Neural Networks

\[ f(x) = \sigma \left( \mathbf{W} \cdot x + \mathbf{b} \right) \]

Recursive:

- "layers" how many axes until we hit \( x^k \)
- then \( \sigma \) and continue until \( x^n \)

\[ x^{(k)} = \sigma \left( \mathbf{W}^{(k)} \cdot x^{(k-1)} + \mathbf{b}^{(k)} \right) \]

- how to pick \( \mathbf{W} \), need more

Features "learned" by the NN: look at \( \mathbf{W} \), which looks like filters similar to the brain

"biologically inspired"

On linearity:

- it's just some
- does add \( \mathbf{b} \)

"synaptic weights"

Training: a lot of weights

- \( \mathbf{W} \) layers, \( \mathbf{b} \) per axis
- \( n \cdot \mathbf{w} \) million parameters: high dimensional
- not sure it had to optimize

"could be in a local optimum"

Overfitting:

- "getting good at peculiarities" in the training set

- training: too high polynomial (minimize error)
- testing

Terminology:

- \( \mathbf{W} \): "weights" - free parameters (matrix)
- \( \mathbf{b} \): "biases" - free parameter
- \( \sigma \): "unit", or "neurons", or "hidden units"

train/learn/optimize: set \( \mathbf{W} \) and \( \mathbf{b} \) (so model data well)

Supervised learning: learn \( y = f(x) \) from labeled data

Regression: "learn \( \mathbf{W} \) in \( \mathbf{W} \) for example"

Classification: "what word is it?"

size of output layer: size of classes

size of output layer: size of classes
Back propagation

An algorithm to compute the gradient of the loss with respect to the weights, efficiently making extensive use of chain rule.

We want to minimize the loss:

\[ z^{(n)} \rightarrow a^{(n)} \rightarrow z^{(n')} \rightarrow a^{(n')} \rightarrow z^{(n'')} \rightarrow L \]  

a dependency graph.

\[ z_j^{(l)} = \sigma(a_j^{(l)}) \]
\[ a_j^{(l)} = \mathbf{w}_j^{(l-1)^T} a_j^{(l-1)} \]

\[ L = \frac{1}{2} \sum_j (z_j - \hat{z}_j)^2 \]  

La loss.

Loss w.r.t. \( W \):

\[ \frac{\delta L}{\delta W_{ij}^{(l)}} = \frac{\delta L}{\delta z_j^{(l)}} \cdot \frac{\delta z_j^{(l)}}{\delta W_{ij}^{(l)}} \]

Each element of \( W \) only depends on a particular column of \( W \).

\[ \frac{\delta L}{\delta W_{ij}^{(l)}} = z_j^{(l)} \cdot a_i^{(l-1)} \]

\[ z_j^{(l)} = \sigma(a_j^{(l-1)}) \]

\[ a_i^{(l-1)} = \mathbf{w}_i^{(l)^T} \cdot a_j^{(l-1)} \]

\[ \frac{\delta L}{\delta W_{ij}^{(l)}} = \left( z_j^{(l)} - \hat{z}_j \right) \sigma'(a_j^{(l-1)}) z_i^{(l-1)} \]

Loss w.r.t. \( W \):

\[ \frac{\delta L}{\delta W_{ij}^{(l)}} = z_k^{(l)} \sigma'(a_k^{(l)}) \frac{\delta L}{\delta z_k^{(l)}} \]

\[ a_k^{(l)} = \mathbf{w}_k^{(l-1)^T} \cdot a_j^{(l-1)} \]

\[ \frac{\delta L}{\delta W_{ij}^{(l)}} = z_k^{(l)} \sigma'(a_k^{(l)}) \mathbf{w}_k^{(l-1)^T} \left( z_j^{(l)} - \hat{z}_j \right) \sigma'(a_j^{(l-1)}) \]

Generally trained with gradient descent, and some "learning rate".

\[ \mathbf{w} \rightarrow \mathbf{w} + \eta \cdot \nabla f(w) \]

\[ \text{normalize to probability} \]

\[ \text{softmax} \]