Artificial "Neuron"

\[
1 = x_0
\]

\[
\begin{align*}
\sum_{j=0}^{N} w_j x_j &= a \\
\hat{y} &= f(a) = f(A^T x)
\end{align*}
\]

Activation/Response Function

Inputs \* Weights \* Connections

Activation

\[
a = \sum_{j=0}^{N} w_j x_j
\]

\[
= A^T x
\]

Many different activation functions

- **Linear**: \( f(a) = a \)

\[
\hat{y} = A^T x \quad \text{[Linear Regression]}
\]

- **Sigmoid**: \( f(a) = \frac{1}{1 + e^{-a}} = \sigma(a) \)

\[
\hat{y} = \frac{1}{1 + e^{-A^T x}} \quad \text{[Logistic Regression]}
\]

- **Tanh**: \( f(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \)

For hidden units in NN, we always prefer tanh over \( \sigma(a) \) \( \text{why?} \)
→ **ReLU** (Rectified linear Unit)

\[ f(a) = \max \{ 0, a \} \]

In hidden layer of deep NN, this is always preferred over $\sigma(a)$ or $\tanh(a)$, why??

→ **Maxout**

\[ f(a) = \max \{ \overrightarrow{W}_1 x, \overrightarrow{W}_2 x \} \]

Each neuron has (say) 2 weights $\overrightarrow{W}_1, \overrightarrow{W}_2$

Take max activation.

ReLU is a special case of this. How?
3. Loss Functions

- functions of both parameters $\theta$ and training data $\mathcal{D}$

1. Log-loss / Cross-Entropy / Maximum-Likelihood / KL-Divergence

$L(\theta; \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} L_i (\mathcal{D})$ - Decomposable Loss

where $L_i (\mathcal{D}) = - \log p(y_i | x_i, \theta)$

How much prob does your model assign to $y_i$ label?

$= \text{negative log-likelihood for this sample}$

Why is this called Cross-Entropy? And where is the KL-divergence coming in?

Consider Multiclass classification w/ 1-HOT encoding

\[ \hat{y}_i \]

\[
\begin{array}{c|c|c}
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\hat{p}(y) & p(y = 1 | x, \theta) \\
\hat{p}(y) & p(y = 0 | x, \theta) \\
\end{array}
\]

[delta distribution] [Model distribution]
\[ KL(p_{\text{st}} \parallel \hat{p}) = \sum_{y=1}^{K} p_{\text{st}}(y) \log \hat{p}(y) \]

\[ = -\log p(y = y^{st} | \tilde{x}_i, W) \]

2) Hinge Loss [for binary classification]

\[ L_i(W) = \max \{ 0, 1 - y_i \tilde{y}_i \} \quad \text{where } y_i \in \{+1, -1\} \]
4. Detour: Matrix/Vector differentiation

\[ S \begin{bmatrix} \frac{\partial y}{\partial x} & \frac{\partial y}{\partial z} & \frac{\partial y}{\partial x} \\ \frac{\partial y}{\partial x} & \frac{\partial y}{\partial z} & \frac{\partial y}{\partial x} \end{bmatrix} \]

\[ x, y \in \mathbb{R}^d \]
\[ z \in \mathbb{R}^2 \]
\[ g \in \mathbb{R}^k \]

 Convention: \[ \frac{\partial y}{\partial x} \lnum \begin{bmatrix} \frac{\partial y}{\partial x} \\ \frac{\partial y}{\partial x} \\ \frac{\partial y}{\partial x} \end{bmatrix} \]
\[ \text{numerator} = \text{dim 1} \]
\[ \text{col-vector} \]
\[ \text{denominator} = \text{dim 2} \]
\[ \text{row-vector} \]

Gradient \[ \frac{\partial y}{\partial x} \]
\[ \text{Jacobian Matrix} \]
\[ \frac{\partial y}{\partial x} \rightarrow \left[ \begin{array}{cc} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_k}{\partial x_1} & \cdots & \frac{\partial y_k}{\partial x_d} \end{array} \right] \]

Easy to prove: \[ \frac{\partial (\mathbf{w}^T \mathbf{x})}{\partial \mathbf{w}} = \left[ \frac{\partial (\mathbf{w}^T \mathbf{x})}{\partial w_1} \cdots \frac{\partial (\mathbf{w}^T \mathbf{x})}{\partial w_k} \right] = \mathbf{x} \]

\[ \frac{\partial (\mathbf{w}^T \mathbf{A} \mathbf{w})}{\partial \mathbf{w}} \]

\[ \mathbf{w} = \mathbf{A} \mathbf{x} \]
\[ \frac{\partial \mathbf{w}}{\partial \mathbf{x}} = \mathbf{A} \]

\[ \frac{\partial (\mathbf{w}^T \mathbf{A} \mathbf{w})}{\partial \mathbf{w}} = 2 \mathbf{w}^T \mathbf{A} \]
Chain Rule

Function composition: \( L(x) = (f \circ g)(x) = f(g(x)) \)

Chain Rule:

\[ \frac{D}{dx} (f \circ g) = Dg \circ f \circ Dg \]

[Most General Notation]

[More concise notation for scalars]

\[ L'(x) = f'(g(x)) \cdot g'(x) \]

[With intermediate variables]

\[ y = g(x) \]
\[ z = f(y) \]

\[ \frac{∂z}{∂x} = \frac{∂z}{∂y} \cdot \frac{∂y}{∂x} \]

Example:

\[ L_i(w) = -\log \left( \frac{1}{1+e^{-x^T w}} \right) \]  
\[ \text{For } y_i = +1 \]

\[ = \begin{pmatrix} -\log (\cdot) & \frac{1}{1+e^{-x^T w}} & x^T(w) \end{pmatrix} \]

\[ \frac{∂L}{∂w} = \begin{bmatrix} 1 + e^{-x^T w} \end{bmatrix} \cdot \begin{bmatrix} -1 & -e^{-x^T w} \end{bmatrix} \cdot x^T \]

\[ = (1-p) x^T \]
Multivariate Chain Rule

\[ g : \mathbb{R}^k \to \mathbb{R}^m \]
\[ f : \mathbb{R}^m \to \mathbb{R}^k \]

\[ h(x) = (f \circ g)(x) \]
\[ y = g(x) \quad z = f(y) \]

Chain Rule:
\[ D_z(h) = D_y(f \circ g) \cdot D_g(h) \]

But what does this mean??

Visualize:

```
\[ \frac{\partial z_i}{\partial x_j} = \sum_{k} \frac{\partial z_i}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_j} \]
```

All intermediate variables how those intermediate variables affect outcome!
Formally,

Jacobian relationship

\[ J_{f \circ g} = (J_{f \circ g}) \cdot J_g \]

\[ \begin{bmatrix} \frac{\partial f_i}{\partial x_j} \\ \frac{\partial f_i}{\partial y_j} \end{bmatrix} = \begin{bmatrix} \frac{\partial z_k}{\partial x_i} \\ \frac{\partial z_k}{\partial y_i} \end{bmatrix} \]

→ What if my \( z, y, z \) are tensors?
    → Writing up into vectors & proceed
    → Matlab notation \( x \cdot \text{vec} = x(i) \);
    → Trust me, this is the cleaner way.

→ In Neural Net \( z \in \mathbb{R}^k \) (Log) \( L(z) \)

In layer \( i \)

\[ h^{(i)} = g(h^{(i-1)}, \theta) \]

\[ L(\theta) = f(h^{(i)}) \]

\[ \frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial h^{(i)}} \cdot \frac{\partial h^{(i)}}{\partial \theta} \]

\[ L \in \mathbb{R}^{1 \times d} \]