Beginning Neural Networks [Assignment 6]

Paolo Gabriel

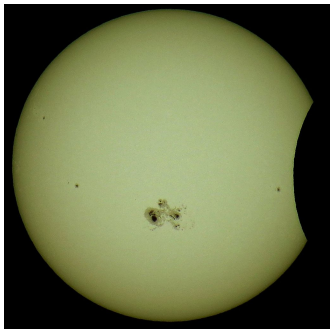
ECE 228, Spring 2018

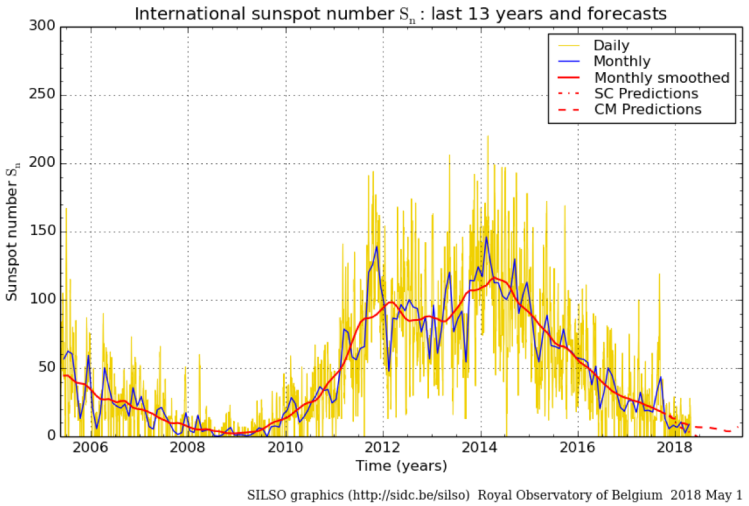
Objectives

- Data description
- Introduce a simple neural network model
- Implement getGradient() for sum-of-squares error function
- Optimize weights of NN using gradient descent

The data

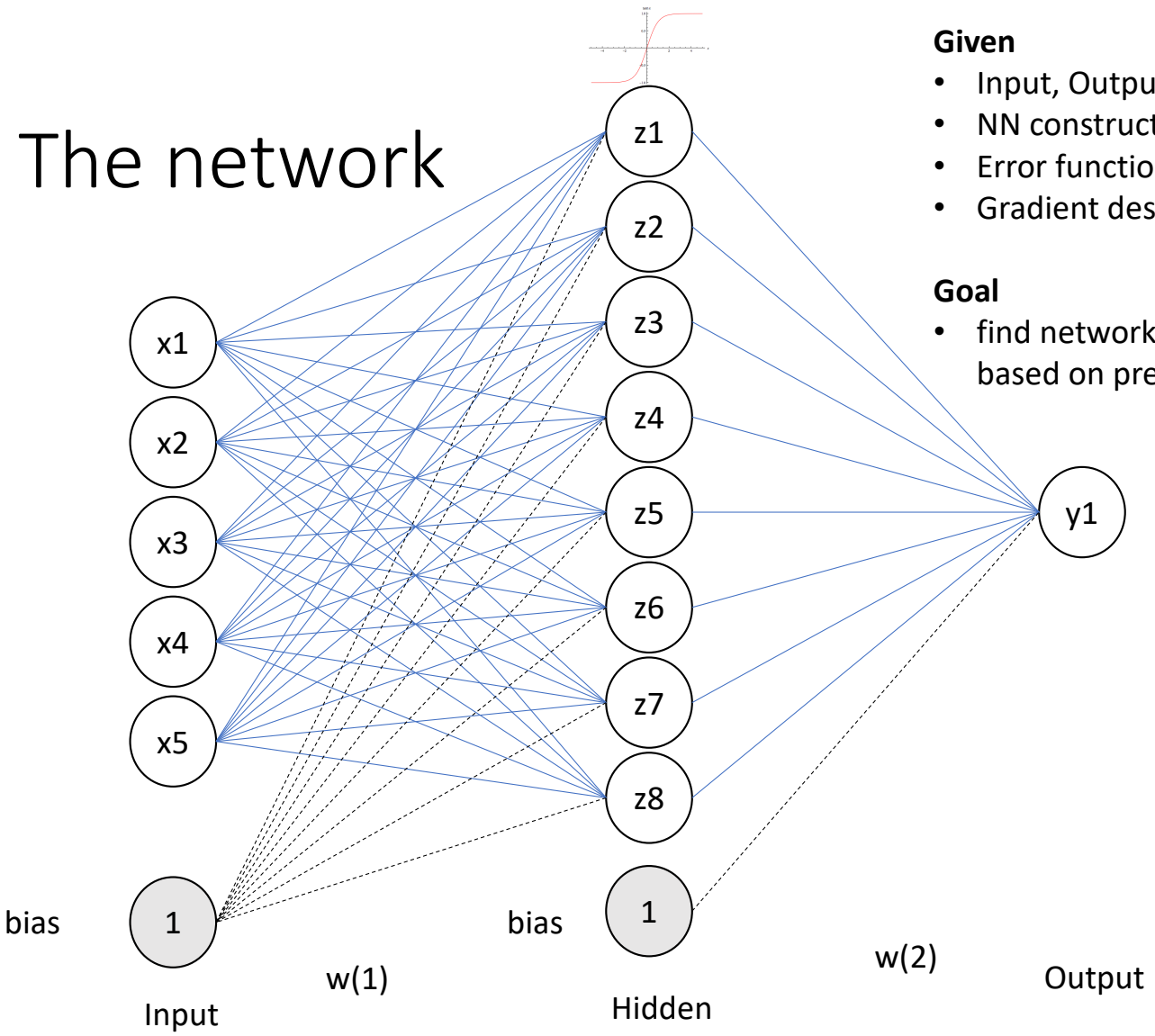
- Sunspot dataset (unit-scaled)
  - Time series of relative sunspot activity





Input					Target
x_1	x_2	x_3	x_4	x_5	x_6
…	…	…	…	…	…
x_(n-5)	x_(n-4)	x_(n-3)	x_(n-2)	x_(n-1)	x_n

The network



Given

- Input, Output (**x**, y)
- NN construction parameters (5x8x1)
- Error function parameters (sum-of-squares, α)
- Gradient descent parameters (stop condition, η)

Goal

- find network parameters that predict value based on previous 5 ordered observations

Finding network weights

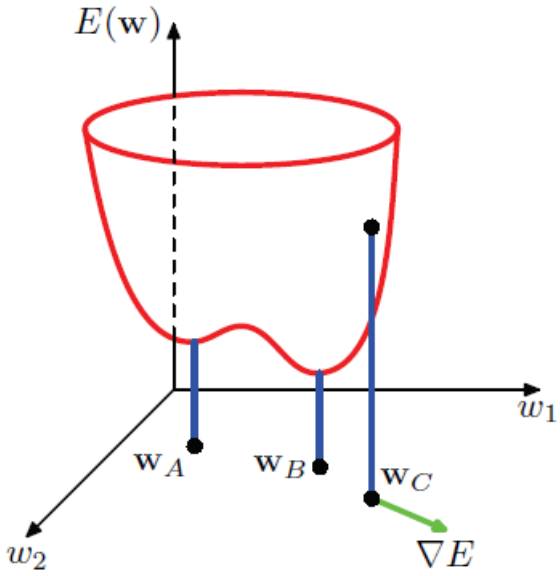
$$\mathbf{w}_{new} = \mathbf{w}_{old} + \triangle\mathbf{w}$$

- Specifically, gradient of network error function E(**w**)
- Find parameter changes that cause most change in error (make opposite changes)

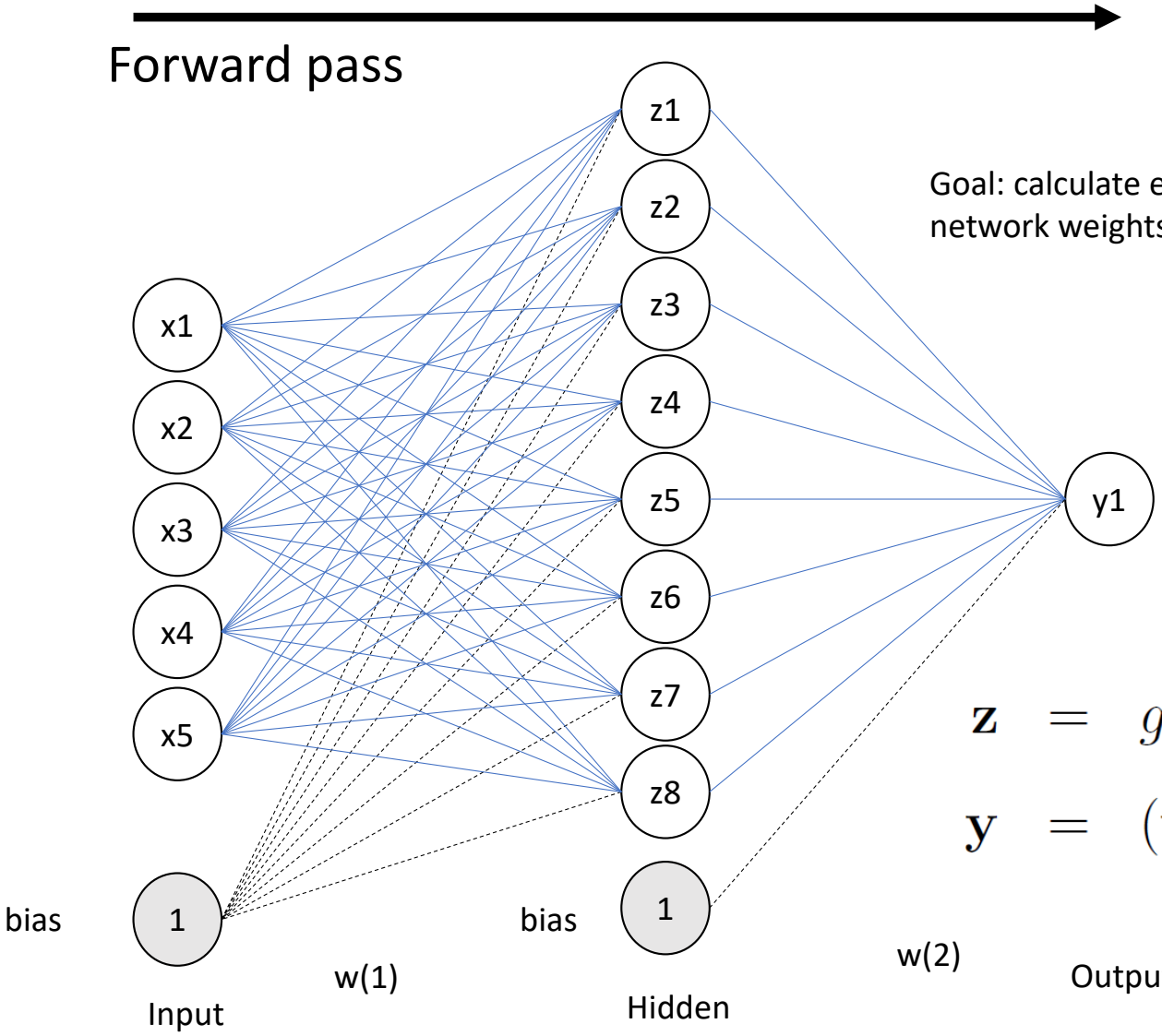
{Prediction error} Noise control

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\{y_k(\mathbf{x}^n;\mathbf{w}) - t_k^n\}^2 + \frac{1}{2}\alpha\mathbf{w}^2 .$$

$$\triangle\mathbf{w} = -\eta\frac{\partial E(\mathbf{w})}{\partial\mathbf{w}}$$



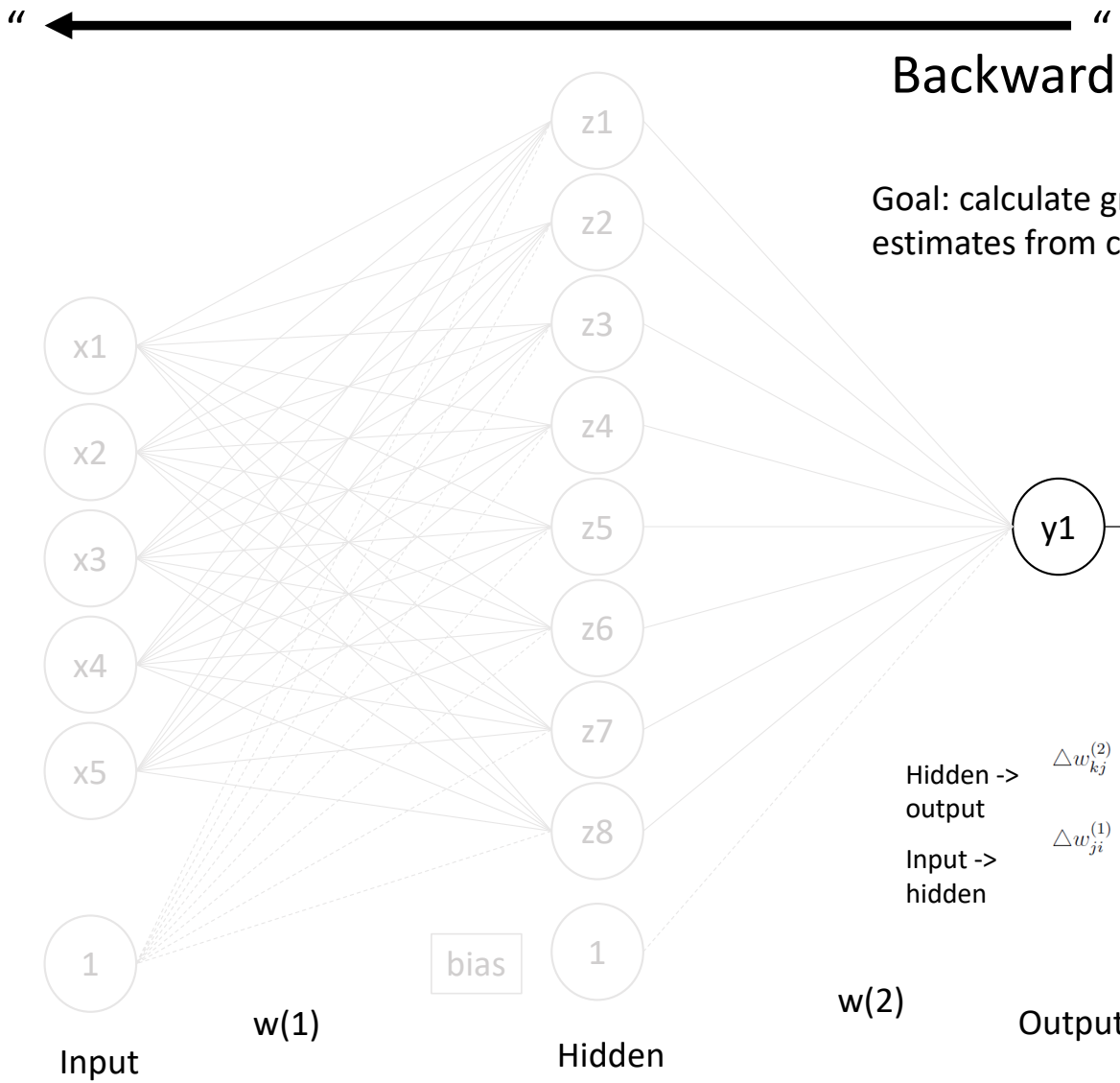
Forward pass



Goal: calculate estimated output using current network weights

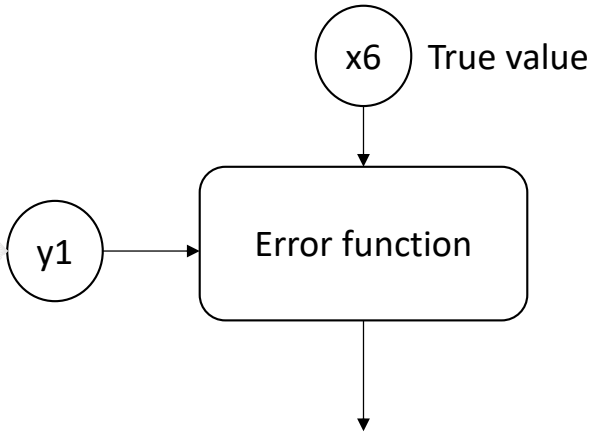
$$\mathbf{z} = g\left((\mathbf{w}^{(1)})^\top \mathbf{x}\right) \quad , \quad x_0 \equiv 1$$

$$\mathbf{y} = (\mathbf{w}^{(2)})^\top \mathbf{z} \quad , \quad z_0 \equiv 1$$



Backward pass

Goal: calculate gradient of error function based on estimates from current network weights



Hidden -> output

$$\triangle w_{kj}^{(2)} = -\eta \left(\sum_{n=1}^{N} (y_{nk} - t_{nk}) z_{nj} + \alpha w_{kj}^{(2)} \right)$$

Input -> hidden

$$\triangle w_{ji}^{(1)} = -\eta \left(\sum_{n=1}^{N} \left((1 - z_{nj}^2) \sum_{k=1}^{c} w_{kj}^{(2)} (y_{nk} - t_{nk}) \right) x_{ni} + \alpha w_{ji}^{(1)} \right)$$

[6.1] getGradient()

$$\mathbf{z} = g\left(\left(\mathbf{w}^{(1)}\right)^{\top}\mathbf{x}\right) \quad , \quad x_0 \equiv 1$$

$$\mathbf{y} = \left(\mathbf{w}^{(2)}\right)^{\top}\mathbf{z} \quad , \quad z_0 \equiv 1$$

Hidden -> output

$$\triangle w_{kj}^{(2)} = -\eta\left(\sum_{n=1}^{N}(y_{nk}-t_{nk})z_{nj} + \alpha w_{kj}^{(2)}\right)$$

Input -> hidden

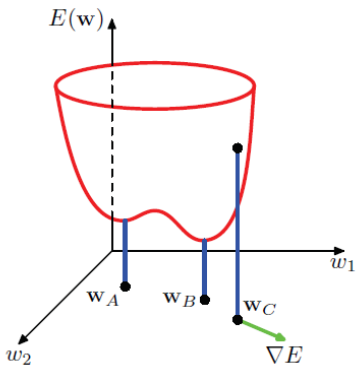
$$\triangle w_{ji}^{(1)} = -\eta\left(\sum_{n=1}^{N}\left((1-z_{nj}^2)\sum_{k=1}^{c} w_{kj}^{(2)}(y_{nk}-t_{nk})\right)x_{ni} + \alpha w_{ji}^{(1)}\right)$$

```
21  %%%%%%%%%%%%%%%%%%%%%
22  %%% FORWARD PASS %%%
23  %%%%%%%%%%%%%%%%%%%%%
24  %
25  % Propagate examples forward through network
26  % calculating all hidden- and output unit outputs
27  % NOTE: tanhf.m has been provided for g(.)
28  %
29  Inputs = [Inputs ones((exam),1)];
30  % Calculate hidden unit outputs for every example
31  z = tanhf(Inputs * Wi');
32
33  z = [z ones(exam,1)];
34  % Calculate (linear) output unit outputs for every example
35  y = z*Wo';
```

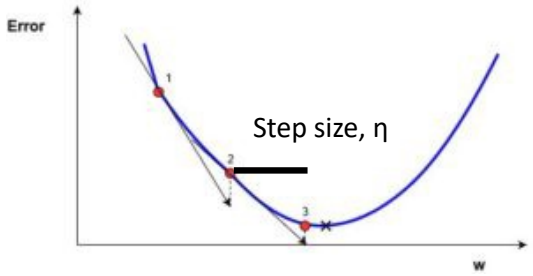
```
40  %%%%%%%%%%%%%%%%%%%%%
41  %%% BACKWARD PASS %%%
42  %%%%%%%%%%%%%%%%%%%%%
43
44  % Calculate derivative of
45  % by backpropagating the errors from the
46  % desired outputs
47
48  % Output unit deltas
49  dO = -(Targets - y);
50  % Hidden unit deltas
51  dI = (1-z.^2).*(dO*Wo);
52  dI = dI(:,1:end-1); %%%%%%%%!
53
54  % Partial derivatives for the output weights
55  dWo = z'*dO;
56  % Partial derivatives for the input weights
57  dWi = dI'*Inputs;
58  % Add derivatives of the weight decay term
59  dWo = dWo'+(alpha_o.*Wo);
60  dWi = dWi+(alpha_i.*Wi);
```

[6.2] Gradient Descent

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$



Single step



Multiple steps

```
32 -  rng('default');
33
34 -  range = 0.1;              % Initial weight range
35 -  eta=0.001;                % gradient descent parameter
36 -  max_iter=10;              % maximum number of iterations
37
38    % Initialize network weights
39 -  Wi = range * randn(Nh,Ni+1);
40 -  Wo = range * randn(No,Nh+1);
41
42 -  iter = 1;
43 -  while iter < max_iter
44 -    [dWi,dWo] =  getGradient(Wi,Wo,alpha_i,alpha_o,train_inp,train_tar);
45      % Update weights
46 -    Wi = Wi - eta * dWi;
47 -    Wo = Wo - eta * dWo;
48
49 -    iter = iter + 1;
50 -  end
```

Typical stopping criteria can be based on number of iterations or change in error

Final comments

- Bias added to end of arrays in this assignment… generally start/end ok
- tanh used to keep hidden node values near [-1,1]
- Things that affect neural network:
  - Parameters: step size, stop criteria, iteration range, initialization, network size and shape, error/optimization algorithms, hidden unit functions

1st	2nd	Name
–	–	Amoebe / simplex / Nelder-Mead
+	–	**Gradient descent**
+	–	Gradient descent with momentum
+	–	Natural gradient
+	–	Conjugate gradient algorithm
		— **Hestenes-Stiefel**
		— **Fletcher-Reeves**
		— **Polak-Ribiere**
		— Scaled conjugate gradient
+	–	Quasi-Newton
		— Davidson-Fletcher-Powell (DFP)
		— Rank-one-formula
		— Broyden-Fletcher-Goldfarb-Shanno (BFGS)
+	(+)	**Pseudo(-Gauss)-Newton**
+	(+)	Gauss-Newton
+	+	Levenberg-Marquardt
+	+	Newton(-Ralphson)

Table 1: Some optimization algorithms.