Cross Validation, Support Vector Machine

Lecture 24:

Cross Validation, SVM, Intro to Neural Networks

ECE/CS 498 DS

Professor Ravi K. Iyer

Department of Electrical and Computer Engineering

University of Illinois

Announcements

- MP3 Checkpoint 1.5 due **tonight @ 11:59 PM** via Google Form
  - https://forms.gle/TyKhX8CVMVigpAoK6
- HW 4 due **Wed 4/22 @ 11:59 PM** via Compass
- Discussion section on Fri 4/24 will be an additional office hour with the TAs
- Final Exam will be Friday 5/15 from 8-11 AM
  - More details about logistics to come soon

Cross Validation

Validation: Binary Classification Performance

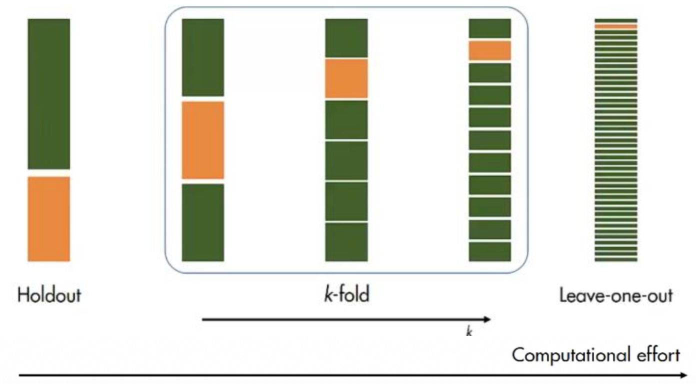
You are solving a Binary Classification Problem, Predict if a person has cancer (1) or not (0). You have learned a model using your training dataset. **How do you test your model?**

Explanation:

1) **Holdout**: Randomly divide dataset into 2 parts, learn model on training data and validate on the test data.
2) **K-fold**: Divide data into K equal parts, run the following K-times: (hold out one of the K parts of the dataset, train on the remaining, test on the holdout). Every data-sample is part of the training data K-1 times and part of the test data 1 time.
3) **Leave-one-out**: For every data sample, leave it out, train the model on the remaining data and predict for the left-out data sample.
4) **Monte Carlo Cross Validation**: In each of multiple trials, randomly divide the dataset into training and testing sets (of prespecified sizes). Always retrain the model on the trial's training set and validate on the trial's testing set.

Cross Validation: For a given training iteration, divide the dataset into 2 parts:
- training set (for training the model)
- test set (for validating the model)

Different ways of doing this:
(1) Holdout
(2) K-fold
(3) Leave-one-out
(4) Monte Carlo

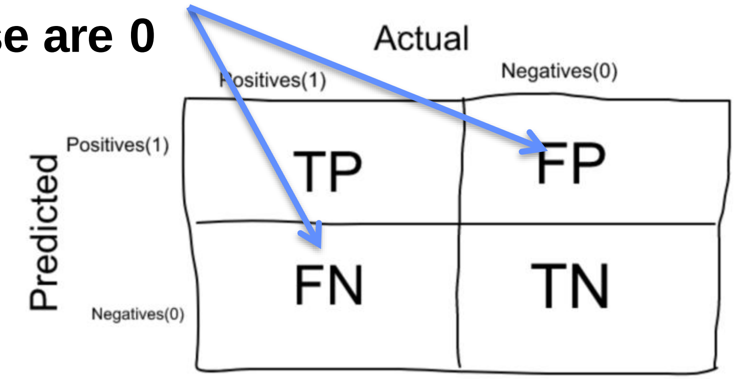


green – training data
orange - test data

What does it mean to **validate** on the test data?

Ideal if these are 0

Confusion Matrix



Depending on the **cost of making a mistake**, minimize the number of FN or FP

Metrics

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

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

$$\text{Sensitivity (Recall)} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{F1-score} = \frac{2 * Precision * Recall}{Precision + Recall}$$

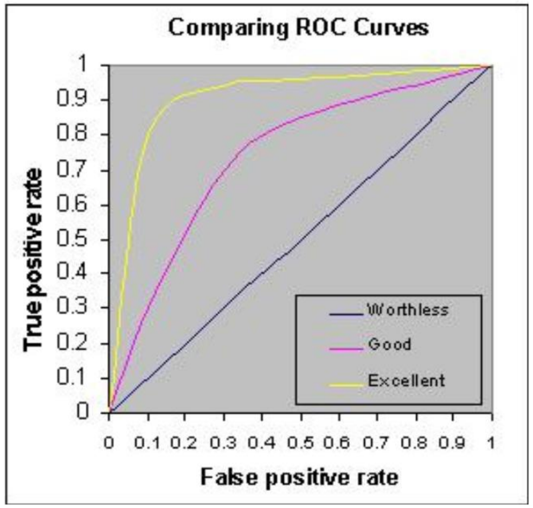
Harmonic mean of Precision and Recall (when large disparity between the 2 values, the score is closer to the smaller value)

Common issues

Precision = 1 if FP=0, but FN may be high
Sensitivity = 1 if predict everything as positive
Specificity = 1 if predict everything as negative
How to strike a **balance**? Use F1-score!

ROC and AUC

Receiver Operating Characteristic curve: A plot of **False Positive Rate** Vs **True Positive Rate**



$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{TN + FP}$$

An ML model has parameters and for different values of the parameters, the model gives different FPR and TPR.

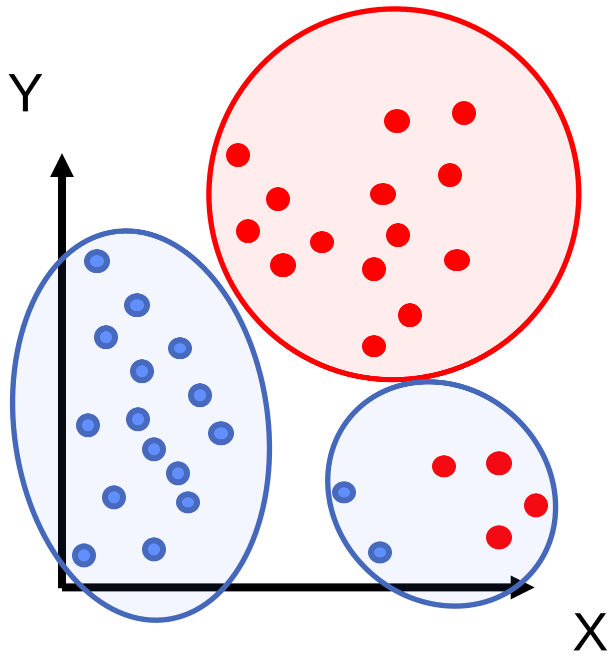
An **ROC (Receiver Operating Characteristic) curve** can be plotted for these different settings. A **good setting** of the parameters i.e. a good classifier, would have a **high TPR and low FPR**.

Area Under Curve (AUC) in [0,1]:
A good classifier would have the largest area under the ROC curve.
The random predictor would have an ROC of 0.5 (the curve of the 'worthless' model in fig.)

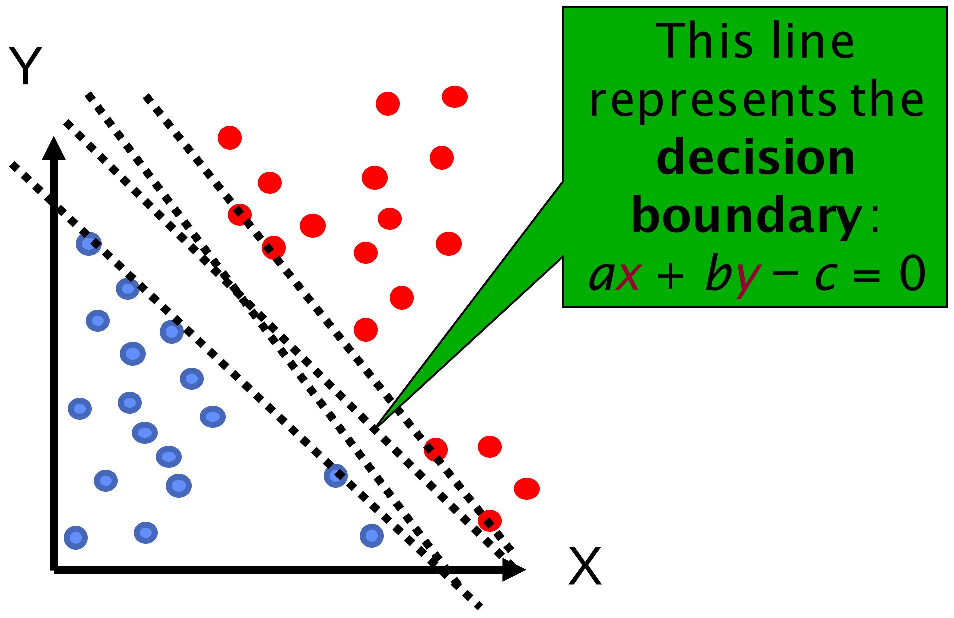
AUC value **interpretation** –
If a pair of data samples are drawn independently, one each from the positive and negative sets, AUC gives the probability that the classifier will predict a lower score for the negative sample as compared to the positive sample

Supervised Learning and SVM

Unsupervised to Supervised Learning

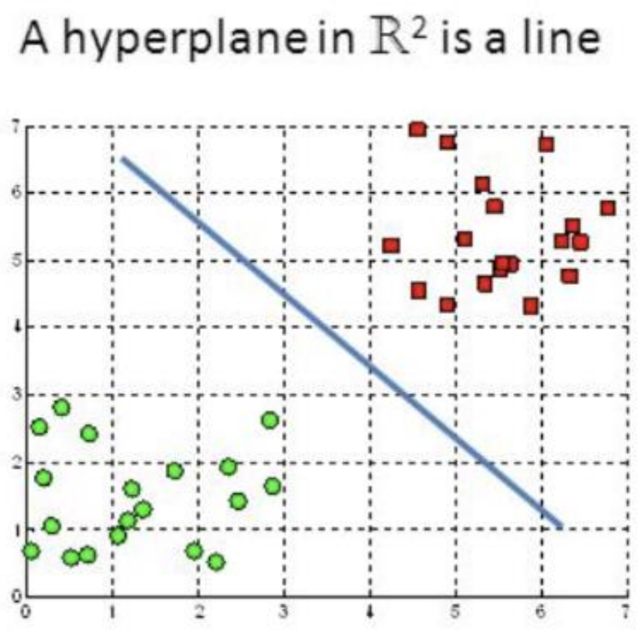


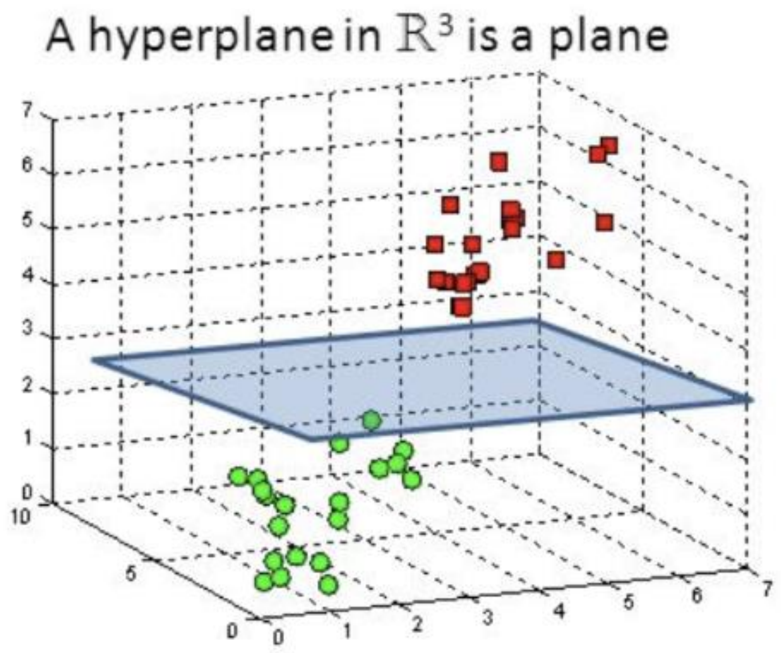
Can we do better? Yes


Y
X
This line represents the **decision boundary**: $ax + by - c = 0$

If in a classification task the data is linearly separable, a decision boundary with the same dimensionality as the data can be used to separate the data

Hyperplane as decision boundary





Example of decision boundary in 2D space (line) and 3D space (plane).

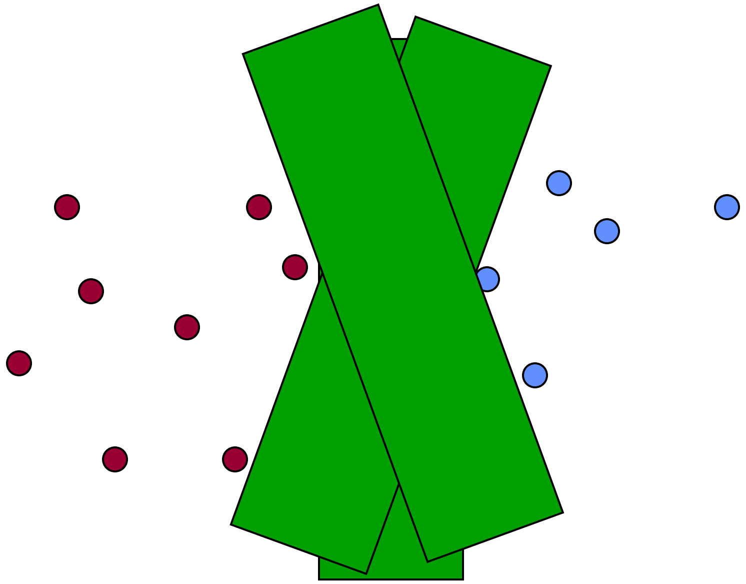
In d dimensional space, hyperplane is given by:

$$a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_d x_d = 0$$

Image Source: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Linear classifiers: Which Hyperplane?

- Lots of possible solutions for *a, b, c* in hyperplane equation *ax+by+c=0*.
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of goodness]
- Support Vector Machine (SVM) finds an optimal* solution.
  - Maximizes the distance between the hyperplane and the "difficult points" close to decision boundary
  - SVMs maximize the *margin* around the separating hyperplane.
    - A.k.a. large margin classifiers
  - The decision function is fully specified by a subset of training samples, *the support vectors*.



If you have to place a fat separator between classes, you have less choices, and so the capacity of the model has been decreased

Linear Support Vector Machine (SVM)

Assumption: Data is linearly separable

Let $\boldsymbol{x_0} = (x_{0,1}, x_{0,2})$ be a point $\in \mathbb{R}^2$ on $\mathbf{w}^T\mathbf{x}+b = 1$

Then its distance to the separating plane $\mathbf{w}^T\mathbf{x}+b = 0$ is:

$$\frac{|\boldsymbol{w}^T\boldsymbol{x_0} + b|}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||}$$

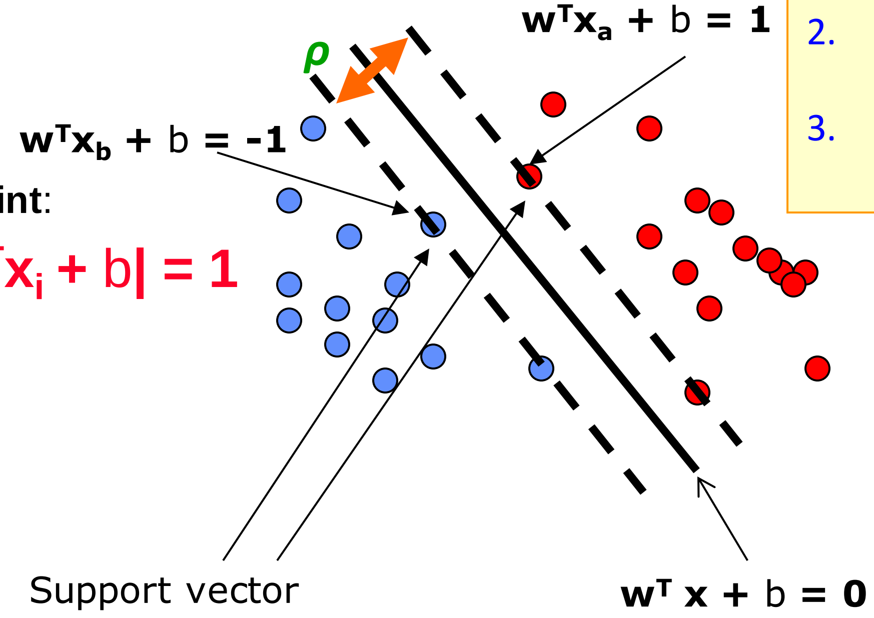
Distance between $\mathbf{w}^T\mathbf{x}+b = +1$ and -1 is $\rho = \frac{2}{||w||}$

What we did:

1. Consider all possible w with different angles
2. Scale w such that the constraints are tight
3. Pick the one with largest margin/minimal size

- **Hyperplane**
  $\mathbf{w}^T\mathbf{x} + b = 0$
- **Extra scale constraint**:

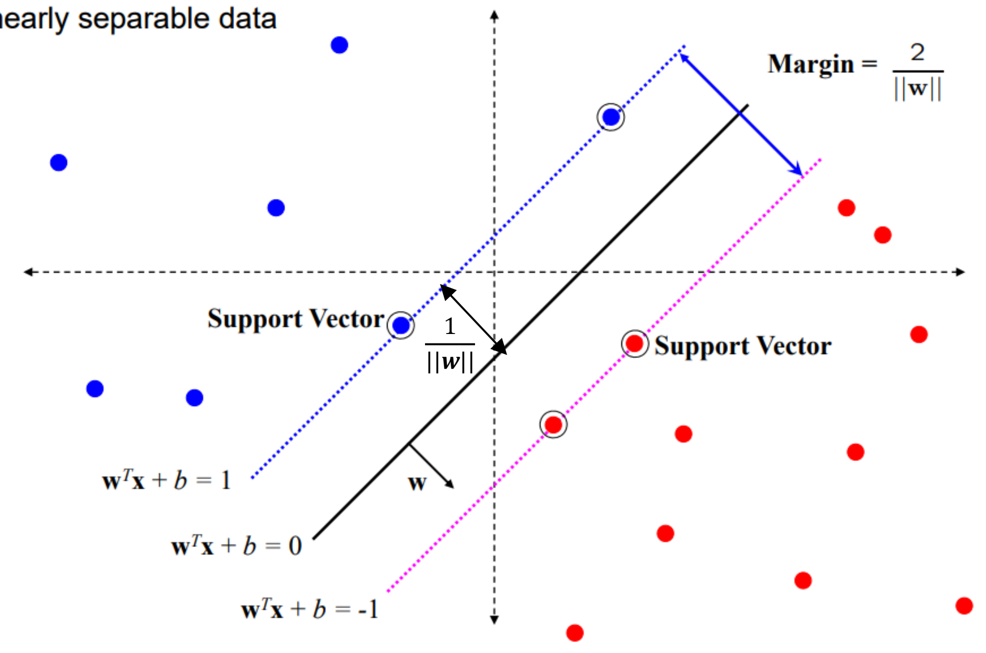
$$\min_{i=1,\ldots,n} |\mathbf{w}^T\mathbf{x}_i + b| = 1$$



SVM Optimization:
Maximize: ρ **or**
Minimize: $\frac{1}{2}||w||^2$

SVM : How to find such hyperplane?

linearly separable data



Margin = $\frac{2}{||\mathbf{w}||}$
Support Vector
$\frac{1}{||w||}$
Support Vector
$\mathbf{w}^T\mathbf{x} + b = 1$
w
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$

Constraint and Objective:

1. Define the hyperplane via the following rule:

$$y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1$$

-- **Constraint**

2. Maximize margin distance:

Margin distance= $\frac{\boldsymbol{w}^T\boldsymbol{x}+b+\mathbf{1} - (\boldsymbol{w}^T\boldsymbol{x}+b-\mathbf{1})}{||\boldsymbol{w}||} = \frac{2}{||\boldsymbol{w}||}$

$$\max \frac{2}{||\boldsymbol{w}||} \Rightarrow \min ||\boldsymbol{w}|| \Rightarrow \min \frac{||\boldsymbol{w}||^2}{2}$$

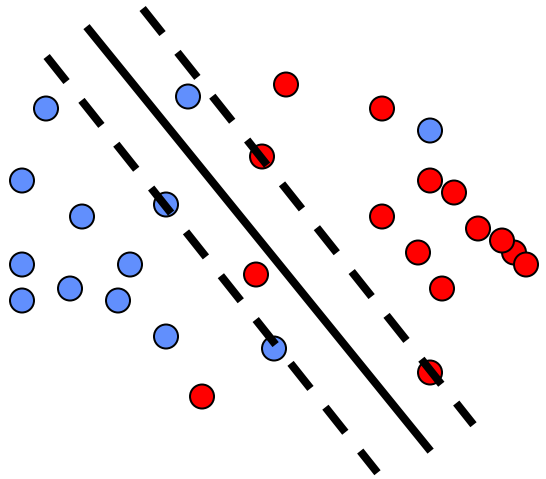
-- **Objective**

Each training point: $\boldsymbol{x_i} \in R^d \ \forall i \in \{1, \dots, n\}$
Label: $y \in \{-1, 1\}$
$\boldsymbol{w} \in R^d$

Maximum margins linear classifier:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{2}||\boldsymbol{w}||^2 \text{such that } 1 \leq y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \quad \forall i \in \{1, \dots, N\}$$

Dataset with noise

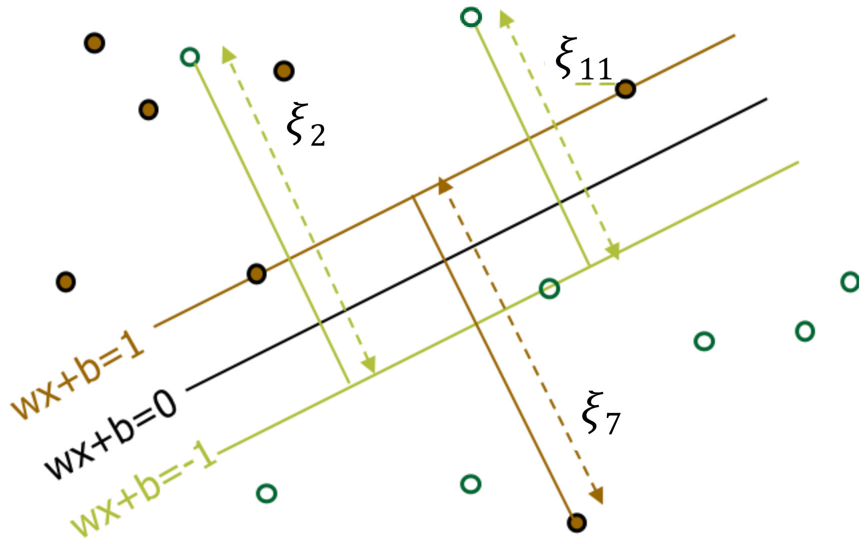


- **Hard Margin:** So far we require all data points be classified correctly
  - No training error
- **What if the training set is noisy?**
  - Solution: Use **Soft Margin**

The two red points and two blue points do not satisfy the criteria $y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1$

Soft Margin Classification

Slack variables **$\xi_i \geq 0$ can be added to allow misclassification of difficult or noisy examples.**



What should our optimization criterion be?

Minimize $$\frac{||\boldsymbol{w}||^2}{2} + C\sum_{j=1}^{k} \xi_j$$

The slack variables will be zero for the samples which can be classified correctly and a positive value for the remaining points.

Hard Margin v.s. Soft Margin

- **The old formulation (Hard Margin):**

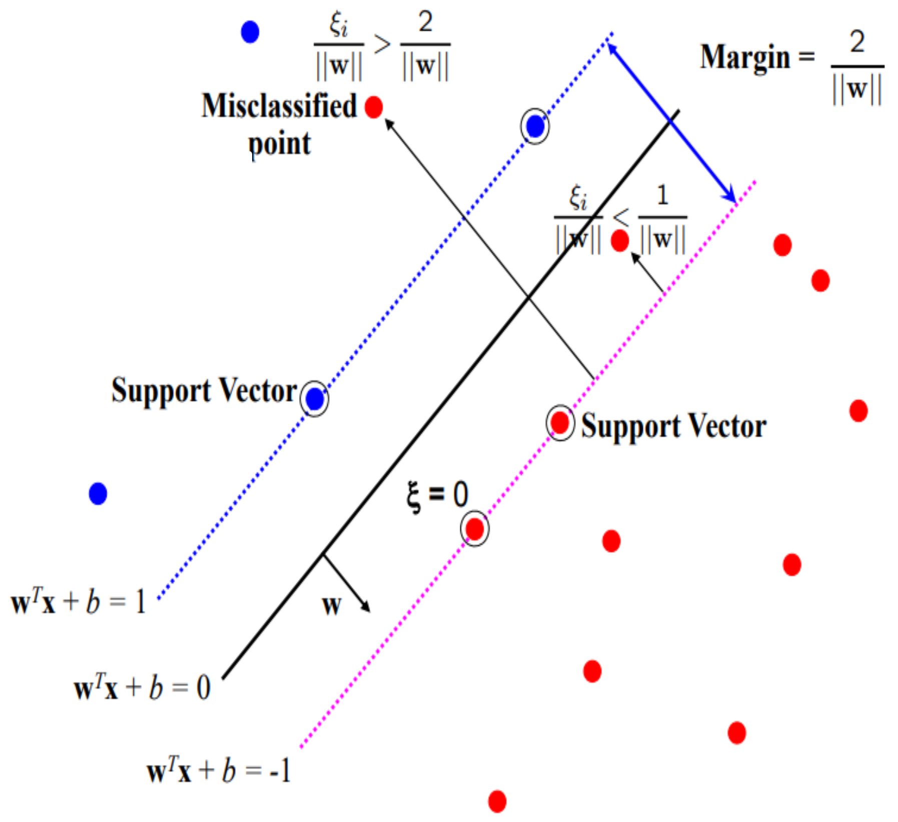
Find **w** and b such that
Objective: $\frac{1}{2}\mathbf{w}^T\mathbf{w} = \frac{1}{2}||\mathbf{w}||^2$ is minimized
Constraint: for all $\{(\mathbf{x_i}, y_i)\}$, $y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

- **The new formulation incorporating slack variables (Soft Margin):**

Find **w** and b such that
Objective: $\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
Constraint: for all $\{(\mathbf{x_i}, y_i)\}$, $y_i(\boldsymbol{w}^T\boldsymbol{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- **Parameter *C* can be viewed as a way to control overfitting.**
  - **As $C \rightarrow \infty$, the amount of allowed slack goes to 0 and we approach hard margin**
  - **As $C \rightarrow 0$, the amount of allowed slack can grow arbitrarily large and $||w|| \rightarrow 0$ (the margin size grows to ∞)**

SVM: Non-separable case



$\frac{\xi_i}{||w||} > \frac{2}{||w||}$
Misclassified point
Margin = $\frac{2}{||w||}$
$\frac{\xi_i}{||w||} < \frac{1}{||w||}$
Support Vector
Support Vector
$\xi = 0$
$w^T x + b = 1$
w
$w^T x + b = 0$
$w^T x + b = -1$

- $\xi_i \geq 0$
- $0 < \xi_i \leq 1$: point is between margin and correct side of hyper plane

  —**margin violation**
- $\xi_i > 1$: point is on the wrong side of margin

  —**misclassification**

Maximum margins linear classifier:

$$\underset{w, \xi_I \geq 0}{\operatorname{argmin}} \frac{1}{2} ||w||^2 + C \sum_i^N \xi_i \text{ such that } 1 - \xi_i \leq y_i(w^T x_i + b) \quad \forall i \in \{1, \ldots, N\}$$

Transformation to data

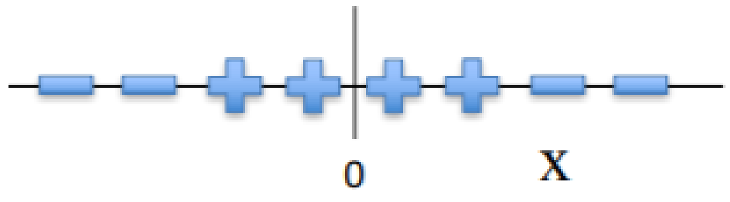


Fig 1

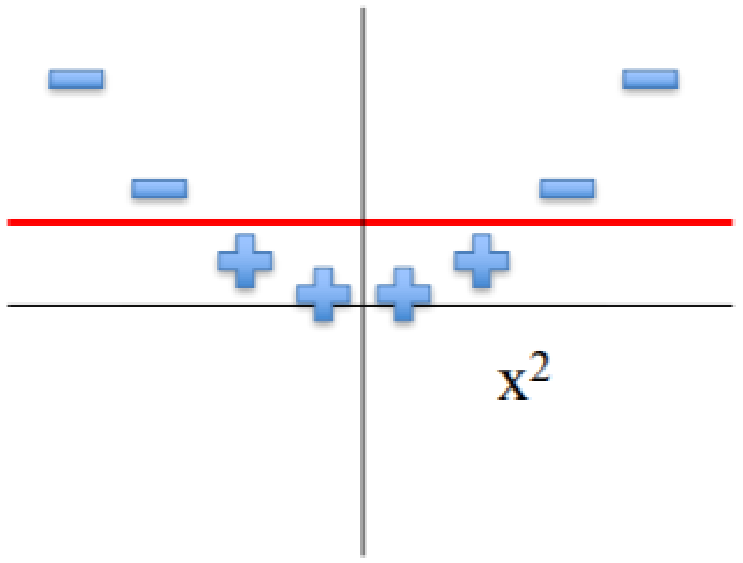
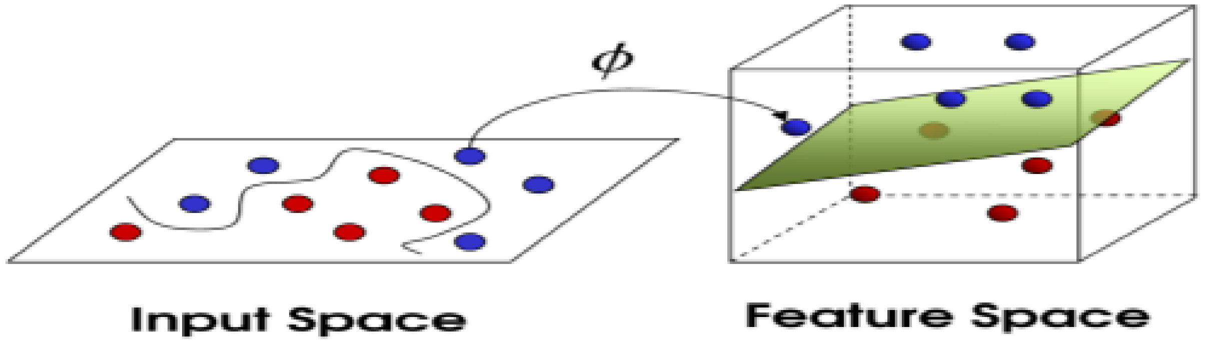


Fig 2

- Data points in Fig 1 cannot be separated using SVM
- Applying **transformation** $\Phi(x) = x^2$ gives data points in Fig 2
  - Can be separated by a line
- Apply SVM on transformed data

Mapping data to new feature space



$$\Phi : \mathcal{X} \mapsto \hat{\mathcal{X}} = \Phi(\mathbf{x})$$

- For example, if $x_i = [x_{i1}, x_{i2}]$, i.e., $\boldsymbol{x} \in \mathbb{R}^2$,

$\Phi(x) = [1, x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2]$

- Run SVM on $\Phi(x)$ instead of x

Kernels

- Define Kernel $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ = $\Phi(\boldsymbol{x_i})^T \Phi(\boldsymbol{x_j})$
- Use $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ in SVM
- Because of the way the solution of SVM optimization problem is computed, it is easy to use $K(\boldsymbol{x_i}, \boldsymbol{x_j})$

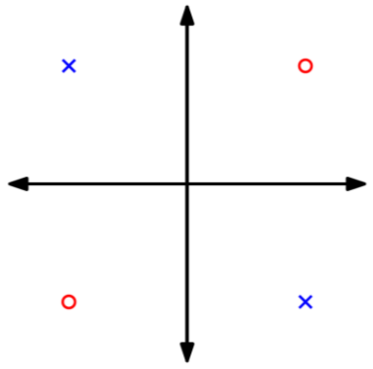
Example of commonly used Kernels ($x_i, x_j \in \mathbb{R}$)

- Polynomial kernel of order 2: $K(x_i, x_j) = 1 + x_i + x_j + x_i^2 + x_j^2 + 2x_i x_j$
- Radial Basis Function: $K(\mathrm{x}_i, x_j) = \exp\left(-\frac{(\mathbf{x}_i - \boldsymbol{x_j})^2}{2\gamma^2}\right)$

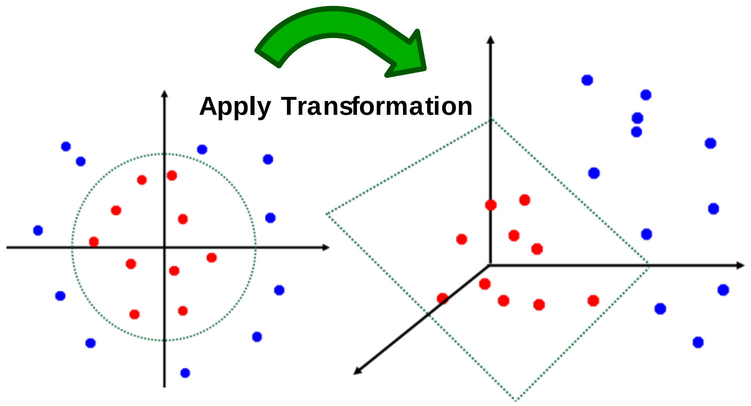
Intro to Neural Networks

Neural Networks

- Limitations in SVM Models
  - Kernel-SVM can separate data using transformation $\phi(x)$, i.e., x is not linearly separable but $\phi(x)$ is linearly separable; think of $\phi(x)$ as the new feature
  - Identifying the appropriate kernel is difficult
  - Computation time for large problems



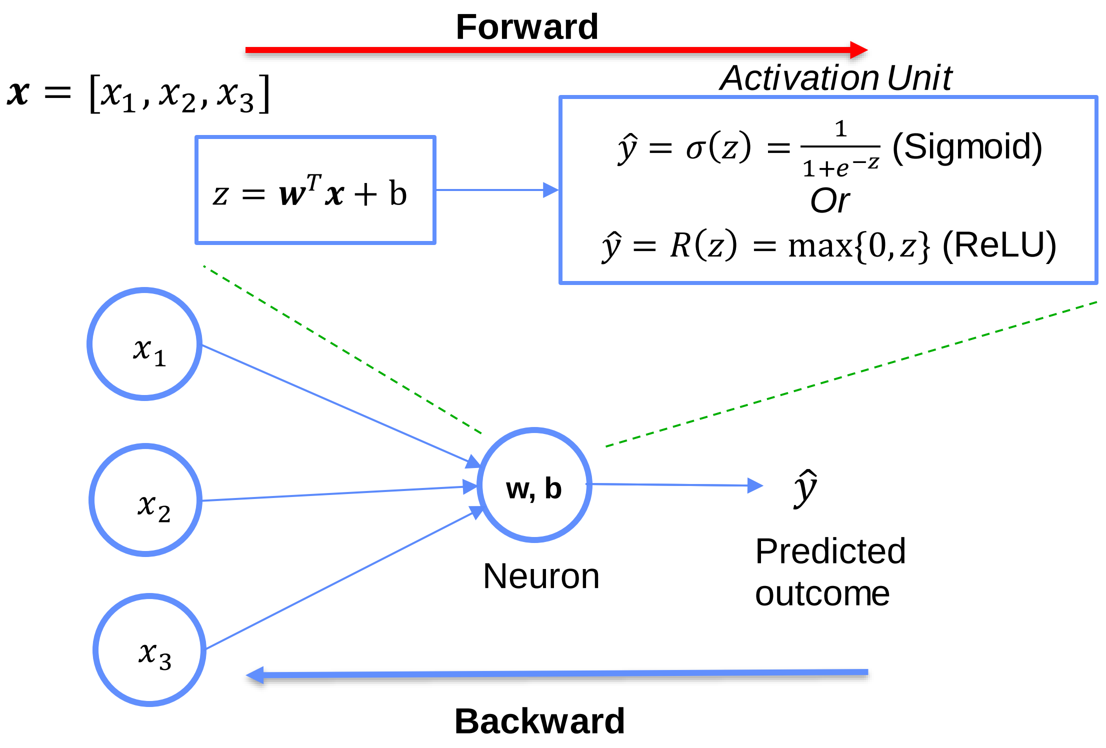
No linear separator classifies perfectly!



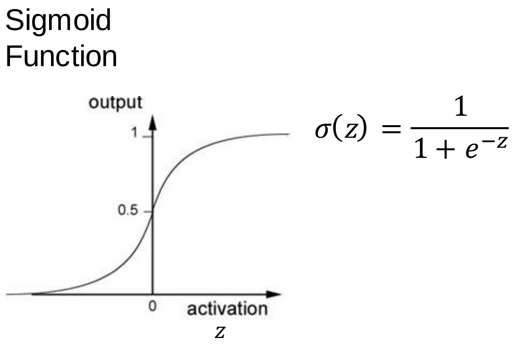
- Neural Network
  - Derive features of the data automatically as a constrained optimization problem

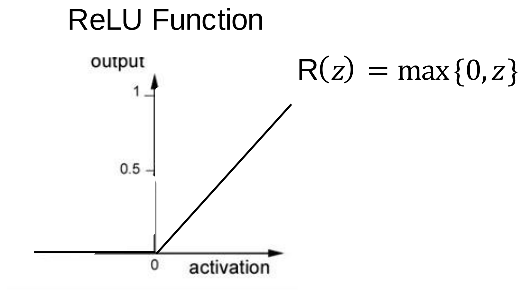
Perceptron Model

- The core of the neural network is perceptron model



Forward
$\boldsymbol{x} = [x_1, x_2, x_3]$
$z = \boldsymbol{w}^T\boldsymbol{x} + \mathrm{b}$
Activation Unit
$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$ (Sigmoid)
Or
$\hat{y} = R(z) = \max\{0, z\}$ (ReLU)
x_1
x_2
x_3
w, b
Neuron
$\hat{y}$
Predicted outcome
Backward





Update Rule (Backward):

$$w_{t+1} = w_t - \eta \nabla J(w_t)$$

η : Learning rate

Loss

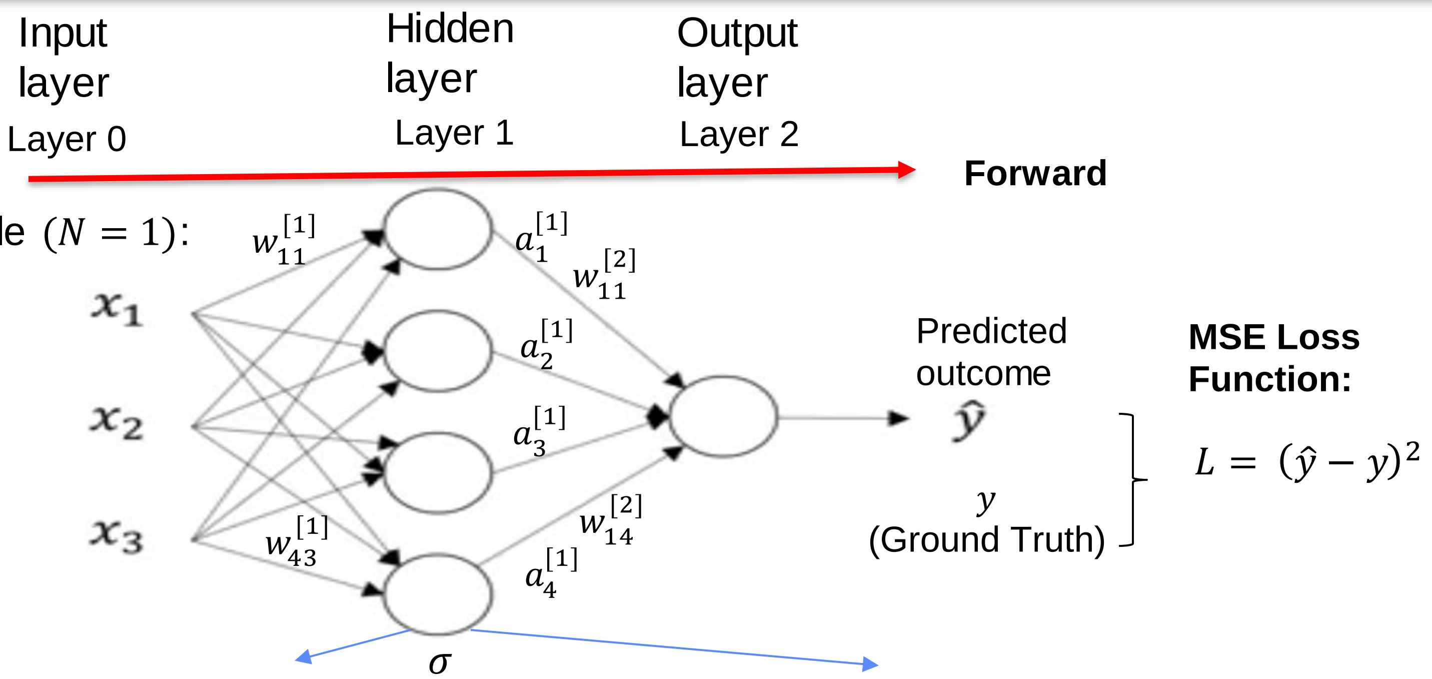
$$J(w) = \frac{1}{N}\sum_{i=1}^{N} L(w.x^{(i)}, y^{(i)})$$

Computing Gradient

$$\nabla J(\mathbf{w}_0) = \left(\frac{\partial J(\mathbf{w})}{\partial w_0}, \frac{\partial J(\mathbf{w})}{\partial w_1}, \cdots, \frac{\partial J(\mathbf{w})}{\partial w_n}\right)_{\mathbf{w}_0}$$

N: number of samples $x^{(i)}$: feature of i^{th} sample

Neural Network (Forward)



Assume 1 sample $(N = 1)$:

$$[x_1 \quad x_2 \quad x_3] \cdot \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} & w_{41}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} & w_{42}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} & w_{43}^{[1]} \end{bmatrix} + [b_1^{[1]} \quad b_2^{[1]} \quad b_3^{[1]} \quad b_4^{[1]}] = \boldsymbol{z}^{[1]} \rightarrow \boldsymbol{a}^{[1]}$$

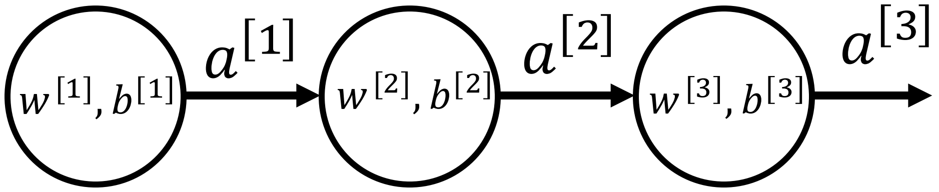
Where,

$$\boldsymbol{z}^{[1]} = [z_1^{[1]} \; z_2^{[1]} \; z_3^{[1]} \;\; z_4^{[1]}]$$

$$\boldsymbol{a}^{[1]} = [a_1^{[1]} \; a_2^{[1]} \; a_3^{[1]} \;\; a_4^{[1]}] \qquad \boldsymbol{a}_j^{[i]} = \sigma(\boldsymbol{z}_j^{[i]})$$

Back Propagation: Cost Function

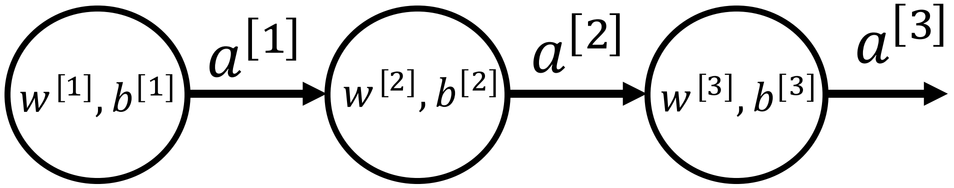
- Suppose we have a simple neural network as follows:

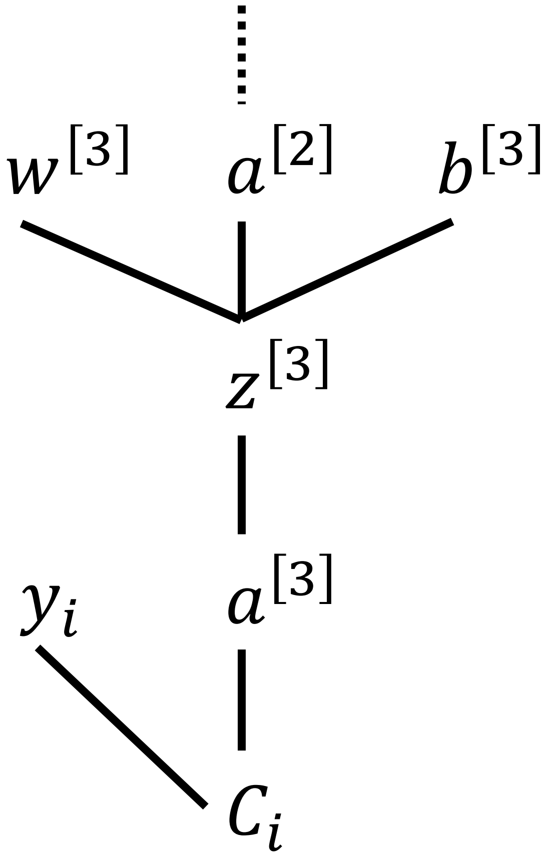


- Each layer j has scalar weight $w^{[j]}$ and scalar bias $b^{[j]}$
- Define $z^{[j]} = w^{[j]}a^{[j-1]} + b^{[j]}$
- We apply a nonlinearity with the sigmoid function: $a^{[j]} = \sigma(z^{[j]})$
- The predicted label for sample i is $\hat{y_i} = a^{[3]}$
- For sample i with the true label y_i, the cost function with this neural network is

$$\begin{aligned} C_i &= (\hat{y_i} - y_i)^2 \\ &= \left(a^{[3]} - y_i\right)^2 \end{aligned}$$

Back Propagation: Chain Rule





- Our goal is to figure out how to update the weights to minimize the cost in the next iteration of gradient descent
- To begin, we wish to calculate $\frac{\partial C_i}{\partial w^{[3]}}$
- Recall that

$$\begin{aligned} C_i &= \left(a^{[3]} - y_i\right)^2 \\ &= \left(\sigma\left(z^{[3]}\right) - y_i\right)^2 \\ &= \left(\sigma\left(w^{[3]}a^{[2]} + b^{[3]}\right) - y_i\right)^2 \end{aligned}$$

- We can visualize this relationship with the dependency tree to the left

Back Propagation: Chain Rule

- Now, let's compute $\frac{\partial C_i}{\partial w^{[3]}}$

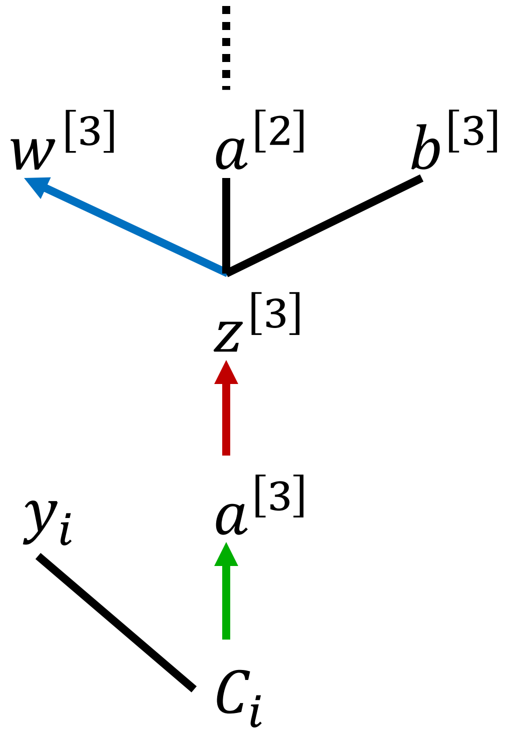
$$\frac{\partial C_i}{\partial w^{[3]}} = \frac{\partial C_i}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial w^{[3]}}$$

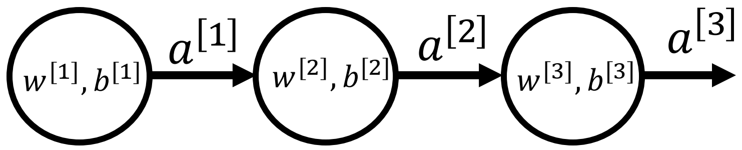
$$\frac{\partial C_i}{\partial a^{[3]}} = \frac{\partial \left(a^{[3]} - y_i\right)^2}{\partial a^{[3]}} = 2\left(a^{[3]} - y_i\right)$$

$$\frac{\partial a^{[3]}}{\partial z^{[3]}} = \frac{\partial \sigma\left(z^{[3]}\right)}{\partial z^{[3]}} = \sigma\left(z^{[3]}\right)\left(1 - \sigma\left(z^{[3]}\right)\right)$$

$$\frac{\partial z^{[3]}}{\partial w^{[3]}} = \frac{\partial \left(w^{[3]} a^{[2]} + b^{[3]}\right)}{\partial w^{[3]}} = a^{[2]}$$

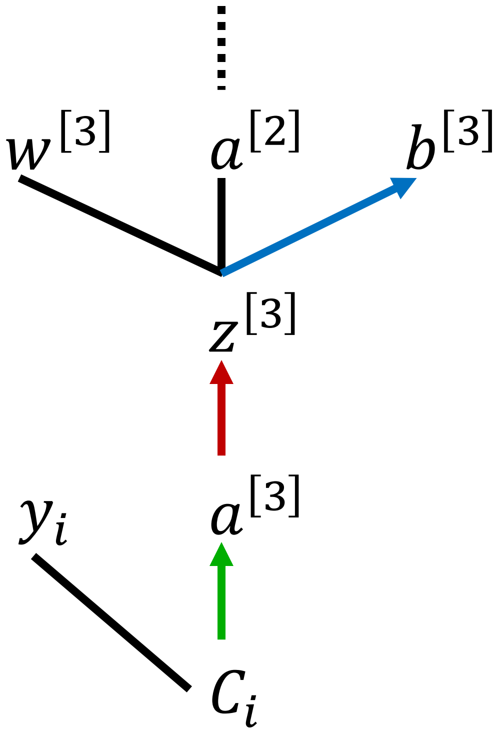
$$\Rightarrow \frac{\partial C_i}{\partial w^{[3]}} = 2\left(a^{[3]} - y_i\right)\sigma\left(z^{[3]}\right)\left(1 - \sigma\left(z^{[3]}\right)\right)a^{[2]}$$





Back Propagation: Chain Rule

- Similarly, we can compute $\frac{\partial C_i}{\partial b^{[3]}}$

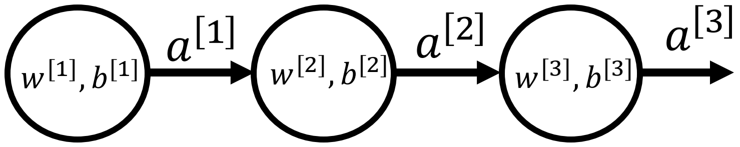


$$\frac{\partial C_i}{\partial b^{[3]}} = \frac{\partial C_i}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial b^{[3]}}$$

$$\frac{\partial C_i}{\partial a^{[3]}} = \frac{\partial \left(a^{[3]} - y_i\right)^2}{\partial a^{[3]}} = 2\left(a^{[3]} - y_i\right)$$

$$\frac{\partial a^{[3]}}{\partial z^{[3]}} = \frac{\partial \sigma\left(z^{[3]}\right)}{\partial z^{[3]}} = \sigma\left(z^{[3]}\right)\left(1 - \sigma\left(z^{[3]}\right)\right)$$

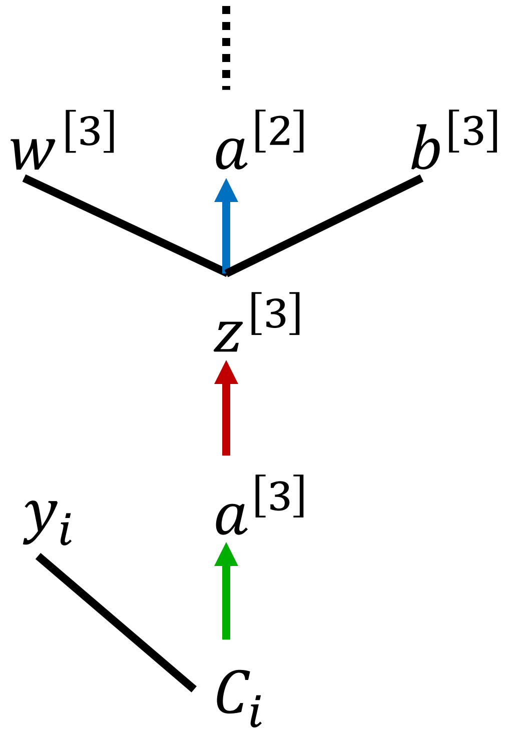
$$\frac{\partial z^{[3]}}{\partial b^{[3]}} = \frac{\partial \left(w^{[3]} a^{[2]} + b^{[3]}\right)}{\partial b^{[3]}} = 1$$



$$\Rightarrow \frac{\partial C_i}{\partial b^{[3]}} = 2\left(a^{[3]} - y_i\right)\sigma\left(z^{[3]}\right)\left(1 - \sigma\left(z^{[3]}\right)\right)$$

Back Propagation: Chain Rule

- Finally, we compute $\frac{\partial C_i}{\partial a^{[2]}}$

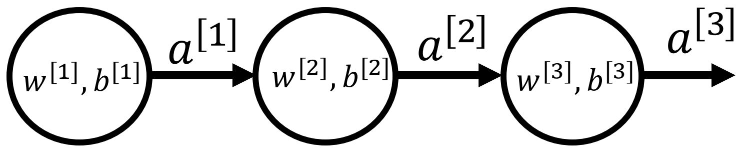


$$\frac{\partial C_i}{\partial a^{[2]}} = \frac{\partial C_i}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}}$$

$$\frac{\partial C_i}{\partial a^{[3]}} = \frac{\partial \left(a^{[3]} - y_i\right)^2}{\partial a^{[3]}} = 2\left(a^{[3]} - y_i\right)$$

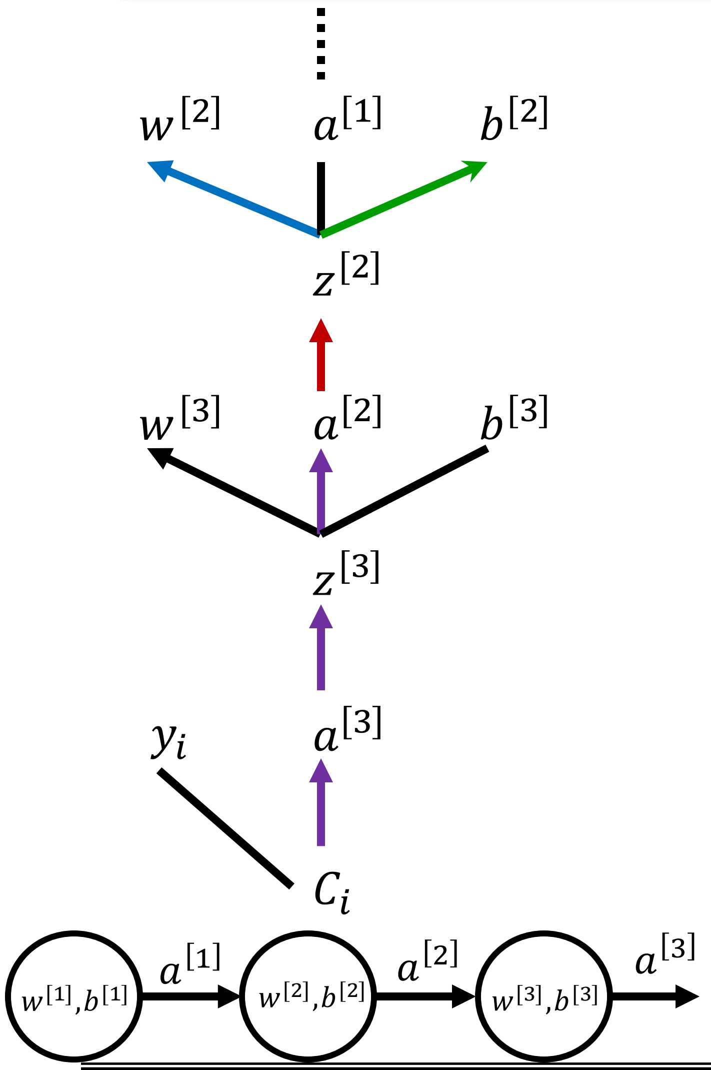
$$\frac{\partial a^{[3]}}{\partial z^{[3]}} = \frac{\partial \sigma\left(z^{[3]}\right)}{\partial z^{[3]}} = \sigma\left(z^{[3]}\right)\left(1 - \sigma\left(z^{[3]}\right)\right)$$

$$\frac{\partial z^{[3]}}{\partial a^{[2]}} = \frac{\partial \left(w^{[3]} a^{[2]} + b^{[3]}\right)}{\partial a^{[2]}} = w^{[3]}$$



$$\Rightarrow \frac{\partial C_i}{\partial a^{[2]}} = 2\left(a^{[3]} - y_i\right)\sigma\left(z^{[3]}\right)\left(1 - \sigma\left(z^{[3]}\right)\right) w^{[3]}$$

Back Propagation: Chain Rule



- We can now *propagate* the gradient calculations we have done *backwards* in order to find the gradients for weights/biases in layer 2

Has been computed already in previous slide

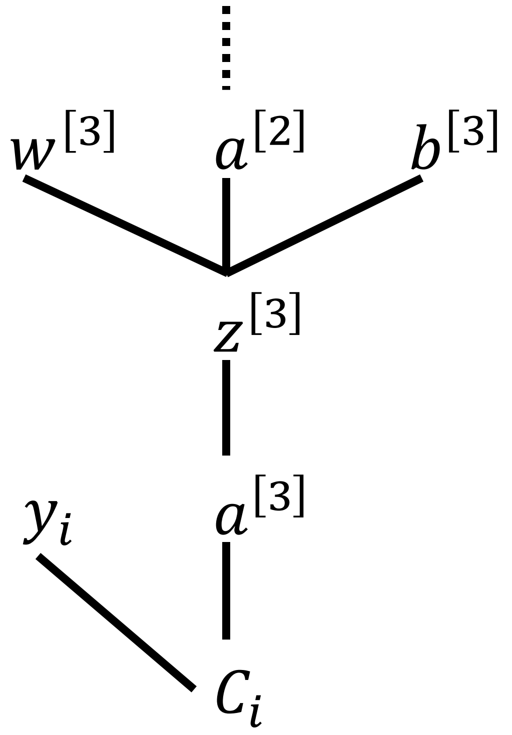
$$\frac{\partial C_i}{\partial w^{[2]}} = \frac{\partial C_i}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}}$$

$$= \frac{\partial C_i}{\partial a^{[2]}} \sigma\left(z^{[2]}\right)\left(1 - \sigma\left(z^{[2]}\right)\right) a^{[1]}$$

$$\frac{\partial C_i}{\partial b^{[2]}} = \frac{\partial C_i}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$$

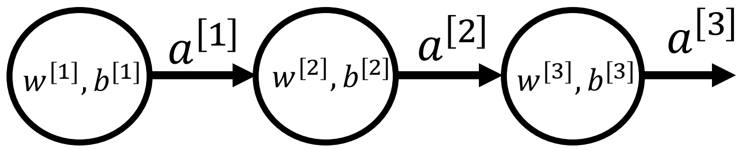
$$= \frac{\partial C_i}{\partial a^{[2]}} \sigma\left(z^{[2]}\right)\left(1 - \sigma\left(z^{[2]}\right)\right)$$

Back Propagation: Full Cost Function

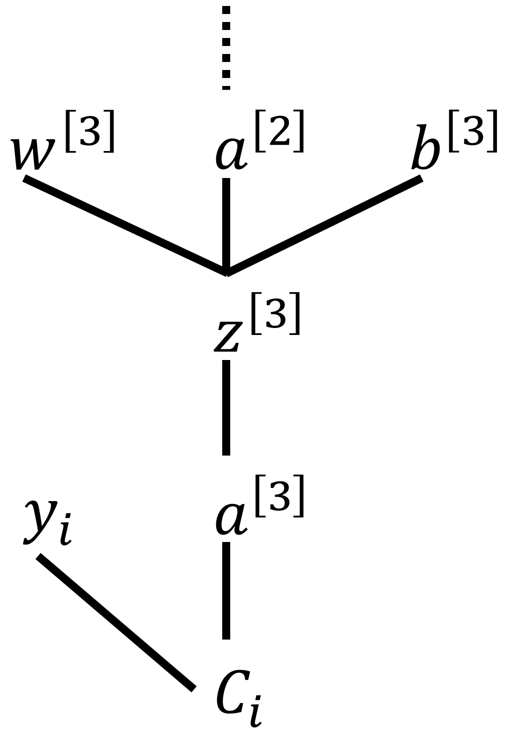


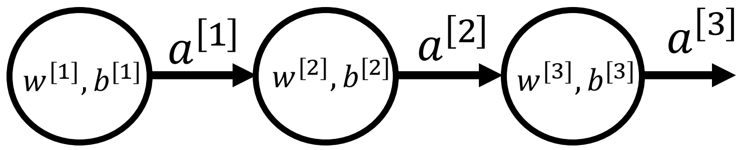
- In order to the partial derivative of the full cost function C with respect to a weight (say, $w^{[3]}$), we need to average the gradients of cost with respect to that weight for all n training samples

$$\frac{\partial C}{\partial w^{[3]}} = \frac{1}{n}\sum_{i=1}^{n} \frac{\partial C_i}{\partial w^{[3]}}$$



Back Propagation: Gradient of Cost Function



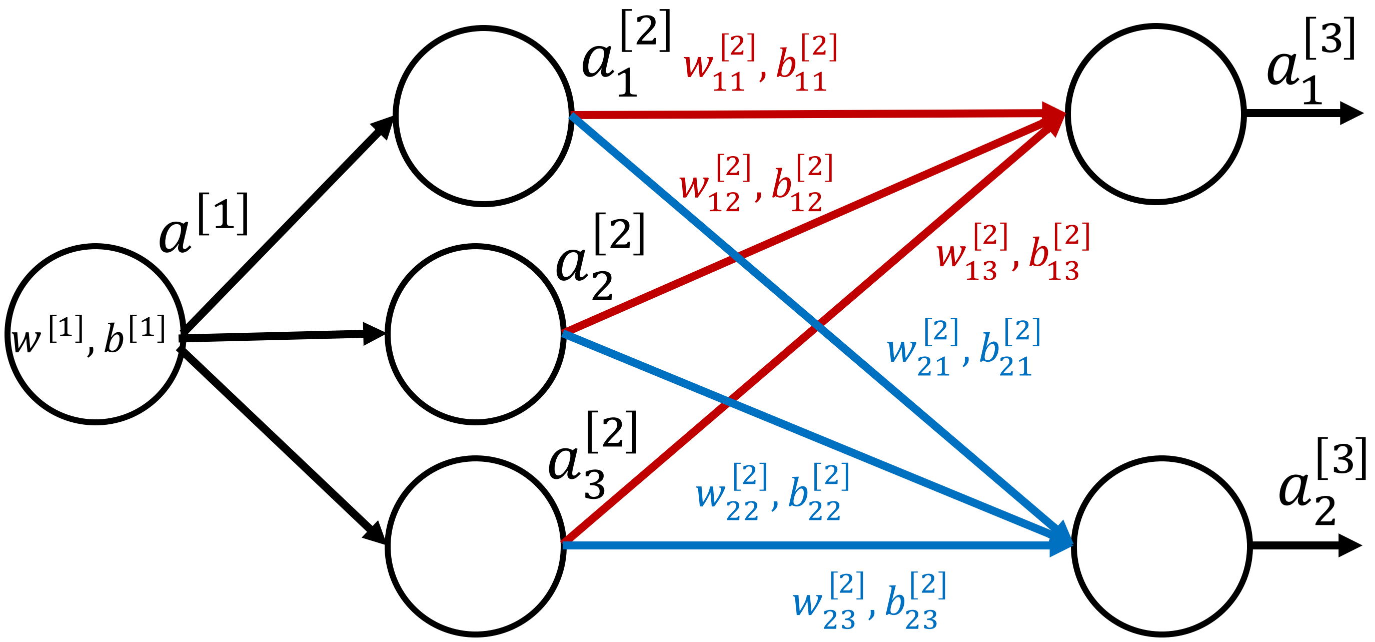


- Finally, we can define the full gradient vector of C as follows:

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{[1]}} \\ \frac{\partial C}{\partial b^{[1]}} \\ \frac{\partial C}{\partial w^{[2]}} \\ \frac{\partial C}{\partial b^{[2]}} \\ \frac{\partial C}{\partial w^{[3]}} \\ \frac{\partial C}{\partial b^{[3]}} \end{bmatrix}$$

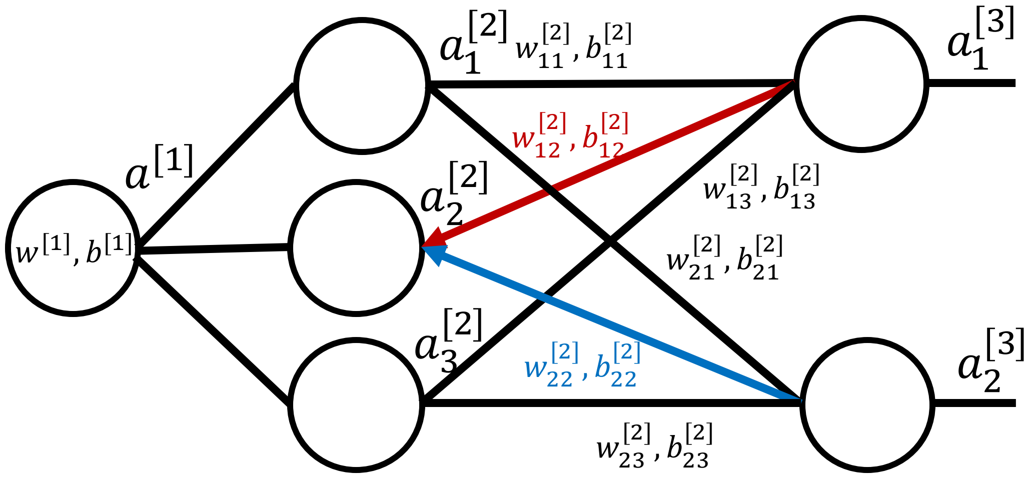
Back Propagation: More Perceptrons

- Suppose instead that you now have a slightly more complex neural network with multiple perceptrons in a layer



$w^{[1]}, b^{[1]}$
$a^{[1]}$
$a_1^{[2]}$
$a_2^{[2]}$
$a_3^{[2]}$
$w_{11}^{[2]}, b_{11}^{[2]}$
$w_{12}^{[2]}, b_{12}^{[2]}$
$w_{13}^{[2]}, b_{13}^{[2]}$
$w_{21}^{[2]}, b_{21}^{[2]}$
$w_{22}^{[2]}, b_{22}^{[2]}$
$w_{23}^{[2]}, b_{23}^{[2]}$
$a_1^{[3]}$
$a_2^{[3]}$

Back Propagation: More Perceptrons



$w^{[1]}, b^{[1]}$
$a^{[1]}$
$a_1^{[2]}$
$w_{11}^{[2]}, b_{11}^{[2]}$
$w_{12}^{[2]}, b_{12}^{[2]}$
$a_2^{[2]}$
$w_{13}^{[2]}, b_{13}^{[2]}$
$w_{21}^{[2]}, b_{21}^{[2]}$
$a_3^{[2]}$
$w_{22}^{[2]}, b_{22}^{[2]}$
$w_{23}^{[2]}, b_{23}^{[2]}$
$a_1^{[3]}$
$a_2^{[3]}$

- To find $\frac{\partial C_i}{\partial a_2^{[2]}}$, we need to consider the back propagation from **both perceptrons** in the final layer

$$\frac{\partial C_i}{\partial a_2^{[2]}} = \frac{\partial C_i}{\partial a_1^{[3]}} \frac{\partial a_1^{[3]}}{\partial z_1^{[3]}} \frac{\partial z_1^{[3]}}{\partial a_2^{[2]}} + \frac{\partial C_i}{\partial a_2^{[3]}} \frac{\partial a_2^{[3]}}{\partial z_2^{[3]}} \frac{\partial z_2^{[3]}}{\partial a_2^{[2]}}$$

In general, if there are n_L neurons in layer L, then

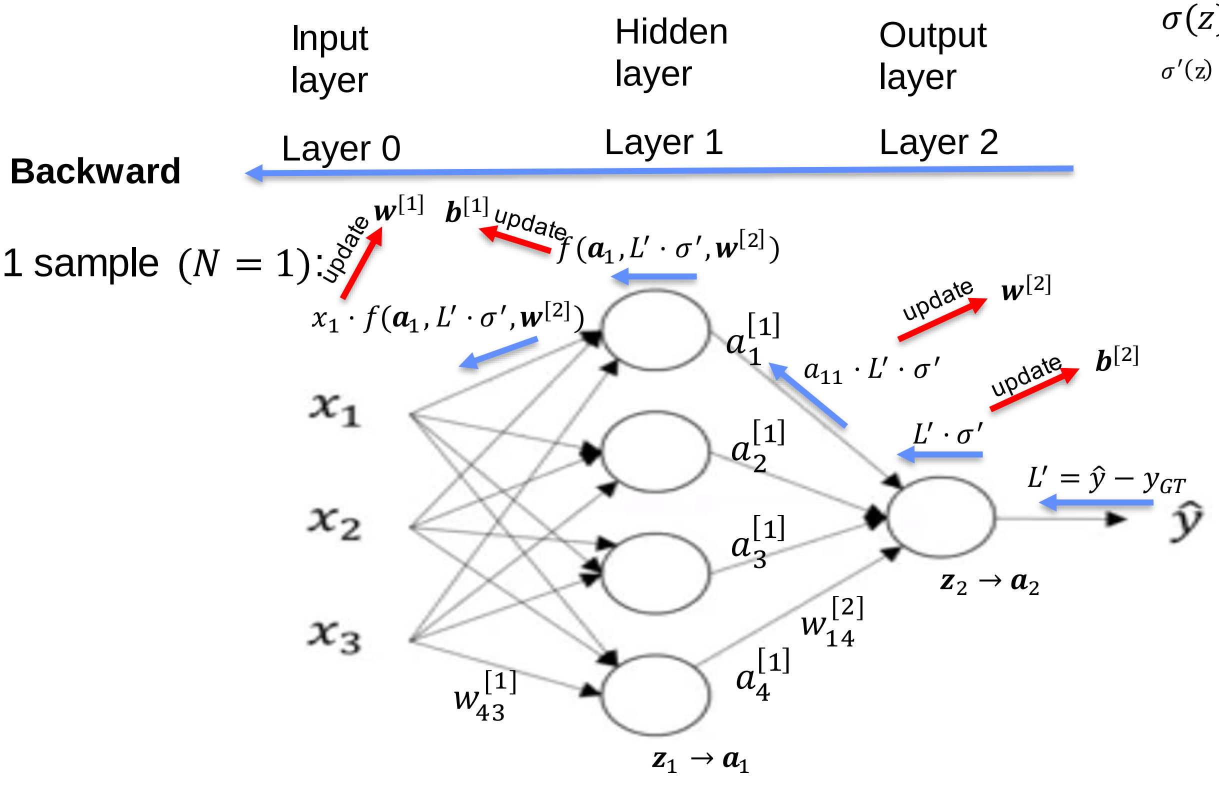
$$\frac{\partial C_i}{\partial a_k^{[L-1]}} = \sum_{i=1}^{n_L} \frac{\partial C_i}{\partial a_i^{[L]}} \frac{\partial a_i^{[L]}}{\partial z_i^{[L]}} \frac{\partial z_i^{[L]}}{\partial a_k^{[L-1]}}$$

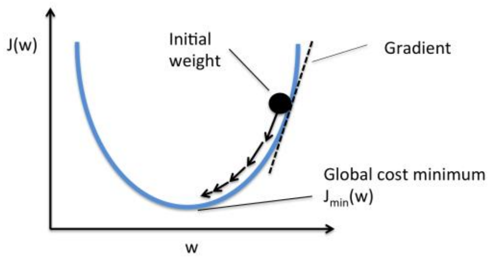
Backpropagation: Gradient Descent

Derivative of Sigmoid function $\sigma(z)$:

$$\sigma'(z) = \frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z}\frac{1}{1+e^{-z}}$$
$$= \frac{e^{-z}}{(1+e^{-z})^2}$$
$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2}$$
$$= \frac{1}{1+e^{-z}} - \frac{1}{(1+e^{-z})^2}$$
$$= \sigma(z) - \sigma^2(z)$$
$$= \sigma(z)(1-\sigma(z))$$

Assume 1 sample $(N = 1)$:





Purpose of backpropagation:

Apply *chain rule of derivative* to update parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$