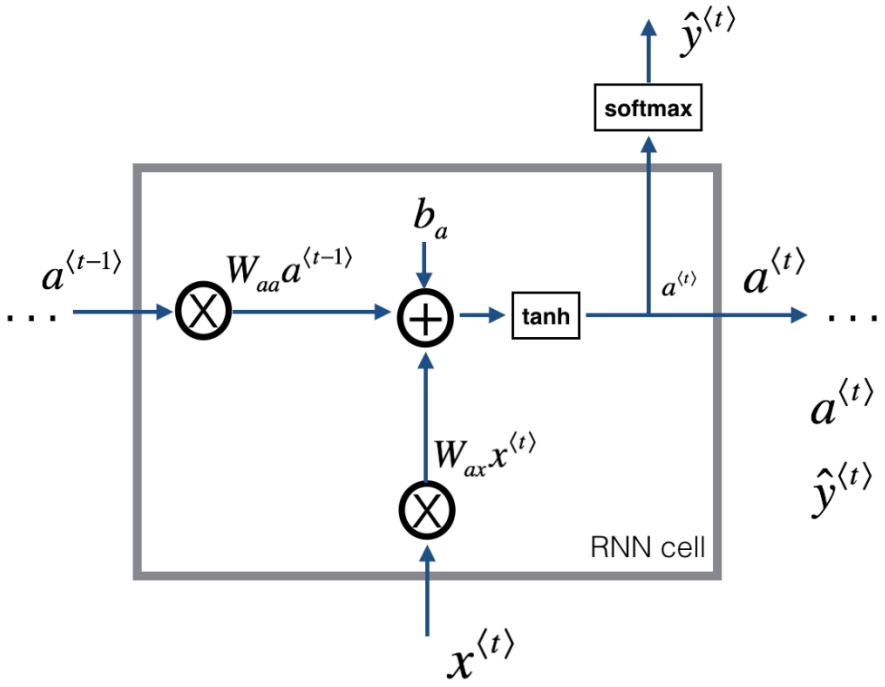
$$\left|\frac{\partial h_i}{\partial h_{i-1}}\right| > 1$$

- Exploding gradients:
  - Learning process becomes unstable
  - Gradient becomes NaN
  - Gradient clipping

Coursera: Natural Language Processing, National Research University Higher School of Economics

Coursera: Deep learning Specialization, Andrew Ng

RNN unit



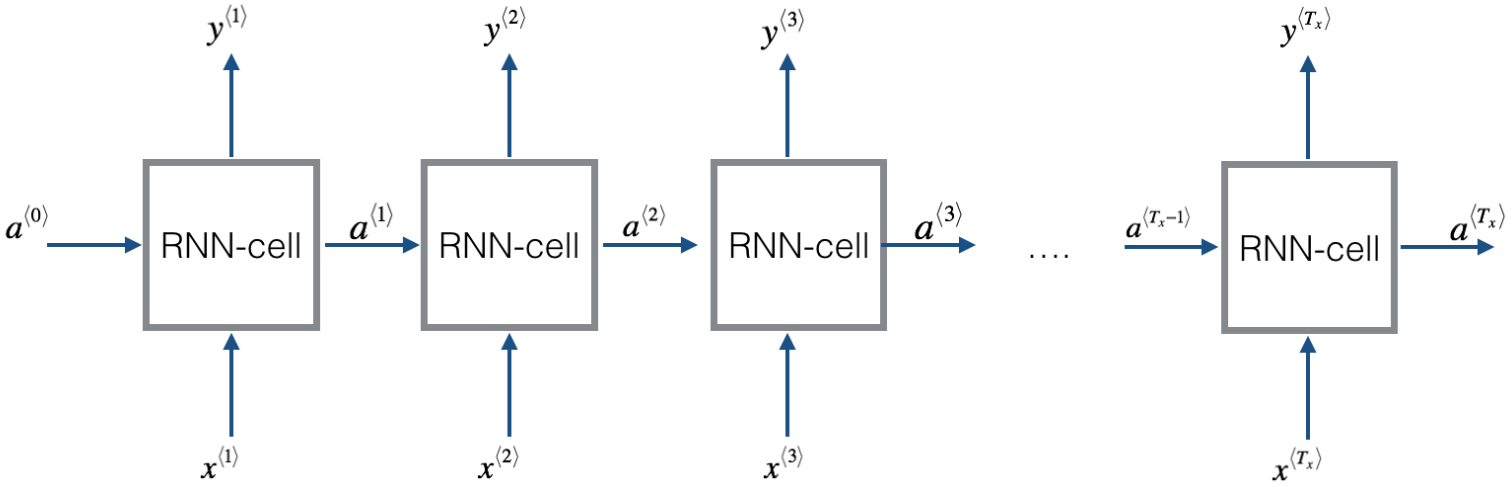
$x^{<t>}$: current input
$a^{<t-1>}$: activation in previous hidden state
$a^{<t>}$: activation in current hidden state
$\hat{y}^{<t>}$: prediction

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

Coursera: Deep learning Specialization, Andrew Ng

RNN forward pass



Coursera: Deep learning Specialization, Andrew Ng

Gated Recurrent Unit (simplified GRU)

c : memory cell
$c^{<t>} = a^{<t>}$
$\tilde{c}^{<t>}$: candidate of activation
Γ_u: Update gate

The cat, which already ate …, was full.

$c^{<t>}$ =1 Γ_u=1 | Γ_u=0 | Γ_u=0 | $c^{<t>}$ =1 Γ_u=1

$$\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

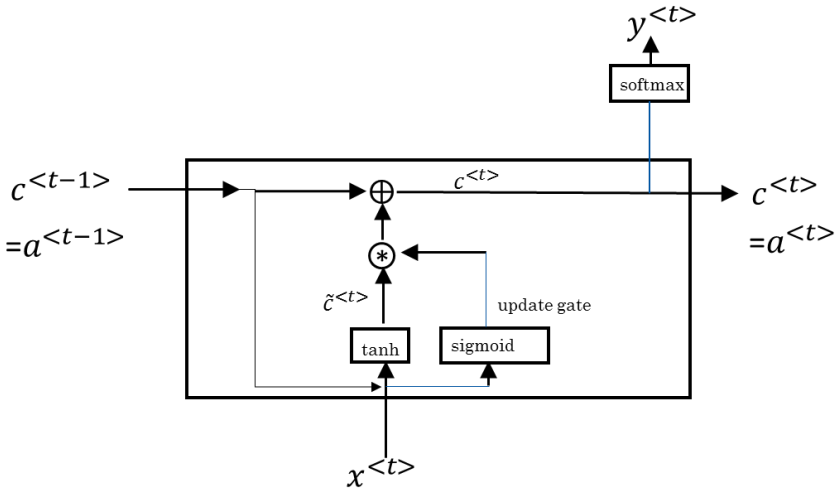
*:Element-wise multiplication

Coursera: Deep learning Specialization, Andrew Ng

On the properties of neural machine translation: Encoder-decoder approaches, Cho et al., 2014.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Chung et al., 2014.

Gated Recurrent Unit (GRU)



$$\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$\Gamma_u \approx 0$, $c^{<t>} = c^{<t-1>}$

$c^{<t>}$ maintains many previous time-steps and helps vanishing gradient problem and allows long range dependencies.

Coursera: Deep learning Specialization, Andrew Ng

On the properties of neural machine translation: Encoder-decoder approaches, Cho et al., 2014.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Chung et al., 2014.

Full GRU

$\tilde{h}$ $\tilde{c}^{<t>} = \tanh(W_c[\Gamma_u * c^{<t-1>}, x^{<t>}] + b_c)$

Γ_r: relevance gate

u $\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$

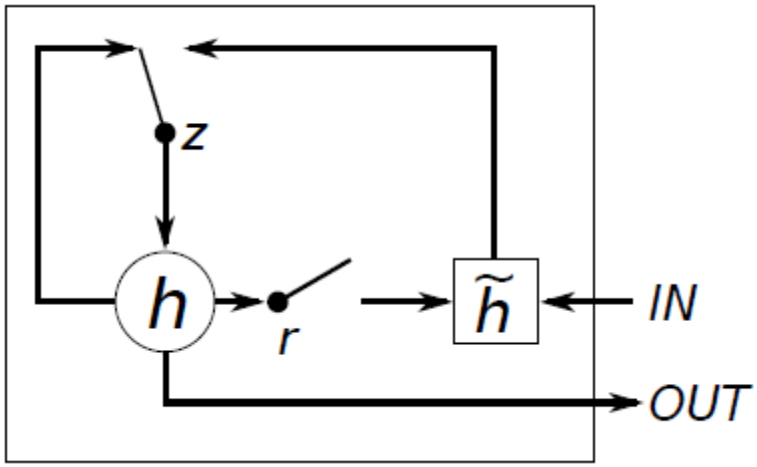
r $\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$

h $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$

Relevance gate is how relevant $c^{<t-1>}$ is to compute $\tilde{c}^{<t>}$.

Coursera: Deep learning Specialization, Andrew Ng

Gated Recurrent Unit (GRU)



r : reset gate
z : update gates
h : activation in hidden state
$\hat{h}$: candidate activation in hidden state.

On the properties of neural machine translation: Encoder-decoder approaches, Cho et al., 2014.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Chung et al., 2014.

GRU vs. LSTM(long short term memory)

GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

update $\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$

forget $\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$

output $\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$

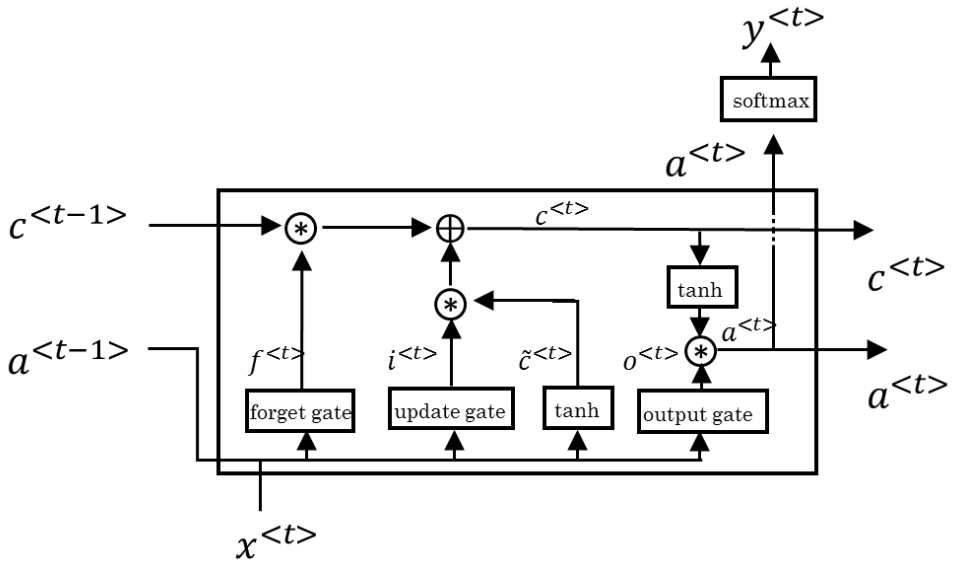
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

Coursera: Deep learning Specialization, Andrew Ng

Long short-term memory, Hochreiter & Schmidhuber 1997.

LSTM (long short term memory) cell



$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

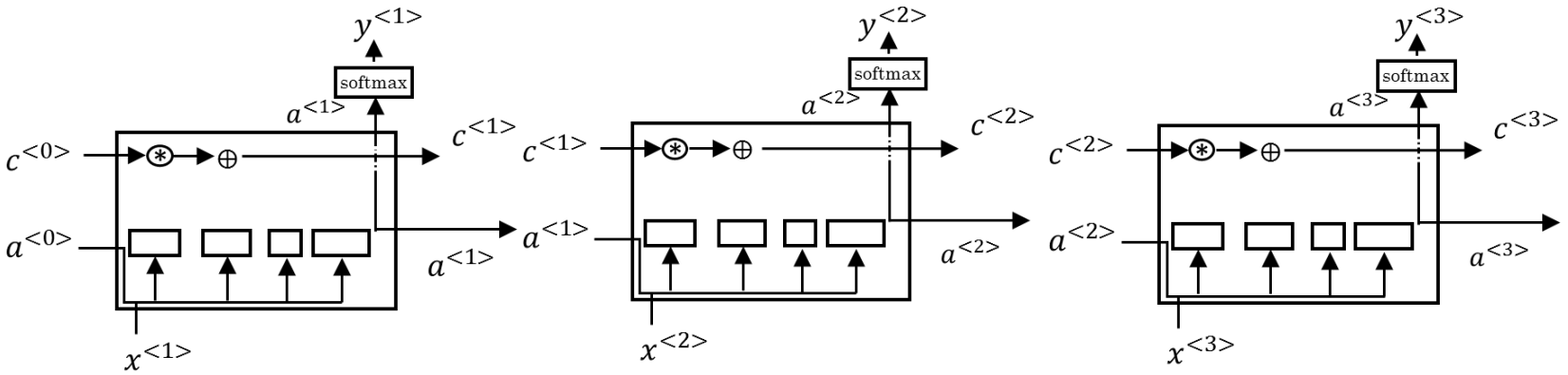
$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

Coursera: Deep learning Specialization, Andrew Ng

LSTM forward



$\Gamma_f \approx 0$, $c^{<3>} = c^{<0>}$

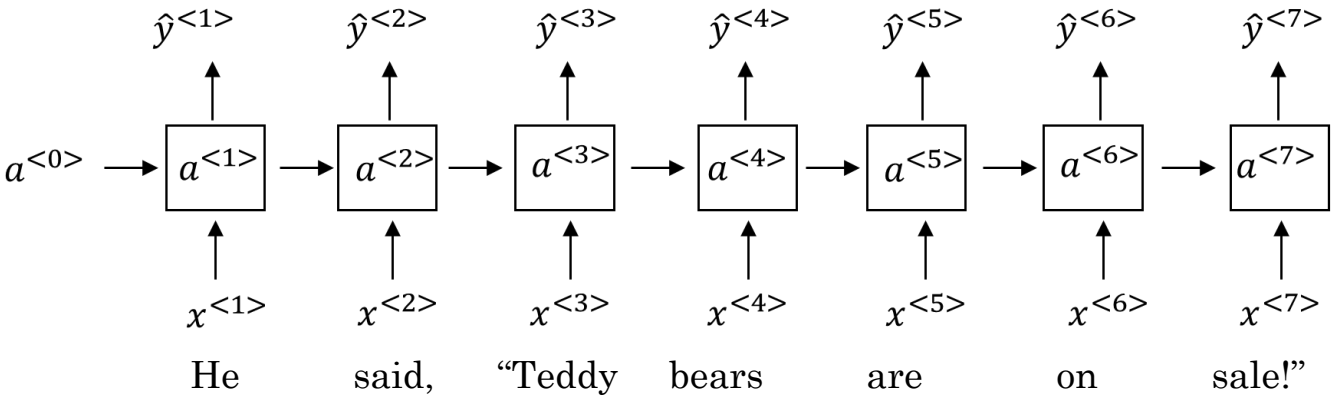
$c^{<t>}$ maintains many previous time-steps and helps vanishing gradient problem and allows long range dependencies.

Coursera: Deep learning Specialization, Andrew Ng

Unidirectional RNN

He said, “Teddy bears are on sale!”

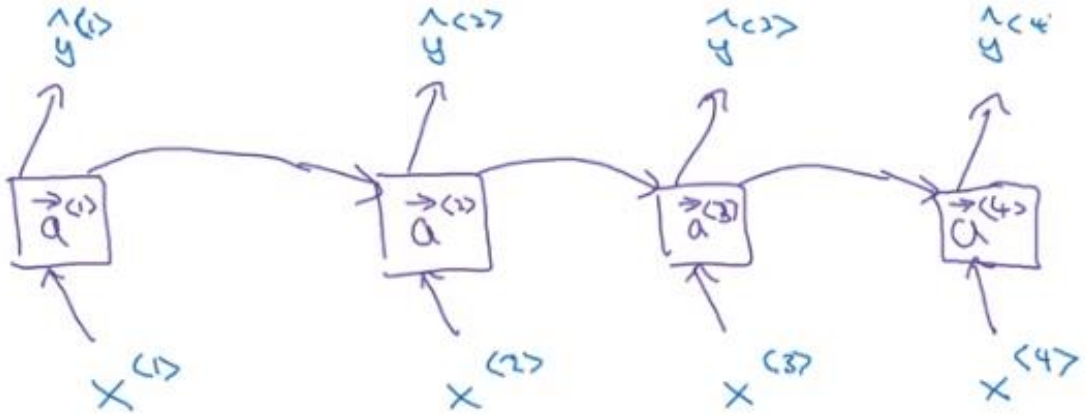
He said, “Teddy Roosevelt was a great President!”



Coursera: Deep learning Specialization, Andrew Ng

Bidirectional RNN

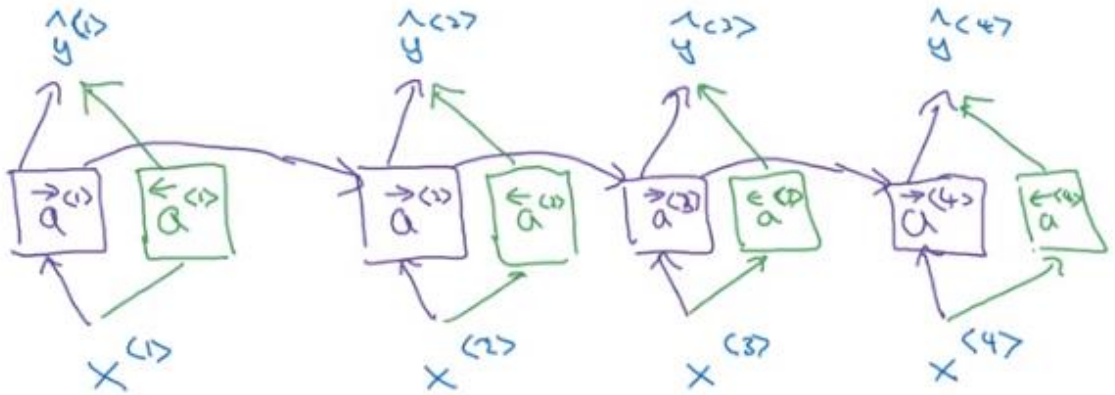
Getting information from the future



Coursera: Deep learning Specialization, Andrew Ng

Bidirectional RNN

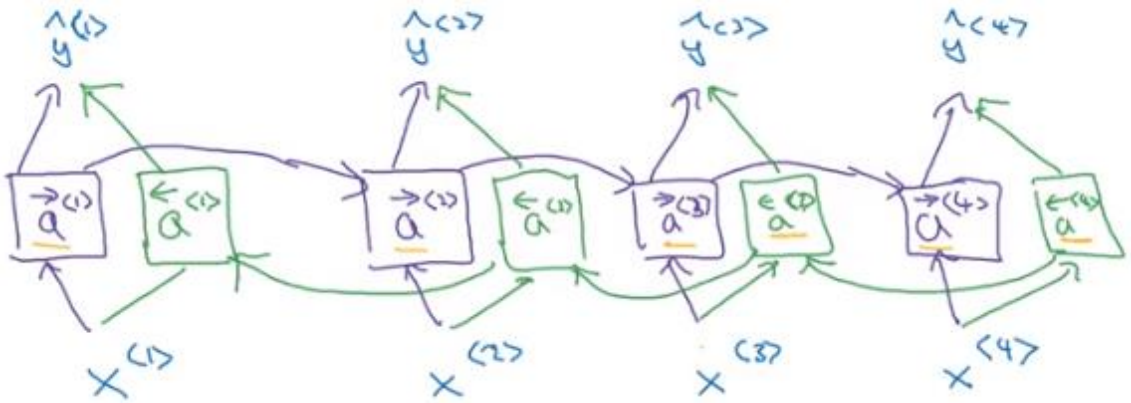
Getting information from the future



Coursera: Deep learning Specialization, Andrew Ng

Bidirectional RNN

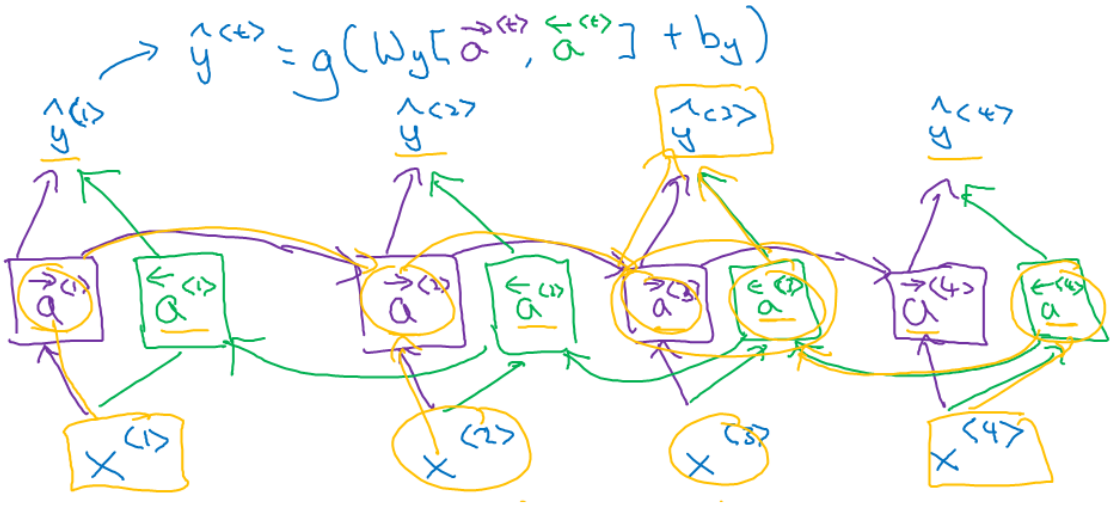
Getting information from the future



Coursera: Deep learning Specialization, Andrew Ng

Bidirectional RNN

Getting information from the future

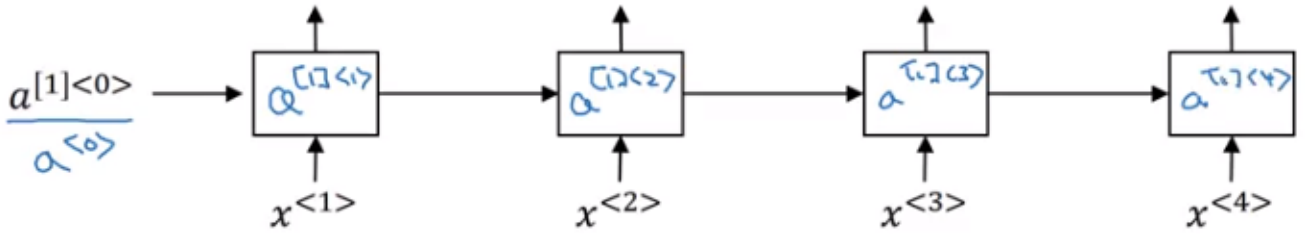


BRNN with GRU/LSTM

Entire sentence is needed to build BRNN.

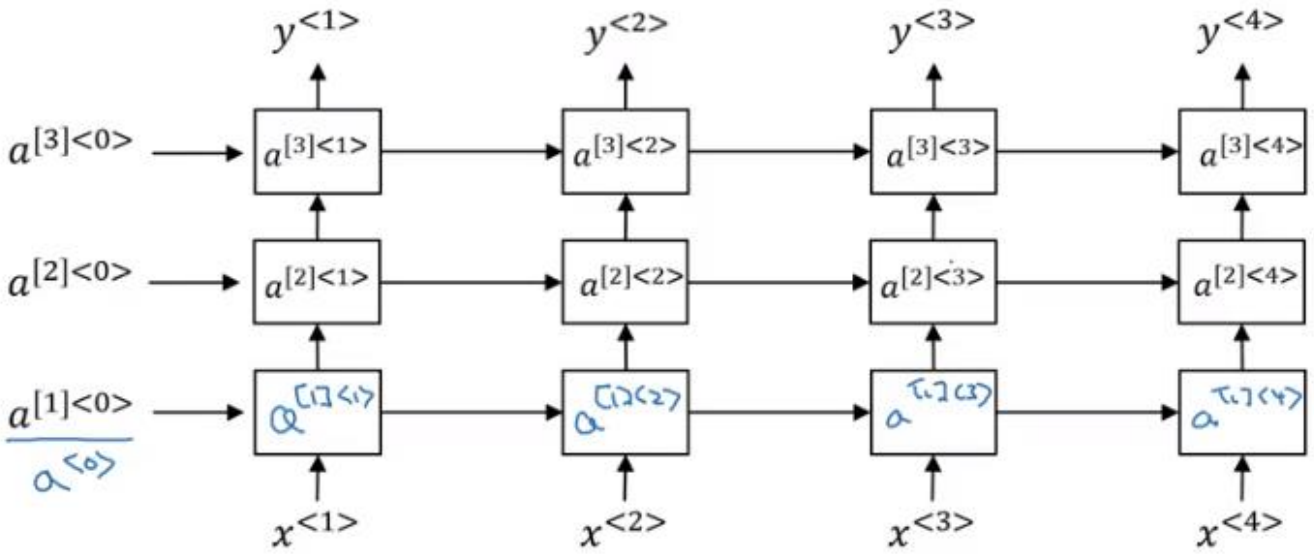
Coursera: Deep learning Specialization, Andrew Ng

Deep RNNs



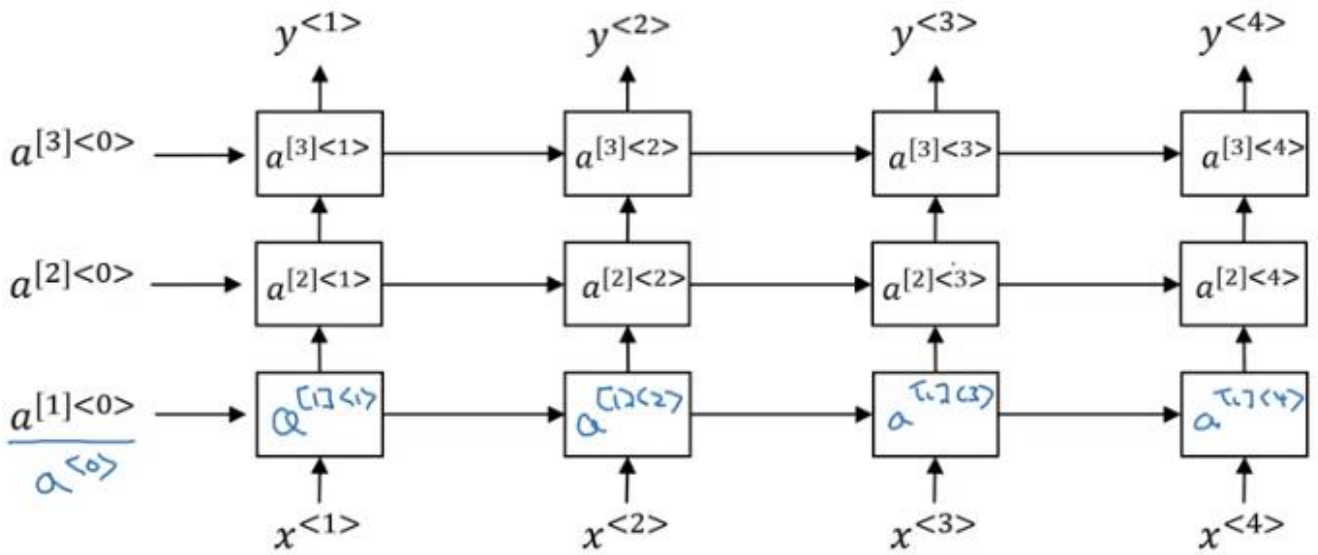
Coursera: Deep learning Specialization, Andrew Ng

Deep RNNs



Coursera: Deep learning Specialization, Andrew Ng

Deep RNNs



$$a^{[2]<3>} = g(W_a^{[2]}[\, a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$$

Coursera: Deep learning Specialization, Andrew Ng