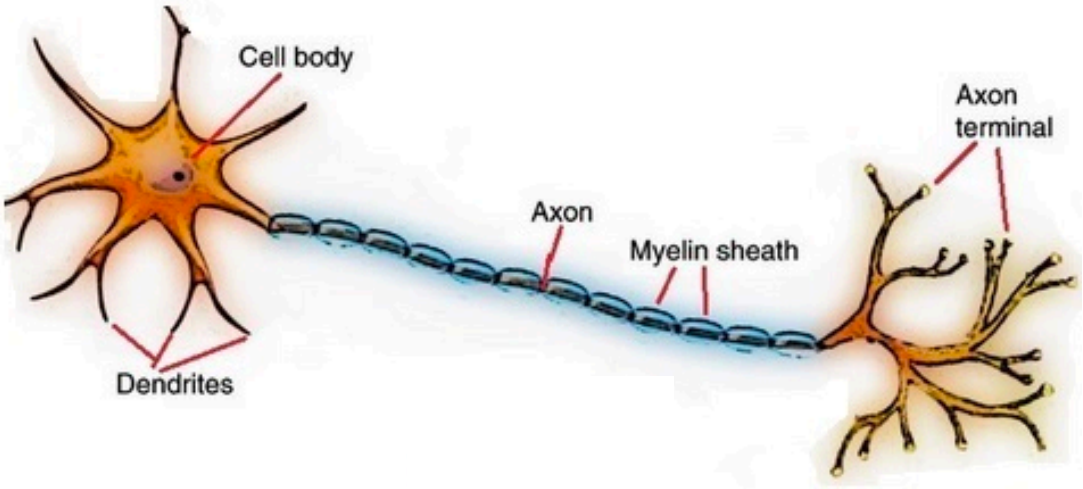
More Perceptron

Instructor: Wei Xu

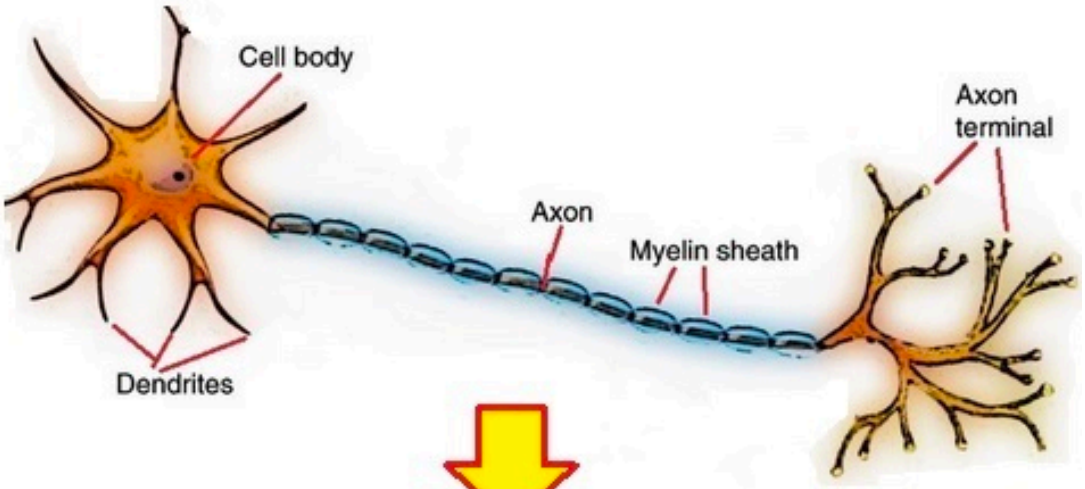
Some slides adapted from Dan Jurfasky, Brendan O'Connor and Marine Carpuat

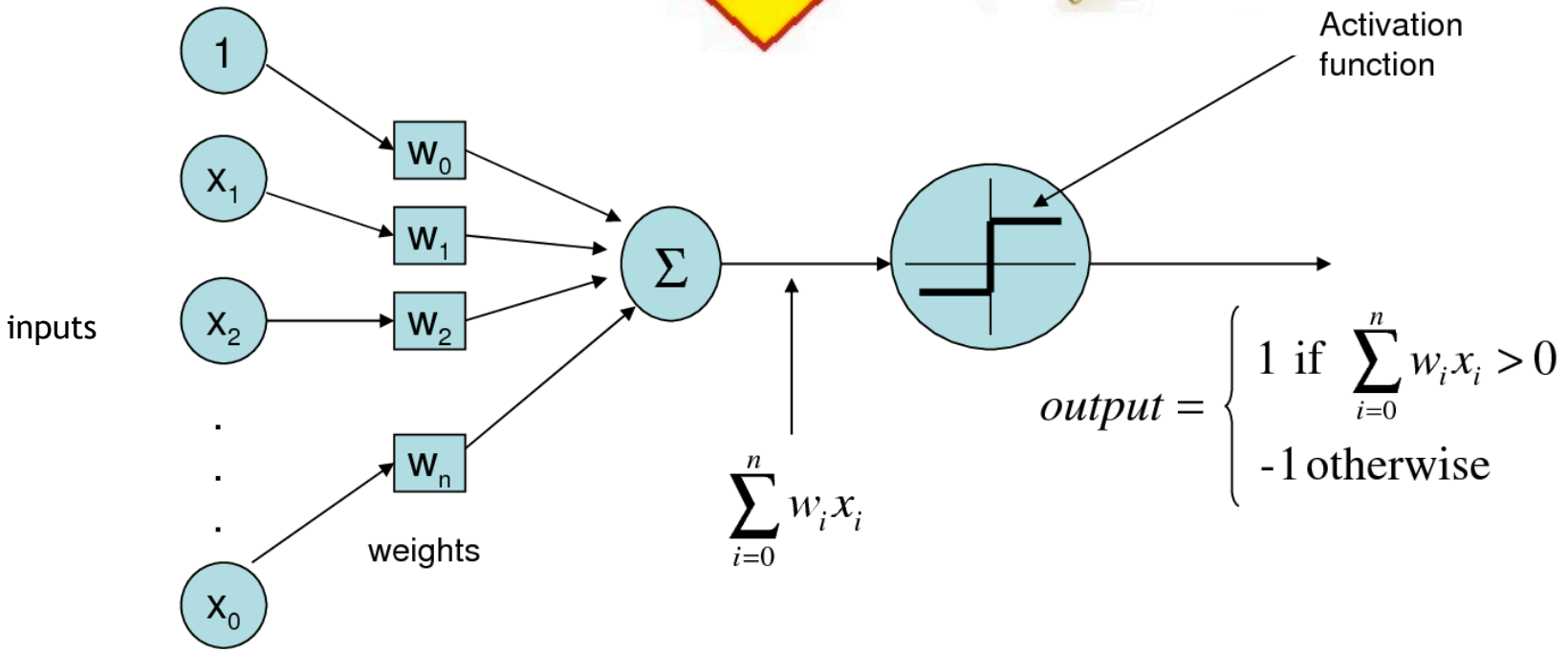
A Biological Neuron



https://www.youtube.com/watch?v=6qS83wD29PY

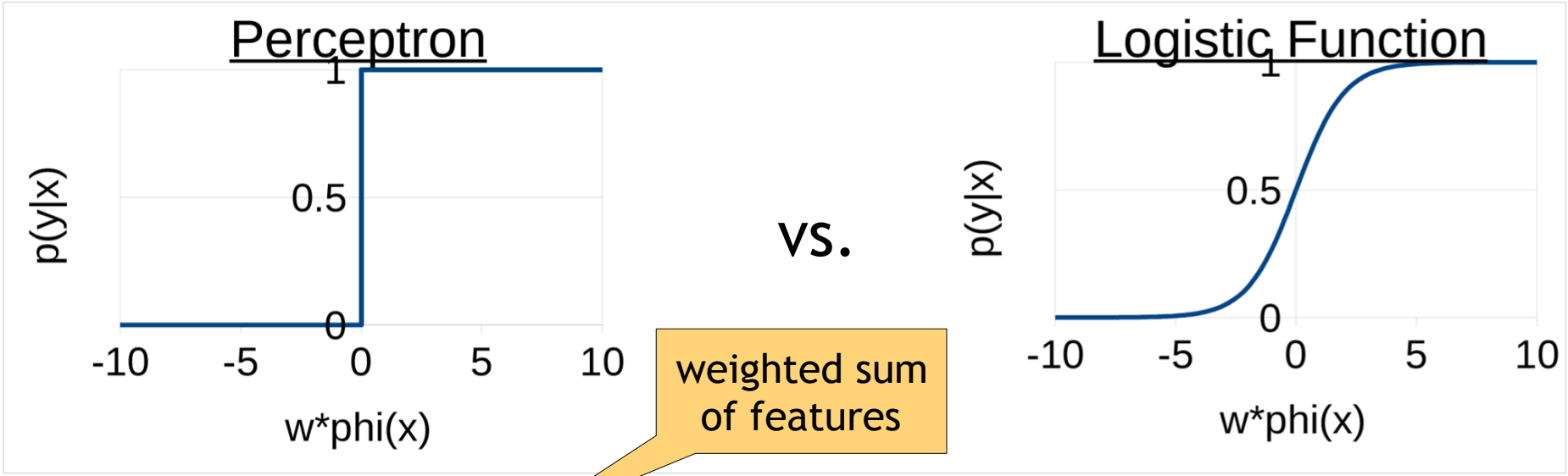
Perceptron (an artificial neuron)





Perceptron Algorithm

- Very similar to logistic regression
- Not exactly computing gradient (simpler)



weighted sum of features

$$P(y=1|x)=1 \text{ if } w\cdot\varphi(x)\geq 0$$
$$P(y=1|x)=0 \text{ if } w\cdot\varphi(x)<0$$

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$

Online Learning

- Rather than making a full pass through the data, compute gradient and update parameters after each training example.

Stochastic Gradient Descent

- Updates (or more specially, gradients) will be less accurate, but the overall effect is to move in the right direction

- Often works well and converges faster than batch learning

Online Learning

- update parameters for each training example

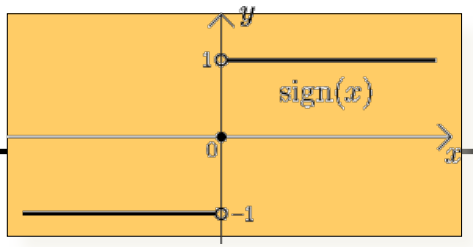
```
Initialize weight vector w = 0
Create features
Loop for K iterations
    Loop for all training examples x_i, y_i
            …
            update_weights(w, x_i, y_i)
```

Perceptron Algorithm

- Very similar to logistic regression
- Not exactly computing gradient

```
Initialize weight vector w = 0
Loop for K iterations
    Loop For all training examples x_i
        if sign(w * x_i) != y_i
            w += y_i * x_i
```

Error-driven!

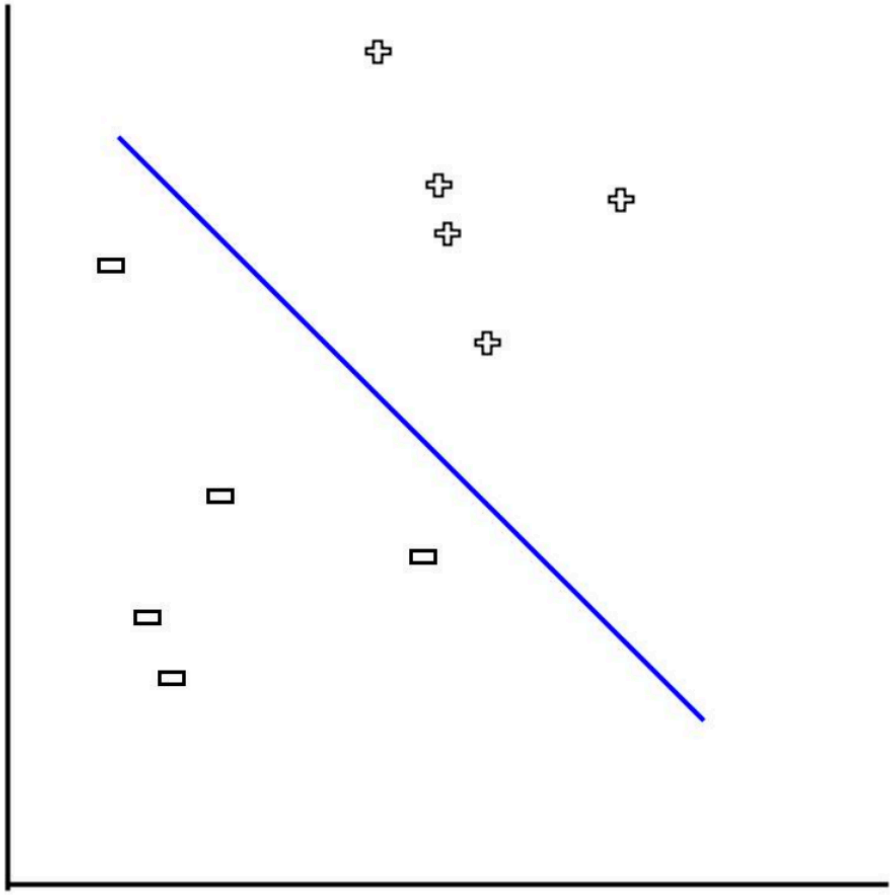


Error-driven Intuition

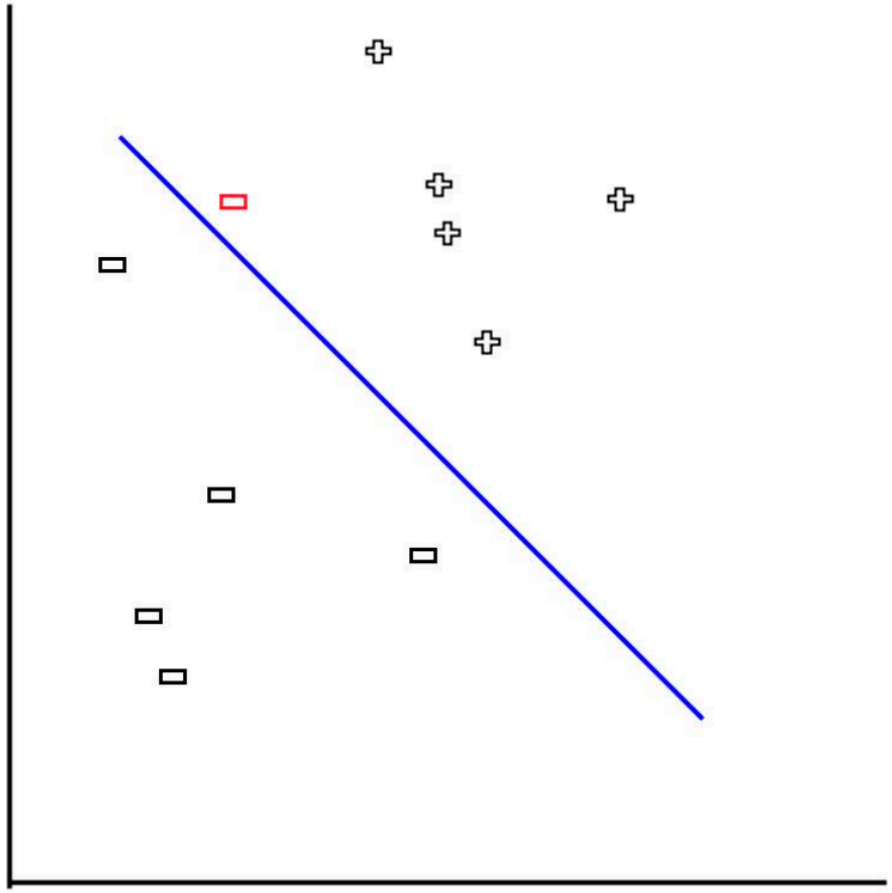
- For a given example, makes a prediction, then checks to see if this prediction is correct.

  - If the prediction is correct, do nothing.
  - If the prediction is wrong, change its parameters so that it would do better on this example next time around.

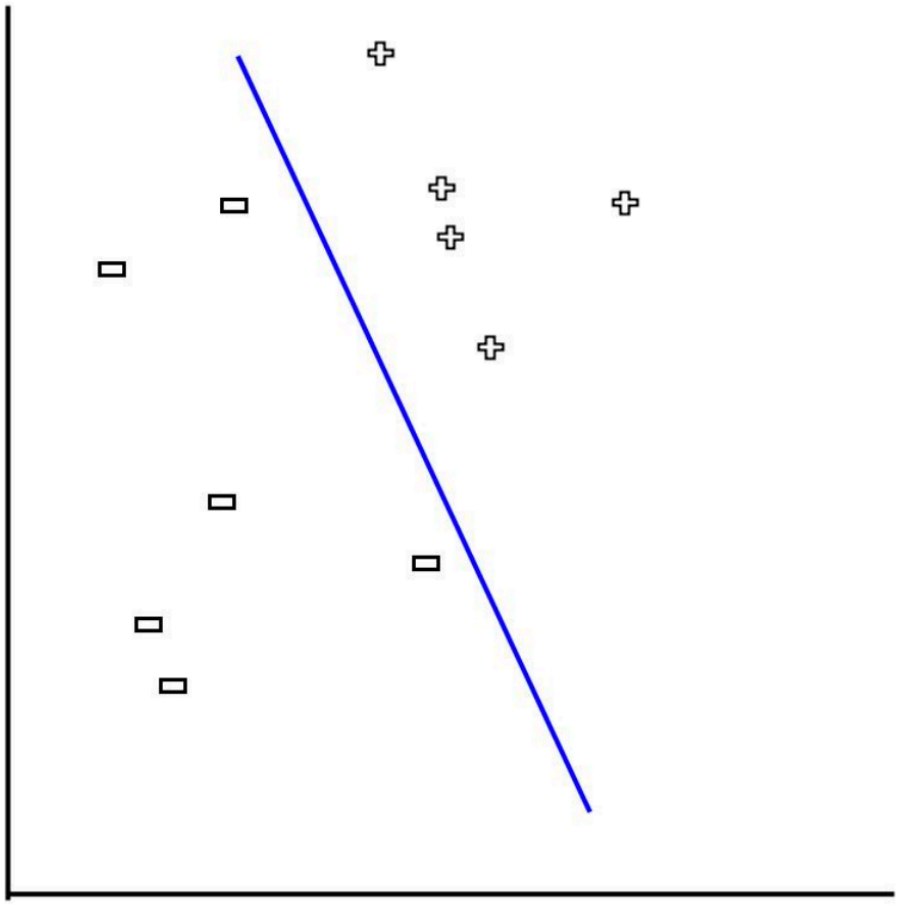
Error-driven Intuition



Error-driven Intuition



Error-driven Intuition



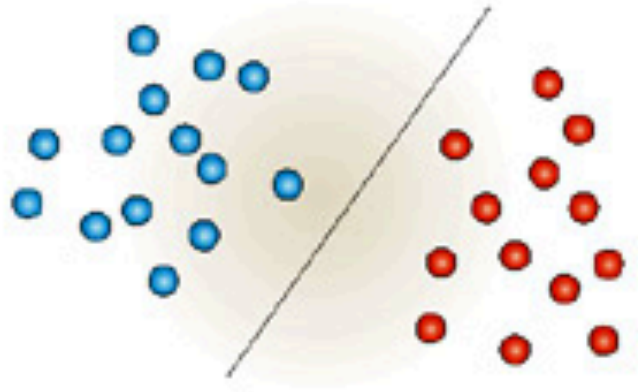
Exercise

Perceptron (vs. LR)

- Only hyperparameter is maximum number of iterations (LR also needs learning rate)

- Guaranteed to converge if the data is linearly separable (LR always converge)

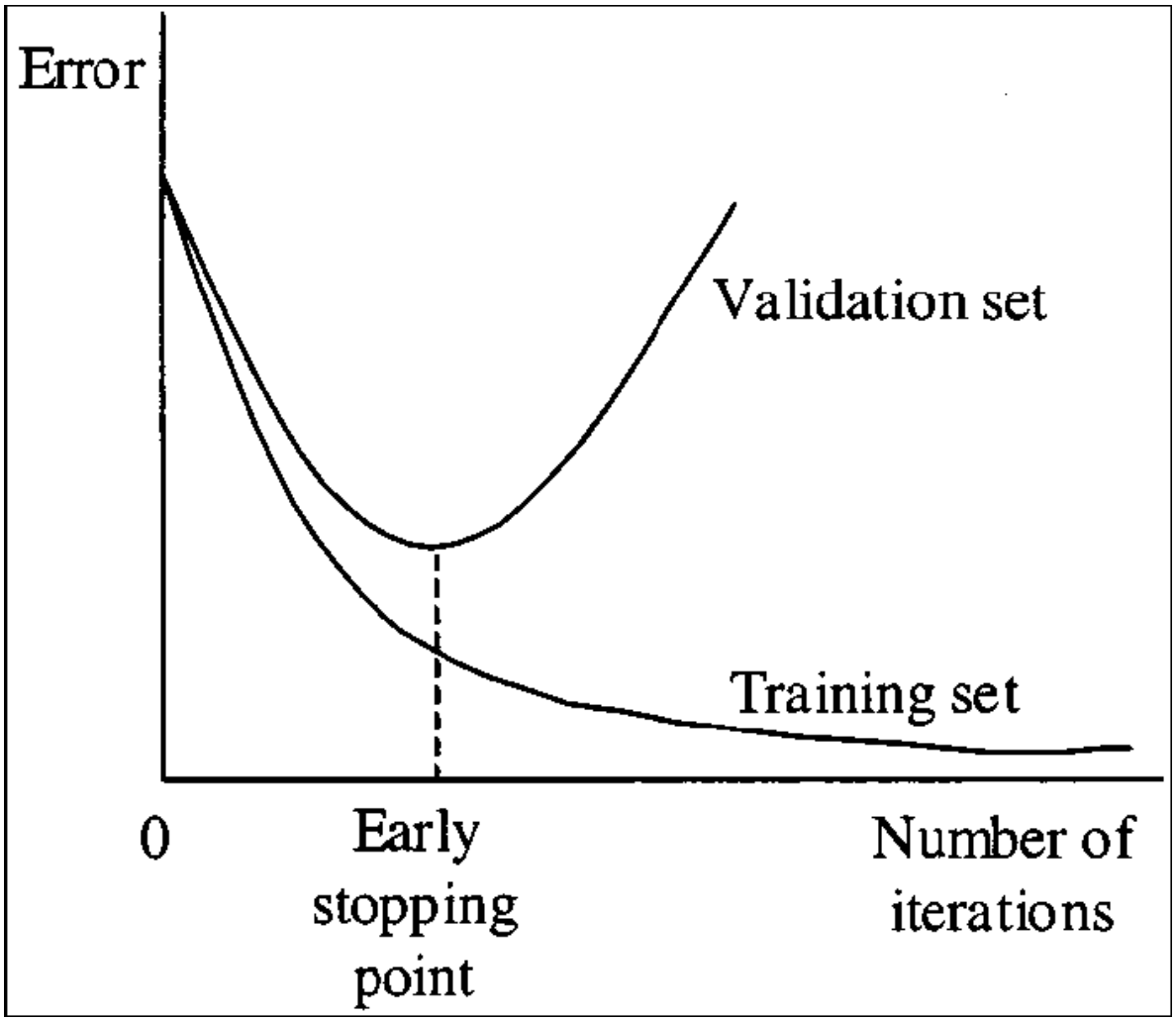
objective function is convex



What if non linearly separable?

- In real-world problems, this is nearly always the case.
- The perceptron will not be able to converge.
  - too many iterations may result in overfitting

Overfitting



Overfitting

- Model fit the training set, too perfectly
- If a feature is perfectly predictive of the outcome because it happens to only occur in one class (accidentally correlation), it will be assigned a very high weight.

Regularization

- Combating overfitting

- Intuition: don't let the weights get very large

$$w_{\text{MLE}} = \text{argmax}_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w)$$

$$\text{argmax}_w \log P(y_1, \ldots, y_d | x_1, \ldots, x_d; w) - \delta \sum_{i=1}^{V} w_i^2$$

penalize large weights
(L2: Euclidean Distance)

Regularization in the Perceptron

- Can’t directly include regularization in gradient
- # of iterations
- Parameter averaging

Perceptron: when to stop?

- Early stopping:
  - use a held-out (dev, validation) set
  - measure performance with current weights
  - stop when performance plateaus

Perceptron: another practical issue

Q: What happens if …

- first half of the training set are all positive instances,
- and the second half are all negative?

Perceptron: another practical issue

- Q: What happens if
  - first half of the training set are all positive instances,
  - and the second half are all negative?

- Perceptron puts too much emphasis to later cases

Perceptron: another practical issue

- Solution — permutation:
  - permute the training data
  - before each iteration (even better)

Perceptron: another practical issue

- Another Solution — averaging:
  - in training, remember how many rounds weight vectors survive
  - Computer the average weight vector
  - when testing, use the averaged weights

Exercise

Averaged Perceptron

Training:

Start with some initial $\mathbf{w}$, $\mathbf{u}$, b and d (such as 0)
$c = 1$
For $k = 1 \dots K$ iterations
    For each training instance, do prediction (calculate activation a)
        if $ya > 0$, do nothing
        if $ya \leq 0$, update the weights:
            $\mathbf{w} = \mathbf{w} + y\mathbf{x}$
            $b = b + y$
            $\mathbf{u} = \mathbf{u} + yc\mathbf{x}$
            $d = d + yc$
        $c = c + 1$
$\bar{\mathbf{w}} = \mathbf{w} - \mathbf{u}/c$
$\bar{b} = b - d/c$
Return the averaged parameter vector $\bar{\mathbf{w}}$, and bias $\bar{b}$

Multinominal Logistic Regression (Recap)

$$w_{\text{MLE}} = \text{argmax}_w \log P(y_1, \ldots, y_n | x_1, \ldots, x_n; w)$$

$$= \text{argmax}_w \sum_i \log P(y_i | x_i; w)$$

$$= \text{argmax}_w \sum_i \log \frac{e^{w \cdot f(x_i, y_i)}}{\sum_{y' \in Y} e^{w \cdot f(x_i, y')}}$$

Multiclass LR Gradient

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} \sum_{y \in Y} f_j(y, d_i) P(y|d_i)$$

empirical feature count

expected feature count

MAP-based learning (perceptron)

$$\frac{\partial \mathcal{L}}{\partial w_j} \approx \sum_{i=1}^{D} f_j(y_i, d_i) - \sum_{i=1}^{D} f_j(\arg\max_{y \in Y} P(y|d_i), d_i)$$

Maximum A Posteriori

approximate using maximization

MultiClass Perceptron Algorithm

- Every class has its own weight vector w_y
- Predict the class whose weight vector produces the highest activation
- If correct, do nothing
- If wrong, update the weights:
  - downweight score of wrong answer
  - increase score of right answer

MultiClass Perceptron Algorithm

```
Initialize weight vector w = 0
Loop for K iterations
    Loop For all training examples x_i
        y_pred = argmax_y w_y * x_i
        if y_pred != y_i
            w_y_gold += x_i
            w_y_pred -= x_i
```

increase score for right answer

decrease score for wrong answer

Homework (due Friday)

- **Market Research** on companies that use speech or language technologies. A short ~5 minute video.