Review for the Midterm

Shuai Li

John Hopcroft Center, Shanghai Jiao Tong University

https://shuaili8.github.io

https://shuaili8.github.io/Teaching/VE445/index.html

Exam code

- Exam on Oct 29 8:00-9:40 at Dong Xia Yuan 113 (lecture classroom)
- Finish the exam paper by yourself
- Allowed:
  - Calculator, watch (not smart)
- Not allowed:
  - Books, materials, cheat sheet, …
  - Phones, any smart device
- No entering after 8:25
- Early submission period: 8:30--9:25

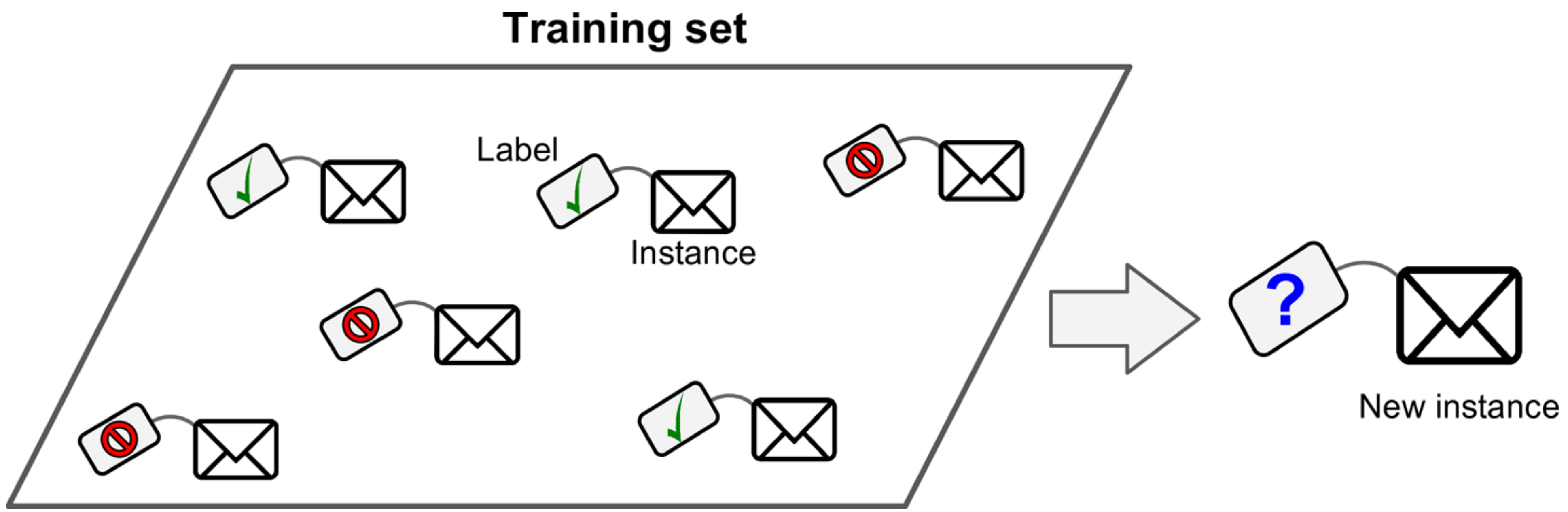
Coverage of midterm

- Basics
- Supervised learning
  - Linear Regression
  - Logistic regression
  - SVM and Kernel methods
  - Decision Tree
- Deep learning
  - Neural Networks
  - Backpropagation
  - Convolutional Neural Network
  - Recurrent Neural Network
- Unsupervised learning
  - K-means, PCA, EM, GMM
- Reinforcement learning
  - Multi-armed bandits
  - MDP
  - Bellman equations
  - Q-learning
- Learning theory
  - PAC, VC-dimension, bias-variance decomposition

Machine Learning Categories

- Unsupervised learning
    - No labeled data
- Supervised learning
    - Use labeled data to predict on unseen points
- Semi-supervised learning
    - Use labeled data and unlabeled data to predict on unlabeled/unseen points
- Reinforcement learning
    - Sequential prediction and receiving feedbacks

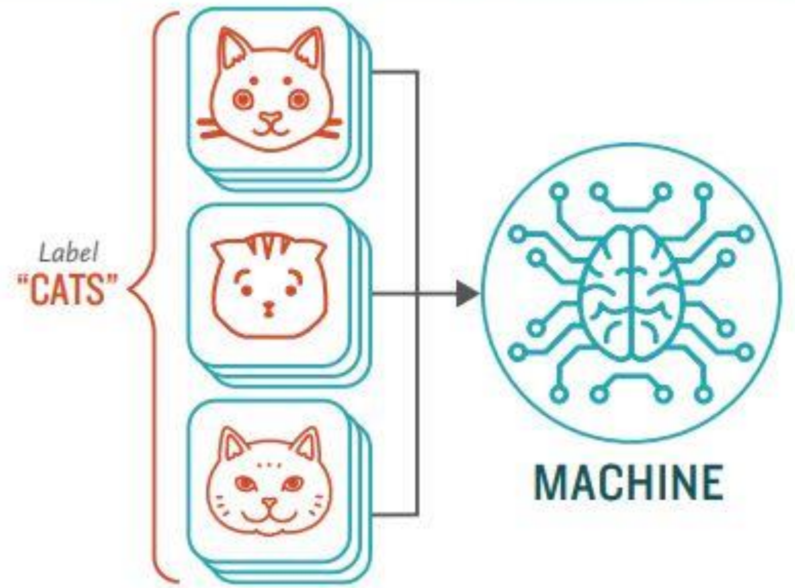
Supervised learning example



How **Supervised** Machine Learning Works

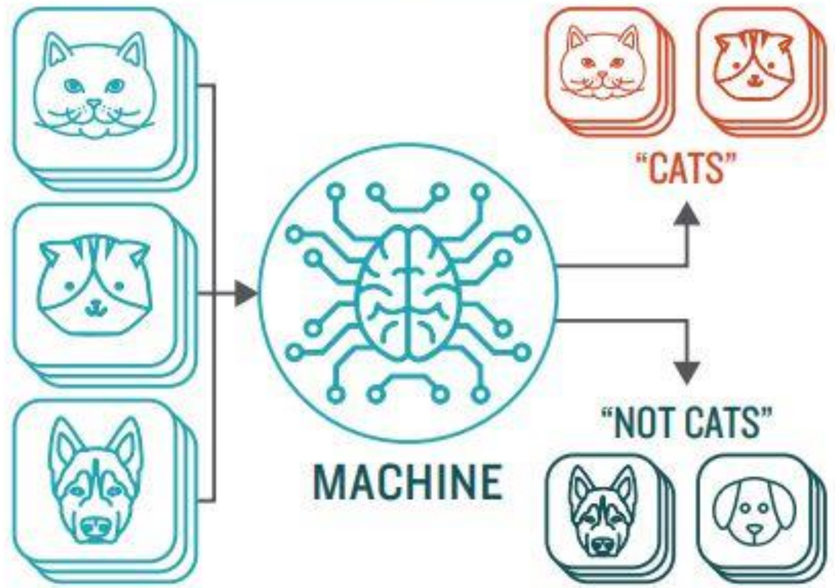
STEP 1

Provide the machine learning algorithm categorized or "labeled" input and output data from to learn

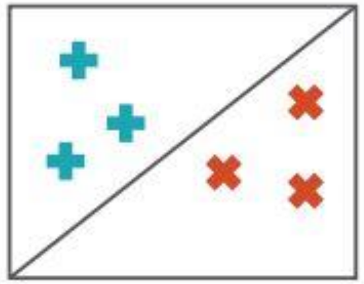


STEP 2

Feed the machine new, unlabeled information to see if it tags new data appropriately. If not, continue refining the algorithm

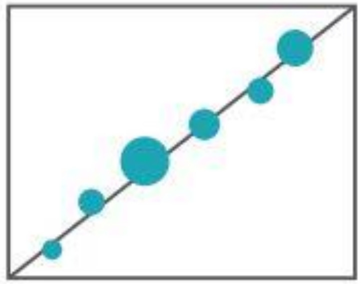


TYPES OF PROBLEMS TO WHICH IT'S SUITED



CLASSIFICATION

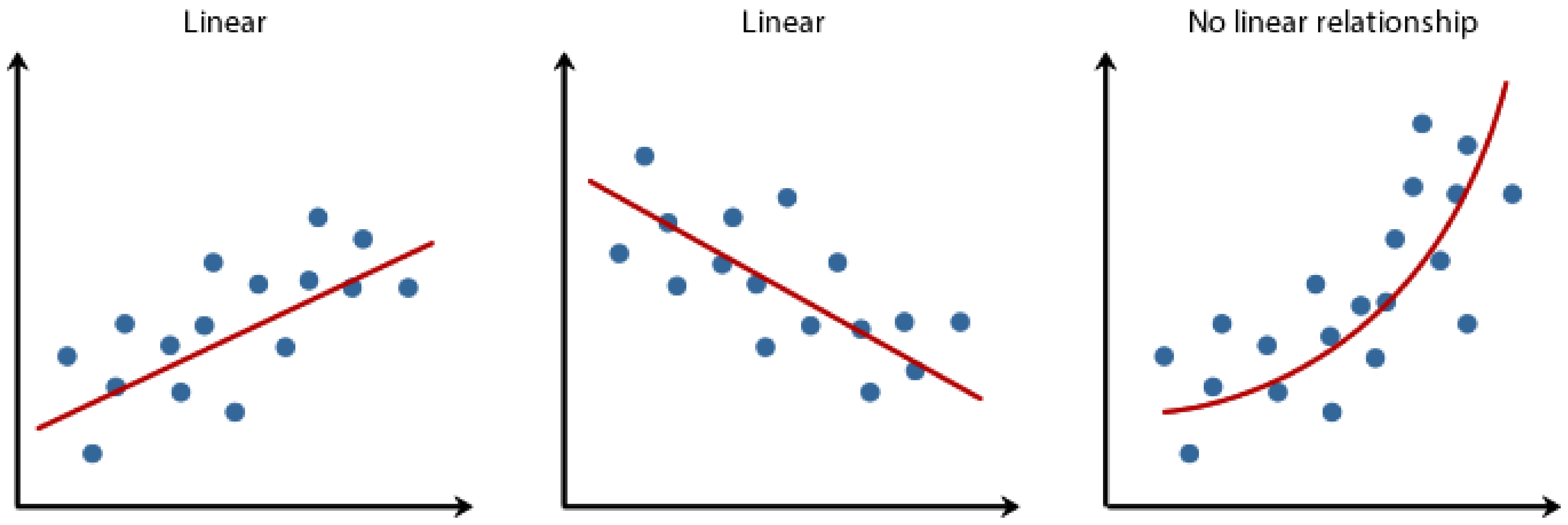
Sorting items into categories



REGRESSION

Identifying real values (dollars, weight, etc.)

Regression example



Model Evaluations

Classification -- Model evaluations

- Confusion Matrix
  - TP – True Positive ; FP – False Positive
  - FN – False Negative; TN – True Negative

	Predicted Class		
Actual Class		Class = Yes	Class = No
	Class = Yes	a (TP)	b (FN)
	Class = No	c (FP)	d (TN)

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Classification -- Model evaluations

- Limitation of Accuracy
  - Consider a 2-class problem
    - Number of Class 0 examples = 9990
    - Number of Class 1 examples = 10
  - If a “stupid” model predicts everything to be class 0, accuracy is 9990/10000 = **99.9** %

- The accuracy is misleading because the model does not detect any example in class 1

Classification -- Model evaluations

- Cost-sensitive measures

	Predicted Class	
Actual Class		
	Class = Yes	Class = No
Class = Yes	a (TP)	b (FN)
Class = No	c (FP)	d (TN)

$$\text{Precision (p)} = \frac{TP}{TP + FP} = \frac{a}{a + c}$$

$$\text{Recall (r)} = \frac{TP}{TP + FN} = \frac{a}{a + b}$$

$$\text{F - measure (F)} = \frac{2rp}{r + p} = \frac{2a}{2a + b + c}$$

Harmonic mean of Precision and Recall
(Why not just average?)

Example

- Given 30 human photographs, a computer predicts 19 to be male, 11 to be female. Among the 19 male predictions, 3 predictions are not correct. Among the 11 female predictions, 1 prediction is not correct.

	Predicted Class		
Actual Class		Male	Female
	Male	a = TP = 16	b = FN = 1
	Female	c = FP = 3	d = TN = 10

Example

	Predicted Class		
Actual Class		Male	Female
	Male	a = TP = 16	b = FN = 1
	Female	c = FP = 3	d = TN = 10

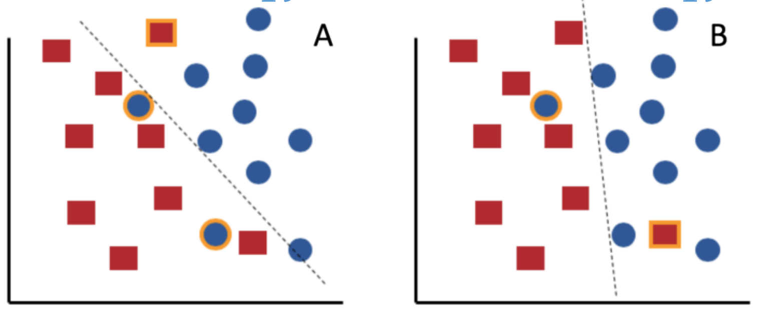
- Accuracy = (16 + 10) / (16 + 3 + 1 + 10) = 0.867
- Precision = 16 / (16 + 3) = 0.842
- Recall = 16 / (16 + 1) = 0.941
- F-measure = 2 (0.842)(0.941) / (0.842 + 0.941)
  = 0.889

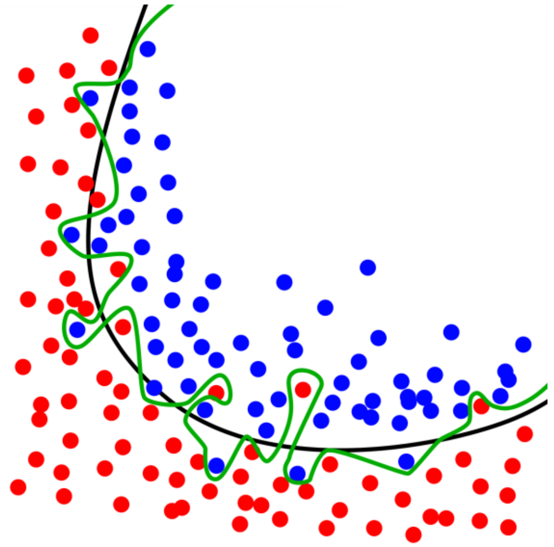
Model Selections

Minimize the error rate?

- Given a data set S
- Error rate = $\frac{\text{\# of Errors}}{\text{\# of Total Samples}}$
- Accuracy = 1 - Error rate

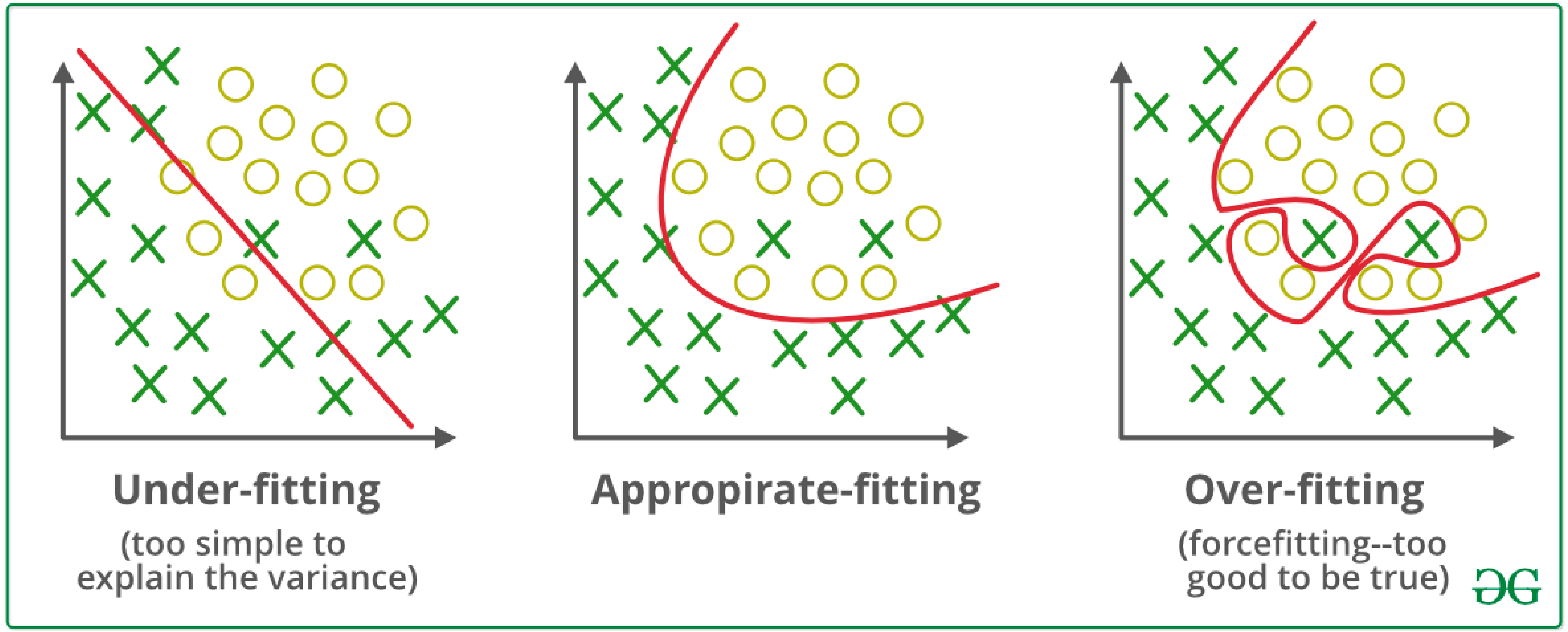
Error rate = $\frac{3}{19}$ Error rate = $\frac{2}{19}$





https://malware.news/uploads/default/original/3X/6/d/6df12e50b7f97cdba92697ce164cbe4a5502a349.png
https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Overfitting.svg/1200px-Overfitting.svg.png

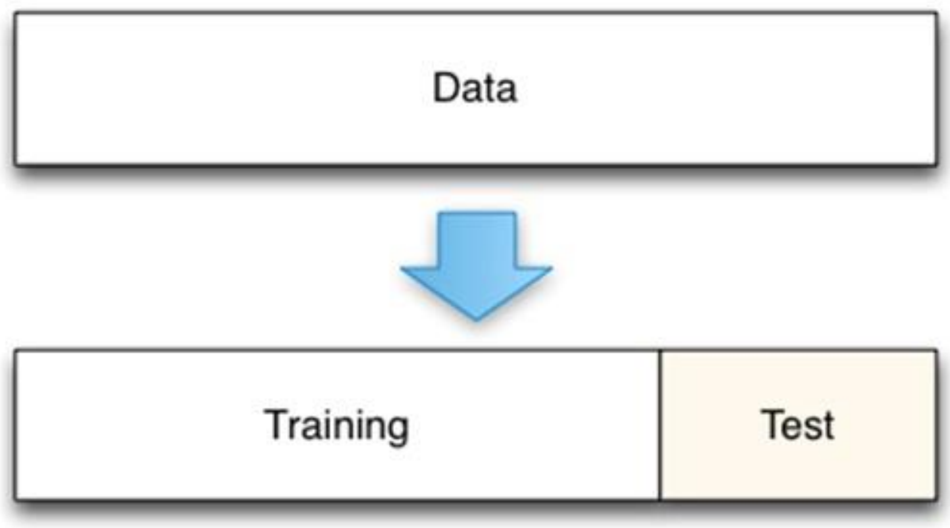
Fitting


Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

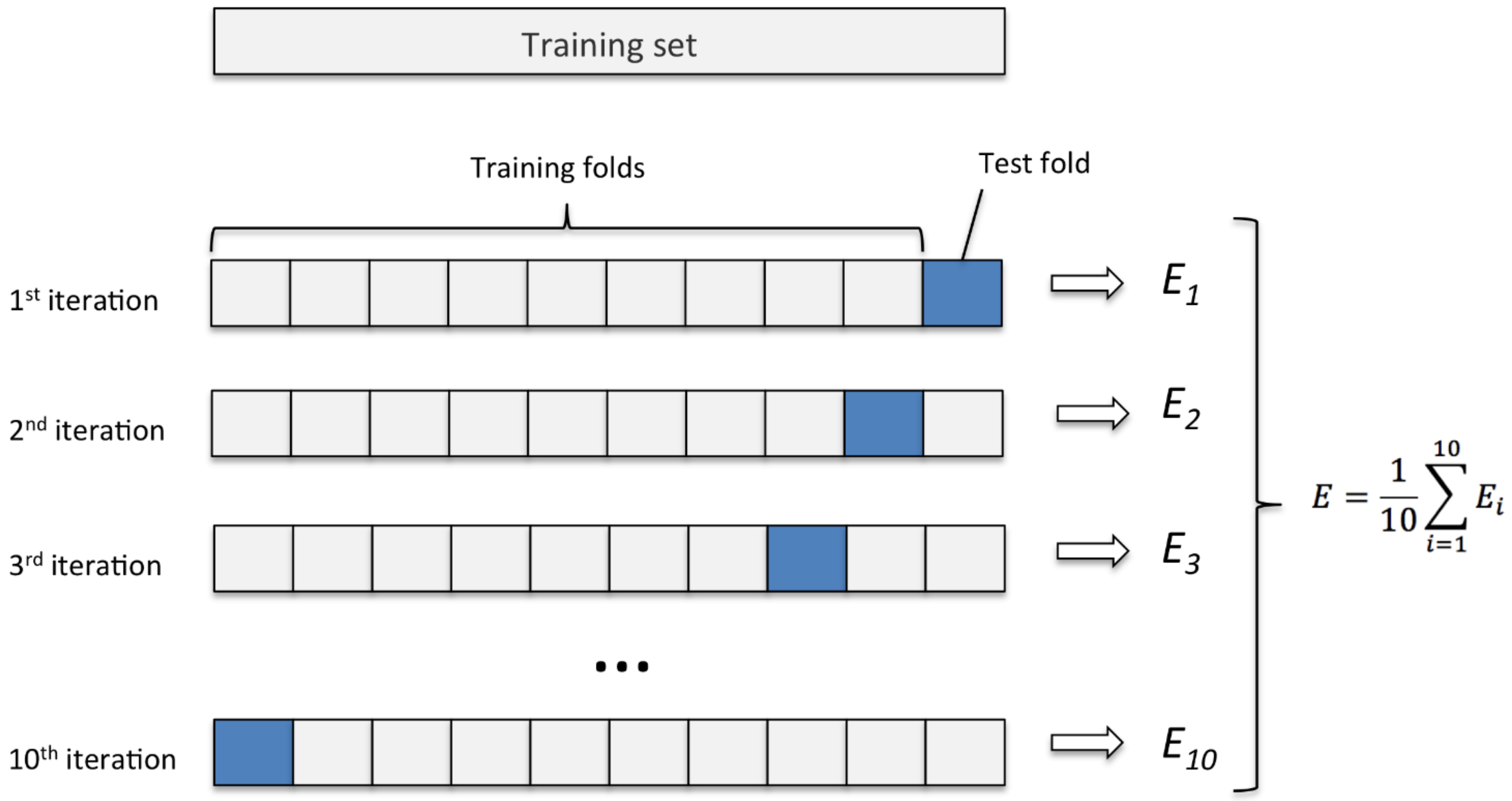
https://media.geeksforgeeks.org/wp-content/cdn-uploads/20190523171258/overfitting_2.png

Split training and test

- Split dataset to training and test
- Train models on training dataset
- The evaluation of the model is the error on test dataset
- Might overfit the training dataset

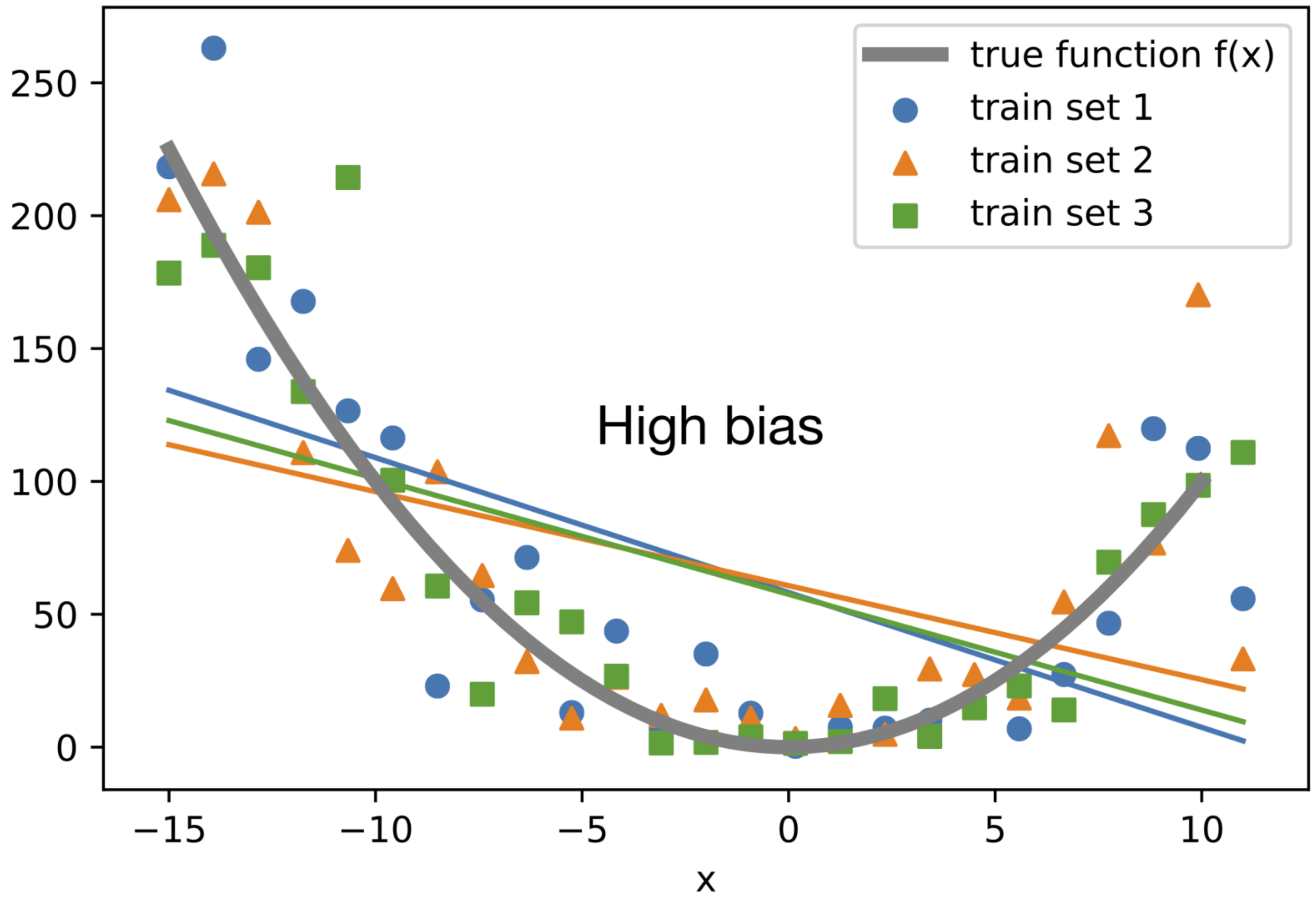


Cross validation



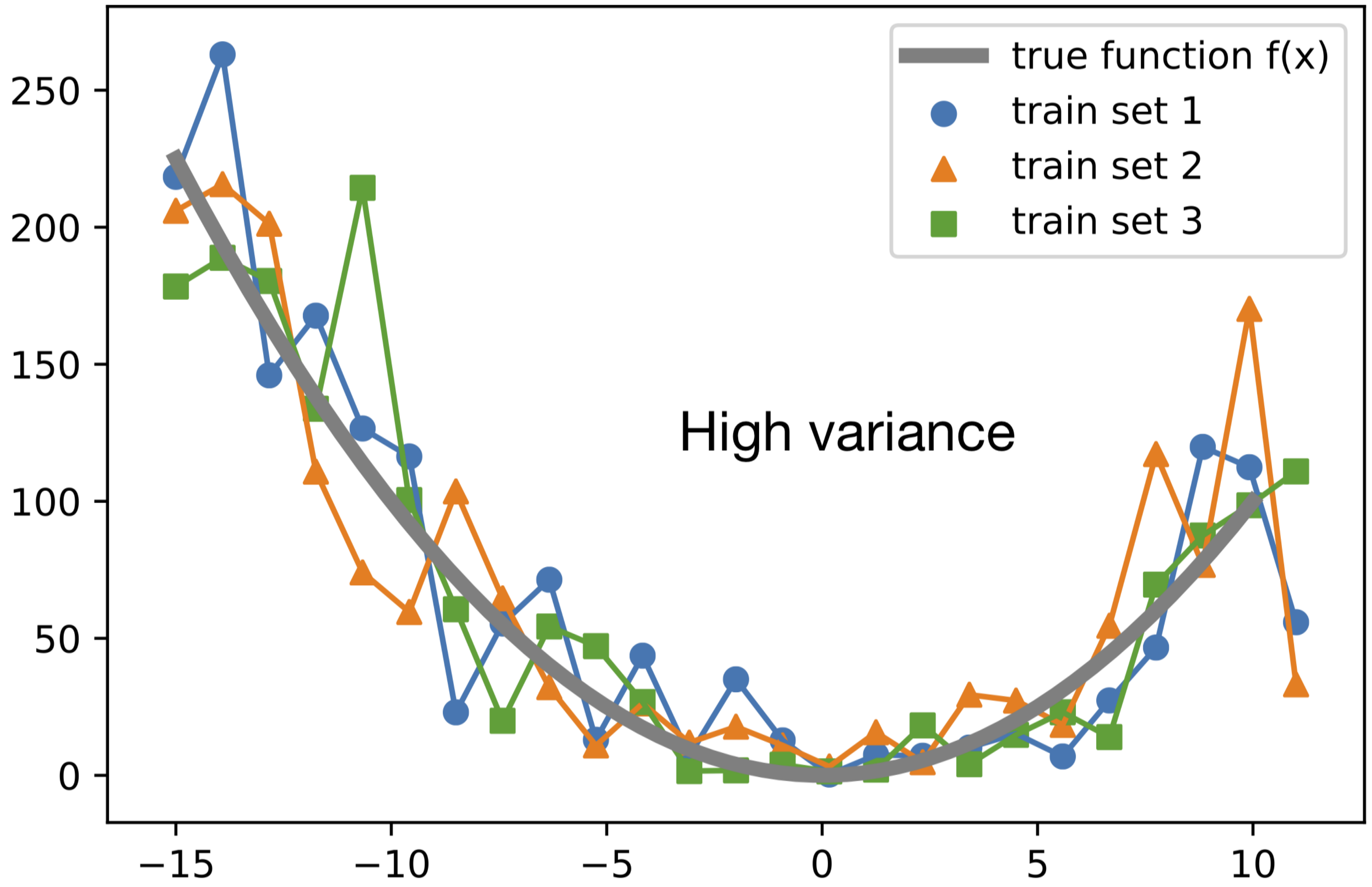
Bias

$$\text{Bias} = E[\hat{\theta}] - \theta.$$

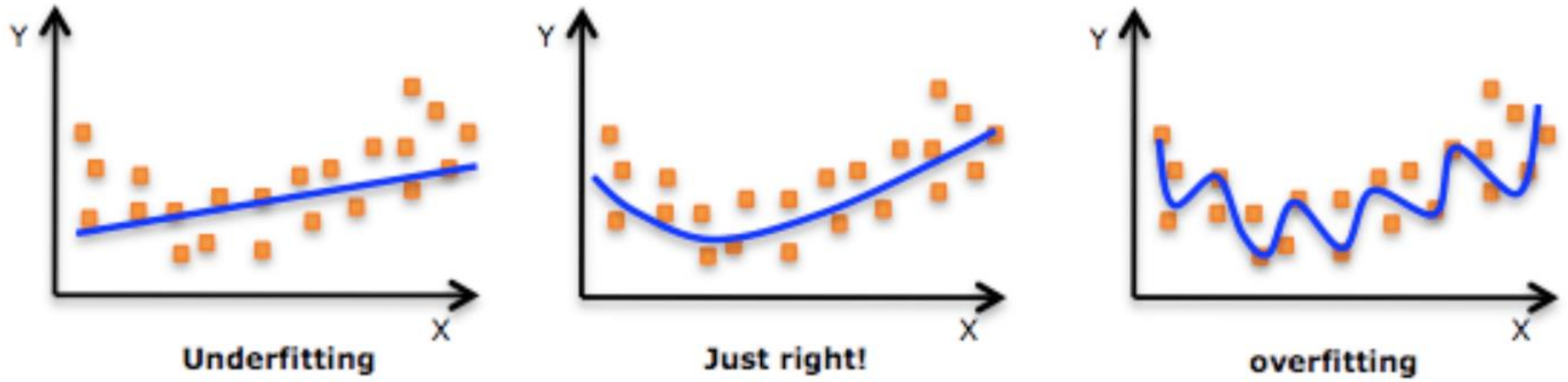


Variance

$$\mathrm{Var}(\hat{\theta}) = E[(E[\hat{\theta}] - \hat{\theta})^2].$$



Underfitting and overfitting



Bias-variance decomposition

True value

Estimated value

$$S = (y - \hat{y})^2$$

$$\begin{aligned}(y - \hat{y})^2 &= (y - E[\hat{y}] + E[\hat{y}] - \hat{y})^2 \\ &= (y - E[\hat{y}])^2 + (E[\hat{y}] - y)^2 + 2(y - E[\hat{y}])(E[\hat{y}] - \hat{y}).\end{aligned}$$

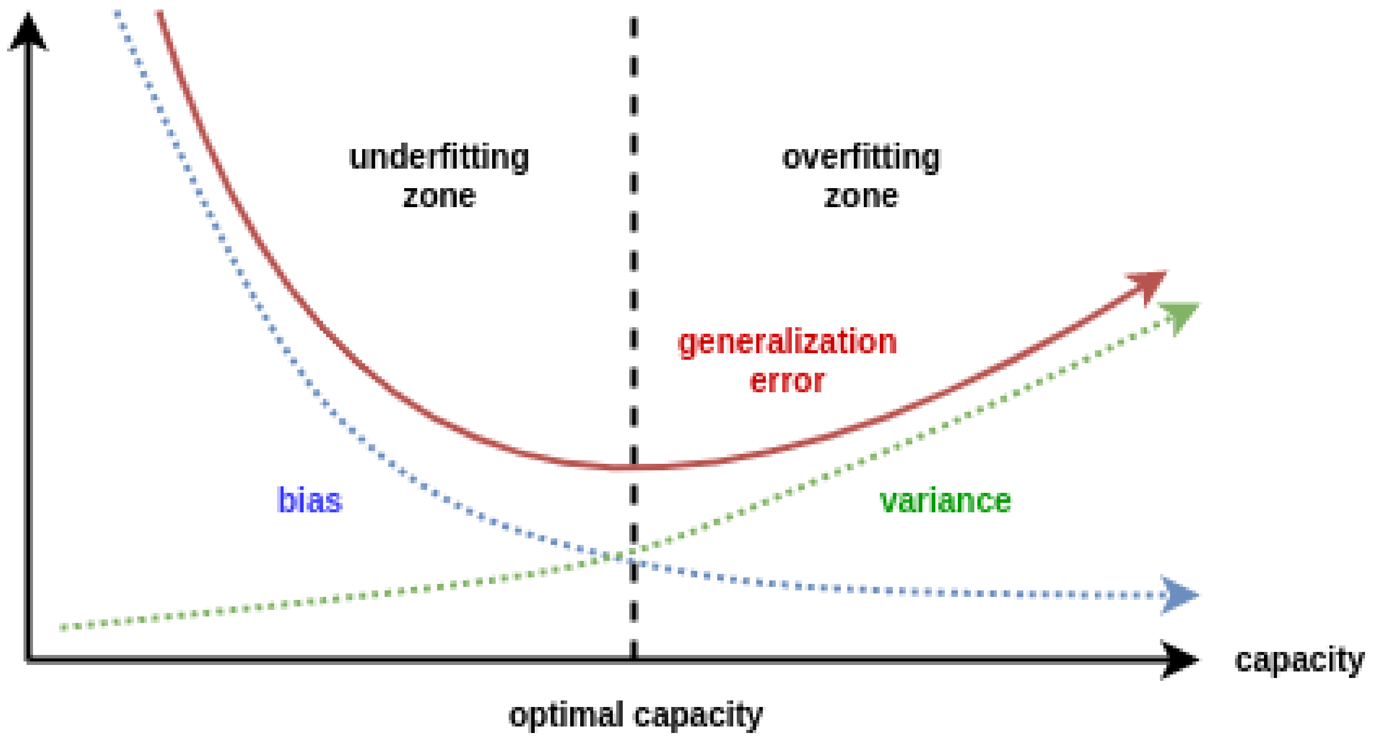
$$\begin{aligned}E[2(y - E[\hat{y}])(E[\hat{y}] - \hat{y})] &= 2E[(y - E[\hat{y}])(E[\hat{y}] - \hat{y})] \\ &= 2(y - E[\hat{y}])E[(E[\hat{y}] - \hat{y})] \\ &= 2(y - E[\hat{y}])(E[E[\hat{y}]] - E[\hat{y}]) \\ &= 2(y - E[\hat{y}])(E[\hat{y}] - E[\hat{y}]) \\ &= 0.\end{aligned}$$

$$E[S] = E[(y - \hat{y})^2]$$

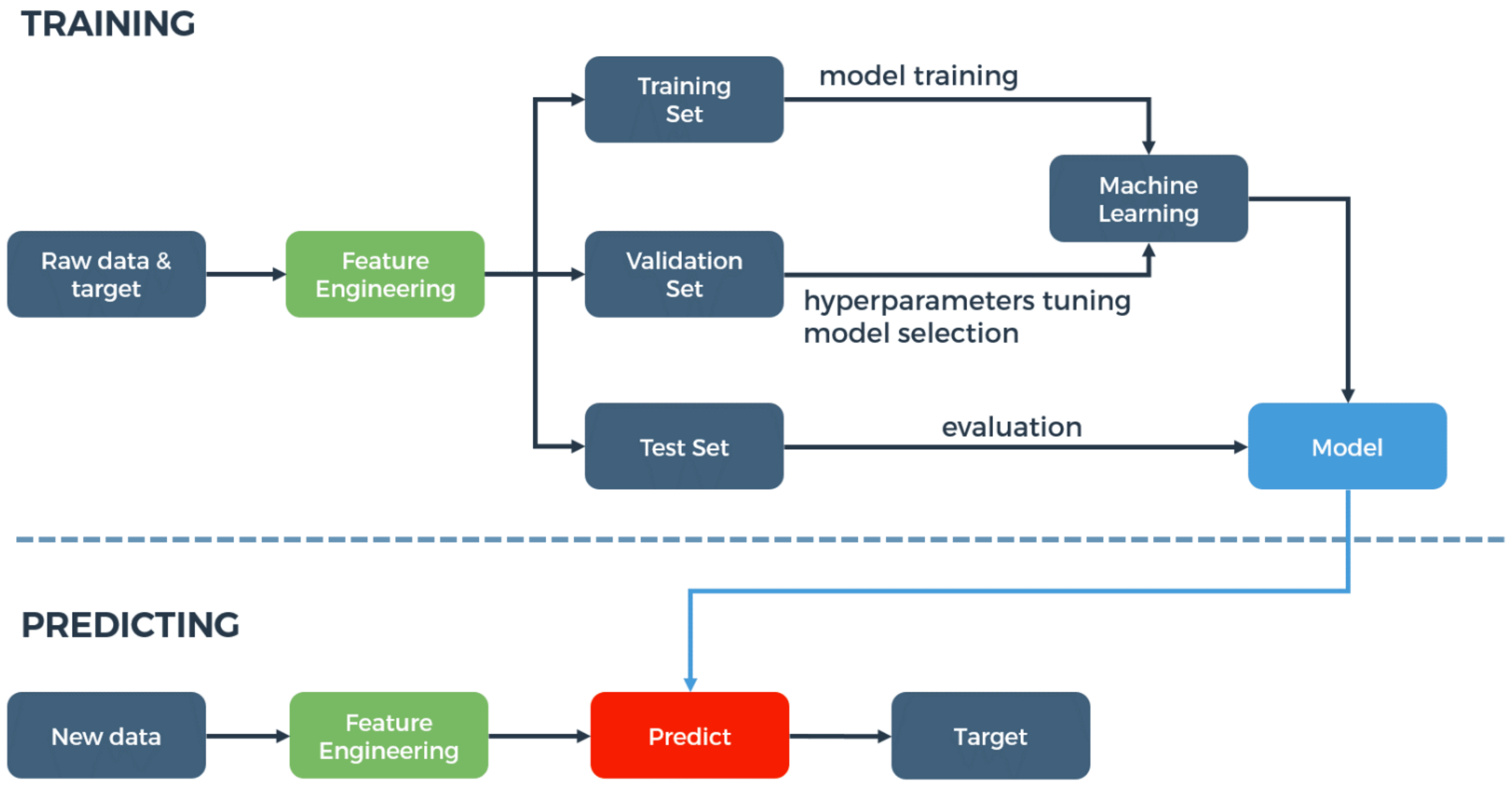
$$\begin{aligned}E[(y - \hat{y})^2] &= (y - E[\hat{y}])^2 + E[(E[\hat{y}] - \hat{y})^2] \\ &= [\text{Bias}]^2 + \text{Variance}.\end{aligned}$$

Can be understood by replacing y and $\hat{y}$ with model parameters θ and $\hat{\theta}$

Balance bias-variance trade-off



Machine Learning Process



https://techblog.cdiscount.com/assets/images/DataScience/automl/ML_process.png

Linear Regression

Linear regression

- Use linear relationship to approximate the function of Y on X

- How to select the most appropriate linear model?
- Error: Mean squared error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

  - Where Y and $\hat{Y}$ are the true values and predicted values
- Find the linear model with the smallest MSE

Question

- Given the dataset

$$(1,1),(2,4),(3,5)$$

  and the linear model

$$Y = 2X + 1$$

- What is the mean squared error?

- The predicted points are

$$(1,3),(2,5),(3,7)$$

- So the mean squared error (MSE) is

$$\frac{1}{3}(2^2 + 1^2 + 2^2) = 3$$

Question 2

- Given the <real value, predicted value> pairs as: $< 1.2, 1.7 >, < 0.9, 0.2 >, < -0.3, 0.1 >, < 1.3, 0.3 >, < 1.1, 1.2 >$ compute the means squared error

- The answer is

$$\frac{1}{5}(0.5^2 + 0.7^2 + 0.4^2 + 1^2 + 0.1^2) = 0.382$$

How to get linear model with minimal MSE

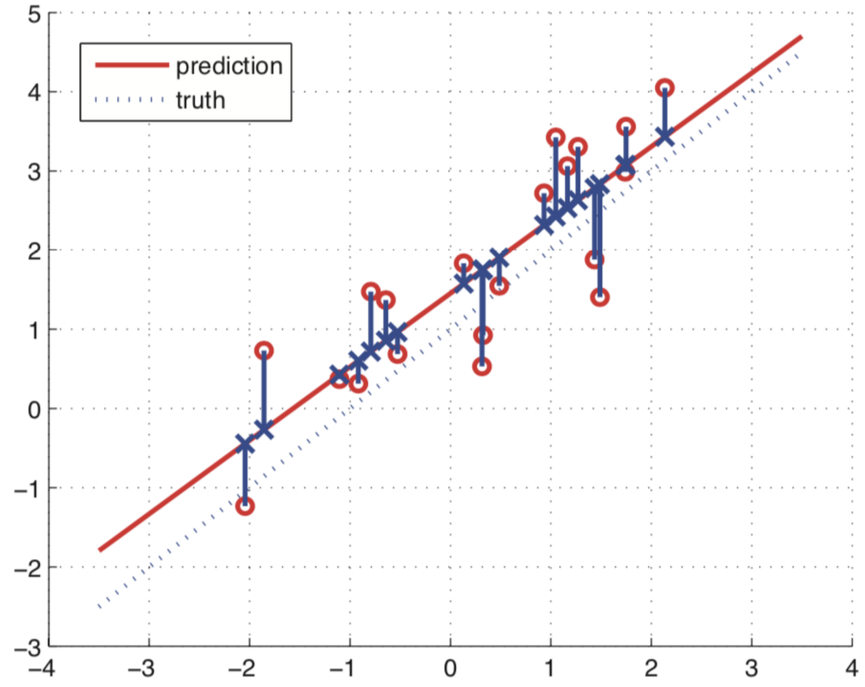
- MSE for model parameter θ:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \theta^\top x_i)^2$$

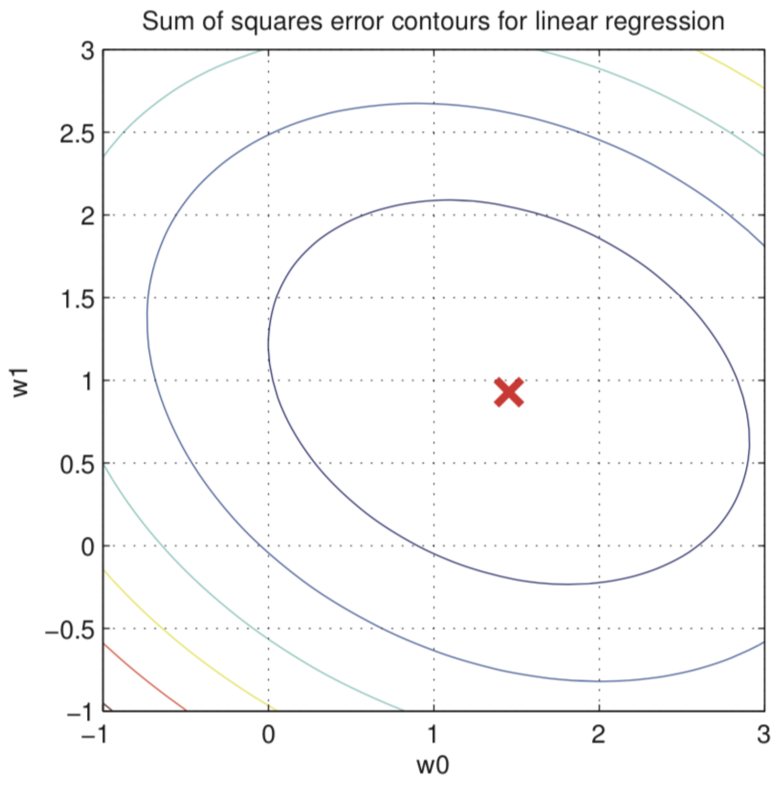
- Find an estimator $\hat{\theta}$ to minimize $J(\theta)$
- $y = \theta^\top x + b + \varepsilon$. Then we can write $x' = \left(1, x^1, \dots, x^d\right), \theta = (b, \theta_1, \dots, \theta_d)$, then $y = \theta^\top x' + \varepsilon$

- Note that $J(\theta)$ is a convex function in θ, so it has a unique minimal point

Interpretation



(a)



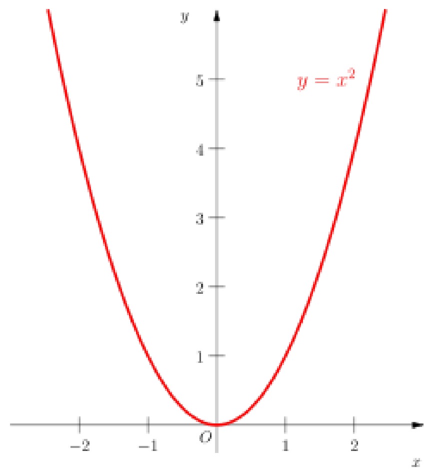
(b)

Figure credit: Kevin Murphy

$J(\theta)$ is convex

Check it by yourself !

- $f(x) = (y-x)^2 = (x-y)^2$ is convex in x



- $g(\theta) = f(\theta^\top x)$

$$\begin{aligned}&g\big((1-t)\theta_1 + t\theta_2\big)\\ &= f\left((1-t){\theta_1}^\top x + t{\theta_2}^\top x\right)\\ &\le (1-t)f\big({\theta_1}^\top x\big) + tf\big({\theta_2}^\top x\big)\\ &= (1-t)g(\theta_1) + tg(\theta_2)\end{aligned}$$

Convexity of f

- The sum of convex functions is convex
- Thus $J(\theta)$ is convex

Minimal point (Normal equation)

- $\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{N}\sum_{i=1}^{N}(\theta^\top x_i - y_i)x_i = \frac{2}{N}\sum_{i=1}^{N}\left(x_i x_i^\top \theta - x_i y_i\right)$
- Letting the derivative be zero

$$\left(\sum_{i=1}^{N} x_i x_i^\top\right)\theta = \sum_{i=1}^{N} x_i y_i$$

- If we write $X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_N^\top \end{bmatrix} = \begin{bmatrix} x_1^1 & \cdots & x_1^d \\ & \vdots & \\ x_N^1 & \cdots & x_N^d \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$, then

$$X^\top X\theta = X^\top y$$

Minimal point (Normal equation) (cont.)

- $X^\top X\theta = X^\top y$
- When $X^\top X$ is invertible

$$\hat{\theta} = (X^\top X)^{-1} X^\top y$$

Question 1

- Given the dataset

$$(1,1), (2,4), (3,5)$$

compute the normal equation for θ, solve θ and compute the MSE

$$\boxed{X^\top X\theta = X^\top y}$$

- $X = \begin{bmatrix} x_1^\top \\ x_2^\top \\ x_3^\top \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, y = \begin{bmatrix} 1 \\ 4 \\ 5 \end{bmatrix}$

$$X^\top X = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}, X^\top y = \begin{bmatrix} 10 \\ 24 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 10 \\ 24 \end{bmatrix}$$

- $\theta = \left[-\frac{2}{3}, 2\right], \mathrm{y} = -\frac{2}{3} + 2\mathrm{x}$. MSE=$\frac{2}{9}$

Question 2

- Some economist say that the impact of GDP in ‘current year’ will have effect on vehicle sales ‘next year’. So whichever year GDP was less, the coming year sales was lower and when GDP increased the next year vehicle sales also increased.

- Let’s have the equation as $y = \theta_0 + \theta_1 x$, where
- y = number of vehicles sold in the year
- x = GDP of prior year

We need to find θ_0 and θ_1

Question 2 (cont.)

- Here is the data between 2011 and 2016.

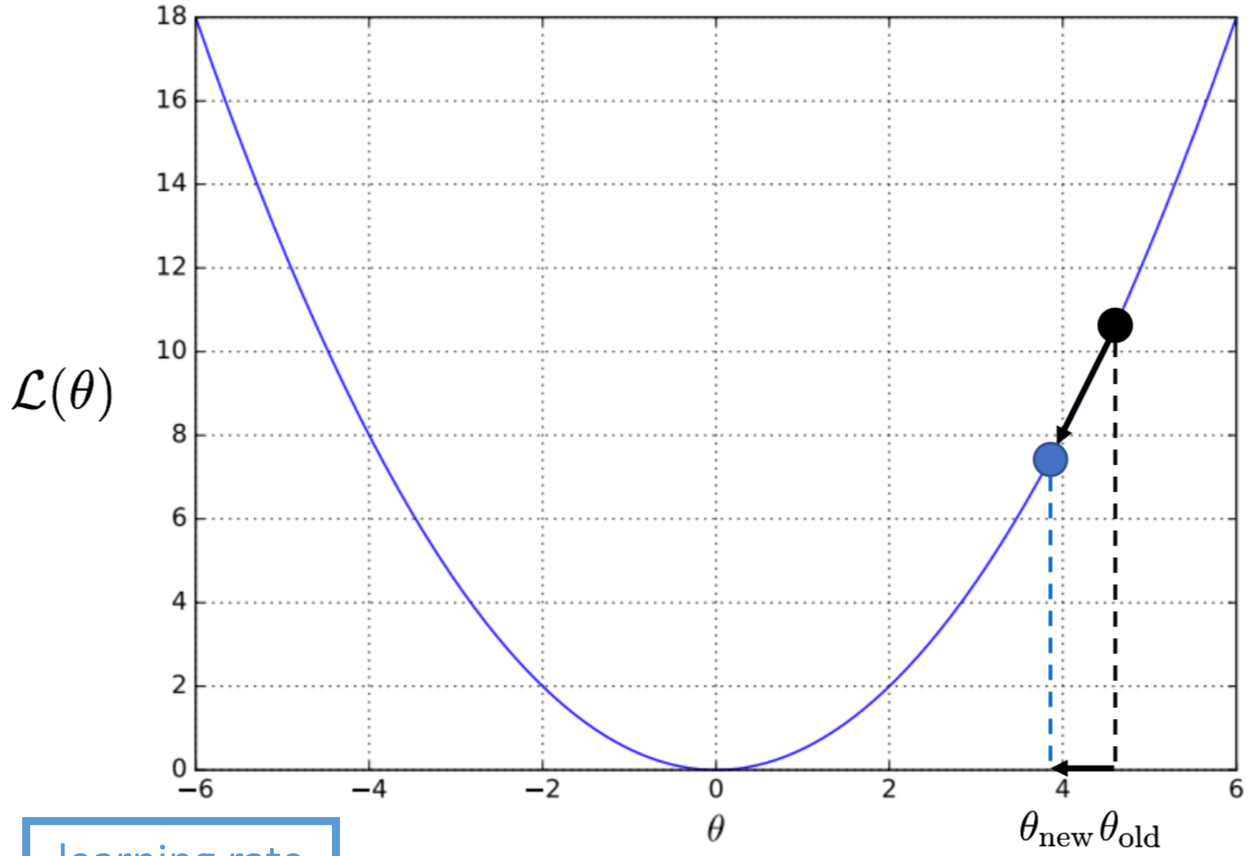
Year	GDP	Sales of vehicle
2011	6.2	
2012	6.5	26.3
2013	5.48	26.65
2014	6.54	25.03
2015	7.18	26.01
2016	7.93	27.9
2017		30.47
2018		

Homework

- Question 1: what is the normal equation?
- Question 2: suppose the GDP increasement in 2017 is 7%, how many vehicles will be sold in 2018?

Gradient method motivation – large dataset

- Too big to compute directly
$$\hat{\theta} = (X^\top X)^{-1} X^\top y$$
- Recall the objective is to minimize the loss function
$$L(\theta) = J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \theta^\top x_i)^2$$
- Gradient descent method



learning rate

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} - \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta}$$

Batch gradient descent

- $f_\theta(x) = \theta^\top x$

$$J(\theta) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))^2 \qquad \min_\theta J(\theta)$$

- Update $\theta_{\text{new}} \leftarrow \theta_{\text{old}} - \eta\frac{\partial J(\theta)}{\partial \theta}$ for the whole batch

$$\frac{\partial J(\theta)}{\partial \theta} = -\frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))\frac{\partial f_\theta(x_i)}{\partial \theta}$$

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

$$\theta_{\text{new}} = \theta_{\text{old}} + \eta\frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))x_i$$

Stochastic gradient descent

$$J^{(i)}(\theta) = \frac{1}{2}(y_i - f_\theta(x_i))^2 \qquad \min_\theta \frac{1}{N}\sum_i J^{(i)}(\theta)$$

- Update $\theta_{\text{new}} = \theta_{\text{old}} - \eta \frac{\partial J^{(i)}(\theta)}{\partial \theta}$ for $\boxed{\text{every single instance}}$

$$\begin{aligned}\frac{\partial J^{(i)}(\theta)}{\partial \theta} &= -(y_i - f_\theta(x_i))\frac{\partial f_\theta(x_i)}{\partial \theta} \\ &= -(y_i - f_\theta(x_i))x_i \\ \theta_{\text{new}} &= \theta_{\text{old}} + \eta(y_i - f_\theta(x_i))x_i\end{aligned}$$

- Compare with BGD
  - Faster learning
  - Uncertainty or fluctuation in learning

Mini-Batch Gradient Descent

- A combination of batch GD and stochastic GD

- Split the whole dataset into *K* mini-batches

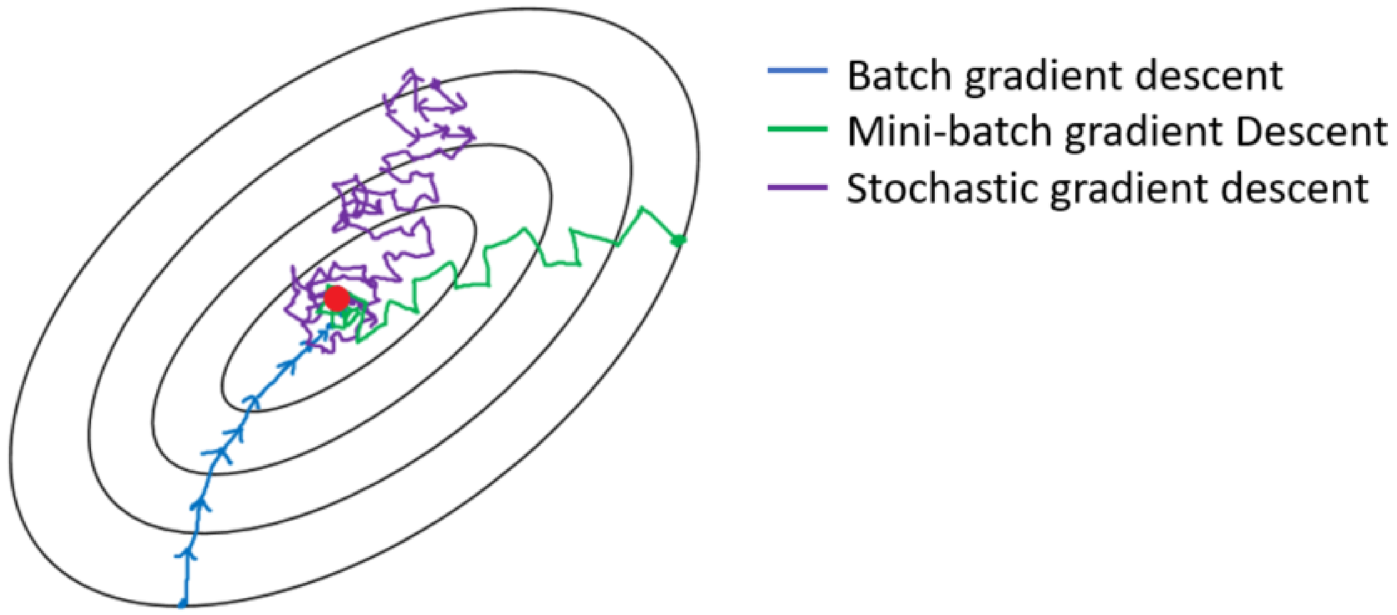
$$\{1, 2, 3, \ldots, K\}$$

- $\boxed{\text{For each mini-batch } k,}$ perform one-step BGD towards minimizing

$$J^{(k)}(\theta) = \frac{1}{2N_k} \sum_{i=1}^{N_k} (y_i - f_\theta(x_i))^2$$

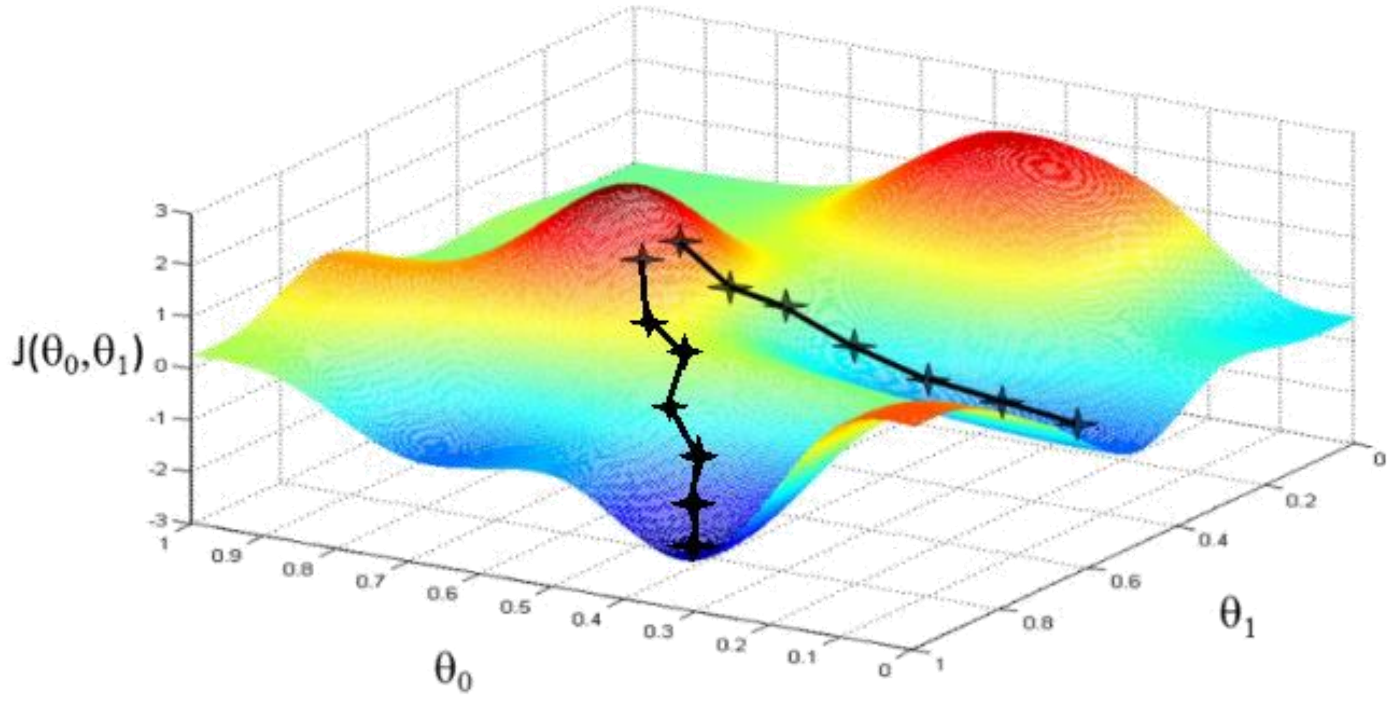
- Update $\theta_{\text{new}} = \theta_{\text{old}} - \eta \frac{\partial J^{(k)}(\theta)}{\partial \theta}$ for each mini-batch

Comparisons

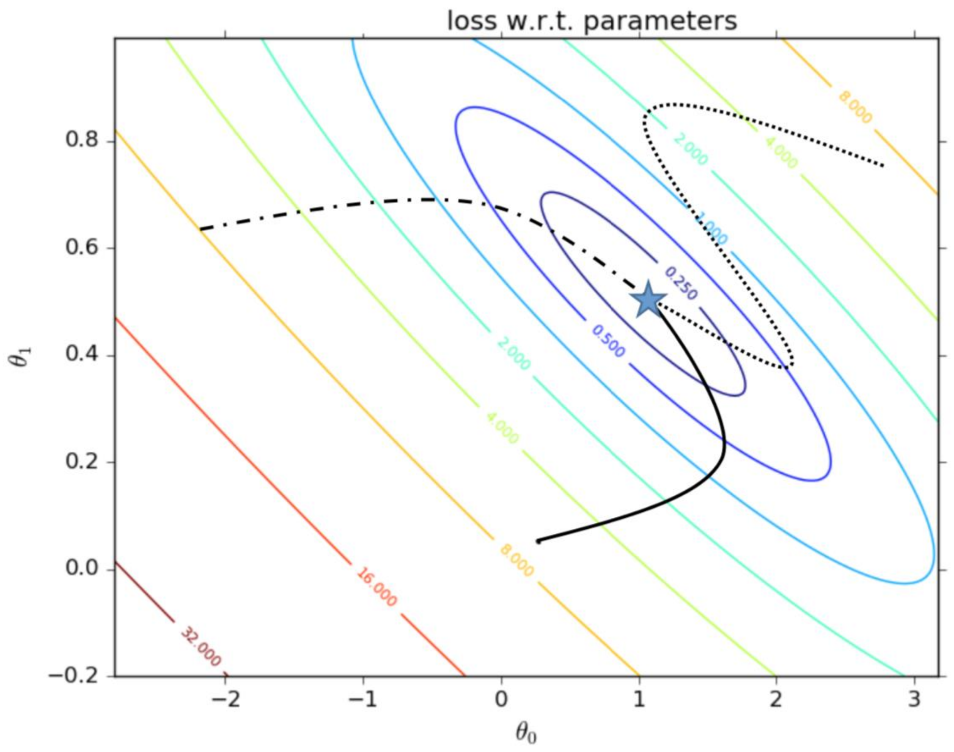


Searching

- Start with a new initial value θ
- Update θ iteratively (gradient descent)
- Ends at a minimum



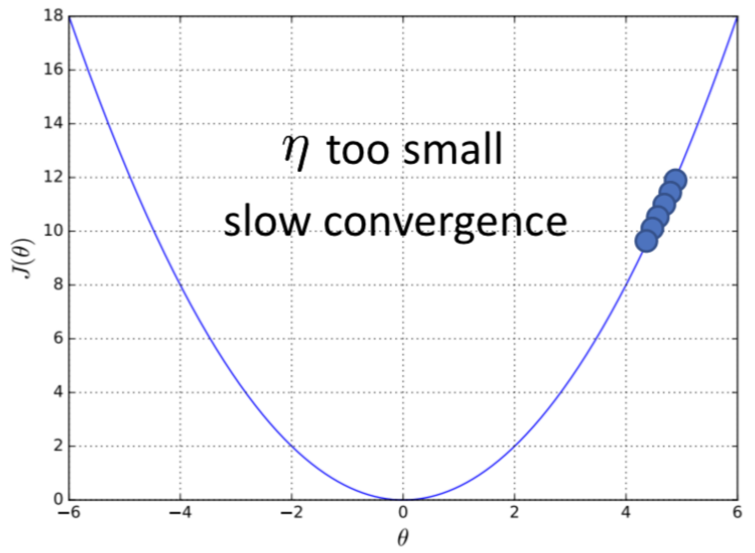
Uniqueness of minimum for convex objectives

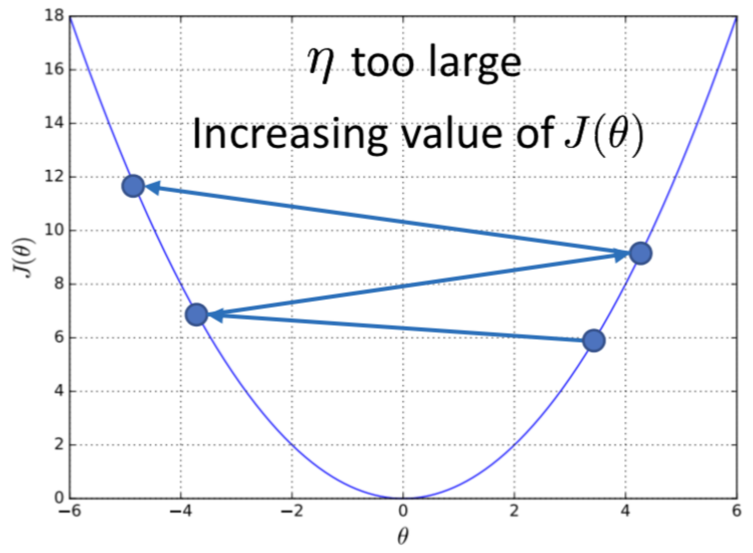


- Different initial parameters and different learning algorithm lead to the same optimum

Learning rate

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \frac{\partial J(\theta)}{\partial \theta}$$





- The initial point may be too far away from the optimal solution, which takes much time to converge

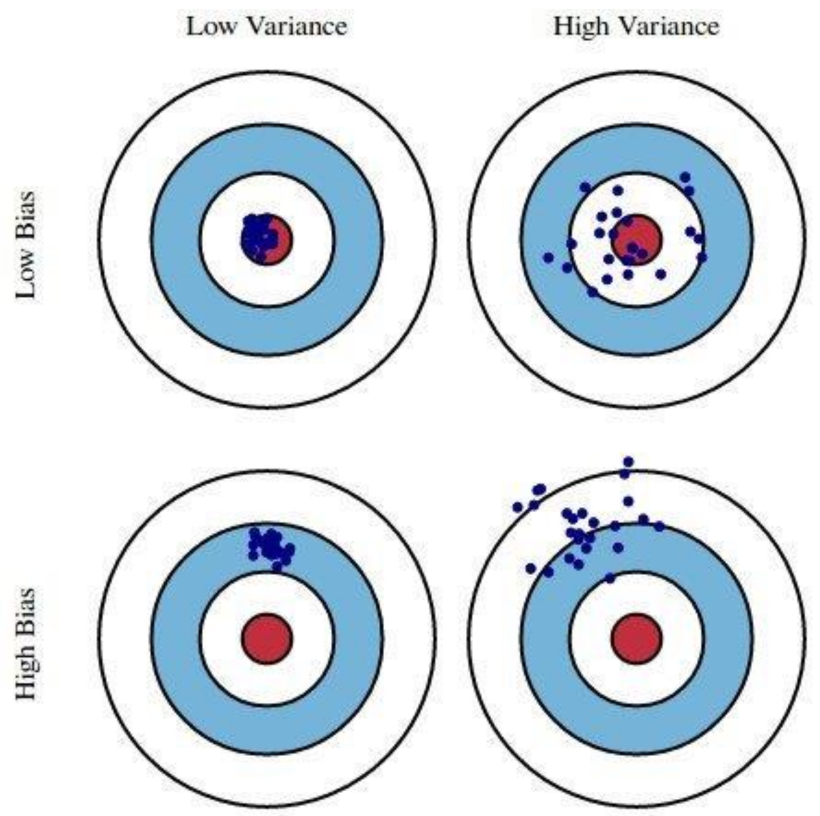
- May overshoot the minimum
- May fail to converge
- May even diverge

- To see if gradient descent is working, print out $J(\theta)$ for each or every several iterations. If $J(\theta)$ does not drop properly, adjust η

Regularization

Problems of ordinary least squares (OLS)

- Best model is to minimize both the bias and the variance
- Ordinary least squares (OLS)
  - Previous linear regression
  - Unbiased
  - Can have huge variance
    - Multi-collinearity among data
      - When predictor variables are correlated to each other and to the response variable
      - E.g. To predict patient weight by the height, sex, and diet. But height and sex are correlated
    - Many predictor variables
      - Feature dimension close to number of data points
- Solution
  - Reduce variance at the cost of introducing some bias
  - Add a penalty term to the OLS equation

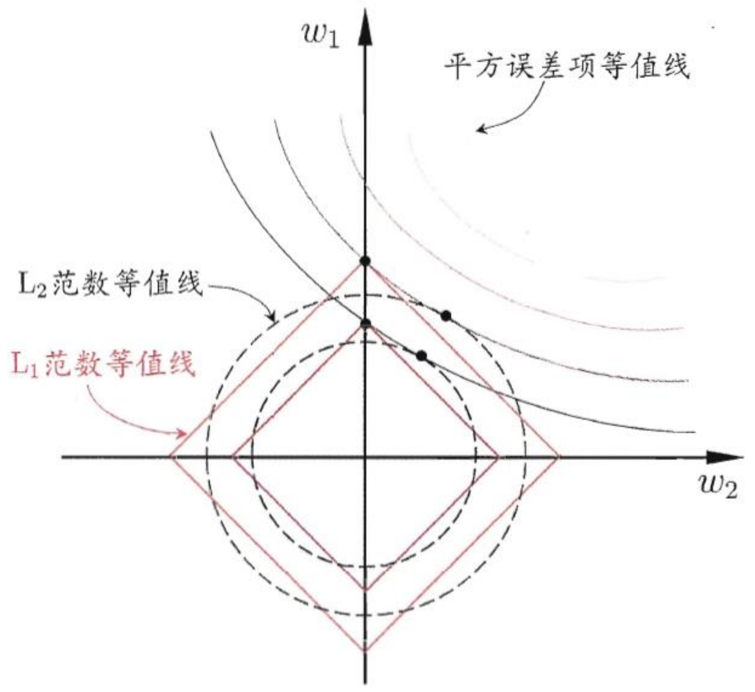


Ridge regression

- Regularization with L2 norm

$$L_{Ridge} = (y - X\theta)^2 + \lambda\|\theta\|_2^2$$

- $\lambda \to 0, \hat{\theta}_{Ridge} \to \hat{\theta}_{OLS}$
- $\lambda \to \infty, \hat{\theta} \to 0$
- As λ becomes larger, the variance decreases but the bias increases
- λ: Trade-off between bias and variance
  - Choose by cross-validation
- Ridge regression decreases the complexity of a model but does not reduce the number of variables (compared to Lasso later)



Solution of the ridge regression

- $\frac{\partial L_{Ridge}}{\partial \theta} = 2\sum_{i=1}^{N}(\theta^\top x_i - y_i)x_i + 2\lambda\theta$
- Letting the derivative be zero

$$\left(\lambda I + \sum_{i=1}^{N} x_i x_i^\top\right)\theta = \sum_{i=1}^{N} x_i y_i$$

- If we write $X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_N^\top \end{bmatrix} = \begin{bmatrix} x_1^1 & \cdots & x_1^d \\ & \vdots & \\ x_N^1 & \cdots & x_N^d \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$, then

$$(\lambda I + X^\top X)\theta = X^\top y$$

Recall the normal equation for OLS is $X^\top X\theta = X^\top y$

$$\hat{\theta}_{ridge} = (\lambda I + X^\top X)^{-1} X^\top y$$

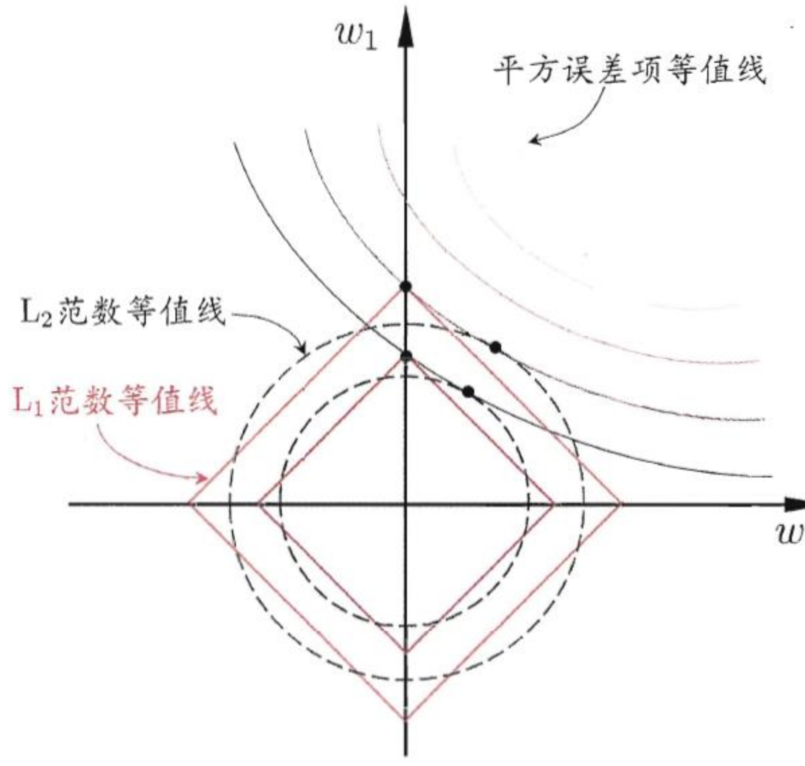
Always invertible

Lasso regression

- Lasso regression: linear regression with L1 norm

$$L_{Lasso} = (y - X\theta)^2 + \lambda||\theta||_1$$

- Lasso eliminates some features entirely and gives a subset of predictors that helps mitigate multi-collinearity and model complexity
- Predictors do not shrink towards zero significantly that they are important and thus L1 regularization allows for feature selection (sparse selection)

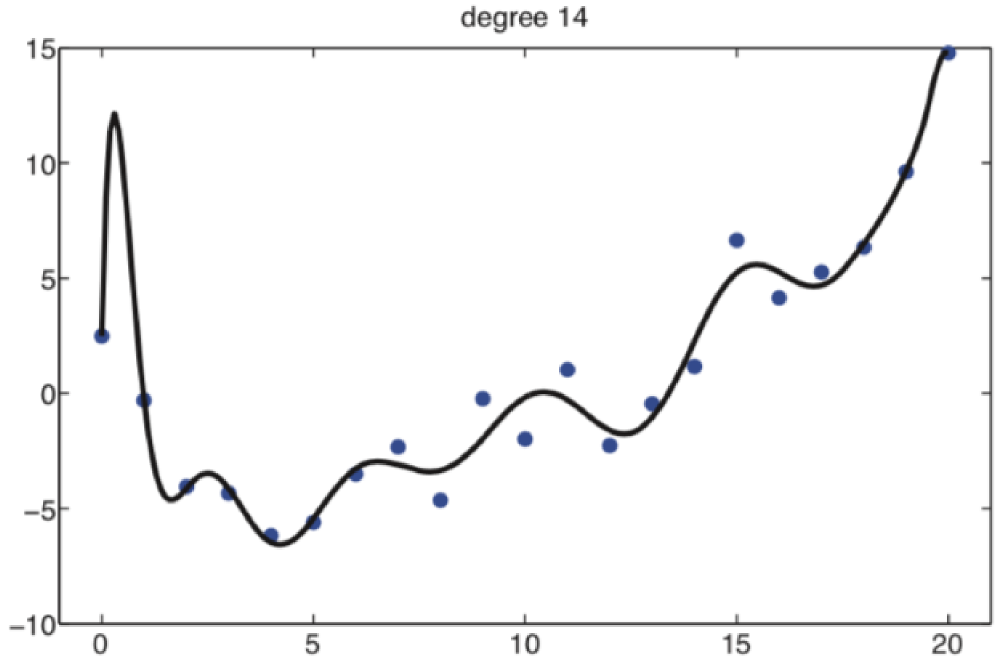


Ridge vs Lasso

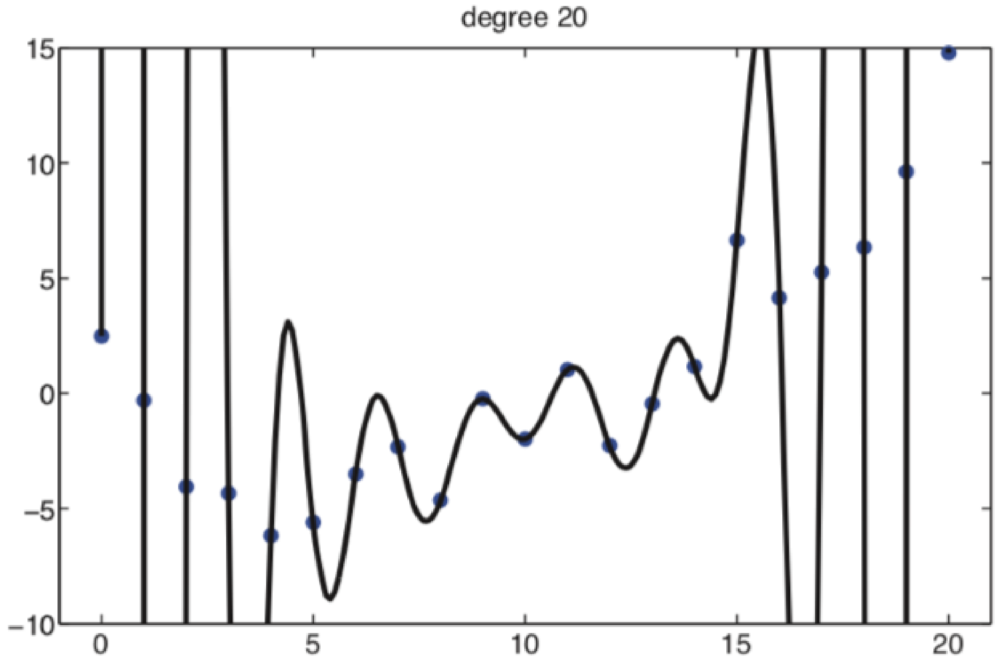
- Often neither one is overall better.
- Lasso can set some coefficients to zero, thus performing variable selection, while ridge regression cannot.
- Both methods allow to use correlated predictors, but they solve multicollinearity issue differently:
  - In ridge regression, the coefficients of correlated predictors are similar;
  - In lasso, one of the correlated predictors has a larger coefficient, while the rest are (nearly) zeroed.
- Lasso tends to do well if there are a small number of significant parameters and the others are close to zero
  - when only a few predictors actually influence the response
- Ridge works well if there are many large parameters of about the same value
  - when most predictors impact the response
- However, in practice, we don't know the true parameter values, so the previous two points are somewhat theoretical. Just run cross-validation to select the more suited model for a specific case.

Linear regression with non-linear relationships

- E.g. $\phi(x) = (1, x, x^2, \ldots, x^d)$ and $y \sim \mathcal{N}(\theta^\top \phi(x), \sigma^2)$
  - Features: Last hidden layer of Neural Networks



(a)



(b)

Figure credit: Kevin Murphy

Logistic Regression

Linear Discriminative Models

- Discriminative model
  - modeling the dependence of unobserved variables on observed ones
  - also called conditional models
  - **Deterministic**: $y = f_\theta(x)$
  - Probabilistic: $p_\theta(y|x)$

- Linear regression model

$$y = f_\theta(x) = \theta_0 + \sum_{j=1}^{d} \theta_j x_j = \theta^\top x$$

$$x = (1, x_1, x_2, \ldots, x_d)$$

Linear discriminative model

- Discriminative model
  - modeling the dependence of unobserved variables on observed ones
  - also called conditional models.
  - Deterministic: $y = f_\theta(x)$
  - **Probabilistic**: $p_\theta(y|x)$

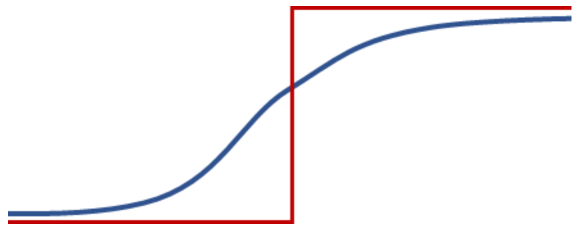
- For binary classification
  - $p_\theta(y = 1 \mid x)$
  - $p_\theta(y = 0 \mid x) = 1 - p_\theta(y = 1 \mid x)$

Classification problem

- Given:
  - A description of an instance $x \in X$
  - A fixed set of categories: $C = \{c_1, c_2, \ldots, c_m\}$
- Determine:
  - The category of $x : f(x) \in C$ where $f(x)$ is a categorization function whose domain is $\mathbb{X}$ and whose range is C
  - If the category set binary, i.e. $C = \{0, 1\}$ ({false, true}, {negative, positive}) then it is called binary classification

Cross entropy loss

- Cross entropy
  - Discrete case: $H(p,q) = -\sum_x p(x) \log q(x)$
  - Continuous case: $H(p,q) = -\int_x \; p(x) \log q(x)$
- Cross entropy loss in classification:
  - Red line p: the ground truth label distribution.
  - Blue line q: the predicted label distribution.



Entropy example for binary classification

- Cross entropy: $H(p, q) = -\sum_x p(x) \log q(x)$

- Given a data point $(x, 0)$ with prediction probability
$$q_\theta(y = 1|x) = 0.4$$
the cross entropy loss on this point is
$$L = -p(y = 0|x) \log q_\theta(y = 0|x) - p(y = 1|x) \log q_\theta(y = 1|x)$$
$$= -\log(1 - 0.4) = \log \frac{5}{3}$$

- What is the cross entropy loss for data point $(x, 1)$ with prediction probability
$$q_\theta(y = 1|x) = 0.3$$

Cross entropy loss for binary classification

- Loss function for data point (x, y) with prediction model
$$p_\theta(\cdot\,|x)$$
is
$$L(y, x, p_\theta)$$
$$= -1_{y=1} \log p_\theta(1|x) - 1_{y=0} \log p_\theta(0|x)$$
$$= -y \log p_\theta(1|x) - (1 - \mathrm{y}) \log\left(1 - p_\theta(1|x)\right)$$

Binary classification: linear and logistic

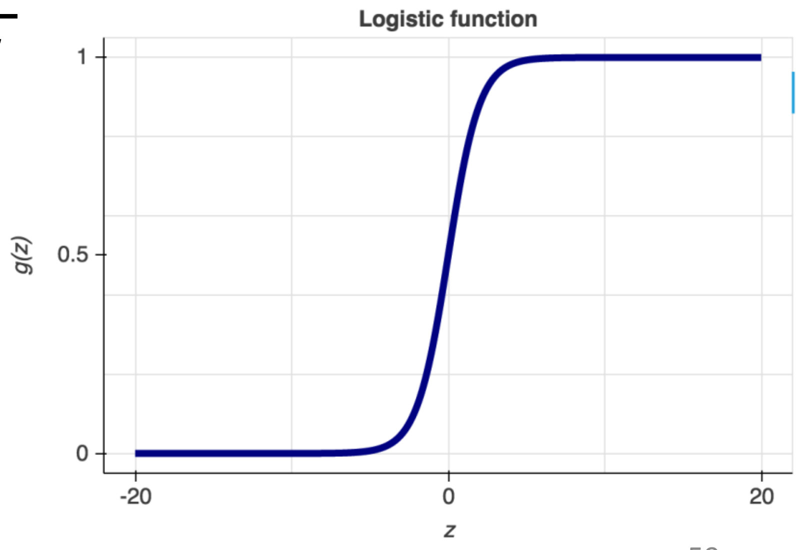
- Linear regression:
  - Target is predicted by $h_\theta(x) = \theta^\top x$

- Logistic regression
  - Target is predicted by $h_\theta(x) = \sigma(\theta^\top x) = \frac{1}{1+ e^{-\theta^\top x}}$

    where

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

    is the logistic function or the sigmoid function



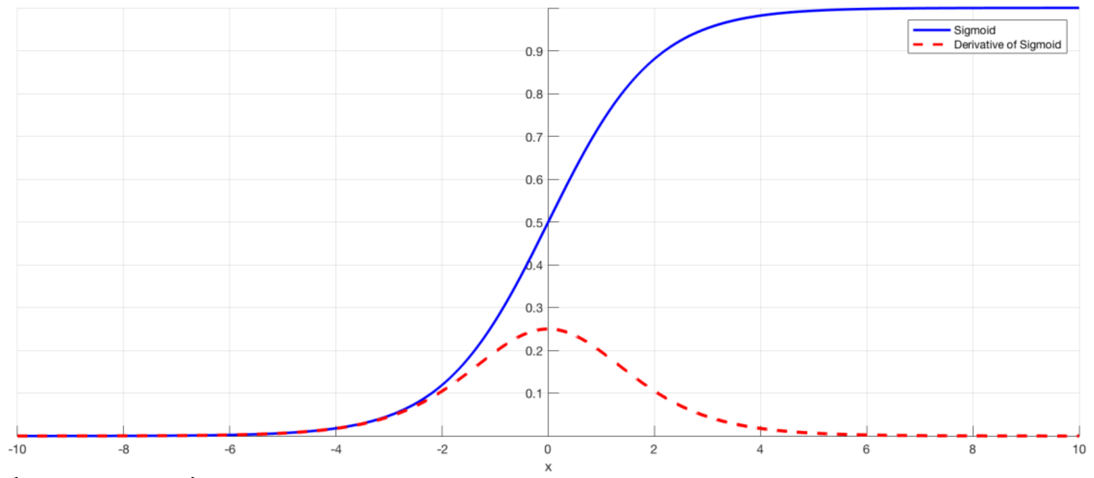
Properties for the sigmoid function

- $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Bounded in $(0,1)$
  - $\sigma(z) \to 1$ when $z \to \infty$
  - $\sigma(z) \to 0$ when $z \to -\infty$
- $\underline{\sigma'(z)} = \frac{d}{dz}\frac{1}{1+e^{-z}} = -(1+e^{-z})^{-2} \cdot (-e^{-z})$

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

$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\underline{= \sigma(z)\big(1-\sigma(z)\big)}$$



Logistic regression

- Binary classification

$$p_\theta(y=1|x) = \sigma(\theta^\top x) = \frac{1}{1+e^{-\theta^\top x}}$$

$$p_\theta(y=0|x) = \frac{e^{-\theta^\top x}}{1+e^{-\theta^\top x}}$$

- Cross entropy loss function

is also convex in θ

$$\mathcal{L}(y, x, p_\theta) = -y \log \sigma(\theta^\top x) - (1-y)\log(1-\sigma(\theta^\top x))$$

- Gradient

$$\frac{\partial \mathcal{L}(y,x,p_\theta)}{\partial \theta} = -y\frac{1}{\sigma(\theta^\top x)}\sigma(z)(1-\sigma(z))x - (1-y)\frac{-1}{1-\sigma(\theta^\top x)}\sigma(z)(1-\sigma(z))x$$

$$= (\sigma(\theta^\top x) - y)x$$

$$\theta \leftarrow \theta + \eta(y - \sigma(\theta^\top x))x$$

$$\boxed{\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))}$$

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} - \eta\frac{\partial \mathcal{L}(\theta)}{\partial \theta}$$

Label decision

- Logistic regression provides the probability

$$p_\theta(y=1|x) = \sigma(\theta^\top x) = \frac{1}{1+e^{-\theta^\top x}}$$

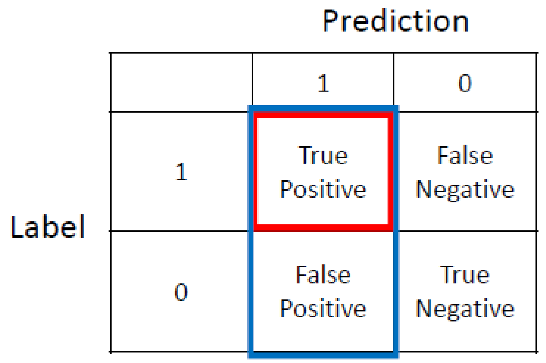
$$p_\theta(y=0|x) = \frac{e^{-\theta^\top x}}{1+e^{-\theta^\top x}}$$

- The final label of an instance is decided by setting a threshold h

$$\hat{y} = \begin{cases} 1, & p_\theta(y=1|x) > h \\ 0, & \text{otherwise} \end{cases}$$

Confusion matrix

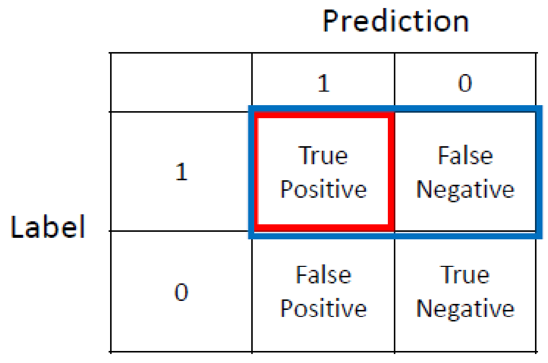
- Remember what we have learned about the confusion matrix



		Prediction	
		1	0
Label	1	True Positive	False Negative
	0	False Positive	True Negative

- **Precision**: the ratio of true class 1 cases in those with prediction 1

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$



		Prediction	
		1	0
Label	1	True Positive	False Negative
	0	False Positive	True Negative

- **Recall**: the ratio of cases with prediction 1 in all true class 1 cases

$$\text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

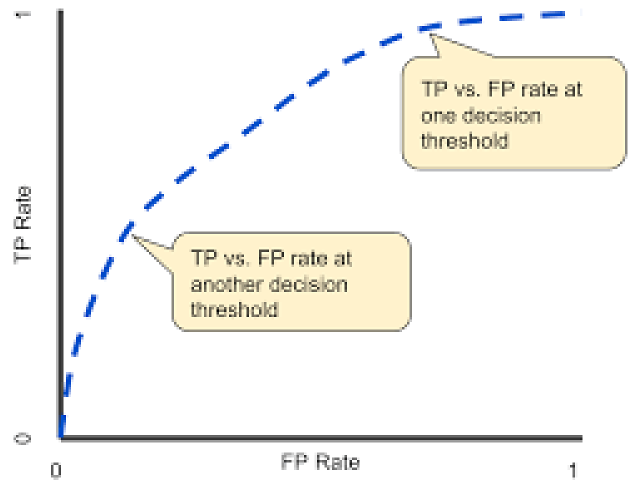
$$\text{F1} = \frac{2 \times \text{Prec} \times \text{Recall}}{\text{Prec} + \text{Rec}}$$

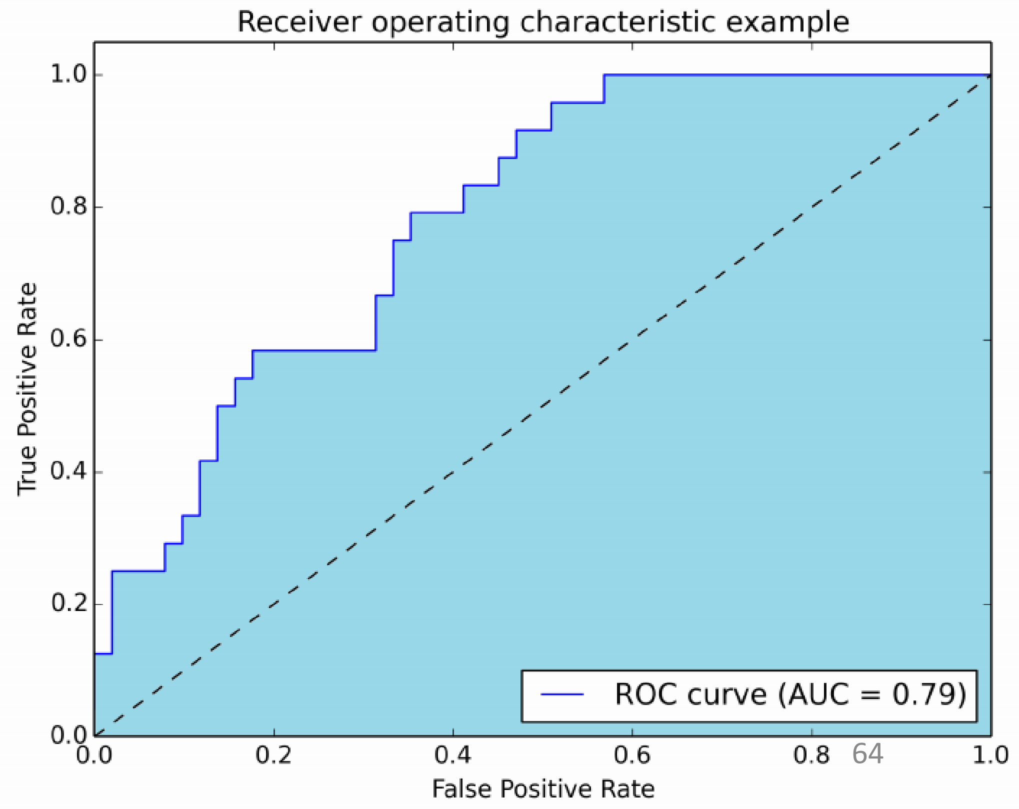
- These are the basic metrics to measure the classifier

Area Under ROC Curve (AUC)

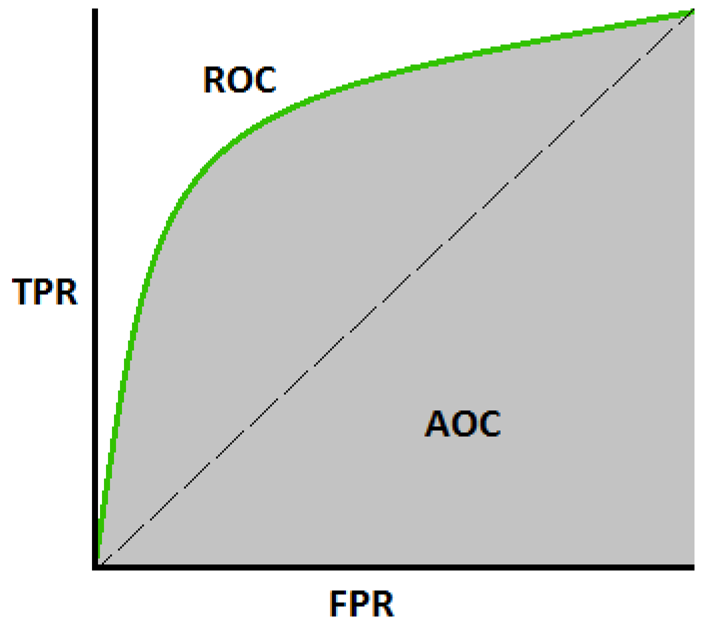
- A performance measurement for classification problem at various thresholds settings
- Tells how much the model is capable of distinguishing between classes
- The higher, the better
- Receiver Operating Characteristic (ROC) Curve
  - TPR against FPR
  - TPR/Recall/Sensitivity $=\frac{\text{TP}}{\text{TP+FN}}$

    FPR=1-Specificity= $\frac{\text{FP}}{\text{TN+FP}}$





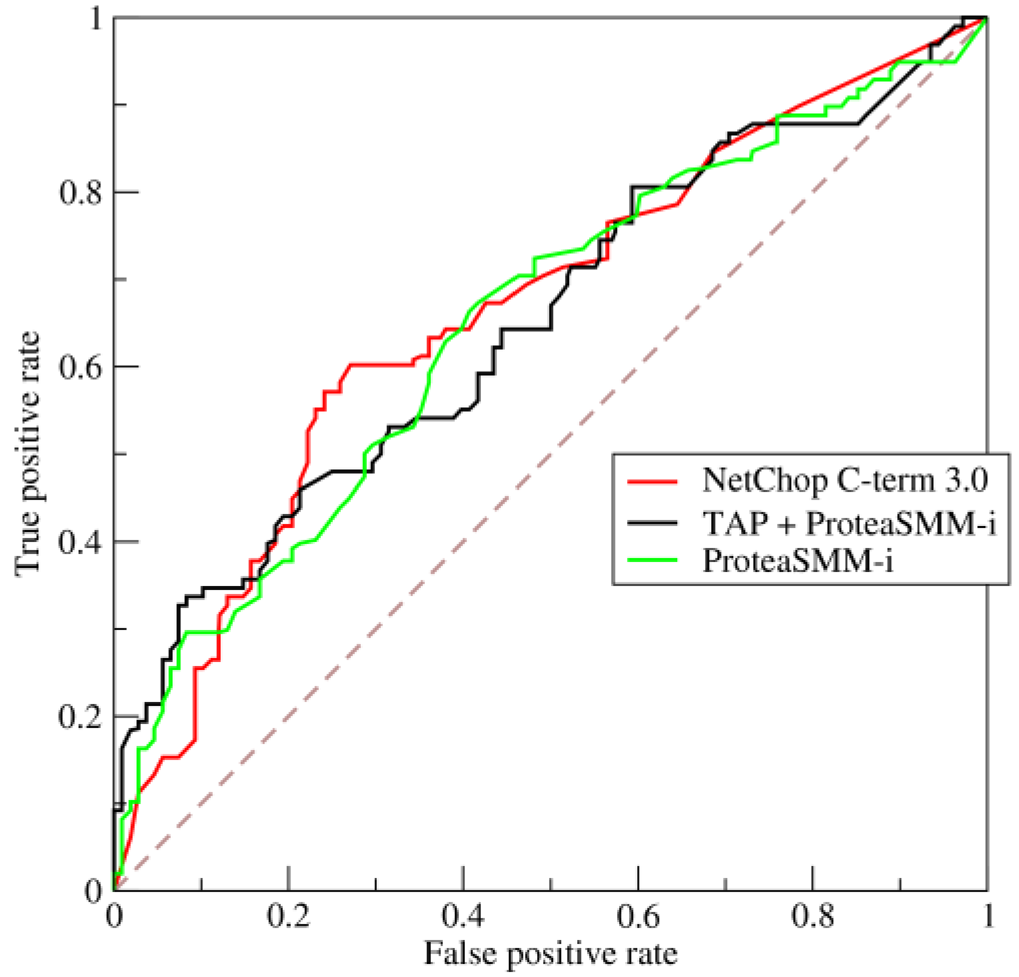
AUC (cont.)



TPR: true positive rate
FPR: false positive rate

- It’s the relationship between TPR and FPR when the threshold is changed from 0 to 1
- In the top right corner, threshold is 0, and every thing is predicted to be positive, so both TPR and FPR is 1
- In the bottom left corner, threshold is 1, and every thing is predicted to be negative, so both TPR and FPR is 0
- The size of the area under this curve (AUC) is an important metric to binary classifier
- Perfect classifier get AUC=1 and random classifier get AUC = 0.5

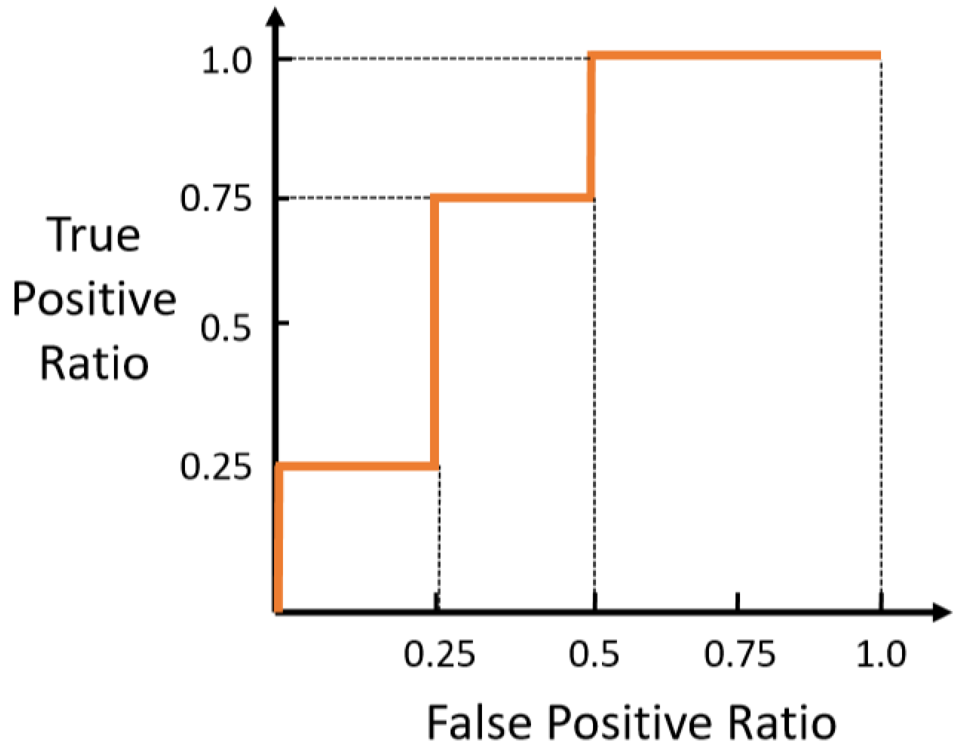
AUC (cont.)



- It considers all possible thresholds.
- Various thresholds result in different true/false positive rates.
- As you decrease the threshold, you get more true positives, but also more false positives.

- From a random classifier you can expect as many true positives as false positives. That's the dashed line on the plot. AUC score for the case is 0.5. A score for a perfect classifier would be 1. Most often you get something in between.

AUC example



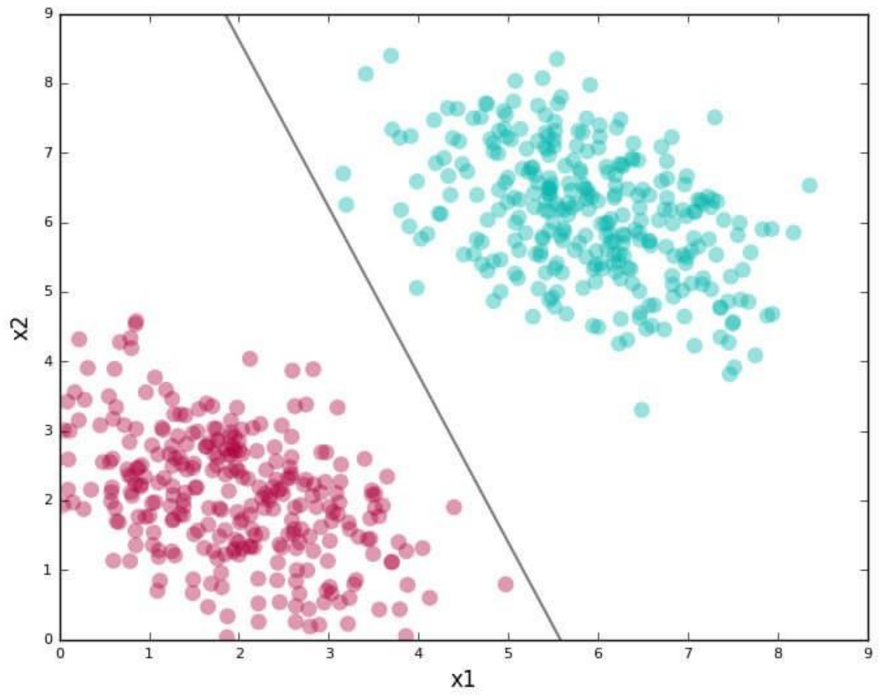
AUC = 0.75

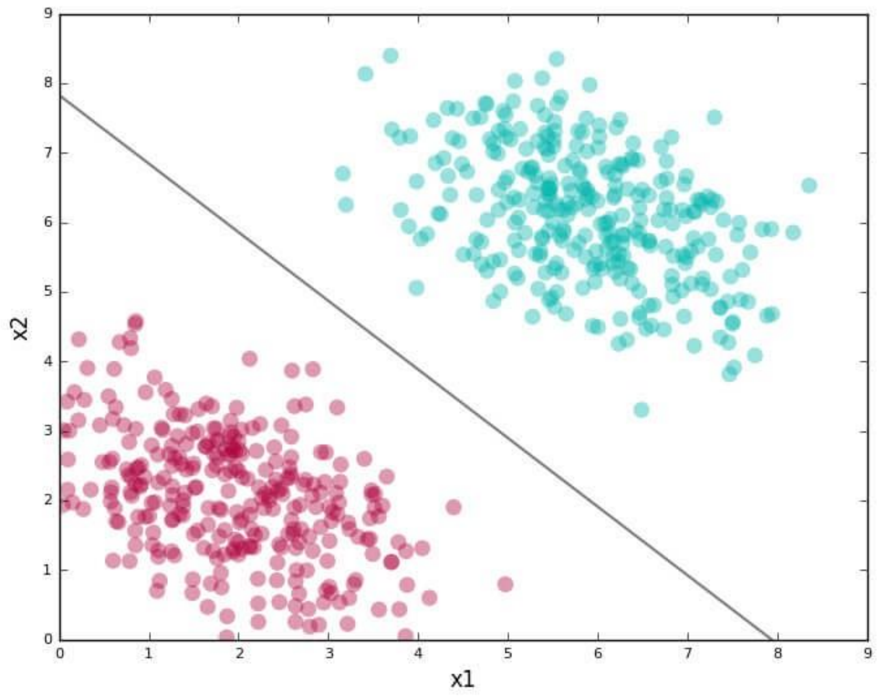
Prediction	Label
0.91	1
0.85	0
0.77	1
0.72	1
0.61	0
0.48	1
0.42	0
0.33	0

Support Vector Machine

Example

- Both the two solutions can separate the data perfectly, but we prefer the one on the right, why?

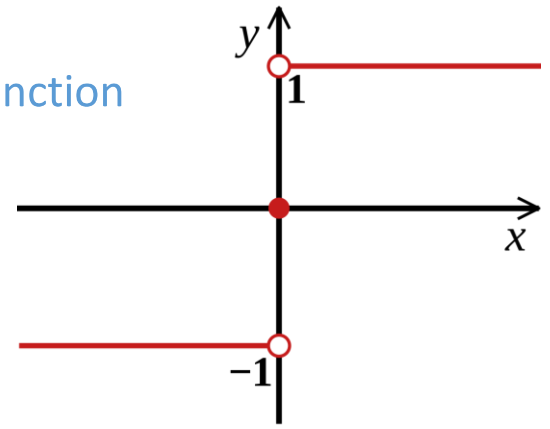




Notations for SVM

- Feature vector x
- Class label $y \in \{-1, 1\}$
  - Instead of $\{0,1\}$
- Parameters
  - Intercept b
    - We also drop the convention we had previously of letting $x_0 = 1$ be an extra coordinate in the input feature vector
  - Feature weight vector w
- Label prediction
  - $h_{w,b}(x) = g(w^\top x + b)$
  - $g(z) = \begin{cases} +1 & z \geq 0 \\ -1 & \text{otherwise} \end{cases}$
  - Directly output the label
    - Without estimating probability first (compared with logistic regression)

Sign function



Scores of logistic regression

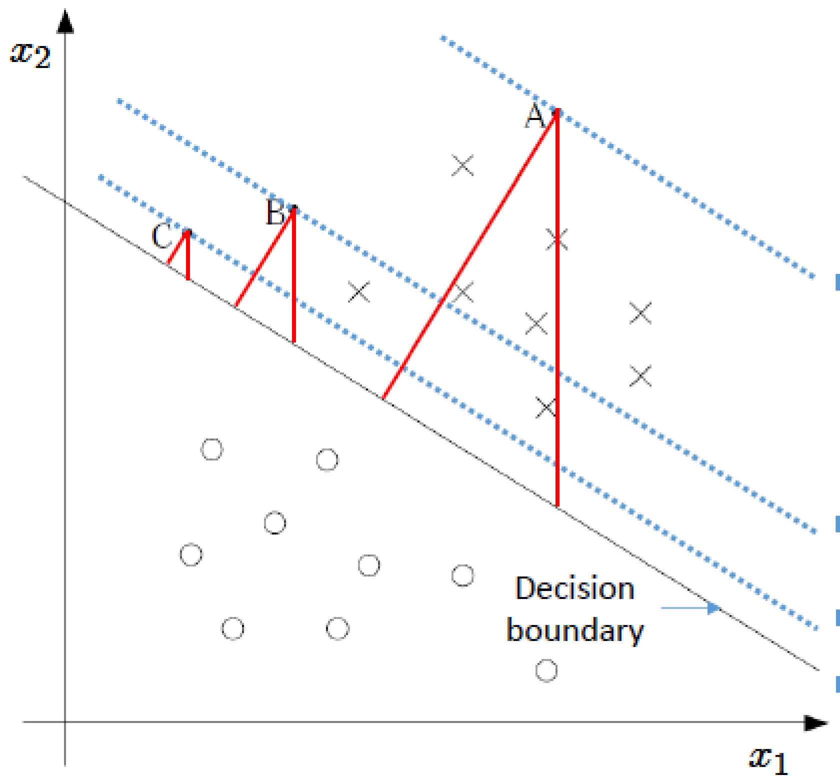
- Let $s(x) = \theta_0 + \theta_1 x_2 + \theta_2 x_2$, so the probability in logistic regression is defined as $p_\theta(y = 1|x) = \frac{1}{1+e^{-s(x)}}$

- Positive prediction means positive scores
- Negative prediction means negative scores

- The absolute value of the score $s(x)$ is proportional to the distance x to the decision boundary $\theta_0 + \theta_1 x_2 + \theta_2 x_2 = 0$

Illustration of logistic regression

- The higher score, the larger distance to the decision boundary, the higher confidence. E.g.



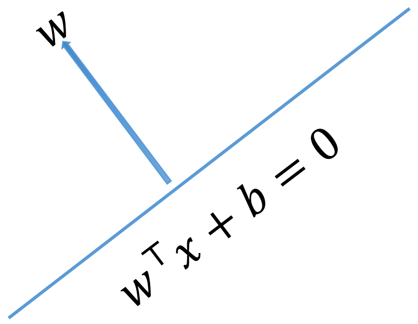
$$s(x^A) = \theta_0 + \theta_1 x_1^A + \theta_2 x_2^A = 7$$

$$s(x^B) = \theta_0 + \theta_1 x_1^B + \theta_2 x_2^B = 3$$

$$s(x^C) = \theta_0 + \theta_1 x_1^C + \theta_2 x_2^C = 1$$

$$s(x) = 0$$

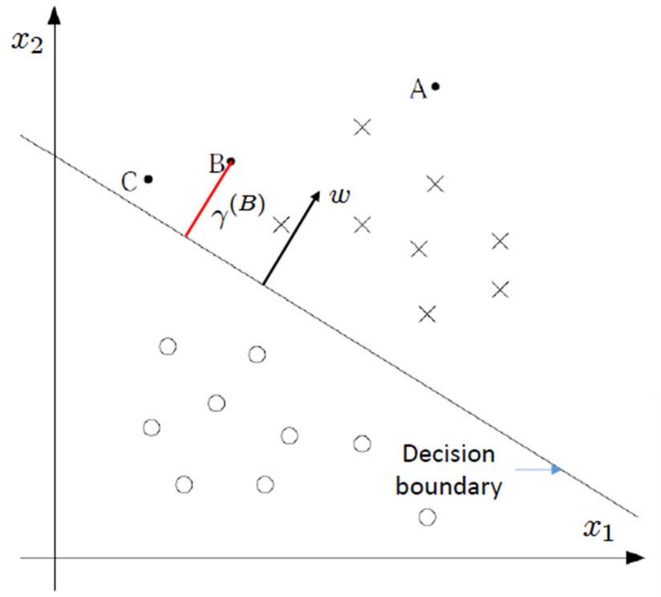
Geometric margin



- w vector is orthogonal to the decision boundary
- Geometric margin is the distance of the point to the decision boundary
    - For positive prediction points
    - $x - \gamma \frac{w}{\|w\|}$ lies on the decision boundary
    - $w^\top \left(x - \gamma \frac{w}{\|w\|}\right) + b = 0$
    - Solve it, get

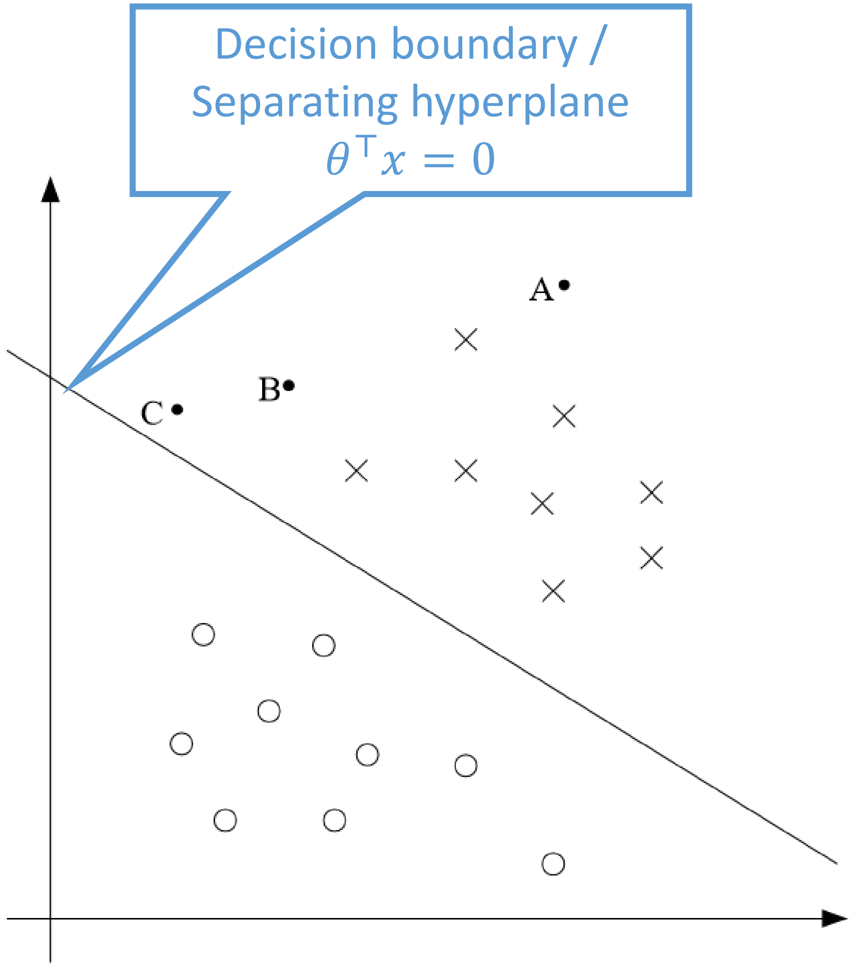
$$\gamma = \frac{w^\top x + b}{\|w\|}$$

    - In general, $\gamma = y(w^\top x + b)$ with $\|w\| = 1$



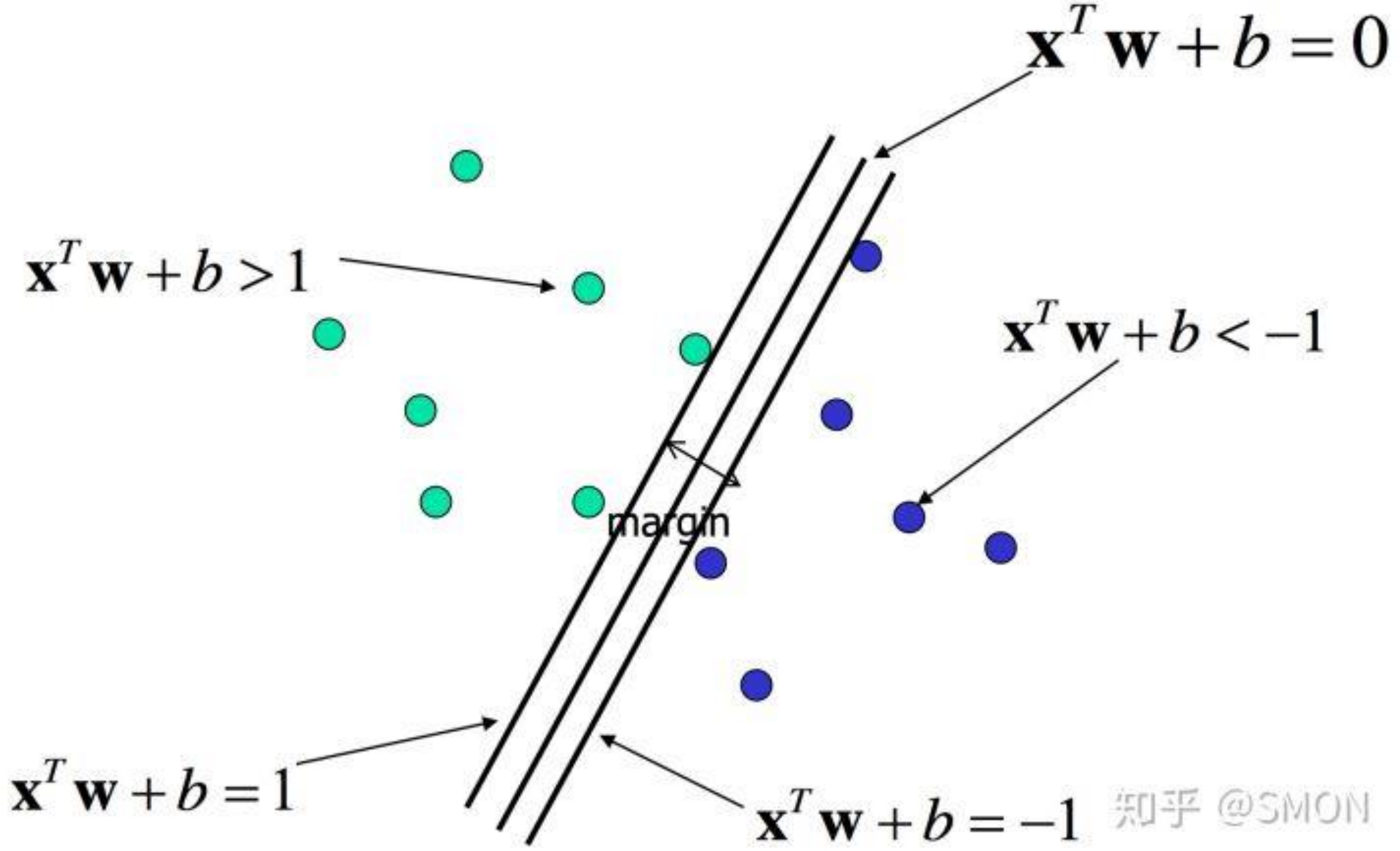
Intuition

- Positive when
  $$p_\theta(y=1|x) = h_\theta(x) = \sigma(\theta^\top x) \geq 0.5$$
  or
  $$\theta^\top x \geq 0$$
- Point A
  - Far from decision boundary
  - More confident to predict the label 1
- Point C
  - Near decision boundary
  - A small change to the decision boundary could cause prediction to be $y = 0$



Hyperplane and margin

- Idea of using $y \in \{-1, 1\}$


$\mathbf{x}^T\mathbf{w} + b = 0$
$\mathbf{x}^T\mathbf{w} + b > 1$
$\mathbf{x}^T\mathbf{w} + b < -1$
margin
$\mathbf{x}^T\mathbf{w} + b = 1$
$\mathbf{x}^T\mathbf{w} + b = -1$
知乎 @SMON

Objective of an SVM

- Given a training set
$$S = \{(x_i, y_i)\}, i = 1, \dots, N$$
margin is the smallest geometric margin
$$\gamma = \min_{i=1,\dots,n} \gamma^i$$
- Objective is to maximize the margin, or equivalently
$$\begin{aligned} &\min_{w,b} \ \frac{1}{2}\|w\|^2 \\ &s.t. \quad y^i\left(w^\top x^i + b\right) \geq 1, \qquad i = 1, \dots, N \end{aligned}$$

Karush-Kuhn-Tucker (KKT) Conditions

- Suppose
  - f and g_i's are convex
  - h_i's are affine
  - g_i's are all strictly feasible
    - There exists w such that $g_i(w) < 0$ for all i
- Then there must exist w^*, α^*, β^*
  - w^* is the solution of the primal problem
  - α^*, β^* are the solution of the dual problem
  - And the values of the two problems are equal

$$p^* = d^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$$

- w^*, α^*, β^* satisfy the KKT conditions:

$$\frac{\partial}{\partial w_i}\mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \; i = 1, \ldots, n$$
$$\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*, \alpha^*, \beta^*) = 0, \; i = 1, \ldots, l$$

KKT dual complementarity condition
$$\boxed{\xrightarrow{} \alpha_i^* g_i(w^*) = 0, \; i = 1, \ldots, k}$$
$$g_i(w^*) \leq 0, \; i = 1, \ldots, k$$
$$\alpha^* \geq 0, \; i = 1, \ldots, k$$

  - If $\alpha_i^* > 0$, then $g_i(w^*) = 0$
- The converse is also true
  - If some w, a, b satisfy the KKT conditions, then it is also a solution to the primal and dual problems
  - More details can be found in Boyd's book "Convex optimization"

Rewrite the SVM objective

- The objective of SVM is

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2 \\ s.t. \quad & y^i\left(w^\top x^i + b\right) \geq 1, \qquad i = 1, \ldots, N \end{aligned}$$

- Rewrite the constraints as

$$g_i(w) = -y^i\left(w^\top x^i + b\right) + 1 \leq 0$$

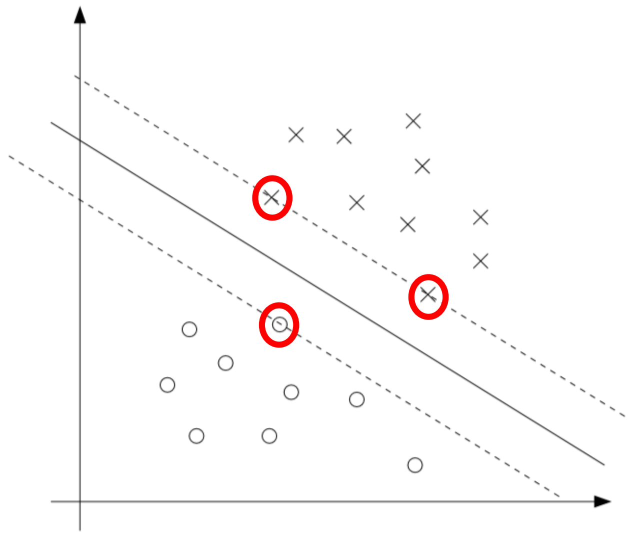
- It is equivalent to solve the dual problem
- Note that from the KKT dual complementarity condition, $\alpha_i > 0$ is only possible for training samples with $g_i(w) = 0$

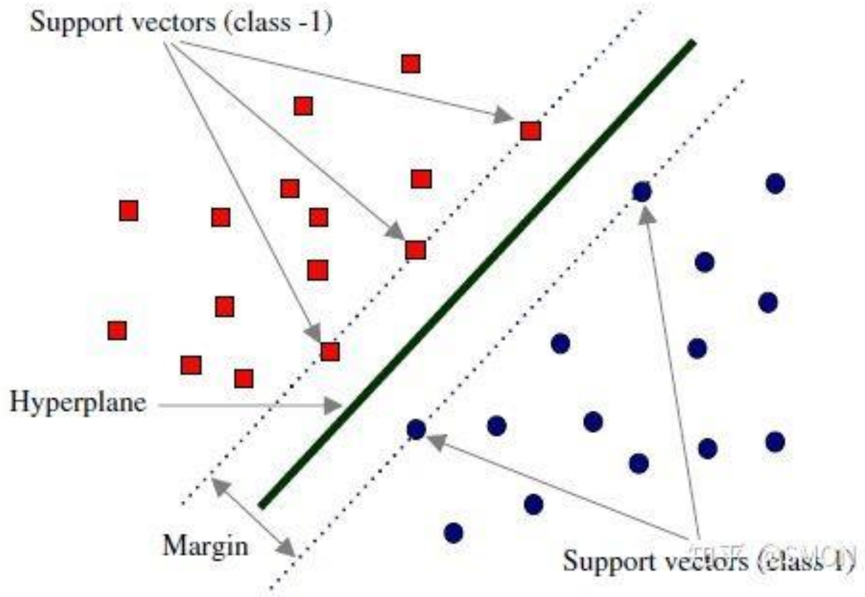
Support vectors

- The points with smallest margins

- $g_i(w) = 0$
  - $-y^i(w^\top x^i + b) + 1 = 0$
  - Positive support vectors
    - $w^\top x + b = 1$
  - Negative support vectors
    - $w^\top x + b = -1$

- Only support vectors decide the decision boundary
  - Moving or deleting non-support points doesn't change the decision boundary





Lagrangian of SVM

- SVM objective:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$
$$g_i(w) = -y^i\left(w^\top x^i + b\right) + 1 \leq 0, i = 1, \dots, N$$

- Lagrangian

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \alpha_i\left[1 - y^i\left(w^\top x^i + b\right)\right]$$

- It is equivalent to solve the dual problem

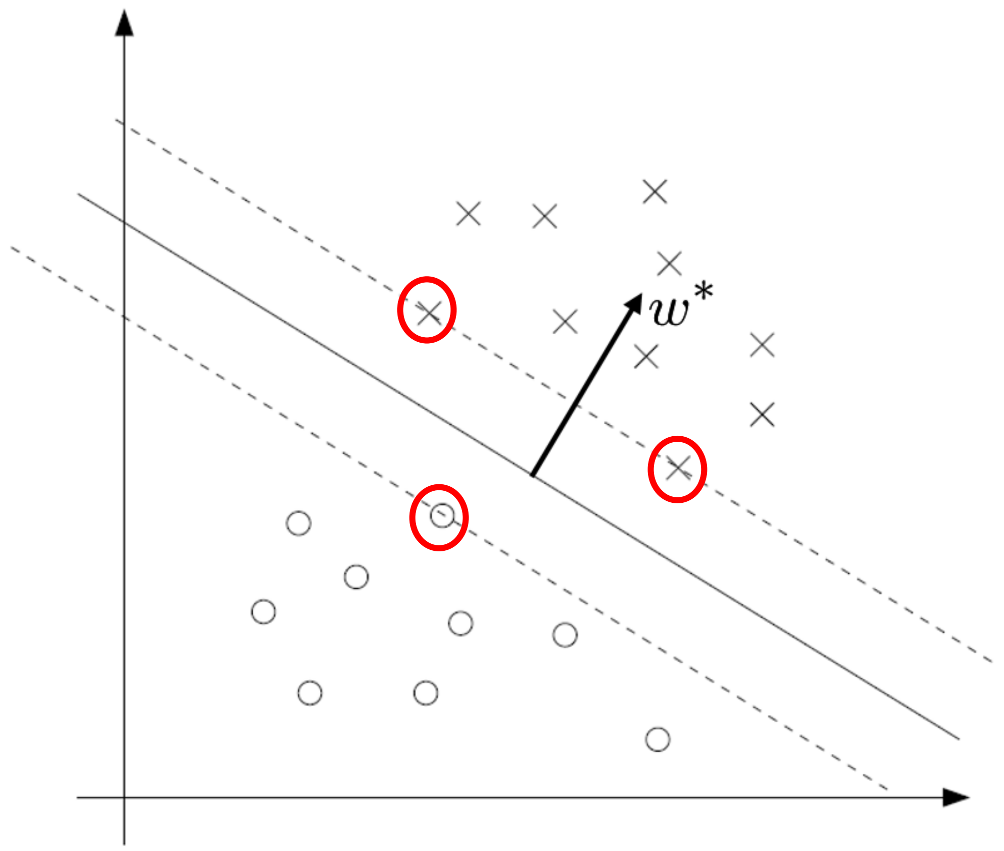
$$\max_{\alpha \geq 0} \min_{w,b} L(w, b, \alpha)$$

Solve w^* and b^*

- With α^*

$$w = \sum_{i=1}^{N} \alpha_i y^i x^i$$

- $\alpha_i > 0$ only holds on support vectors



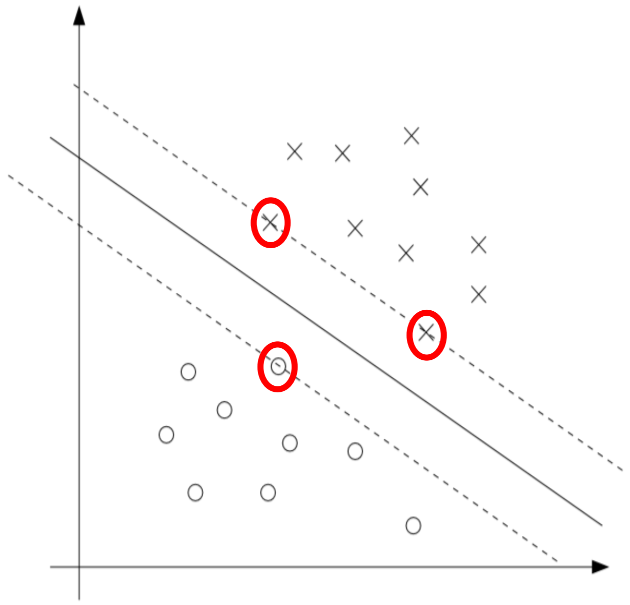
- Then

$$b^* = -\frac{\max_{i:y^{(i)}=-1} {w^*}^\top x^{(i)} + \min_{i:y^{(i)}=1} {w^*}^\top x^{(i)}}{2}$$

Check it!

Predicting values

- $w^\top x + b = \left(\sum_{i=1}^{N} \alpha_i y^i x^i\right)^\top x + b$
$$= \sum_{i=1}^{N} \alpha_i y^i \langle x^i, x \rangle + b$$

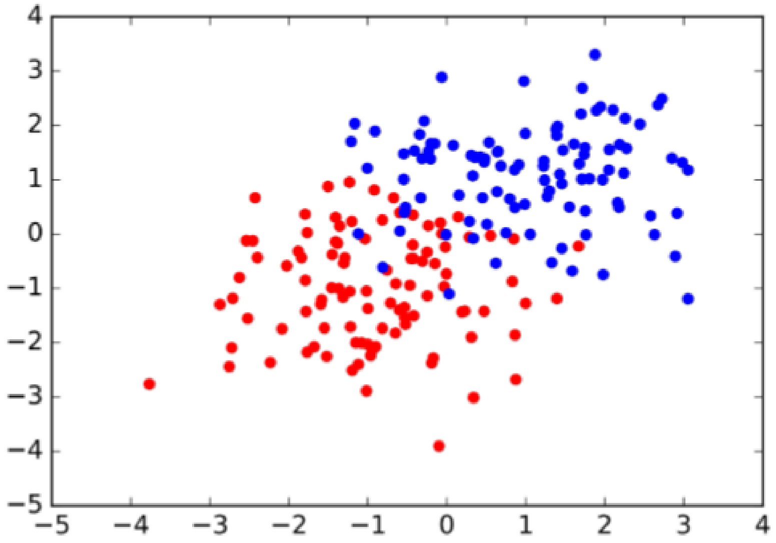


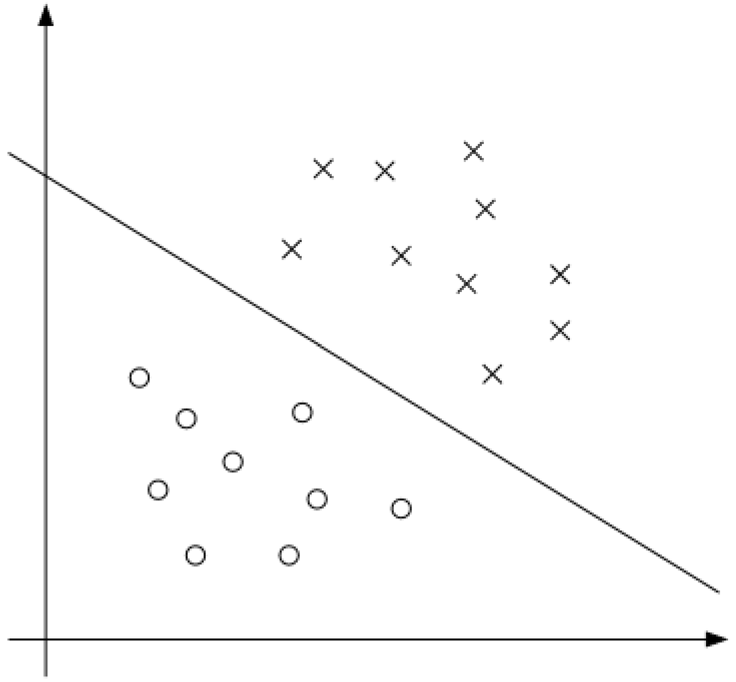
- Only need to calculate the inner product of x with support vectors

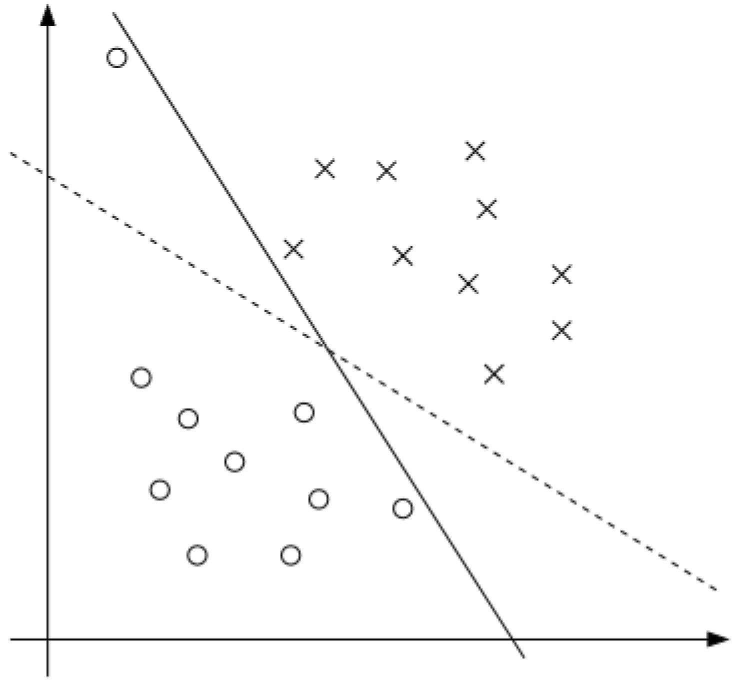
- Prediction is
$$y = \mathrm{Sign}(w^\top x + b)$$

Regularization motivation

- SVM assumes data is linearly separable
  - But some data is linearly non-separable
  - SVM is susceptible to outliers







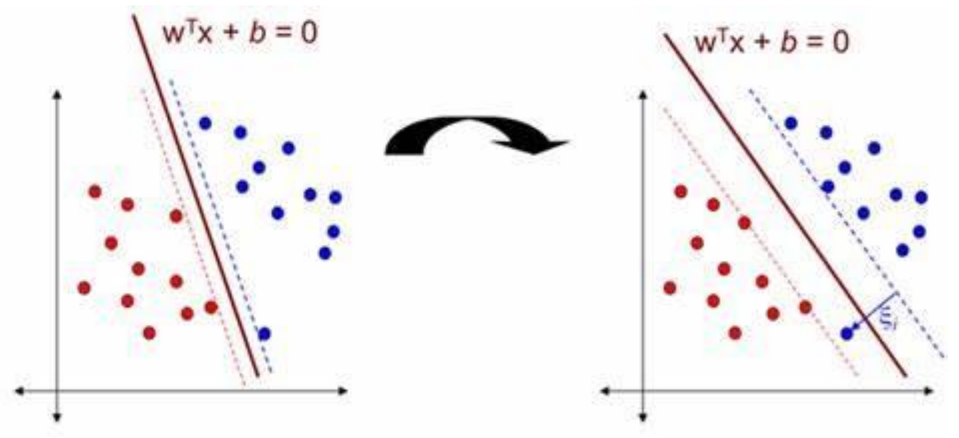
Solution – Soft margin

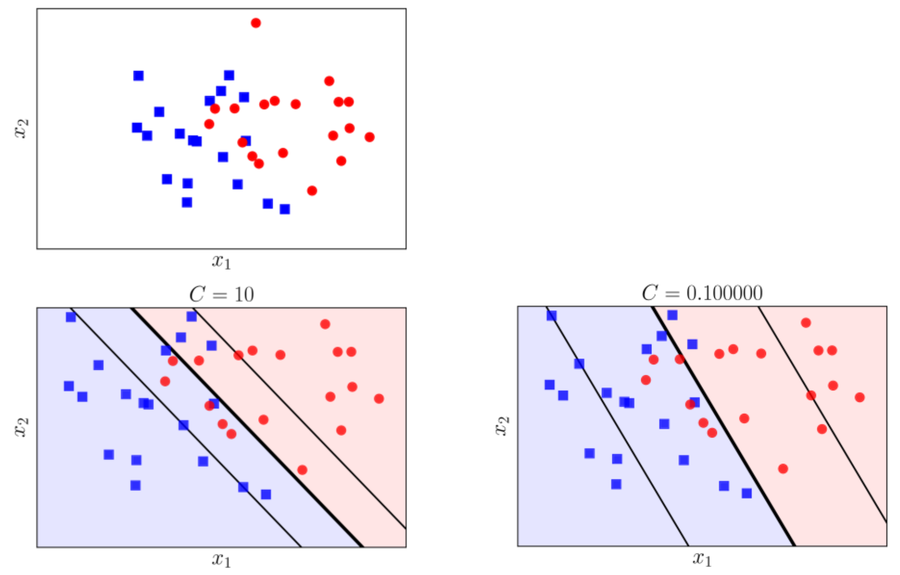
- To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers
- Add slack variables

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i$$

L^1 regularization

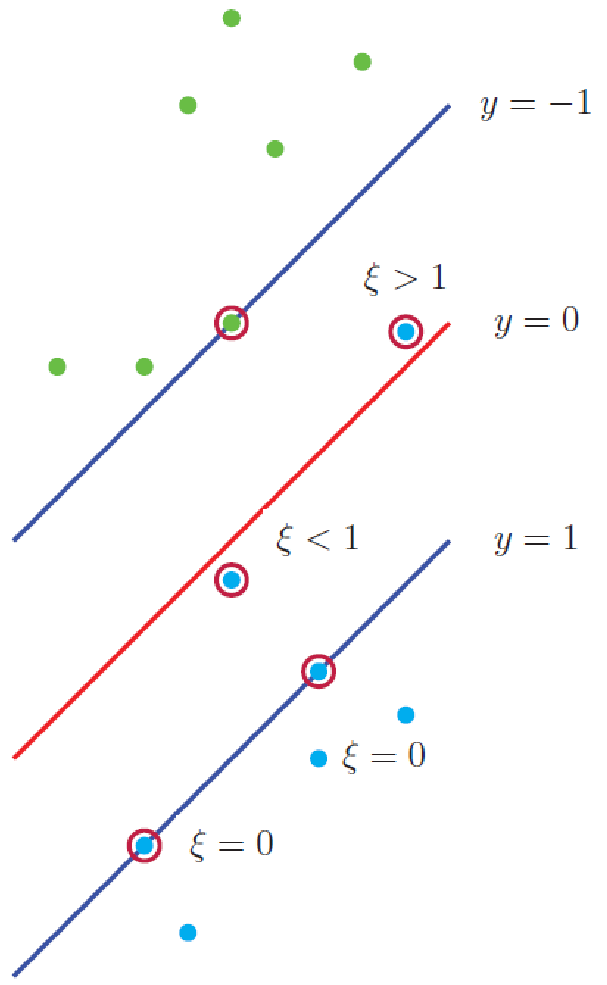
$$s.t. \quad y^i\left(w^\top x^i + b\right) \geq 1 - \xi_i, \qquad i = 1, \dots, N$$
$$\xi_i \geq 0, \qquad i = 1, \dots, N$$





Example

- Correctly classified points beyond the support line with $\xi = 0$
- Correctly classified points on the support line (support vectors) with $\xi = 0$
- Correctly classified points inside the margin with $0 < \xi < 1$
- The misclassified points inside the margin with slack $1 < \xi < 2$
- The misclassified points outside the margin with slack $\xi > 2$



The Lagrangian

- Lagrangian

$$L(w, b, \xi, \alpha, r) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i + \sum_{i=1}^{N} \alpha_i\left[1 - \xi_i - y^i\left(w^\top x^i + b\right)\right] - \sum_{i=1}^{N} r_i \xi_i$$

- And the dual problem is

$$\max_{\alpha \geq 0, r \geq 0} \min_{w, b, \xi} L(w, b, \xi, \alpha, r)$$

Revisit the regularized objective

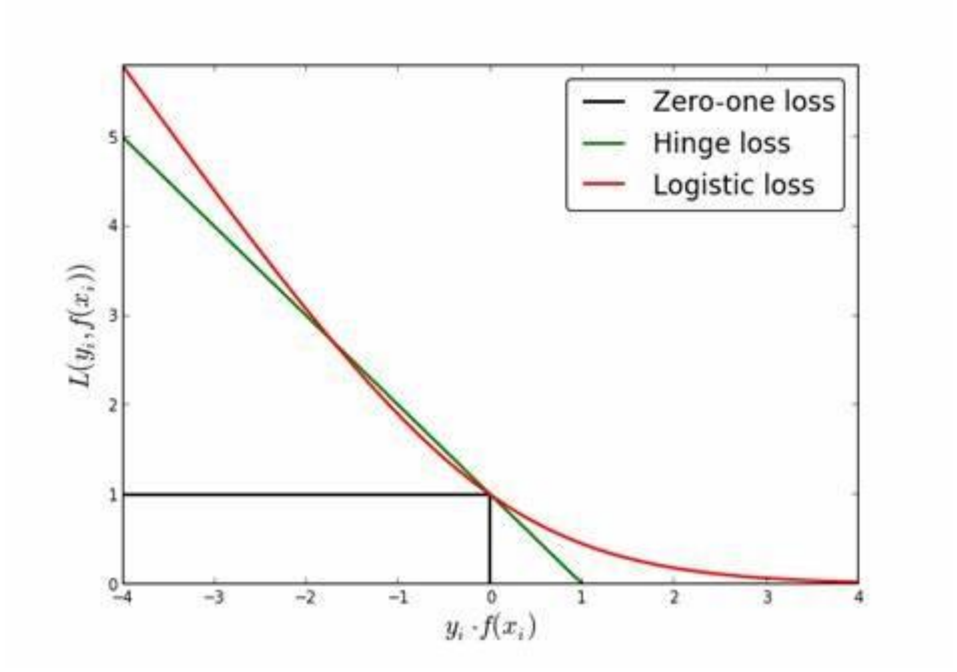
- $$\min_{w,b} \ \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} \xi_i$$
  $$\text{s.t.} \quad y^i\left(w^\top x^i + b\right) \geq 1 - \xi_i, \qquad i = 1, \dots, N$$
  $$\xi_i \geq 0, \qquad i = 1, \dots, N$$

- $$\min_{w,b} \ \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} \underbrace{\max\{0, 1 - y^i\left(w^\top x^i + b\right)\}}_{\text{SVM hinge loss}}$$

SVM hinge loss vs. logistic loss

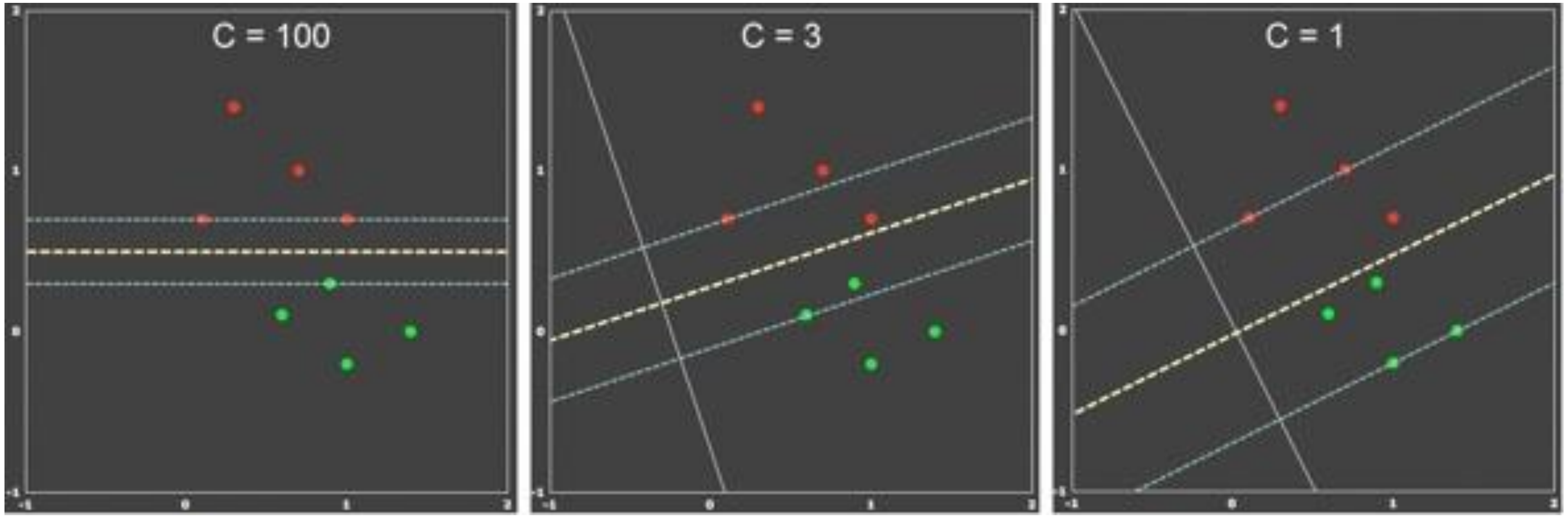
- SVM hinge loss
  - $L(y, f(x)) = \max\{0, 1 - yf(x)\}$
- For $y = 1$

- Logistic loss
  - $L(y, f(x)) = -y \log \sigma(f(x)) - (1 - y) \log\left(1 - \sigma(f(x))\right)$



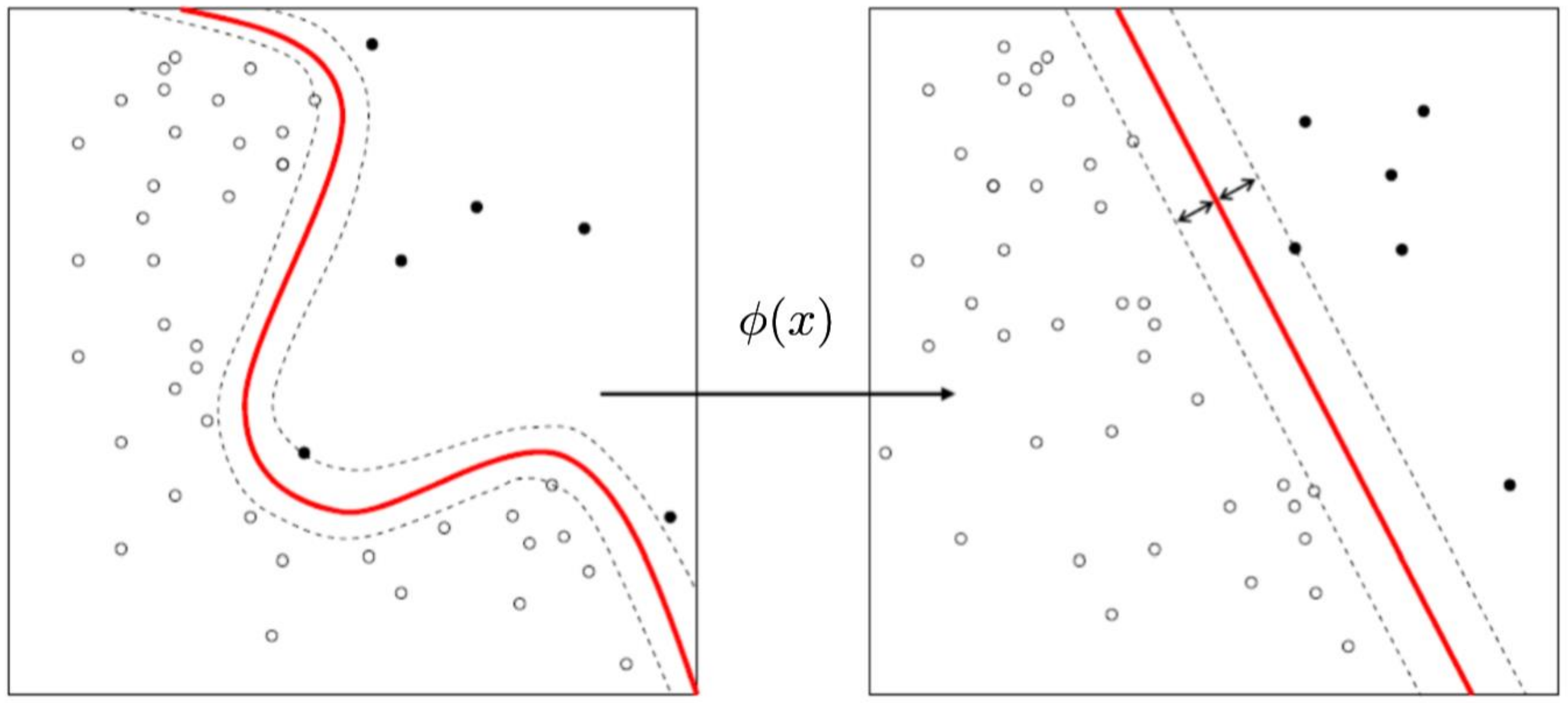
The effect of penalty coefficient

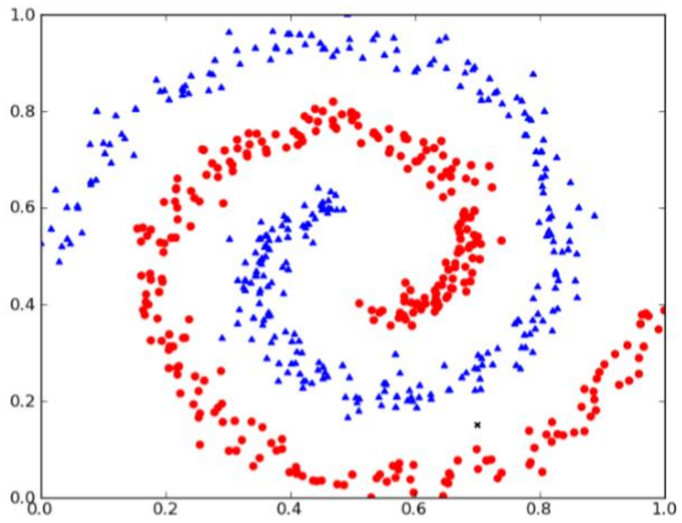
- $\min_{w,b} \; \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} \xi_i$
- Large C will result in narrow margin

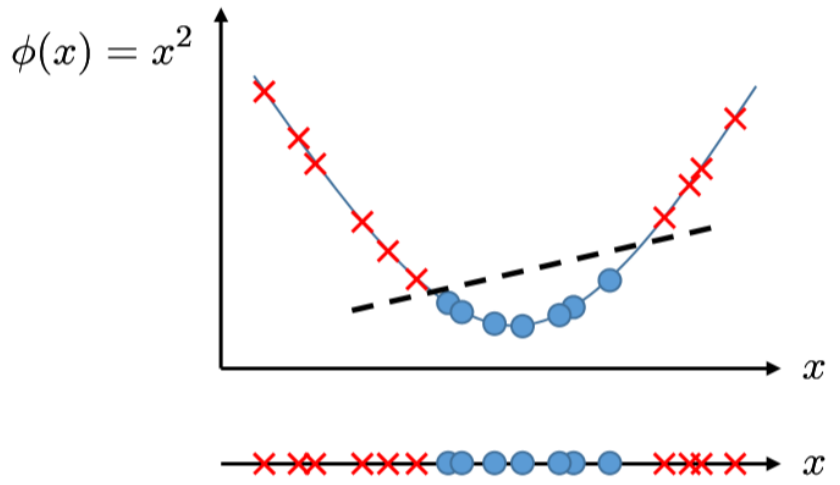


Non-linearly separable case

- Feature mapping







From inner product to kernel function

- SVM

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^i y^j {x^j}^\top x^i$$

- Kernel

$$W(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^i y^j K(x^i, x^j)$$

  - $K(x^i, x^j) = \langle \Phi(x^i), \Phi(x^j) \rangle$
- Kernel trick:
  - For many cases, only $K(x^i, x^j)$ are needed, so we can only define these K_{ij} without explicitly defining Φ
  - For prediction, only need $K(x^i, x)$ on support vectors

Property

- If K is a valid kernel (that is, is defined by some feature mapping Φ), then the kernel matrix $K = \left(K_{ij}\right)_{ij} \in \mathbb{R}^{N\times N}$ is symmetric positive semi-definite
- Symmetric
  - $K_{ij} = K\left(x^i, x^j\right) = \left\langle \Phi\left(x^i\right), \Phi\left(x^j\right)\right\rangle = \left\langle \Phi\left(x^j\right), \Phi\left(x^i\right)\right\rangle = K\left(x^j, x^i\right) = K_{ji}$

- Positive semi-definite

$$
\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&\geq 0.
\end{aligned}
$$

Examples on kernels

- Gaussian kernel

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

  - Radial basis function (RBF) kernel
  - What is the feature mapping Φ? (Hint: by using Taylor series)
- Simple polynomial kernel $K(x, z) = (x^\top z)^d$
- Cosine similarity kernel $K(x, z) = \frac{x^\top z}{\|x\| \cdot \|z\|}$
- Sigmoid kernel

$$K(x, z) = \tanh(\alpha x^\top z + c)$$

$$\tanh(b) = \frac{1 - e^{-2b}}{1 + e^{-2b}}$$

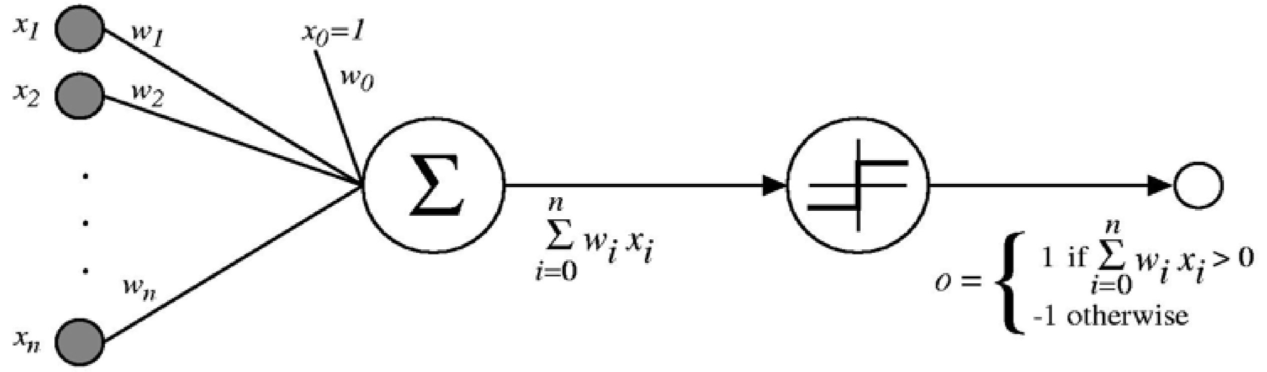
Pros and cons of SVM

- Advantages:
  - The solution, which is based on convex optimization, is globally optimal
  - Can be applied to both linear/non-linear classification problems
  - Can be applied to high-dimensional data
    - since the complexity of the data set mainly depends on the support vectors
  - Complete theoretical guarantee
    - Compared with deep learning
- Disadvantages:
  - The number of parameters α is number of samples, thus hard to apply to large-scale problems
    - SMO can ease the problem a bit
  - Mainly applies to binary classification problems
    - For multi-classification problems, can solve several binary classification problems, but might face the problem of imbalanced data

Neural Networks

Perceptron

- Inspired by the biological neuron among humans and animals, researchers build a simple model called Perceptron



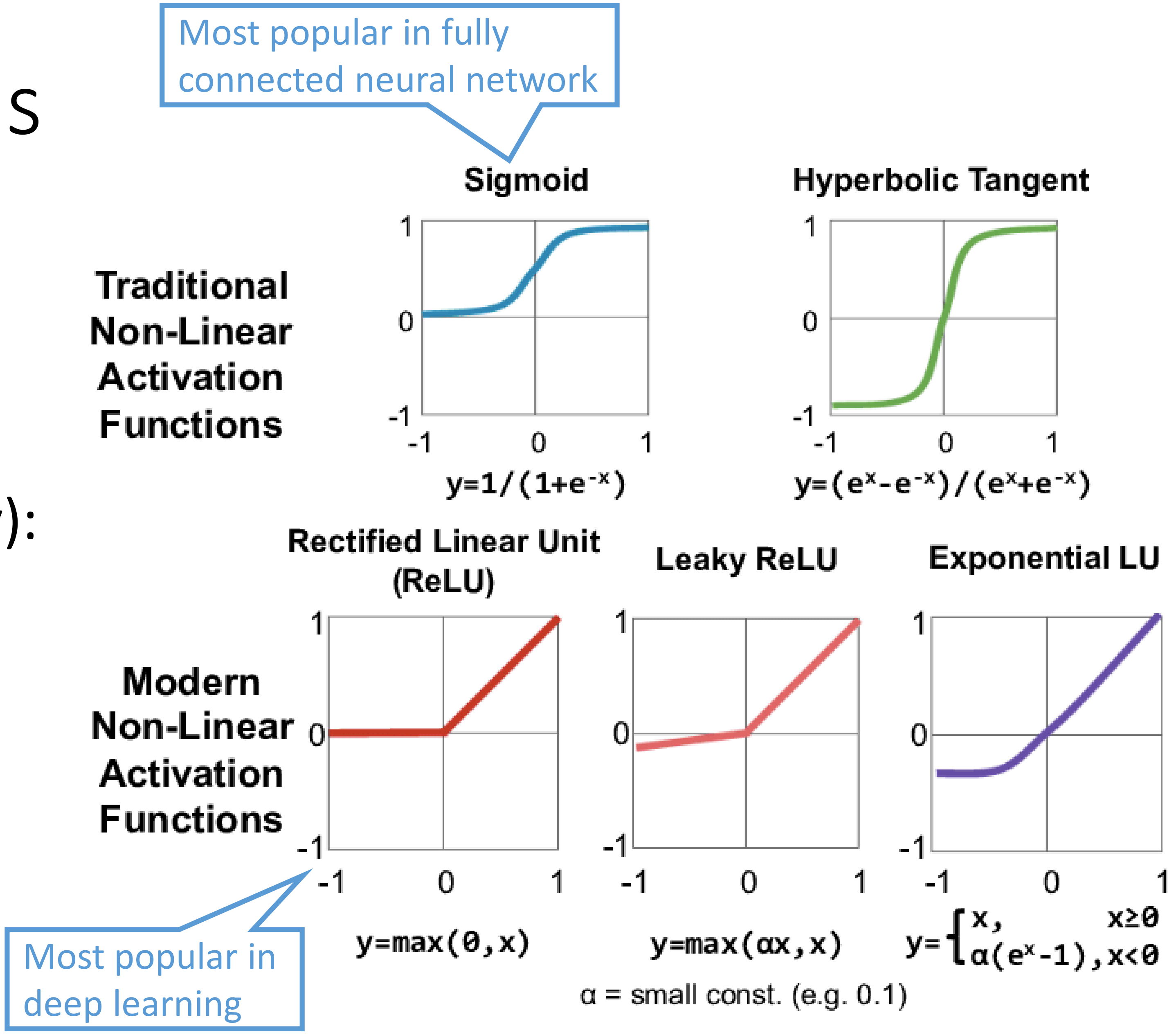
x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

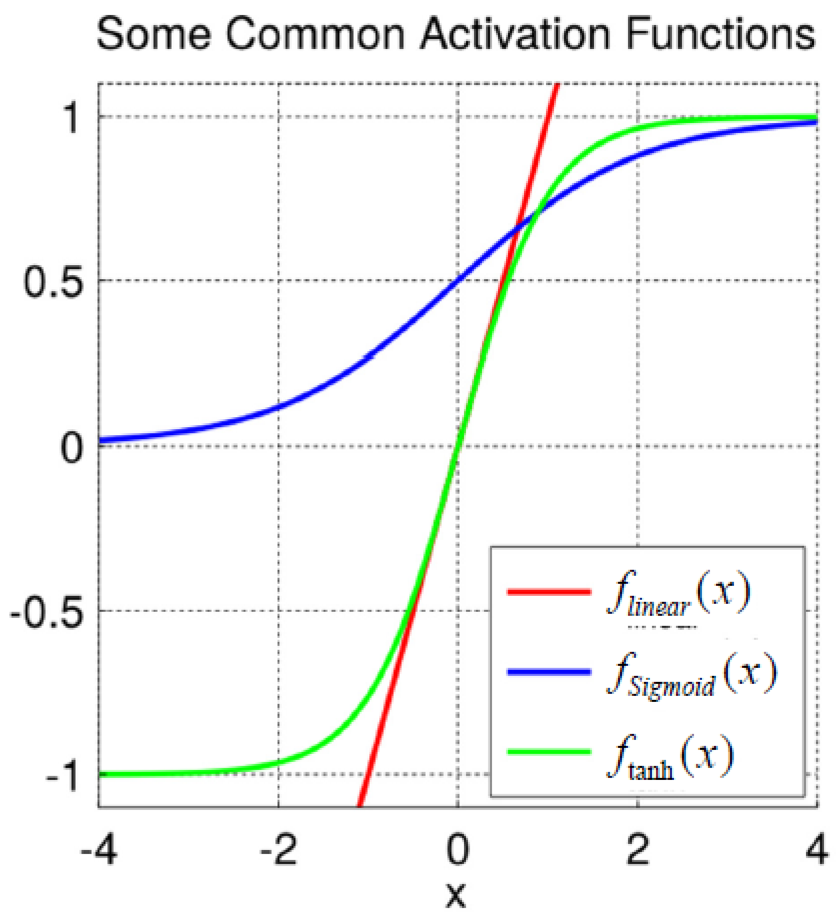
- It receives signals x_i's, multiplies them with different weights w_i, and outputs the sum of the weighted signals after an activation function, step function

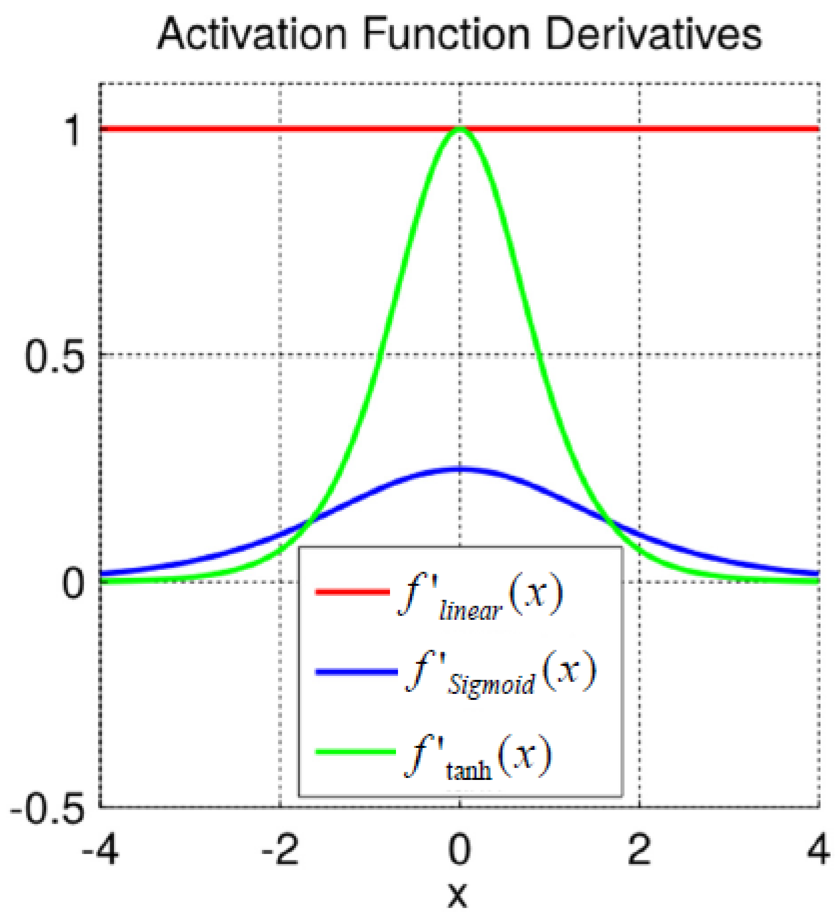
Activation functions

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- Tanh: $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
- ReLU (Rectified Linear Unity): $\mathrm{ReLU}(z) = \max(0, z)$



Activation function values and derivatives

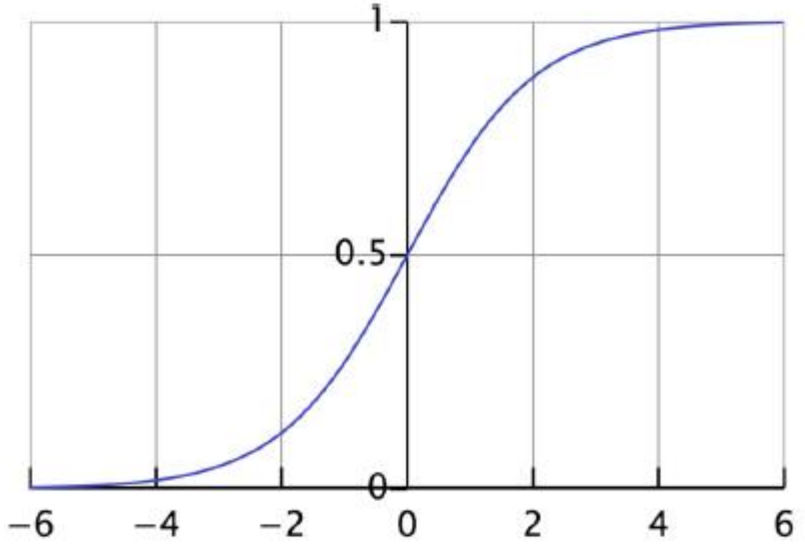




Sigmoid activation function

- Sigmoid

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



- Its derivative

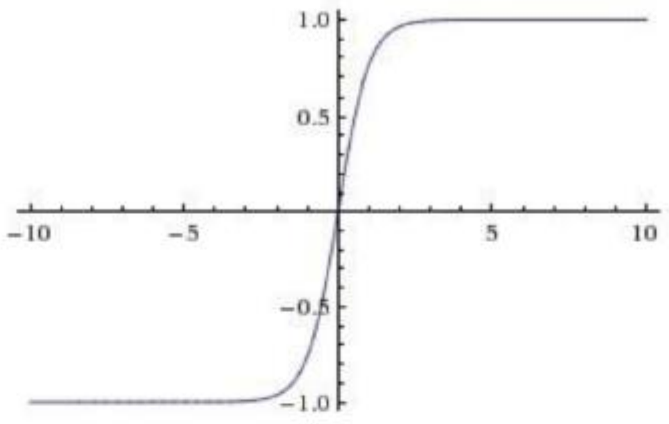
$$\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big)$$

- Output range $(0,1)$
- Motivated by biological neurons and can be interpreted as the probability of an artificial neuron “firing” given its inputs
- However, saturated neurons make value vanished **(why?)**
    - $f\Big(f\big(f(\cdots)\big)\Big)$
    - $f([0,1]) \subseteq [0.5, 0.732)$
    - $f\big([0.5, 0.732)\big) \subseteq (0.622, 0.676)$

Tanh activation function

- Tanh function

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

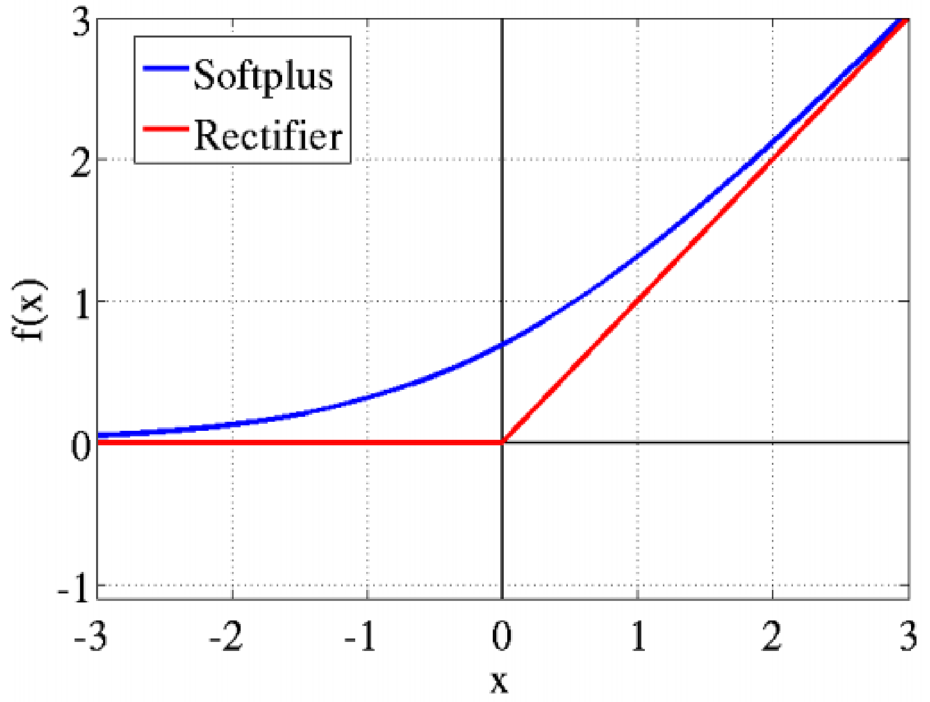


- Its derivative
$$\tanh'(z) = 1 - \tanh^2(z)$$
- Output range $(-1,1)$
- Thus strongly negative inputs to the tanh will map to negative outputs
- Only zero-valued inputs are mapped to near-zero outputs
- These properties make the network less likely to get “stuck” during training

ReLU activation function

- ReLU (Rectified linear unity) function

$$\mathrm{ReLU}(z) = \max(0, z)$$



- Its derivative

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \le 0 \end{cases}$$

- ReLU can be approximated by softplus function

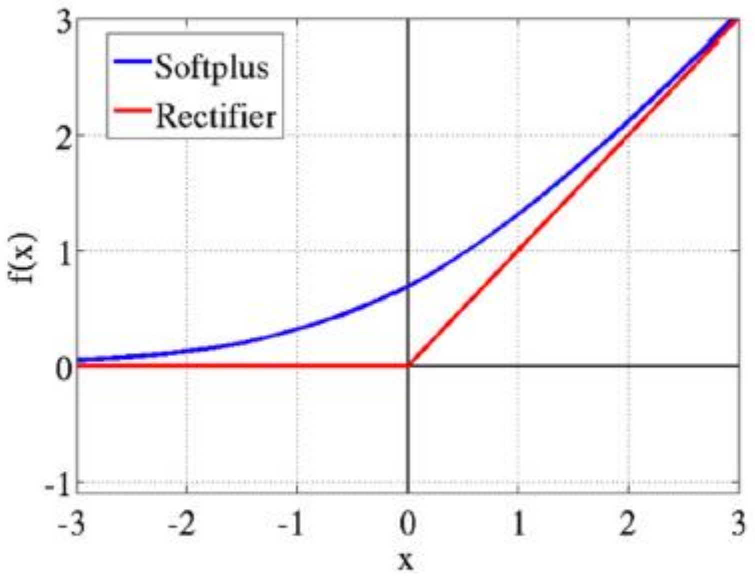
$$f_{\text{Softplus}}(x) = \log(1 + e^{x})$$

- ReLU's gradient doesn't vanish as x increases
- Speed up training of neural networks
  - Since the gradient computation is very simple
  - The computational step is simple, no exponentials, no multiplication or division operations (compared to others)
- The gradient on positive portion is larger than sigmoid or tanh functions
  - Update more rapidly
  - The left "dead neuron" part can be ameliorated by Leaky ReLU

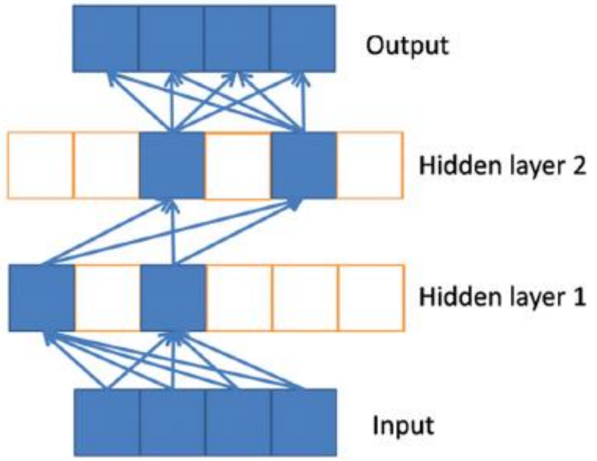
ReLU activation function (cont.)

- ReLU function

$$\mathrm{ReLU}(z) = \max(0, z)$$

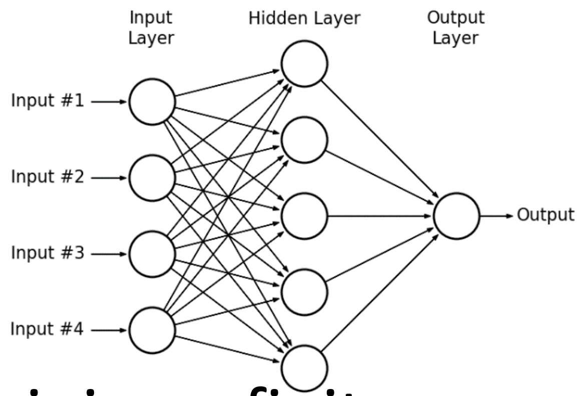


- The only non-linearity comes from the path selection with individual neurons being active or not
- It allows sparse representations:
  - for a given input only a subset of neurons are active



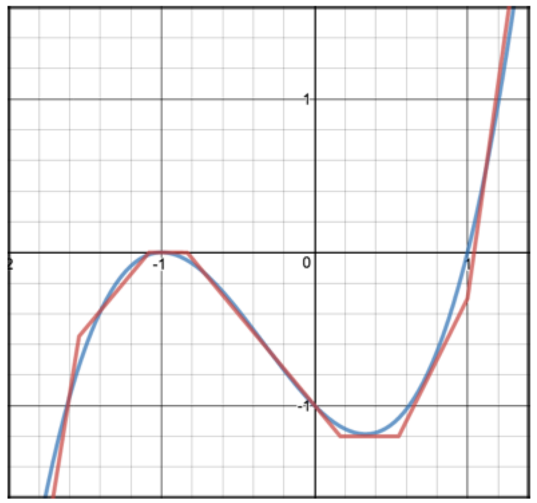
Sparse propagation of activations and gradients

Universal approximation theorem



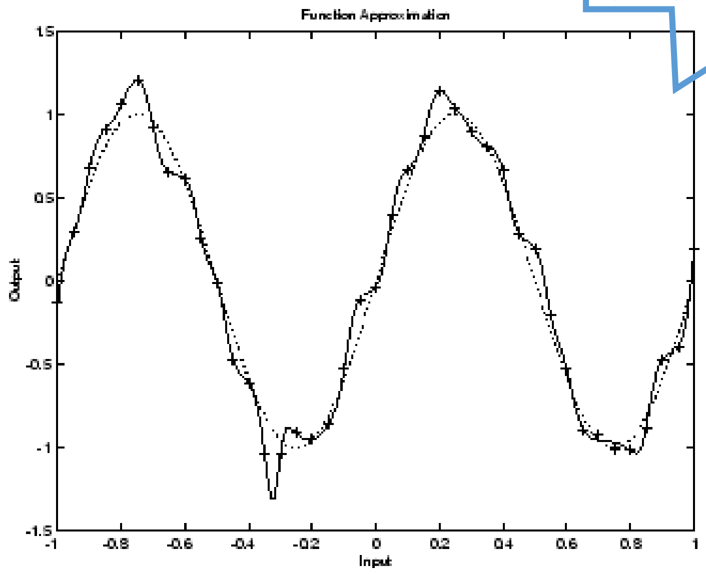
- A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions

1-20-1 NN approximates a noisy sine function



$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

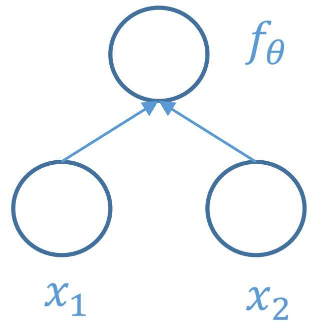
$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$



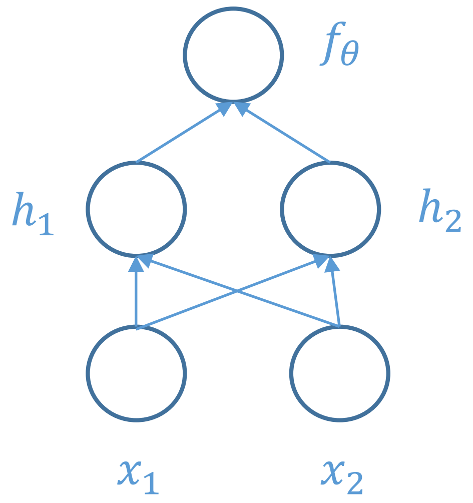
Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." *Neural networks* 2.5 (1989): 359-366

Single / Multiple layers of calculation

- Single layer function
  $$f_\theta(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

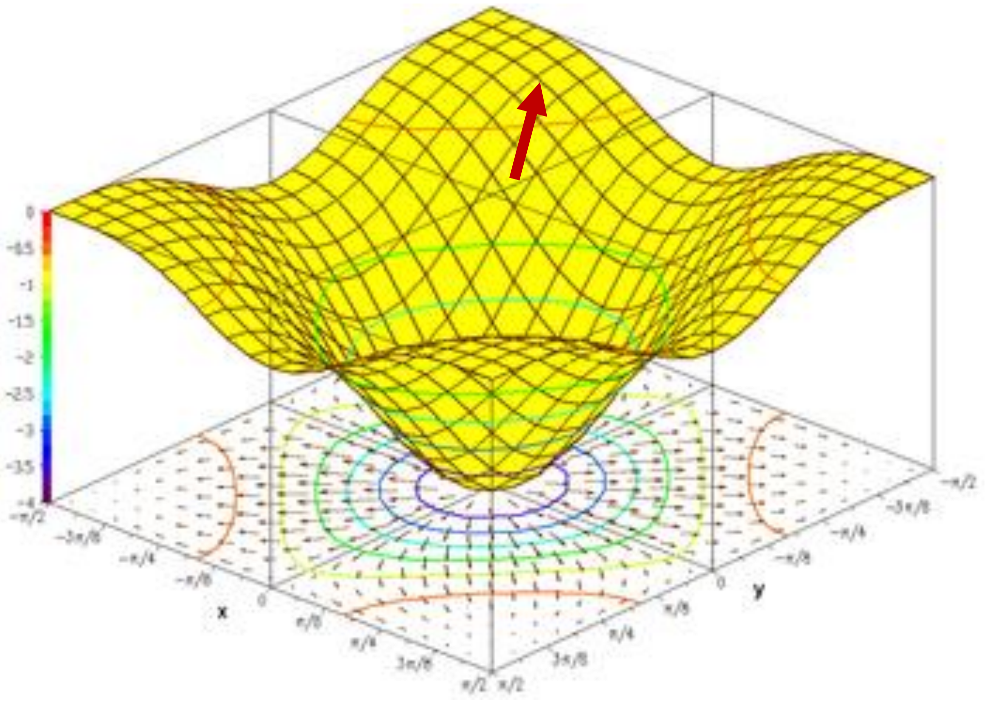


- Multiple layer function
  - $h_1(x) = \sigma(\theta_0^1 + \theta_1^1 x_1 + \theta_2^1 x_2)$
  - $h_2(x) = \sigma(\theta_0^2 + \theta_1^2 x_1 + \theta_2^2 x_2)$
  - $f_\theta(h) = \sigma(\theta_0 + \theta_1 h_1 + \theta_2 h_2)$



Gradient interpretation

- Gradient is the vector (the red one) along which the value of the function increases most rapidly. Thus its opposite direction is where the value decreases most rapidly.



Gradient descent

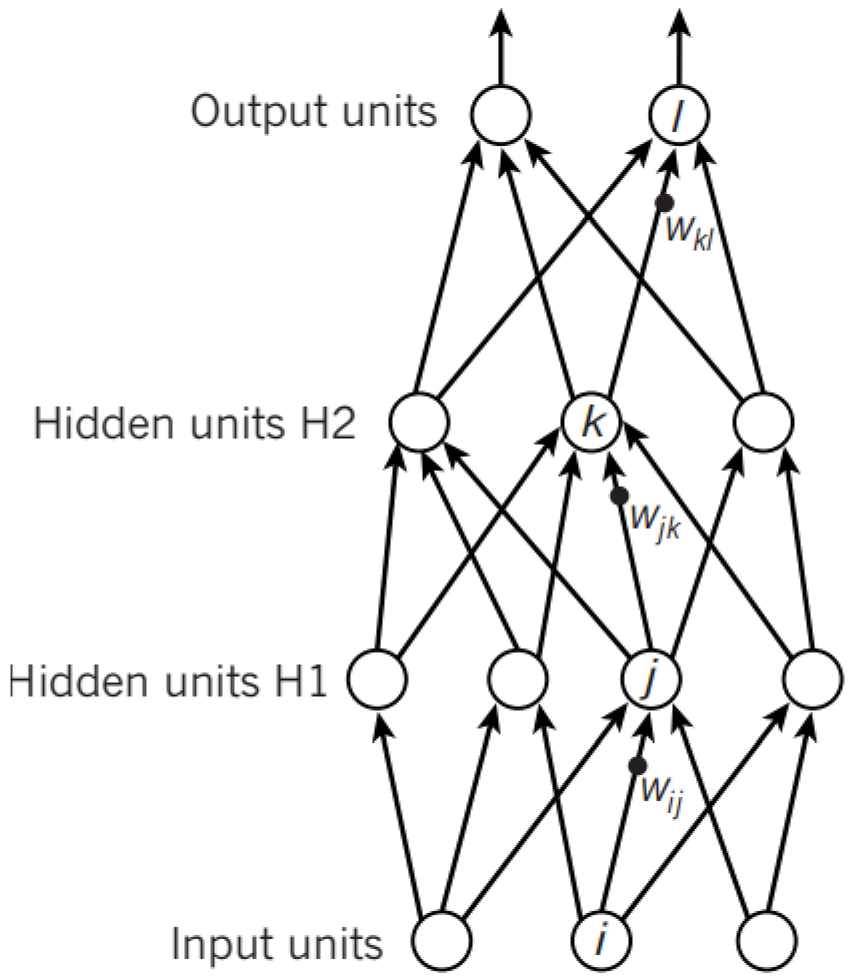
- To find a (local) minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or an approximation) of the function at the current point

- For a smooth function $f(x)$, $\frac{\partial f}{\partial x}$ is the direction that f increases most rapidly. So we apply

$$x_{t+1} = x_t - \eta \frac{\partial f}{\partial x}(x_t)$$

  until x converges

Feed forward vs. Backpropagation



$$y_l = f(z_l)$$

$$z_l = \sum_{k \,\varepsilon\, \mathrm{H2}} w_{kl}\, y_k$$

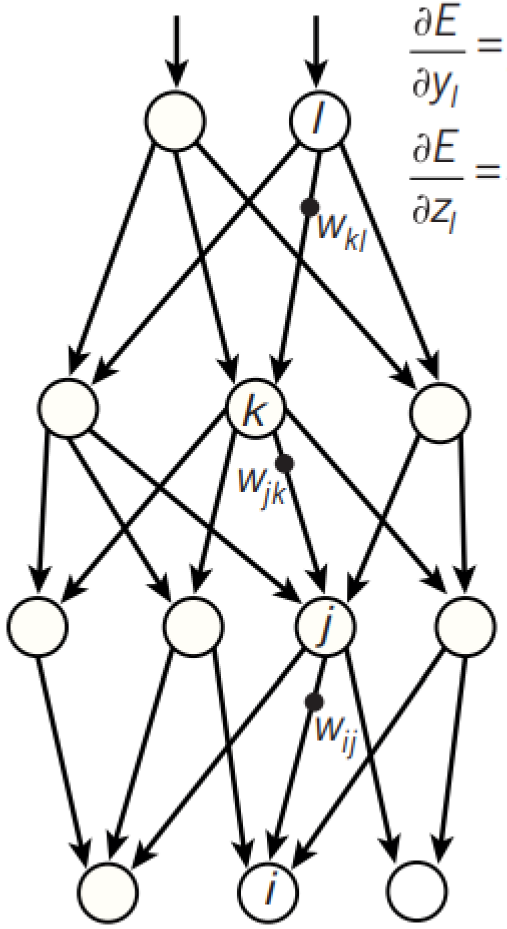
$$y_k = f(z_k)$$

$$z_k = \sum_{j \,\varepsilon\, \mathrm{H1}} w_{jk}\, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \,\varepsilon\, \mathrm{Input}} w_{ij}\, x_i$$

Compare outputs with correct answer to get error derivatives



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$

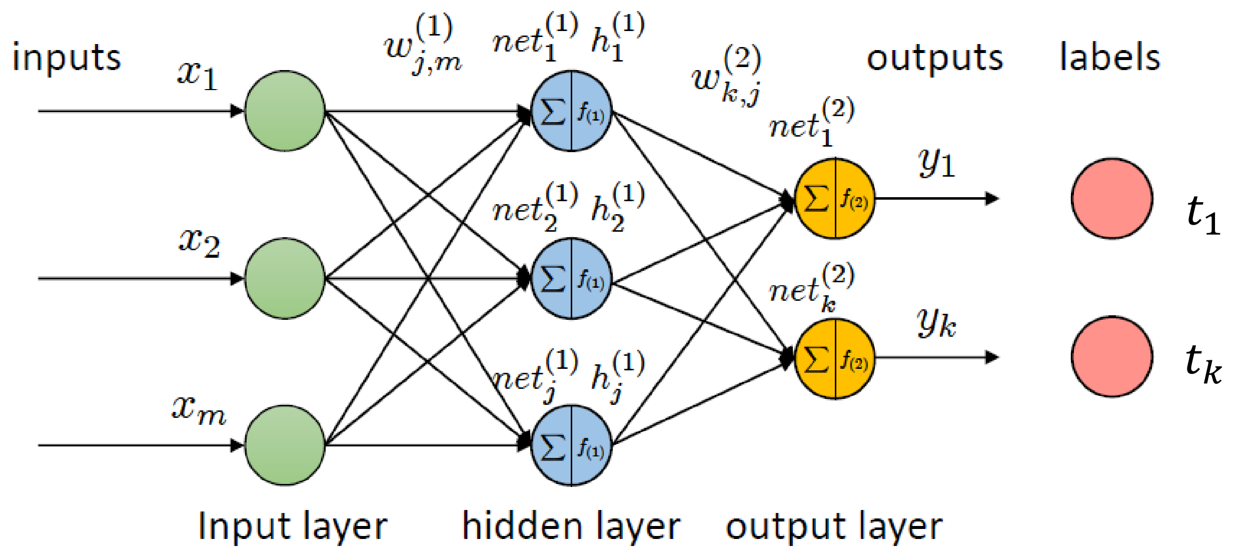
$$\frac{\partial E}{\partial y_k} = \sum_{l \,\varepsilon\, \mathrm{out}} w_{kl}\frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \,\varepsilon\, \mathrm{H2}} w_{jk}\frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

Make a prediction



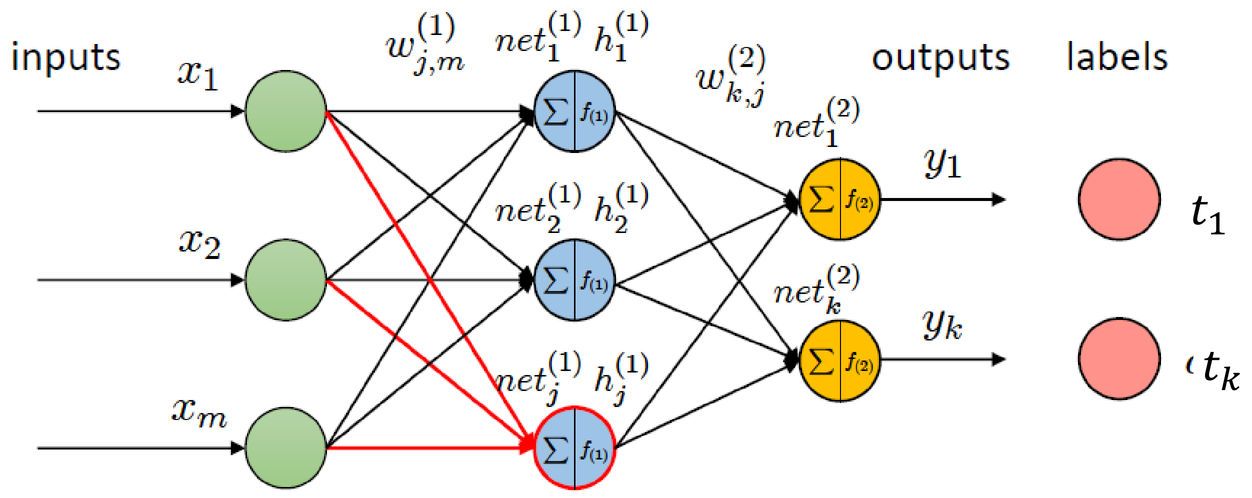
Two-layer feedforward neural network

Feed-forward prediction:

$$x = (x_1, \ldots, x_m) \xrightarrow{h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \xrightarrow{y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})} y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad$ $net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Make a prediction (cont.)



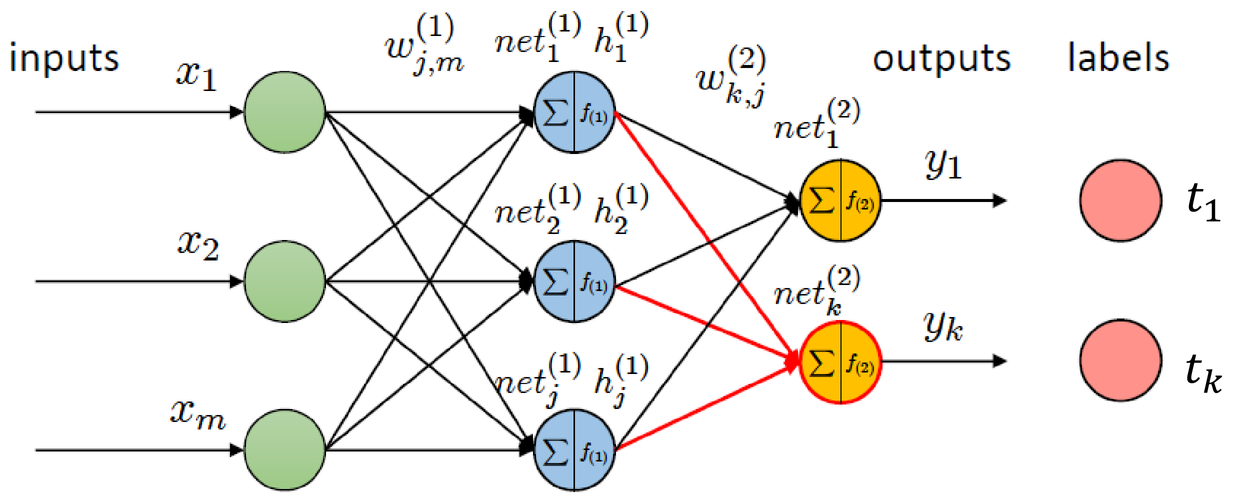
Two-layer feedforward neural network

Feed-forward prediction:

$$x = (x_1, \dots, x_m) \xrightarrow{h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \xrightarrow{y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})} y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Make a prediction (cont.)



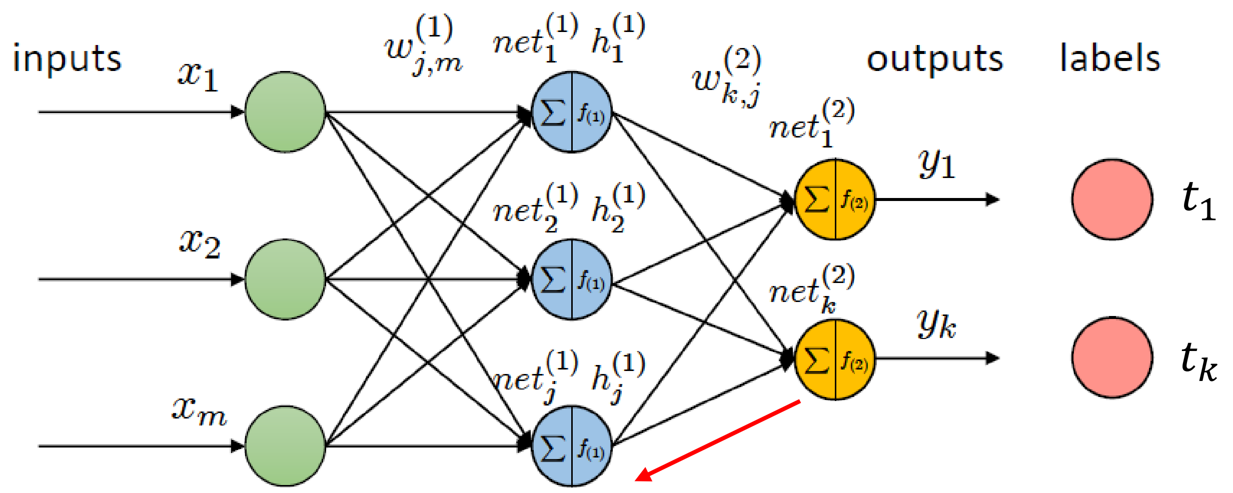
Two-layer feedforward neural network

Feed-forward prediction:

$$x = (x_1, \dots, x_m) \xrightarrow{h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \xrightarrow{y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})} y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Backpropagation



Two-layer feedforward neural network

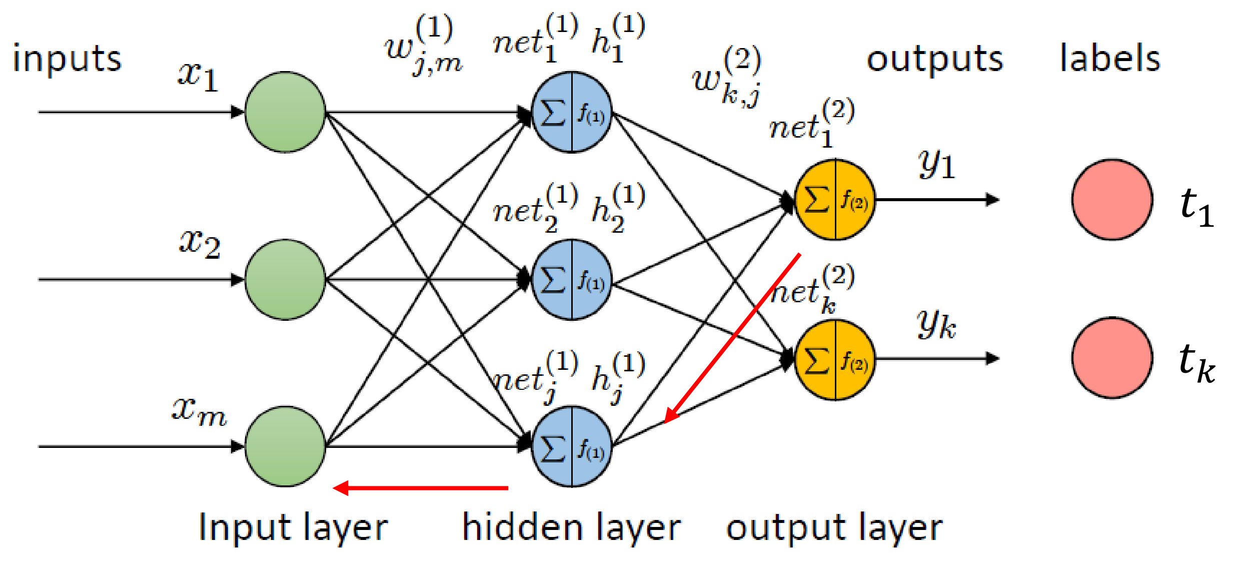
- Assume all the activation functions are sigmoid
- Error function $E = \frac{1}{2}\Sigma_k (y_k - t_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - t_k$
- $\frac{\partial y_k}{\partial w_{k,j}^{(2)}} = f'_{(2)}\left(net_k^{(2)}\right) h_j^{(1)} = y_k(1 - y_k) h_j^{(1)}$
- $\Rightarrow \frac{\partial E}{\partial w_{k,j}^{(2)}} = -\underbrace{(t_k - y_k) y_k (1 - y_k)}_{\delta_k^{(2)}} h_j^{(1)}$
- $\Rightarrow w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} + \eta \delta_k^{(2)} \underbrace{h_j^{(1)}}_{\text{Output of unit } j}$

Feed-forward prediction:

$$x = (x_1, \ldots, x_m) \xrightarrow{h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \xrightarrow{y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})} y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Backpropagation (cont.)



Two-layer feedforward neural network

- Error function $E = \frac{1}{2}\sum_k (y_k - t_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - t_k$
- $\delta_k^{(2)} = (t_k - y_k) y_k (1 - y_k)$
- $\Rightarrow w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} + \eta \delta_k^{(2)} h_j^{(1)}$
- $\frac{\partial y_k}{\partial h_j^{(1)}} = y_k (1 - y_k) w_{k,j}^{(2)}$
- $\frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = f'_{(1)}\left(net_j^{(1)}\right) x_m = h_j^{(1)}\left(1 - h_j^{(1)}\right) x_m$
- $\frac{\partial E}{\partial w_{j,m}^{(1)}} = -h_j^{(1)}\left(1 - h_j^{(1)}\right) \sum_k w_{k,j}^{(2)} (t_k - y_k) y_k (1 - y_k)\, x_m$

$$= \underbrace{-h_j^{(1)}\left(1 - h_j^{(1)}\right) \sum_k w_{k,j}^{(2)} \delta_k^{(2)}}_{\boxed{\delta_j^{(1)}}} x_m$$

- $\Rightarrow w_{j,m}^{(1)} \leftarrow w_{j,m}^{(1)} + \eta \delta_j^{(1)} x_m$

Feed-forward prediction:

$$\boxed{x = (x_1, \dots, x_m) \xrightarrow{h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)}} \xrightarrow{y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(1)} h_j^{(1)})} y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

- Error function $E = \frac{1}{2}\sum_k (y_k - t_k)^2$
- $\delta_k^{(2)} = (t_k - y_k)y_k(1 - y_k)$
- $\Rightarrow w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} + \eta\delta_k^{(2)} h_j^{(1)}$
- $\delta_j^{(1)} = h_j^{(1)}\left(1 - h_j^{(1)}\right)\sum_k w_{k,j}^{(2)}\delta_k^{(2)}$
- $\Rightarrow w_{j,m}^{(1)} \leftarrow w_{j,m}^{(1)} + \eta\delta_j^{(1)} x_m$

Backpropagation algorithms

- Activation function: sigmoid

Initialize all weights to small random numbers

Do until convergence

- For each training example:
  1. Input it to the network and compute the network output
  2. For each output unit k, o_k is the output of unit k

     $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$
  3. For each hidden unit j, o_j is the output of unit j

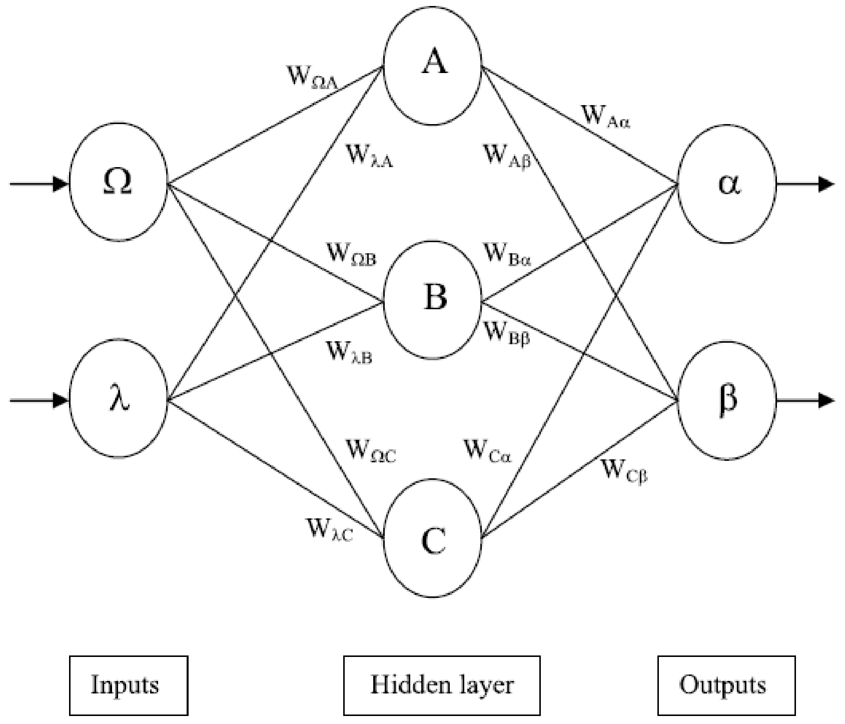
     $$\delta_j \leftarrow o_j(1 - o_j) \sum_{k \in next\ layer} w_{k,j}\delta_k$$
  4. Update each network weight, where x_i is the output for unit i

     $$w_{j,i} \leftarrow w_{j,i} + \eta\delta_j x_i$$

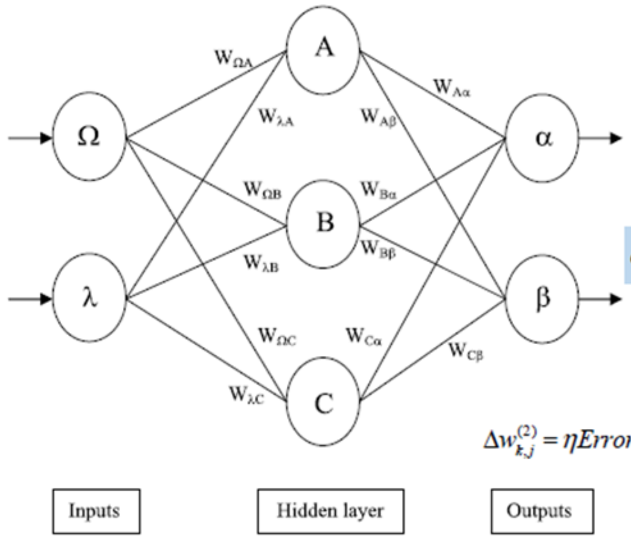
See the backpropagation demo

- https://google-developers.appspot.com/machine-learning/crash-course/backprop-scroll/

Formula example for backpropagation



Formula example for backpropagation (cont.)



1. Calculate errors of output neurons

$$\delta_k = (d_k - y_k)\ f_{(2)}{}'(net_k^{(2)})$$

$\delta_\alpha = \text{out}_\alpha\ (1 - \text{out}_\alpha)\ (\text{Target}_\alpha - \text{out}_\alpha)$

$\delta_\beta = \text{out}_\beta\ (1 - \text{out}_\beta)\ (\text{Target}_\beta - \text{out}_\beta)$

2. Change output layer weights

$$\Delta w_{k,j}^{(2)} = \eta Error_k Output_j = \eta \delta_k h_j^{(1)}$$

$W^{+}_{A\alpha} = W_{A\alpha} + \eta\delta_\alpha\ \text{out}_A$ $\qquad$ $W^{+}_{A\beta} = W_{A\beta} + \eta\delta_\beta\ \text{out}_A$

$W^{+}_{B\alpha} = W_{B\alpha} + \eta\delta_\alpha\ \text{out}_B$ $\qquad$ $W^{+}_{B\beta} = W_{B\beta} + \eta\delta_\beta\ \text{out}_B$

$W^{+}_{C\alpha} = W_{C\alpha} + \eta\delta_\alpha\ \text{out}_C$ $\qquad$ $W^{+}_{C\beta} = W_{C\beta} + \eta\delta_\beta\ \text{out}_C$

3. Calculate (back-propagate) hidden layer errors

$$\delta_j = f_{(1)}{}'(net_j^{(1)}) \sum_k \delta_k w_{k,j}^{(2)}$$

$\delta_A = \text{out}_A\ (1 - \text{out}_A)\ (\delta_\alpha W_{A\alpha} + \delta_\beta W_{A\beta})$

$\delta_B = \text{out}_B\ (1 - \text{out}_B)\ (\delta_\alpha W_{B\alpha} + \delta_\beta W_{B\beta})$

$\delta_C = \text{out}_C\ (1 - \text{out}_C)\ (\delta_\alpha W_{C\alpha} + \delta_\beta W_{C\beta})$

4. Change hidden layer weights

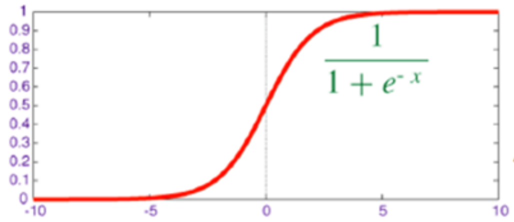
$$\Delta w_{j,m}^{(1)} = \eta Error_j Output_m = \eta \delta_j x_m$$

$W^{+}_{\lambda A} = W_{\lambda A} + \eta\delta_A\ \text{in}_\lambda$ $\qquad$ $W^{+}_{\Omega A} = W^{+}_{\Omega A} + \eta\delta_A\ \text{in}_\Omega$

$W^{+}_{\lambda B} = W_{\lambda B} + \eta\delta_B\ \text{in}_\lambda$ $\qquad$ $W^{+}_{\Omega B} = W^{+}_{\Omega B} + \eta\delta_B\ \text{in}_\Omega$

$W^{+}_{\lambda C} = W_{\lambda C} + \eta\delta_C\ \text{in}_\lambda$ $\qquad$ $W^{+}_{\Omega C} = W^{+}_{\Omega C} + \eta\delta_C\ \text{in}_\Omega$

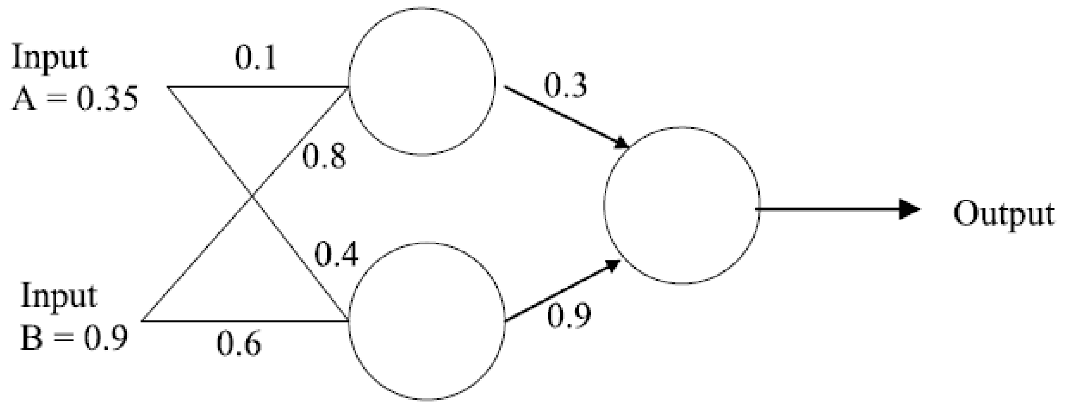
Consider sigmoid activation function $f_{Sigmoid}(x) = \dfrac{1}{1+e^{-x}}$



$$f'_{Sigmoid}(x) = f_{Sigmoid}(x)(1 - f_{Sigmoid}(x))$$

Calculation example

- Consider the simple network below:



- Assume that the neurons have sigmoid activation function and
  - Perform a forward pass on the network and find the predicted output
  - Perform a reverse pass (training) once (target = 0.5) with $\eta = 1$
  - Perform a further forward pass and comment on the result

Calculation example (cont.)

Answer:
(i)
Input to top neuron = (0.35x0.1)+(0.9x0.8)=0.755. Out = 0.68.
Input to bottom neuron = (0.9x0.6)+(0.35x0.4) = 0.68. Out = 0.6637.
Input to final neuron = (0.3x0.68)+(0.9x0.6637) = 0.80133. Out = 0.69.

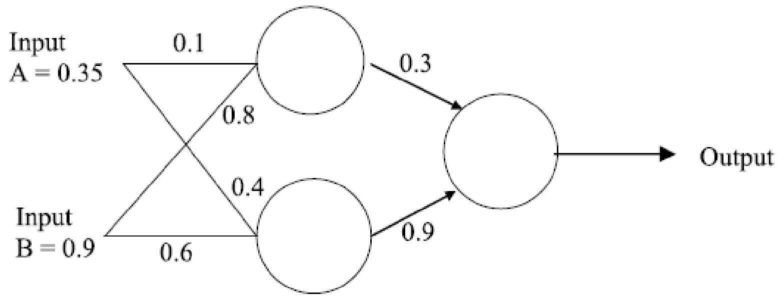
(ii)
Output error δ=(t-o)(1-o)o = (0.5-0.69)(1-0.69)0.69 = -0.0406.

New weights for output layer
$w1^+$ = w1+(δ x input) = 0.3 + (-0.0406x0.68) = 0.272392.
$w2^+$ = w2+(δ x input) = 0.9 + (-0.0406x0.6637) = 0.87305.

Errors for hidden layers:
δ1 = δ x w1 = -0.0406 x 0.272392 x (1-o)o = $-2.406\text{x}10^{-3}$
δ2= δ x w2 = -0.0406 x 0.87305 x (1-o)o = $-7.916\text{x}10^{-3}$

New hidden layer weights:
$w3^+$=0.1 + (-2.406 x 10^{-3} x 0.35) = 0.09916.
$w4^+$ = 0.8 + (-2.406 x 10^{-3} x 0.9) = 0.7978.
$w5^+$ = 0.4 + (-7.916 x 10^{-3} x 0.35) = 0.3972.
$w6^+$ = 0.6 + (-7.916 x 10^{-3} x 0.9) = 0.5928.

(iii)
Old error was -0.19. New error is -0.18205. Therefore error has reduced.



- For each output unit k, o_k is the output of unit k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

- For each hidden unit j, o_j is the output of unit j

$$\delta_j \leftarrow o_j(1-o_j) \sum_{k \in next\ layer} w_{k,j}\delta_k$$

- Update each network weight, where x_i is the input for unit j

$$w_{j,i} \leftarrow w_{j,i} + \eta\delta_j x_i$$

Calculation example (cont.)

- Answer (i)
  - Input to top neuron $= 0.35 \times 0.1 + 0.9 \times 0.8 = 0.755$. Out=0.68
  - Input to bottom neuron $= 0.35 \times 0.4 + 0.9 \times 0.6 = 0.68$. Out= 0.6637
  - Input to final neuron $= 0.3 \times 0.68 + 0.9 \times 0.6637 = 0.80133$. Out= 0.69
- (ii) It is both OK to use new or old weights when computing δ_j for hidden units
  - Output error $\delta = (t - o)o(1 - o) = (0.5 - 0.69) \times 0.69 \times (1 - 0.69) = -0.0406$
  - Error for top hidden neuron $\delta_1 = 0.68 \times (1 - 0.68) \times 0.3 \times (-0.0406) = -0.00265$
  - Error for top hidden neuron $\delta_2 = 0.6637 \times (1 - 0.6637) \times 0.9 \times (-0.0406) = -0.008156$
  - New weights for the output layer
  - $w_{o1} = 0.3 - 0.0406 \times 0.68 = 0.272392$
  - $w_{o2} = 0.9 - 0.0406 \times 0.6637 = 0.87305$
  - New weights for the hidden layer
  - $w_{1A} = 0.1 - 0.00265 \times 0.35 = 0.0991$
  - $w_{1B} = 0.8 - 0.00265 \times 0.9 = 0.7976$
  - $w_{2A} = 0.4 - 0.008156 \times 0.35 = 0.3971$
  - $w_{2B} = 0.6 - 0.008156 \times 0.9 = 0.5927$
- (iii)
  - Input to top neuron $= 0.35 \times 0.0991 + 0.9 \times 0.7976 = 0.7525$. Out=0.6797
  - Input to bottom neuron $= 0.35 \times 0.3971 + 0.9 \times 0.5927 = 0.6724$. Out= 0.662
  - Input to final neuron $= 0.272392 \times 0.6797 + 0.87305 \times 0.662 = 0.7631$. Out= 0.682
  - New error is -0.182, which is reduced compared to old error -0.19

- For each output unit k, o_k is the output of unit k

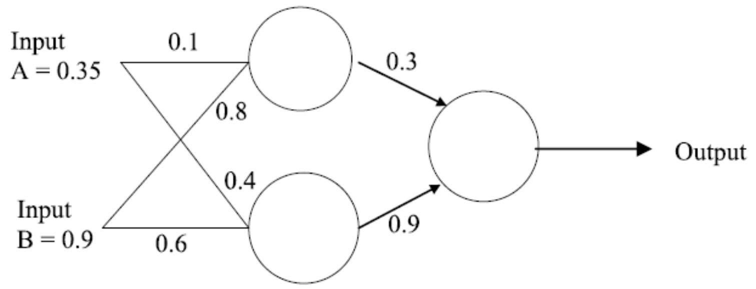
$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

- For each hidden unit j, o_j is the output of unit j

$$\delta_j \leftarrow o_j(1 - o_j) \sum_{k \in next\ layer} w_{k,j}\delta_k$$

- Update each network weight, where x_i is the input for unit j

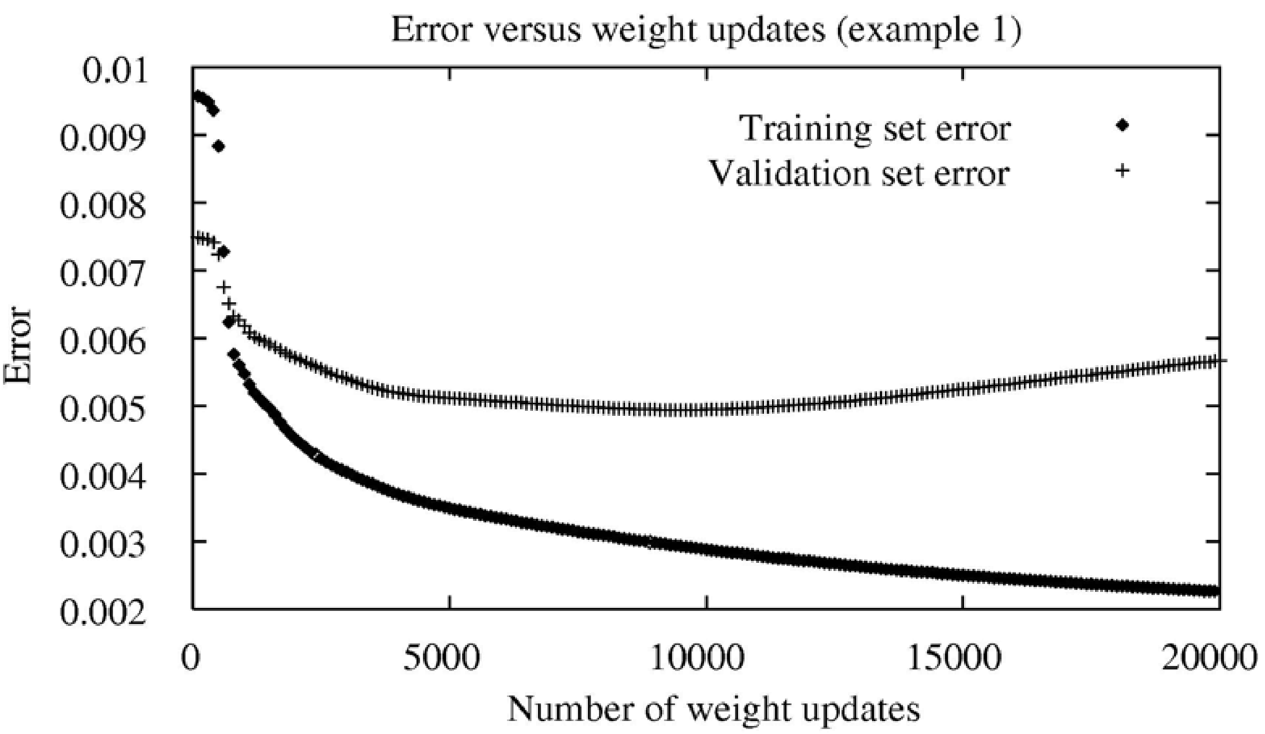
$$w_{j,i} \leftarrow w_{j,i} + \eta\delta_j x_i$$

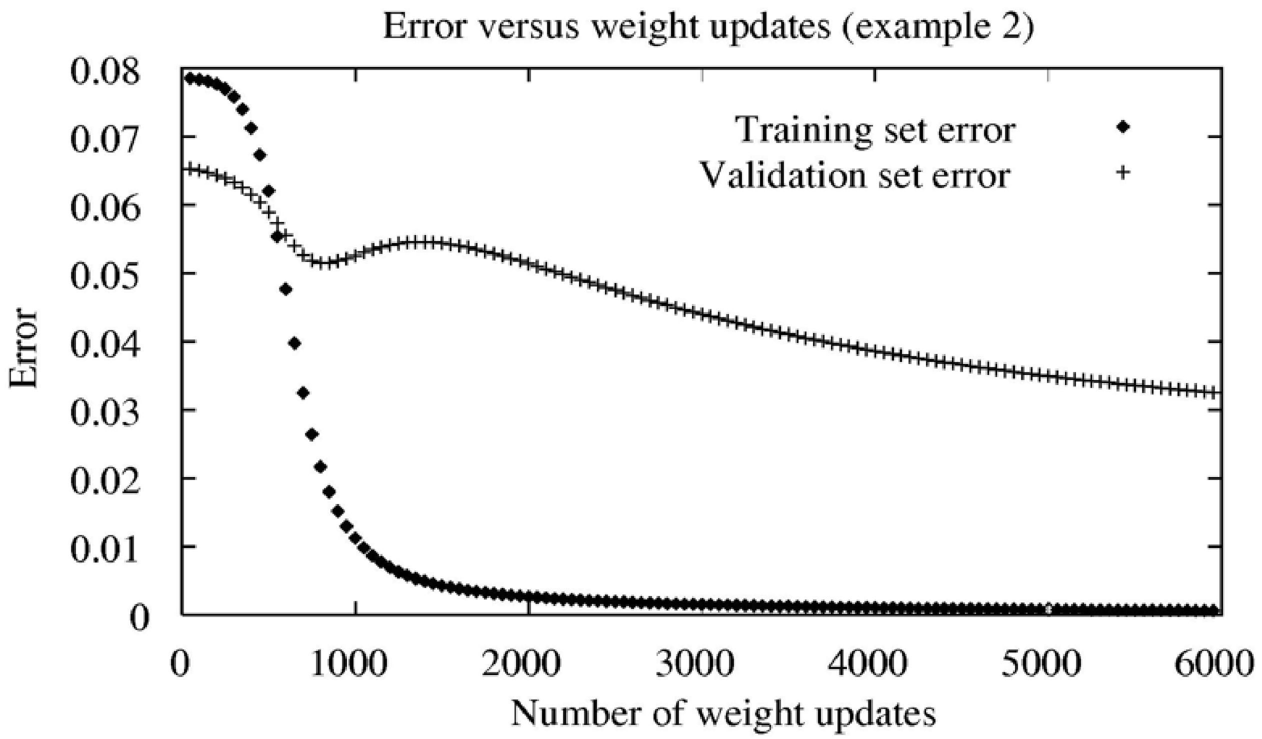


Convergence of backpropagation

- Gradient descent to some local minimum
  - Perhaps not global minimum
  - Add momentum
  - Stochastic gradient descent
  - Train multiple nets with different initial weights
- Nature of convergence
  - Initialize weights near zero
  - Therefore, initial networks near-linear
  - Increasingly approximate non-linear functions as training progresses

Two examples of overfitting





Avoid overfitting

- Penalize large weights: $E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd}-o_{kd})^2 + \boxed{\gamma\sum_{i,j} w_{ji}^2}$
- Train on target slopes as well as values:

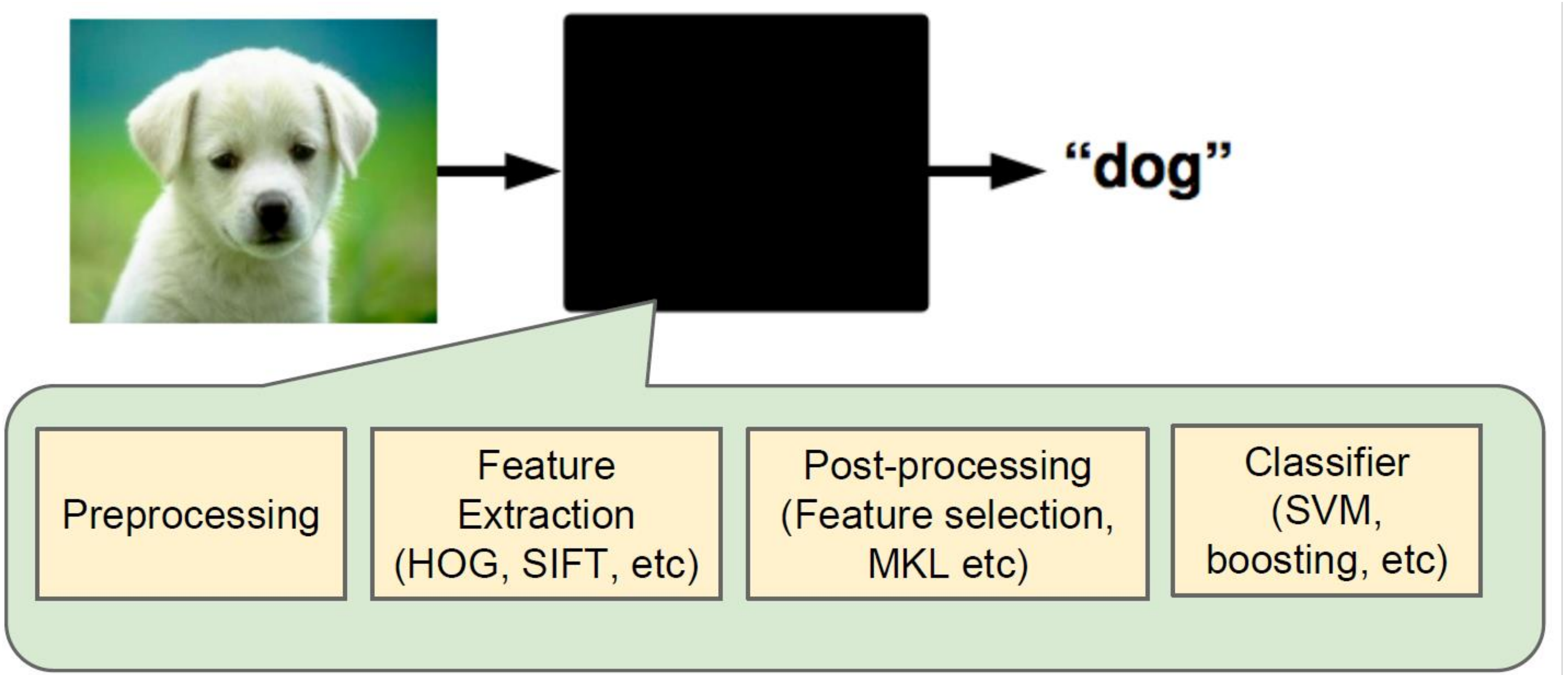
$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}\left[(t_{kd}-o_{kd})^2 + \mu\sum_{j\in inputs}\left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j}\right)^2\right]$$

- Weight sharing
  - Reduce total number of weights
  - Using structures, like convolutional neural networks
- Early stopping
- Dropout

Convolutional Neural Networks

Previous pipeline of pattern recognition

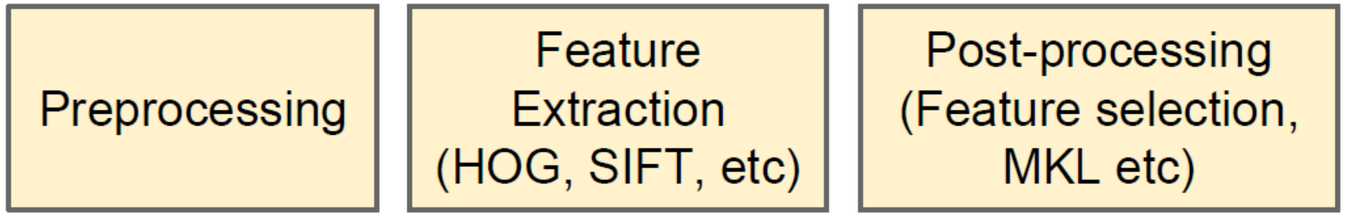
- The black box in a traditional pattern recognition problem


"dog"
Preprocessing
Feature Extraction (HOG, SIFT, etc)
Post-processing (Feature selection, MKL etc)
Classifier (SVM, boosting, etc)

Hand engineered features

- Feature is of critical importance in machine learning, and there are many things to consider when design the features manually:
  - How to design a feature?
  - What is the best feature?
  - Time and money cost in feature engineering.

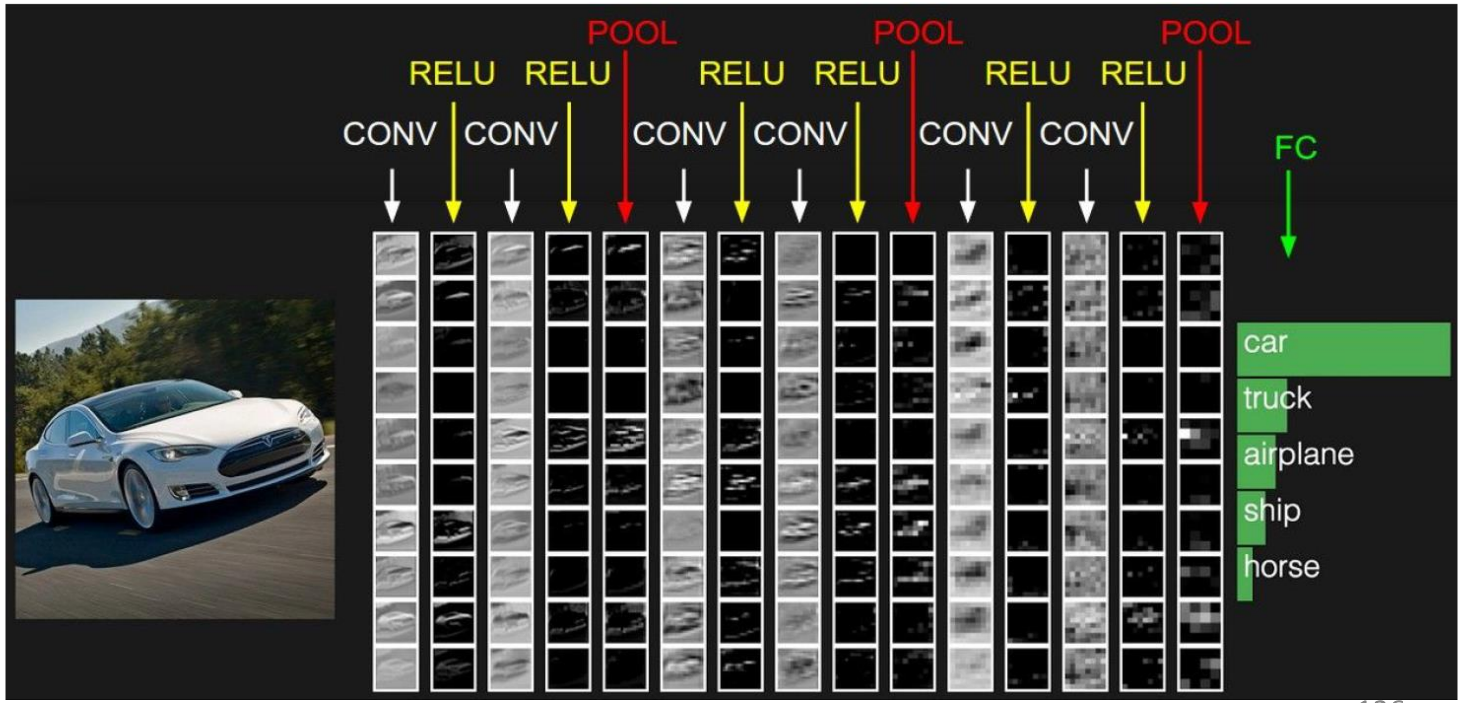
- Question: Can feature be learned automatically?


Preprocessing
Feature Extraction (HOG, SIFT, etc)
Post-processing (Feature selection, MKL etc)

Convolution neural networks

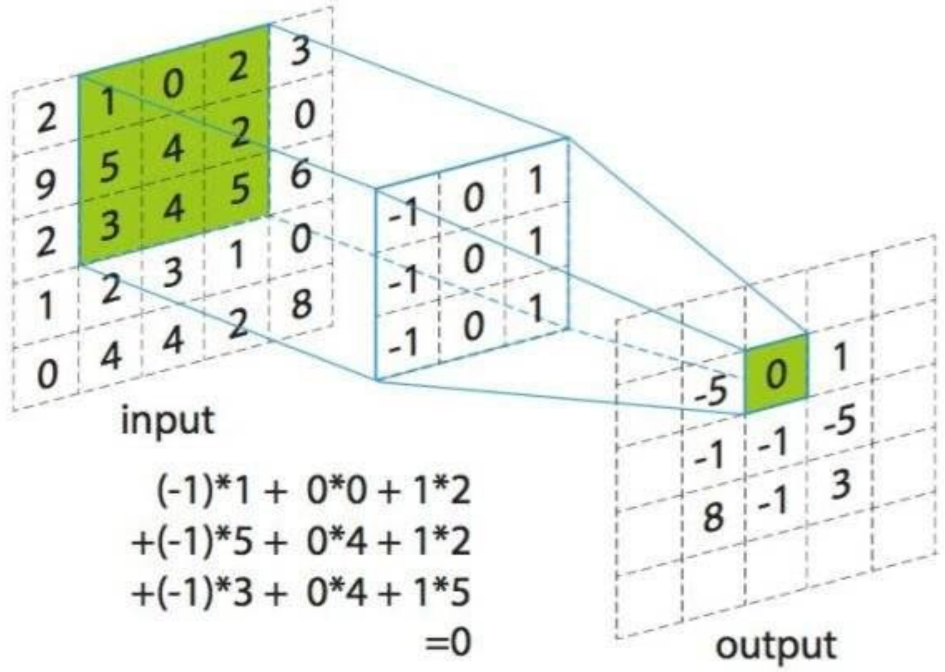
- Is an answer of an end-to-end recognition system
- Contains the following layers with flexible order and repetitions
  - Convolution layer
  - Activation layer (ReLU)
  - Pooling layer

- Example of CNN:

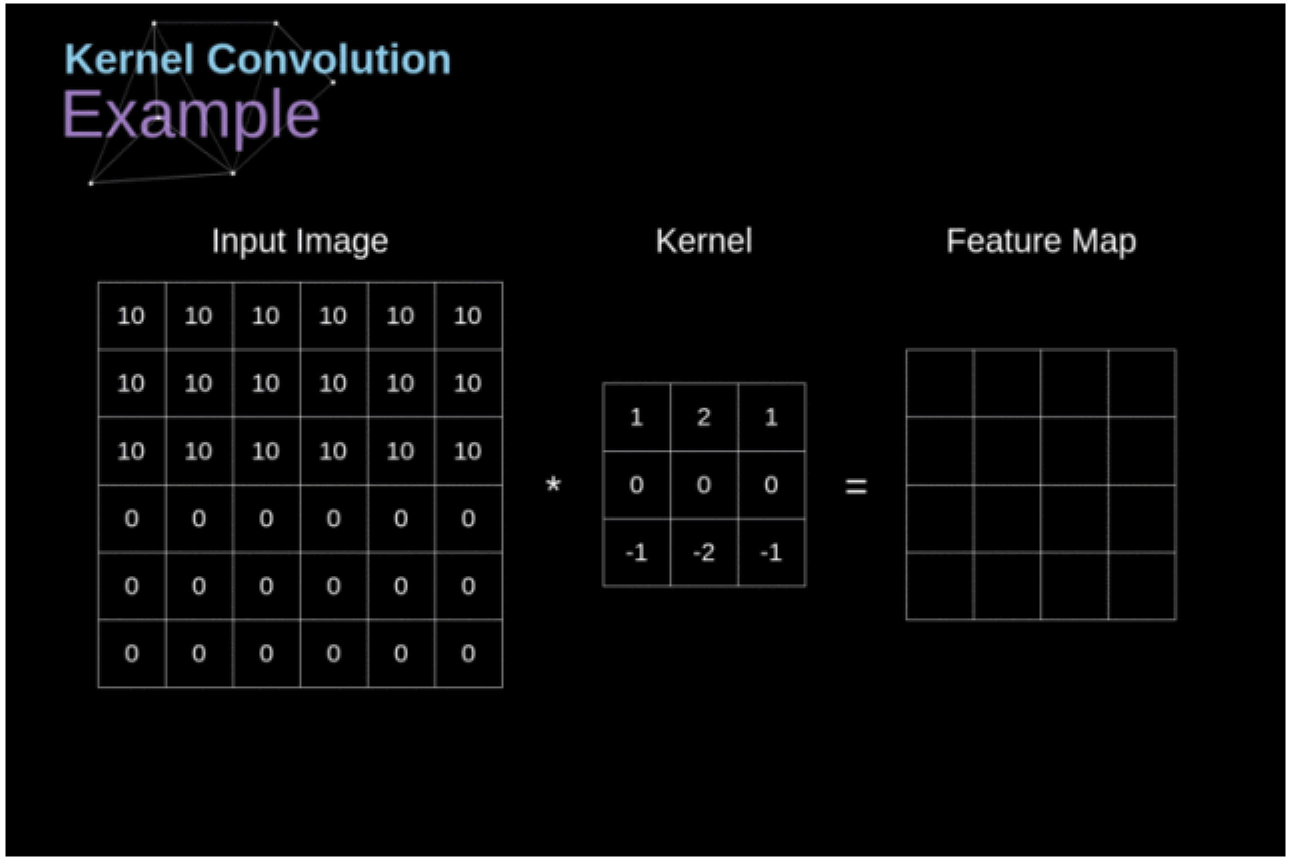


Convolution in neural networks

- Given an input matrix (e.g. an image)
- Use a small matrix (called filter or kernel) to screening the input at every position of the input matrix
- Put the convolution results at corresponding positions



An animation example

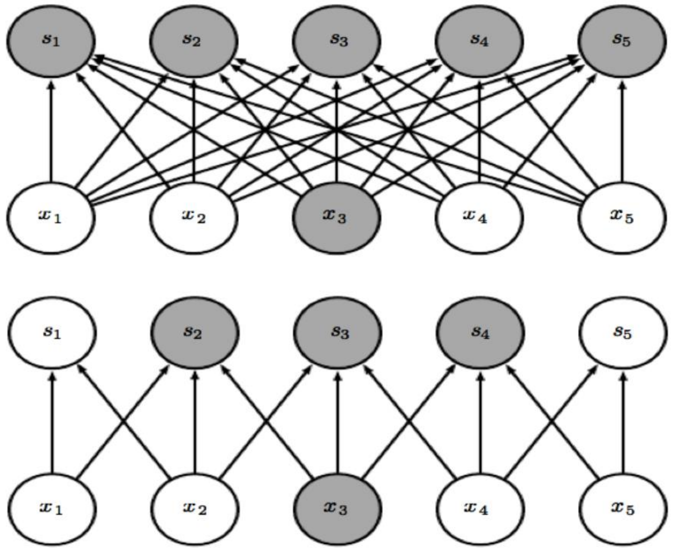


10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

1	2	1
0	0	0
-1	-2	-1

Advantage – sparse connections

- Less computing burden
- In fully connected layer (top), every s is linked to every input x, so there are $5 \times 5 = 25$ connecting edges
- In the convolution layer (bottom) with filter width 3, e.g. s_2 is a weighted sum of x_1, x_2, x_3, so there is no weight connecting s_2 and x_4, x_5. In this example, there are 13 connecting edges

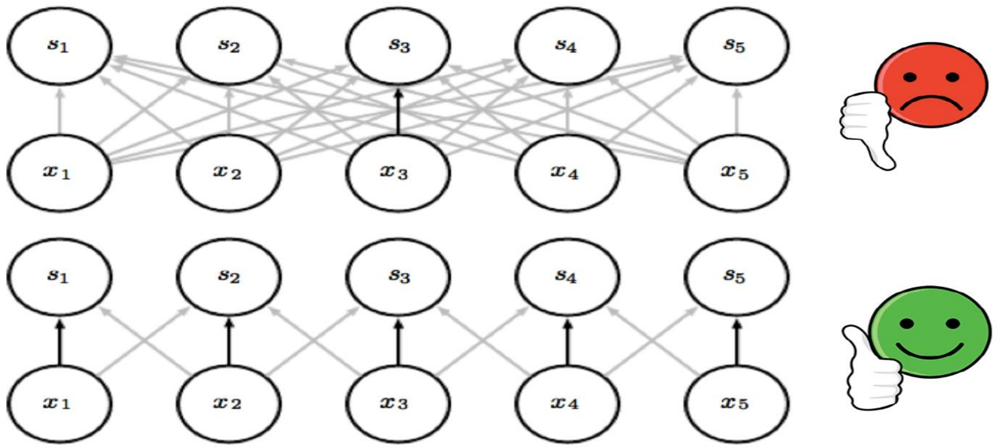


Advantage – weight sharing

- When moving the filter, we don't change the weights inside the filter and these weights are shared at different connecting edges
- In fully connected layer (top), there are 25 connecting edges. The weights on different edges are different parameters.
- In convolution layer (bottom), there are 13 connecting edges. But since

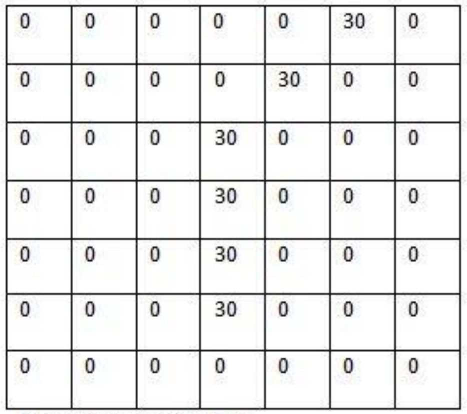
$$s_2 = w_1 x_1 + w_2 x_2 + w_3 x_3$$
$$s_3 = w_1 x_2 + w_2 x_3 + w_3 x_4$$

the number of different weights is even smaller. In this case, there are 3 different weights (just the size of the filter!). E.g. the weights on the black arrows are the same.



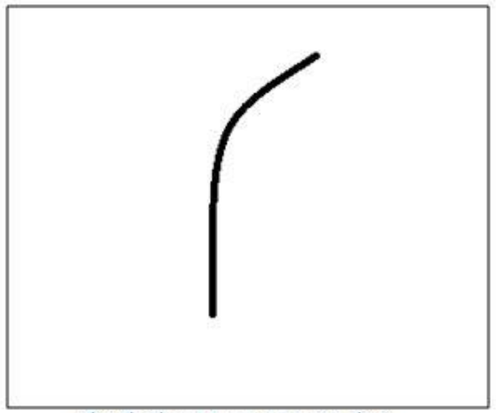
Interpretation of convolution

- Convolution can be used to find an area with particular patterns!
- Example:
    - The filter in the left represents the edge in the right, which is the back of a mouse

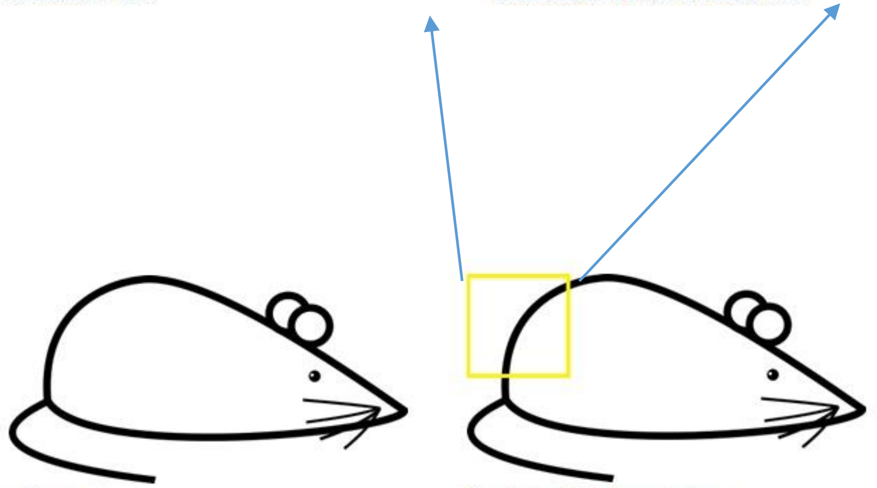


0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

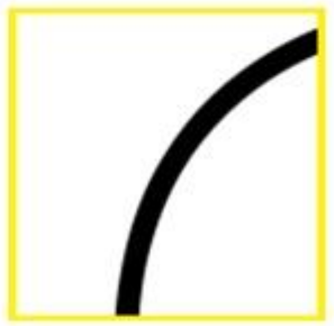


Original image

Visualization of the filter on the image

Interpretation of convolution (cont.)

- When the filter moves to the back of the mouse, the convolution operation will generate a very large value


Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

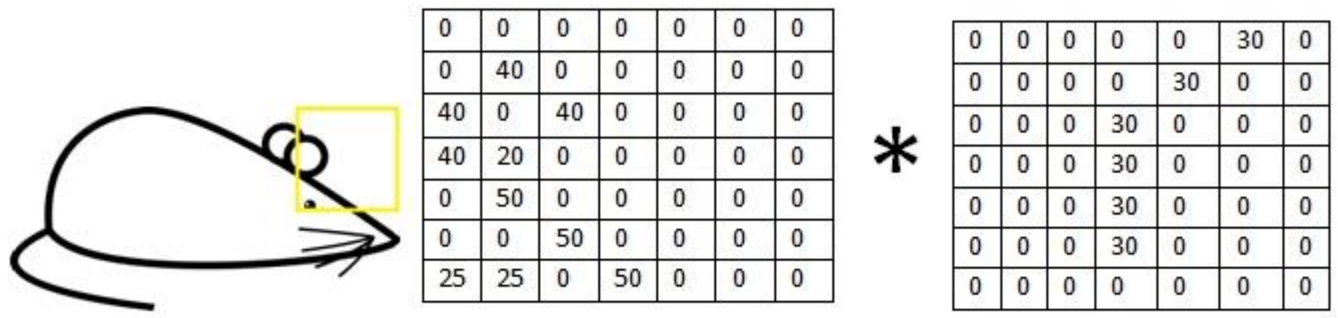
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Interpretation of convolution (cont.)

- When the filter moves to other positions, it will generate small values



0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Visualization of the filter on the image   Pixel representation of receptive field   Pixel representation of filter

Multiplication and Summation = 0

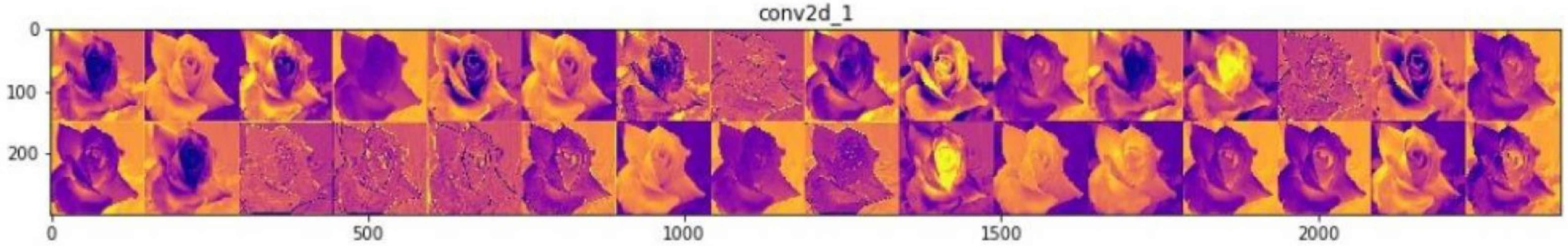
Visualization

- Train the InceptionV3 model (by Google) on the ImageNet dataset
- Then test it on a flower image from the test dataset
    - Example input image:

- Then let’s look at the outputs of different convolutional layers

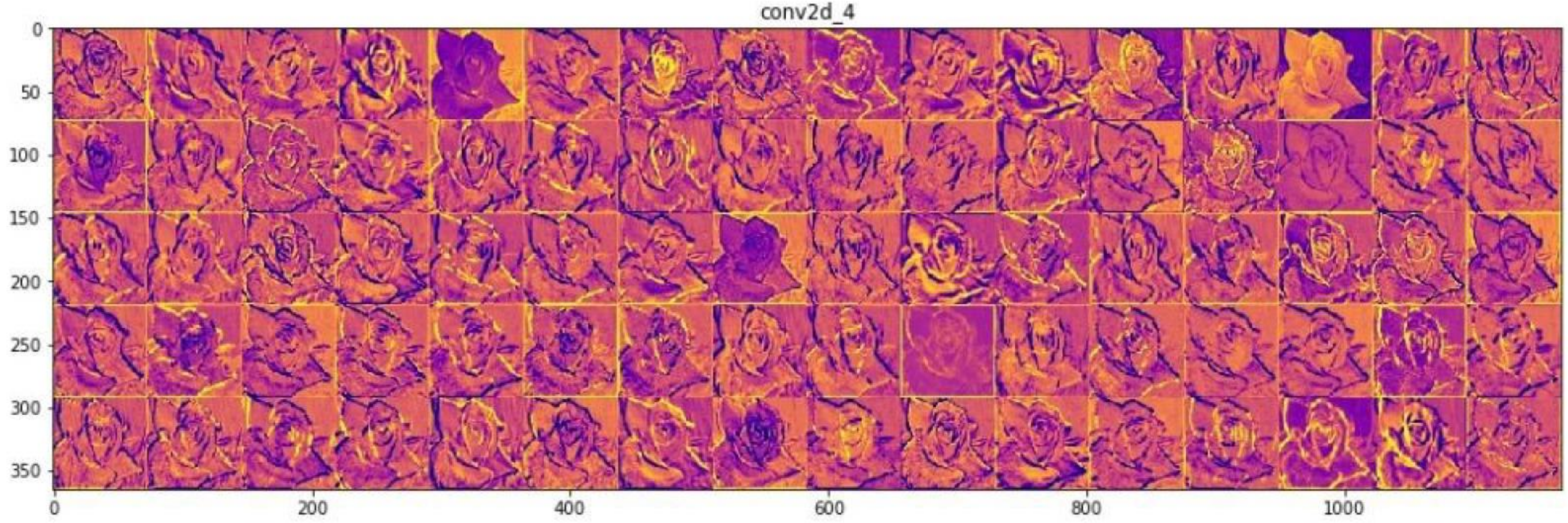
Visualization (cont.)

- The outputs of the filters in the first layer



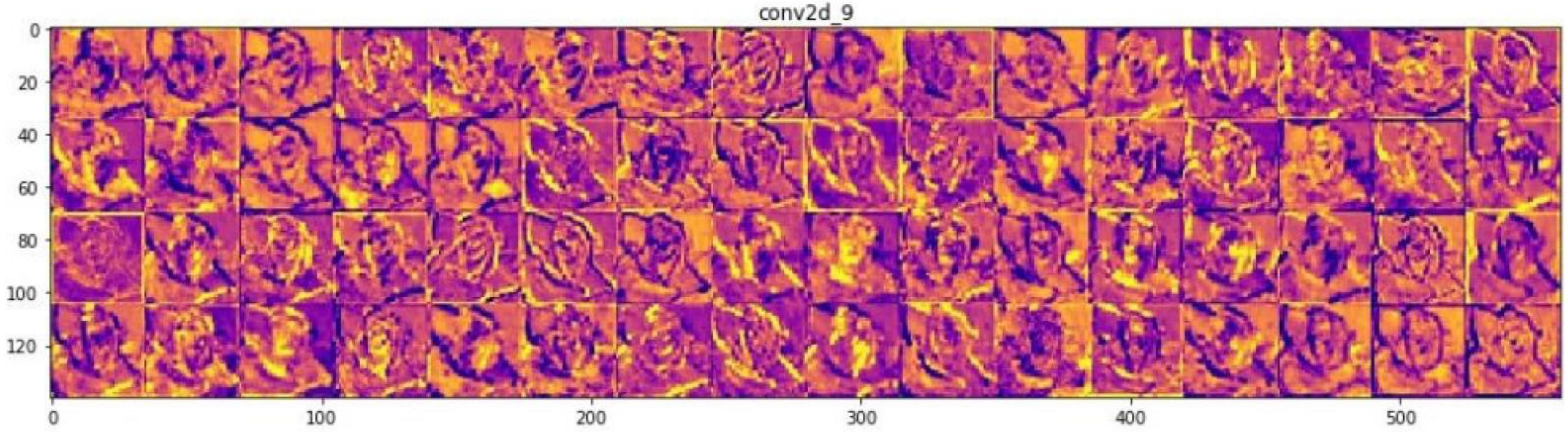
Visualization (cont.)

- The outputs of the filters in the fourth layer



Visualization (cont.)

- The outputs of the filters in the ninth layer

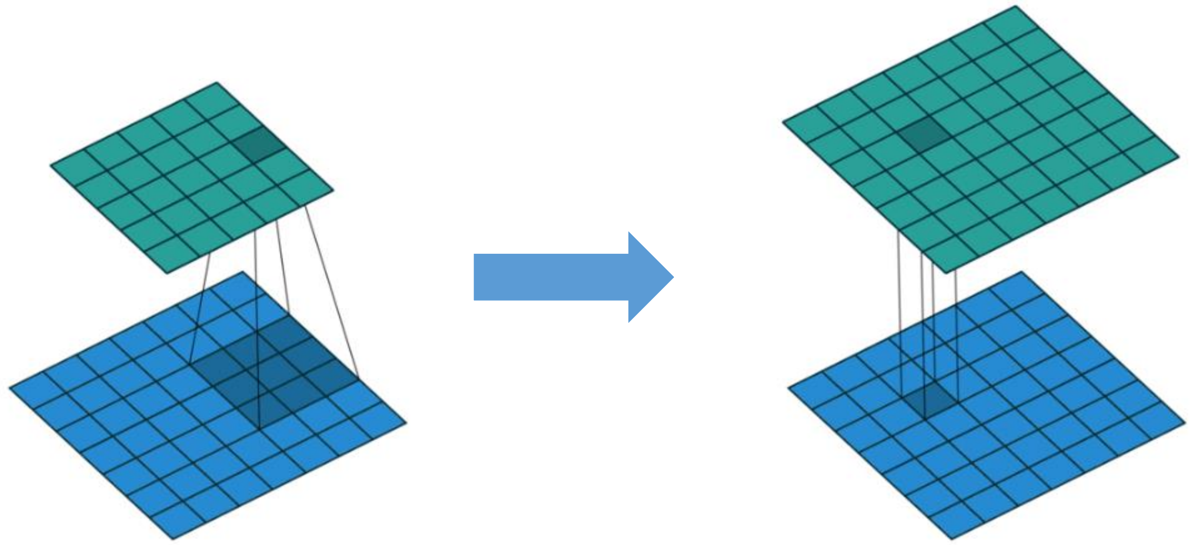


Visualization (cont.)

- Summary
  - The filters in the deeper layers characterize more abstract patterns
  - Layers that are deeper in the network visualize more training data specific features
  - While the earlier layers tend to visualize general patterns like edges, texture, background

1×1 convolution

- Here we introduce a special filter, which as a size of 1×1

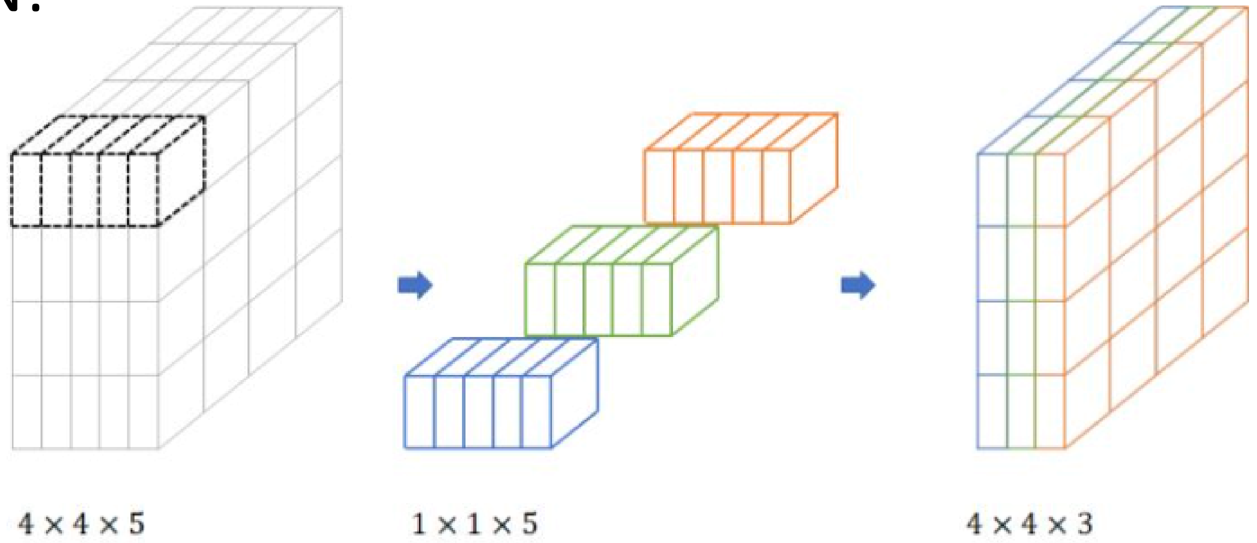


Convolution with large kernel Convolution with 1*1 kernel

- The 1×1 convolution cannot detect edges with any shape, so is it really useful? Or is it redundant?

1×1 convolution (cont.)

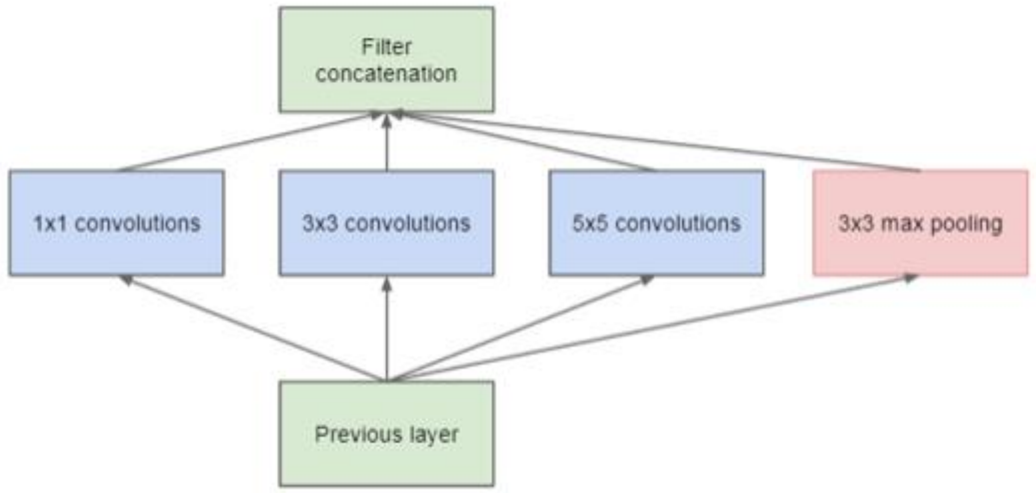
- The 1×1 convolution is very useful as it can reduce the computation complexity in CNN!



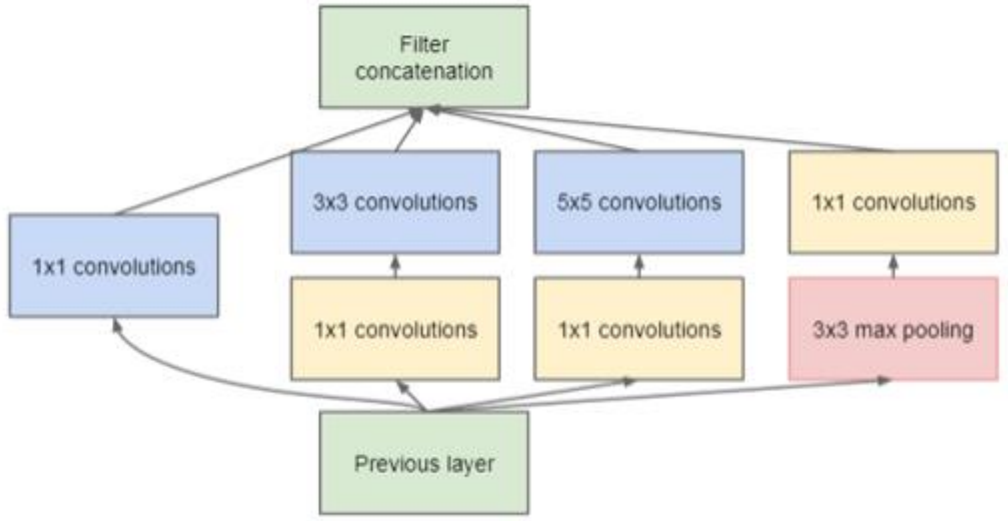
- Help reduce the number of channels
- In the example above, the size of the input data is reduced from $4 \times 4 \times 5$ to $4 \times 4 \times 3$
- Usually we assume the depth of the filter is the same with depth of data

1×1 convolution (cont.)

- 1×1 convolution filter is usually followed by 3×3 or other bigger filters. In this way, the computational complexity is greatly reduced
- This architecture is used in Google's inception model



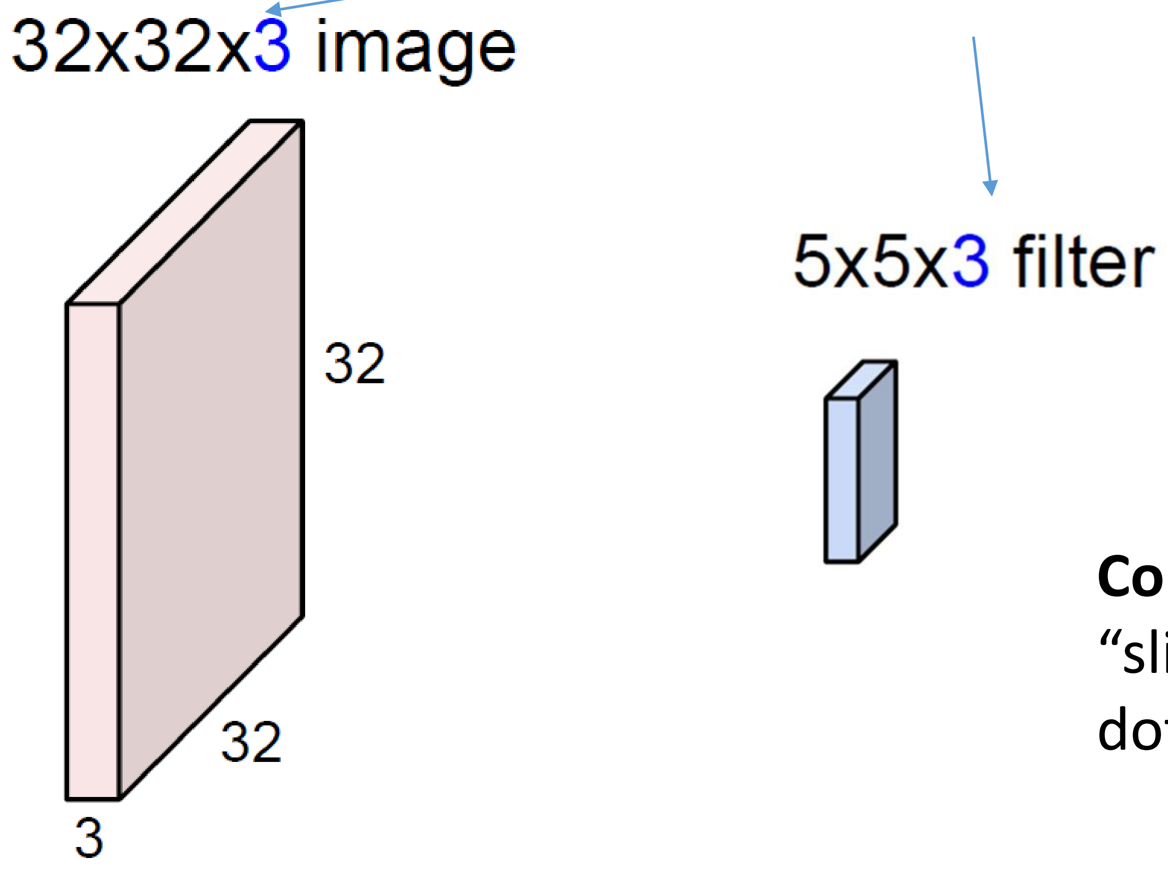
(a) Inception module, naïve version



(b) Inception module with dimension reductions

Filter depth

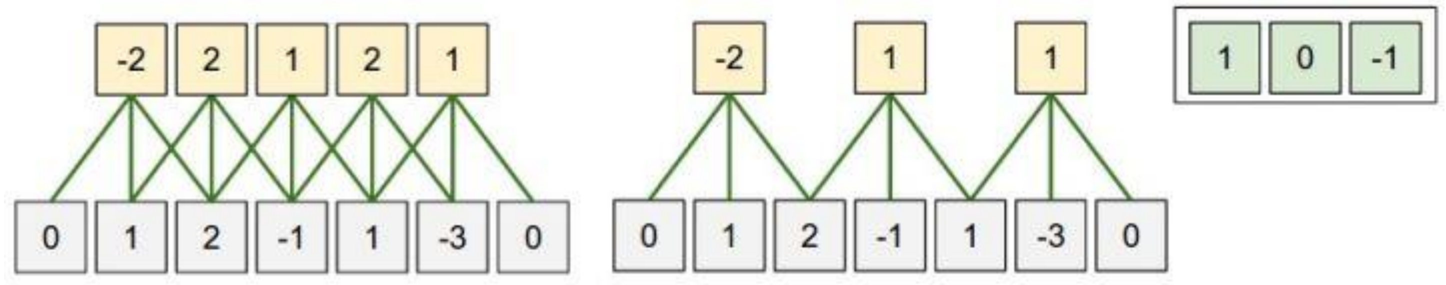
filters always extend the full depth of the input volume



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Stride

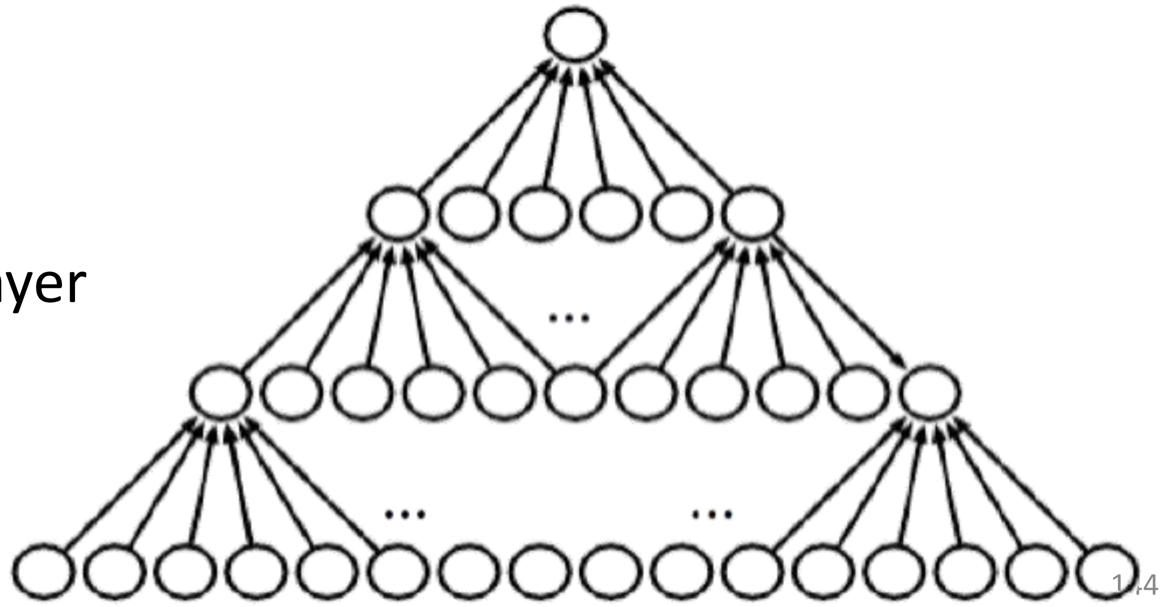
- The distance that the filter is moved in each step
- Examples of stride=1 and stride=2



Padding

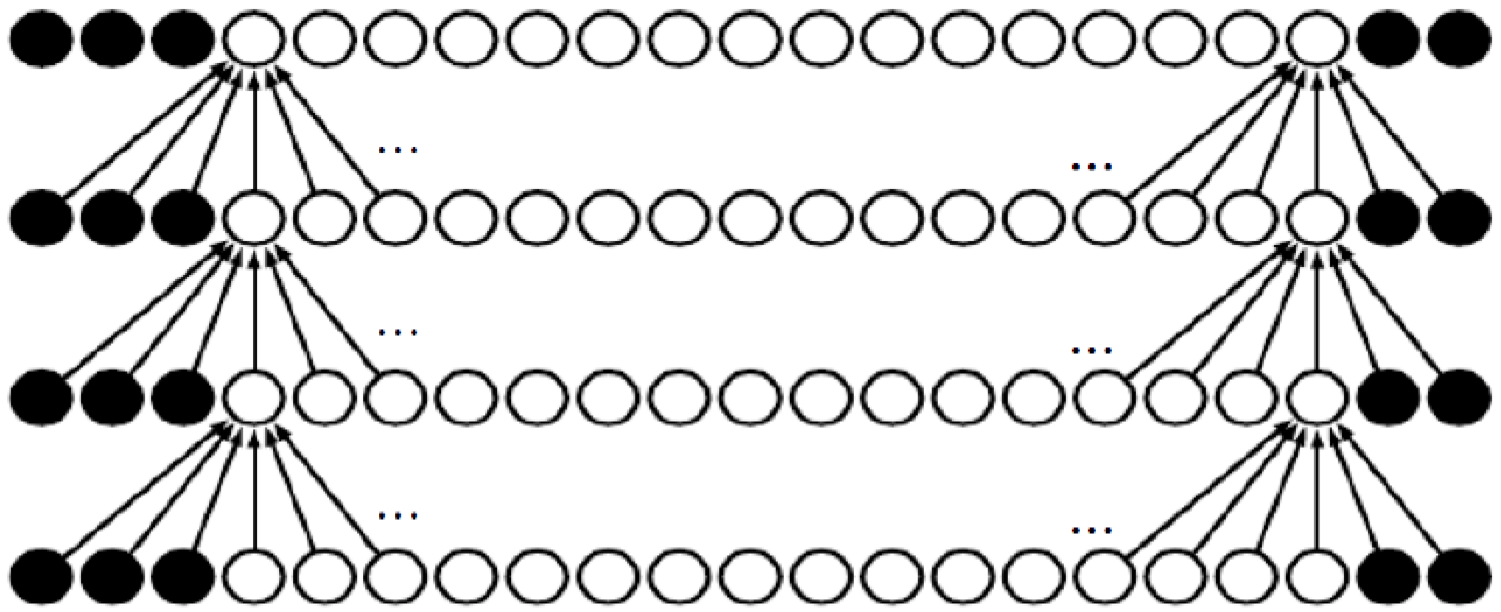
- A solution to the problem of data shrinking
- Data shrinking: as convolution can only happen within the border of the input data, the size of the data will become smaller as the network becomes deeper

- 1D Example:
  - suppose the filter width is 6,
    the data will shrink 5 pixel each layer

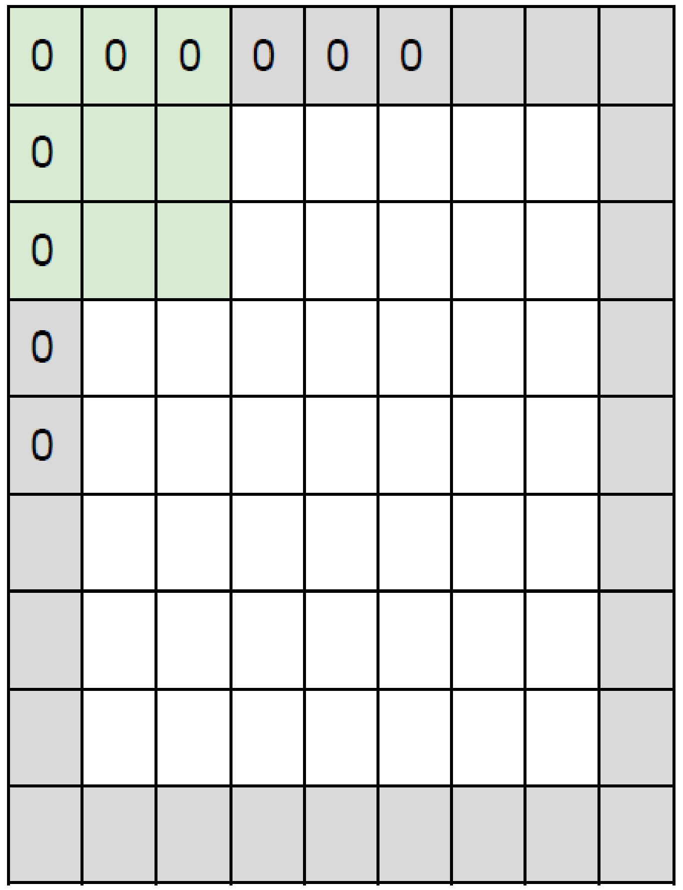


Padding (cont.)

- Add numbers (usually zero, called zero padding) around the input data to make sure that the size of the output data is the same as that of the input data
- Left padding = 3, right padding = 2
- Usually left padding + right padding = filter width – 1

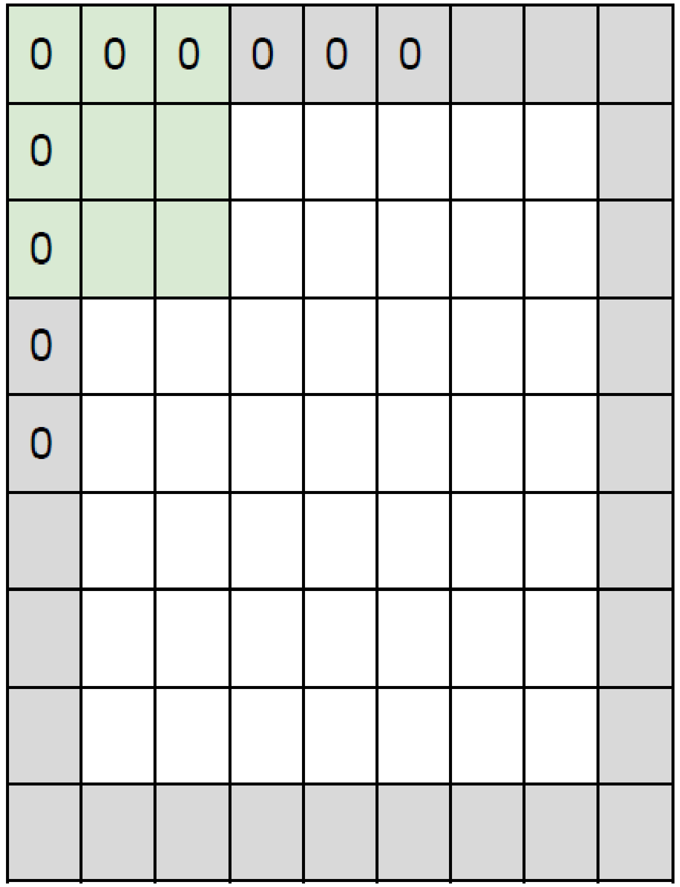


Example



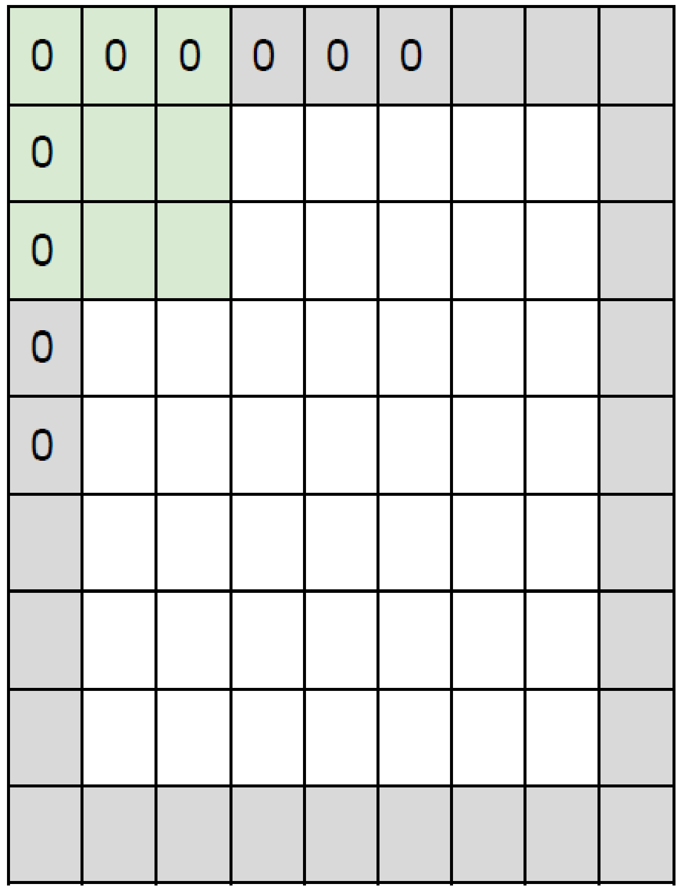
- Input 7x7 (white area)
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border (dark area)
- => what is the size of the output?

Example (cont.)



- Input 7x7 (white area)
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border (dark area)
- => what is the size of the output?
- **7x7 output!**

Example (cont.)



- Input 7x7 (white area)
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border (dark area)
- => what is the size of the output?
- **7x7 output!**
- In general, convolution layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
- e.g. F = 3 => zero pad with 1

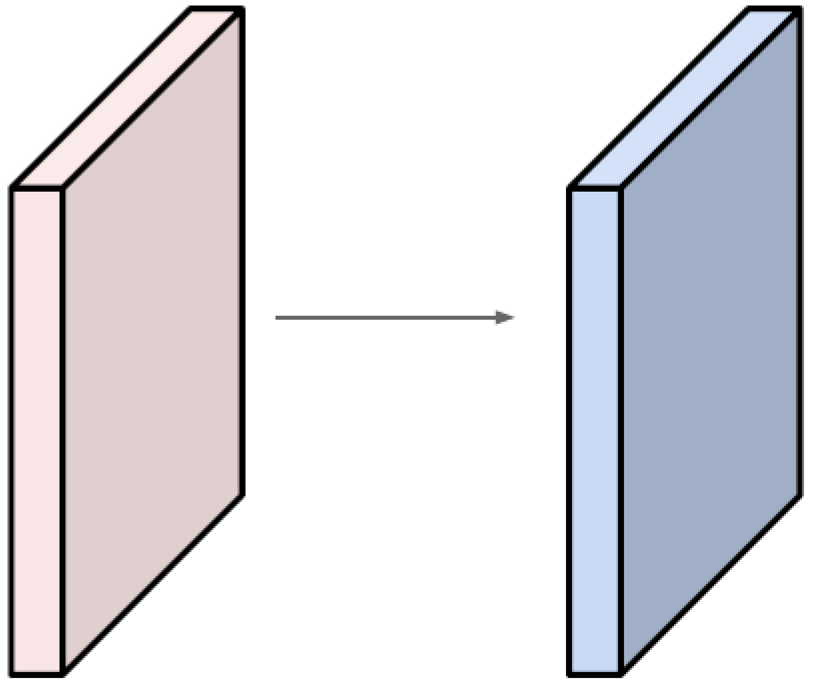
  F = 5 => zero pad with 2

  F = 7 => zero pad with 3

Example 2

- Input volume: $32 \times 32 \times 3$
- 10 5×5 filters with stride 1, pad 2

- Output volume size: ?

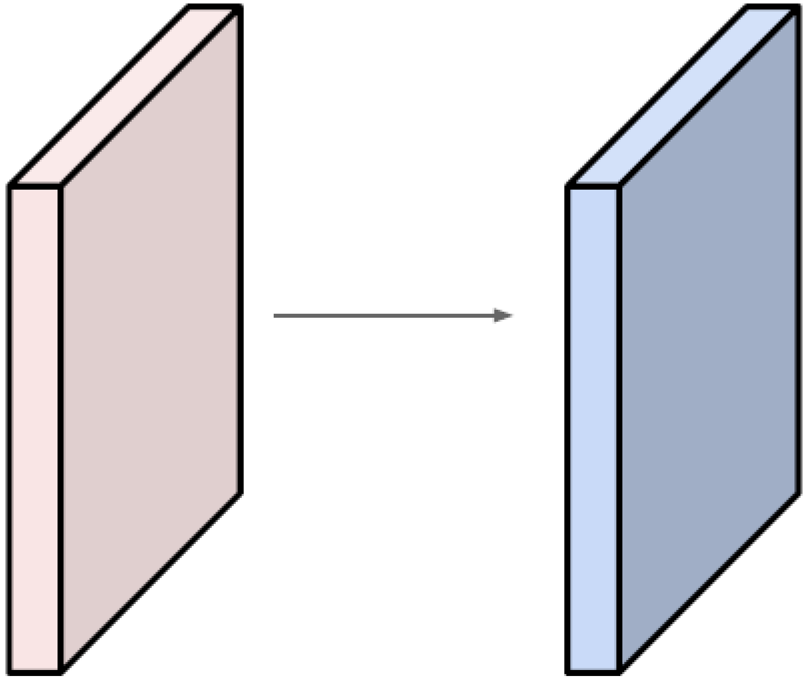


Example 2 (cont.)

- Input volume: $32 \times 32 \times 3$
- 10 5×5 filters with stride 1, pad 2

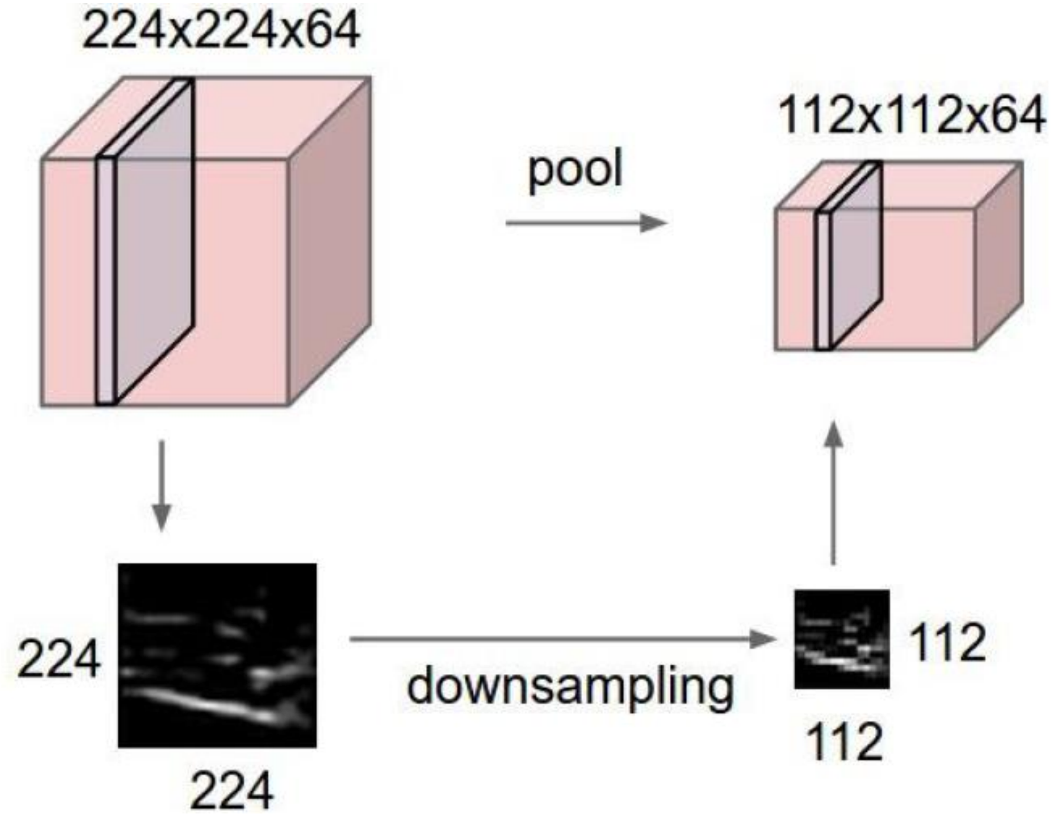
- Output volume size: ?

- Filter size 5, pad = $(F - 1)/2$, so keep the size 32×32 spatially
- 10 5×5 filters means 10 $5 \times 5 \times 3$ filters
- So $32 \times 32 \times 10$



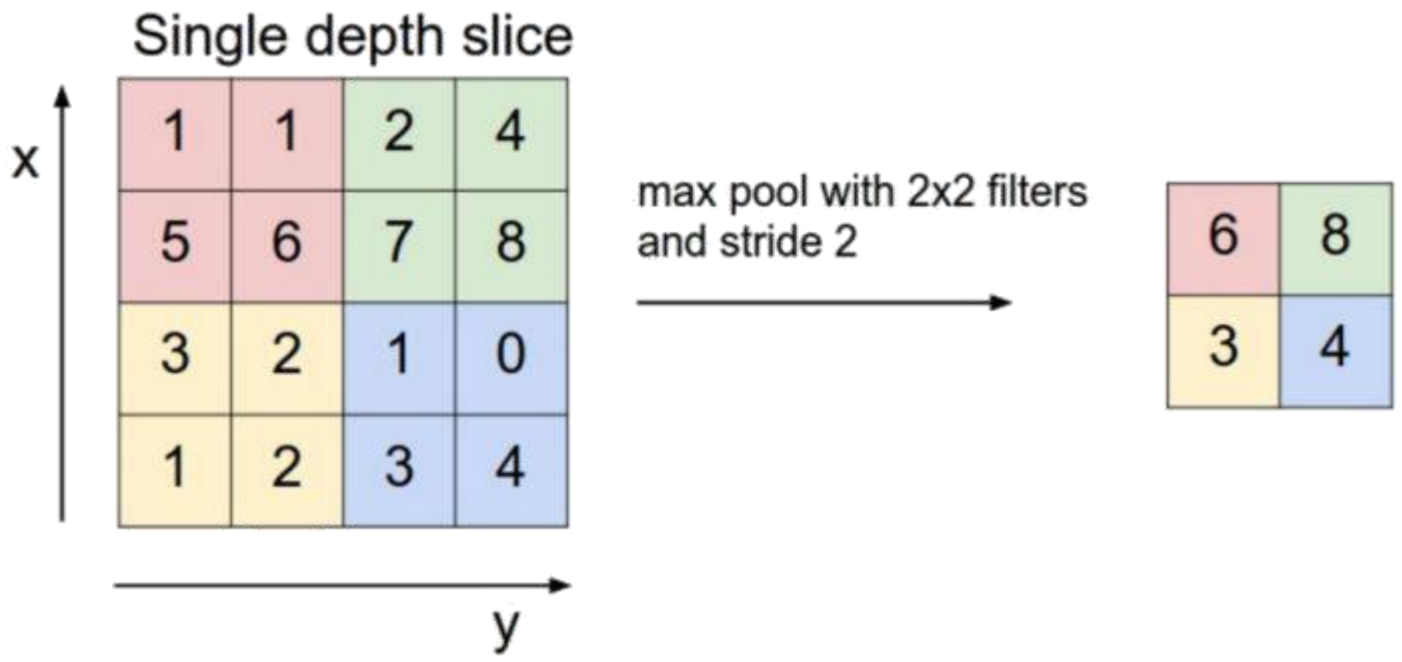
Pooling layer

- Make the representations denser and more manageable
- Operate over each activation map independently:



Example of pooling layer

- Pooling of size 2×2 with stride 2

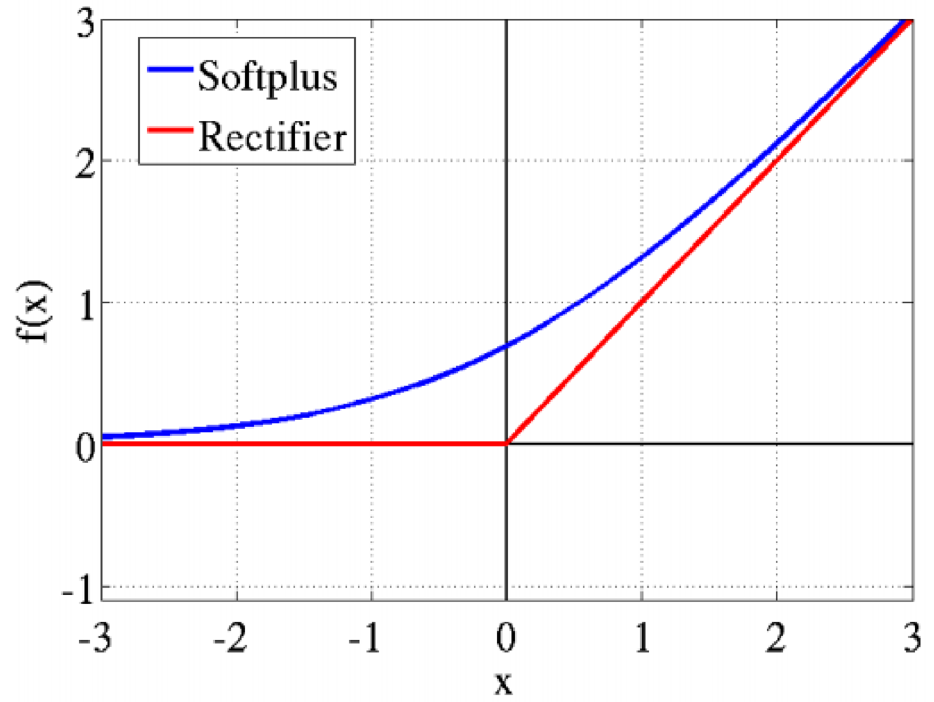


- Most common pooling operations are max and average

ReLU activation function

- ReLU (Rectified linear unity) function

$$\mathrm{ReLU}(z) = \max(0, z)$$



- Its derivative

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

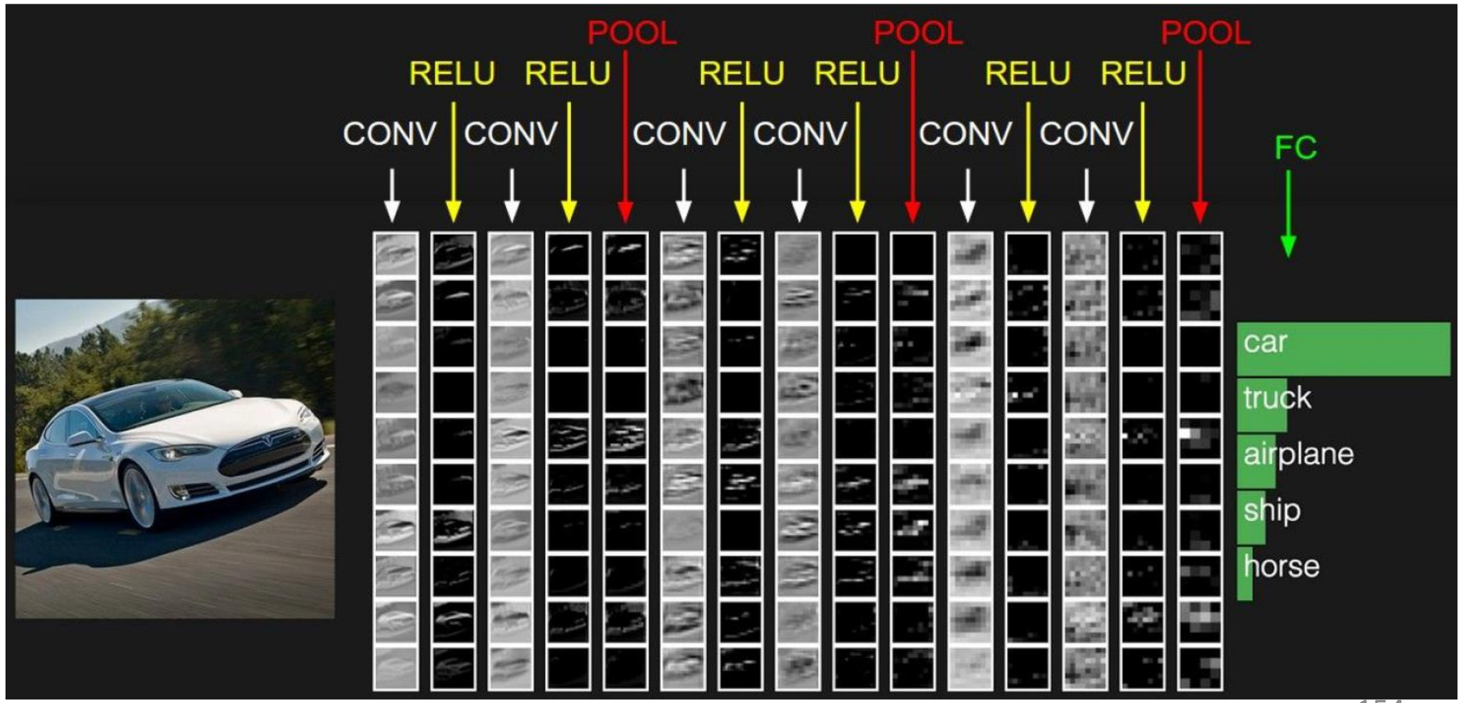
- ReLU can be approximated by softplus function

$$f_{\mathrm{Softplus}}(x) = \log(1 + e^x)$$

- ReLU's gradient doesn't vanish as x increases
- Speed up training of neural networks
  - Since the gradient computation is very simple
  - The computational step is simple, no exponentials, no multiplication or division operations (compared to others)
- The gradient on positive portion is larger than sigmoid or tanh functions
  - Update more rapidly
  - The left "dead neuron" part can be ameliorated by Leaky ReLU

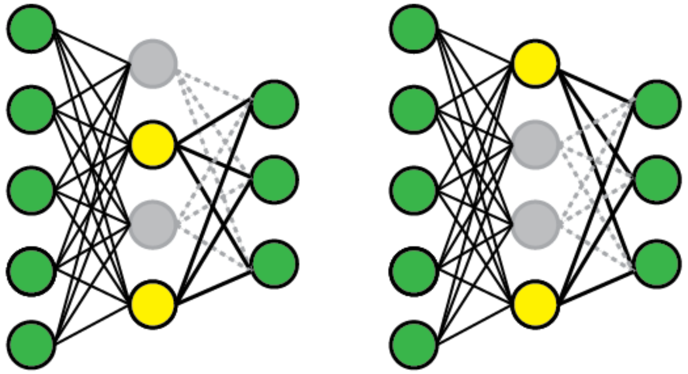
A typical CNN structure

- Convolution/Activation (ReLU)/Pooling layers appear in flexible order and flexible repetitions

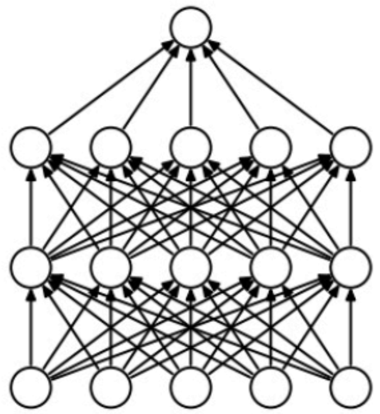


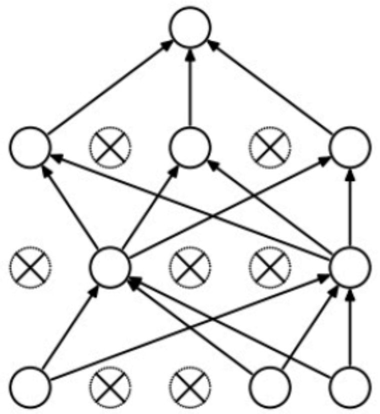
Training Techniques

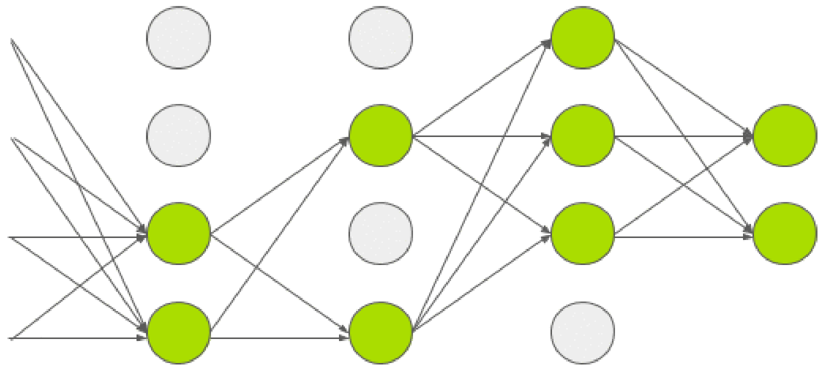
Dropout



- Dropout randomly 'drops' units from a layer on each training step, creating 'sub-architectures' within the model
- It can be viewed as a type of sampling a small network within a large network
- Prevent neural networks from overfitting


(a) Standard Neural Net

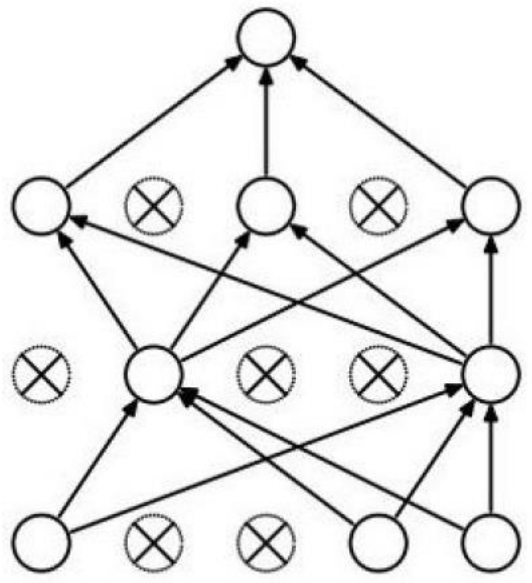

(b) After applying dropout.

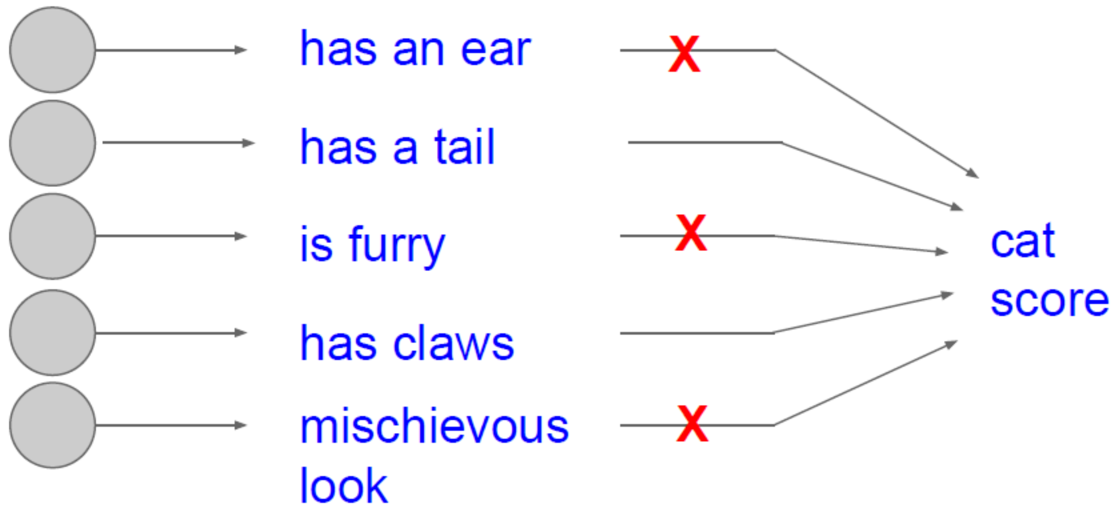


Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Dropout (cont.)

- Forces the network to have a redundant representation
- Increase robustness in prediction power





Weights initialization

- If the weights in a network start too small,
  - then the signal shrinks as it passes through each layer until it’s too tiny to be useful
- If the weights in a network start too large,
  - then the signal grows as it passes through each layer until it’s too massive to be useful

Weights initialization (cont.)

- All zero initialization

- Small random numbers

- Draw weights from a Gaussian distribution
  - with the standard deviation of $\sqrt{\frac{2}{n}}$
    - n is the number of inputs to the ending neuron

Batch normalization

- Batch training:
  - Given a set of data, each time a small portion of data are put into the model for training
- Extreme example
  - Suppose we are going to learn some pattern of people, and the input data are people’s weights and heights
  - Unluckily, women and men are divided into two batches when we randomly split the data
  - As the weights and heights of women are very different from these of men, the neural network have to make huge changes to the weight when we switch the batch during training, which will cause slow convergence or even divergence

Batch normalization (cont.)

- The problem in the example is called *Internal Covariate Shift*
- The solution to Internal Covariate Shift is batch normalization
- Suppose $Z_j^{(i)}$ is the ith input for the jth neuron in the input layer

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} Z_j^{(i)}$$

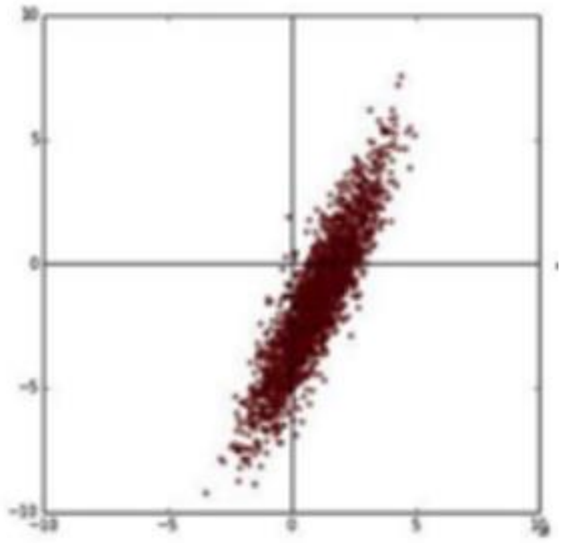
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(Z_j^{(i)} - \mu_j)^2$$

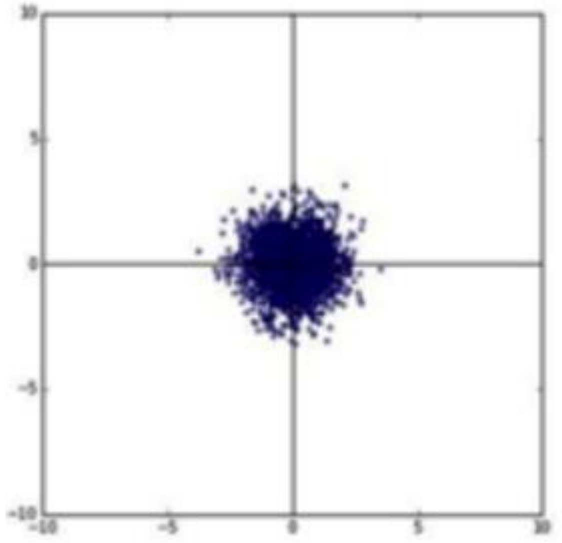
$$\hat{Z}_j = \frac{Z_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

- Or normalize the whole input layer together

Batch normalization (cont.)

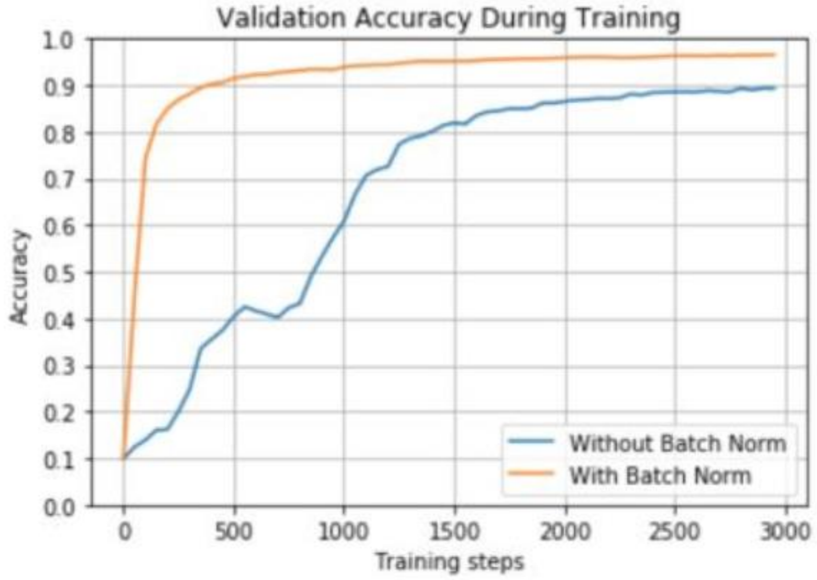
- 2D example

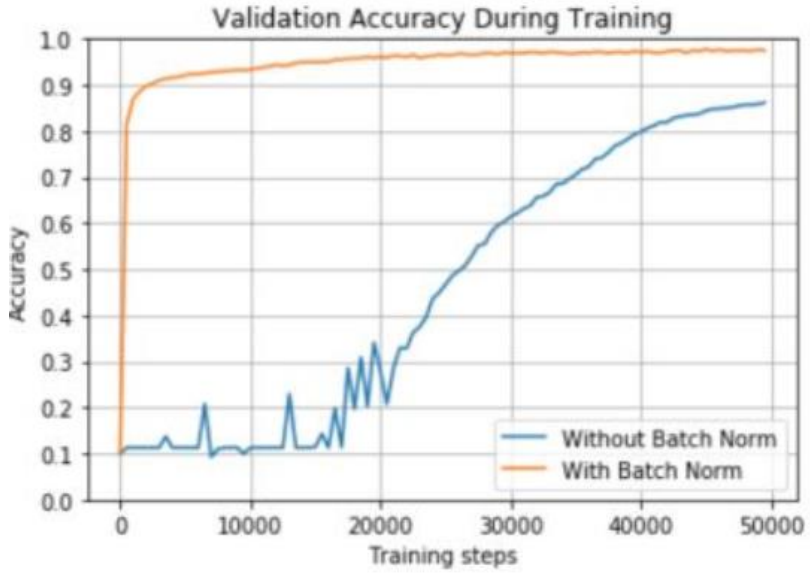




Example of batch normalization

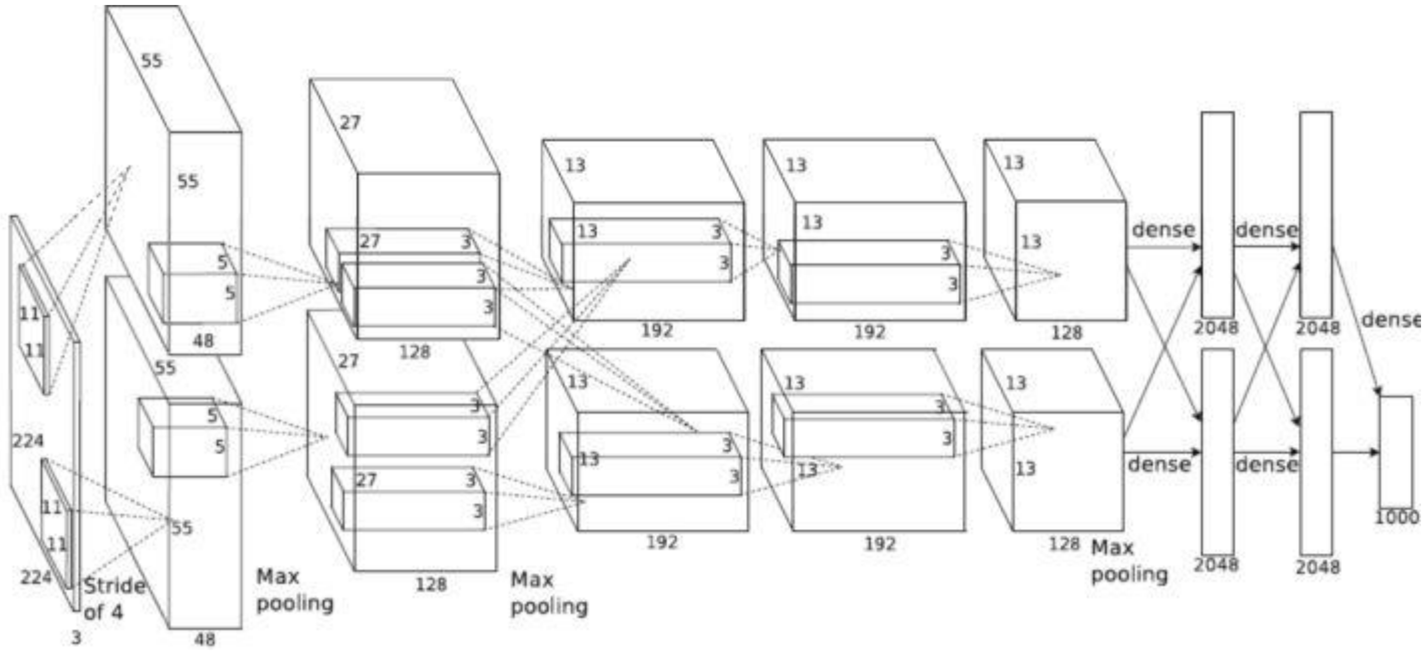
- Without batch normalization
  - Slow convergence and fluctuation





AlexNet [Krizhevsky et al. 2012]

- AlexNet:



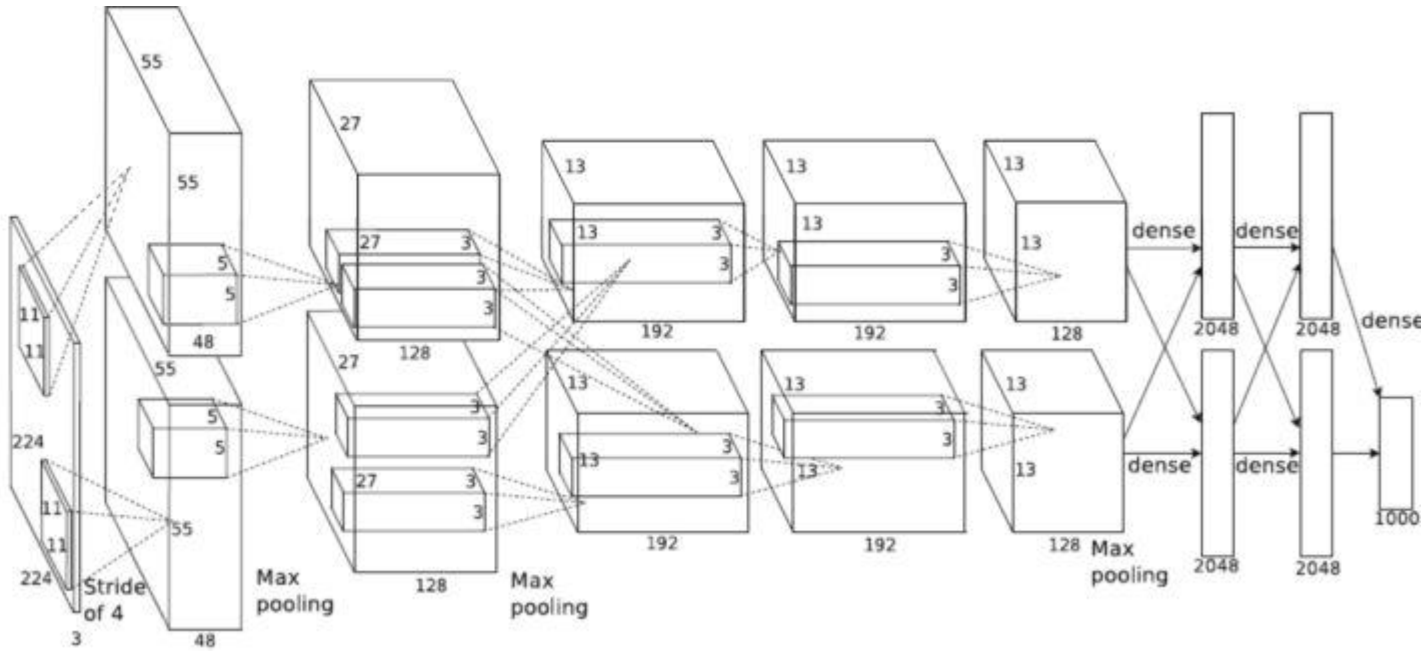
Input: 227x227x3 images

First layer (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size?

AlexNet (cont.)

- AlexNet:



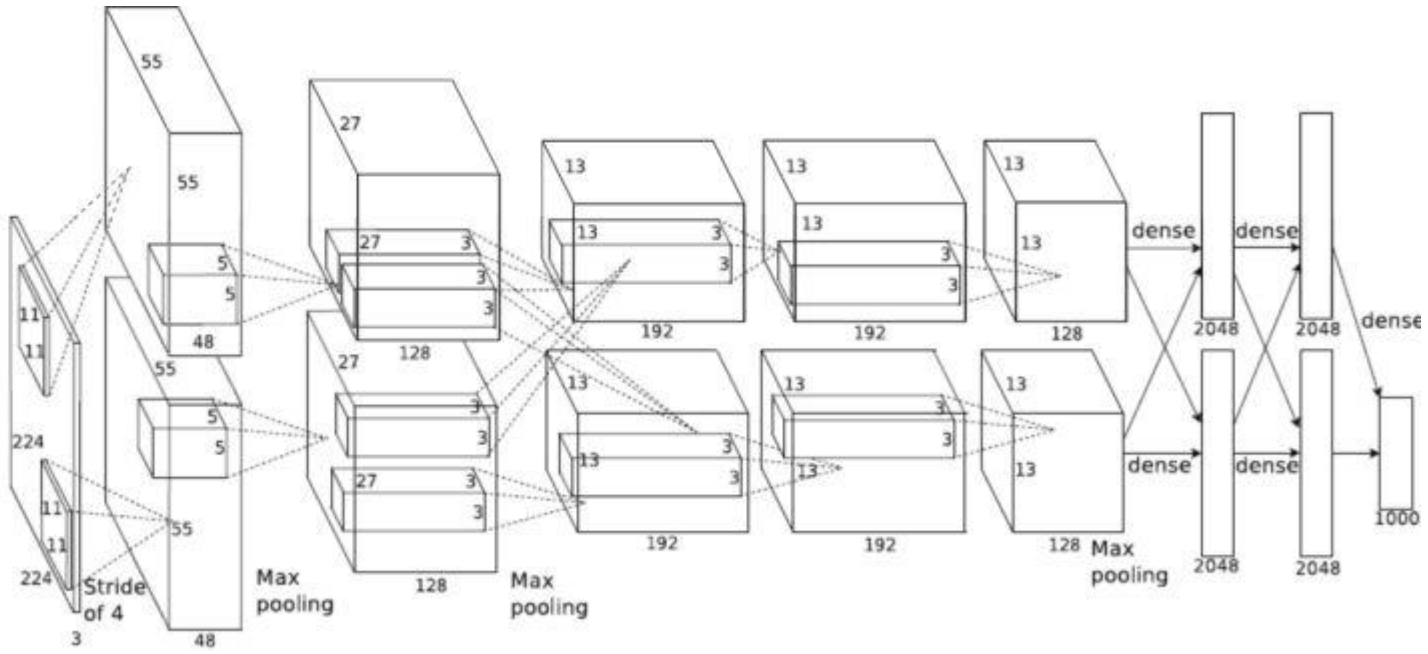
Input: 227x227x3 images

First layer (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55 **[55x55x96]**

AlexNet (cont.)

- AlexNet:



Input: 227x227x3 images

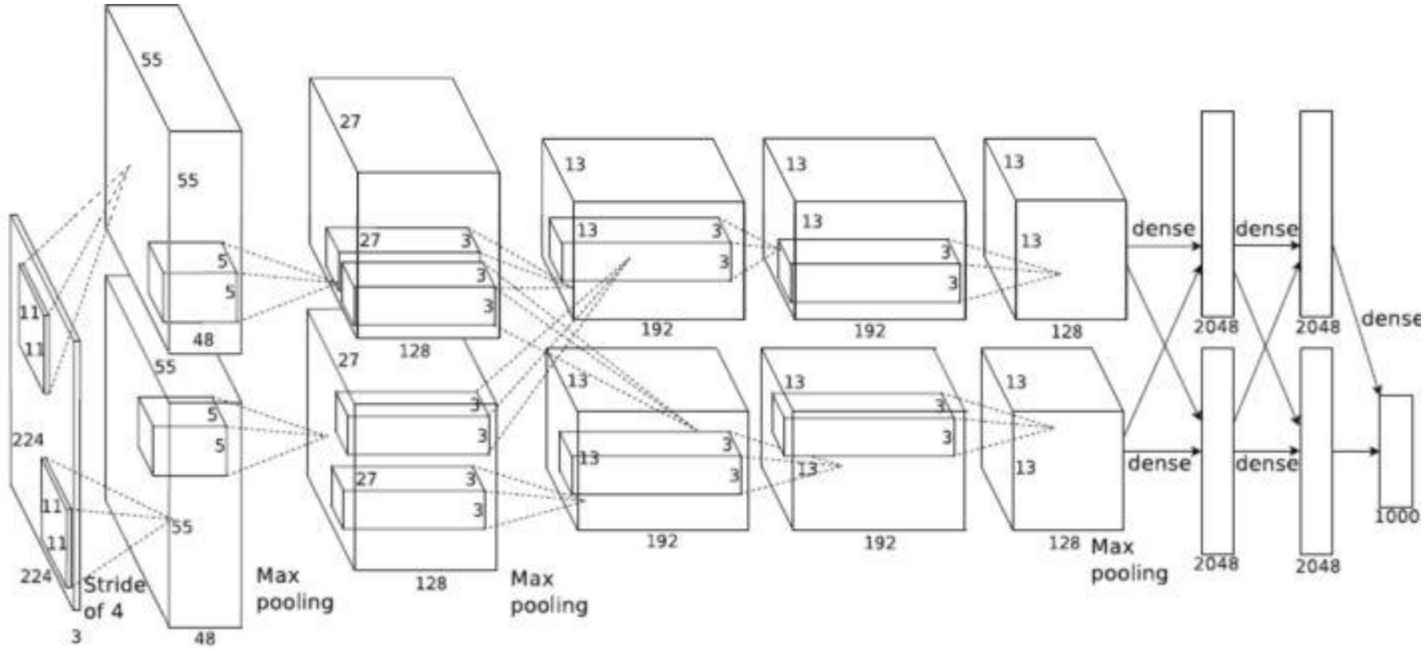
First layer (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55 **[55x55x96]**

Q: What is the total number of parameters in this layer?

AlexNet (cont.)

- AlexNet:



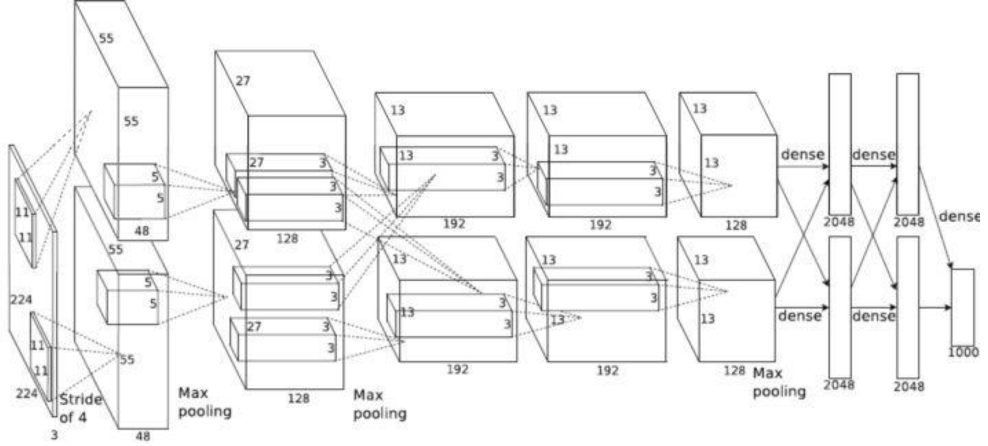
Input: 227x227x3 images

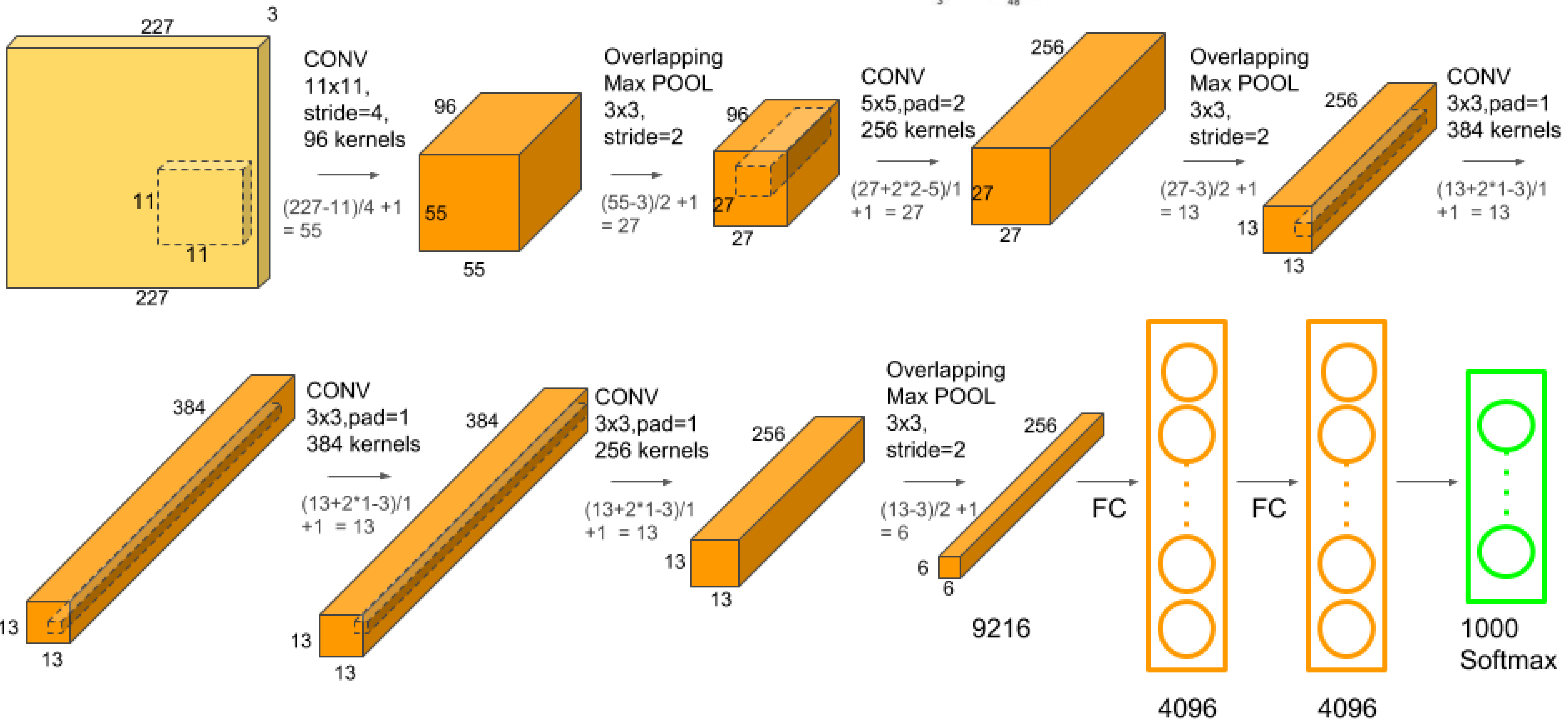
First layer (CONV1): 96 11x11 kernels applied at stride 4

Q: what is the output volume size? Hint: (227-11)/4+1 = 55 **[55x55x96]**

Q: What is the total number of parameters in this layer? (11*11*3)*96 = **35K**

Computation example

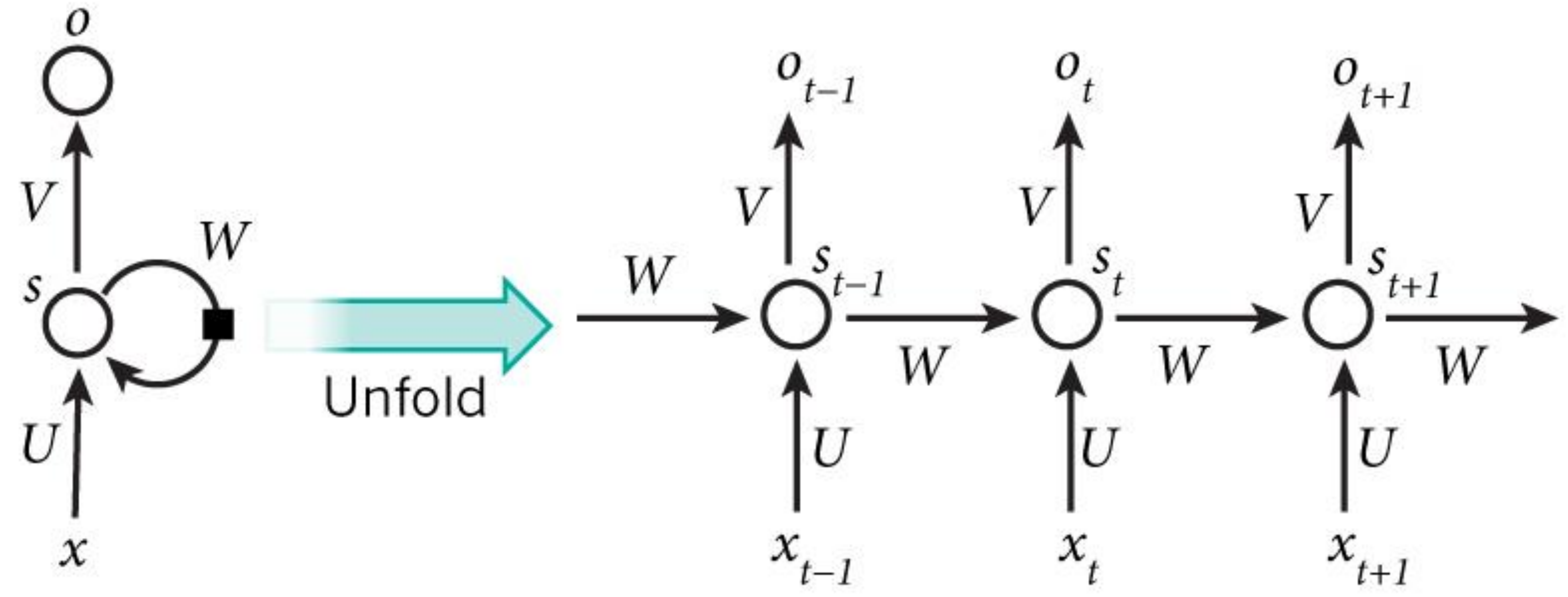




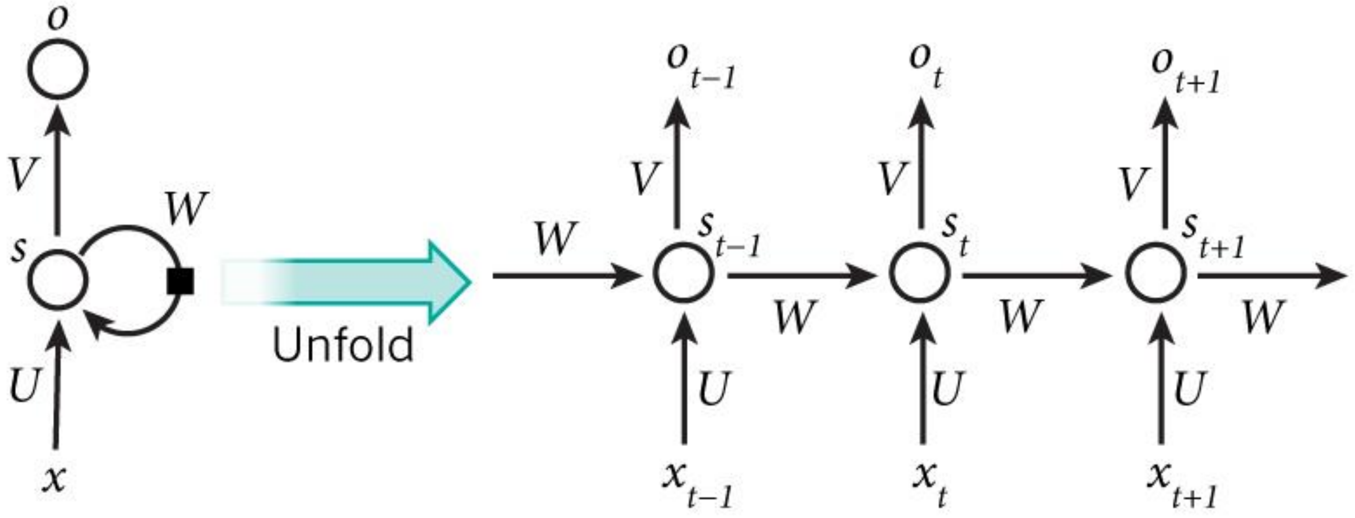
227
227
3
11
11
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

Recurrent Neural Networks

RNN

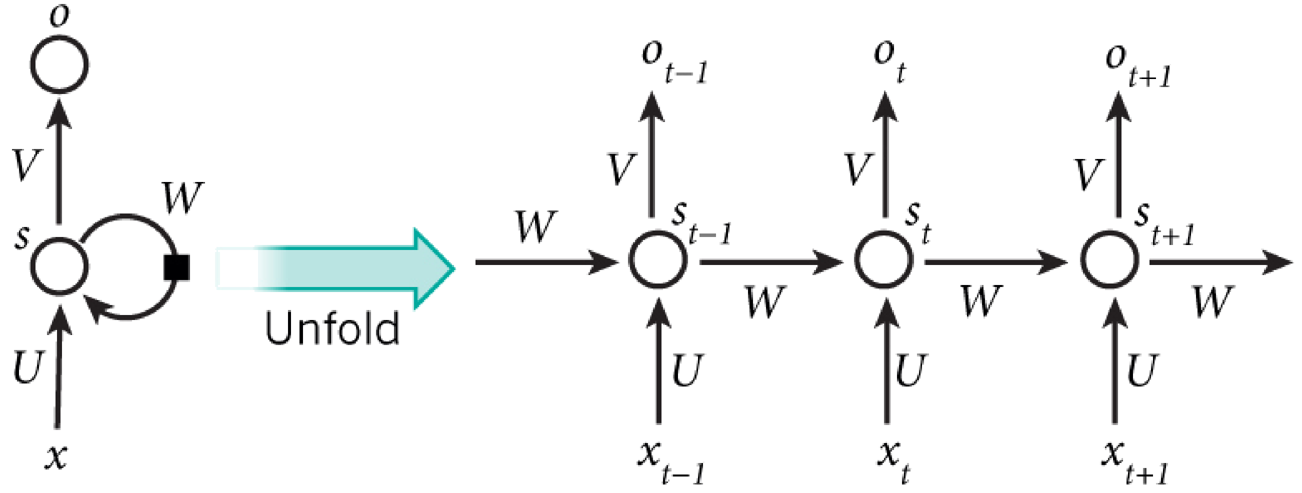


RNN (cont.)



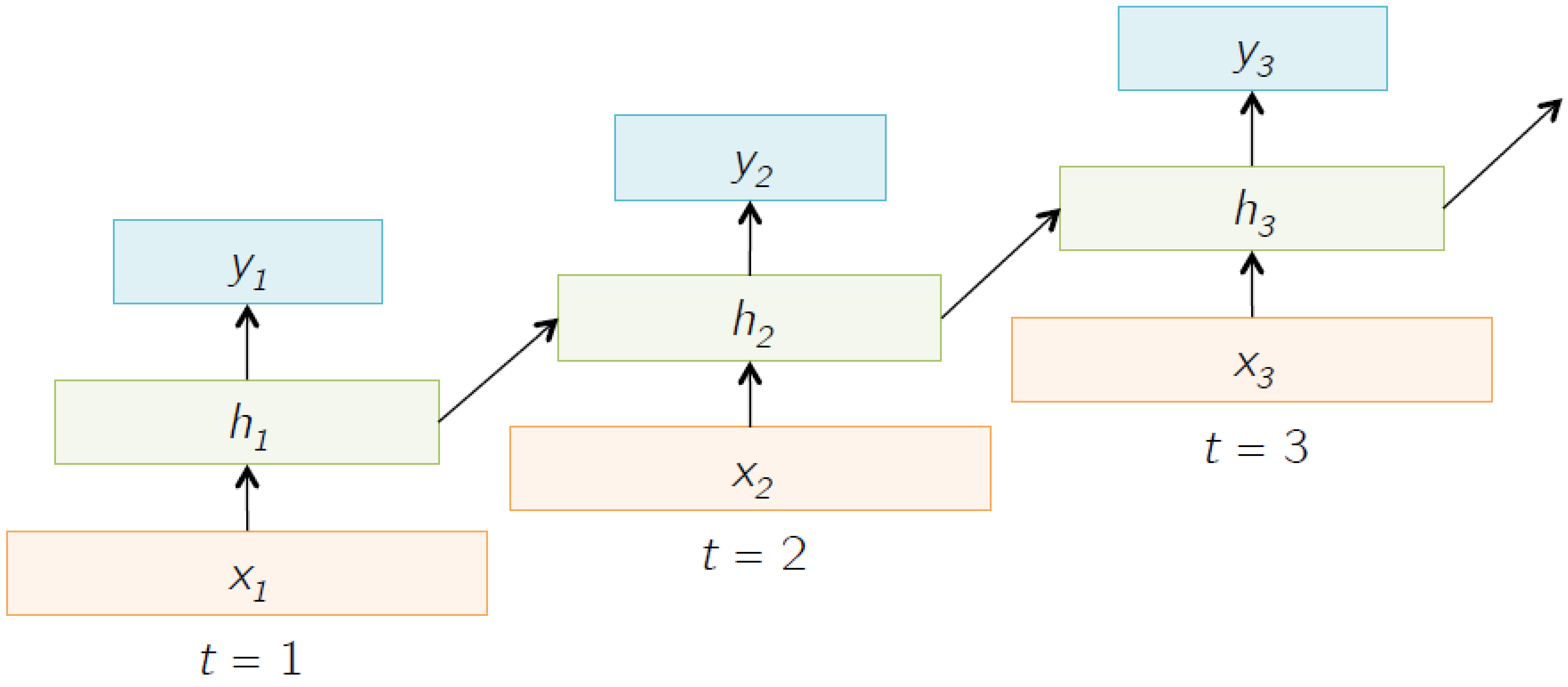
- x_t is the input at time t
- s_t is the hidden state at time t
  - It is the "memory" of the network
  - Is calculated based on previous hidden state and the input at the current step
    $$s_t = f(Ux_t + Ws_{t-1})$$
    where f is nonlinear activation function, such as tanh, ReLU
- o_t is the output at time t
  - E.g. If we want to predict the next word in a sentence, o_t is a vector of probabilities over certain vocabulary

RNN (cont.)



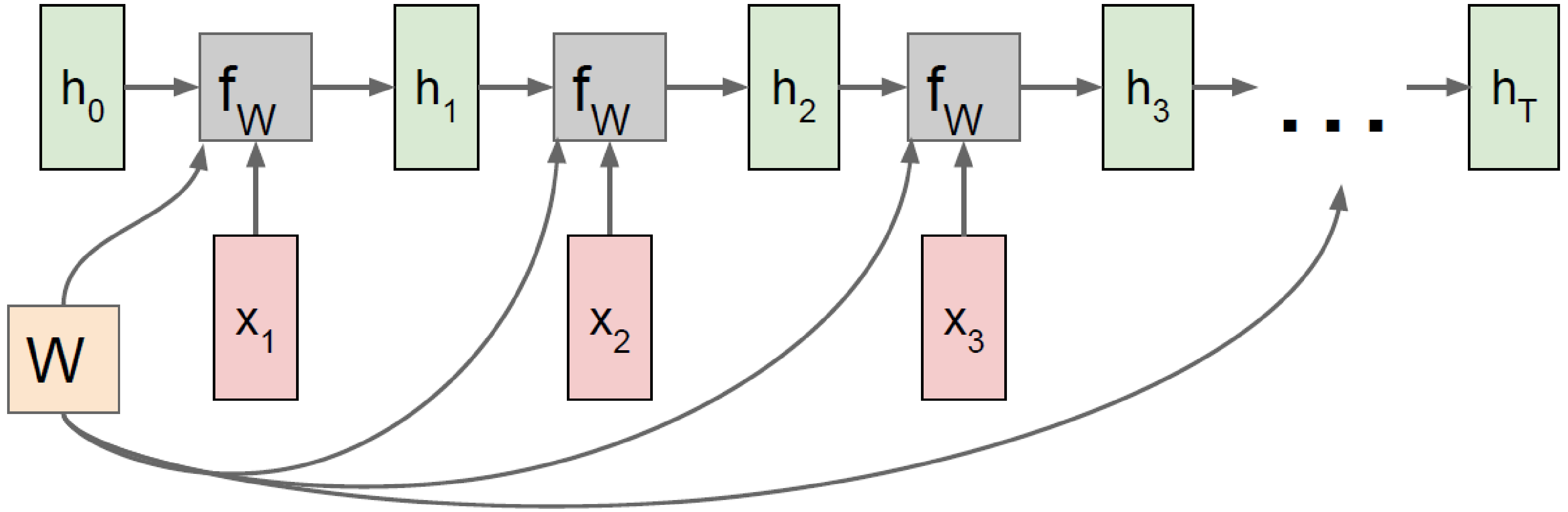
- s_t is the "memory" at time t
- o_t is based on memory at time t
- RNN share weights U and W
  - Unlike traditional NN
  - Greatly reduce the total number of parameters we need to learn
- The output at each time step might be unnecessary
  - E.g. When predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word
- We may not need inputs at each time step
- Main feature:
  - Is the hidden state, which captures some information about a sequence

Recall RNN

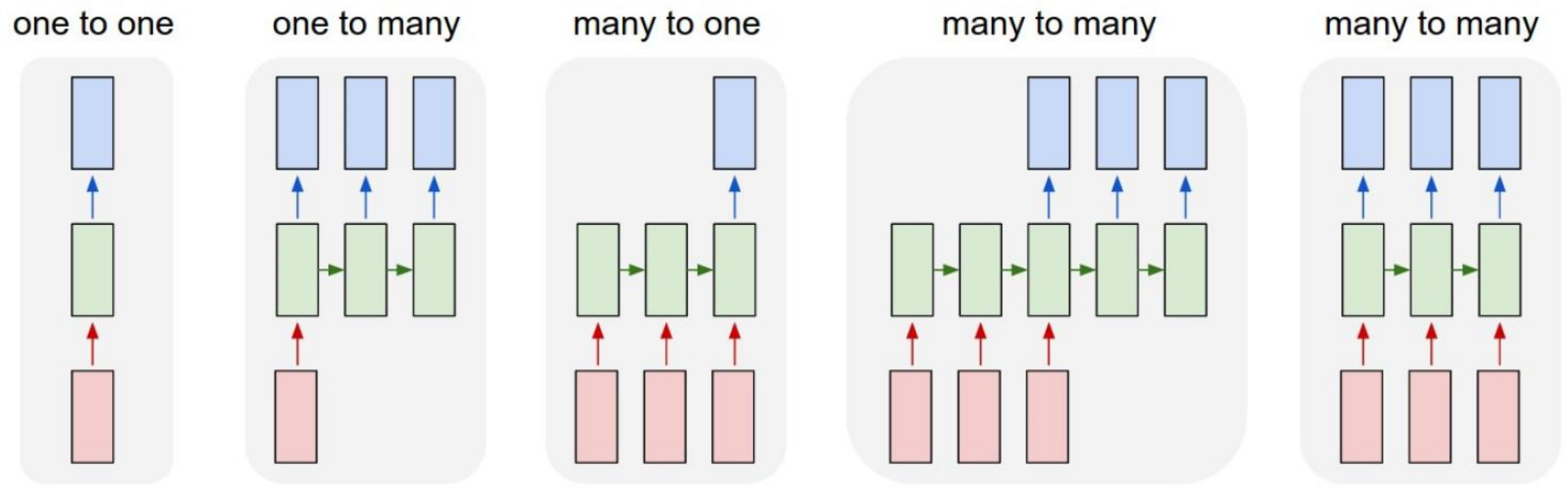


Computational graph of RNN

- Re-use the same weight matrix at every time-step



Different RNNs



Vanilla NN

Image captioning
Text generation

Text classification
Sentiment analysis

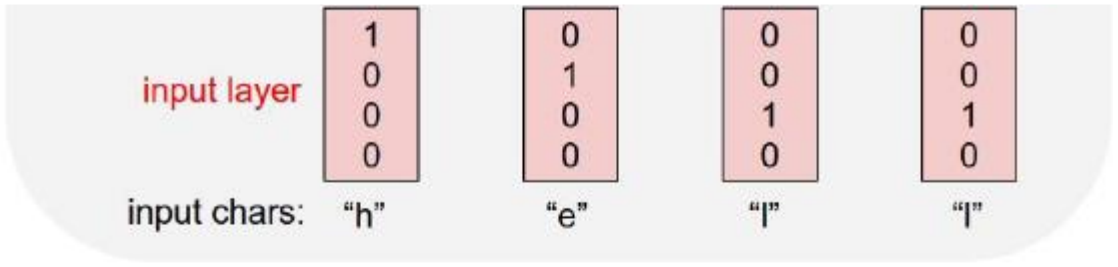
Machine translation
Dialogue system

Stock price estimation
Video frame classification

Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

  “Hello”

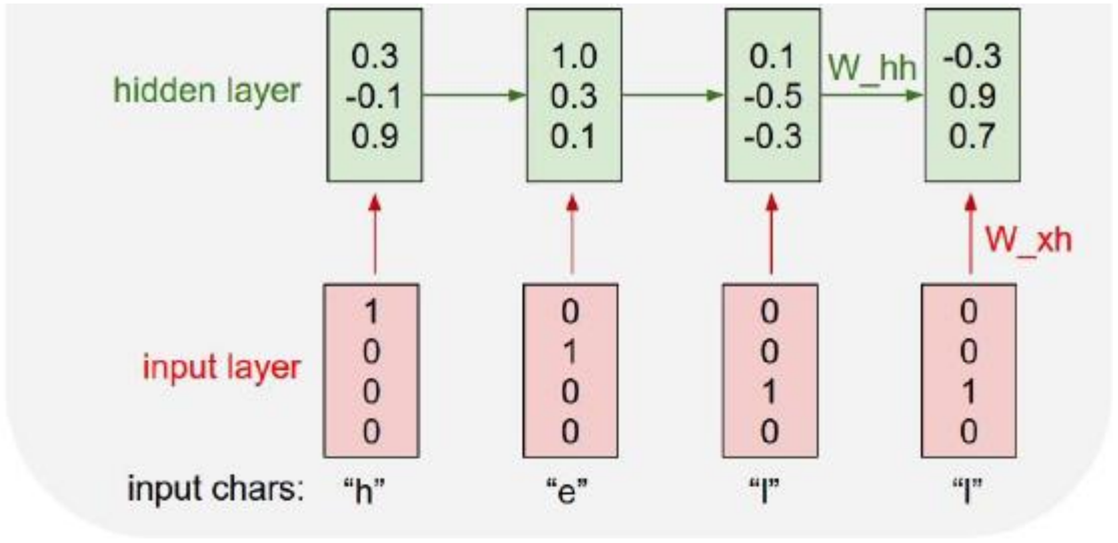


Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

  “Hello”

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



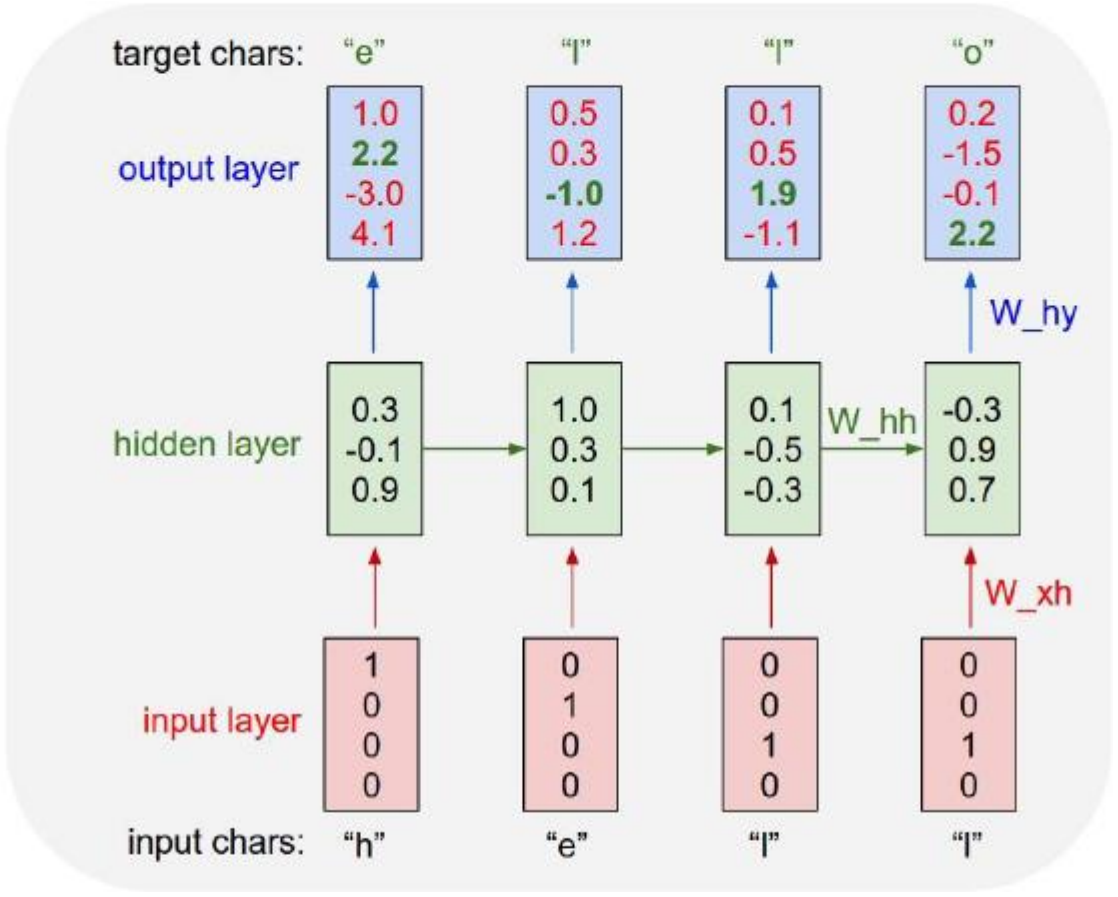
Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

  “Hello”

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

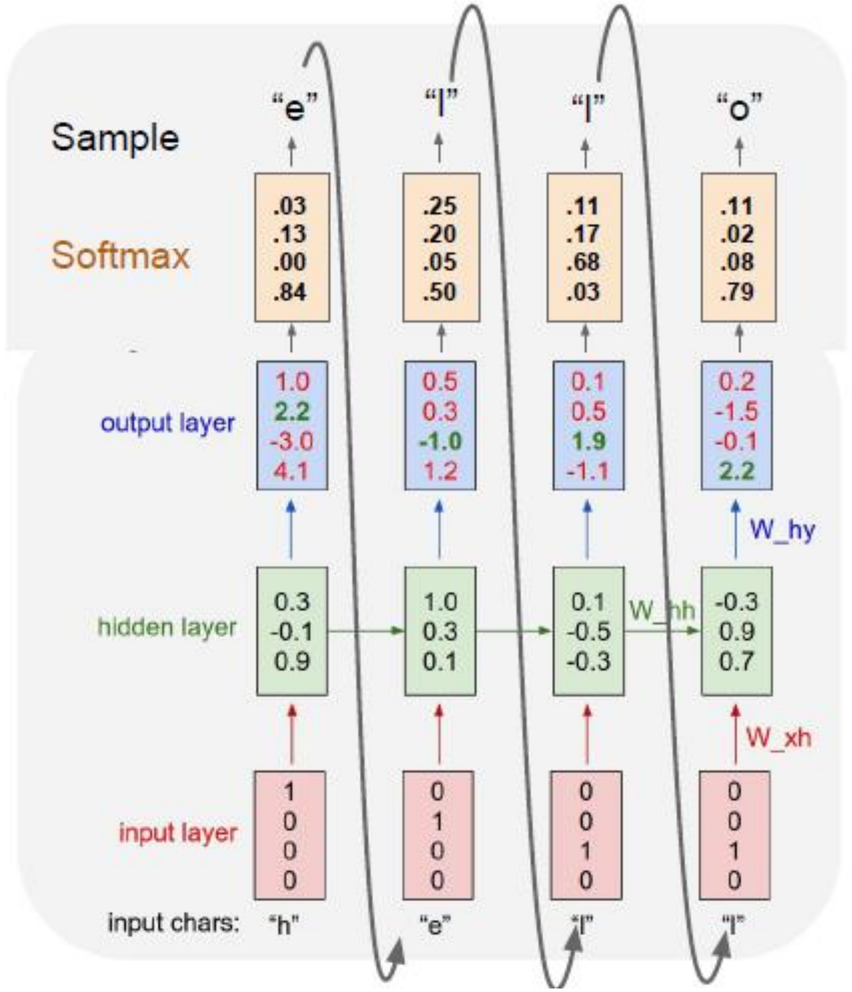


Example 1: Character-level language model

- Given previous words/characters, predict the next
- Vocabulary: [h, e, l, o]
- Example of training sequence:

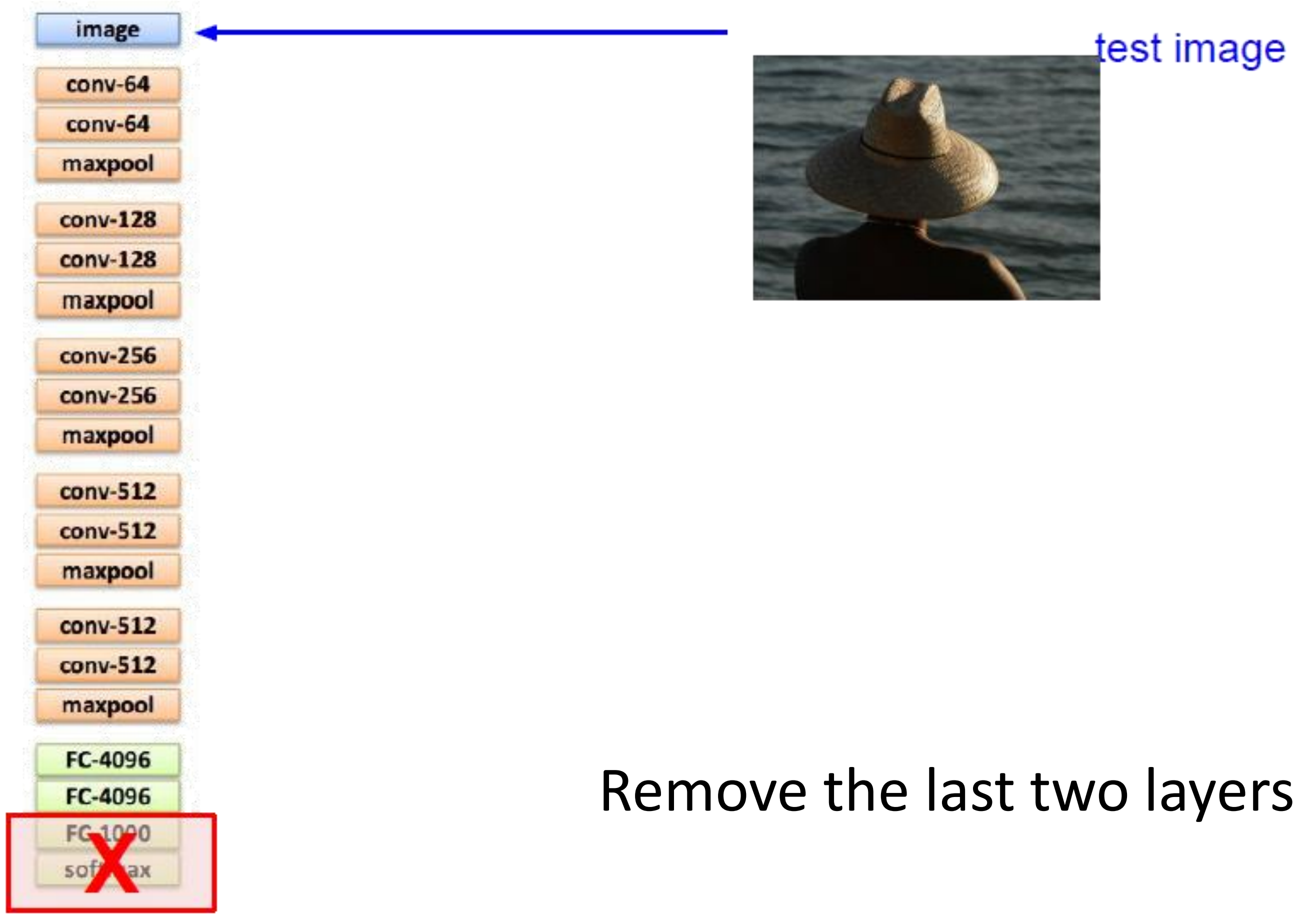
  “Hello”

- At test time, sample characters one at a time, feed back to model



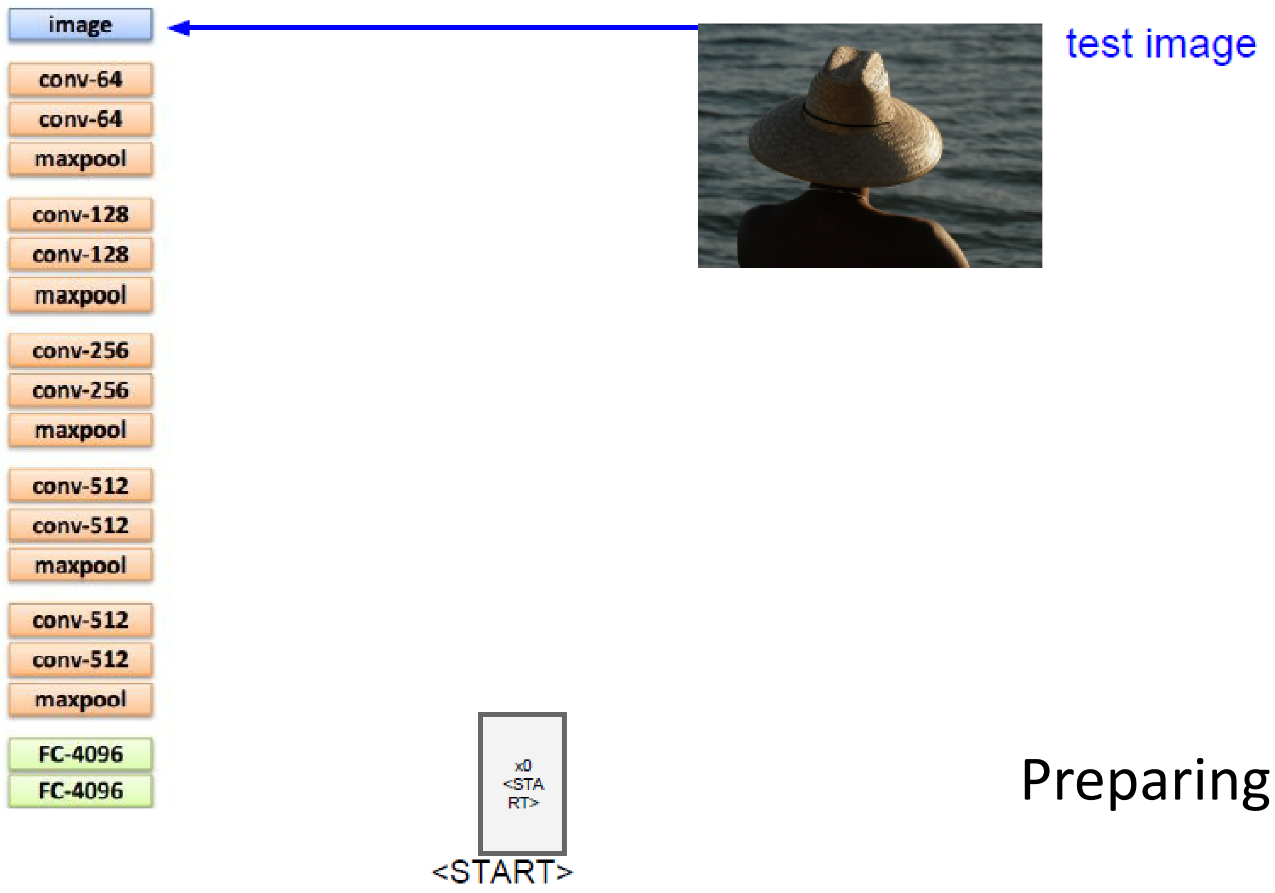
Example 3: Image captioning

- Using RNN to explain the content in an Image



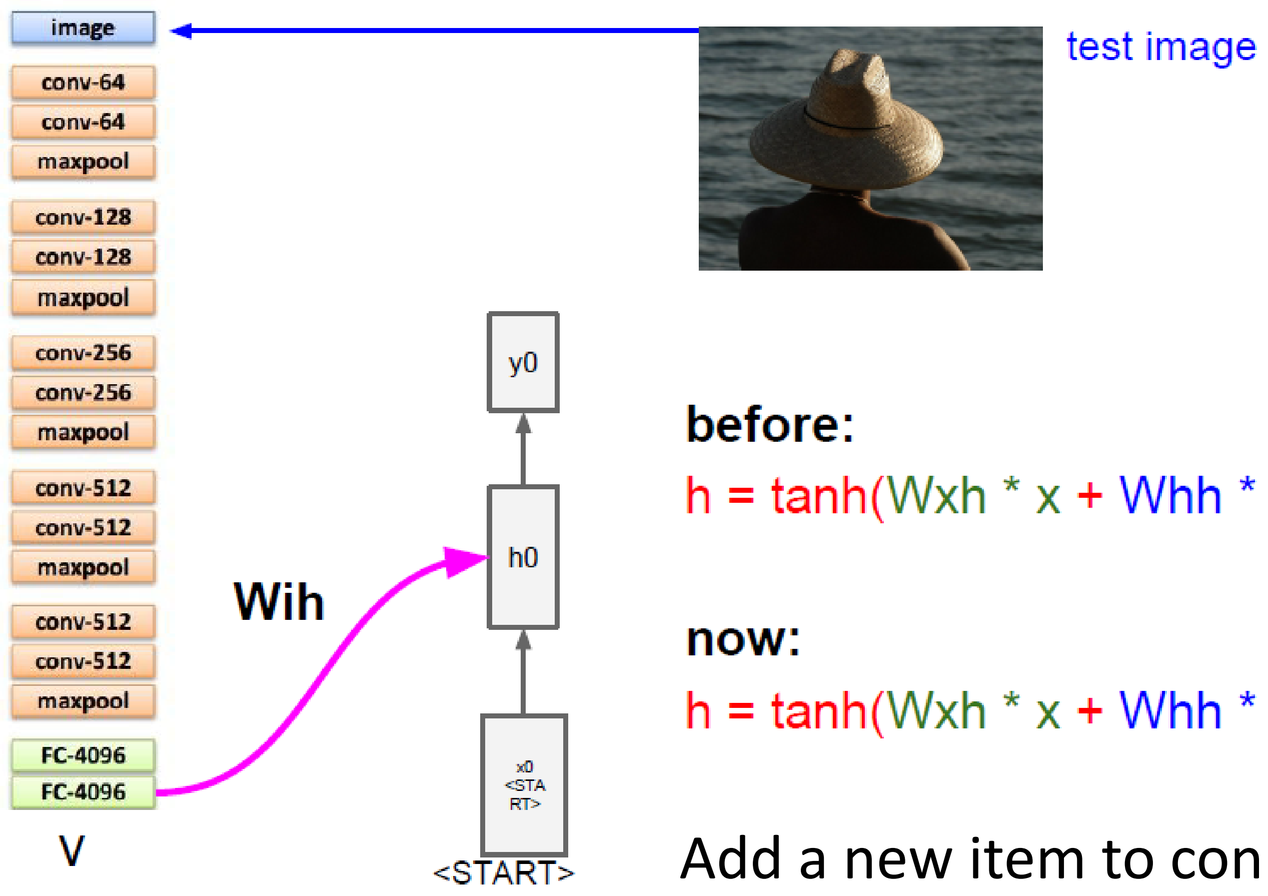
Remove the last two layers in CNN

Example 3: Image captioning (cont.)



Preparing the initial input for RNN

Example 3: Image captioning (cont.)



before:

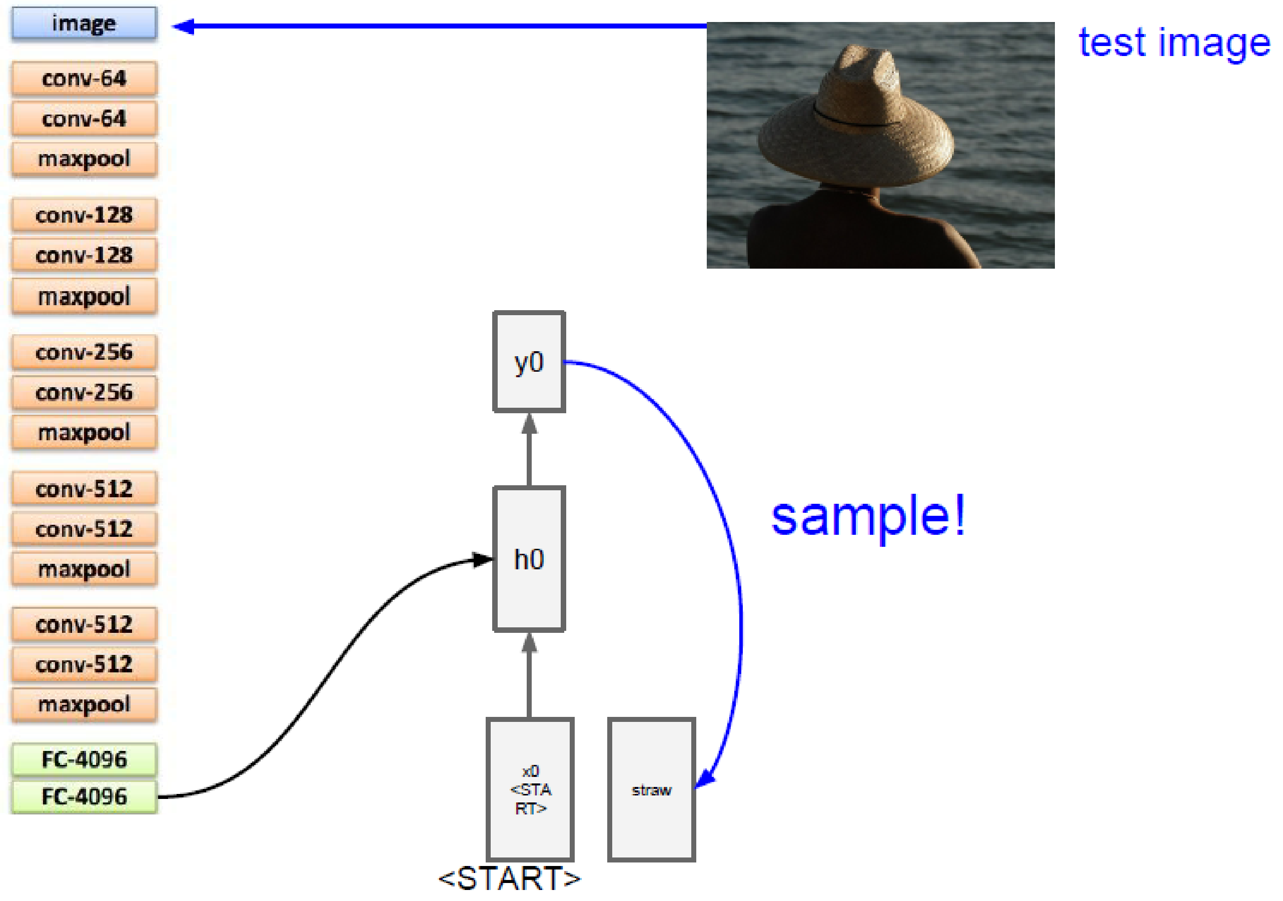
h = tanh(Wxh * x + Whh * h)

now:

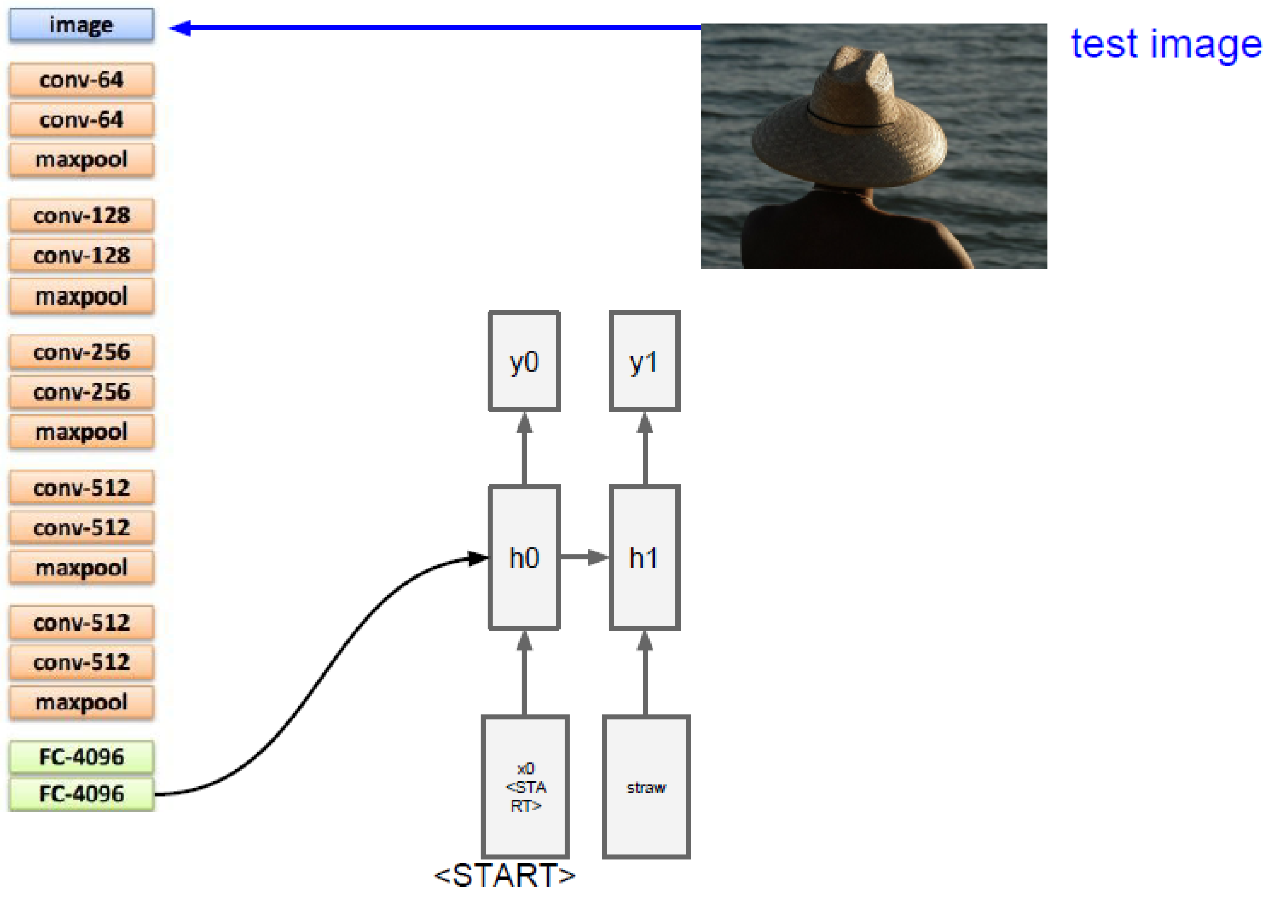
h = tanh(Wxh * x + Whh * h + **Wih * v**)

Add a new item to connect CNN to RNN

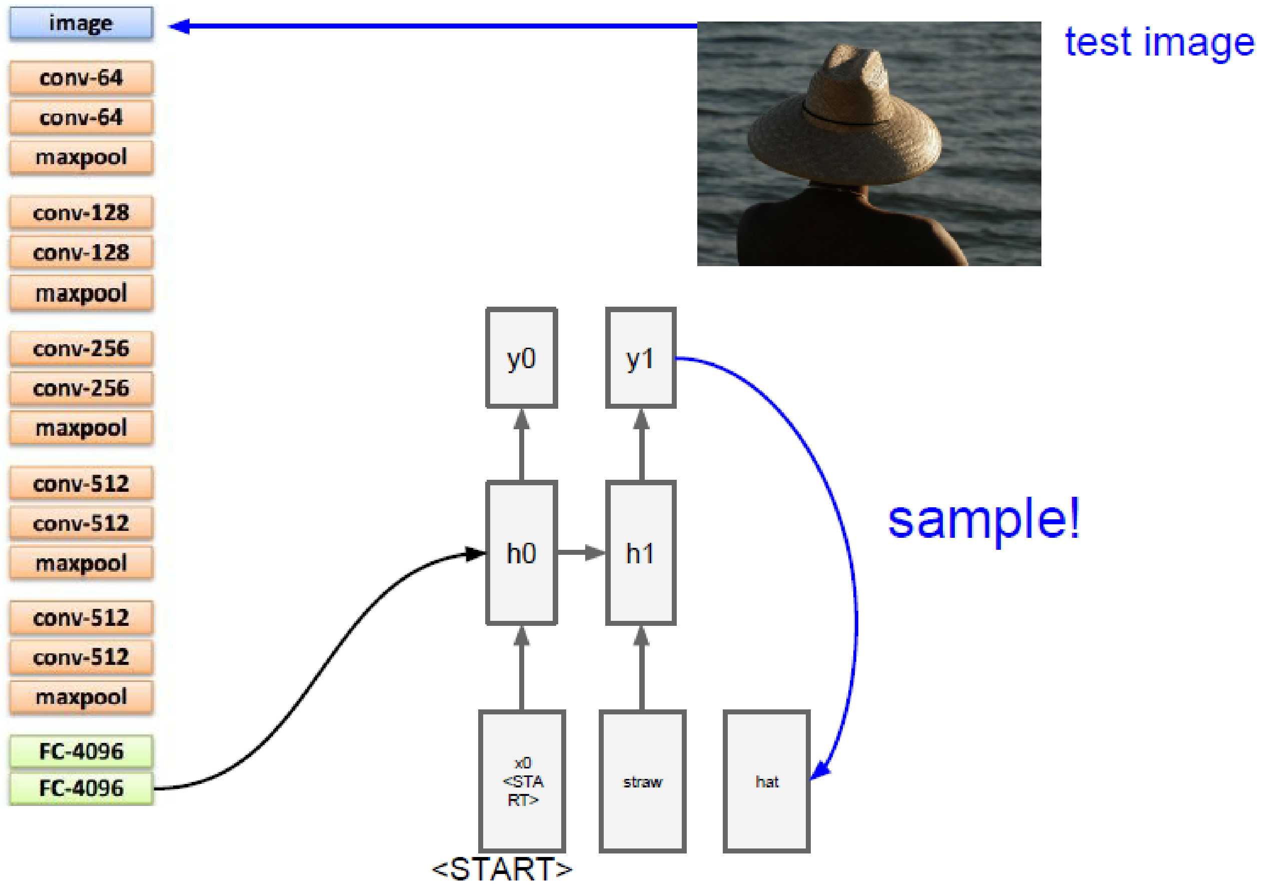
Example 3: Image captioning (cont.)



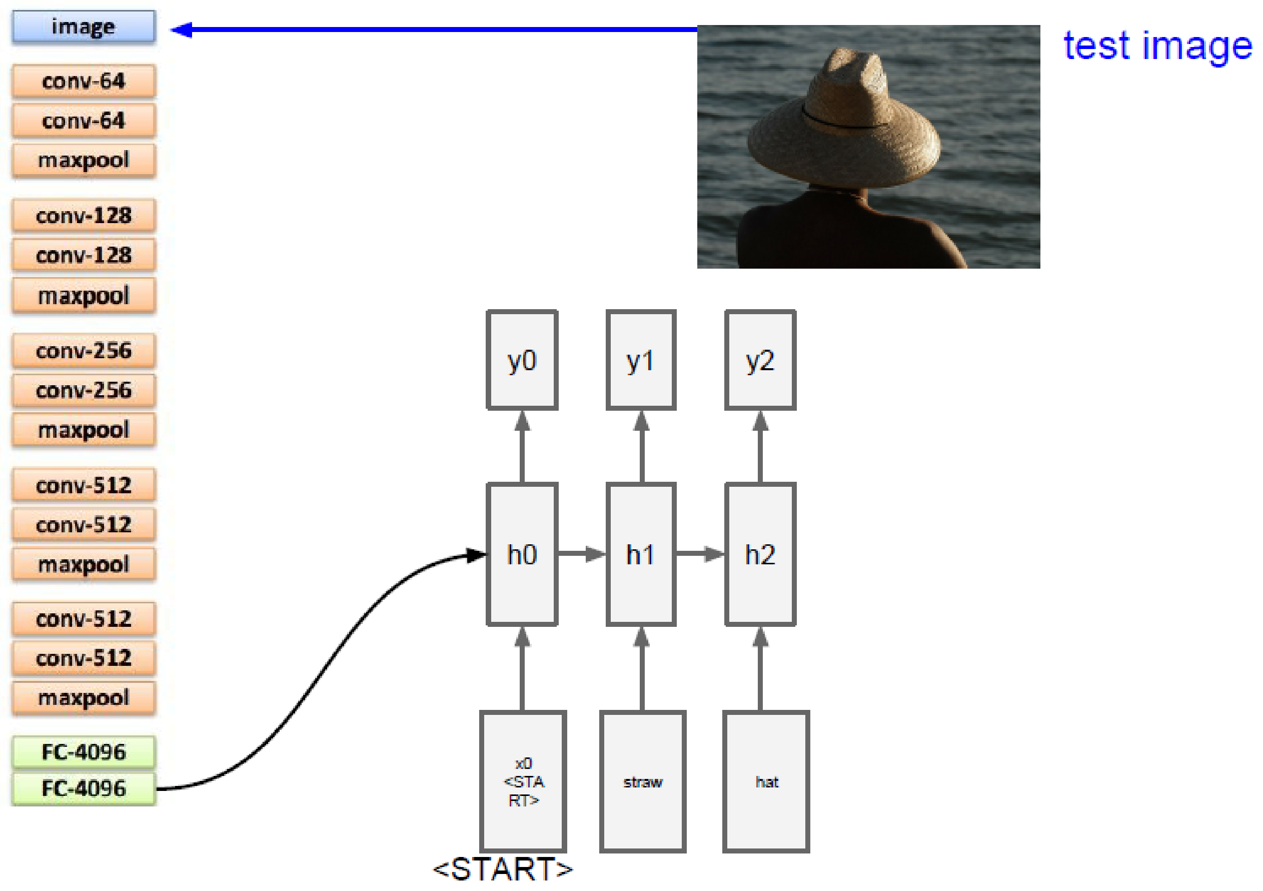
Example 3: Image captioning (cont.)



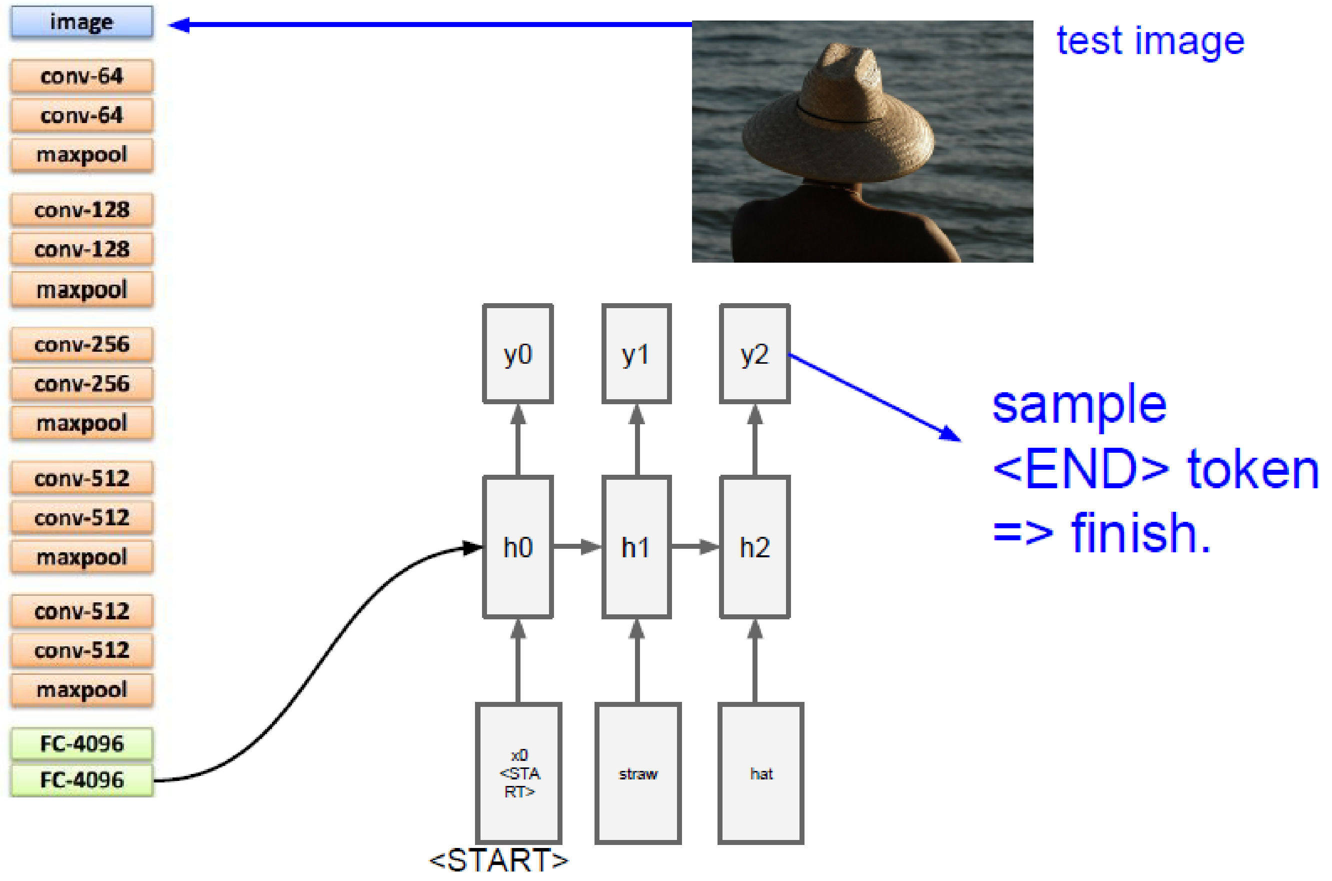
Example 3: Image captioning (cont.)



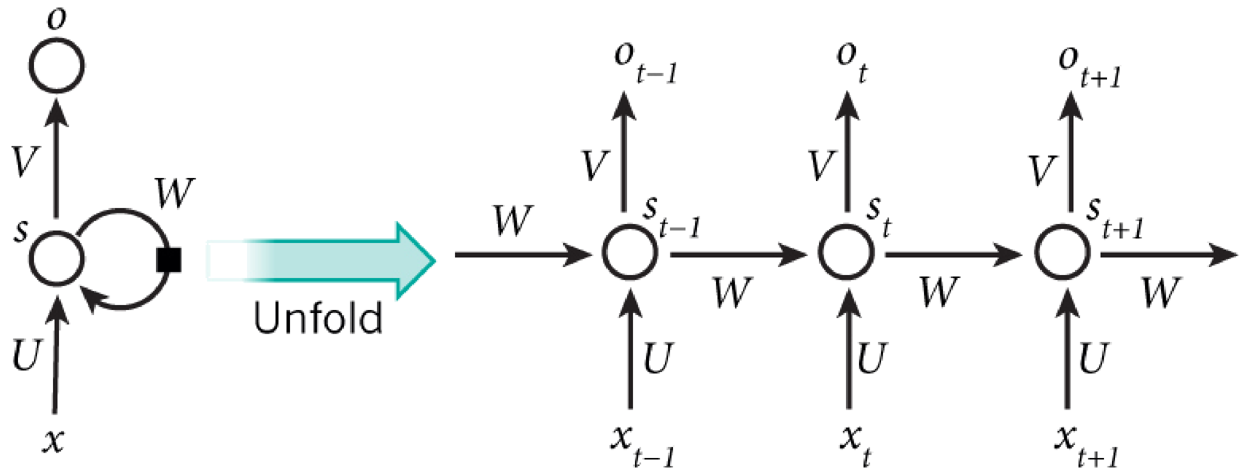
Example 3: Image captioning (cont.)



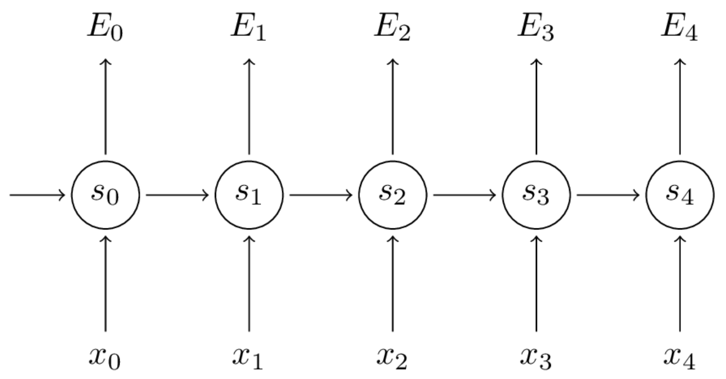
Example 3: Image captioning (cont.)



Training



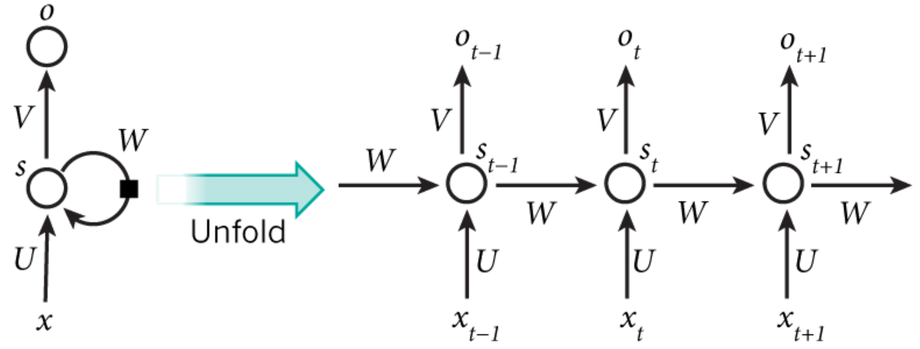
- Still use backpropagation
- But now weights are shared by all time steps
- Backpropagation Through Time (BPTT)
  - E.g., in order to calculate the gradient at $t = 4$, we would need to backpropagate 3 steps and sum up the gradients
  - $s_t = \tanh(Ux_t + Ws_{t-1})$
    $\hat{y}_t = \mathrm{softmax}(Vs_t)$
  - Loss, by cross entropy
    - y_t is the correct word at time t
    - $\hat{y}_t$ is prediction

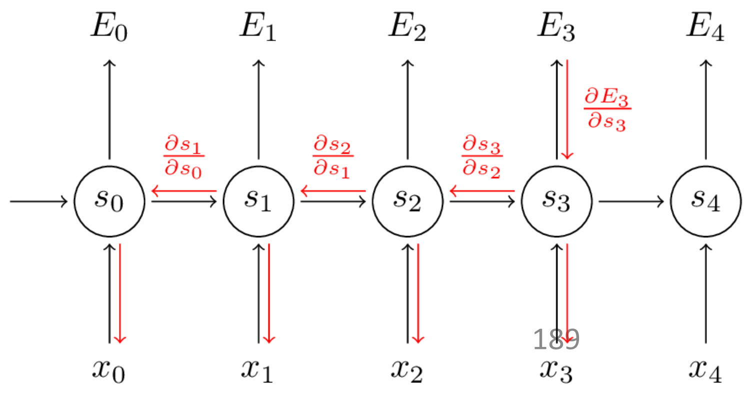


$$
\begin{aligned}
E_t(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\
E(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\
&= -\sum_t y_t \log \hat{y}_t
\end{aligned}
$$

Backpropagation Through Time (BPTT)

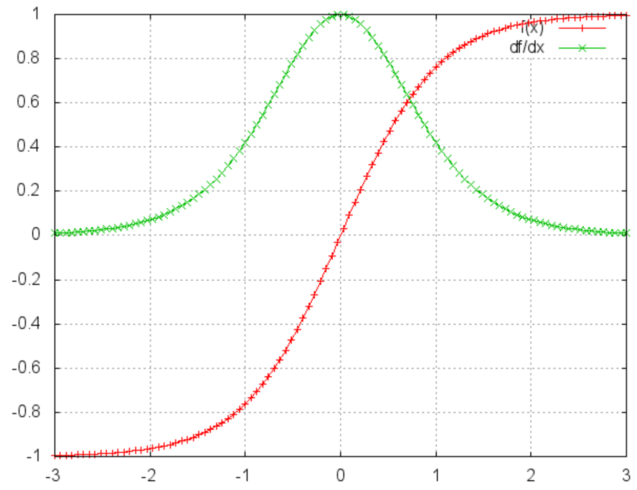
- Recall that our goal is to calculate the gradients of the error with respect to parameters U, V, W
- Also the gradients sum over time $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$
- E.g. $\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\frac{\partial s_3}{\partial W} + \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W} + \cdots \right)$
  - $s_3 = \tanh(Ux_t + Ws_2)$
- So to get gradients of time $t = 3$, we need to backpropagate gradients to $t = 0$





E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

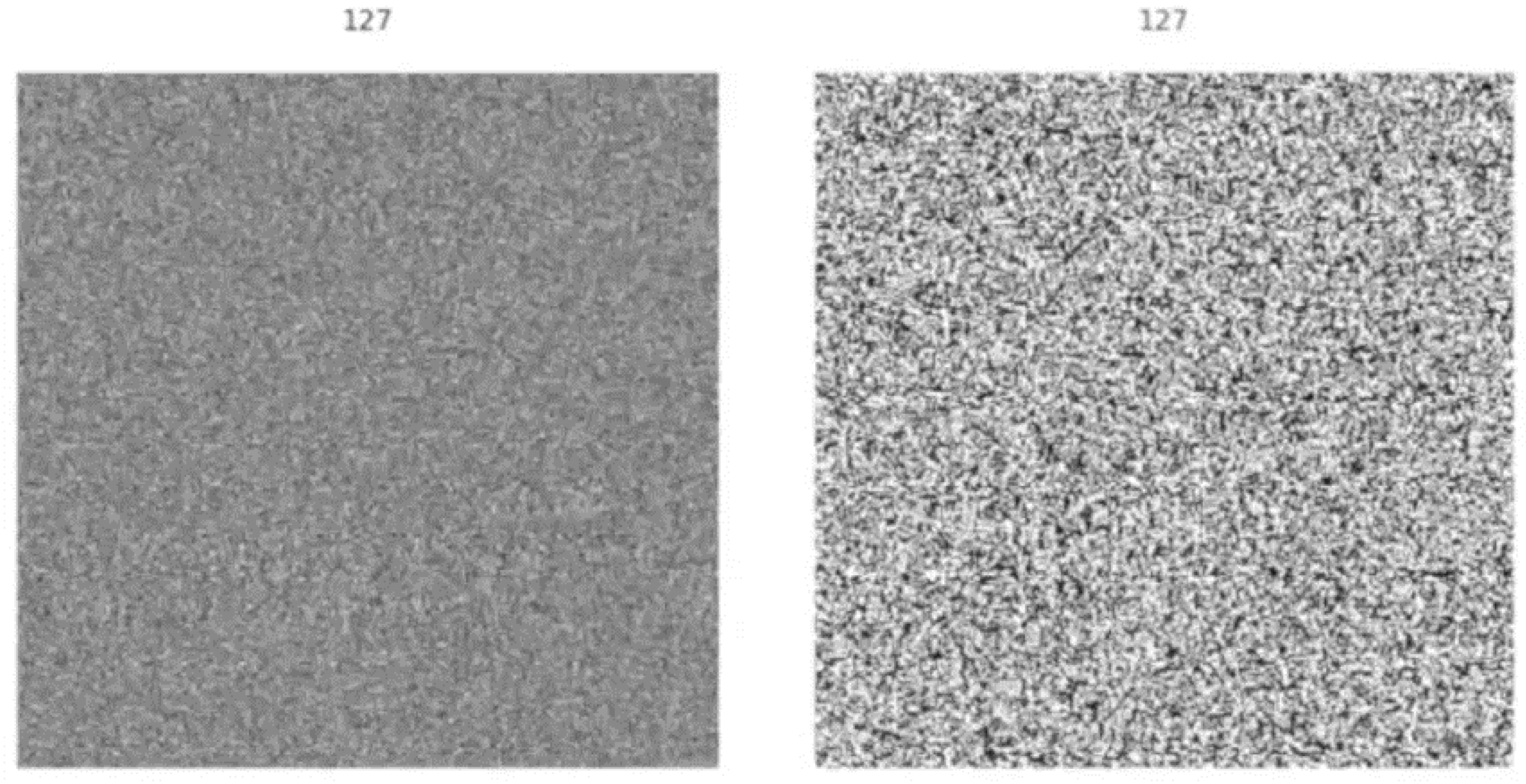
Summary of BPTT



- It is just backpropagation on the unrolled RNN by summing over all gradients for W and U at each time step
- The vanishing gradient problem
  - $\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\frac{\partial s_3}{\partial W} + \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W} + \cdots \right)$

    $\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$
  - This Jacobian matrix (derivative w.r.t. a vector) has 2-norm less than 1
    - As an intuition, the derivative of tanh is bounded by 1
  - Thus, with small values in multiple matrix multiplications, the gradient values are shrinking exponentially fast, eventually vanishing completely after a few time steps
  - Gradient contributions from “far away” steps become zero
  - You end up not learning long-range dependencies
  - Not exclusive to RNNs. It’s just that RNNs tend to be very deep (as sentence length)

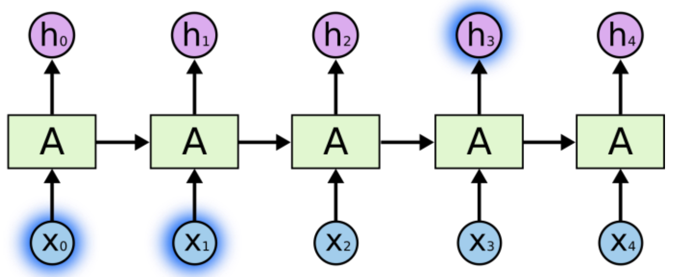
Demo of RNN vs LSTM: Vanishing gradients

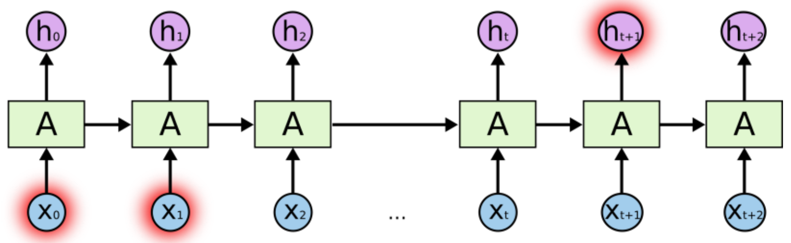


- RNN vs LSTM gradients on the input weight matrix https://imgur.com/gallery/vaNahKE
- Error is generated at 128th step and propagated back. No error from other steps
- Initial weights sampled from Normal Distribution in (-0.1, 0.1)

The problem of long-term dependencies

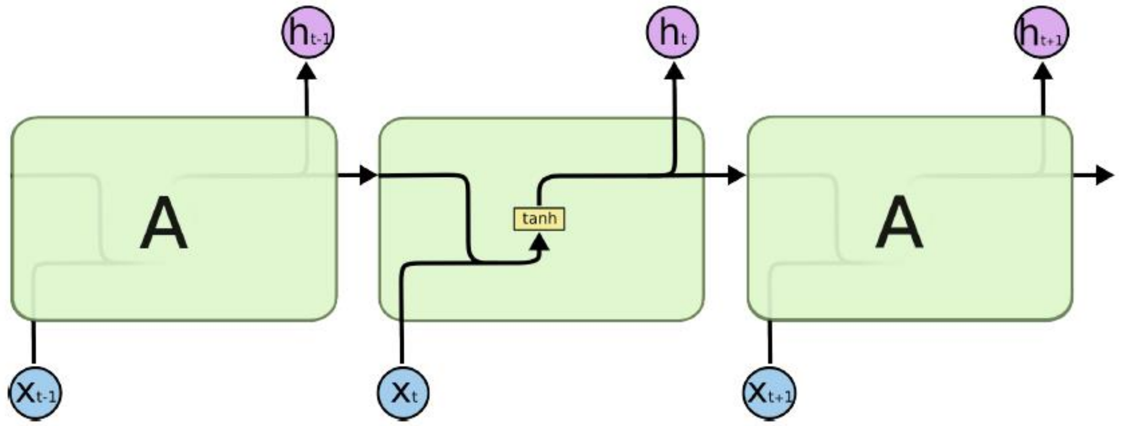
- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame
- RNN usually works well for recent information
  - E.g. The clouds are in the *sky*.
- But might work worse for further back context
  - E.g. I grew up in France… I speak fluent *French*.





Long short-term memory (LSTM)

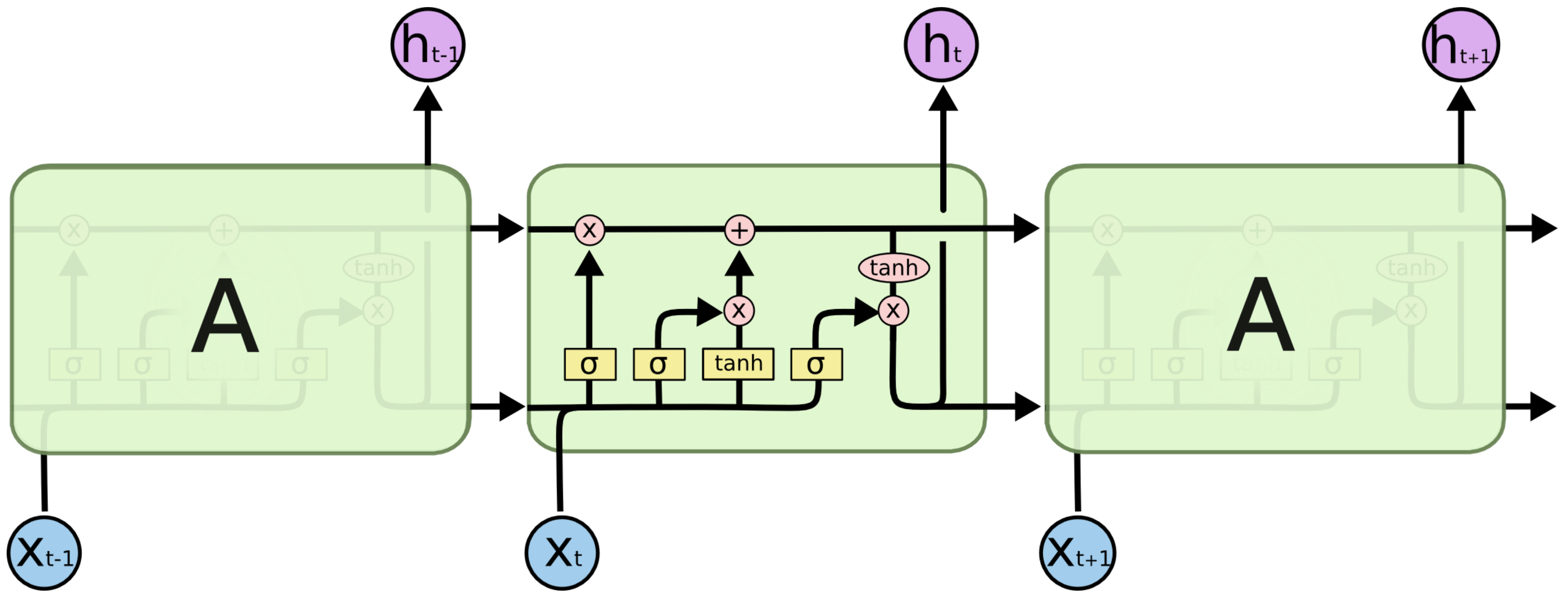
- Hochreiter & Schmidhuber [1997]
- A variant of RNN, capable of learning long-term dependencies
- Can remember information for long periods of time

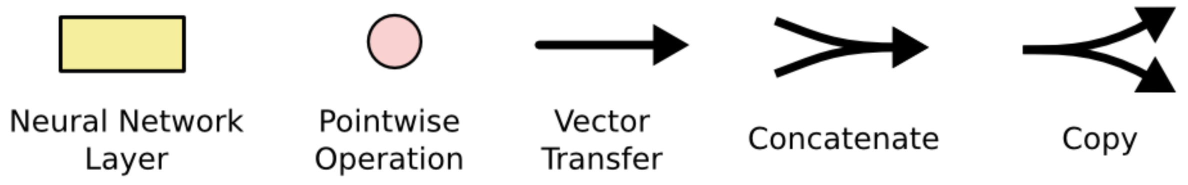


Remember the architecture of RNN cells, we will make many changes on it

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

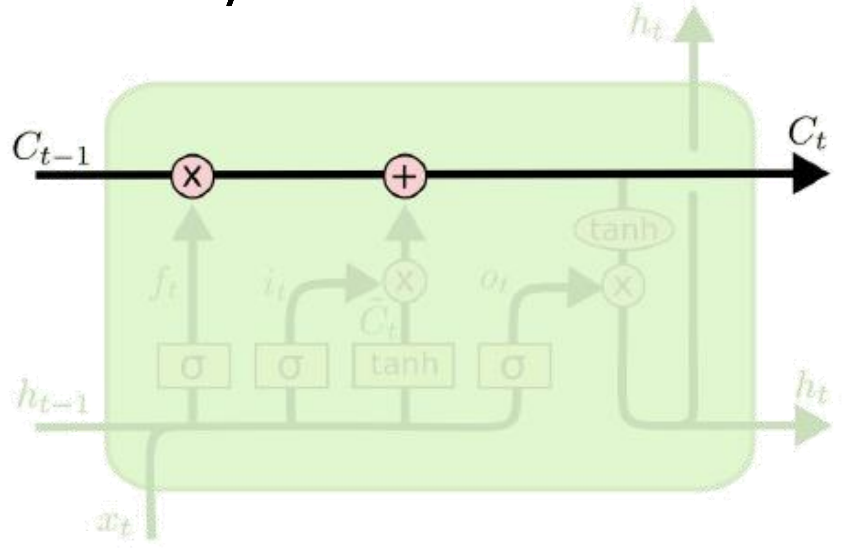
LSTM at a glance





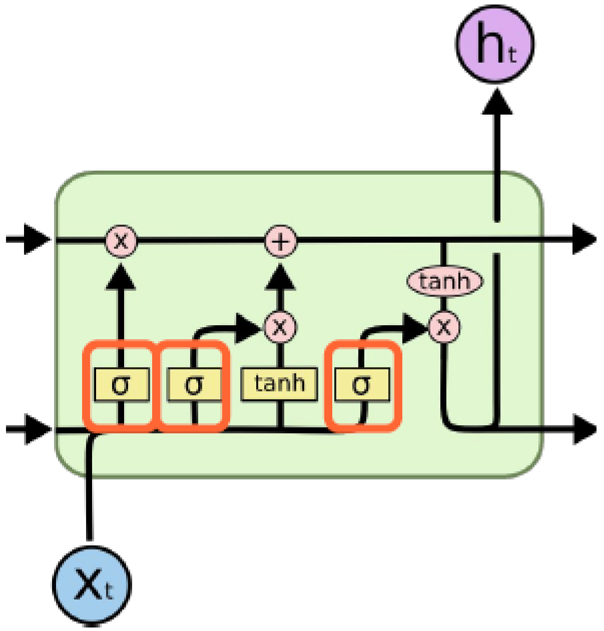
LSTM

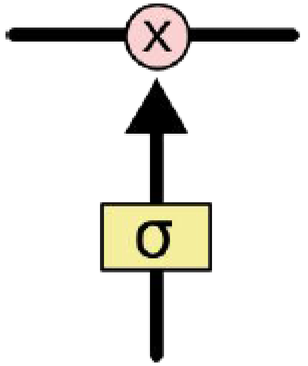
- C_t is the cell state
  - Horizontal line running through the top of the diagram
  - Like a conveyer
  - Run straight down the entire chain, with only some minor linear interactions
  - LSTM uses gates to carefully remove or add information to the cell state

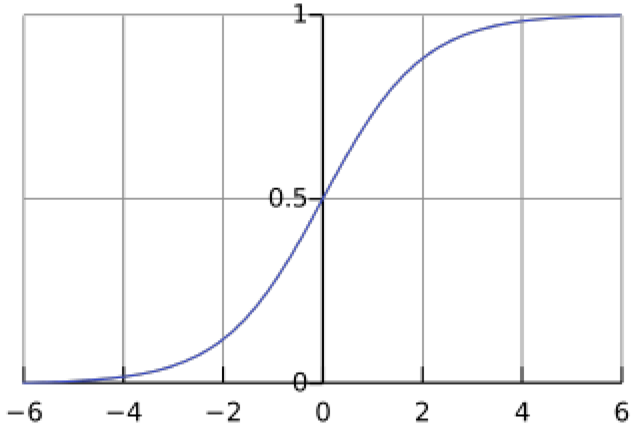


LSTM (cont.)

- Three gates are governed by *sigmoid* units
- Describing how much of each component should be let through
- “let nothing through” (zero value)/“let everything through” (one value)

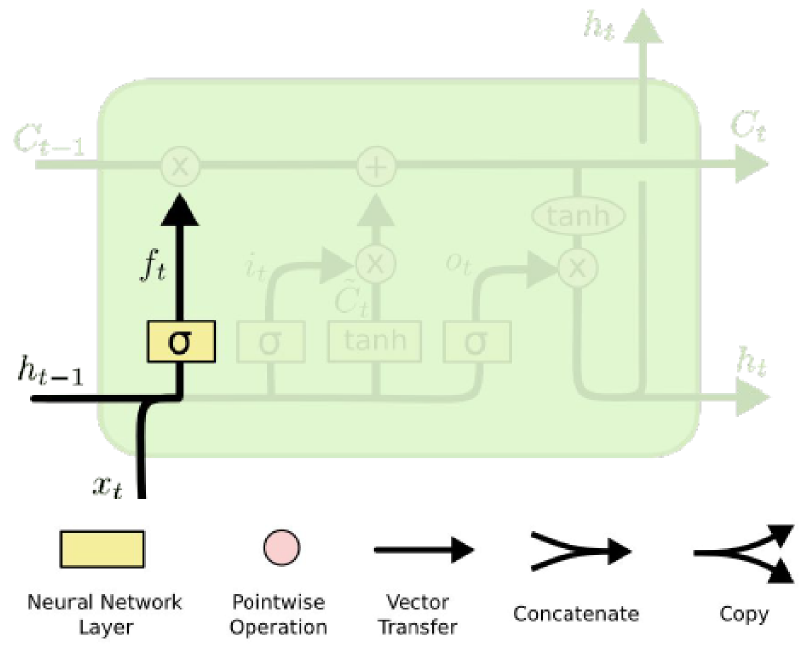






LSTM (cont.)

- The first gate is forget gate layer
  - Decide what information we're going to throw away from the cell state
  - Output a number in $(0,1)$ for each number in the cell state
  - 1/0: completely keep/remove this

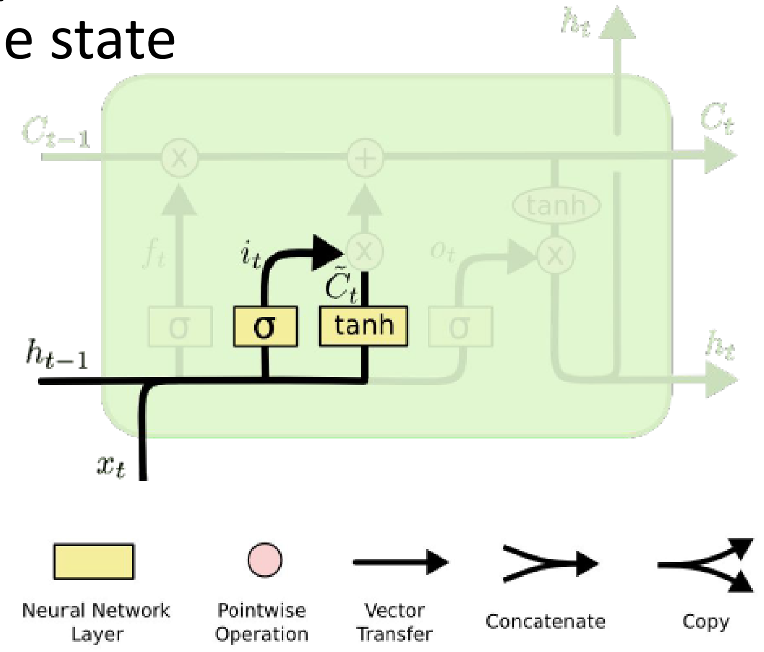


Forget Gate:

$$f_t = \sigma\left(W_f \cdot \underbrace{[h_{t-1}, x_t]}_{\text{Concatenate}} + b_f\right)$$

LSTM (cont.)

- Next is to decide what new information we're going to store in the cell state
  - The sigmoid layer is the input gate layer, deciding which values we'll update
  - The tanh layer creates a vector of new candidate values $\tilde{C}_t$ that should be added to the state



Input Gate Layer

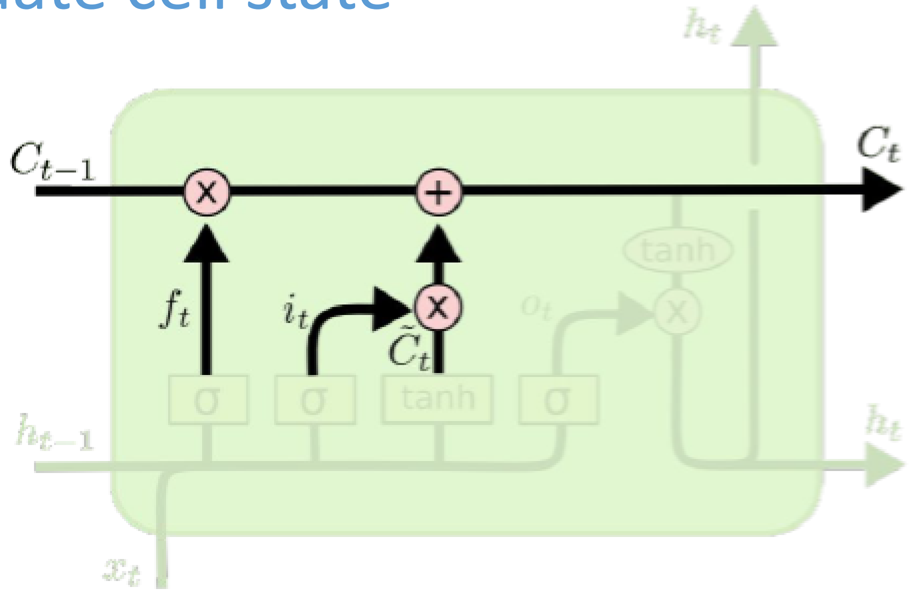
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$

New contribution to cell state

$$\tilde{C}_t = \underbrace{\tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)}_{\text{Classic neuron}}$$

LSTM (cont.)

- Multiply the old state by f_t
  - Forgetting the things from previous state
- Add new candidate values
- Update cell state

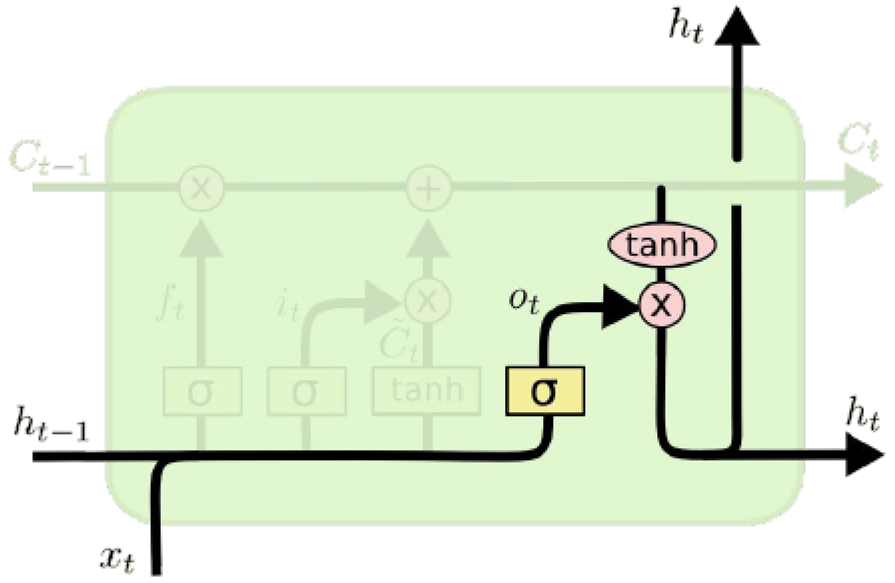


Update Cell State (memory):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM (cont.)

- Last decide the output h_t
  - Filtered (tanh) version of the cell state
  - The sigmoid layer decides what parts of the cell state we're going to output h_{t-1}, x_t are like *context*



Output Gate Layer

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

Output to next layer

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM hyperparameter tuning

- Watch out for *overfitting*
- Regularization helps: regularization methods include L1, L2, and dropout among others.
- The larger the network, the more powerful, but also easier to overfit. Don’t want to try to learn a million parameters from 10,000 examples
  - parameters > examples = trouble
- More data is usually better
- Train over multiple epochs (complete passes through the dataset)
- The learning rate is the single most important hyper parameter