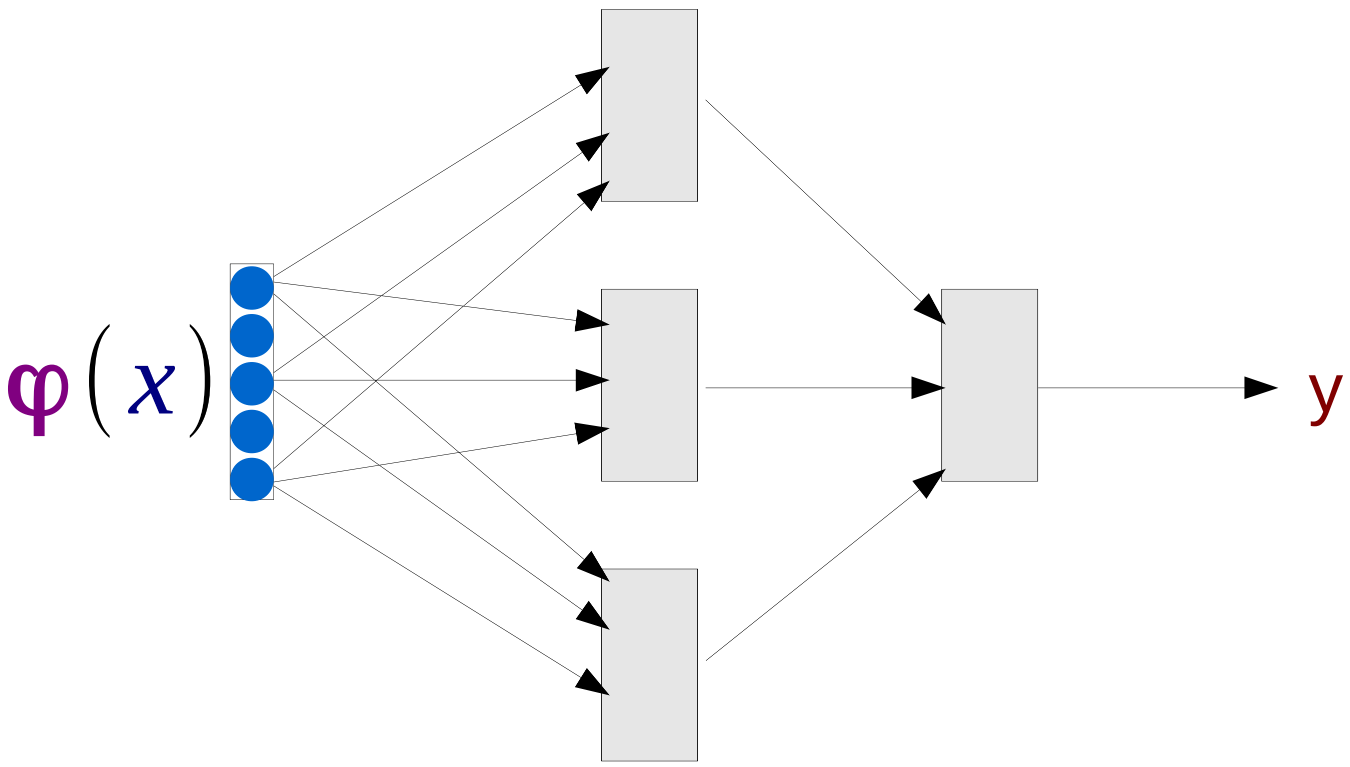
NLP Programming Tutorial 8 - Recurrent Neural Nets

Graham Neubig
Nara Institute of Science and Technology (NAIST)

Feed Forward Neural Nets

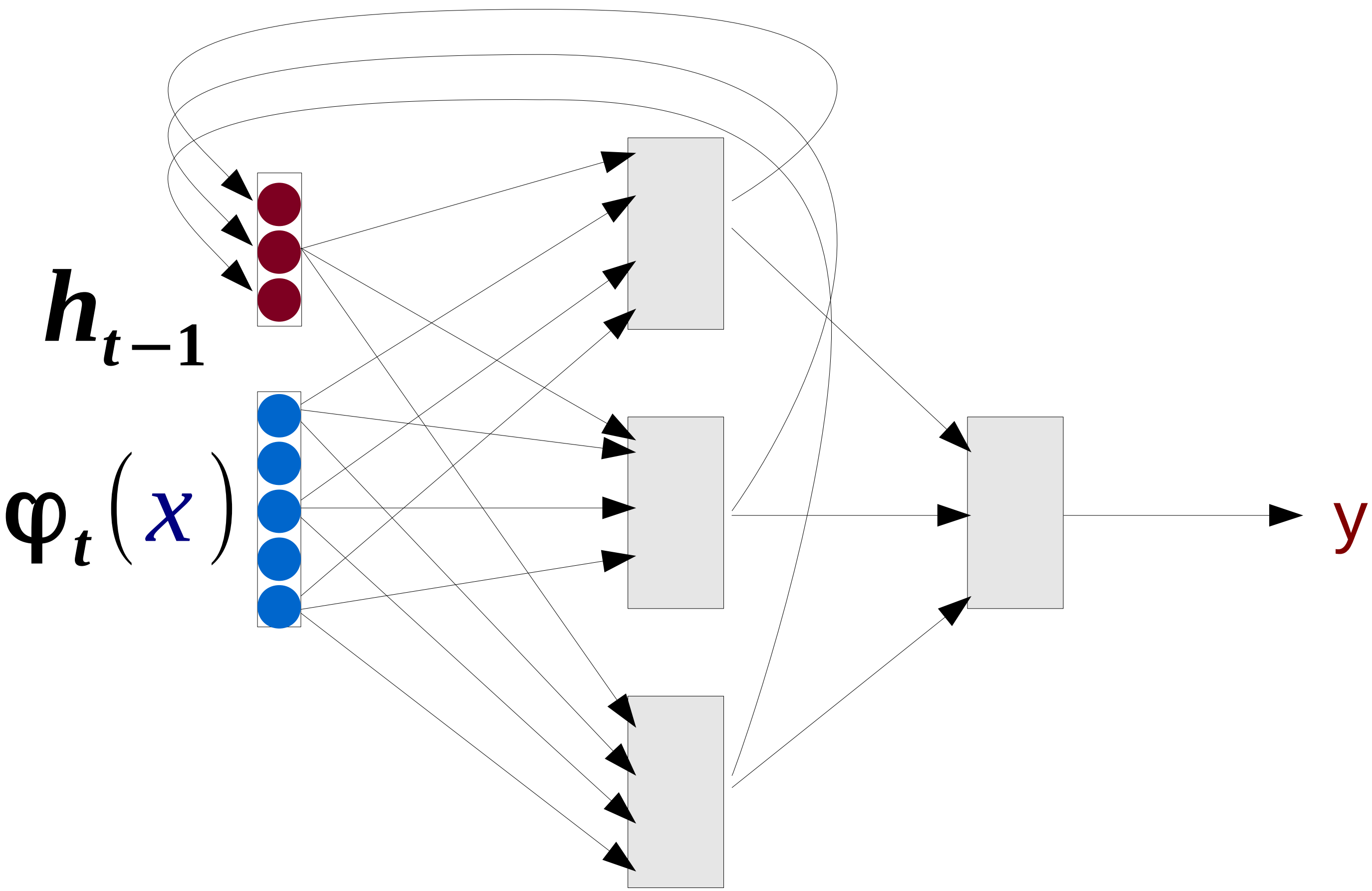
- All connections point forward



- It is a directed acyclic graph (DAG)

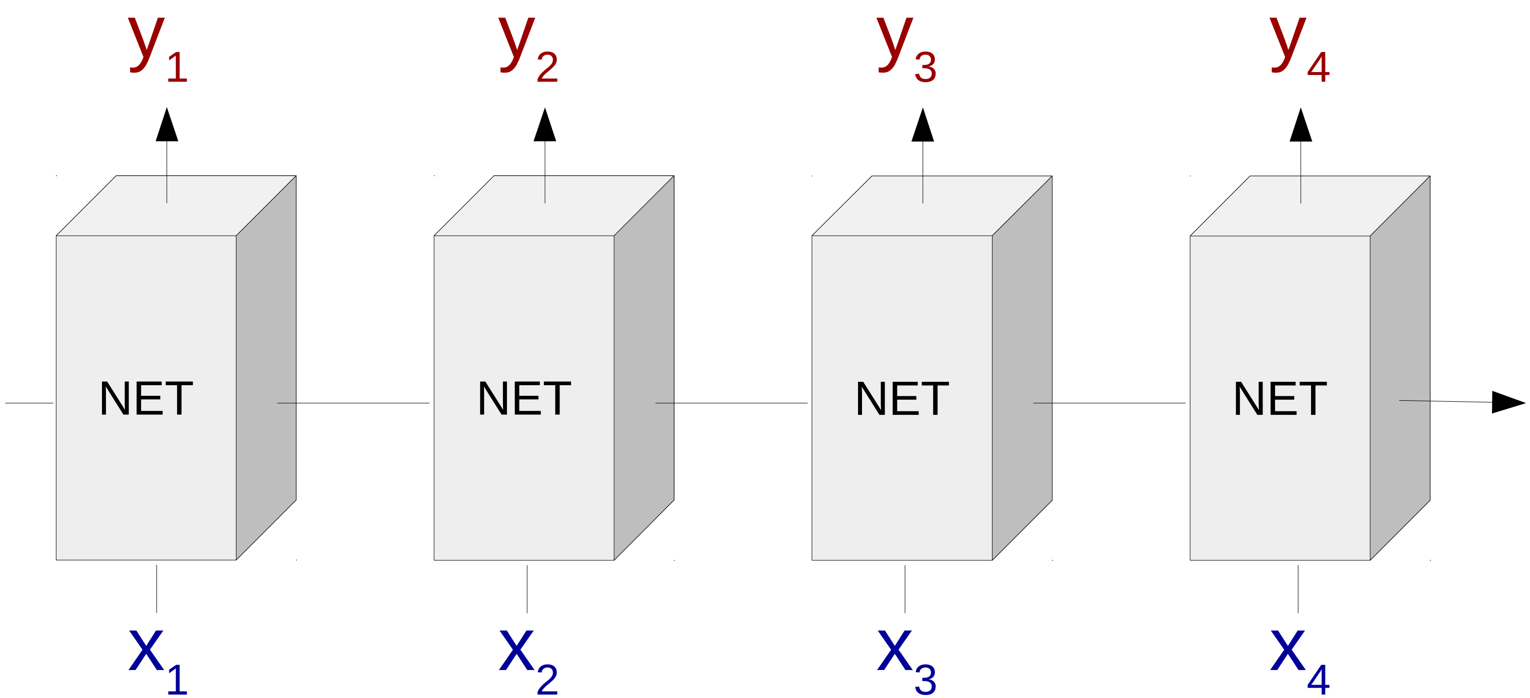
Recurrent Neural Nets (RNN)

- Part of the node outputs return as input

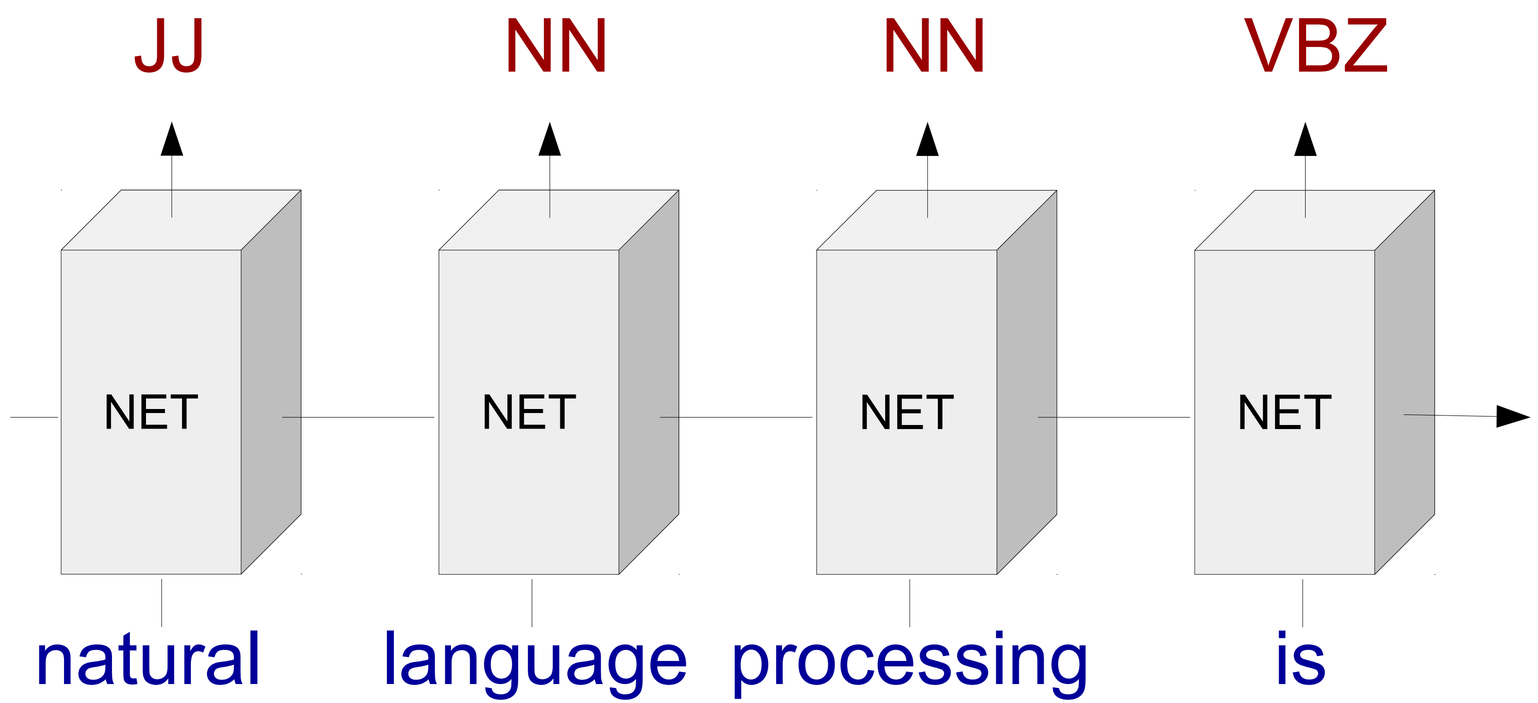


- Why? It is possible to “memorize”

RNN in Sequence Modeling

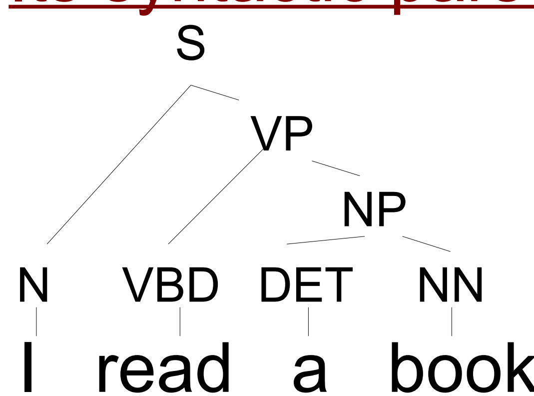


Example: POS Tagging



Multi-class Prediction with Neural Networks

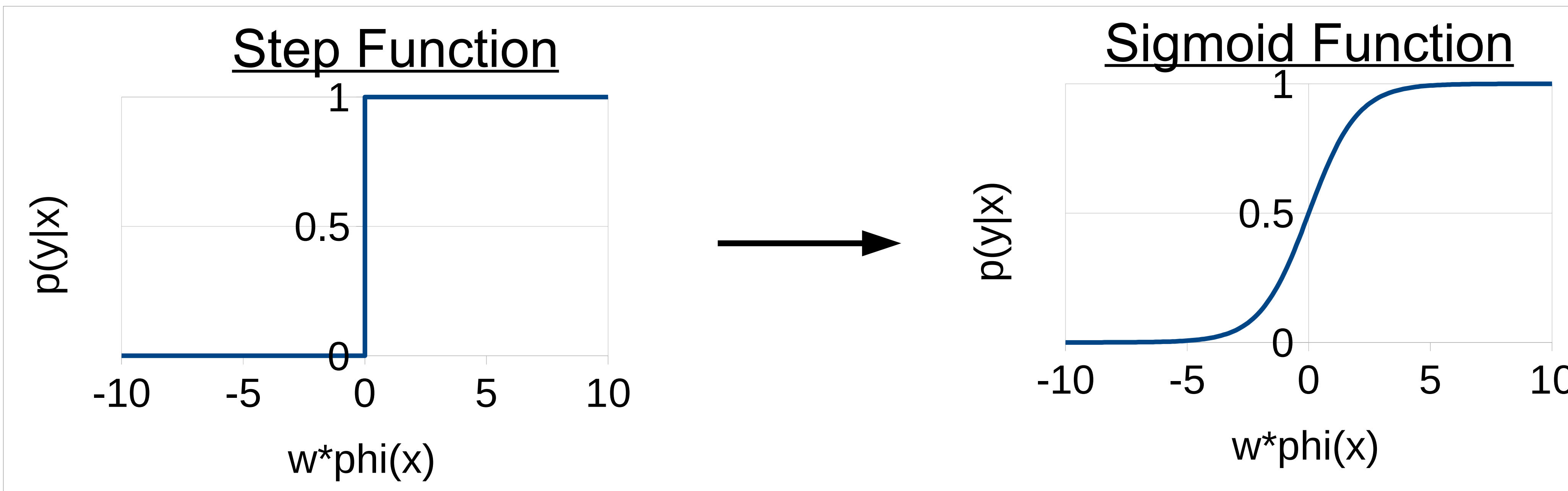
Review: Prediction Problems

Given x,	predict y	
A book review Oh, man I love this book! This book is so boring...	Is it positive? yes no	Binary Prediction (2 choices)
A tweet On the way to the park! 公園に行くなう！	Its language English Japanese	Multi-class Prediction (several choices)
A sentence I read a book	Its syntactic parse 	Structured Prediction (millions of choices)

Review: Sigmoid Function

- The sigmoid softens the step function

$$P(y=1|x)=\frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$



softmax Function

- Sigmoid function for multiple classes

$$P(y|x)=\frac{e^{w\cdot\varphi(x,y)}}{\sum_{\tilde{y}} e^{w\cdot\varphi(x,\tilde{y})}}$$

← Current class

← Sum of other classes

- Can be expressed using matrix/vector ops

$$\boldsymbol{r}=\exp(\boldsymbol{W}\cdot\boldsymbol{\varphi}(x))$$

$$\boldsymbol{p}=\boldsymbol{r}/\sum_{\tilde{r}\in\boldsymbol{r}}\tilde{r}$$

Selecting the Best Value from a Probability Distribution

- Find the index *y* with the highest probability

```
find_best(p):
    y = 0
    for each element i in 1 .. len(p)-1:
        if p[i] > p[y]:
            y = i
    return y
```

softmax Function Gradient

- The difference between the true and estimated probability distributions

$$-d\,\mathrm{err}/d\,\varphi_{out} = \boldsymbol{p}' - \boldsymbol{p}$$

- The true distribution **p'** is expressed with a vector with only the y-th element 1 (a one-hot vector)

$$\boldsymbol{p}' = \{0, 0, \ldots, 1, \ldots, 0\}$$

Creating a 1-hot Vector

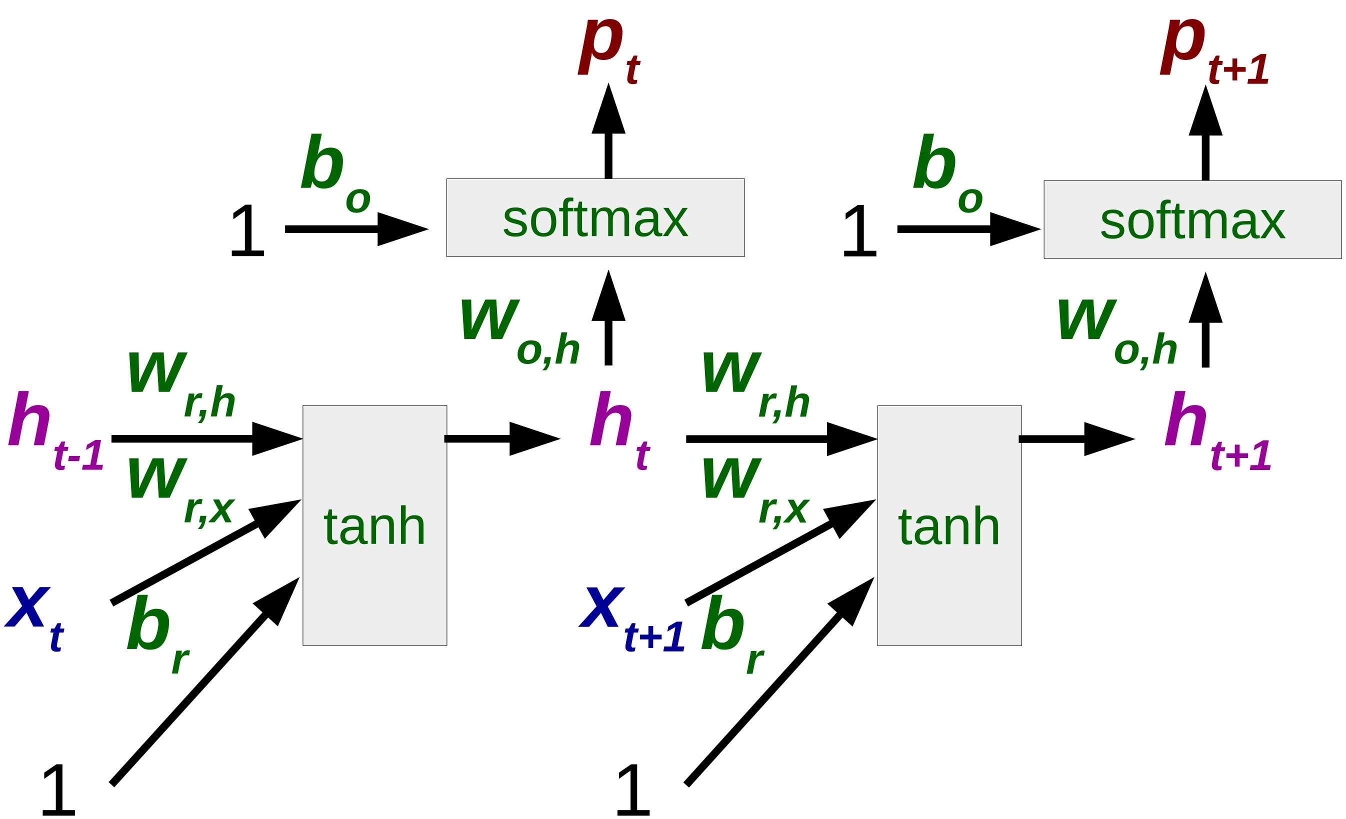
```
create_one_hot(id, size):
    vec = np.zeros(size)
    vec[id] = 1
    return vec
```

Forward Propagation in Recurrent Nets

Review: Forward Propagation Code

```
forward_nn(network, φ0)
    φ= [ φ0 ] # Output of each layer
    for each layer i in 1 .. len(network):
        w, b = network[i-1]
        # Calculate the value based on previous layer
        φ[i] = np.tanh( np.dot( w, φ[i-1] ) + b )
    return φ # Return the values of all layers
```

RNN Calculation



$$h_t = \tanh(w_{r,h} \cdot h_{t-1} + w_{r,x} \cdot x_t + b_r)$$

$$p_t = \text{softmax}(w_{o,h} \cdot h_t + b_o)$$

RNN Forward Calculation

```
forward_rnn(w_{r,x}, w_{r,h}, b_r, w_{o,h}, b_o, x)
    h = [ ] # Hidden layers (at time t)
    p = [ ] # Output probability distributions (at time t)
    y = [ ] # Output values (at time t)
    for each time t in 0 .. len(x)-1:
        if t > 0:
            h[t] = tanh(w_{r,x} x[t] + w_{r,h} h[t-1] + b_r)
        else:
            h[t] = tanh(w_{r,x} x[t] + b_r)
        p[t] = tanh(w_{o,h} h[t] + b_o)
        y[t] = find_max(p[t])
    return h, p, y
```

Review:
Back Propagation in Feed-forward Nets

Stochastic Gradient Descent

- Online training algorithm for probabilistic models (including logistic regression)

```
w = 0
for I iterations
    for each labeled pair x, y in the data
        w += α * dP(y|x)/dw
```

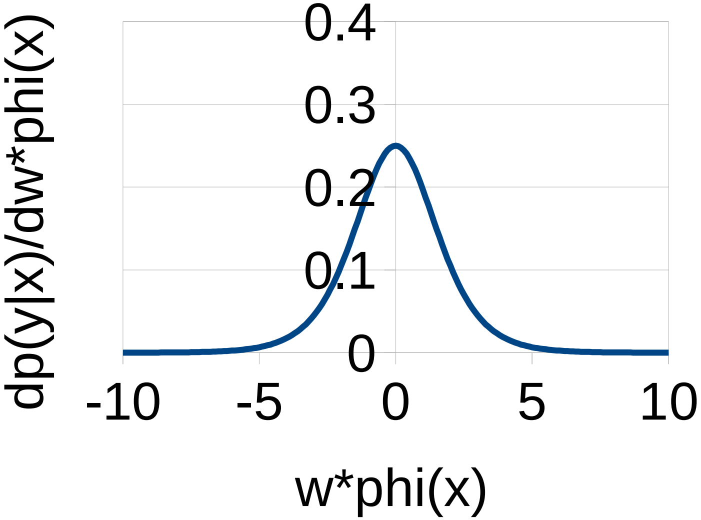
- In other words
  - For every training example, calculate the gradient (the direction that will increase the probability of y)
  - Move in that direction, multiplied by learning rate α

Gradient of the Sigmoid Function

- Take the derivative of the probability

$$\frac{d}{d\,w} P(y=1|x) = \frac{d}{d\,w} \frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}$$

$$= \varphi(x) \frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$



$$\frac{d}{d\,w} P(y=-1|x) = \frac{d}{d\,w} \left(1 - \frac{e^{w\cdot\varphi(x)}}{1+e^{w\cdot\varphi(x)}}\right)$$

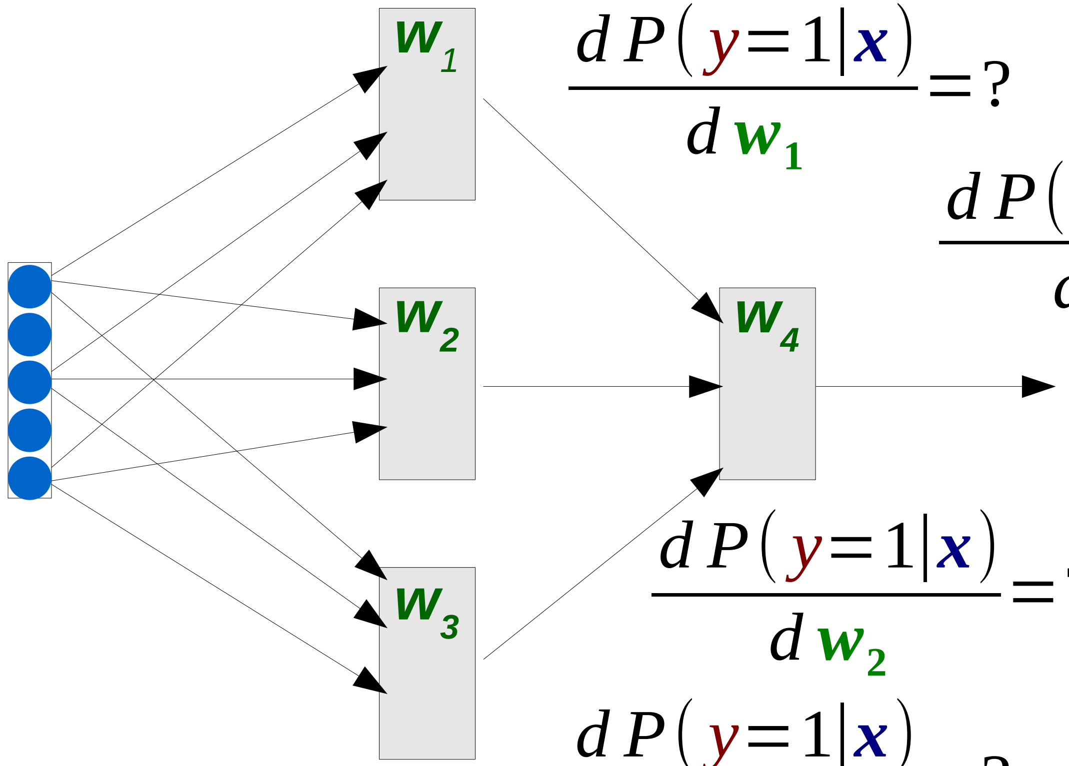
$$= -\varphi(x) \frac{e^{w\cdot\varphi(x)}}{(1+e^{w\cdot\varphi(x)})^2}$$

Learning: Don't Know Derivative for Hidden Units!

- For NNs, only know correct tag for last layer

$h(x)$

$\varphi(x)$



$$\frac{d\,P(y=1|\mathbf{x})}{d\,\mathbf{w_1}}=?$$

$$\frac{d\,P(y=1|\mathbf{x})}{d\,\mathbf{w_4}}=\mathbf{h}(x)\frac{e^{\mathbf{w_4}\cdot \mathbf{h}(x)}}{(1+e^{\mathbf{w_4}\cdot \mathbf{h}(x)})^2}$$

$$\frac{d\,P(y=1|\mathbf{x})}{d\,\mathbf{w_2}}=?$$

$$\frac{d\,P(y=1|\mathbf{x})}{d\,\mathbf{w_3}}=?$$

Answer: Back-Propogation

- Calculate derivative w/ chain rule

$$\frac{d P(y=1|x)}{d \mathbf{w}_1} = \frac{d P(y=1|x)}{d \mathbf{w}_4 \mathbf{h}(\mathbf{x})} \frac{d \mathbf{w}_4 \mathbf{h}(\mathbf{x})}{d h_1(\mathbf{x})} \frac{d h_1(\mathbf{x})}{d \mathbf{w}_1}$$

$$\frac{e^{\mathbf{w}_4 \cdot \mathbf{h}(x)}}{(1+e^{\mathbf{w}_4 \cdot \mathbf{h}(x)})^2} \qquad w_{1,4}$$

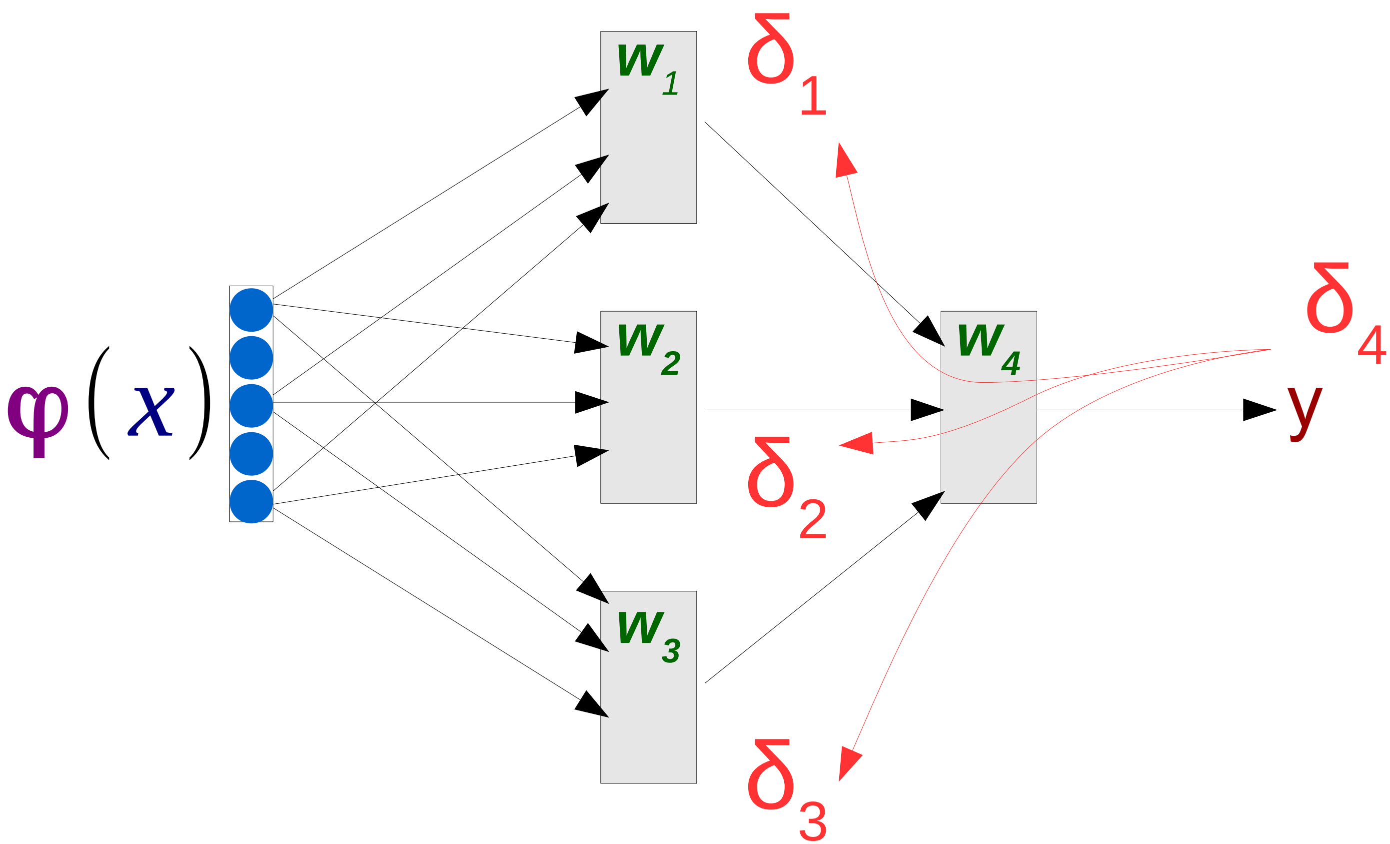
Error of next unit (δ_4) | Weight | Gradient of this unit

In General
Calculate *i* based on next units *j*:

$$\frac{d P(y=1|\mathbf{x})}{\mathbf{w}_i} = \frac{d h_i(\mathbf{x})}{d \mathbf{w}_i} \sum_j \delta_j w_{i,j}$$

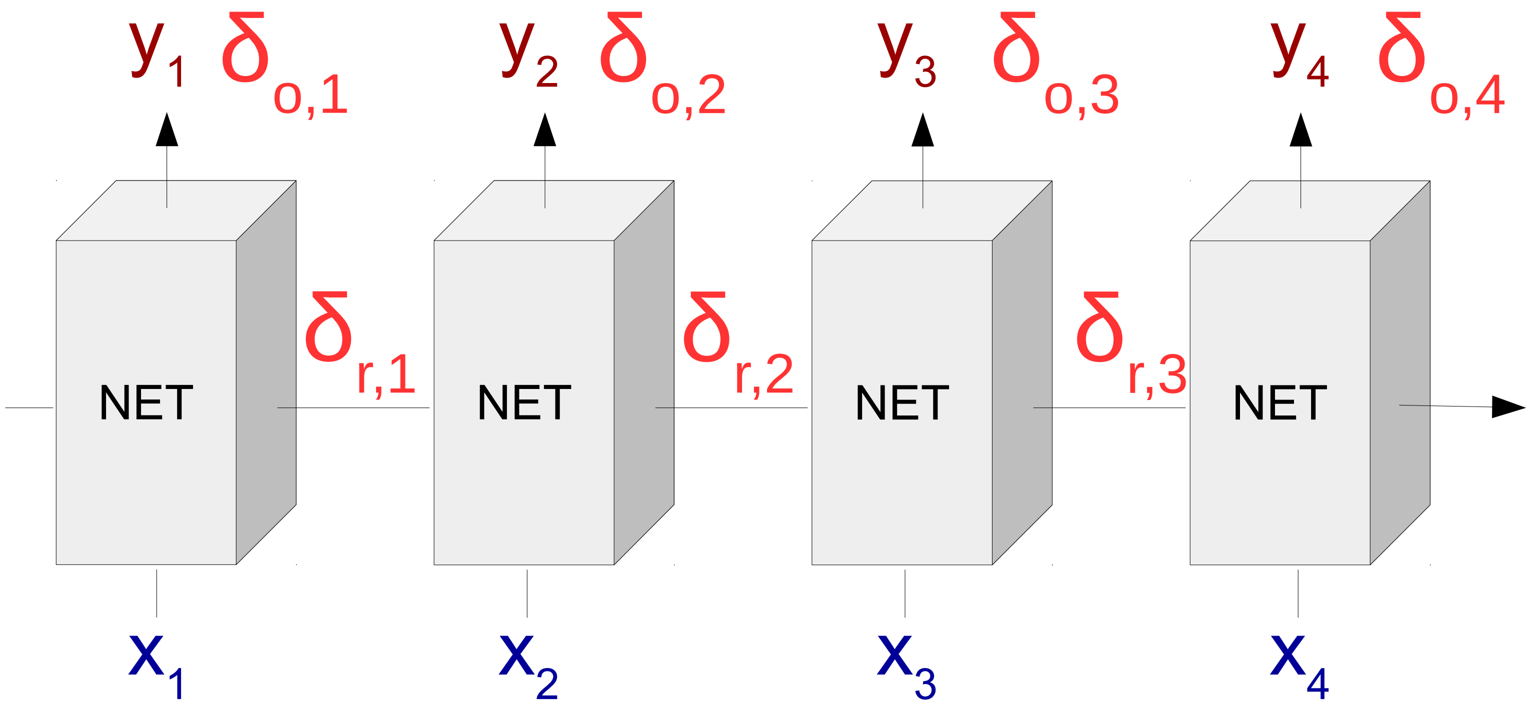
Conceptual Picture

- Send errors back through the net



Back Propagation in Recurrent Nets

What Errors do we Know?

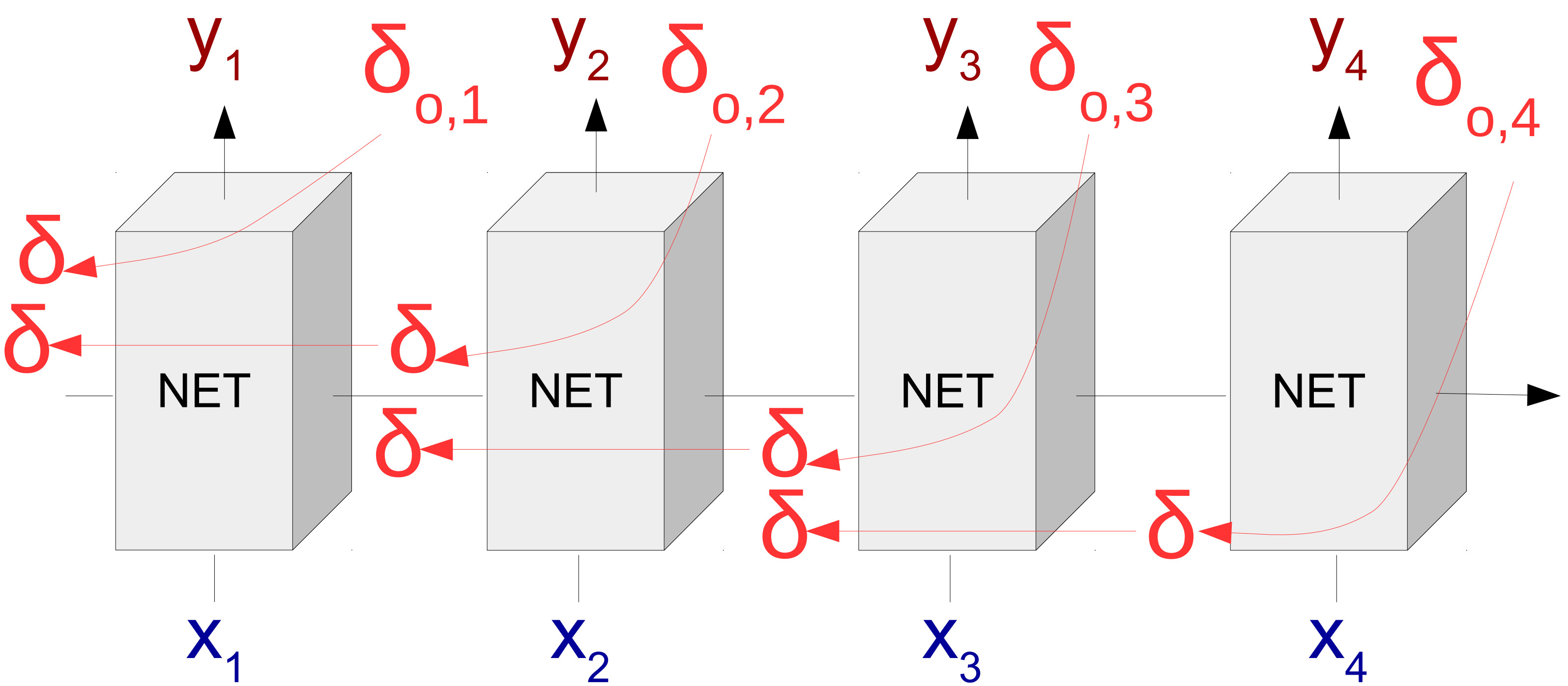


- We know the output errors δ_o
- Must use back-prop to find recurrent errors δ_r

How to Back-Propagate?

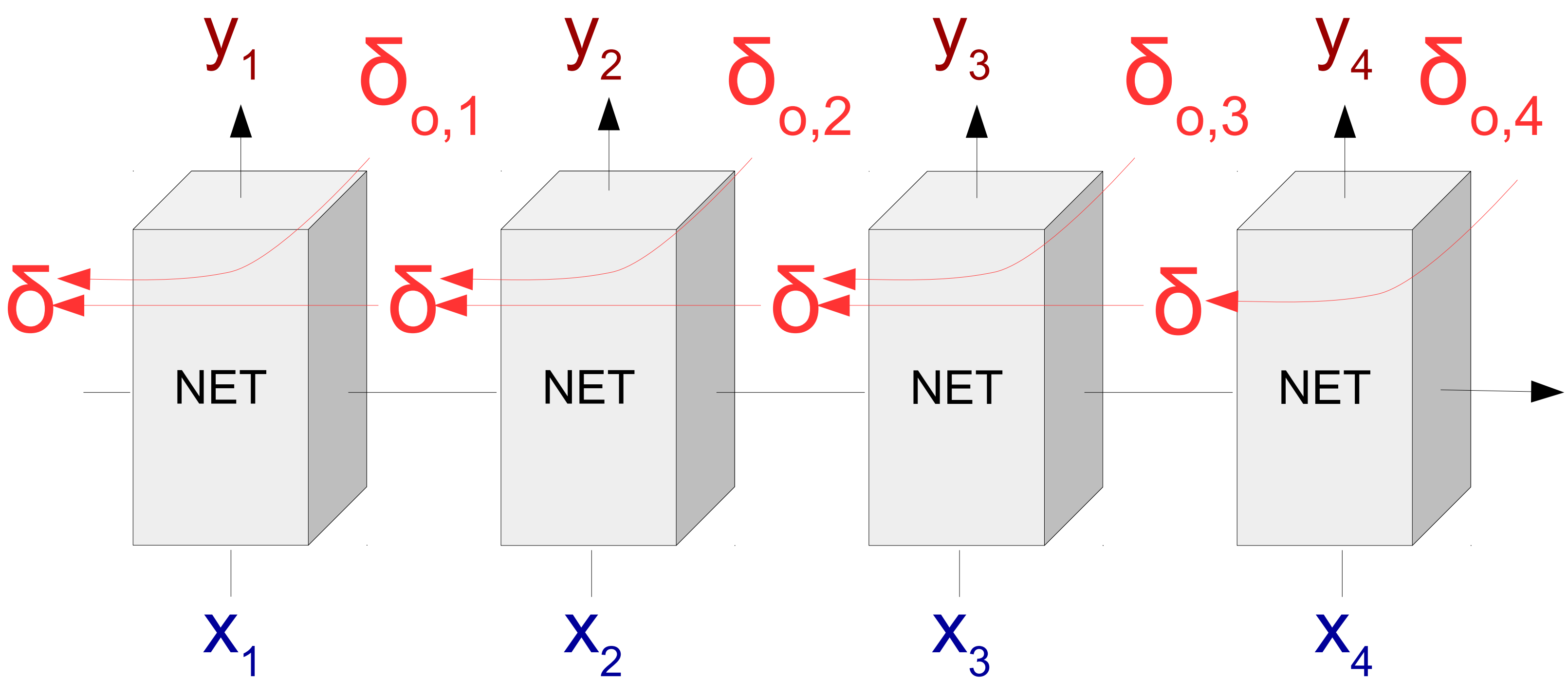
- Standard back propagation through time (BPTT)
  - For each δ_o, calculate n steps of δ_r
- Full gradient calculation
  - Use dynamic programming to calculate the whole sequence

Back Propagation through Time



- Use only one output error
- Stop after n steps (here, n=2)

Full Gradient Calculation

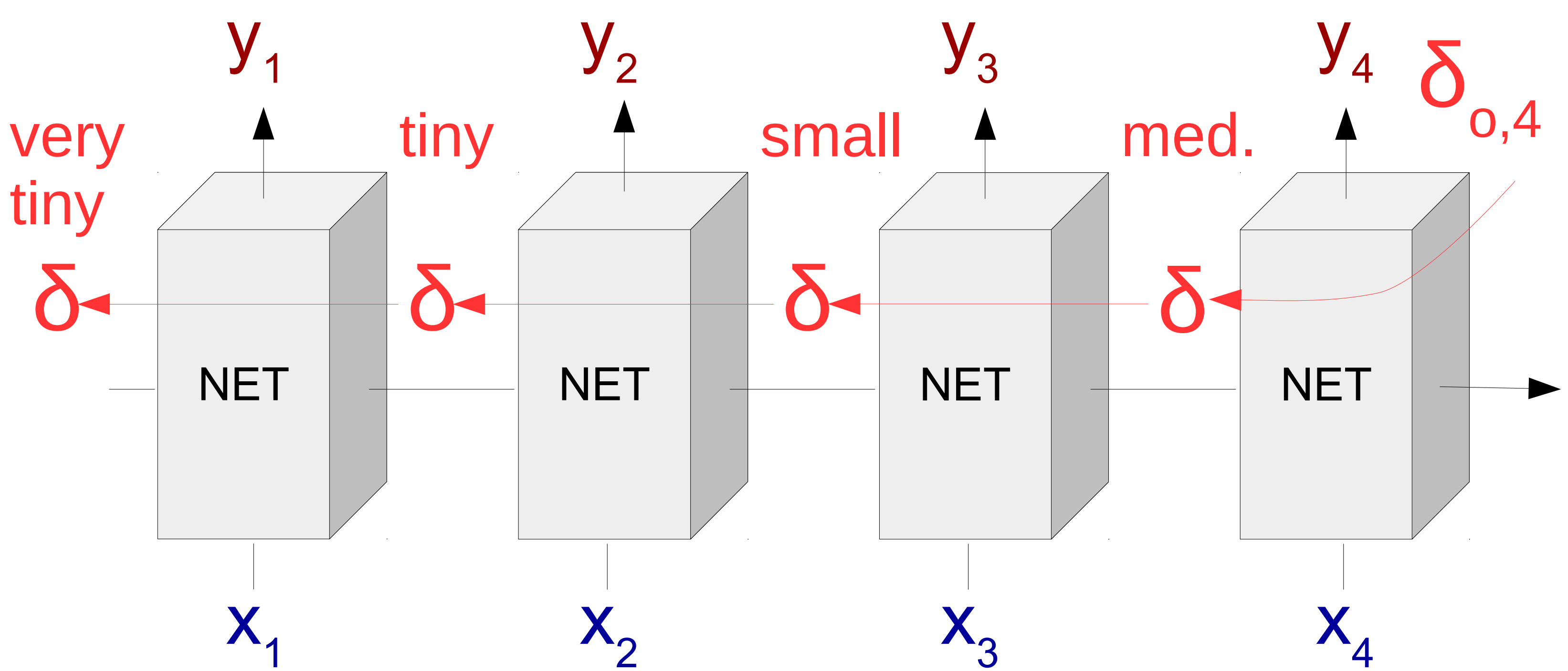


- First, calculate whole net result forward
- Then, calculate result backwards

BPTT? Full Gradient?

- Full gradient:
  - + Faster, no time limit
  - - Must save the result of the whole sequence in memory
- BPTT:
  - + Only remember the results in the past few steps
  - - Slower, less accurate for long dependencies

Vanishing Gradient in Neural Nets



- “Long Short Term Memory” is designed to solve this

RNN Full Gradient Calculation

gradient_rnn($w_{r,x}$, $w_{r,h}$, b_r, $w_{o,h}$, b_o, x, h, p, y')
    initialize $\Delta w_{r,x}$, $\Delta w_{r,h}$, Δb_r, $\Delta w_{o,h}$, Δb_o
    δ_r' = np.zeros(len(b_r)) # Error from the following time step
    for each time t **in** len(x)-1 .. 0**:**
        p' = create_one_hot(y'[t])
        δ_o' = p' – $p[t]$ # Output error
        $\Delta w_{o,h}$ += np.outer($h[t]$, δ_o'); Δb_o += δ_o' # Output gradient
        δ_r = np.dot(δ'_r, $w_{r,h}$) + np.dot(δ'_o, $w_{o,h}$)   # Backprop
        δ'_r = δ_r * (1 – $h[t]^2$)   # tanh gradient
        $\Delta w_{r,x}$ += np.outer($x[t]$, δ_r'); Δb_r += δ_r' # Hidden gradient
        if t != 0:
            $\Delta w_{r,h}$ += np.outer($h[t-1]$, δ_r');
    return $\Delta w_{r,x}$, $\Delta w_{r,h}$, Δb_r, $\Delta w_{o,h}$, Δb_o

Weight Update

update_weights($w_{r,x}$, $w_{r,h}$, b_r, $w_{o,h}$, b_o, $\Delta w_{r,x}$, $\Delta w_{r,h}$, Δb_r, $\Delta w_{o,h}$, Δb_o, λ)

$w_{r,x}$ += λ * $\Delta w_{r,x}$

$w_{r,h}$ += λ * $\Delta w_{r,h}$

b_r += λ * Δb_r

$w_{o,h}$ += λ * $\Delta w_{o,h}$

b_o += λ * Δb_o

Overall Training Algorithm

```
# Create features
create map x_ids, y_ids, array data
for each labeled pair x, y in the data
    add (create_ids(x, x_ids), create_ids(y, y_ids) ) to data
initialize net randomly

# Perform training
for I iterations
    for each labeled pair x, y' in the feat_lab
        h, p, y = forward_rnn(net, φ0)
        Δ= gradient_rnn(net, x, h, y')
        update_weights(net, Δ, λ)

print net to weight_file
print x_ids, y_ids to id_file
```

Exercise

Exercise

- Create an RNN for sequence labeling
- Training train-rnn and testing test-rnn
- Test: Same data as POS tagging
  - Input: test/05-{train,test}-input.txt
  - Reference: test/05-{train,test}-answer.txt
- Train a model with data/wiki-en-train.norm_pos and predict for data/wiki-en-test.norm
- Evaluate the POS performance, and compare with HMM: script/gradepos.pl data/wiki-en-test.pos my_answer.pos