Lecture 10: Deep Learning Multiclass Classification

Fall 2022

Kai-Wei Chang

CS @ UCLA

kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Announcements

- Midterm on 11/3
  - Online open book exam
  - Exam time: 100 min
- Hw1 is due today!
  - 24 hour late credit
- Quiz 3 is due today!

What you will learn today

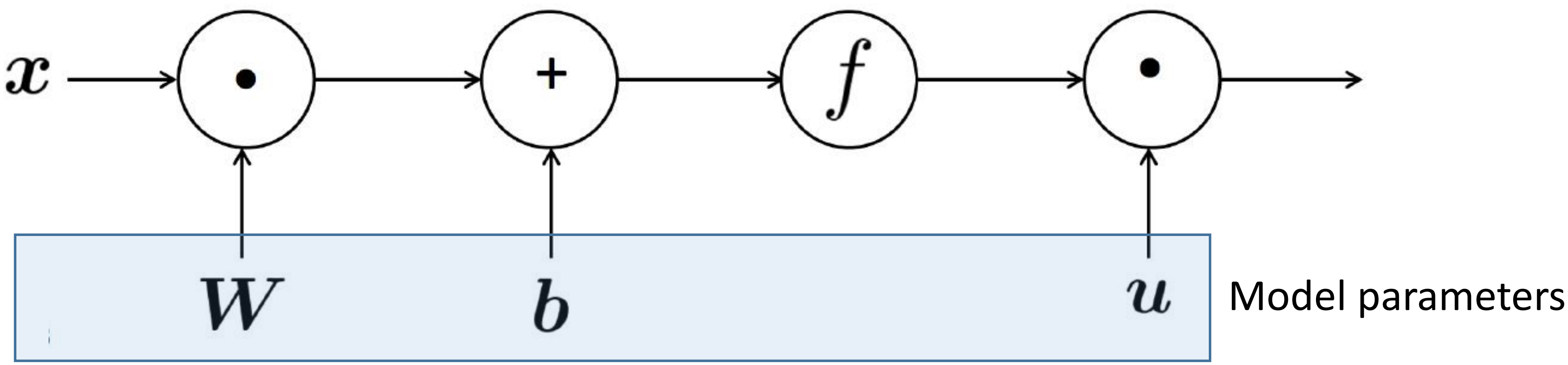
- Deep Learning architectures (not in exam)
- Multiclass Classification
  - One against all
  - One vs one
  - Multinomial Logistic Regression
    - Softmax function

Backpropagation through Computation Graphs

Computation Graphs and Backpropagation

- Consider the NN on the right
- We represent NN as a graph

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

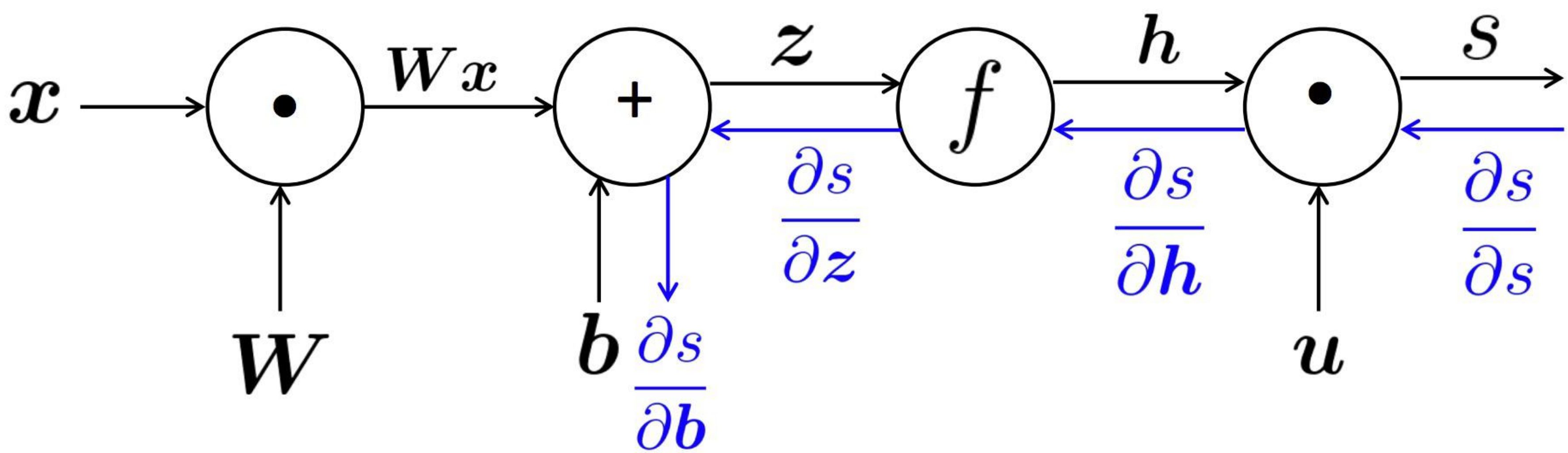


Back Propagation

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

- Compute $\frac{\partial s}{\partial \boldsymbol{b}}$

Chain Rule: $\frac{\partial s}{\partial b} = \frac{\partial s}{\partial z}\frac{\partial z}{\partial b} = \cdots$

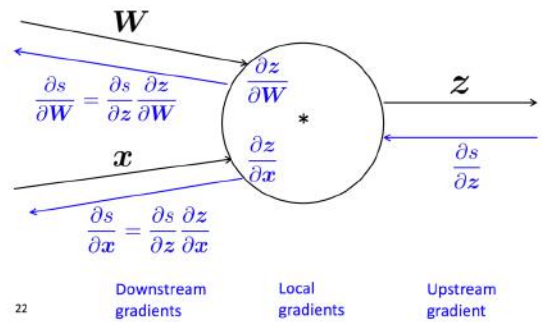


Why you should understand Backprop

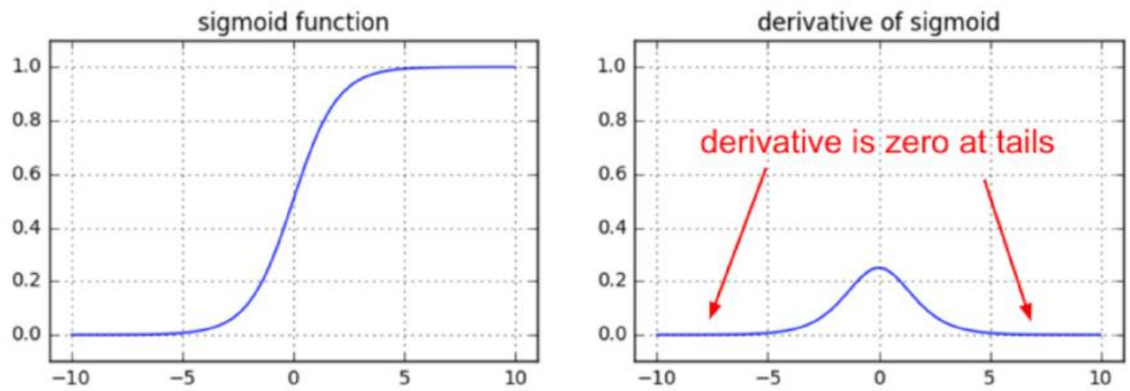
- Modern deep learning library implements backprop as a black-box for you
  - You can take a plane without knowing why it flies
  - but you're designing aircraft…
- Backpropagation doesn't always work perfectly.
  - Understanding why is crucial for debugging and improving models

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Example: Gradient of sigmoid




```
z = 1/(1 + np.exp(-np.dot(W, x))) # forward pass
dx = np.dot(W.T, z*(1-z)) # backward pass: local gradient for x
dW = np.outer(z*(1-z), x) # backward pass: local gradient for W
```



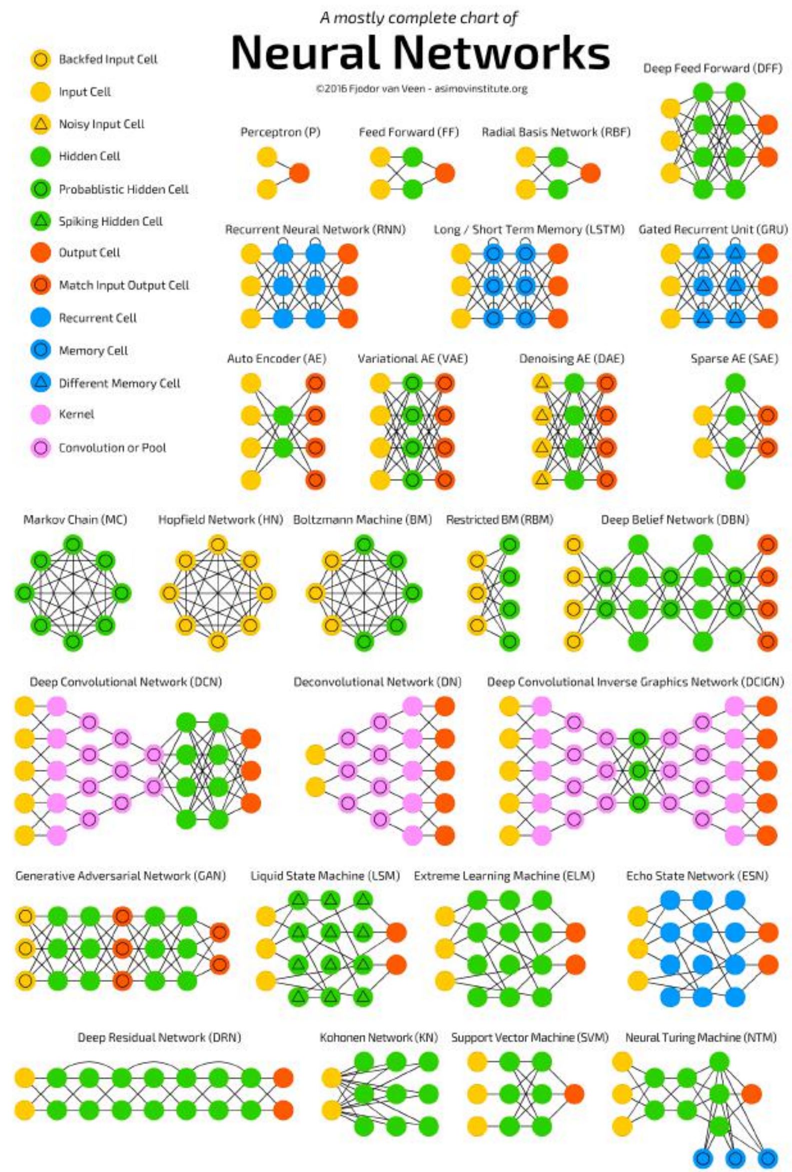
vanish gradient issue

More Details

- Parameter Initialization
  - Normally initialize weights to small random values; various designs
- Optimizer
  - Usually SGD works
  - Several SGD variants (e.g,. ADAM) automatically adjust learning rate based on an accumulated gradient

A neural network zoo

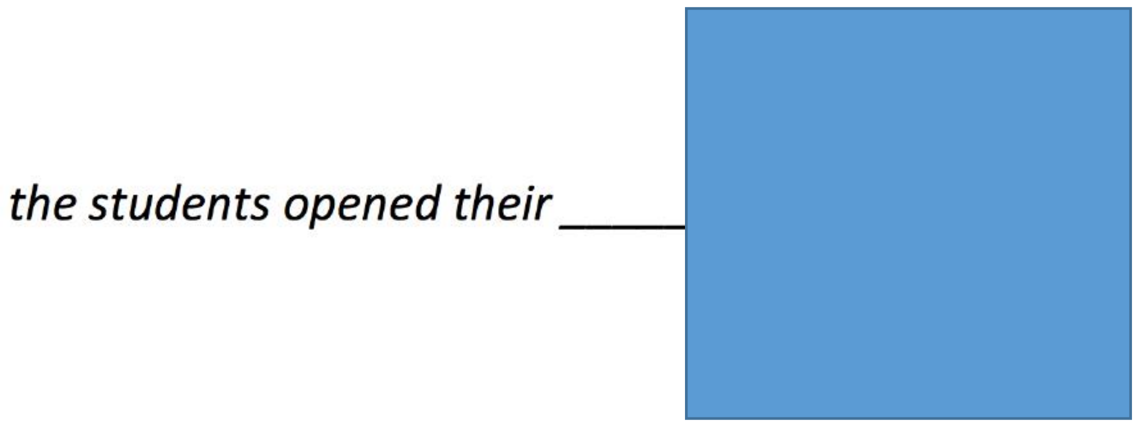
- The flexibility of NN allows us to try out different ideas
- However,
  there is no magic



Modeling with Neural Networks
(Advanced Topic/Not Included in Final)

Example – Language Model

- Predict next word



Idea 1:
A fixed-window neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

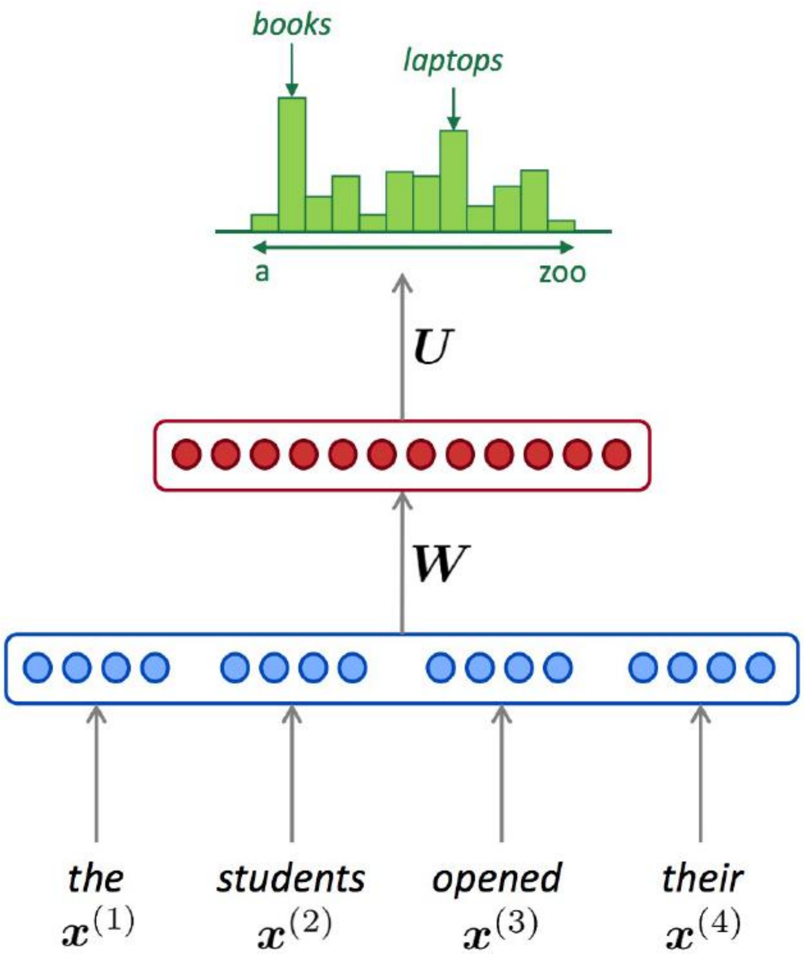
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

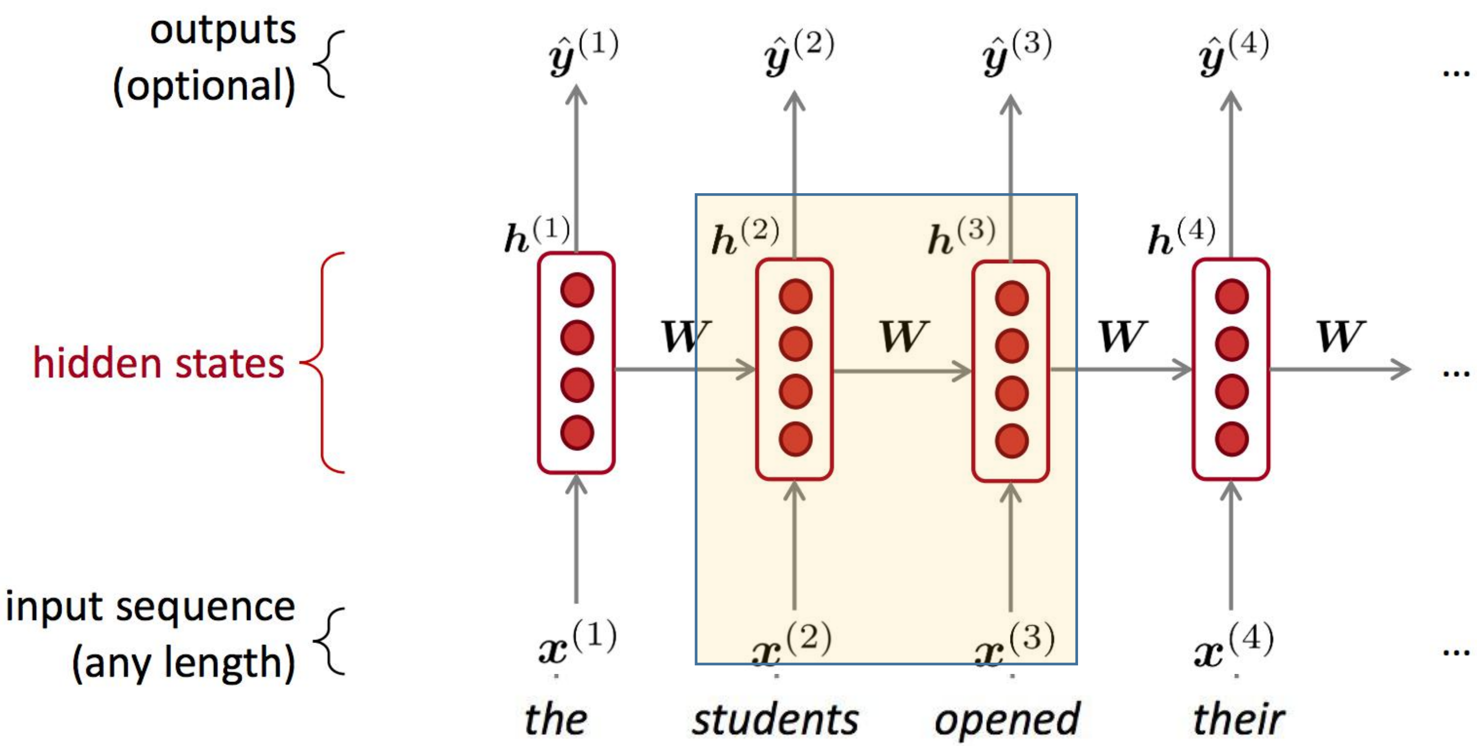
$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$

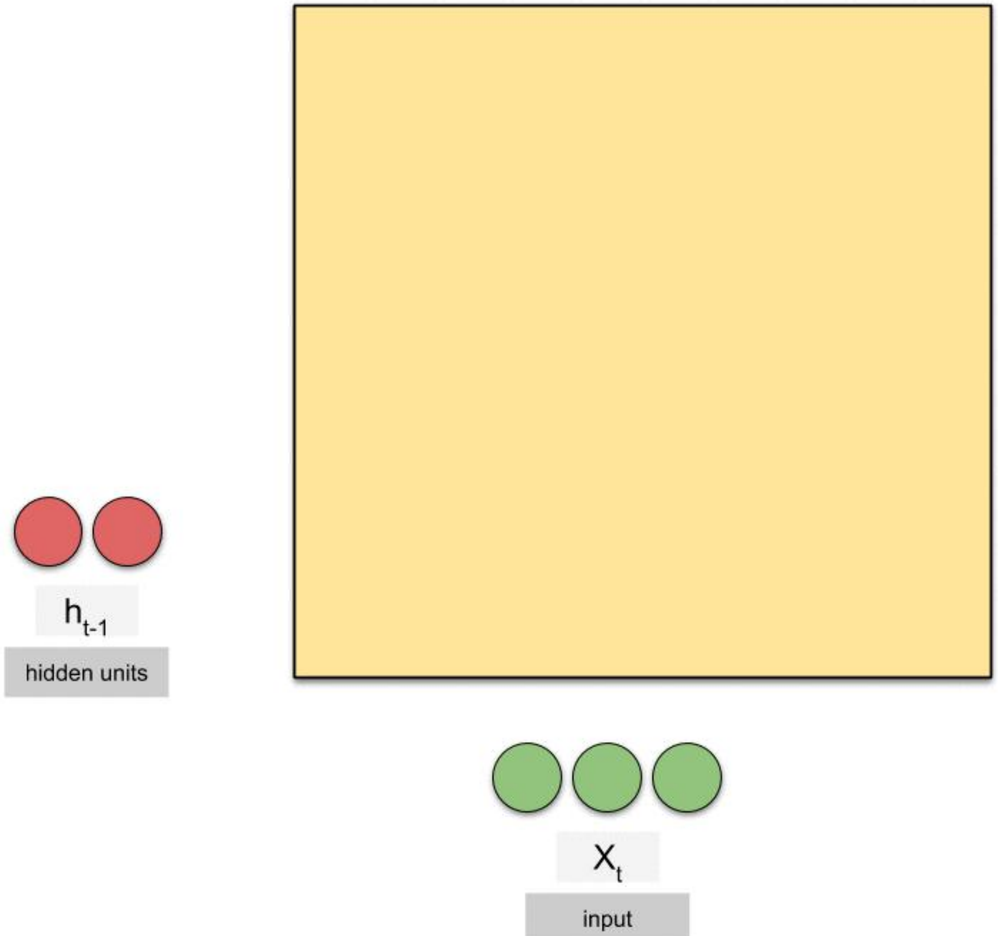


Idea 2:
Recurrent Neural Networks (RNN)



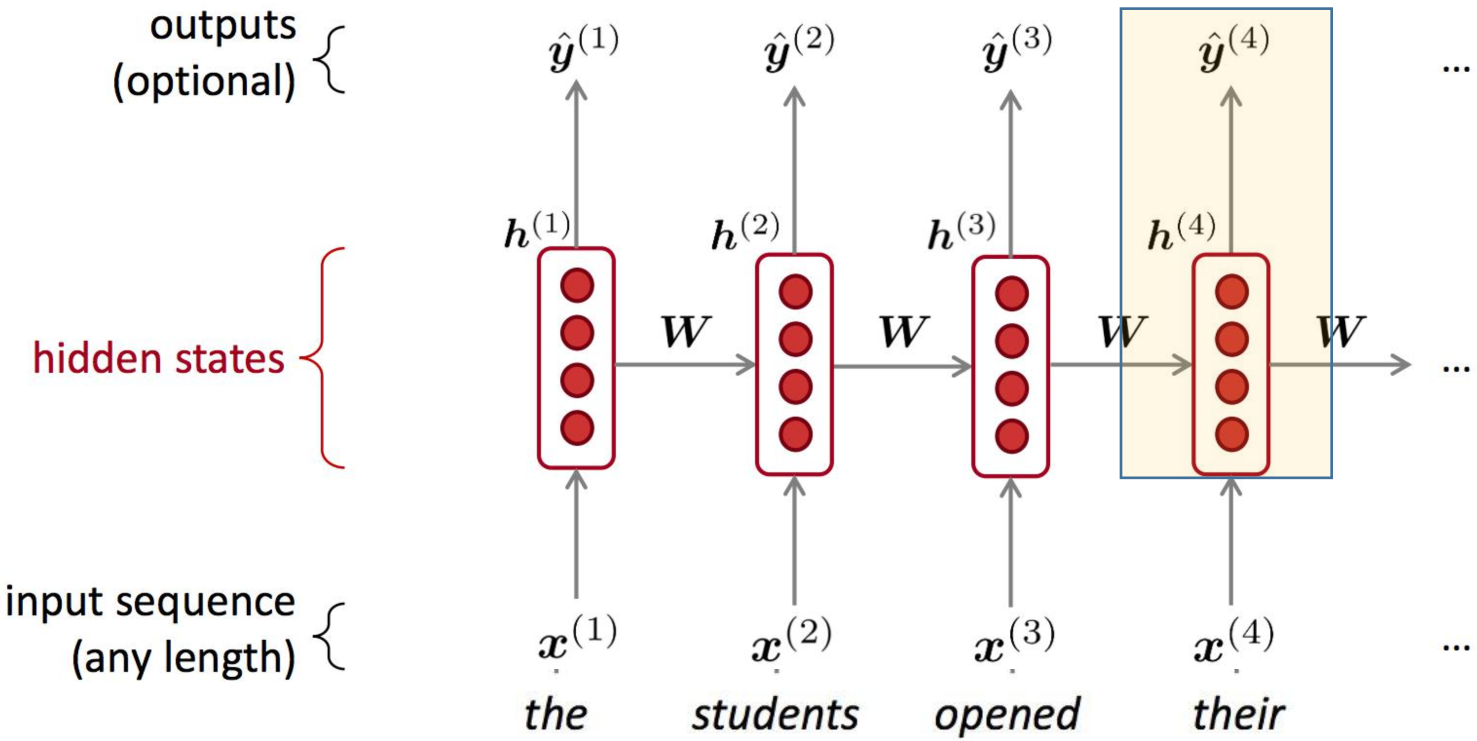
Core idea: Apply the same weights W *repeatedly*

Recurrent Neural Network



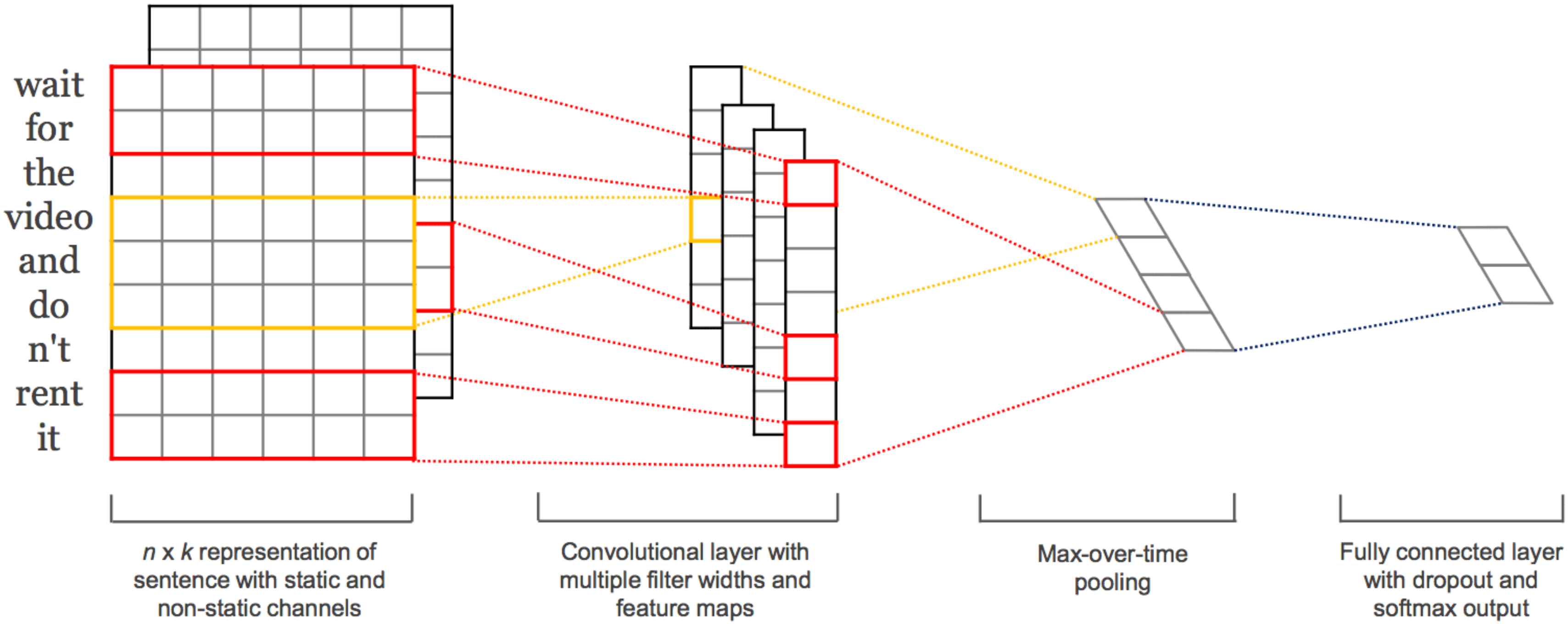
https://towardsdatascience.com/animated-rnn-lstm-and-gru-ef124d06cf45

Prediction using Latent State



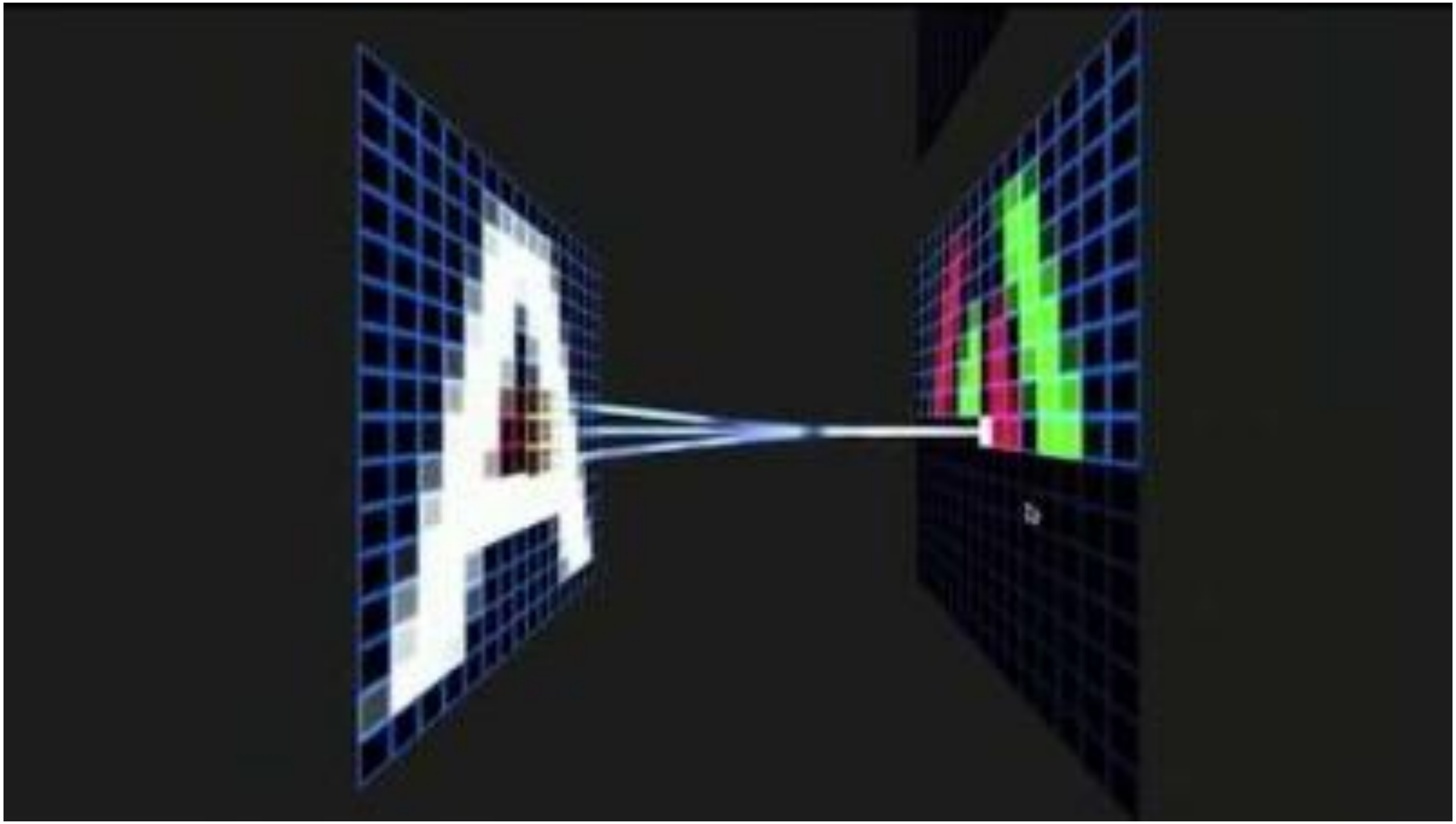
<u>**Core idea:**</u> Apply the same weights W *repeatedly*

Idea 3: Convolutional NN



"Convolutional Neural Networks for Sentence Classification", 2014.

Convolutional NN



https://www.youtube.com/watch?v=f0t-OCG79-U

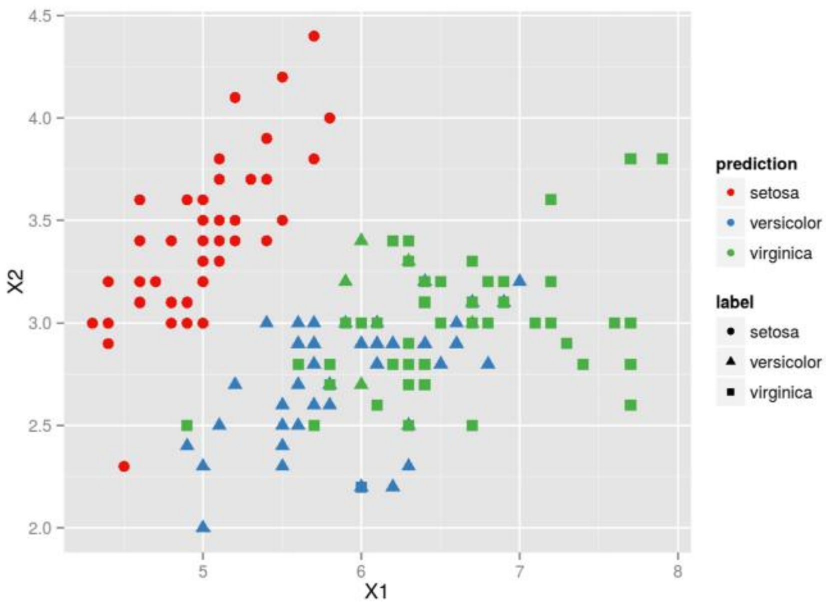
Multi-Class Classification

This Lecture

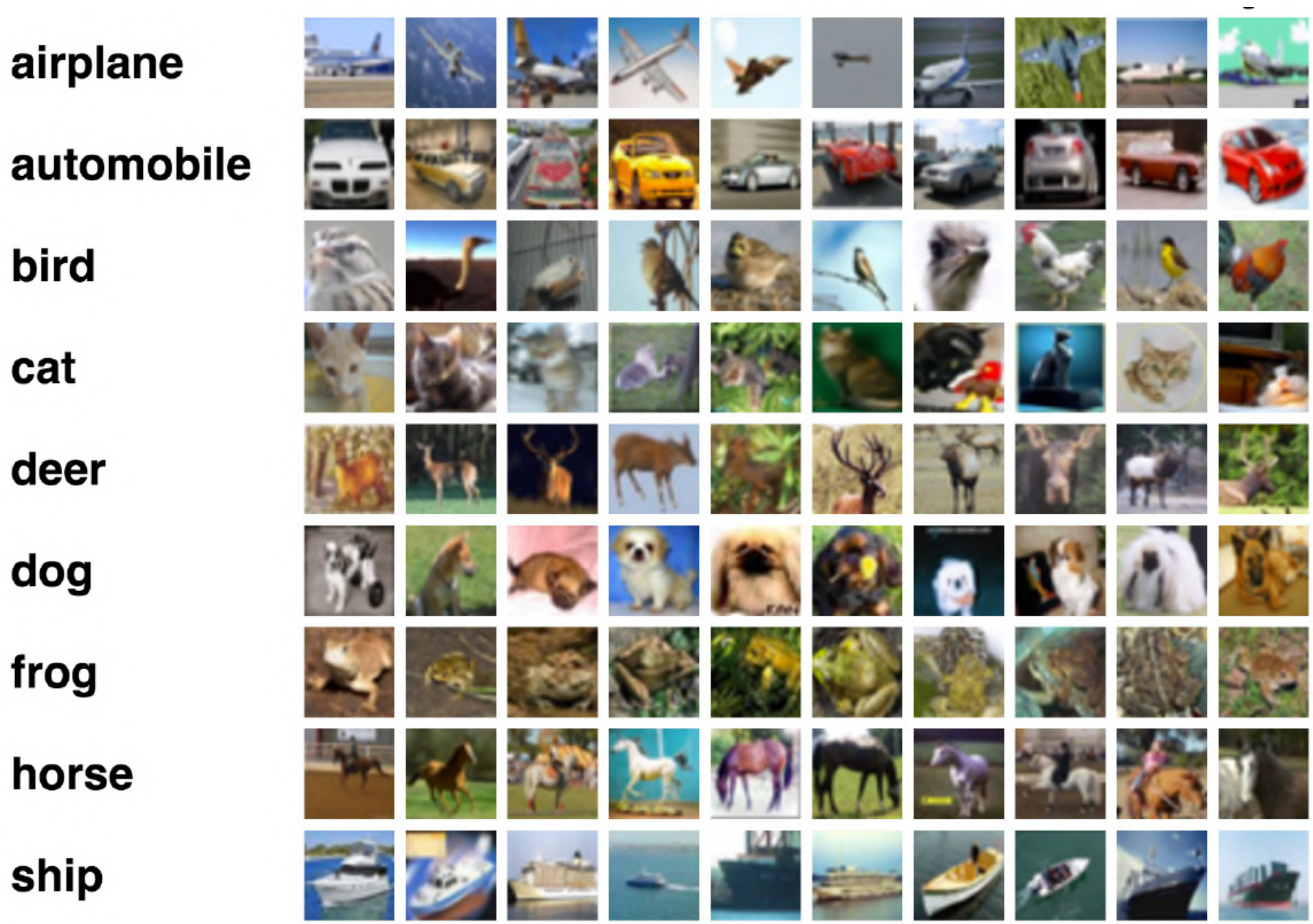
- Multiclass classification overview
- Reducing multiclass to binary
  - One-against-all & One-vs-one
- One classifier approach
  - Multiclass logistic regression

What is multiclass

- Output $\in \{1,2,3, \ldots K\}$
  - In some cases, output space can be very large (i.e., K is very large)
- Each input belongs to exactly one class
  (c.f. in multilabel, input belongs to many classes)



Example applications



Two key ideas to solve multiclass

- Reducing multiclass to binary
  - Decompose the multiclass prediction into multiple binary decisions
  - Make the final decision based on these binary classifiers
- Training a single classifier
  - Consider all classes simultaneously

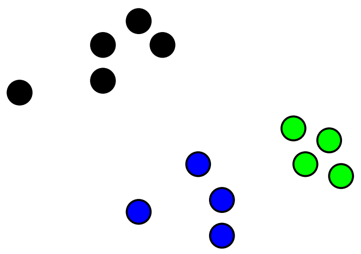
This Lecture

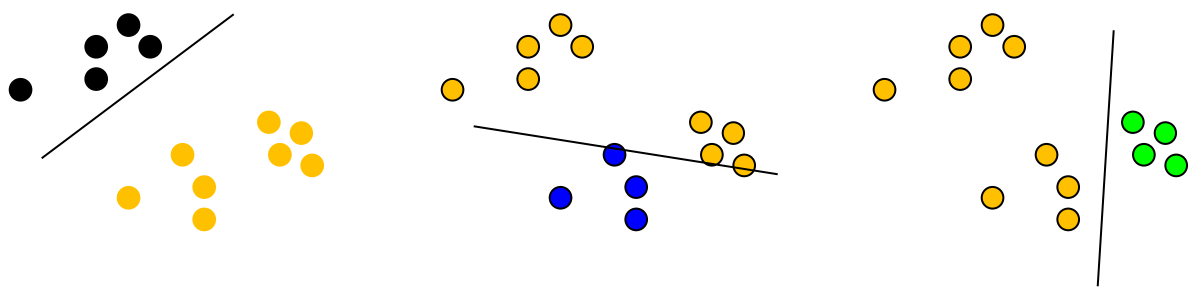
- Multiclass classification overview
- Reducing multiclass to binary
  - One-against-all & One-vs-one

One against all strategy

One against All learning

- Multiclass classifier
  - Function $f : R^n \rightarrow \{1,2,3,...,k\}$
- Decompose into binary problems

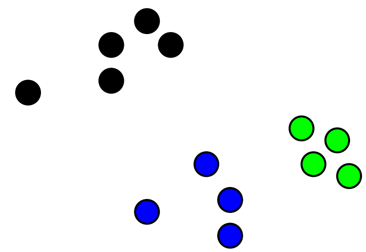




One-against-All learning algorithm

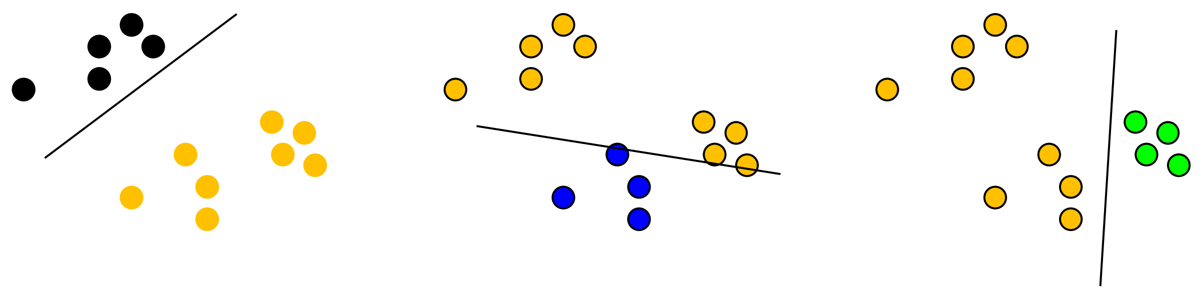
- Learning: Given a dataset $D = \{(x_i, y_i)\}$
  $x_i \in R^n, y_i \in \{1,2,3, \dots K\}$
- Decompose into K binary classification tasks
  - Learn K models: $w_1, w_2, w_3, \dots w_K$
  - For class k, construct a binary classification task as:
    - Positive examples: Elements of D with label k
    - Negative examples: All other elements of D
  - The binary classification can be solved by any algorithm we have seen

One against All learning



- Multiclass classifier
  - Function $f : R^n \rightarrow \{1,2,3,...,k\}$
- Decompose into binary problems

Ideal case: only the correct label will have a positive score



$w_{black}^T x > 0$ $\quad$ $w_{blue}^T x > 0$ $\quad$ $w_{green}^T x > 0$

One-against-All Inference

- Learning: Given a dataset $D = \{(x_i, y_i)\}$
  $x_i \in R^n, y_i \in \{1,2,3, \dots K\}$
- Decompose into K binary classification tasks
  - Learn K models: $w_1, w_2, w_3, \dots w_K$
- Inference: "Winner takes all"
  - $\hat{y} = \mathrm{argmax}_{y \in \{1,2,\dots K\}}\, w_y^T\, x$

For example: $y = \mathrm{argmax}(w_{black}^T\, x\,,\ w_{blue}^T\, x,\ w_{green}^T\, x)$

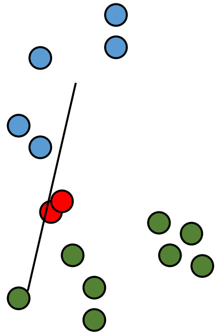
  - An instance of the general form

$$\hat{y} = \mathrm{argmax}_{\boldsymbol{y} \in \mathcal{Y}}\, f(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{x})$$

$w = \{w_1, w_2, \dots w_K\},\ f(\boldsymbol{y}; \boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}_y^T\, x$

One-against-All analysis

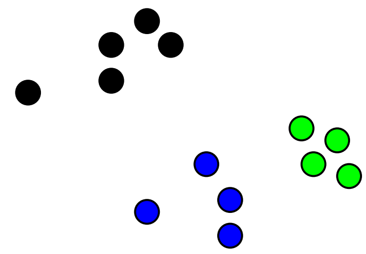
- Not always possible to learn
  - Assumption: each class individually separable from all the others
- Need to make sure the range of all classifiers is the same –K classifiers are trained independently.
- Easy to implement; work well in practice

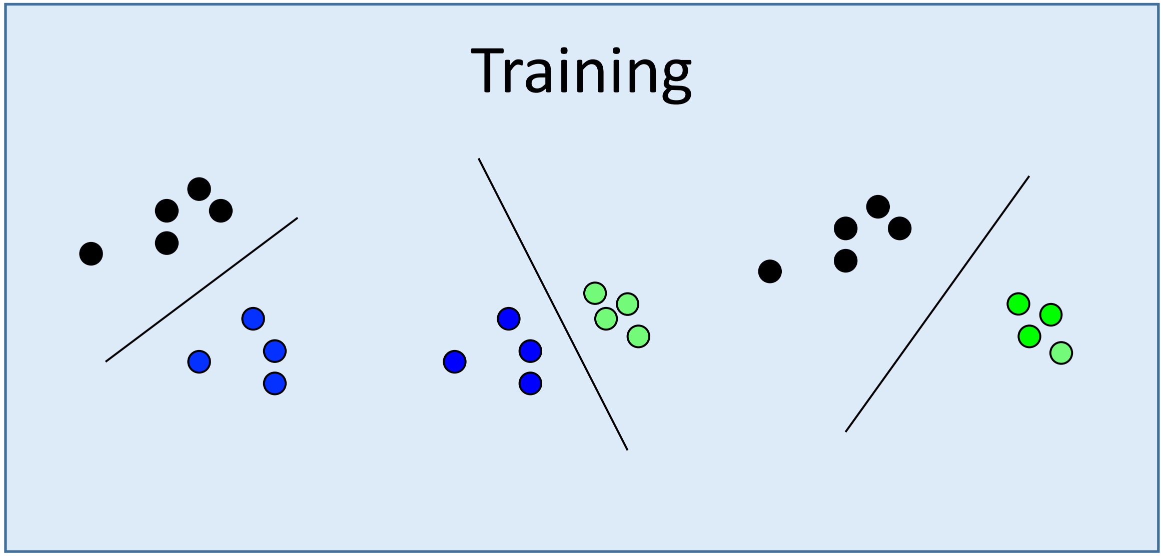


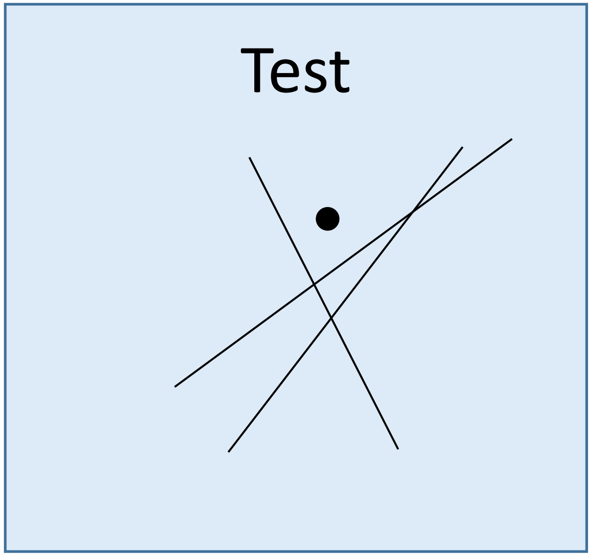
One v.s. One (All against All) strategy

One v.s. One learning

- Multiclass classifier
  - Function $f : R^n \rightarrow \{1,2,3,...,k\}$
- Decompose into binary problems





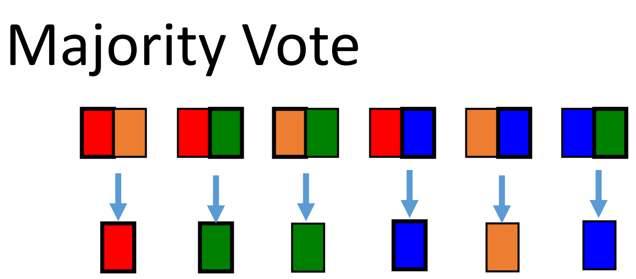


One-v.s-One learning algorithm

- Learning: Given a dataset $D = \{(x_i, y_i)\}$
  $x_i \in R^n, y_i \in \{1,2,3, \dots K\}$
- Decompose into C(K,2) binary classification tasks
  - Learn C(K,2) models: $w_1, w_2, w_3, \dots w_{K*(K-1)/2}$
  - For each class pair (i,j), construct a binary classification task as:
    - Positive examples: Elements of D with label i
    - Negative examples Elements of D with label j
    - The binary classification can be solved by any algorithm we have seen

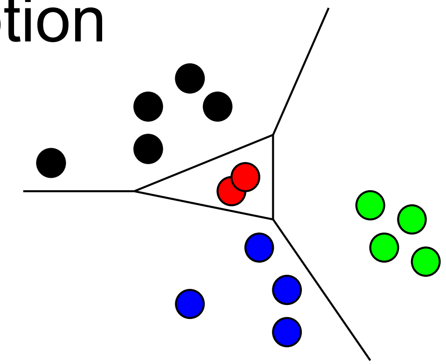
One-v.s-One Inference algorithm

- Decision Options:
  - More complex; each label gets k-1 votes
  - Output of the binary classifier may not coherent.
  - Majority: classify example x to take label i if i wins on x more often than j (j=1,…k)



Comparisons

- One against all
  - O(K) weight vectors to train and store
  - Training set of the binary classifiers may unbalanced
  - Less expressive; make a strong assumption



- One v.s. One (All v.s. All)
  - O(K^2) weight vectors to train and store
  - Size of training set for a pair of labels could be small
    ⇒ overfitting of the binary classifiers
  - Need large space to store model

Exercise

- Consider we have a 10-class classification problem with 29 features, each class has 1,000 examples.
- How many parameters are in total for linear models with one-vs-one?
- How many parameters are in total for linear models with one-against-all?
- How large is the training data for each one-vs-one classifier?
- How large is the training data for each one-against-all classifier?

Problems with Decompositions

- Learning optimizes over *local* metrics
  - Does not guarantee good *global* performance
  - We don't care about the performance of the *local* classifiers
- Poor decomposition $\Rightarrow$ poor performance
  - Difficult local problems
  - Irrelevant local problems
- Efficiency: e.g., All vs. All vs. One vs. All

Decomposition methods: Summary

- General Ideas:
  - Decompose the multiclass problem into many binary problems
  - Prediction depends on the decomposition
    - Constructs the multiclass label from the output of the binary classifiers
- Learning optimizes local correctness
  - Each binary classifier don't need to be globally correct and isn't aware of the prediction procedure

Multi-class Logistic Regression

Recall: (binary) logistic regression

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_i \log(1 + e^{-\mathrm{y_i}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathrm{i}})})$$

Assume labels are generated using the following probability distribution:

$$P(y=1|\mathbf{x},\mathbf{w}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}} = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

$$P(y=-1|\mathbf{x},\mathbf{w}) = \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}}$$

(multi-class) log-linear model

- Assumption:

Partition function

$$P(y|x, w) = \frac{\exp(w_y^T x)}{\sum_{y' \in \{1,2,\ldots K\}} \exp(w_{y'}^T x)}$$

- This is a valid probability assumption. Why?

soft-max function

- Example

$$softmax\left(\begin{bmatrix} -1 \\ 0 \\ 3 \\ 5 \end{bmatrix}\right) = \begin{bmatrix} 0.368/169.87 \\ 1/169.87 \\ 20.09/169.87 \\ 148.41/169.87 \end{bmatrix} = \begin{bmatrix} 0.002 \\ 0.006 \\ 0.118 \\ 0.874 \end{bmatrix}$$

Softmax

- Softmax: let s(y) be the score for output y here s(y)=$w^T\phi(x, y)$ (or $w_y^T x$) but it can be computed by other function.

$$P(y) = \frac{\exp(s(y))}{\sum_{y\prime\in\{1,2,\ldots K\}} \exp(s(y))}$$

Why we call it softmax?

- Softmax: let s(y) be the score for output y
  here s(y)=$w^T\phi(x,y)$ (or $w_y^T x$) but it can be computed by other function.

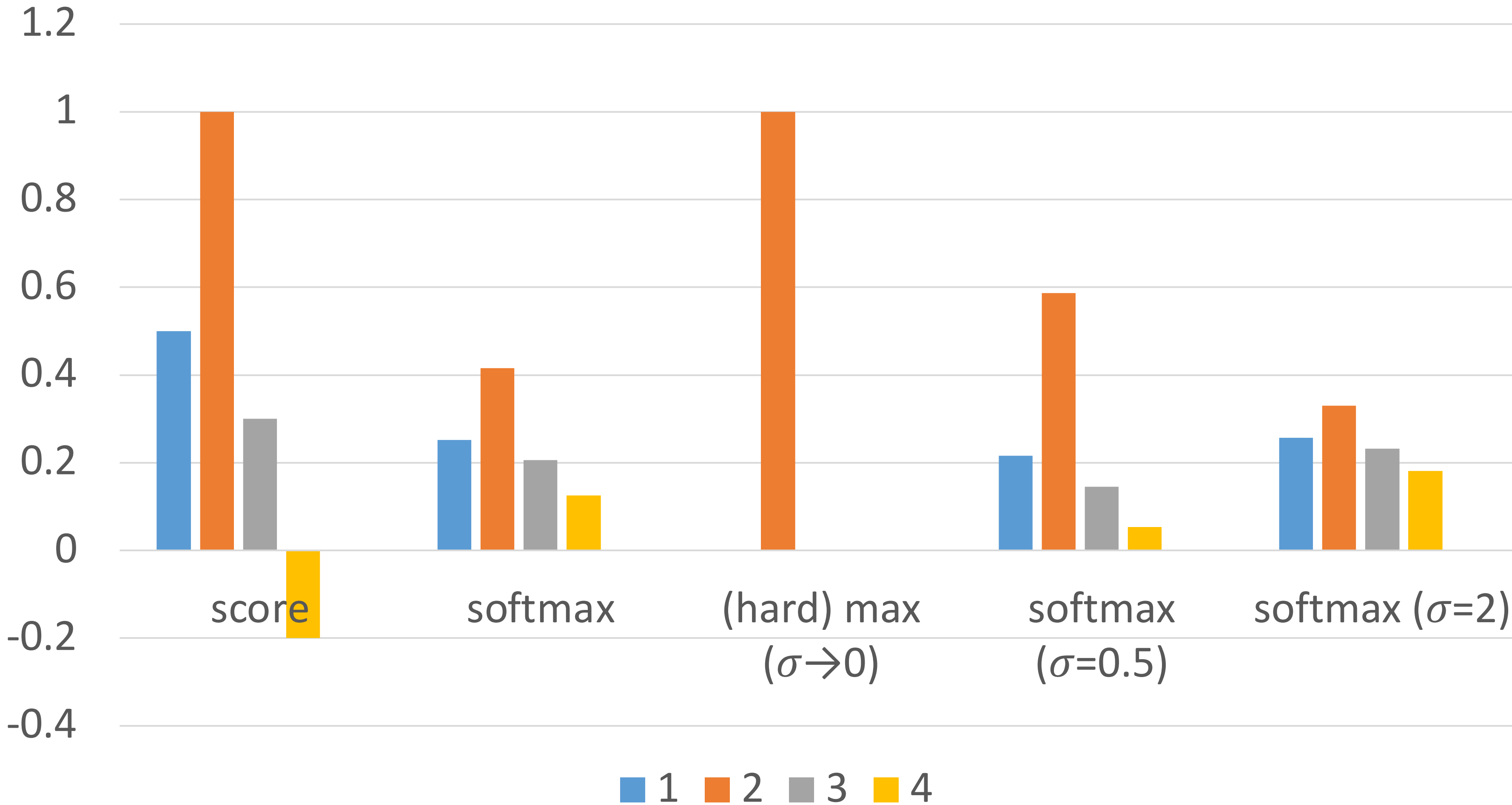
$$P(y) = \frac{\exp(s(y))}{\sum_{y' \in \{1,2,\ldots K\}} \exp(s(y))}$$

- We can control the peakedness of the distribution

$$P(y|\sigma) = \frac{\exp(s(y)/\sigma)}{\sum_{y' \in \{1,2,\ldots K\}} \exp(s(y/\sigma))}$$

Example

$S(1) = .5; \quad s(2)=1; \quad s(3)=0.3; \quad s(4)=-0.2$



Maximum log-likelihood estimation

- Training can be done by maximum log-likelihood estimation i.e. $\max_w \log P(D|w)$

$D=\{(x_i, y_i)\}$

$$P(D|w) = \Pi_i \frac{\exp\left(w_{y_i}^T x_i\right)}{\sum_{y' \in \{1,2,\ldots K\}} \exp(w_{y'}^T x_i)}$$

$$\log P(D|w) = \sum_i [w_{y_i}^T x_i - \log \sum_{y' \in \{1,2,\ldots K\}} \exp(w_{y'}^T x_i)]$$

Comparisons

- Log-linear model (multi-class)

$$\min_{w} \sum_i [\log \sum_{k\in\{1,2,\ldots K\}} \exp\left(w_k^T x_i\right) - w_{y_i}^T x_i]$$

- Log-linear mode (logistic regression)

$$\min_{\boldsymbol{w}} \quad \sum_i \log(1 + e^{-\mathrm{y}_\mathrm{i}(\mathbf{w}^\mathrm{T}\mathbf{x}_\mathrm{i})})$$

Reduction v.s. single classifier

- Reduction
  - Future-proof: if we improve the binary classification model ⇒ improve muti-class classifier
  - Easy to implement
- Single classifier
  - Global optimization: directly minimize the empirical loss; easier for joint prediction