Feedforward neural nets
The architecture

\[ y \]
\[ h^{(\ell)} \]
\[ \vdots \]
\[ h^{(2)} \]
\[ h^{(1)} \]
\[ x \]

\[ p(y=1|x) \quad p(y=2|x) \quad p(y=3|x) \]
The value at a hidden unit

\[ h \]

\[ z_1 \quad z_2 \quad \cdots \quad z_m \]

How is \( h \) computed from \( z_1, \ldots, z_m \)?

\[ h = \sigma \left( w_1 z_1 + w_2 z_2 + \cdots + w_m z_m + b \right) \]

\( \sigma (\cdot) \) is a nonlinear activation function, e.g. "rectified linear"

\[ \sigma (u) = \begin{cases} u & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases} \]
The value at a hidden unit

How is \( h \) computed from \( z_1, \ldots, z_m \)?

- \( h = \sigma(w_1 z_1 + w_2 z_2 + \cdots + w_m z_m + b) \)

- \( \sigma(\cdot) \) is a nonlinear activation function, e.g. “rectified linear”

\[
\sigma(u) = \begin{cases} 
    u & \text{if } u \geq 0 \\
    0 & \text{otherwise}
\end{cases}
\]

\[
\max(0, u)
\]
Why do we need nonlinear activation functions?
The output layer

Classification with $k$ labels: want $k$ probabilities summing to 1.

$$
\begin{align*}
    y_1 & \quad y_2 & \cdots & \quad y_k \\
    z_1 & \quad z_2 & \quad z_3 & \cdots & \quad z_m \\
\end{align*}
$$

• $y_1, \ldots, y_k$ are linear functions of the parent nodes $z_i$.  
• Get probabilities using softmax:

$$
    \Pr(y = k | x) = \frac{\exp(W_k^T z)}{\sum_{k'} \exp(W_{k'}^T z)}
$$
The output layer

Classification with $k$ labels: want $k$ probabilities summing to 1.

- $y_1, \ldots, y_k$ are linear functions of the parent nodes $z_i$.
- Get probabilities using \textbf{softmax}:

$$\Pr(\text{label } j) = \frac{e^{y_j}}{e^{y_1} + \cdots + e^{y_k}}.$$
The complexity
The effect of depth

• **Universal approximator**
  Any function can be arbitrarily well approximated by a neural net with one hidden layer.
The effect of depth

- **Universal approximator**
  Any function can be arbitrarily well approximated by a neural net with one hidden layer.

- **Concerns about size**
  To fit certain classes of functions:
  - Either: one hidden layer of enormous size
  - Or: multiple hidden layers of moderate size
Learning a net: the loss function

Classification problem with $k$ labels.

- Parameters of entire net: $W$
- For any input $x$, net computes probabilities of labels:

$$\Pr_W(\text{label }= j|x)$$
Learning a net: the loss function

Classification problem with $k$ labels.

- Parameters of entire net: $W$
- For any input $x$, net computes probabilities of labels:
  \[ Pr_W(\text{label} = j | x) \]
- Given data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$, loss function:
  \[ L(W) = -\sum_{i=1}^{n} \ln Pr_W(y^{(i)} | x^{(i)}) \]
  (sometimes called cross-entropy).
Nature of the loss function

$L(w)$

$w$

$L(w)$

$w$
Initialize $W$ and then repeatedly update.

1. **Gradient descent**  
   Each update involves the entire training set.

2. **Stochastic gradient descent**  
   Each update involves a single data point.

3. **Mini-batch stochastic gradient descent**  
   Each update involves a modest, fixed number of data points.
Derivative of the loss function

Update for a specific parameter: derivative of loss function wrt that parameter.
Suppose \( h(x) = g(f(x)) \), where \( x \in \mathbb{R} \) and \( f, g : \mathbb{R} \to \mathbb{R} \).

Then: \( h'(x) = g'(f(x)) f'(x) \)
Chain rule

1. Suppose \( h(x) = g(f(x)) \), where \( x \in \mathbb{R} \) and \( f, g : \mathbb{R} \to \mathbb{R} \).

   Then: \( h'(x) = g'(f(x))f'(x) \)

2. Suppose \( z \) is a function of \( y \), which is a function of \( x \).

   Then:
   \[
   \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}
   \]
A single chain of nodes

A neural net with one node per hidden layer:

\[ x = h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow \cdots \rightarrow h_\ell \]

For a specific input \( x \),

- \( h_i = \sigma(w_i h_{i-1} + b_i) \)
- The loss \( L \) can be gleaned from \( h_\ell \)
A single chain of nodes

A neural net with one node per hidden layer:

\[ x = h_0 \quad h_1 \quad h_2 \quad h_3 \quad \cdots \quad h_\ell \]

For a specific input \( x \),

- \( h_i = \sigma(w_i h_{i-1} + b_i) \)
- The loss \( L \) can be gleaned from \( h_\ell \)

To compute \( \frac{dL}{dw_i} \) we just need \( \frac{dL}{dh_i} \):

\[
\frac{dL}{dw_i} = \frac{dL}{dh_i} \frac{dh_i}{dw_i} = \frac{dL}{dh_i} \sigma'(w_i h_{i-1} + b_i) h_{i-1}
\]
Backpropagation

- On a single forward pass, compute all the $h_i$.
- On a single backward pass, compute $dL/dh_\ell, \ldots, dL/dh_1$
Backpropagation

- On a single forward pass, compute all the $h_i$.
- On a single backward pass, compute $dL/dh_\ell, \ldots, dL/dh_1$

\[
x = h_0 \quad h_1 \quad h_2 \quad h_3 \quad \cdots \quad h_\ell
\]

From $h_{i+1} = \sigma(w_{i+1}h_i + b_{i+1})$, we have

\[
\frac{dL}{dh_i} = \frac{dL}{dh_{i+1}} \frac{dh_{i+1}}{dh_i} = \frac{dL}{dh_{i+1}} \sigma'(w_{i+1}h_i + b_{i+1}) w_{i+1}
\]
Improving generalization

1 Early stopping

- Validation set to better track error rate
- Revert to earlier model when recent training hasn’t improved error
Improving generalization

1 Early stopping
   - Validation set to better track error rate
   - Revert to earlier model when recent training hasn’t improved error

2 Dropout
   During training, delete each hidden unit with probability $1/2$, independently.

\[
\begin{align*}
&y \\
h^{(\ell)} \\
&\vdots \\
h^{(2)} \\
h^{(1)} \\
x
\end{align*}
\]