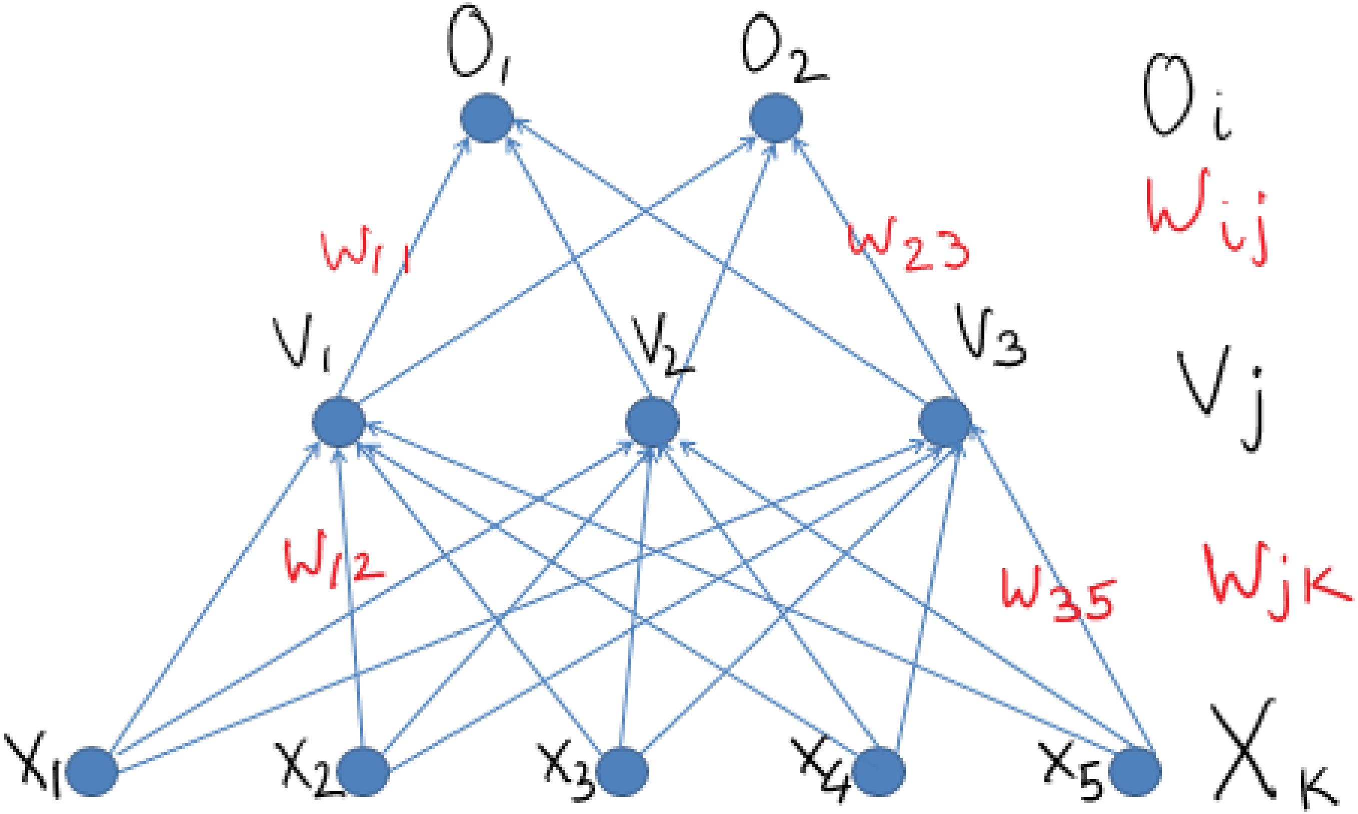
Training a neural network

$$O_i = g\left(\sum_j W_{ij}\, g\left(\sum_k W_{jk}\, x_k\right)\right)$$



Gradient Descent

- **Numerical gradient**: easy to write ☺, slow ☹, approximate ☹
  - $O(N_w^2)$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- **(native) analytic gradient**: exact ☺, compicated ☹, slow ☹
  - $O(N_w^2)$
- **back-propagation (cached analytic gradient)**: exact ☺, fast ☺, error-prone ☹
  - $O(N_w)$, similar to dynamic programming
  - glorified chain rule

In practice: Derive analytic gradient, check your implementation with numerical gradient

The idea behind backpropagation

- We don't know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
- We can compute error derivatives for all the hidden units efficiently at the same time.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

Slide by Geoff Hinton

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Example: x = 4, y = -3. => f(x,y) = -12

$$\boxed{\frac{\partial f}{\partial x} = -3} \quad \boxed{\frac{\partial f}{\partial y} = 4}$$

partial derivatives

$$\boxed{\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]}$$

gradient

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Example: x = 4, y = -3. => f(x,y) = -12

$$\boxed{\frac{\partial f}{\partial x} = -3} \qquad \boxed{\frac{\partial f}{\partial y} = 4}$$

$$\boxed{\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]}$$

partial derivatives

gradient

Question: If I increase x by h, how would the output of f change?

Compound expressions: $f(x,y,z) = (x+y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Compound expressions: $f(x,y,z) = (x+y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/fz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

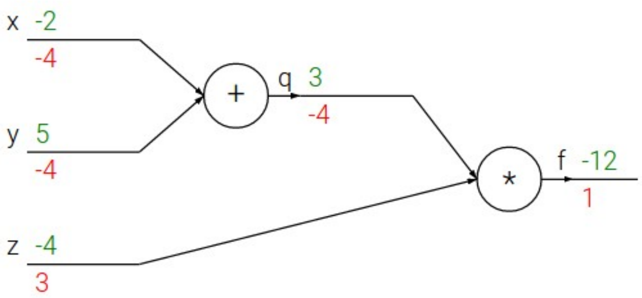
Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



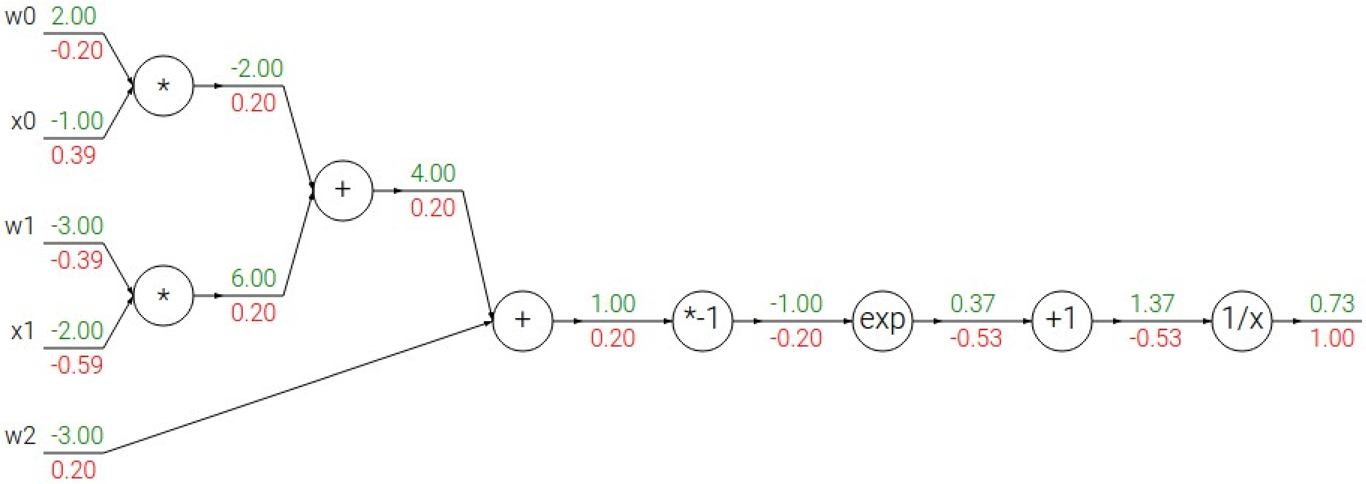

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/fz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

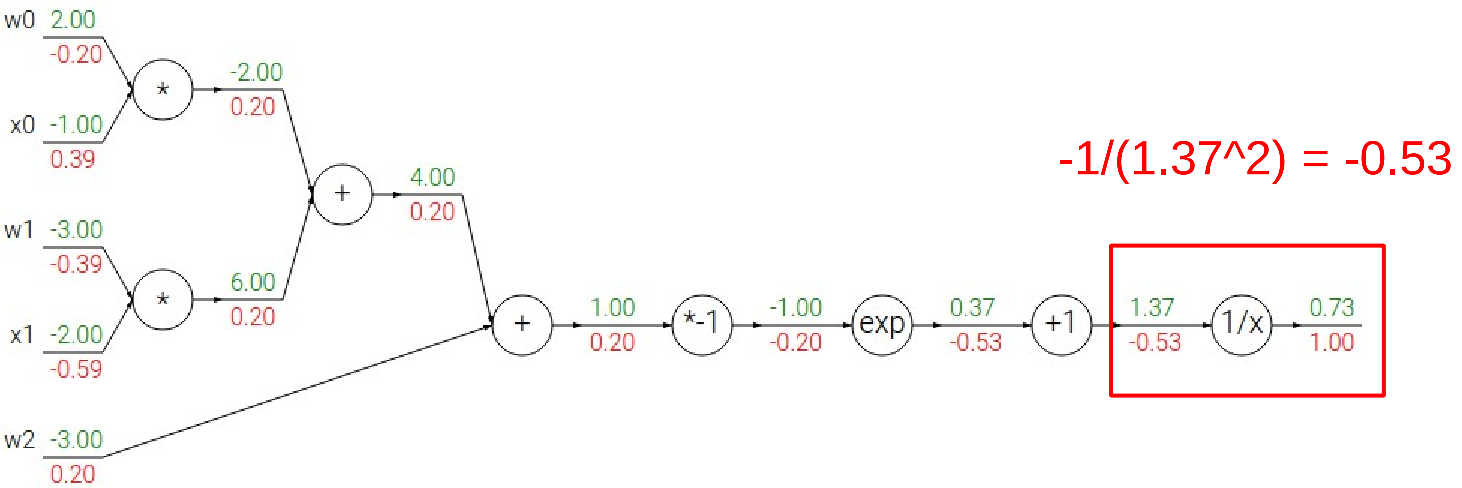
Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$		$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$	
$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$		$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$	

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

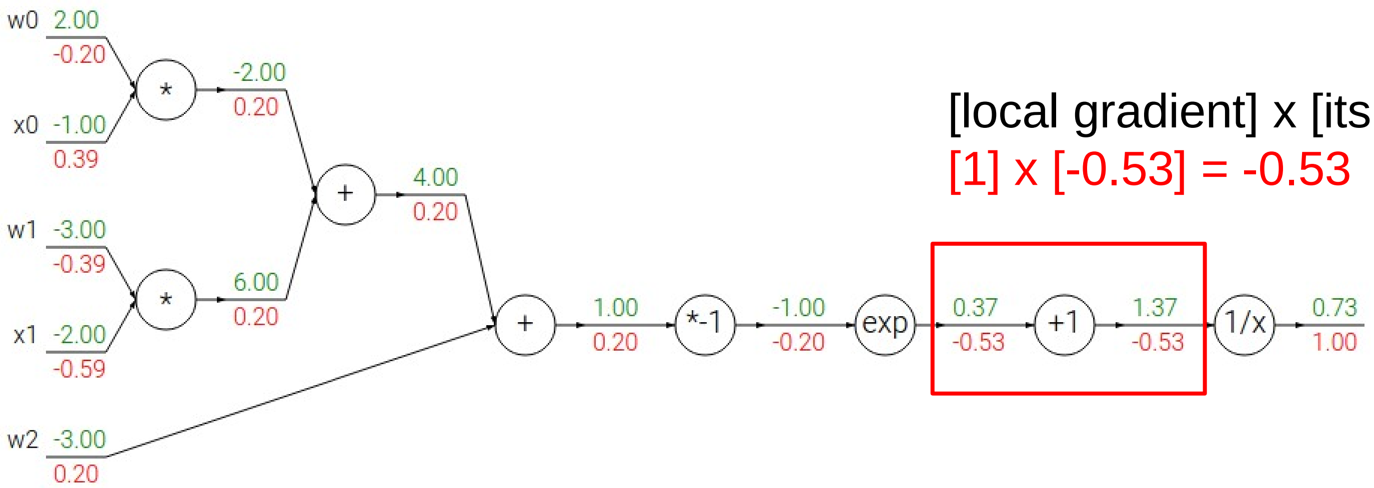
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]

[1] x [-0.53] = -0.53

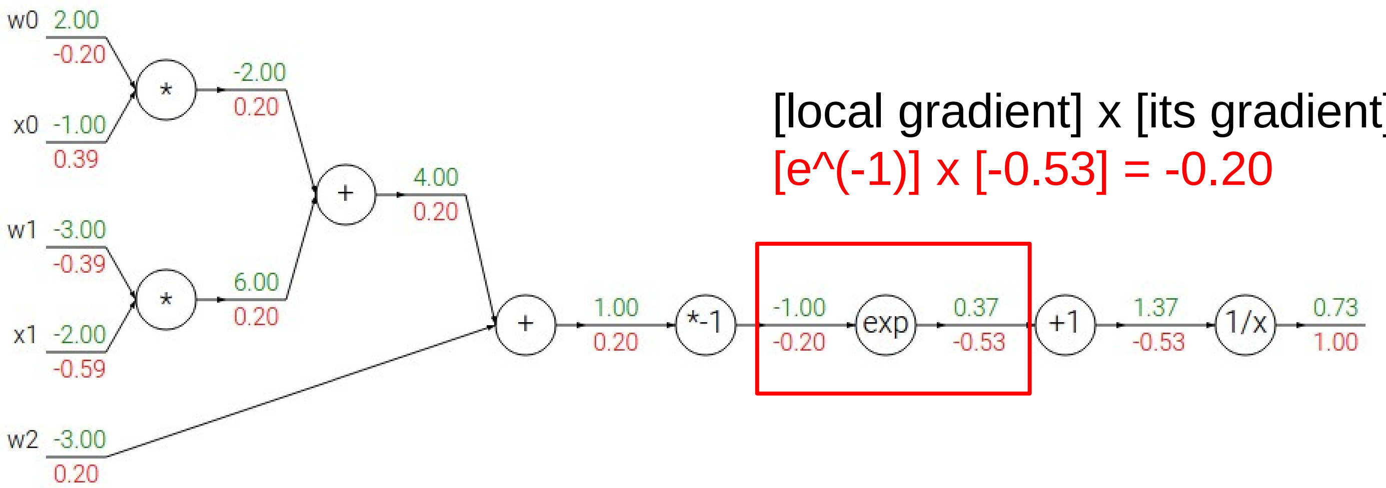
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



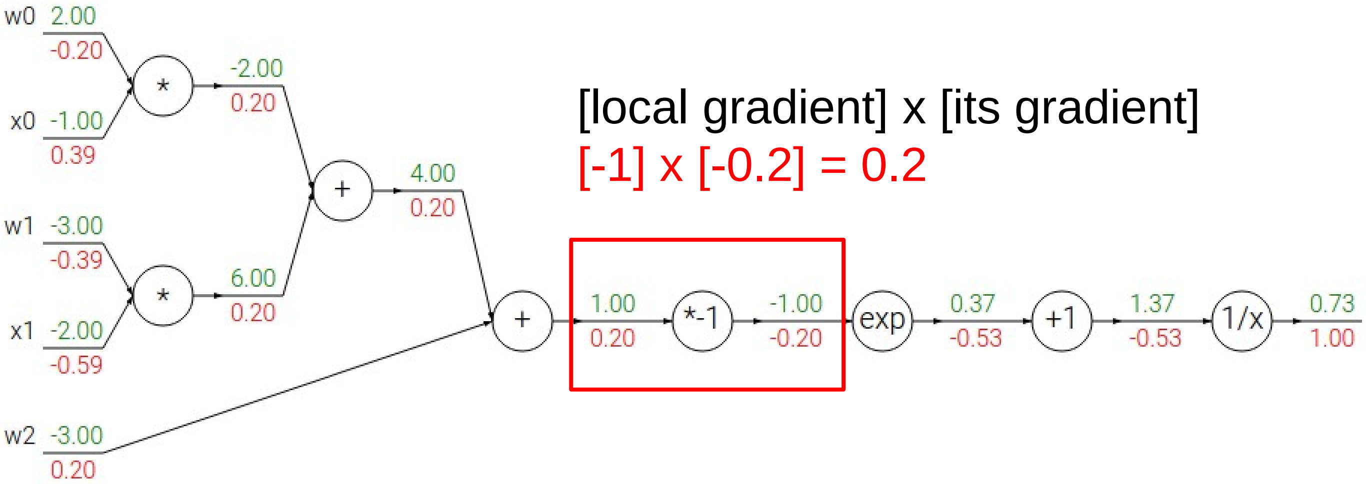
$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

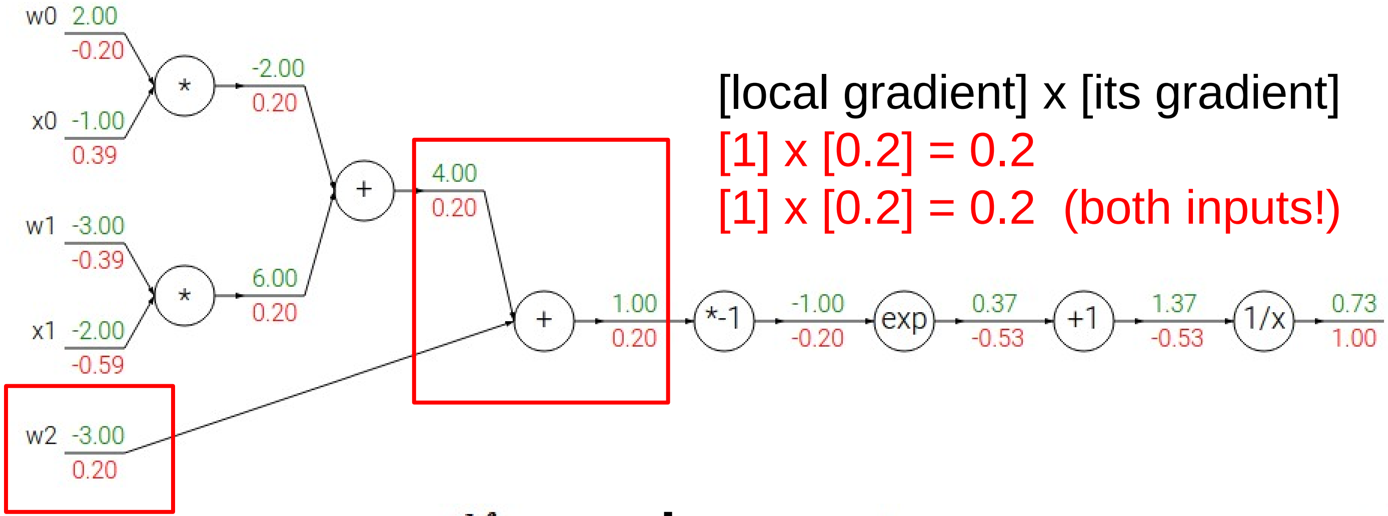


[local gradient] x [its gradient]

[-1] x [-0.2] = 0.2

$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$	$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$
$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$	$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[local gradient] x [its gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$

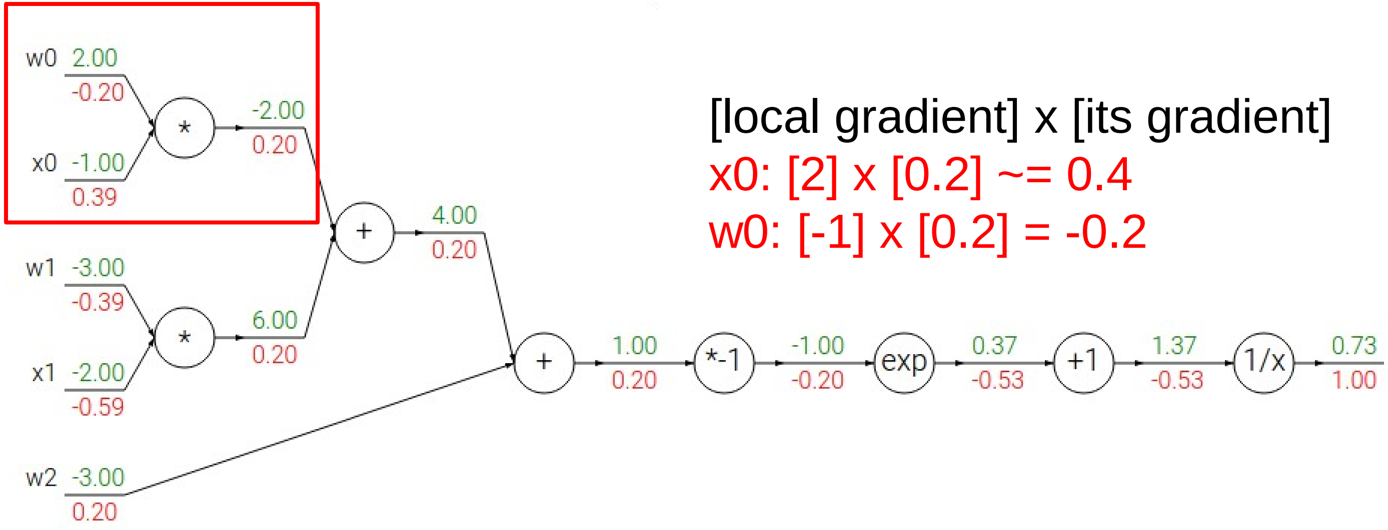
$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]

x0: [2] x [0.2] ~= 0.4

w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Every gate during backprop computes, for all its inputs:

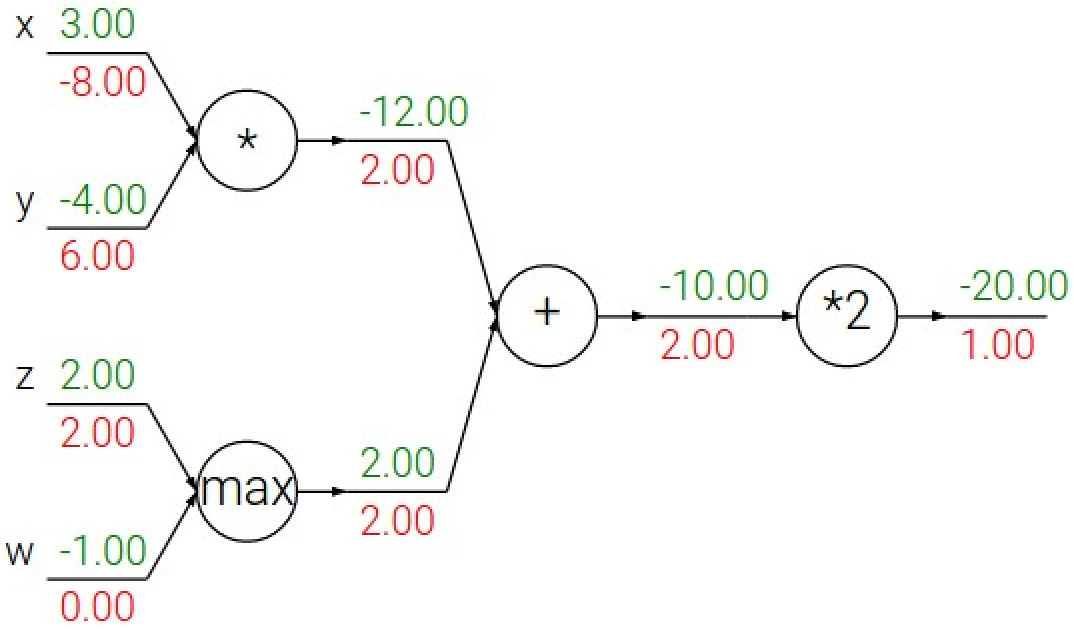
[LOCAL GRADIENT] x [GATE GRADIENT]

Can be computed right away, even during forward pass

The gate receives this during backpropagation

Patterns in backward flow

add gate: gradient distributor
max gate: gradient router
mul gate: gradient… “switcher”?

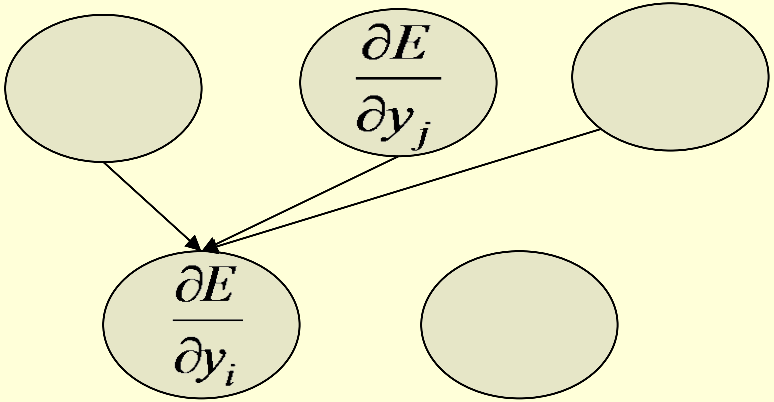


Sketch of the backpropagation algorithm on a single case

- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.
- Then use error derivatives *w.r.t.* activities to get error derivatives *w.r.t.* the incoming weights.

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



The derivatives of a logistic neuron

- The derivatives of the logit, z, with respect to the inputs and the weights are very simple:

$$z = b + \sum_i x_i w_i$$

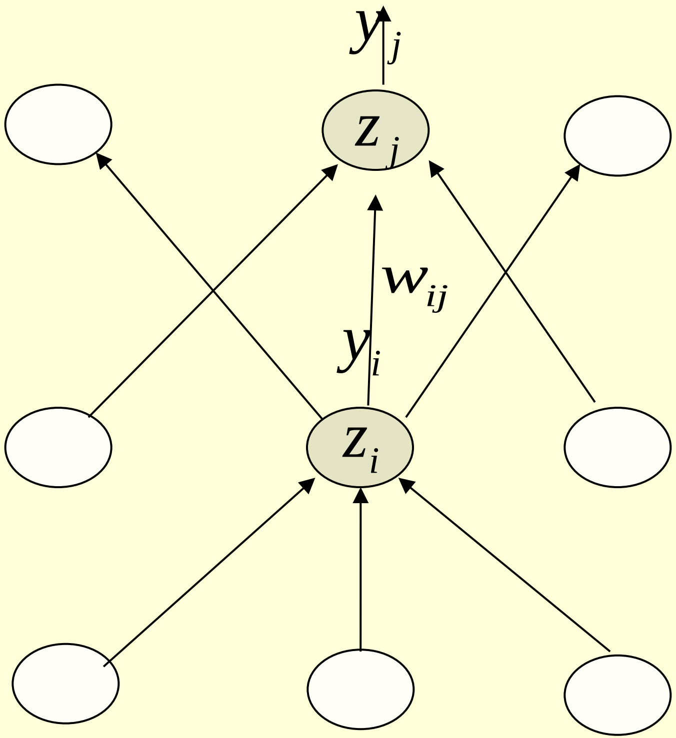
$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- The derivative of the output with respect to the logit is simple if you express it in terms of the output:

$$y = \frac{1}{1+e^{-z}}$$

$$\frac{dy}{dz} = y\,(1-y)$$

Backpropagating dE/dy



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

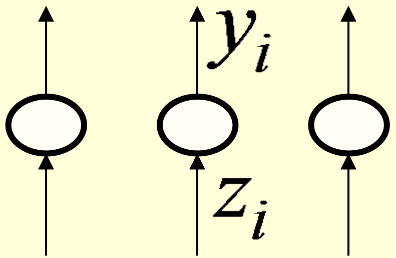
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

Problems with squared error

- The squared error measure has some drawbacks:
  - If the desired output is 1 and the actual output is 0.00000001 there is almost no gradient for a logistic unit to fix up the error.
  - If we are trying to assign probabilities to mutually exclusive class labels, we know that the outputs should sum to 1, but we are depriving the network of this knowledge.
- Is there a different cost function that works better?
  - Yes: Force the outputs to represent a probability distribution across discrete alternatives.

Softmax

The output units in a softmax group use a non-local non-linearity:



softmax group

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

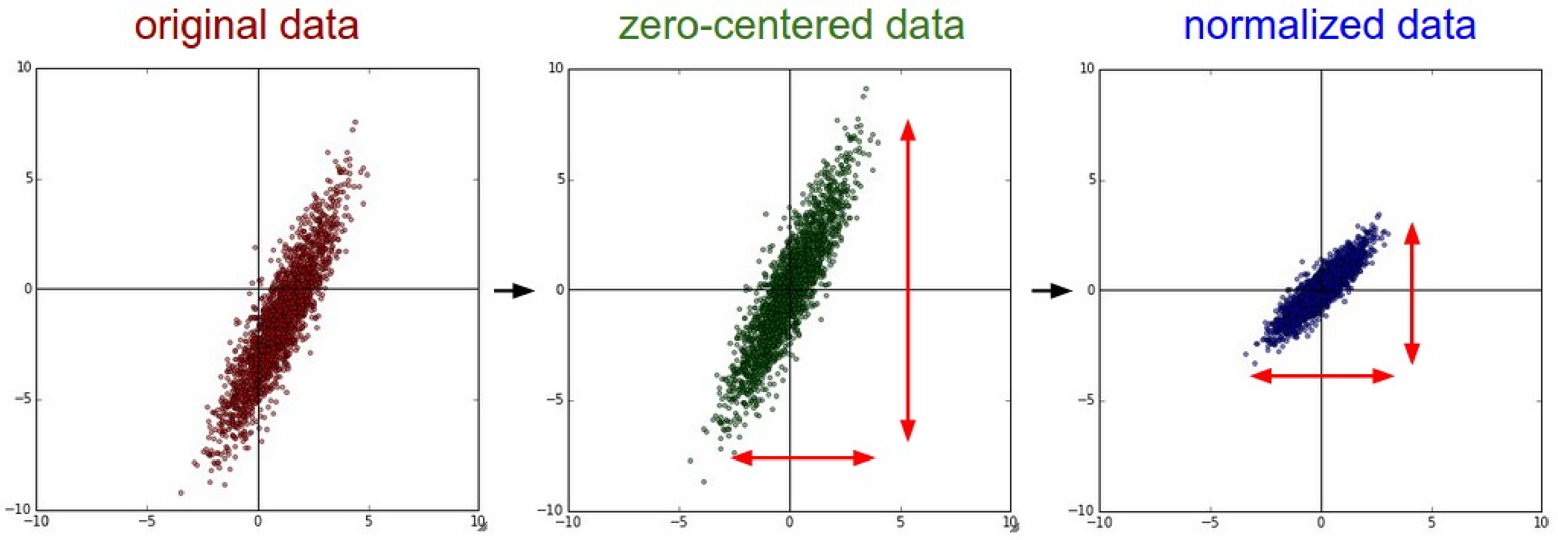
$$\frac{\partial y_i}{\partial z_i} = y_i \, (1 - y_i)$$

Cross-entropy: the right cost function to use with softmax

- The right cost function is the negative log probability of the right answer.
- C has a very big gradient when the target value is 1 and the output is almost zero.

$$C = -\sum_j \underbrace{t_j}_{\text{target value}} \log y_j$$

Preprocessing the data




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

Initializing Weights

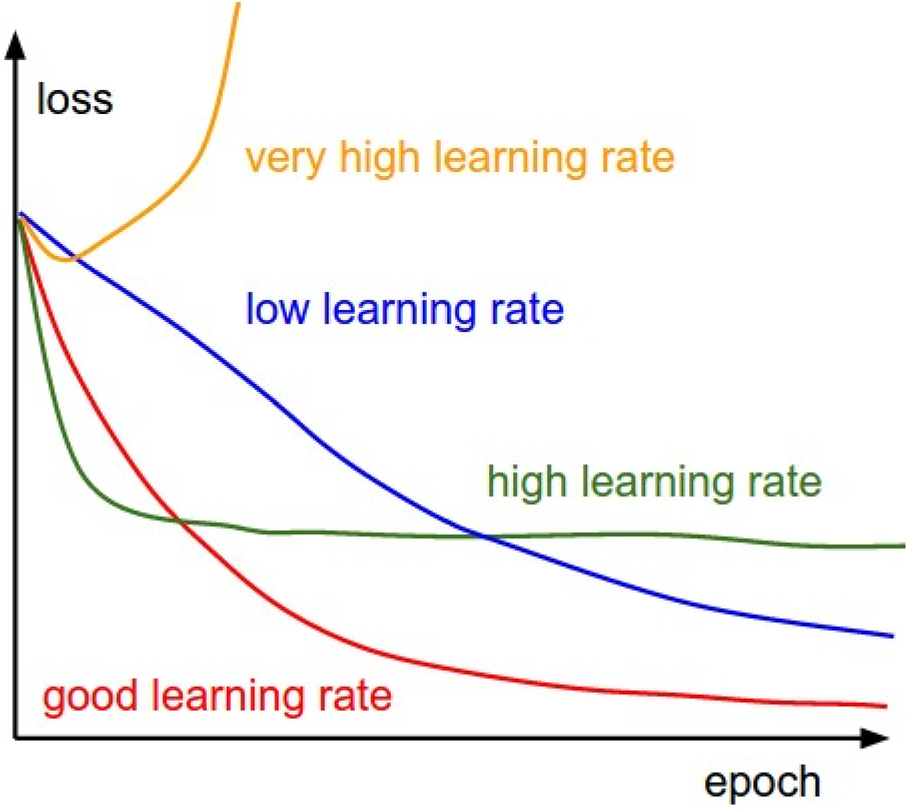
- set weights to small random numbers

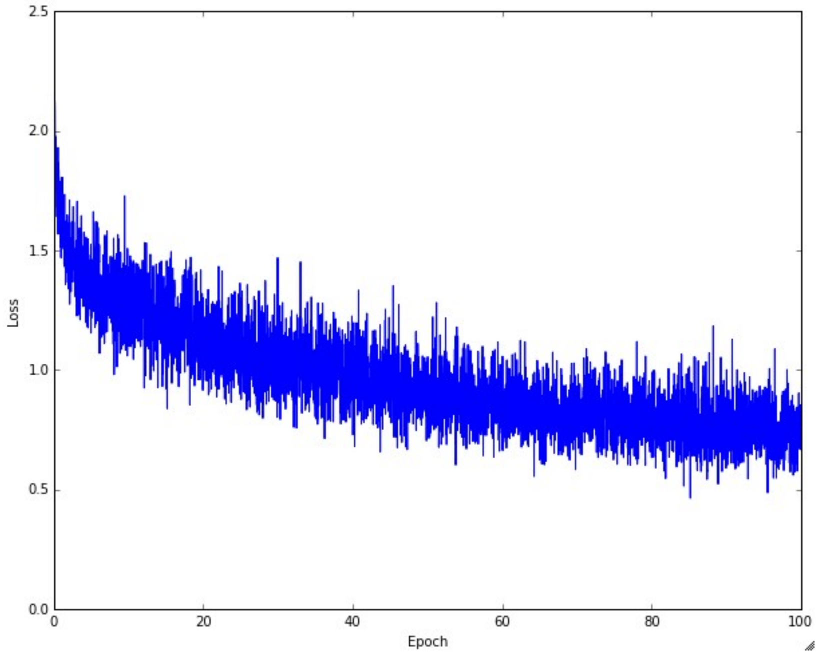
```
W = 0.001* np.random.randn(D,H)
```

(Matrix of small numbers drawn randomly from a gaussian)

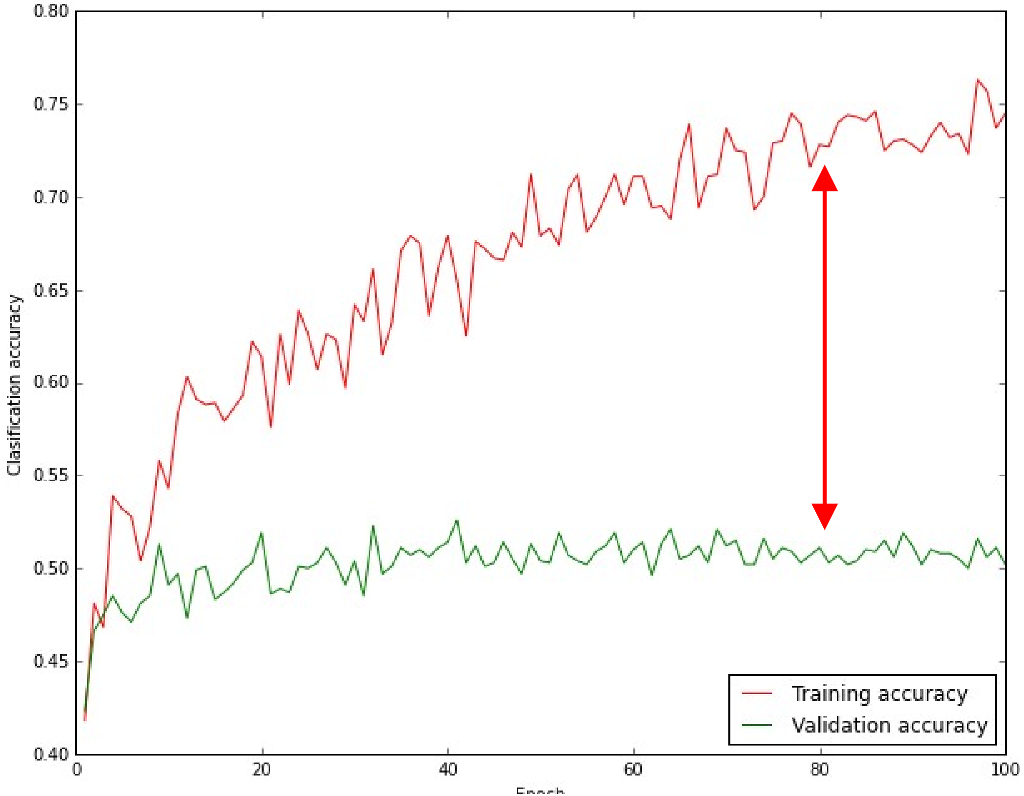
- set biases to zero

Baby-sitting your network: monitoring loss





Baby-sitting your network: monitoring accuracy



big gap = overfitting

=> increase regularization strength

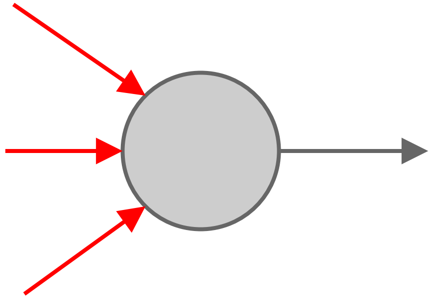
no gap

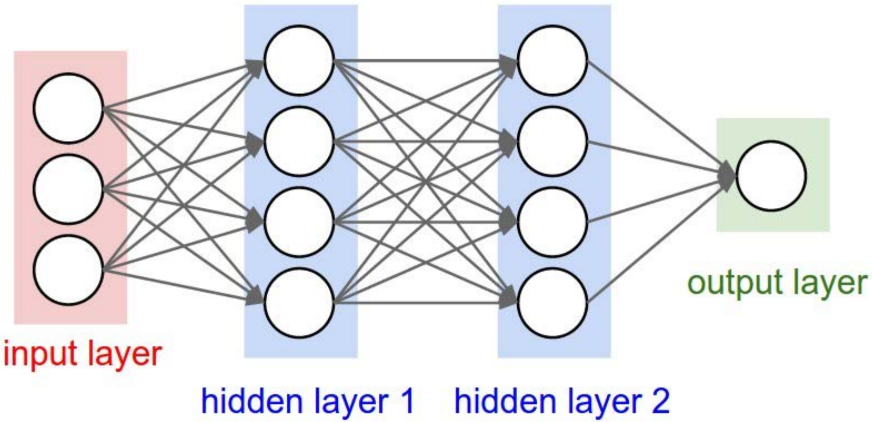
=> increase model capacity

Regularization knobs

- L2 regularization $\frac{1}{2}\lambda w^2$
- L1 regularization $\lambda |w|$
- L1 + L2 can also be combined

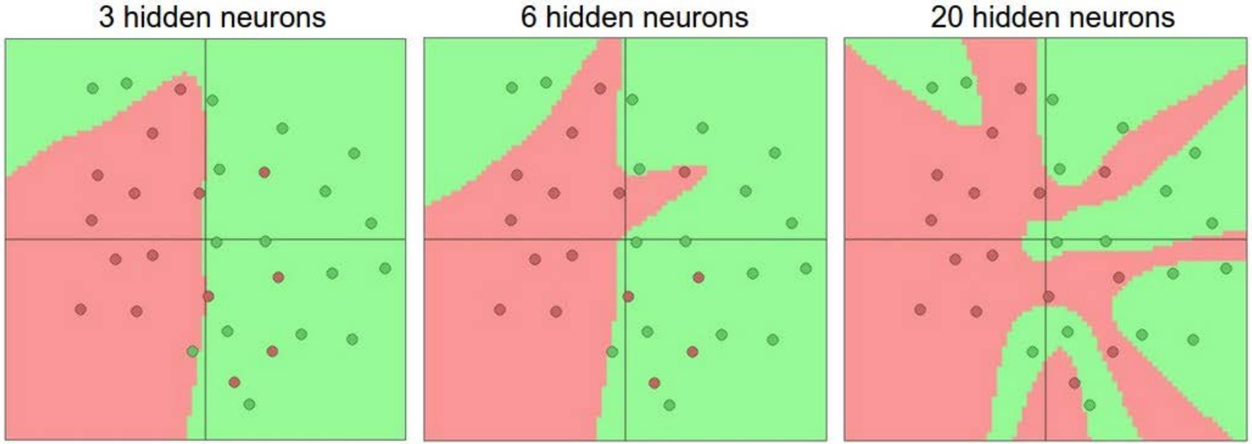
L1 is "sparsity inducing" (many weights become almost exactly zero)



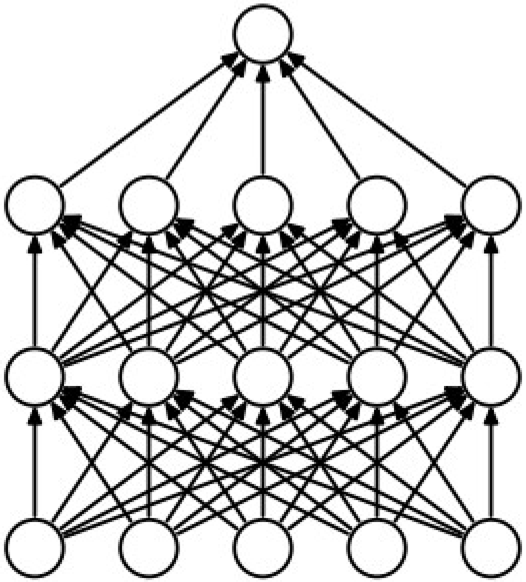


Seemingly unrelated: **Model Ensembles**

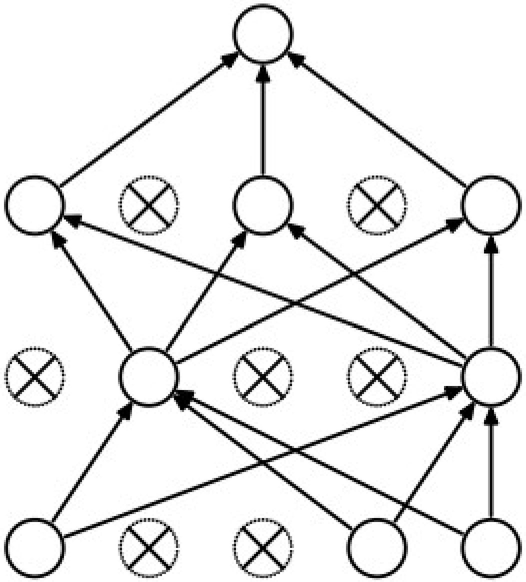
- One way to *always* improve final accuracy: take several trained models and average their predictions



Regularization: **Dropout**
"randomly set some neurons to zero"



(a) Standard Neural Net



(b) After applying dropout.

[Srivastava et al.]