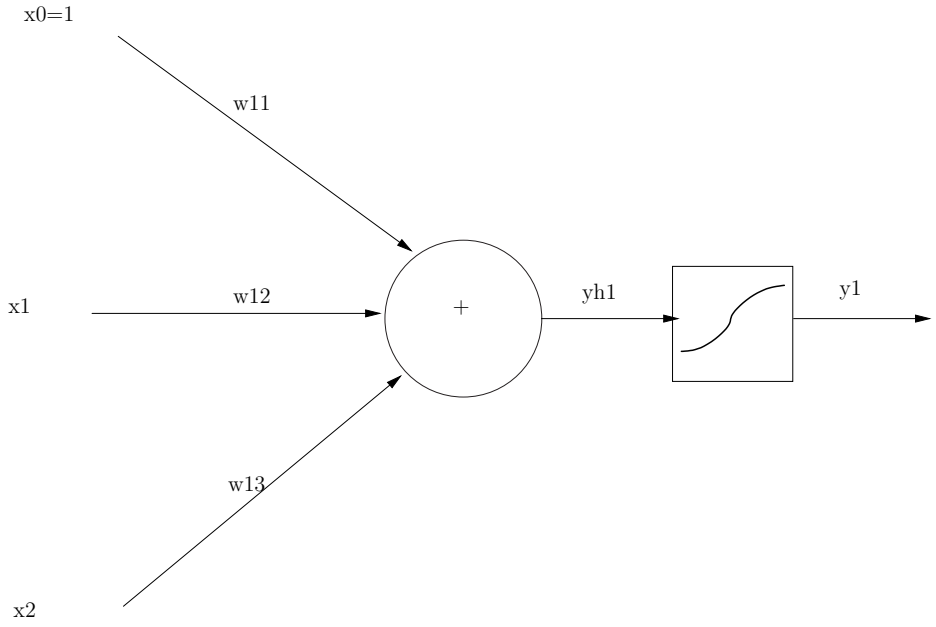
Neural Network

Presented by Daniel Boley

Outline

- Perceptron
- Nonlinearities
- Training
- Recent Developments

Perceptron Unit



Feed Forward:

- $\hat{y}_j = \mathbf{w}_{j:} \circ \mathbf{x}_{:i}$
- $y_j = f(\hat{y}_j)$

Back Propagation:

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

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

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: $f_k(\hat{\mathbf{y}}) = \exp(\alpha\hat{y}_k)/\sum_\ell \exp(\alpha\hat{y}_\ell)$;
  $\frac{\mathrm{d}f_k(\hat{\mathbf{y}})}{\mathrm{d}y_j} = g_k(\mathbf{y}) = \alpha f_k(\mathbf{y})(\delta_{jk} - f_j(\mathbf{y}))$, $\delta_{jk} =$ Kronecker delta.

Error Functionals

- Quadratic: $E = \sum_\ell (z_\ell - t_\ell)^2$; $\frac{\mathrm{d}E}{\mathrm{d}z_j} = 2(z_j - t_j)$.
- 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
  - g: an [elementwise] nonlinearity "activation function"
- Training:
  - Training samples presented to network one by one.
  - Outputs z_i compared to "true" desired labels t_i.
  - Weights 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			
$\mathbf{x}$	$\rightarrow V \rightarrow$		$\hat{\mathbf{y}}$	$\rightarrow g \rightarrow$	$\mathbf{y}$	$\rightarrow W \rightarrow$	$\hat{\mathbf{z}}$	$\rightarrow g \rightarrow$	$\mathbf{z}$	
$x_0 \equiv 1$			$\cdots$		$y_0 \equiv 1$	$\cdots$ artificial variables for biases				
x_1			$\hat{y}_1$	$\rightarrow g \rightarrow$	y_1		$\hat{z}_1$	$\rightarrow g \rightarrow$	z_1	
x_2	$\searrow$	$\nearrow$	$\hat{y}_2$	$\rightarrow g \rightarrow$	y_2	$\searrow$	$\nearrow$	$\hat{z}_2$	$\rightarrow g \rightarrow$	z_2
$\cdot$	$\rightarrow V \rightarrow$		$\cdot$		$\cdot$	$\rightarrow W \rightarrow$		$\cdot$		$\cdot$
$\cdot$	$\nearrow$	$\searrow$	$\cdot$		$\cdot$	$\nearrow$	$\searrow$	$\cdot$		$\cdot$
x_m			$\hat{y}_n$	$\rightarrow g \rightarrow$	y_n		$\hat{z}_p$	$\rightarrow g \rightarrow$	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.

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 protoype derivative computation

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

% Forward Propagation
yhat = V*[1;x];
y = tanh(yhat);
zhat = W*[1;y];
z = tanh(zhat);
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;
```

Modern Neural Networks

Popular Concepts

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

Novel Architectures

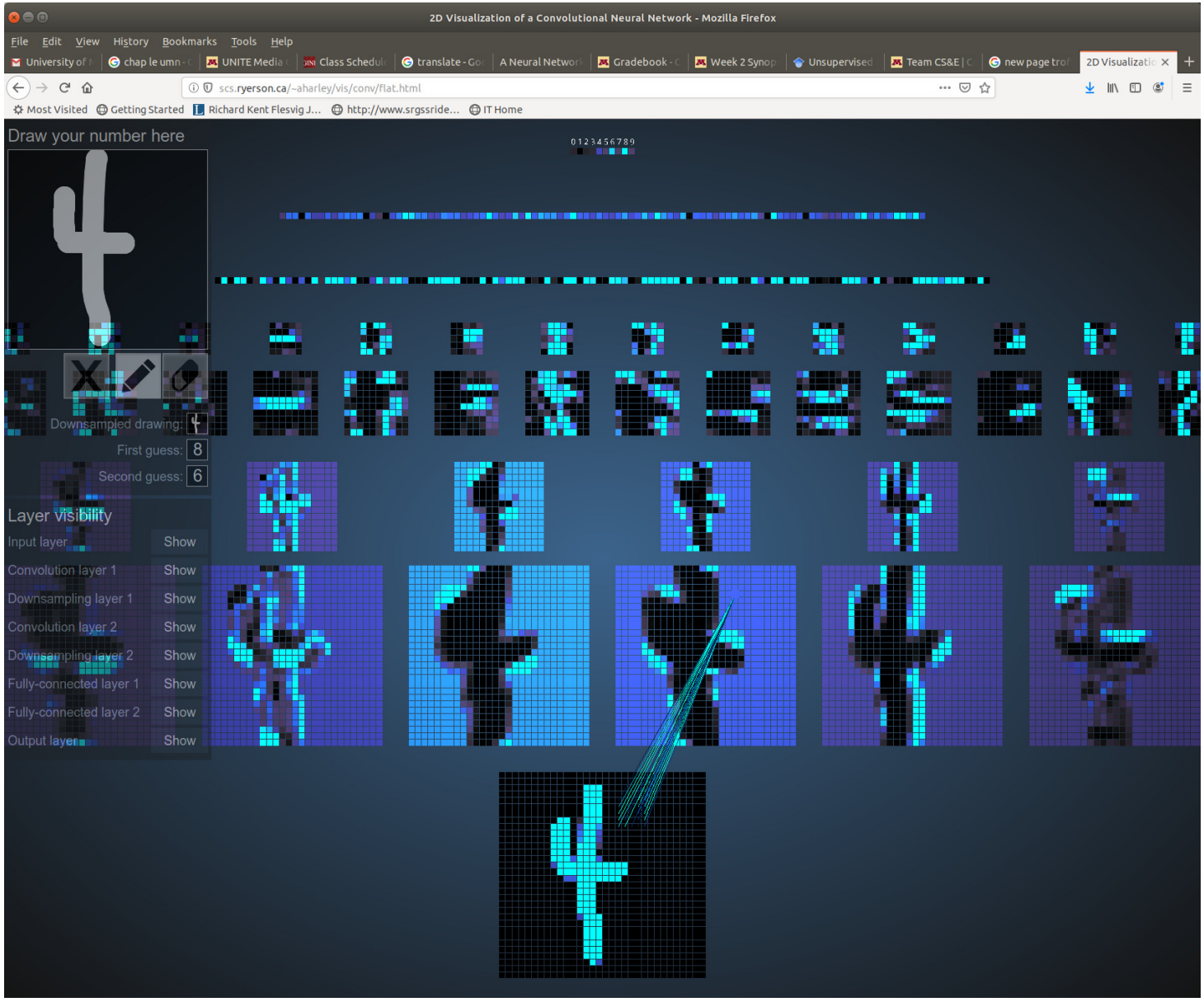
- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTM)
- Encoders/Decoders
- Adverserial Neural Networks: Generator + Discriminator (GANs)

Recent Developments

Popular Concepts

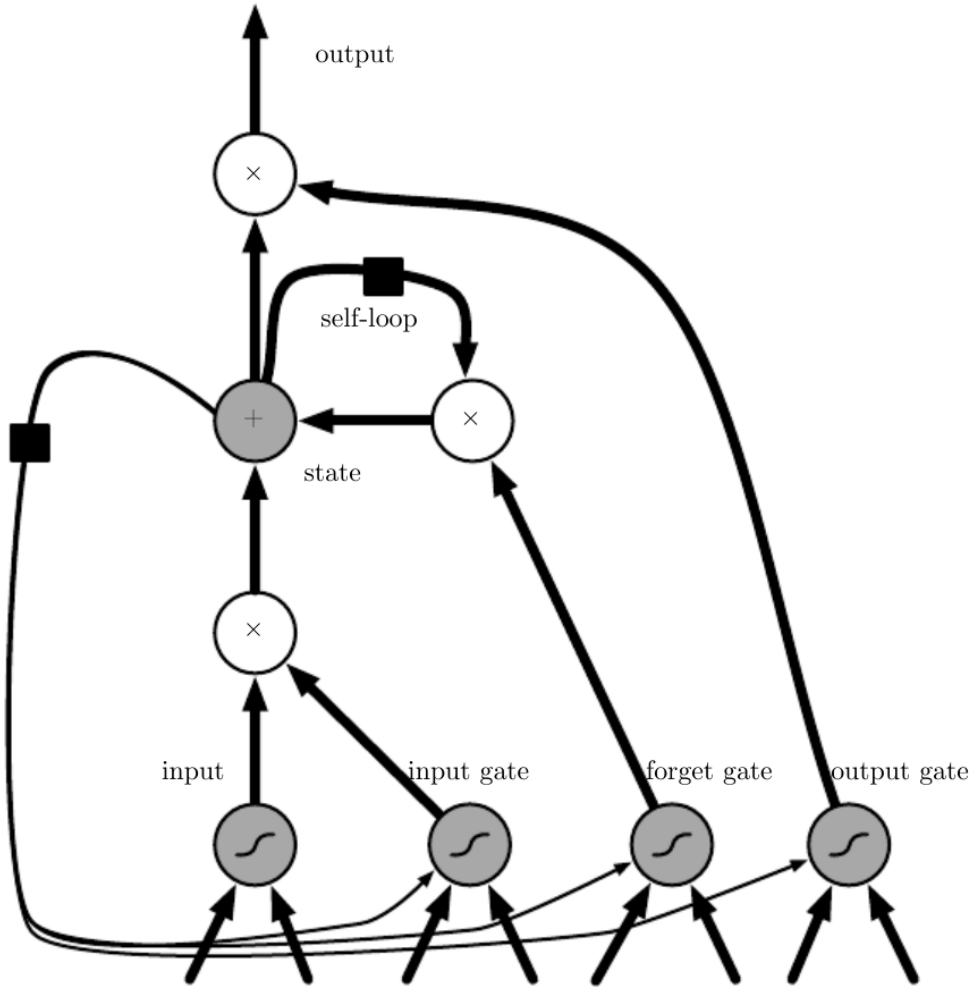
- Transformers – Attention
  - capture long-distance context
- Diffusion Models
  - stabilized generative models compared to GANs
- Contrastive Learning Sampling
  - merge 2 different modalities into a common latent space (e.g. text & images)
- Reinforcement Learning
  - Soft Actor Critic to broaden training data
- Teacher-student training
  - adapt large foundation models to similar but different data

Illustration



https://adamharley.com/nn_vis

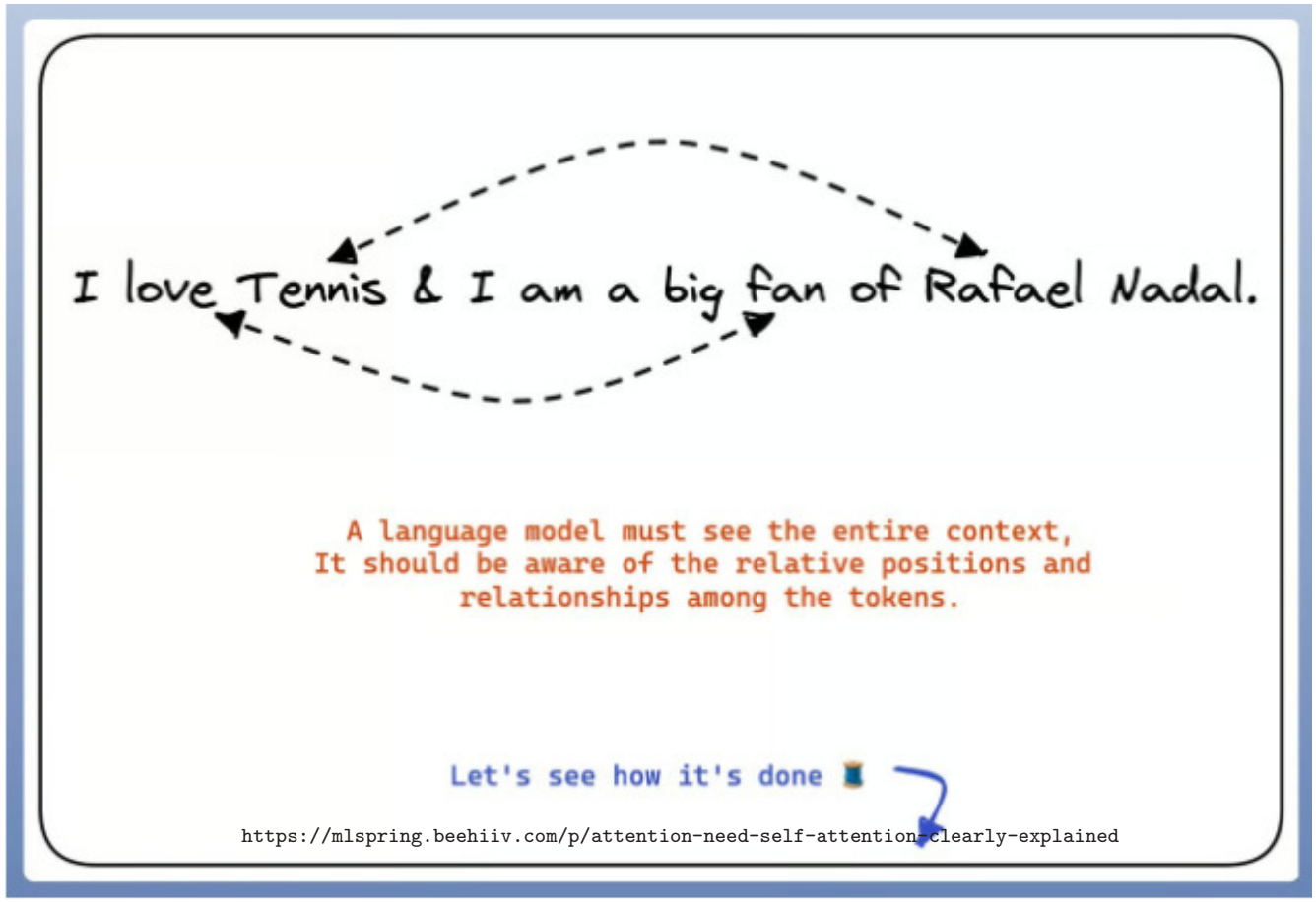
Long Short-Term Memory



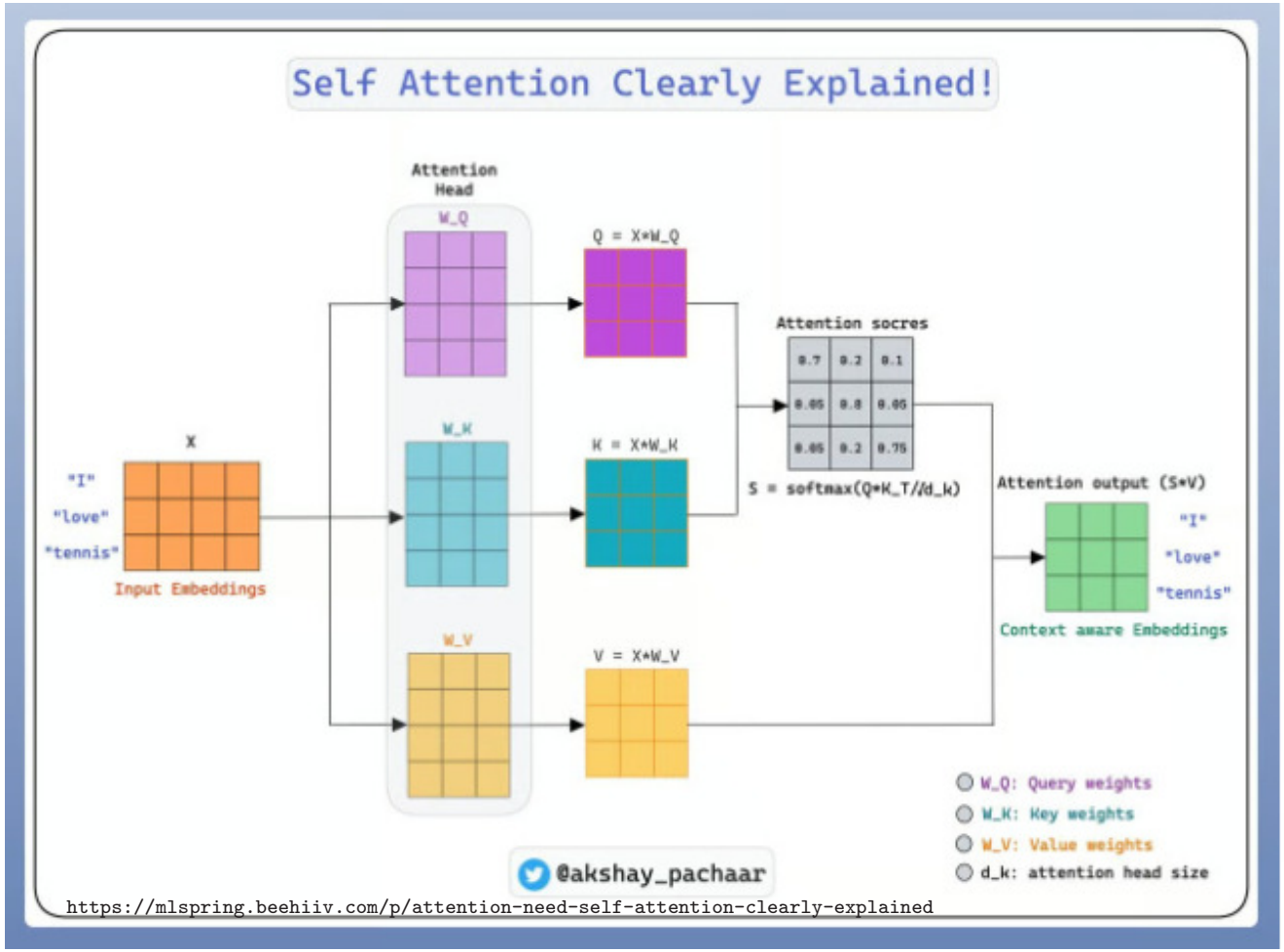
Block diagram of the LSTM recurrent network "cell." Cells are connected recurrently to each other, replacing the usual hidden units of ordinary recurrent networks. An input feature is computed with a regular artificial neuron unit. Its value can be accumulated into the state if the sigmoidal input gate 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 – Motivation



Attention – Transformers – Detail



Diffusion Models

11:09 Tue Aug 26 arxiv.org 78%

https://arxiv.org/pdf/2208.11970

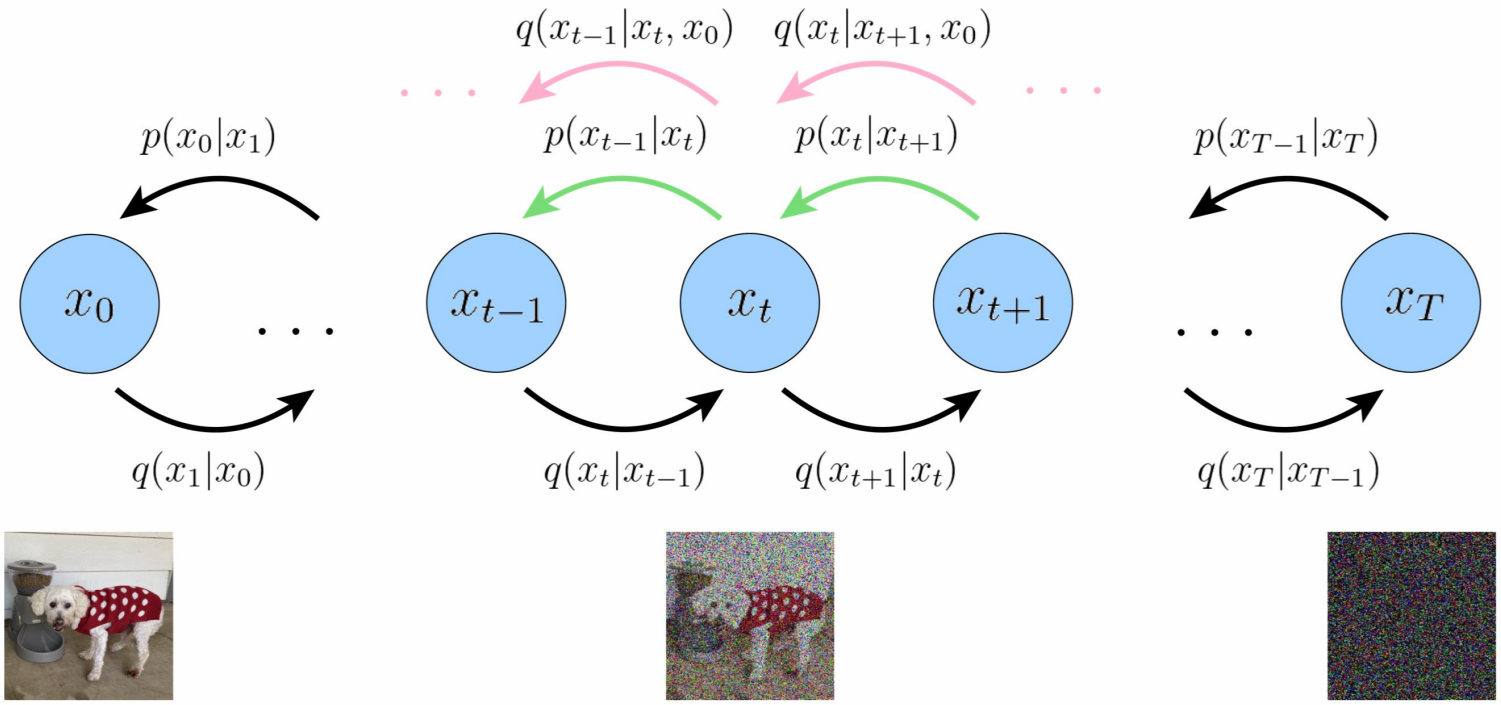
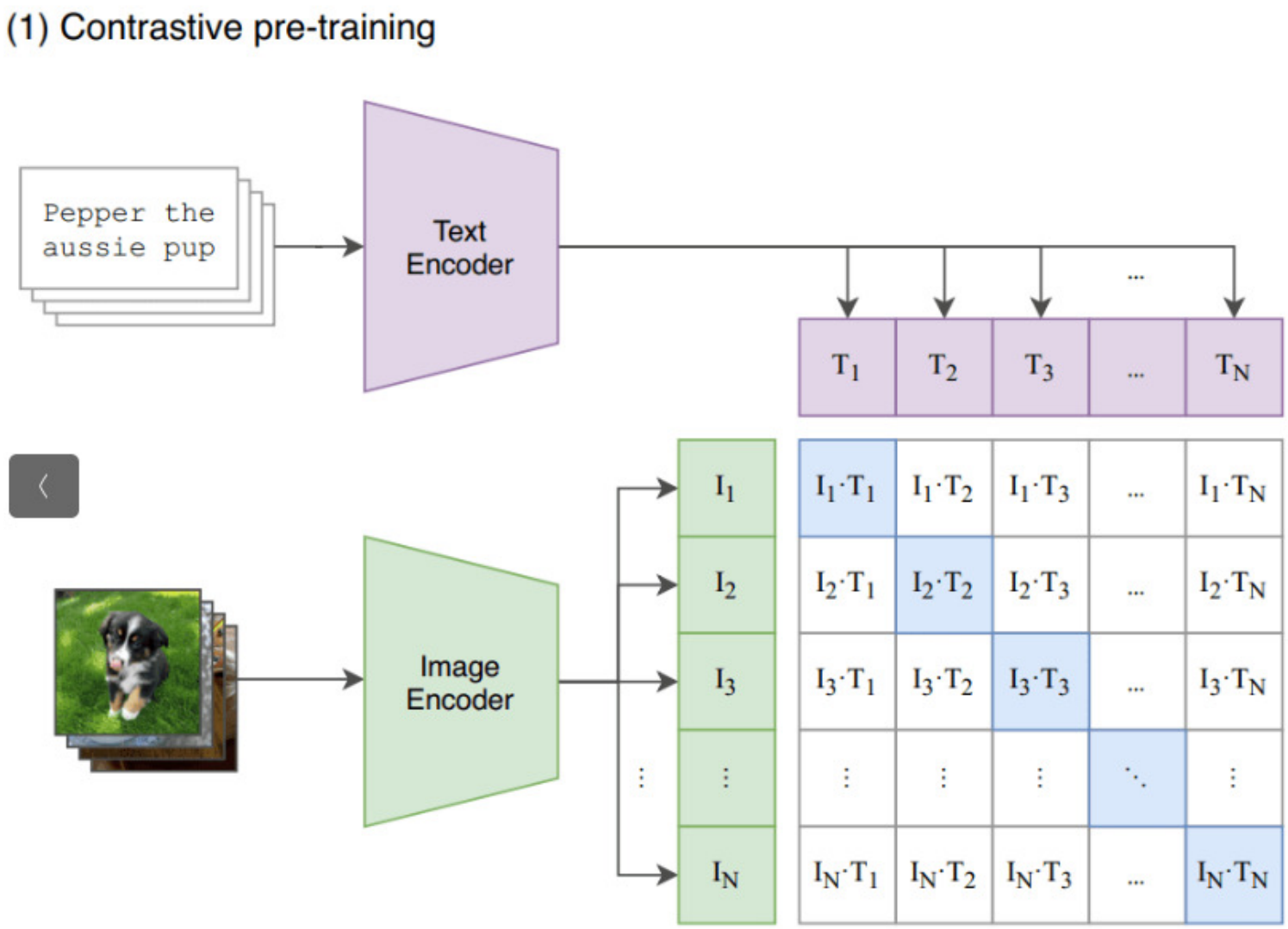


Figure 5: Depicted is an alternate, lower-variance method to optimize a VDM; we compute the form of ground-truth denoising step $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0)$ using Bayes rule, and minimize its KL Divergence with our approximate denoising step $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$. This is once again denoted visually by matching the distributions represented by the green arrows with those of the pink arrows. Artistic liberty is at play here; in the full picture, each pink arrow must also stem from $\boldsymbol{x}_0$, as it is also a conditioning term.

Merge Modalities: Contrastive Learning



Merge Modalities: CLIP