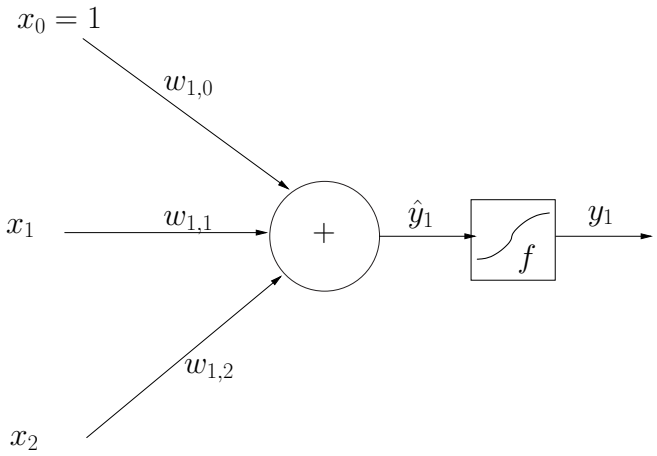
Neural Network

Presented by Daniel Boley

Outline

- Perceptron
- Nonlinearities
- Training
- Recent Developments

Perceptron Unit



Feed Forward (single unit) :

- $\hat{y} = \mathbf{w} \circ \mathbf{x}_i$
- $y = f(\hat{y})$
- $E = (y - y^{\text{desired}})^2$
  (or other distance measure)

Feed Forward (several units in parallel):

- $\hat{y}_j = \mathbf{w}_{j:} \circ \mathbf{x}_i$
- $y_j = f(\hat{y}_j)$
- $E = \sum_j (y_j - y_j^{\text{desired}})^2$

Back Propagation:

- $\frac{\mathrm{d}y_j}{\mathrm{d}\hat{y}_j} = f'(\hat{y}_j) = g(y_j)$
  fcn of ***output*** *of* f
- $\frac{\mathrm{d}\hat{y}_j}{\mathrm{d}w_{ji}} = x_i$
- $\frac{\mathrm{d}\hat{y}_j}{\mathrm{d}x_i} = w_{ji}$

Product Rule:

- $\frac{\mathrm{d}y_j}{\mathrm{d}w_{ji}} = x_i \cdot g(y_j)$
- $\frac{\mathrm{d}\hat{y}_j}{\mathrm{d}x_i} = w_{ji} \cdot g(y_j)$

Propagate derivative of error:

- $\frac{\mathrm{d}E}{\mathrm{d}w_{ji}} = x_i \cdot g(y_j) \cdot \frac{\mathrm{d}E}{\mathrm{d}y_j}$
- $\frac{\mathrm{d}E}{\mathrm{d}x_i} = w_{ji} \cdot g(y_j) \cdot \frac{\mathrm{d}E}{\mathrm{d}y_j}$

  ⋮ Propagate to the previous layer.

- $\frac{\mathrm{d}E}{\mathrm{d}\hat{x}_i} = g(x_i) \cdot w_{ji} \cdot g(y_j) \cdot \frac{\mathrm{d}E}{\mathrm{d}y_j}$

Nonlinearities

Alternative Activation Functions:

- Sigmoid: $f(\hat{y}) = 1/(1+\exp(-\hat{y}))$, $f'(\hat{y}) = g(y) = y(1-y)$.
- Tanh: $f(\hat{y}) = \frac{\exp(2\hat{y})-1}{\exp(2\hat{y})+1}$, $f'(\hat{y}) = g(y) = 1-y^2$.
- ReLU: $f(\hat{y}) = \max\{0, \hat{y}\}$, $f'(\hat{y}) = g(y) = \{1 \text{ if } y > 0;\ 0 \text{ o.w.}\}$.
- Leaky ReLU: $f(\hat{y}) = \max\{\alpha\hat{y}, \hat{y}\}$, $f'(\hat{y}) = g(y) = \{1 \text{ if } y > 0;\ \alpha \text{ o.w.}\}$ where $0 < \alpha << 1$.
- Softmax: $f_k(\hat{\mathbf{y}}) = \exp(\alpha\hat{y}_k)/\sum_\ell \exp(\alpha\hat{y}_\ell)$;
  $\frac{\mathrm{d}f_k(\hat{\mathbf{y}})}{\mathrm{d}y_j} = g_k(\mathbf{y}) = \alpha f_k(\mathbf{y})(\delta_{jk} - f_j(\mathbf{y}))$, $\delta_{jk} = \begin{Bmatrix} 1 \text{ if } j{=}k \\ 0 \text{ o.w.} \end{Bmatrix}$ *(Kronecker delta.)*

Alternative Error Functionals

- Quadratic: $E = \sum_\ell (z_\ell - t_\ell)^2$; $\quad \frac{\mathrm{d}E}{\mathrm{d}z_j} = 2(z_j - t_j)$.
- KL Divergence: $E = -\sum_\ell [t_\ell \log z_\ell + (1-t_\ell)\log(1-z_\ell)]$;
  $\frac{\mathrm{d}E}{\mathrm{d}z_j} = \frac{(z_j - t_j)}{z_j(1-z_j)}$.

Simple Multilayer Network

- Simple multilayer network with 2 fully connected layers (not counting inputs).
- Each layer consists of
  - V, W: linear combinations of its inputs, represented by matrices
  - f: an [elementwise] nonlinearity "activation function"
- Training:
  - Training samples presented to network one by one.
  - Outputs z_i compared to "true" desired labels t_i.
  - Weight matrices V, W updated to improve match $z_i \leftrightarrow t_i$.
  - Updates: propagate gradients back through the network, layer by layer.

input		hidden layer		outer layer
$\mathbf{x}$	$\rightarrow V \rightarrow$	$\hat{\mathbf{y}} \rightarrow f \rightarrow \mathbf{y}$	$\rightarrow W \rightarrow$	$\hat{\mathbf{z}} \rightarrow f \rightarrow \mathbf{z}$
$x_0 \equiv 1$		$\cdots \quad y_0 \equiv 1$	$\cdots$ artificial variables for biases	
x_1		$\hat{y}_1 \rightarrow f \rightarrow y_1$		$\hat{z}_1 \rightarrow f \rightarrow z_1$
x_2	$\searrow \quad \nearrow$	$\hat{y}_2 \rightarrow f \rightarrow y_2$	$\searrow \quad \nearrow$	$\hat{z}_2 \rightarrow f \rightarrow z_2$
$\cdot$	$\rightarrow V \rightarrow$	$\cdot \qquad \cdot$	$\rightarrow W \rightarrow$	$\cdot \qquad \cdot$
$\cdot$	$\nearrow \quad \searrow$	$\cdot \qquad \cdot$	$\nearrow \quad \searrow$	$\cdot \qquad \cdot$
x_m		$\hat{y}_n \rightarrow f \rightarrow y_n$		$\hat{z}_p \rightarrow f \rightarrow z_p$

Training

- **Construct** network.
- **Initialize** weights to random values.
- **For s** in sample training set:
  - **Feed s** to network computing all internal activation values.
  - **Compute** derivatives $\frac{\mathrm{d}E}{\mathrm{d}w_{ij}^k}$ for all layers k.
  - **Update** weights: w_{ij}^k += $-\frac{\mathrm{d}E}{\mathrm{d}w_{ij}^k} \circ$ learning_rate.

    On each sample, take a small step to slightly reduce the error

Training protoype code

```
Ts= true labels
rate=.01;

[d,n]=size(digits);  % training data
V=[...];W=[...];  random initialization
for epoch = 1:1000;
    for k=1:n
        [dE_dV,dE_dW,E,z,y]=deltaNN(V,W,digits(:,k),Ts(:,k));
        V = V - rate*dE_dV;
        W = W - rate*dE_dW;
    end;
end;
```

Training prototype derivative computation

```
function [dE_dV,dE_dW,E,z,y]=deltaNN2(V,W,x,t)
% x -- V --> y -- W --> z

% Forward Propagation
% First Layer:
yhat = V*[1;x];
y = tanh(yhat);

% Second Layer:
zhat = W*[1;y];
z = tanh(zhat);

% Measure output error:
E = sum((z-t).^2)/2;
```

```
% Backward Propagation - Last Layer
dE_dz    = z-t;
dz_dzhat = 1-z.^2;
dzhat_dW = [1,y'];

dE_dzhat = dE_dz .* dz_dzhat;
dE_dW    = dE_dzhat * dzhat_dW;

% Backward Propagation - Hidden Layer
dzhat_dy = W;
dy_dyhat = 1-y.^2;
dyhat_dV = [1,x'];

% Backward Propagation - Input Layer
dE_dy    = dzhat_dy' * dE_dzhat;
dE_dyhat = dE_dy(2:end) .* dy_dyhat;
dE_dV    = dE_dyhat * dyhat_dV;
```

Recent Developments

Most Popular New Concepts

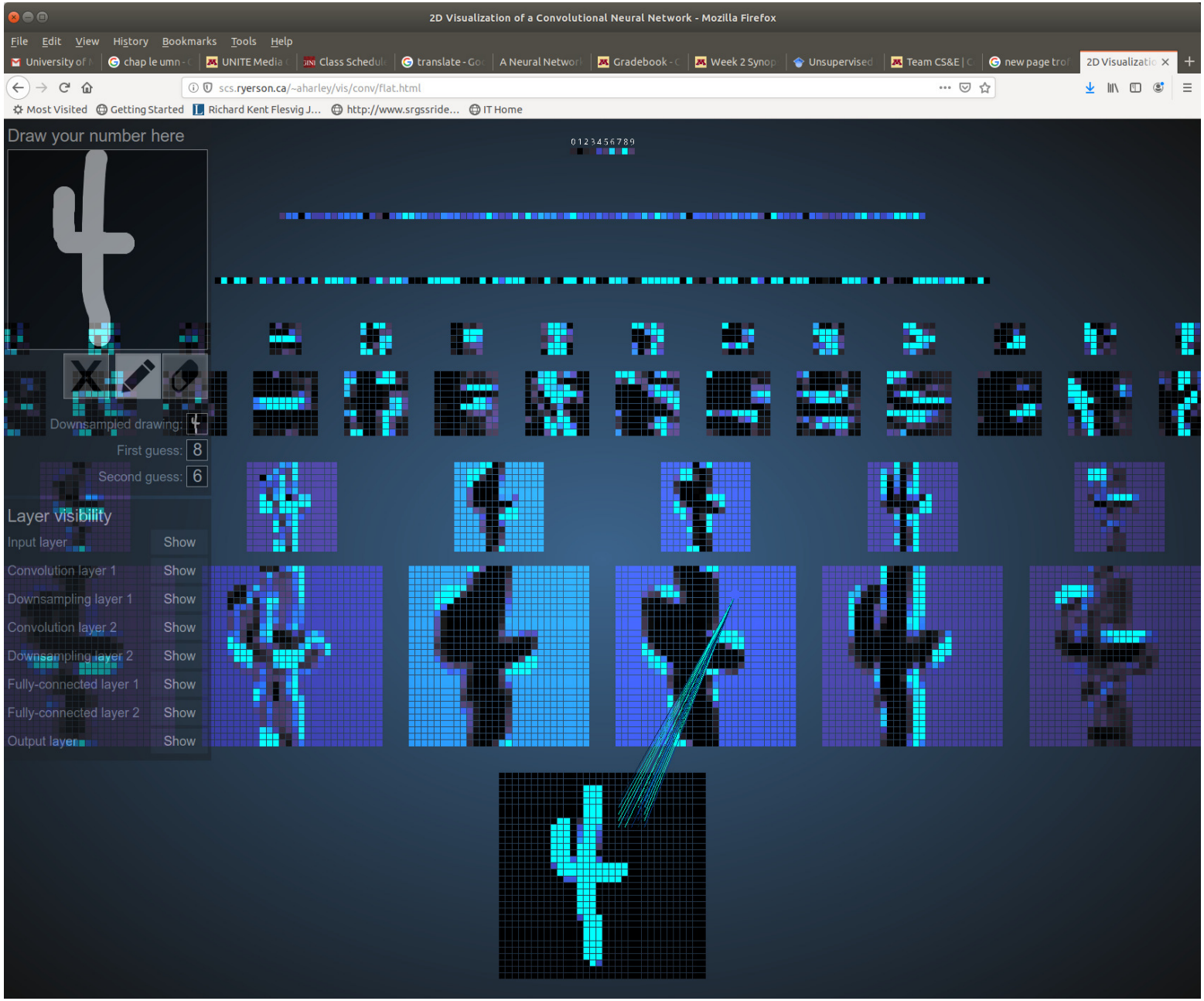
- Convolution
- Activation Functions: ReLU, Softmax
- Down Sampling
- Stochastic Gradient Descent
- Dropout (to reduce overfitting)

Novel Architectures

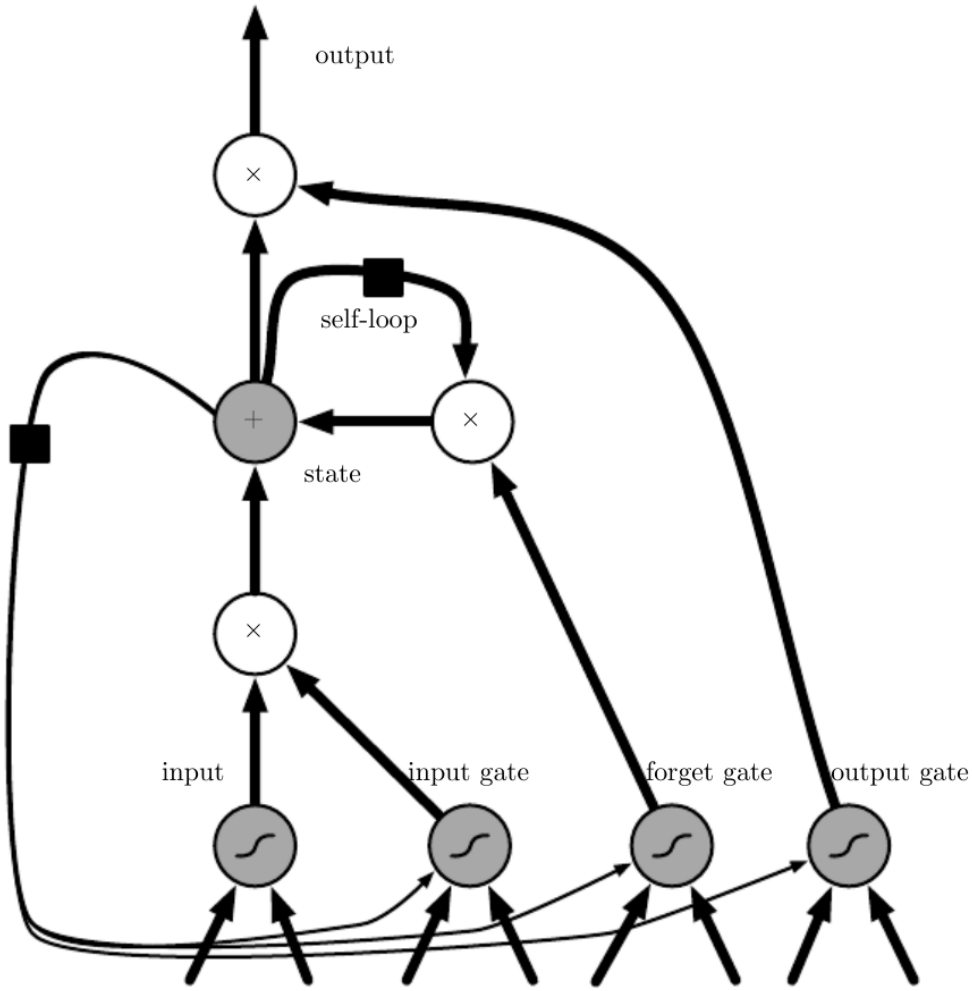
- Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM)
- Encoders/Decoders $\Longrightarrow$ Variational Autoencoders
- Adverserial Neural Networks: Generator + Discriminator
- Attention and Transformers (good for sequences)

Illustration

https://adamharley.com/nn_vis



Long Short-Term Memory



Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate allows it. The state unit has a linear self-loop whose weight is controlled by the forget gate. The output of the cell can be shut off by the output gate. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity. The state unit can also be used as an extra input to the gating units. The black square indicates a delay of a single time step.

From *Deep Learning* by I Goodfellow, Y Bengio, A Courville.

Attention, Transformers

Attention:

- Feed an entire sequence of words or tokens at once.
- Measure affinity between each token and every other token.
- Pass a weighted linear combination of tokens to next layer.
- Usually followed by a fully connected layer.

Transformers:

- Encoder/Decoder architecture.
- Encoder & Decoder each consists of a few Attention layers.

Performance:

- Can extract long distance relations better than RNNs.
- Lack of backward feedback loops means faster training.