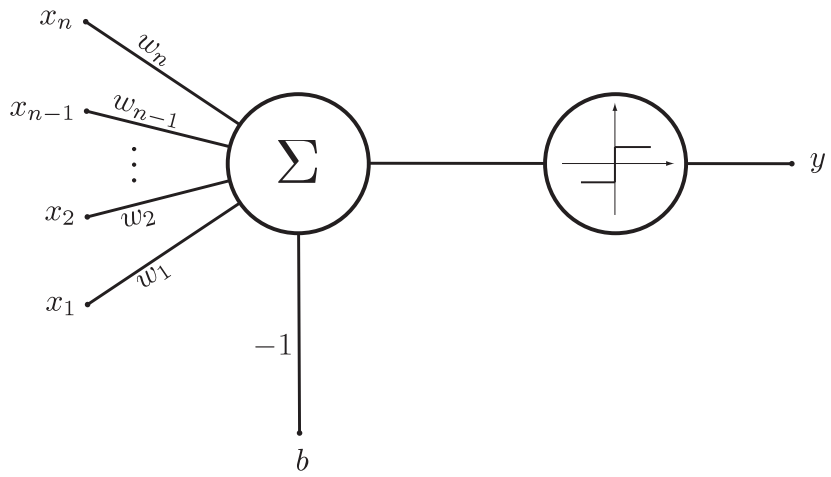
Artificial Neural Networks

Consider the perceptron



with

$$\hat{f}(\overline{x}) = y = \text{sign}\left(\left[\sum_{k=1}^{n} w_k x_k\right] + (-1)b\right) = \text{sign}\left(\overline{w} \bullet \overline{x} - b\right).$$

where $\overline{w} = (w_1, w_2, \ldots, w_n)$ and $\overline{x} = (x_1, x_2, \ldots, x_n)$. The free parameters of the perceptron are $\overline{w}$ and b and they need to be estimated using some training set D.

Perceptron Learning

let $D = \{(\overline{x}_1, y_1), (\overline{x}_2, y_2), \ldots, (\overline{x}_l, y_l)\} \subset \mathbb{R}^n \times \{+1, -1\}$
let $0 < \eta < 1$
$\overline{w} \leftarrow \overline{0}$
$b \leftarrow 0$
$r \leftarrow \max\{|\overline{x}| \mid (\overline{x}, y) \in D\}$
repeat
  for $i = 1$ **to** l
    if $\hat{f}(\overline{x}_i) \neq y_i$ **then**
      $\overline{w} \leftarrow \overline{w} + \Delta\overline{w}$
      $b \leftarrow b - \Delta b$
    end if
  end for
until $\hat{f}(\overline{x}_j) = y_j$ with $j = 1, \ldots, l$
return $(\overline{w}, b)$

where $\Delta\overline{w} = \eta y_i \overline{x}_i$ and $\Delta b = \eta y_i r^2$.

Perceptron Learning

Problem: Learning only works for linearly separable data; otherwise the algorithm will not converge.

Now assume that for the transfer function we use some function $t(\overline{x})$ instead of $\mathrm{sign}(\overline{x})$.

Also assume that we use the squared error at point $\overline{x}_i$,

$$se_i = (y_i - \hat{f}(\overline{x}_i))^2$$

instead of the standard $0-1$ loss function usually associated with classification. It still gives us the correct classification in the sense of a loss function:

Correct classifications: $(1-1)^2 = 0$ and $((-1)-(-1))^2 = 0$

Incorrect classifications: $(1-(-1))^2 = 4$ and $((-1)-1)^2 = 4$

Perceptron Learning

With this we can describe the error of a perceptron at a given point $\overline{x}_i$ with a set of weights $\overline{w}$ as follows,

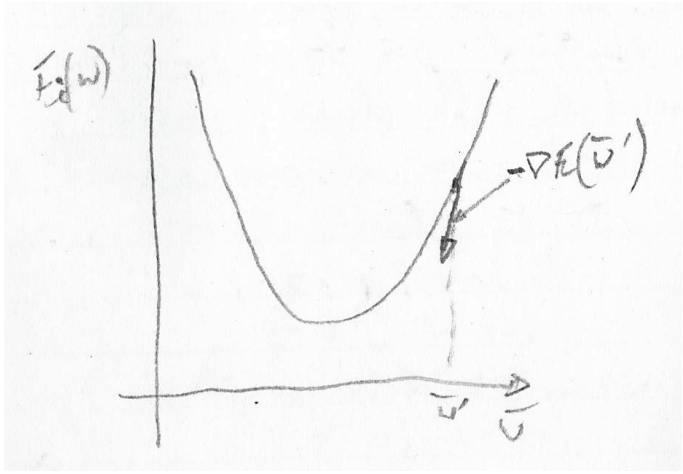
$$E_i(\overline{w}) = \frac{1}{2}(y_i - \hat{f}(\overline{x}_i))^2 = \frac{1}{2}(y_i - t(\overline{w} \bullet \overline{x}_i - b))^2$$

Recall that we swapped out the $sign$ function and replaced with the transfer function t.

Perceptron Learning

Now that we have a convenient error description we can look at changes of the error in terms of changes of the weights - *gradient*.

$$\nabla E_i = (\frac{\partial E_i}{\partial w_1}, \cdots, \frac{\partial E_i}{\partial w_n})$$



This allows us to rewrite our weight update rule $\overline{w} \leftarrow \overline{w} + \Delta\overline{w}$ with

$$\Delta\overline{w} = \eta \nabla E_i(\overline{w})$$

Perceptron Learning

Observation: We no longer try to learn a decision surface that separate the two classes perfectly, instead we are trying to learn a decision surface that *minimizes* the classification error.

Perceptron Learning

In order to compute the gradient we need to take the partial derivatives of the error: The components of ∇E_i are then,

$$\begin{aligned}
\frac{\partial E_i}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}(y_i - t(\overline{w} \bullet \overline{x}_i - b))^2 \\
&= \frac{1}{2}\frac{\partial}{\partial w_j}(y_i - t(\overline{w} \bullet \overline{x}_i - b))^2 \\
&= -(y_i - t(\overline{w} \bullet \overline{x}_i - b))\frac{\partial t}{\partial w_j}(\overline{w} \bullet \overline{x}_i - b)
\end{aligned}$$

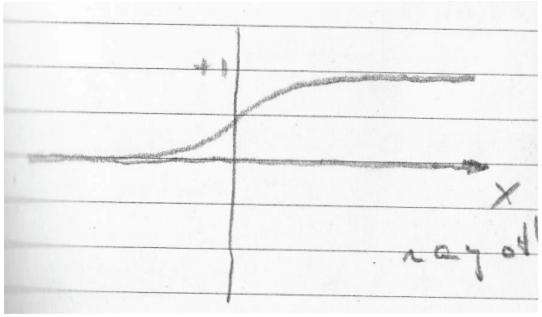
Observation: We can only train using the error gradient if the transfer function is differentiable!

Note: The sign function is *not* differentiable because it contains a discontinuity at 0.

Perceptron Learning

A convenient transfer function that looks like the sign function is the $sigmoid$ function,

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



Looks like the sign function but no discontinuities.

It has a nice derivative,

$$\frac{d\sigma}{d\overline{x}}(\overline{x}) = \sigma(\overline{x})(1 - \sigma(\overline{x}))$$

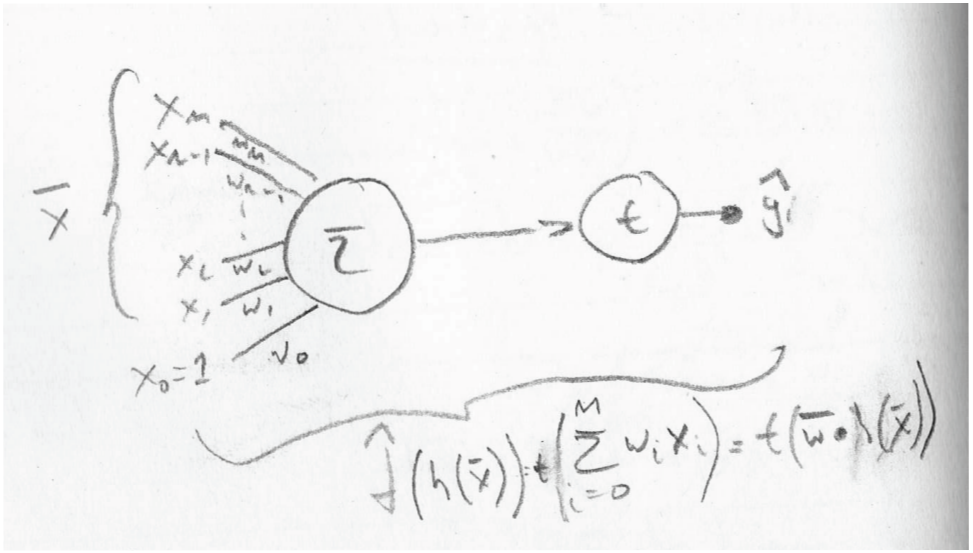
Perceptron Learning

What about b?

Here we apply another trick - we embed our training instances $\overline{x} \in R^n$ in a higher dimensional space, namely R^{n+1}, with the embedding function h as follows,

$$h(\overline{x}) = h(x_1, x_2, \ldots, x_n) = (1, x_1, x_2, \ldots, x_n)$$

With this our new perceptron looks like this



Perceptron Learning

Observation: The offset is trained as part of the weight training – no explicit offset needed.

Note: with $t = sigmoid$ and the embedding function h we talk about single layer ANNs.

Perceptron Learning

Our new training algorithm is as follows,

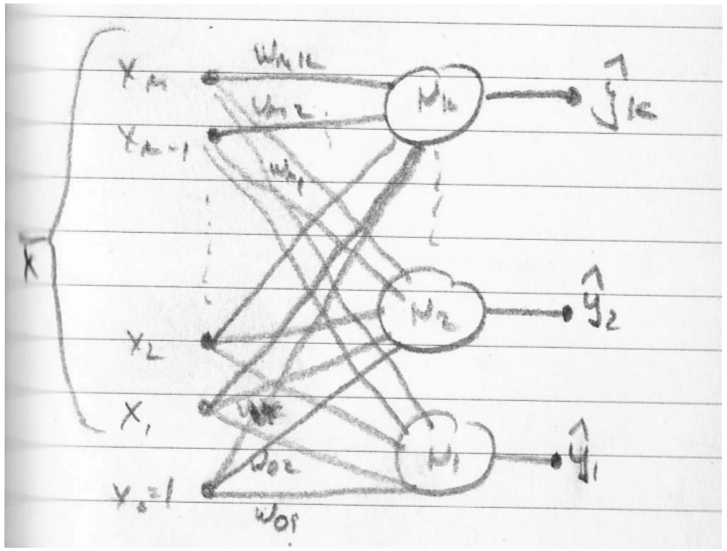
let $D = \{(\overline{x}_1, y_1), (\overline{x}_2, y_2), \ldots, (\overline{x}_l, y_l)\} \subset \mathbb{R}^n \times \{+1, -1\}$
let $0 < \eta < 1$
$\overline{w} \leftarrow \overline{0}$
repeat
  for $i = 1$ **to** l
    $\overline{w} \leftarrow \overline{w} + \eta \nabla E_i(\overline{w})$
  end for
until $\nabla E(\overline{w}) \approx 0$
return $\overline{w}$

Observation: We don't require the error to be zero, just the gradient.

$$\frac{\partial E_i}{\partial w_j} = -(y_i - t(\overline{w} \bullet h(\overline{x}_i))) \frac{\partial t}{\partial w_j}(\overline{w} \bullet h(\overline{x}_i))$$

Multi-Class ANNs

We can easily extend our single layer ANN to do multi-class classification. Consider a classification problem with k classes. We can construct an ANN for this as follows:



We now have $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_k)$ with

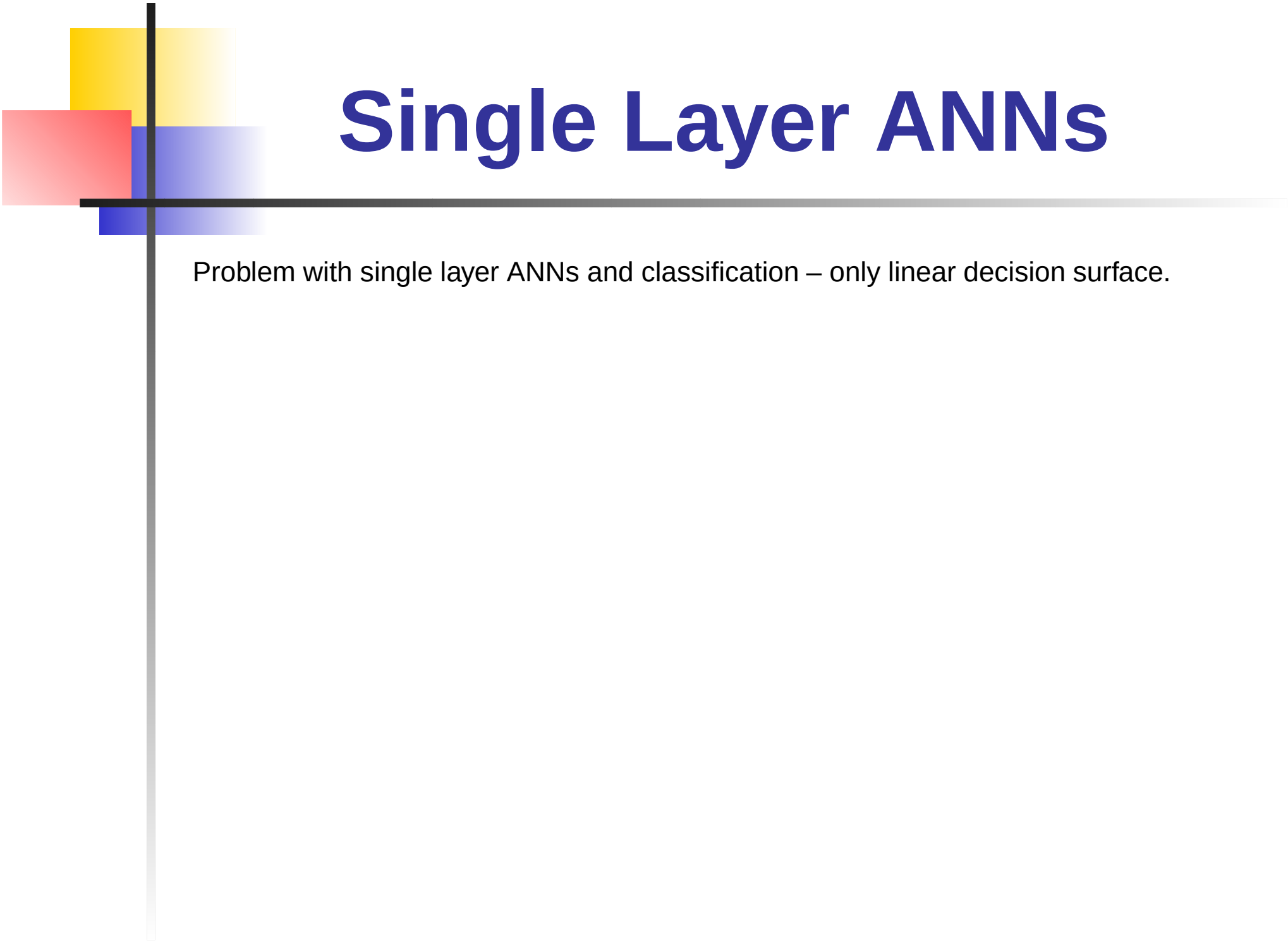
$$\overline{x} \in \text{ class } i \text{ iff } \hat{y} = (0, \ldots, \hat{y}_i, \ldots, 0) \text{ and } y_i = 1.$$

Multi-Class ANNs

Our training data needs to be adjusted to the vector notation of class membership,

$$D = \{(\overline{x}_1, \overline{y}_1), \ldots, (\overline{x}_l, \overline{y}_l)\} \in R^n \times \{0,1\}^k$$

This means that the multi-class ANN is trained component wise and all our previous result generalize very nicely.



Single Layer ANNs

Problem with single layer ANNs and classification – only linear decision surface.