MxNet Gluon

Basics, Computer Vision, NLP (and even more NLP)
Part II (Neural Networks 101)

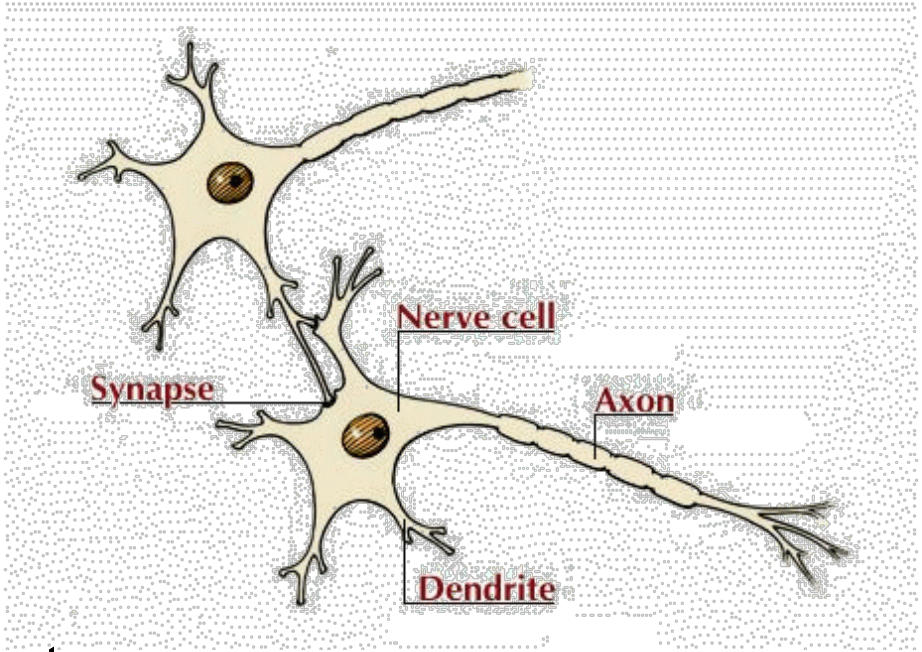
Leonard Lausen
Haibin Lin
Alex Smola

Outline

8:30-9:15	Installation and Basics (NDArray, AutoGrad, Libraries)
9:15-9:30	Neural Networks 101 (MLP, ConvNet, LSTM, Loss, SGD) - Part I
9:30-10:00	Break
10:00-10:30	Neural Networks 101 (MLP, ConvNet, LSTM, Loss, SGD) - Part II
10:30-11:00	Computer Vision 101 (Gluon CV)
11:00-11:30	Parallel and distributed training
11:30-12:00	Data I/O in NLP (and iterators)
12:00-13:30	Break
13:30-14:15	Embeddings
14:15-15:00	Language models (LM)
15:00-15:30	Sequence Generation from LM
15:30-16:00	Break
16:00-16:15	Sentiment analysis
16:15-17:00	Transformer Models & machine translation
17:00-17:30	Questions

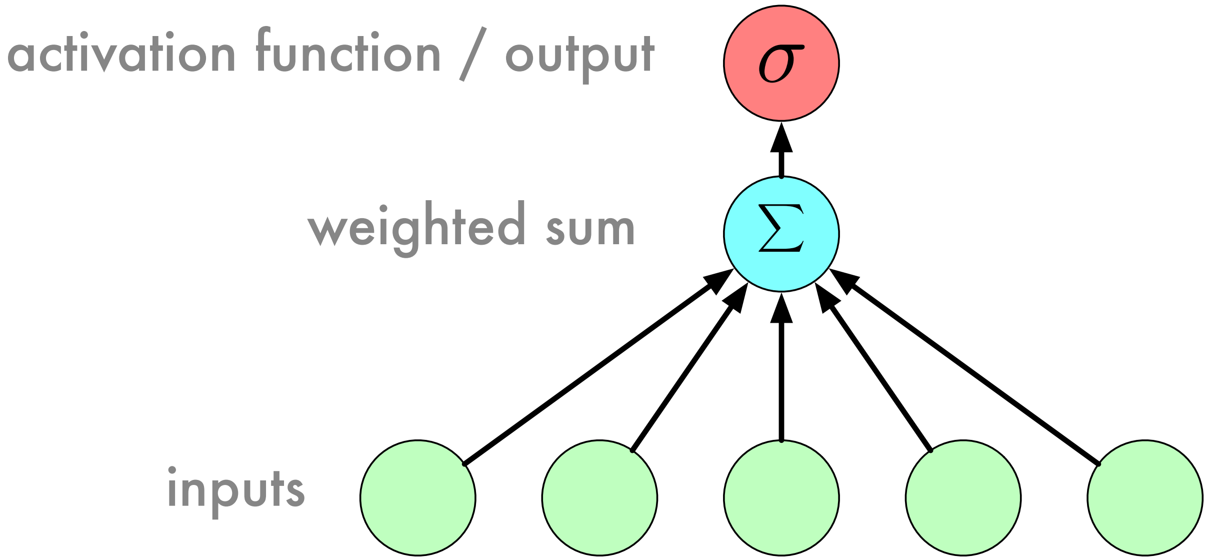
My first Neural Network

Neurons



- **Soma** (CPU)
  Cell body - combines signals
- **Dendrite** (input bus)
  Combines the inputs from several other nerve cells
- **Synapse** (interface)
  Interface and parameter store between neurons
- **Axon** (cable)
  May be up to 1m long and will transport the activation signal to neurons at different locations

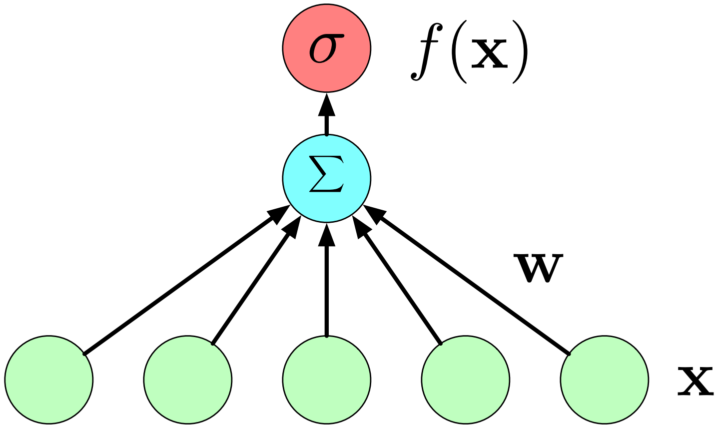
Neurons

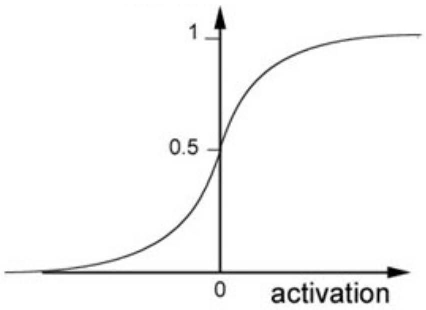


$$f(x) = \sum_i w_i x_i = \langle w, x \rangle$$

Components of a neural net

- **Inputs**
  Data vector $\mathbf{x}$
- **Targets**
  What we want model to output
- **Architecture**
  Connectivity pattern and activation functions
- **Learning rule**
  Updates the parameters





$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

MxNet Code

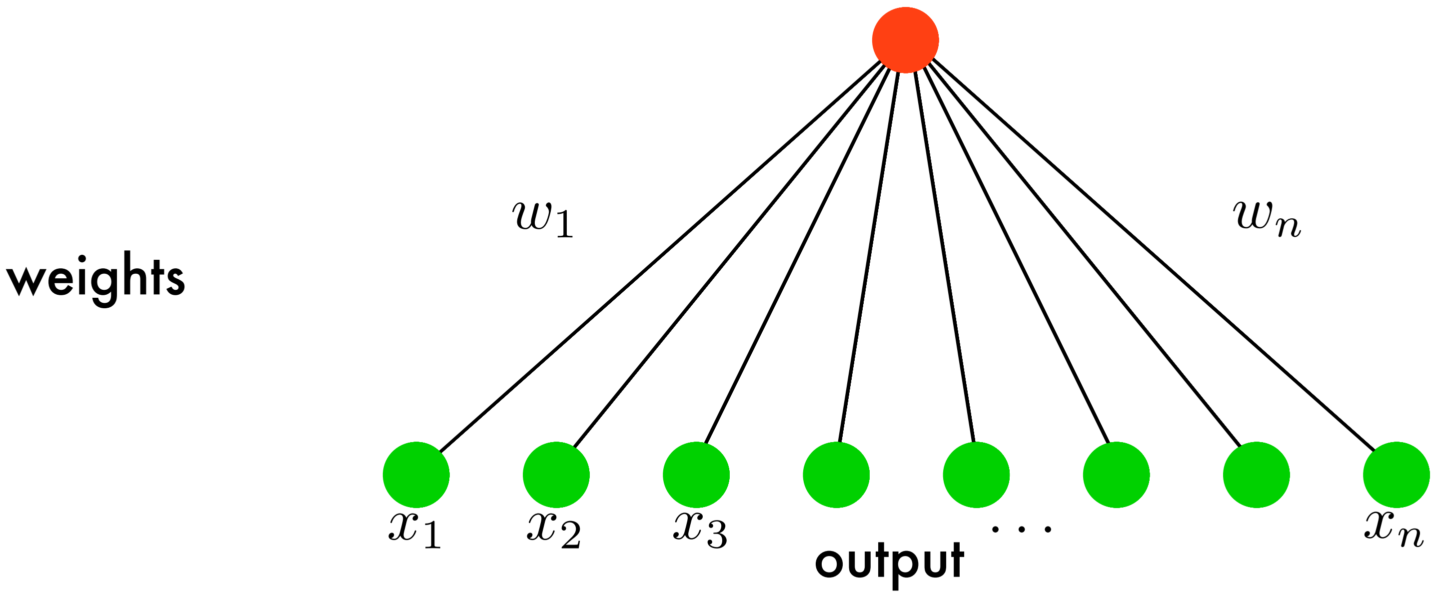
```
import mxnet as mx
from mxnet import gluon

net = gluon.nn.Sequential()
net.add(gluon.nn.Dense(1))
loss = gluon.loss.SoftmaxCrossEntropyLoss()

+ iterator over data (and loader)
+ training loop
```

create a new model

add fully-connected layer

instantiate a loss

Linear models

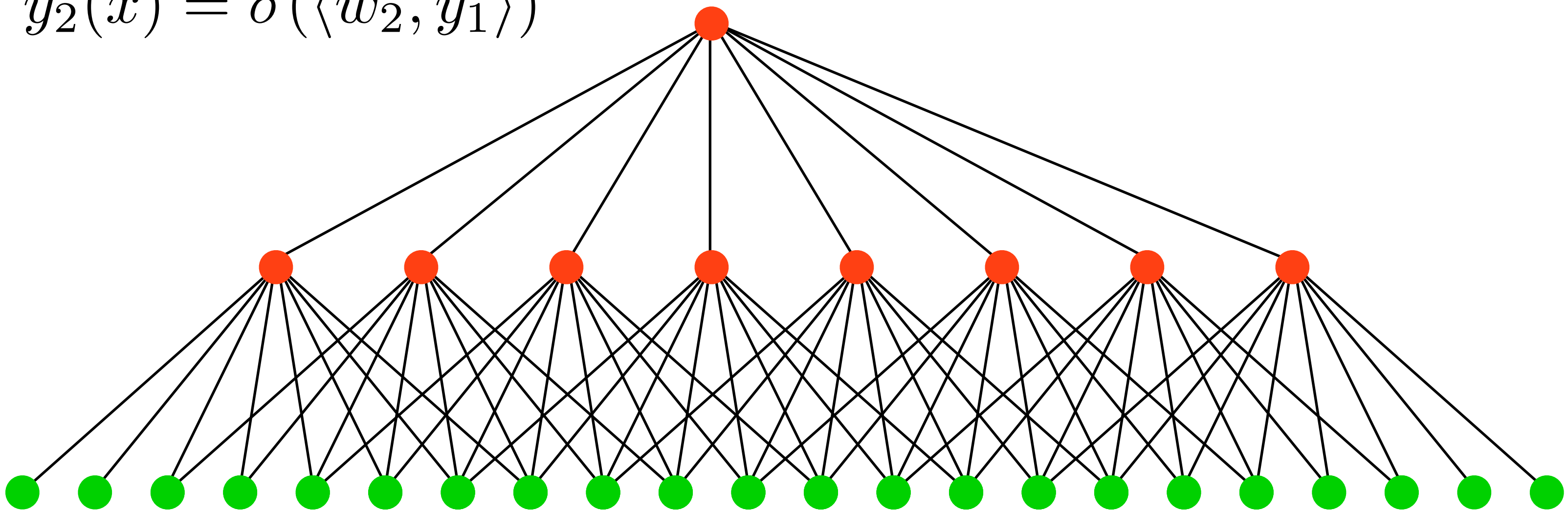


$$y(x) = \sigma(\langle w, x \rangle)$$

Multilayer Perceptron

$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$

$$y_2(x) = \sigma(\langle w_2, y_1 \rangle)$$

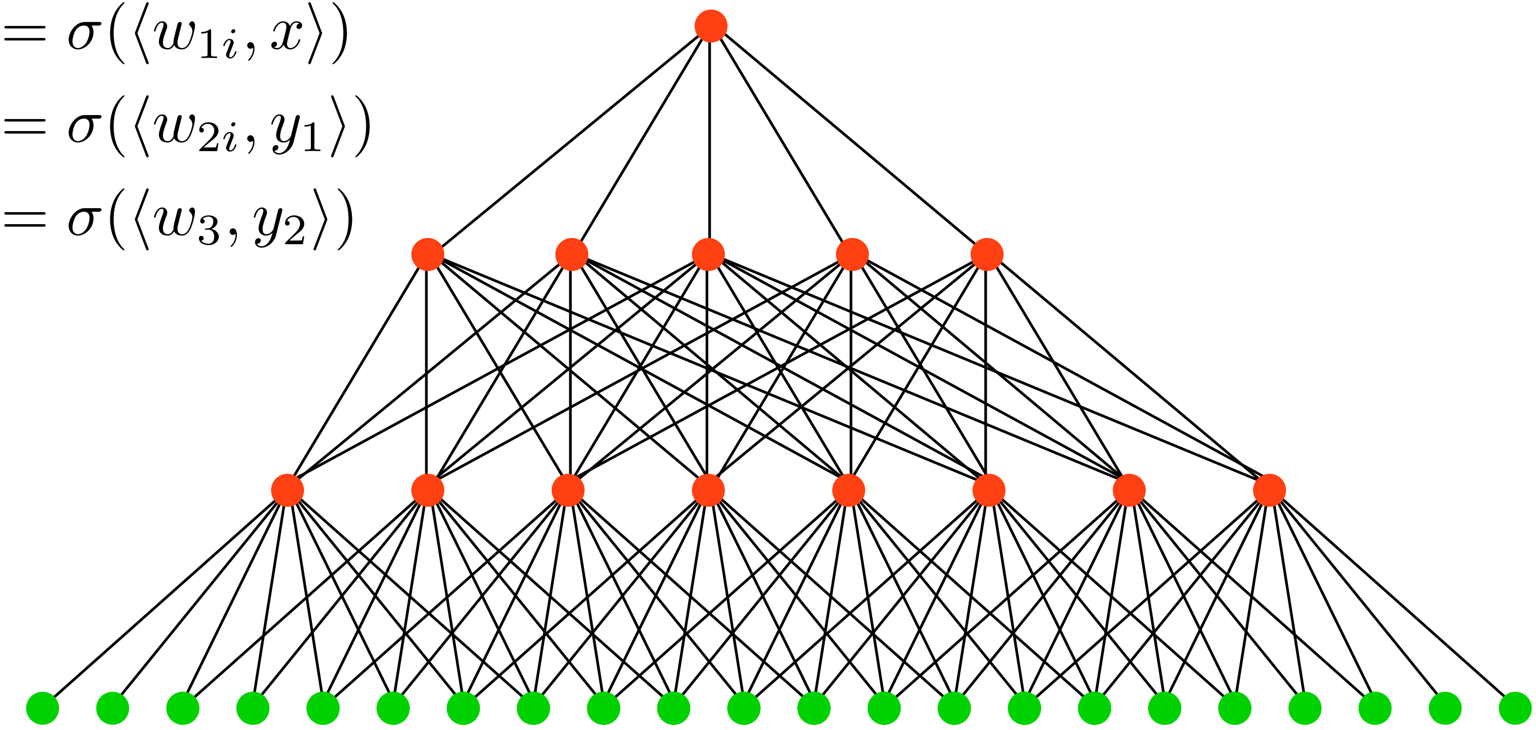


Multilayer Perceptron

$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$

$$y_{2i}(x) = \sigma(\langle w_{2i}, y_1 \rangle)$$

$$y_3(x) = \sigma(\langle w_3, y_2 \rangle)$$



Multilayer Perceptron Training

- **Layer Representation**
  (each layer performs a transformation)

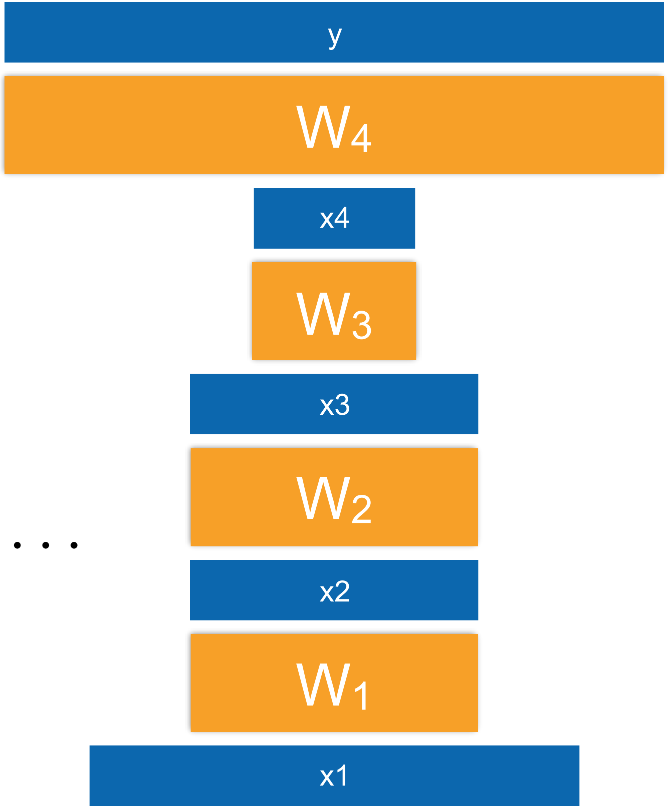
  $$x_{i+1} = f_i(x_i, w_i)$$

- **Recursive Definition**

  $$x_{i+1} = f_i(x_i, w_i) = f_i(f_{i-1}(x_{i-1}, w_{i-1}))\ldots$$

- **Use the chain rule for gradients**

  $$\partial_x f(g(x)) = f'(g(x))g'(x)$$



amazon web services

Backpropagation

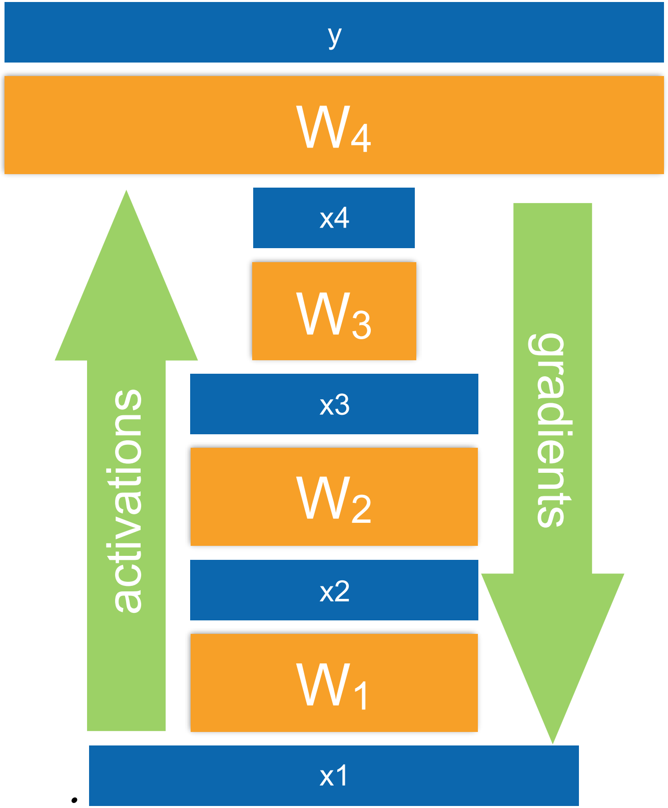
- **Layer Representation**
  (each layer performs a transformation)
  $x_{i+1} = f_i(x_i, w_i)$
- **Gradient**

$$\partial_{w_1} x_2(x) = \partial_{w_1} f_1(x_1, w_1)$$
$$\partial_{w_1} x_3(x) = \partial_{x_2} f_2(x_2, w_2) \partial_{w_1} x_2(x)$$
$$\ldots$$
$$\partial_{w_i} x_{i+1}(x) = \partial_{w_i} f_i(x_i, w_i)$$
$$\partial_{w_i} x_{j+1}(x) = \partial_{x_j} f_j(x_j, w_j) \partial_{w_i} x_j(x) \text{ if } i > j$$

Backpropagation

- **Layer Representation**
  (each layer performs a transformation)
  $$x_{i+1} = f_i(x_i, w_i)$$
- **Gradient**

$$\partial_{w_1} x_2(x) = \partial_{w_1} f_1(x_1, w_1)$$
$$\partial_{w_1} x_3(x) = \partial_{x_2} f_2(x_2, w_2) \partial_{w_1} x_2(x)$$
$$\dots$$
$$\partial_{w_i} x_{i+1}(x) = \partial_{w_i} f_i(x_i, w_i)$$
$$\partial_{w_i} x_{j+1}(x) = \partial_{x_j} f_j(x_j, w_j) \partial_{w_i} x_j(x) \text{ if } i > j$$



amazon
web services

That was complicated!
MXNet takes care of this for you …

```
import mxnet as mx
from mxnet import gluon, autograd

net = gluon.nn.Sequential()
with net.name_scope():
   net.add(gluon.Dense(128, activation=‘relu'))
   net.add(gluon.Dense(64, activation='relu'))
   net.add(gluon.Dense(1))

cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()

with autograd.record():
    output = net(data)
    loss = cross_entropy(output, label)
loss.backward()
```

second layer

nonlinearity

loss function

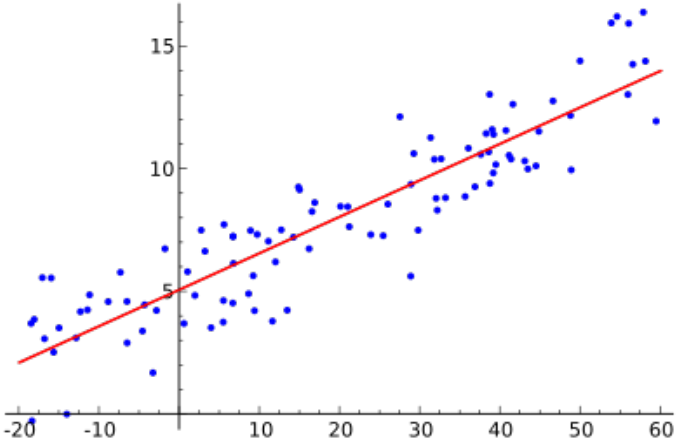
forward pass

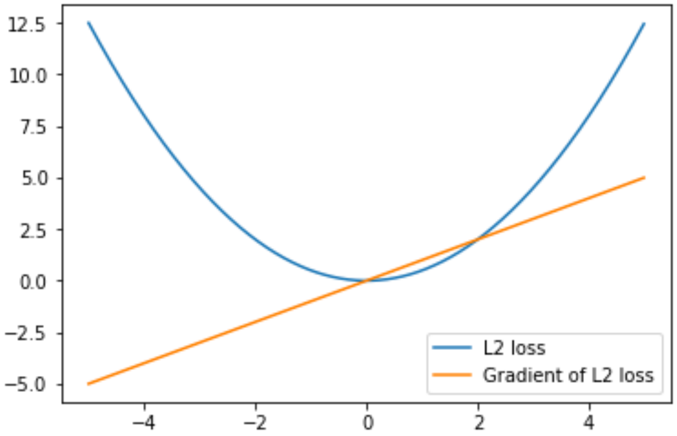
backward pass

Loss functions
(aka - telling the network what to do)

Linear Regression (aka How Much)

- Model $$f(x) = \langle w, x \rangle + b$$
- Loss $$l(y, f(x)) = \frac{1}{2}(y - f(x))^2$$



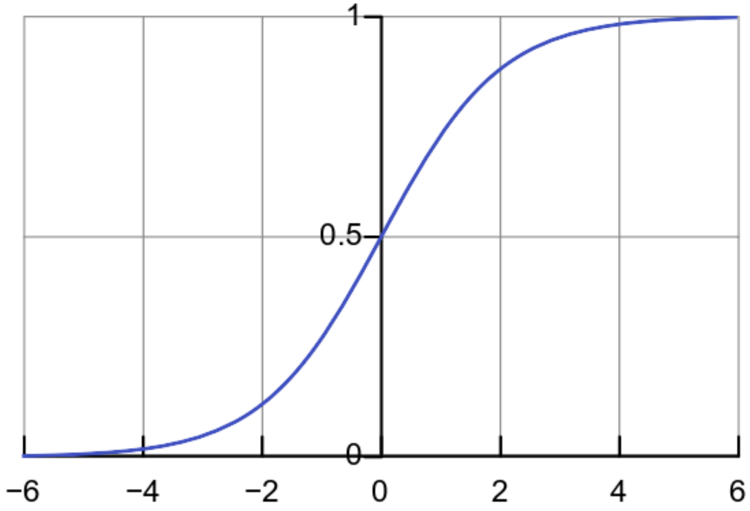


amazon web services

Logistic Regression — for “yes/no?” problems

- Model (binary classification)

$$p(y|f(x)) = \frac{1}{1 + \exp(-yf(x))}$$



- Loss (negative log likelihood)

$$-\log p(y|f(x)) = \log\left(1 + \exp(-yf(x))\right)$$

$$\partial_f - \log p(y|f(x)) = \frac{-y}{1 + \exp(yf(x))}$$

Softmax (aka Many Classes)

- We have k outputs
- We want our output layer to assign *probabilities* to each output
- Exponential family model ensures that our output is a valid multinomial distribution

$$p(y|z) = \frac{\exp(z_y)}{\sum_{y'} \exp(z_{y'})}$$

- As before, loss is negative log-likelihood

$$-\log p(y|z) = \log \sum_{y'} \exp(z_{y'}) - z_y$$

Convolutional Networks

Convolutional Layers

- **Feature Locality**
  Relevant information only in neighborhood of pixel

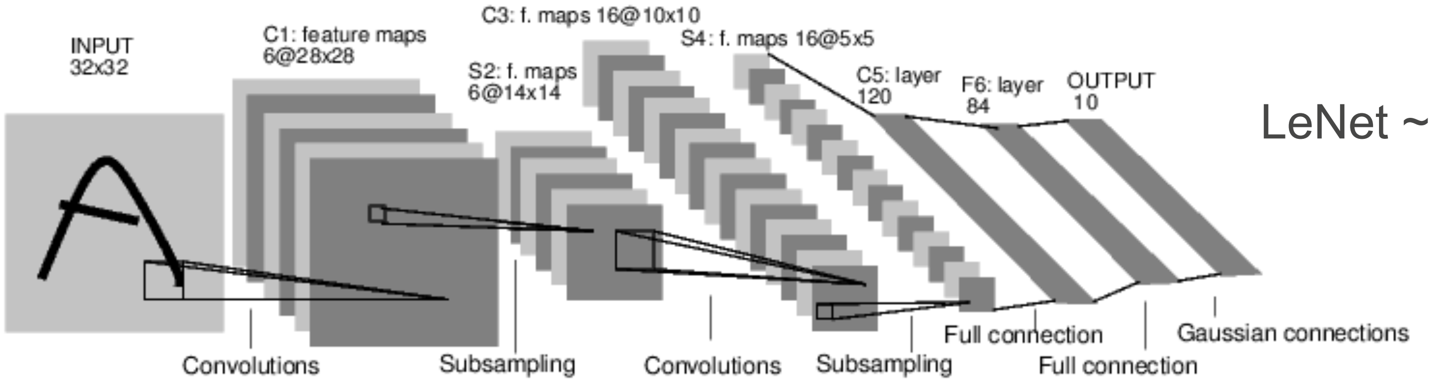
$$y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ij,ab} x_{i+a,j+b}$$

- **Translation Invariance**
  Weights invariant relative to shift in point of view

$$y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ab} x_{i+a,j+b}$$

Subsampling & MaxPooling

- Multiple convolutions blow up dimensionality



LeNet ~1995

- Subsampling - average over patches (works OK)
- MaxPooling - pick the maximum over patches (much better)

image credit - Le Cun et al, 1998

LeNet in MXNet

```
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Conv2D(channels=20, kernel_size=5, activation='tanh'))
    net.add(gluon.nn.AvgPool2D(pool_size=2))
    net.add(gluon.nn.Conv2D(channels=50, kernel_size=5, activation='tanh'))
    net.add(gluon.nn.AvgPool2D(pool_size=2))
    net.add(gluon.nn.Flatten())
    net.add(gluon.nn.Dense(500, activation='tanh'))
    net.add(gluon.nn.Dense(10))

loss = gluon.loss.SoftmaxCrossEntropyLoss()

(size and shape inference is automatic)
```

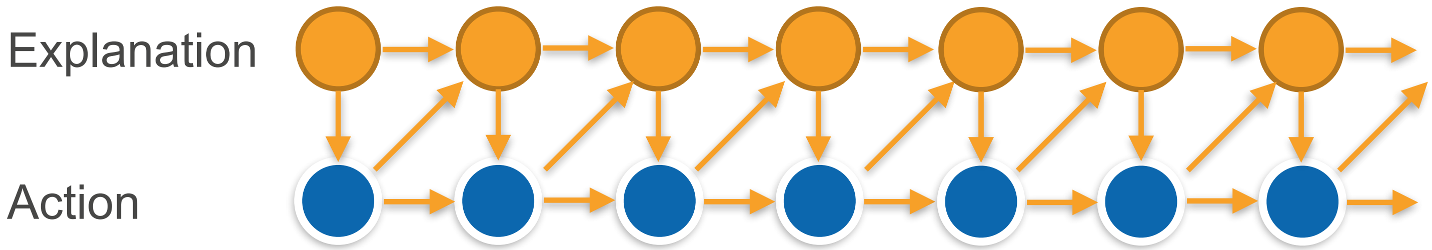
More Layers

- **The usual suspects**
  - gluon.nn.Dense(units=50, …)
  - gluon.nn.Activation(activation=‘relu’)
  - gluon.nn.Dropout(rate=0.3)
  - gluon.nn.BatchNorm(…)
  - gluon.nn.Embedding(…) for text
- **Convolutions**
  - gluon.nn.Conv1D, 2D, 3D
  - gluon.nn.Conv1DTranspose, 2D, 3D for Deconvolution
- **Pooling Layers**
  - gluon.nn.MaxPool1D, 2D, 3D, gluon.nn.AvgPool1D, 2D, 3D

Sequence Models

Latent Variable Models

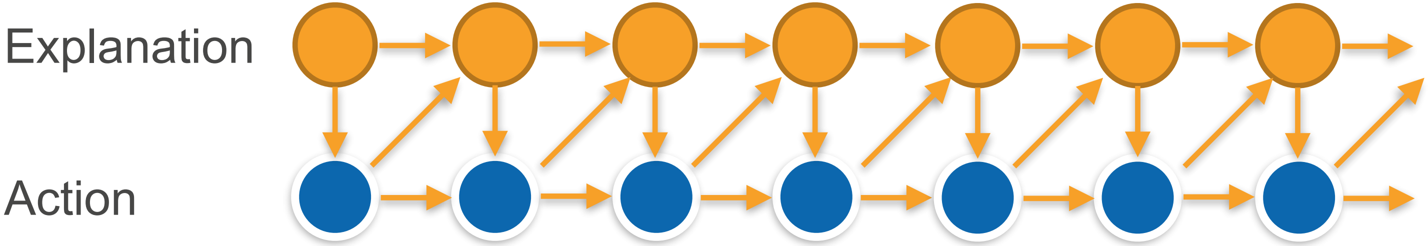
- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
  - Clusters (navigational, informational queries in search)
  - Topics (interest distributions for users over time)
  - Kalman Filter (trajectory and location modeling)



Latent Variable Models

- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**

Are the parametric models really true?

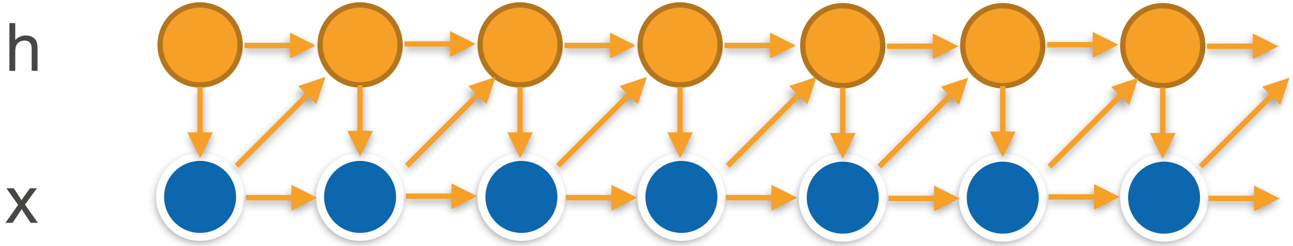


Latent Variable Models

- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
  - Nonparametric model / spectral
  - Use data to determine shape
  - Sidestep approximate inference

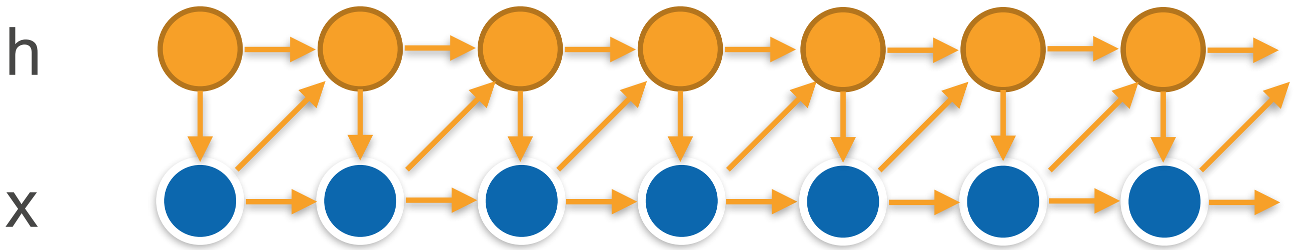
$$h_t = f(x_{t-1}, h_{t-1})$$

$$x_t = g(x_{t-1}, h_t)$$



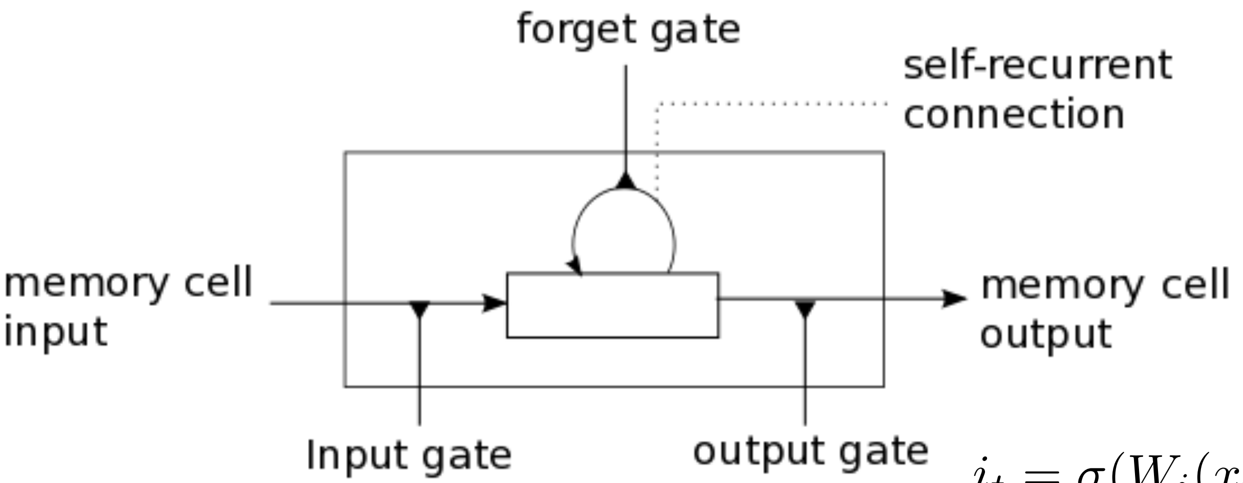
Latent Variable Models

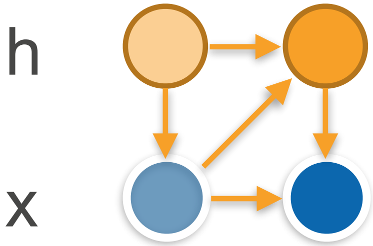
- **Temporal sequence of observations**
  Purchases, likes, app use, e-mails, ad clicks, queries, ratings
- **Latent state to explain behavior**
  - Plain deep network = RNN
  - Deep network with attention = LSTM / GRU …
    (learn when to update state, how to read out)



Long Short Term Memory

Hochreiter & Schmidhuber, 1997

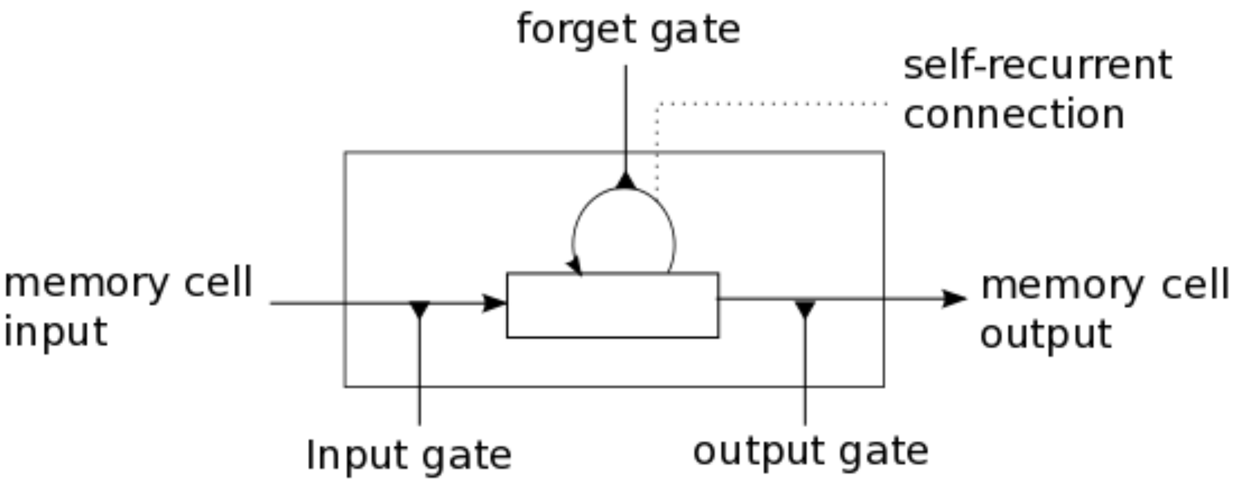


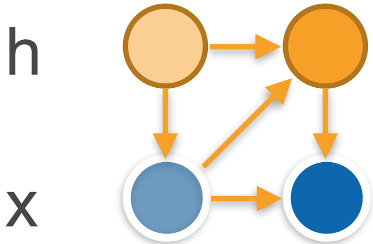


$$
\begin{aligned}
i_t &= \sigma(W_i(x_t, h_t) + b_i) \\
f_t &= \sigma(W_f(x_t, h_t) + b_f) \\
z_{t+1} &= f_t \cdot z_t + i_t \cdot \tanh(W_z(x_t, h_t) + b_z) \\
o_t &= \sigma(W_o(x_t, h_t, z_{t+1}) + b_o) \\
h_{t+1} &= o_t \cdot \tanh z_{t+1}
\end{aligned}
$$

Long Short Term Memory

Hochreiter & Schmidhuber, 1997





$$(z_{t+1}, h_{t+1}, o_t) = \mathrm{LSTM}(z_t, h_t, x_t)$$

```
rnn.LSTM(num_hidden, num_layers, dropout, input_size)
```

don’t worry - we will practice this A LOT