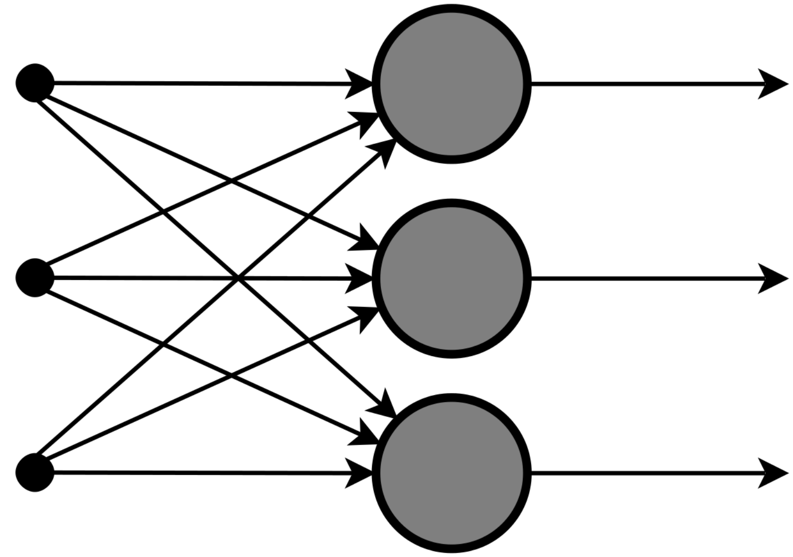
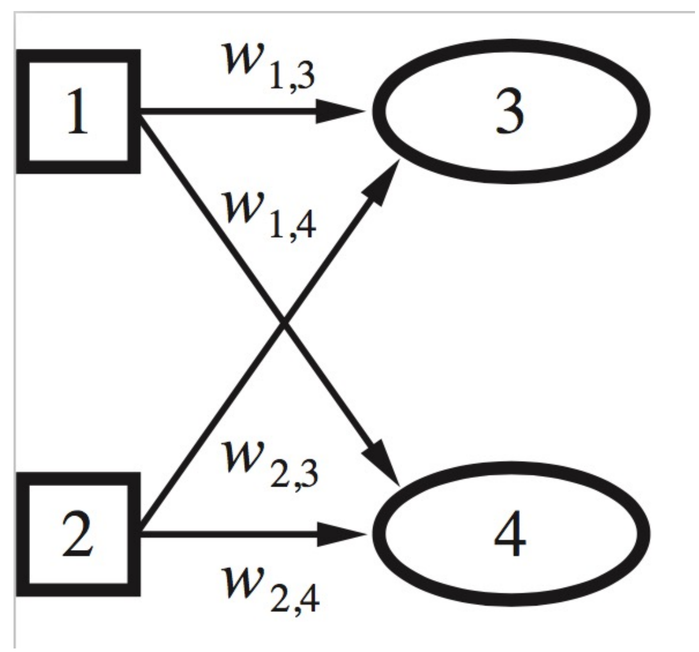
Single layer feed forward networks

- One input layer (which is just the raw inputs).
- One output layer (of perceptron units).
- Let's design a network to add two bits together.
- Needs two inputs (x_1, x_2), and two outputs (y_3, y_4).





Single layer feed forward networks

- There is an algorithm to change the weights of a single-layer network to make the network learn any function...
- Initialize starting weights randomly
- Do until you want to stop (*typically when accuracy is good enough or weights stop changing*):
  - for each training example (x, y):
    - use NN to get prediction of h(x)
    - if h(x) differs from y, update all weights:
    - w[i] = w[i] + (y – h(x)) * x[i]
  - compute accuracy over entire training data = (# predicted correctly)/(# of training examples)

Single layer feed forward networks

- There is an algorithm to change the weights of a single-layer network to make the network learn any function...
- as long as it is linearly-separable!

Multi-layer feed forward networks

- Learning is done through the backpropagation algorithm (*backprop*).
- Derived through calculus (we will skip).

```
repeat
    for each weight w_{i,j} in network do
        w_{i,j} ← a small random number
    for each example (x, y) in examples do
        /* Propagate the inputs forward to compute the outputs */
        for each node i in the input layer do
            a_i ← x_i
        for ℓ = 2 to L do
            for each node j in layer ℓ do
                in_j ← Σ_i w_{i,j} a_i
                a_j ← g(in_j)
        /* Propagate deltas backward from output layer to input layer */
        for each node j in the output layer do
            Δ[j] ← g′(in_j) × (y_j − a_j)
        for ℓ = L − 1 to 1 do
            for each node i in layer ℓ do
                Δ[i] ← g′(in_i) Σ_j w_{i,j} Δ[j]
        /* Update every weight in network using deltas */
        for each weight w_{i,j} in network do
            w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
until some stopping criterion is satisfied
return network
```

Backprop highlights

repeat
  for each weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow$ a small random number
  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    / * *Propagate the inputs forward to compute the outputs* * /
    for each node i in the input layer **do**
      $a_i \leftarrow x_i$
    for $\ell = 2$ **to** L **do**
      for each node j in layer ℓ **do**
        $in_j \leftarrow \sum_i w_{i,j}\, a_i$
        $a_j \leftarrow g(in_j)$

Backprop highlights

/ $\star$ *Propagate deltas backward from output layer to input layer* $\star$ /

for each node j in the output layer **do**

$\quad \Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

for $\ell = L - 1$ **to** 1 **do**

$\quad$ **for each** node i in layer ℓ **do**

$\quad\quad \Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$

/ $\star$ *Update every weight in network using deltas* $\star$ /

for each weight $w_{i,j}$ in *network* **do**

$\quad w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$

Compare

- w[i] = w[i] + (y – h(x)) * x[i]

$$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$$

$$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j} \Delta[j]$$

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$$

History

- 1943 – McCullough-Pitts neuron (can't be trained)
- 1958 – Rosenblatt's perceptron (can be trained)
- 1969 – Minsky and Papert publish *Perceptrons*, which explains the limits of single-layer NNs.
  - Ushers in first "AI Winter"
- 1982 – Backprop algorithm for NNs is published.
  - Was known in the 60s! AI Winter eliminated a lot of AI funding and people were discouraged from working on AI projects.
- 1980s – NNs rise again!
- 1989 – NNs are "universal approximators."

History

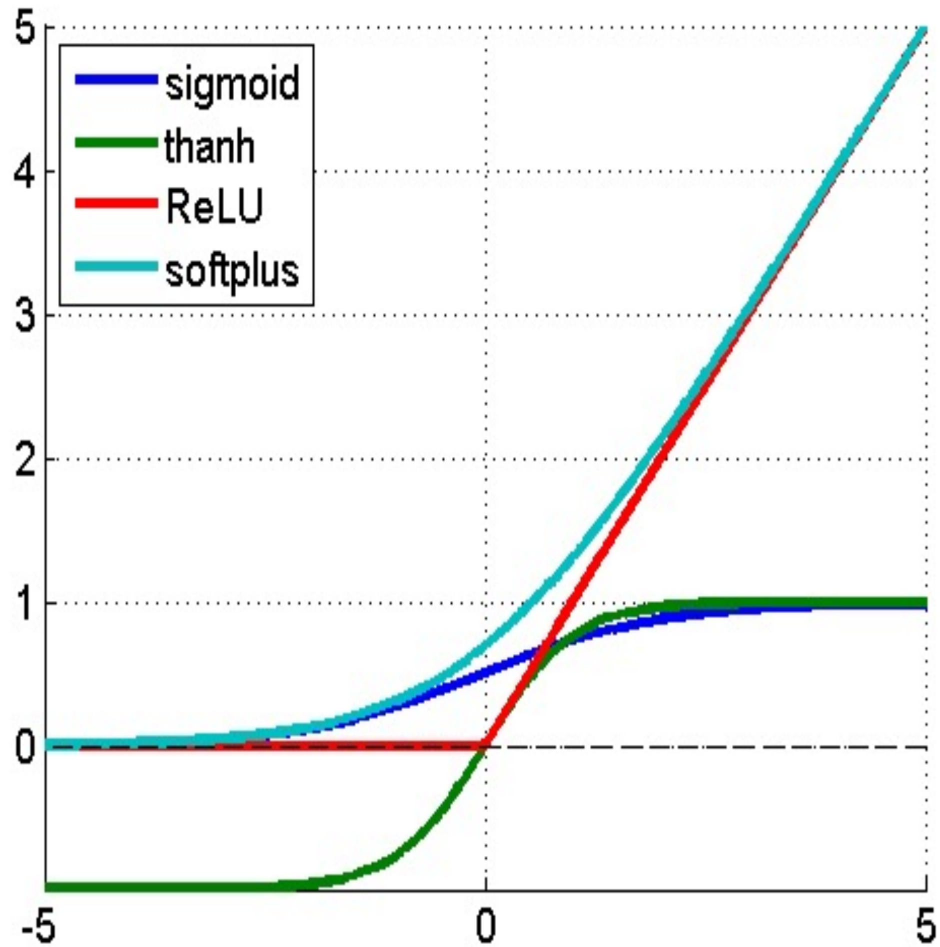
- 1989 – Convolutional NN used to do handwritten digit recognition for ZIP codes. (Yann LeCun)
- 1990s – NNs start to be seen as "painfully slow." Takes a long time to train them. At the same time, people start making more and more modifications to make NNs predict things better – adding more layers, making them recurrent etc.
- Mid 90s – 2nd AI Winter occurs when everything breaks down and the community loses faith in NNs (too slow, too hard to train with backprop, don't work well, nobody understands them anyway).
    - Move to other models, especially probabilistic.

History

- Winter continues through early 2000s, though some people continue working on NNs.
- 2006 paper: "A fast learning algorithm for deep belief nets"
  - Key idea – don't initialize weights randomly. Start off with a round of unsupervised learning to find reasonable initial values for the weights, then finish with regular supervised learning.
- 2^{nd} key idea – pure computational power of GPUs.
  - Massively parallel! 70x faster than training on CPUs.
- 3^{rd} key idea – huge data sets.

History

- 2010 – Turns out the activation function used makes a huge difference on training time and performance.



Lessons

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- We initialized the weights in a stupid way.
- We used the wrong type of non-linearity.