Regression as Machine Learning

Given

- A data universe X.
- A sample set S where $S \subset X$.
- Some target function $f : X \to \mathbb{R}$.
- A training set D, where $D = \{(x, y) \mid x \in S \text{ and } y = f(x)\}$.

Compute a model $\hat{f} : X \to \mathbb{R}$ using D such that,

$$\hat{f}(x) \cong f(x),$$

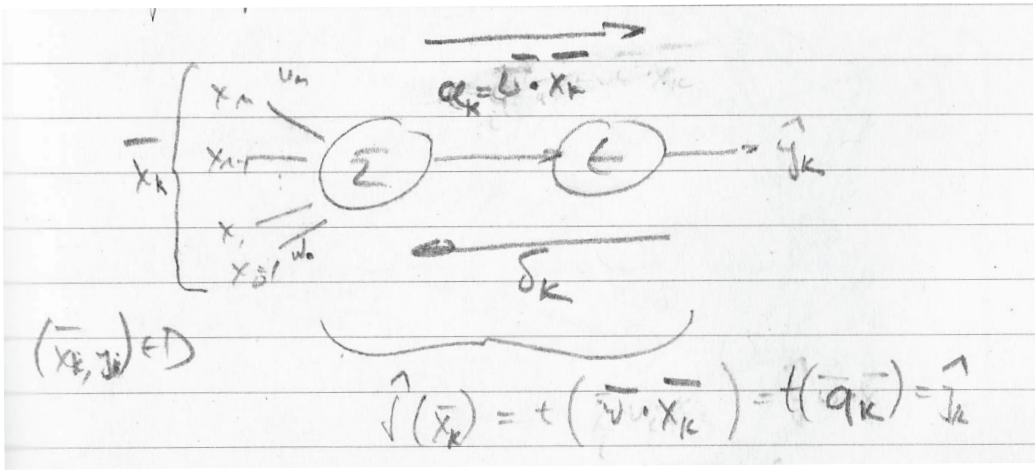
for all $x \in X$.

Observation: Same as machine learning in classification except for the co-domains of the target function and the model.

Question: How do we compute the model?

Regression ANNs

The perceptron revisited



Regression ANNs

Recall

$$\begin{aligned} E_k(\overline{w}) &= \frac{1}{2}(y_k - \hat{y}_k)^2 \\ &= \frac{1}{2}(y_k - t(\overline{w} \bullet \overline{x}_k)^2 \\ &= \frac{1}{2}(y_k - t(a_k))^2 \end{aligned}$$

Learning defined in terms of numerical error instead of classification error - regression by default!

We turn the regression problem into a classification problem by applying *thresholding*.

Regression ANNs

We can now look at the gradient,

$$
\begin{aligned}
\nabla E_k(\overline{w}) &= \frac{d}{d\overline{w}} E_k(\overline{w}) \\
&= \frac{1}{2}\frac{d}{d\overline{w}}(y_k - t(a_k))^2 \\
&= -(y_k - t(a_k))\frac{dt}{d\overline{w}}(a_k) \\
&= -(y_k - \hat{y}_k)\frac{dt}{d\overline{w}}(a_k) \\
&= -(y_k - \hat{y}_k)\frac{dt}{da_k}(a_k)\frac{da_k}{d\overline{w}} \text{ (chain rule)} \\
&= -(y_k - \hat{y}_k)t'(a_k)\frac{d}{d\overline{w}}(\overline{w} \bullet \overline{x}_k) \\
&= -(y_k - \hat{y}_k)t'(a_k)\overline{x}_k \\
&= \delta_k \overline{x}_k
\end{aligned}
$$

where $\delta_k = -(y_k - \hat{y}_k)t'(a_k)$ is called the error.

Regression ANNs

Now recall our update rule,

$$\overline{w} \leftarrow \overline{w} + \Delta\overline{w}$$

From before we have

$$\overline{w} \leftarrow \overline{w} + \eta\nabla E_k(\overline{w})$$

From our discussion above it follows that

$$\overline{w} \leftarrow \overline{w} + \eta\delta_k\overline{x}_k$$

Observation: The weights are updated using a scaled version of the input vector. It is also easy to see that the weights are scaled proportional to the error.

This is called *back propagation.*

If we can take the derivative of the transfer function then we can easily extend this to multi-layer neural networks.