Intro to ML

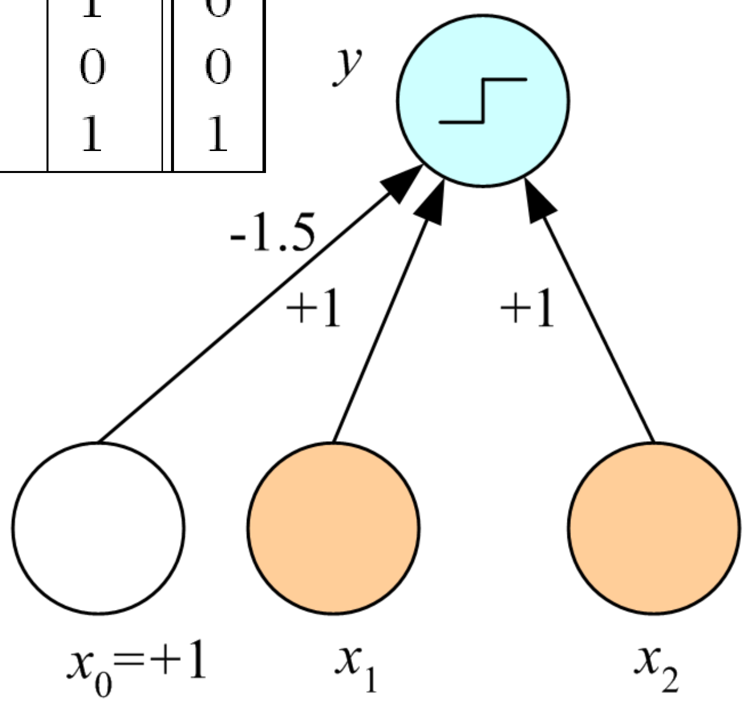
December 1st, 2021

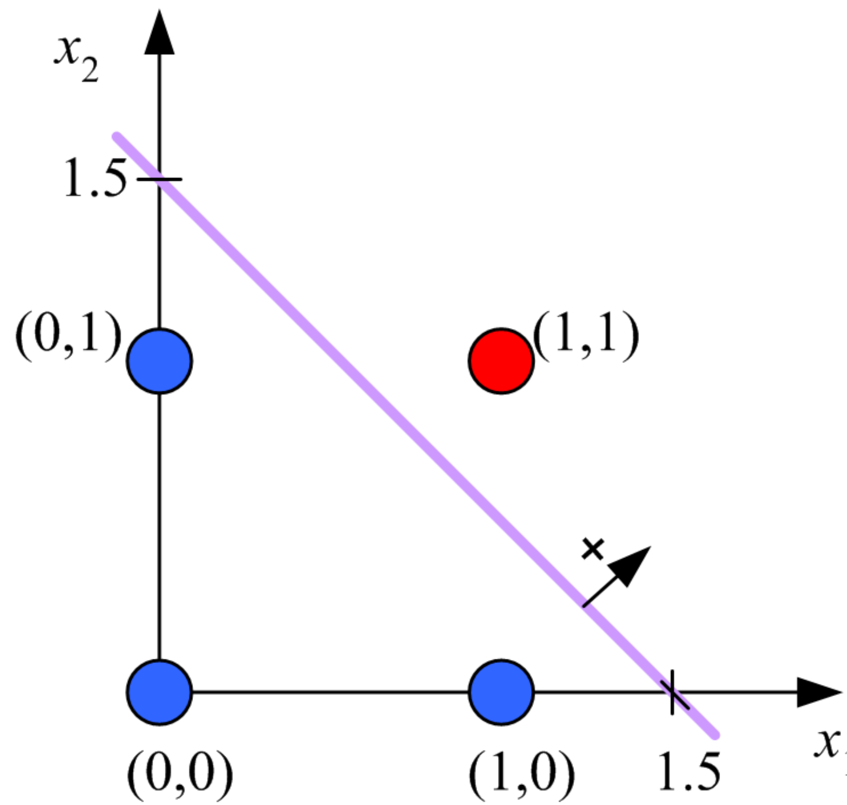
CHAPTER 11:

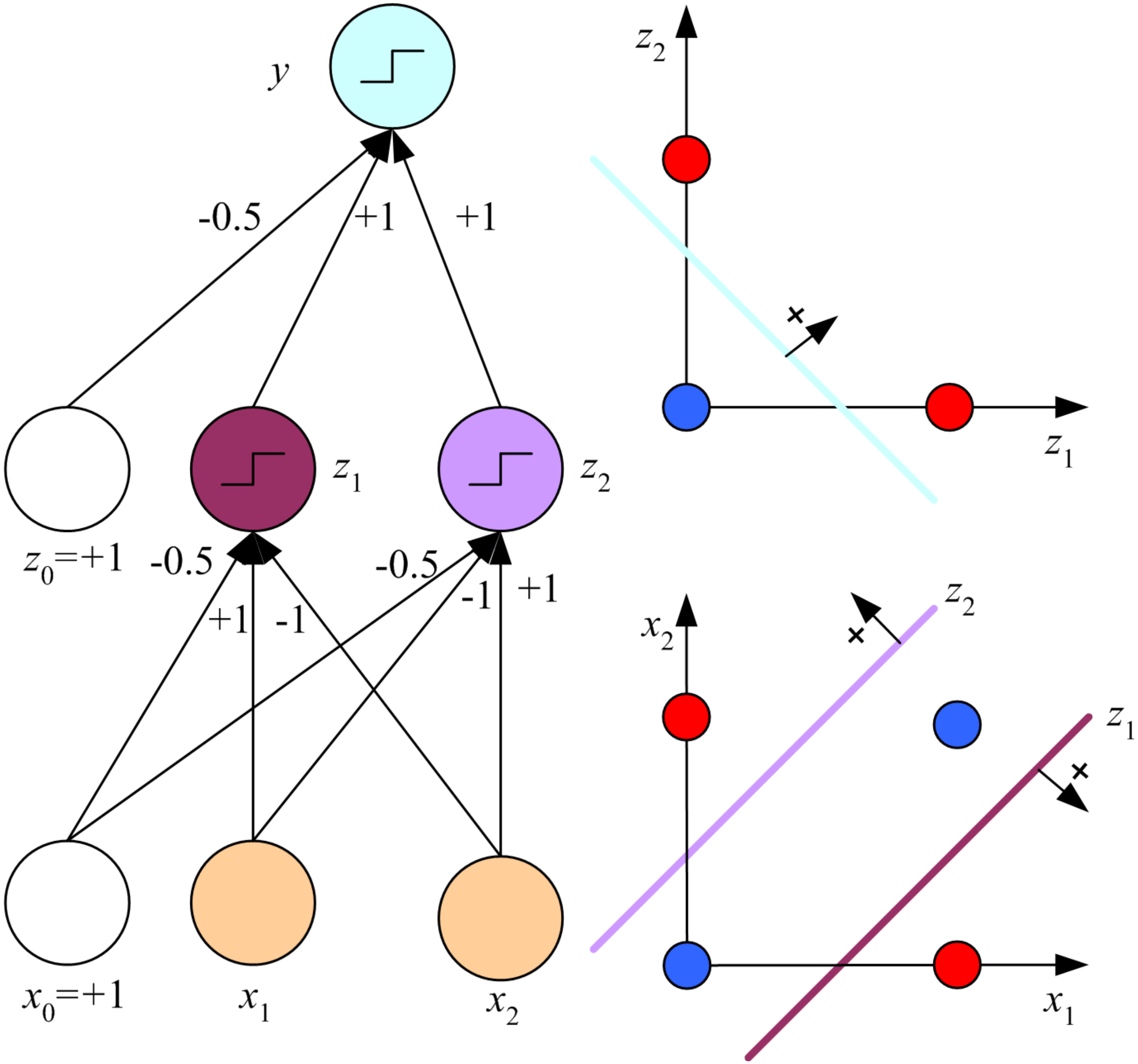
Multilayer Perceptrons

Learning Boolean AND

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1







x_1	x_2	z_1	z_2	y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

Learning Hidden Representations

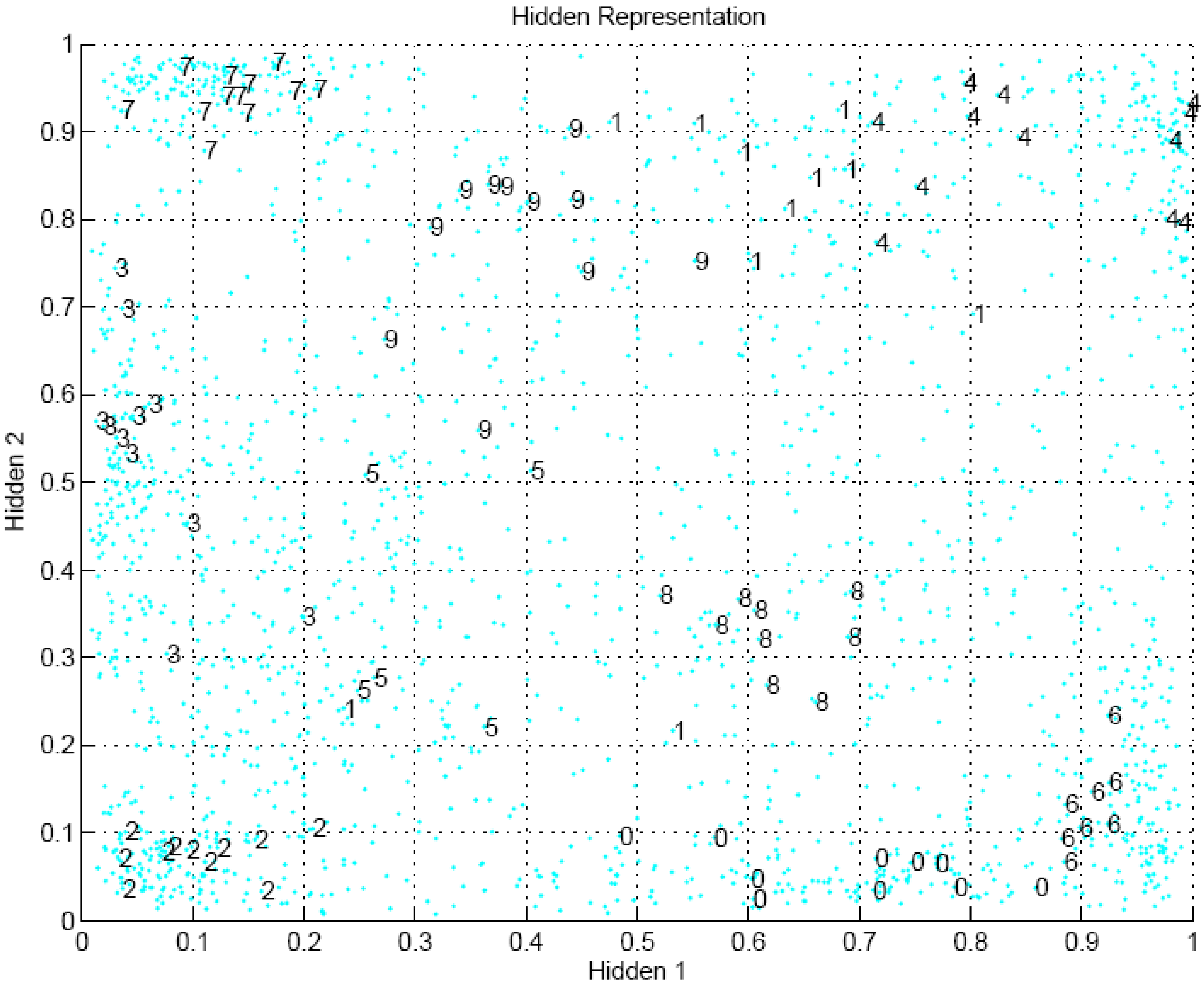
- MLP is a generalized linear model where hidden units are the nonlinear basis functions:

$$y = \sum_{h=1}^{H} v_h \phi(\boldsymbol{x}|\boldsymbol{w}_h)$$

where

$$\phi(\boldsymbol{x}|\boldsymbol{w}_h) \equiv \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x})$$

- The advantage is that the basis function parameters can also be learned from data.
- The hidden units, z_h, learn a *code/embedding*, a representation in the hidden space
  - If H space is < original d dimension: dimension reduction
- Transfer learning: Use code in another task
- Semisupervised learning: Transfer from an unsupervised to supervised problem



100 data points

64 inputs
2 hidden units

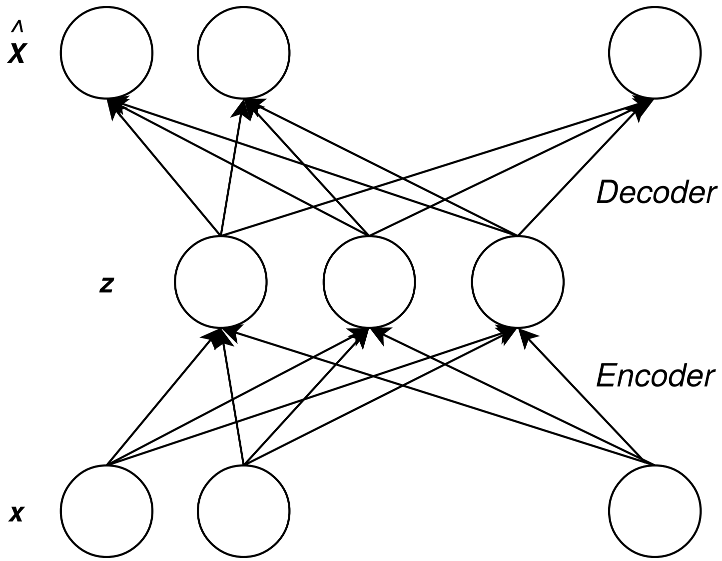
10 output

Autoencoders

An MLP structure

Equal number of input and ouput

Dimension reduction



$$\hat{\boldsymbol{x}}^t = \mathrm{Dec}(\boldsymbol{z}^t|\mathbf{V})$$

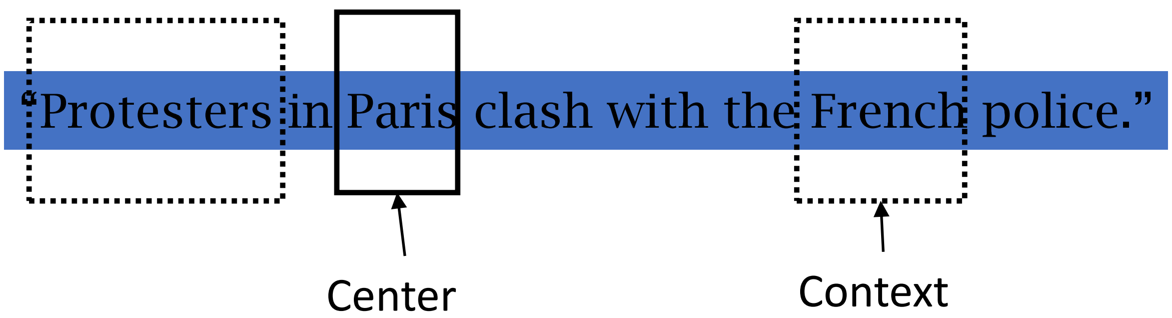
$$\boldsymbol{z}^t = \mathrm{Enc}(\boldsymbol{x}^t|\mathbf{W})$$

$$E(\mathbf{W},\mathbf{V}|\mathcal{X}) = \sum_t \|\boldsymbol{x}^t - \hat{\boldsymbol{x}}^t\|^2 = \sum_t \|\boldsymbol{x}^t - \mathrm{Dec}(\mathrm{Enc}(\boldsymbol{x}^t|\mathbf{W})|\mathbf{V})\|^2$$

- Variants: Denoising, sparse autoencoders

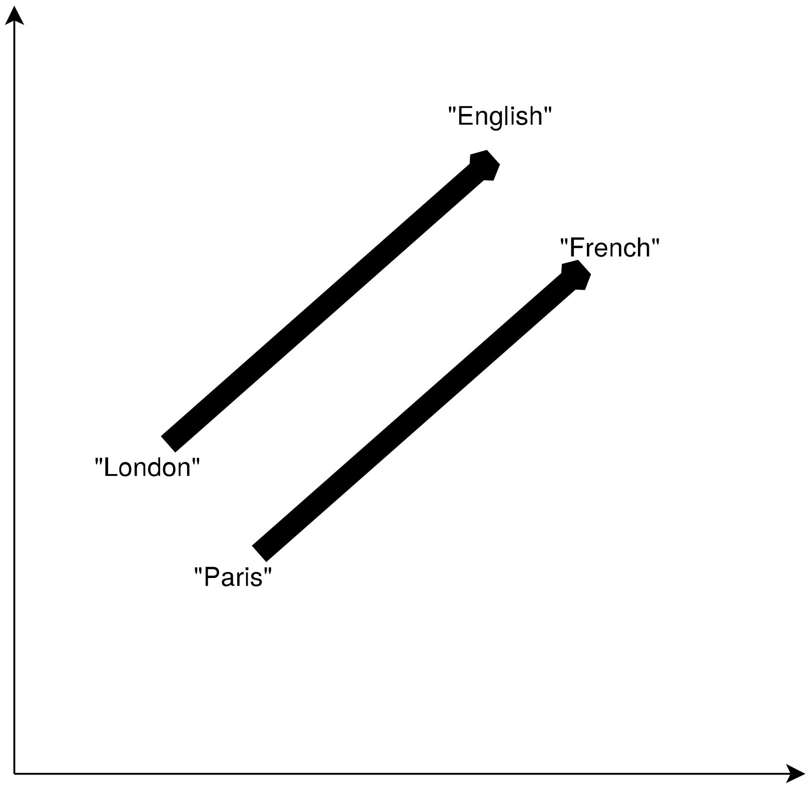
Example: Word2vec

- Learn an embedding for words for NLP
- Skipgram: An autencoder with linear encoder and decoder where input is the center word and output is a context word



- Similar words appear in similar contexts, so similar codes will be learned for them

Vector Algebra



Because they will always appear in similar contexts in pairs in a large corpus, we expect

vec("English")-vec("London")

to be similar to

vec("French")-vec("Paris")

so,

vec("Paris") + vec("English")-vec("London")

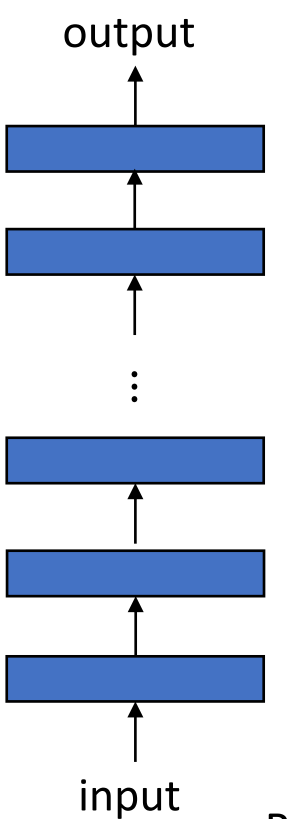
will be similar to

vec("French")

CHAPTER 12:

Deep Learning

Deep Neural Networks



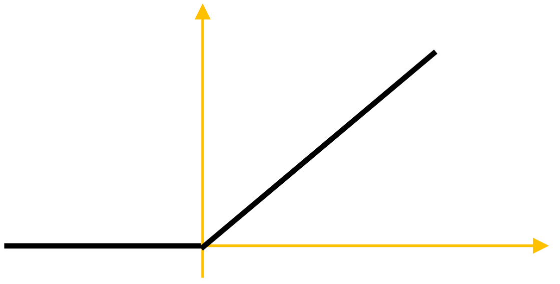
- Many hidden layers
  - Problematic in training: chain rule multiplication of derivatives (vanish of gradients or explosion)
- End-to-end training
- Learn increasingly abstract representations with minimal human contribution
  - Layers of abstraction in intuitive
  - Vision, speech, language, and so on

Representation learning is really the KEY behind Deep learning

Rectified Linear Unit (ReLU)

Left deriviation:
Relu’(a) = 1 if a>0, 0 otherwise

$$\mathrm{ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$



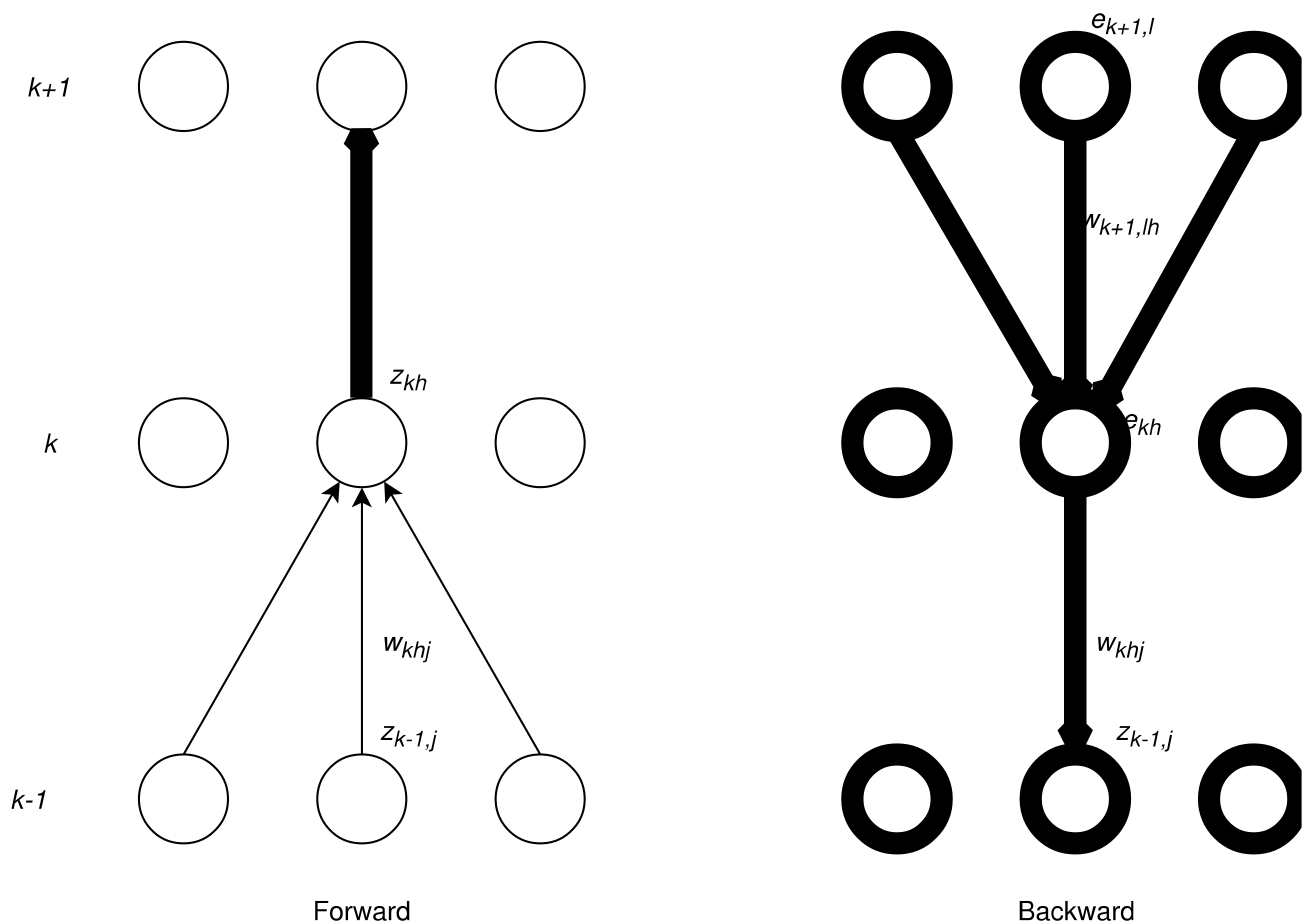
- Does not saturate for large *a*
- Leads to a sparse representation
  - Some of the nodes will results in 0
- No learning for *a*<0, be careful with initialization
  - Initialization should make sure all weights are positive
- Leaky ReLU: {a if a>0, αa otherwise, α is set 0.01)

Some derivative left

Generalizing Backpropagation

$$z^t_{kh} = f\left(\sum_j w_{khj} z^t_{k-1,j}\right)$$

$$e^t_{kh} = \sum_l e^t_{k+1,l} w_{k+1,lh}$$



$$\Delta w_{khj} = \eta \sum_t e_{kh}^t f'(z_{kh}^t) z_{k-1,j}^t$$

- If k is the output layer, there is no layer k+1
  - e_{kh}^t is the derivative of the error function with respect to the output z_{kh}.
- If not, follow the previous backward rules
- f' is the derivative of the activation function

Improving Training Convergence

- Gradient decent. Simply, local, changes uses only the values of the presynaptic and postsynaptic and the error (suitably backpropagated) – can converge slowly

- Momentum. At each update, also add a fraction of the average of past gradients (t denotes iteration):

$$
\begin{aligned}
s_i^t &= \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i} \\
\Delta w_i^t &= -\eta s_i^t
\end{aligned}
$$

Running average of past gradient

Update using such an average

Adaptive Learning Factor

- RMSprop. Make update inversely proportional to the sum of past gradients, so, update more when gradient is small and less where it is large.

$$\Delta w_i^t = -\frac{\eta}{\sqrt{r_i^t}} \frac{\partial E^t}{\partial w_i}$$

where r_i is the accumulated past gradient,

$$r_i^t = \rho r_i^{t-1} + (1-\rho) \left| \frac{\partial E^t}{\partial w_i} \right|^2$$

p is generally set at 0.999

ADAM: Adaptive Learning Factor w/ Momentum

First moment –
just the gradient

$$s_i^t = \alpha s_i^{t-1} + (1-\alpha)\frac{\partial E^t}{\partial w_i} \qquad \Delta w_i^t = -\eta \frac{\tilde{s}_i^t}{\sqrt{\tilde{r}_i^t}}$$

$$r_i^t = \rho r_i^{t-1} + (1-\rho)\left|\frac{\partial E^t}{\partial w_i}\right|^2$$

Second moment –
square of gradient

Initially both *s* and *r* terms are 0, so we bias-correct them (both α and ρ are <1, so they get smaller as *t* gets large):

$$\tilde{s}_i^t = \frac{s_i^t}{1-\alpha^t} \text{ and } \tilde{r}_i^t = \frac{r_i^t}{1-\rho^t}$$