Machine Learning
Parametric Nonlinearity

Rajesh Ranganath
What we did last time?

- Build non-linear models “nonparametrically”
- What about parametric models?
- Features $x_i$ real valued

- Label $y_i$

*Try to find a 2 dimensional vector to do nonlinear regression*
An idea. Reduce the dimensionality of $x$

- Do something that preserves the variance of $x$
- Multiply $x$ by a random matrix
- Take the first two features
An idea. Reduce the dimensionality of $x$

- Do something that preserves the variance of $x$
- Multiply $x$ by a random matrix
- Take the first two features

All of these approaches do not look at the response
Define hidden units for $k = 1, 2$

$$h_{i,k,1}(x_i; W)$$

Define

$$h_{i,1} = [h_{i,k=1,1}, h_{i,k=2,1}]$$

Try to minimize the error

$$\min_{\theta, W} \sum_{i=1}^{n} (y_i - \theta ^\top h(x_i; W))^2$$

Learn the parameters with $\theta, W$ by gradient descent
How do we choose $h$?
How do we choose $h$?

- Maybe a decision tree?
- Maybe a random forest?
- What’s the simplest?
How do we choose $h$?

$$h_{i,k=1,1} = w_{k=1,1}^\top x_i$$

A linear function seems simple.

Is this a good idea?
Start with

$$\min_{\theta, W} \sum_{i=1}^{n} (y_i - \theta^\top h(x_i; W))^2$$

Use

$$h_{i,k=1,1} = w_{k=1,1}^\top x_i$$

Get

$$\min_{\theta, W} \sum_{i=1}^{n} \left( y_i - \sum_{k=1}^{2} \theta_k w_{k,1}^\top x_i \right)^2$$
\[
\min_{\theta, W} \sum_{i=1}^{n} \left( y_i - \sum_{k=1}^{2} \theta_k w_{k,1}^T x_i \right)^2
\]

Rearrange

\[
\min_{\theta, W} \sum_{i=1}^{n} \left( y_i - x_i^T \sum_{k=1}^{2} \theta_k w_{k,1} \right)^2
\]
\[
\min_{\theta, w} \sum_{i=1}^{n} \left( y_i - \sum_{k=1}^{2} \theta_k w_{k,1} x_i \right)^2
\]

Rearrange

\[
\min_{\theta, w} \sum_{i=1}^{n} \left( y_i - x_i^\top \sum_{k=1}^{2} \theta_k w_{k,1} \right)^2
\]

Define

\[
\beta_j = \sum_{k=1}^{2} \theta_k w_{k,1,j}
\]
\[
\min_{\theta, w} \sum_{i=1}^{n} \left( y_i - \sum_{k=1}^{2} \theta_k w_{k,1} x_i \right)^2
\]

Rearrange

\[
\min_{\theta, w} \sum_{i=1}^{n} \left( y_i - x_i^\top \sum_{k=1}^{2} \theta_k w_{k,1} \right)^2
\]

Define

\[
\beta_j = \sum_{k=1}^{2} \theta_k w_{k,1,j}
\]

\[
\min_{\theta, w} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2
\]

Less flexible than linear regression!
What happened?

- $h$ was a linear function
What happened?

- $h$ was a linear function

- The functions $h$ were combined via an inner product
What happened?

- $h$ was a linear function
- The functions $h$ were combined via an inner product
- Inner products are linear
What happened?

- $h$ was a linear function
- The functions $h$ were combined via an inner product
- Inner products are linear
- Linear functions of linear functions are linear
What happened?

- $h$ was a linear function
- The functions $h$ were combined via an inner product
- Inner products are linear
- Linear functions of linear functions are linear
- Solution?
What happened?

- $h$ was a linear function
- The functions $h$ were combined via an inner product
- Inner products are linear
- Linear functions of linear functions are linear

Solution?

We need nonlinearities?
$$\min_{\theta, w} \sum_{i=1}^{n} (y_i - \theta^T h(x_i; w))^2$$

Where should the nonlinearities go?
\[
\min_{\theta, \mathbf{w}} \sum_{i=1}^{n} (y_i - \theta^\top h(x_i; \mathbf{w}))^2
\]

Where should the nonlinearities go?
Draw inspiration from last class

\[ y_i = \theta_1 x_i, \quad \text{vitamin – d} + \theta_2 x_i, \quad \text{vitamin – k} + \theta_0 + \epsilon_i + \theta_4 x_i, \quad \text{vitamin – d} \times x_i \]

Make the functions \( h \) non-linear
Want a function $\alpha : \mathbb{R} \to \mathbb{R}$

- Fast to compute
- Fast to differentiate
- Relatively smooth
Options

- Sigmoid logistic

\[ \alpha(a) = \frac{1}{1 + \exp(-a)} \]

- Hyperbolic Tangent

\[ \alpha(a) = \frac{\exp(2a) - 1}{\exp(2a) + 1} \]

- Softplus

\[ \alpha(a) = \log(1 + \exp(a)) \]

All are smooth, fast to compute, relatively smooth
Now we can define $h_i, k^{1,1} = \alpha(w^\top k^{1,1} x_i)$. We get

$$\min_{\theta, W} \sum_{i=1}^{n} y_i - 2 \sum_{k=1}^{2} \theta_k \alpha(w^\top k^{1,1} x_i)$$
\[ \text{Compare } \min \theta, \sum_{i=1}^{W} (y_i - 2 \sum_{k=1}^{\theta} \alpha (w_k \top x_i)) \]

\[ \text{with } y_i = \theta_1 x_i, \text{vitamin-d} + \theta_2 x_i, \text{vitamin-k} + \theta_0 + \varepsilon_i + \theta_4 x_i, \text{vitamin-d} x_i, \text{vitamin-k} \]

Linear regression with learned features!
Compare \( \min \theta \), \( \sum_{i=1}^{W} y_i^{'} - 2 \sum_{k=1}^{\theta} \alpha (w_k^\top x_i) \).

Linear regression with learned features!
Why only two representations?

\[ \min \theta, W \]

\[ \sum_{i=1}^{n} y_i - K \sum_{k=1}^{\theta} \alpha \left( w_k^\top x_i \right) \]
Why only two representations?

$\min \theta, W \sum_{i=1}^{n} y_i - K \sum_{k=1}^{\alpha} \theta_k (w_k^\top, 1) x_i$
What we did:

1. Take the input $x_i$
2. Convert it to $h_{i,1}$ with $h_{i,1} = \alpha (W_1 x_i + b_1)$
3. Learn the new "features" with regression weights
   $$\min_{\theta, W_n} \sum_{i=1}^{\sum} y_i - \theta^\top h_{i,1}^2$$

Why do 1 and 2 only once?
Similarly define

$$h_i, \ell = \alpha (W_\ell h_{i, \ell - 1} + b_\ell)$$
Similarly define $h_{i,1} = \alpha (W_1 x_i + b_1)$

Similarly define $h_{i,2} = \alpha (W_2 h_{i,2} + b_2)$

Similarly define $h_{i,\ell} = \alpha (W_\ell h_{i,\ell-1} + b_\ell)$
1. Take the input $x_i$
2. Convert it to $h_{i,1}$ with $h_{i,1} = \alpha(W_1 x_i + b_1)$
3. Convert it to $h_{i,2}$ with $h_{i,2} = \alpha(W_2 h_{i,1} + b_2)$
4. More of the same
5. Convert it to $h_{i,L}$ with $h_{i,L} = \alpha(W_L h_{i,L-1} + b_L)$
6. Learn the new "features" with regression weights
   $$\min_{\theta, W_n} \sum_{i=1}^{\sum} y_i - \theta^\top h_{i,L}^2$$
1. Take the input $x_i$

2. Convert it to $h_{i,1}$ with $h_{i,1} = \alpha (W_1 x_i + b_1)$

3. Convert it to $h_{i,2}$ with $h_{i,2} = \alpha (W_2 h_{i,1} + b_2)$

4. More of the same

5. Convert it to $h_{i,L}$ with $h_{i,L} = \alpha (W_L h_{i,L-1} + b_L)$

6. Learn the new "features" with regression weights

$$\min_{\theta} \sum_{i=1}^{\sum} y_i - \theta^\top h_{i,L}$$
1. Take the input $x_i$

2. Convert it to $h_{i,1}$ with $h_{i,1} = \alpha (W_1 x_i + b_1)$

3. Convert it to $h_{i,2}$ with $h_{i,2} = \alpha (W_2 h_{i,1} + b_2)$

4. More of the same

5. Convert it to $h_{i,L}$ with $h_{i,L} = \alpha (W_L h_{i,L-1} + b_L)$

6. Learn the new "features" with regression weights

$$\min_{\theta, W} \sum_{i=1}^{L} (y_i - \theta^\top h_{i,L})^2$$
1. Take the input $x_i$

2. Convert it to $h_{i,1}$ with $h_{i,1} = \alpha \left( W_1 x_i + b_1 \right)$

3. Convert it to $h_{i,2}$ with $h_{i,2} = \alpha \left( W_2 h_{i,1} + b_2 \right)$

4. More of the same

5. Convert it to $h_{i,L}$ with $h_{i,L} = \alpha \left( W_L h_{i,L-1} + b_L \right)$

6. Learn the new “features” with regression weights

$$\min_{\theta, W_n} \sum_{i=1}^{\sum} y_i - \theta^\top h_{i,L}$$
1. Take the input $x_i$

2. Convert it to $h_{i,1}$ with $h_{i,1} = \alpha (W_1 x_i + b_1)$

3. Convert it to $h_{i,2}$ with $h_{i,2} = \alpha (W_2 h_{i,1} + b_2)$

4. More of the same

5. Convert it to $h_{i,L}$ with $h_{i,L} = \alpha (W_L h_{i,L-1} + b_L)$

6. Learn the new “features” with regression weights

$$\min_{\theta, W} \sum_{i=1}^{n} (y_i - \theta^\top h_{i,L})^2$$
\[ \min_{\theta, w} \sum_{i=1}^{n} (y_i - \theta^T h_{i,L})^2 \]

How do we solve the minimization problem?
\[
\min_{\theta, w} \sum_{i=1}^{n} (y_i - \theta^T h_{i,L})^2
\]

How do we solve the minimization problem?

*Gradients!*
\[ L = \sum_{i=1}^{n} y_i - \theta^\top h_i, \]

**Warmup.** Compute derivative with respect to \( h_i \),

\[ \nabla_{h_i} L = \theta^\top h_i - y_i. \]

What about derivative with respect to \( h_i \),

\[ \nabla_{h_i} L = \alpha (W h_i, L - 1 + b). \]

Use chain rule

\[ \nabla_{h_i} L = (\nabla_{h_i} L - 1 h_i)^\top \nabla_{h_i} L = W^\top \alpha' (W h_i, L - 1 + b) \cdot \theta^\top h_i - y_i \theta. \]
\[ L = \sum_{i=1}^{n} \left( y_i - \theta^\top h_i \right), \]

Warmup. Compute derivative with respect to \( h_i \),

\[ \nabla_{h_i} L = \theta^\top h_i - y_i, \]

What about derivative with respect to \( h_i \),

\[ L^{-1} h_i = \alpha \left( W L h_i + b L \right) \]

Use chain rule

\[ \nabla_{h_i} L \left( L^{-1} h_i \right)^\top \nabla_{h_i} L = W^\top \alpha' \left( W L h_i + b L \right) \cdot \theta^\top h_i \left( L^{-1} - y_i \right). \]
More generally,\[ \nabla h_i, \ell - 1 L = (\nabla h_i, \ell - 1 h_i, \ell)^\top \nabla h_i, \ell L \]

Compute all
1. Compute $\nabla h_i, L$
2. Compute $\nabla h_i, L - 1 L$
3. ...
4. Compute $\nabla h_i, 1 L$
\[ \nabla h_i, 1 L = (\nabla h_i, 1 h_i, 2)^\top \nabla h_i, 2 L \]

42
More generally, \( \nabla h_i, L = (\nabla h_i, L - 1 h_i, L) \top \nabla h_i, L \)

Compute all

1. Compute \( \nabla h_i, L \)

2. Compute \( \nabla h_i, L - 1 \)

\( \nabla h_i, L = (\nabla h_i, L - 1 h_i, L) \top \nabla h_i, L \)

3. ...

4. Compute \( \nabla h_i, 1 \)

\( \nabla h_i, 1 = (\nabla h_i, 1 h_i, 2) \top \nabla h_i, 2 \)
How do we use this?

\[ \nabla_{W_\ell} \mathcal{L} = \sum_{i=1}^{n} \nabla_{h_{i,\ell}} \mathcal{L} \nabla_{W} h_{i,\ell} \]

and

\[ \nabla_{b_\ell} \mathcal{L} = \sum_{i=1}^{n} \nabla_{h_{i,\ell}} \mathcal{L} \nabla_{b} h_{i,\ell} \]

Derivatives with respect to all parameters are sums of hidden derivatives.
What did we do?

- Forward compute $h_i$, $\ell$
- Compute the derivative of the loss function with $h_i$, $L$
- Reverse compute the derivative of $h_i$, $\ell$
- Use derivative of $h_i$, $\ell$ to compute derivatives of parameters
- Use derivatives to update parameters

Called backpropagation
How do we use this?

\[ \nabla_{W_{\ell}} \mathcal{L} = \sum_{i=1}^{n} \nabla_{h_{i,\ell}} \mathcal{L} \nabla_{W_{h_{i,\ell}}} \]

and

\[ \nabla_{b_{\ell}} \mathcal{L} = \sum_{i=1}^{n} \nabla_{h_{i,\ell}} \mathcal{L} \nabla_{b_{h_{i,\ell}}} \]

Derivatives are sums; we can use stochastic optimization
We evaluate both top-1 and top-5 error rates.

We first evaluate 18-layer and 34-layer plain nets. The 34-layer plain net is in Fig. 3. We also obtain a final 50k validation error.

For best results, we adopt the fully-connected layer with a ReLU activation followed by a dropout regularization of 0.5.

Figure 4 shows the degradation problem - the plain net. To reveal the reasons, in Fig. 4.1 we evaluate our method on the ImageNet 2012 classification task. Table 1 shows the results.

We evaluate both top-1 and top-5 error rates.

We use a weight decay of 0.0001 and a momentum of 0.9. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus. We also adopt the fully-connected layer with a ReLU activation followed by a dropout regularization of 0.5.
We first evaluate 18-layer and 34-layer plain nets. The 34-layer plain net is in Fig. 4.1. ImageNet Classification.

We evaluate both top-1 and top-5 error rates. For best results, we adopt the fully-connected layer combination and the single mini-batch setting. We use a weight decay of 0.0001 and a momentum of 0.9. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, as in [21]. We do not use dropout [14].

The image is resized with its shorter side randomly sampled in [216 ± 32]. For best results, we adopt the fully-connected layer combination and the single mini-batch setting. We use a weight decay of 0.0001 and a momentum of 0.9. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, as in [21]. We do not use dropout [14].

In testing, for comparison studies we adopt the standard 10-crop testing [21]. The models are trained on the 1.28 million training images, and evaluated on the 50k validation images. We also obtain a final result on the 100k test images, reported by the test server.

We have observed the degradation problem - the networks fail to improve their training/validation errors during the training procedure. We have observed the degradation problem - the networks fail to improve their training/validation errors during the training procedure. We have observed the degradation problem - the networks fail to improve their training/validation errors during the training procedure. We have observed the degradation problem - the networks fail to improve their training/validation errors during the training procedure. We have observed the degradation problem - the networks fail to improve their training/validation errors during the training procedure.
Automatic Differentiation

- Tools to compute derivatives
- Write the forward computation
- Derivative computed by software!
- Works by keeping track of dependency graph
Automatic Differentiation: Easy to Use

Tensorflow

Stan

Pytorch

Many More Systems
The only way?

\[ \nabla h_{i,\ell-1} \mathcal{L} = (\nabla h_{i,\ell-1} h_{i,\ell})^\top \nabla h_{i,\ell} \mathcal{L} \]

Implies

\[ \nabla h_{i,\ell-1} \mathcal{L} = \left( \prod_{m=L}^{\ell} (\nabla h_{i,m-1} h_{i,m})^\top \right) \nabla h_{i,L} \mathcal{L} \]

- Backpropagation computes from right to left
- Why not compute it from left to right?
The only way?

\[ \nabla h_{i,\ell-1} \mathcal{L} = (\nabla h_{i,\ell-1} h_{i,\ell})^\top \nabla h_{i,\ell} \mathcal{L} \]

Implies

\[ \nabla h_{i,\ell-1} \mathcal{L} = \left( \prod_{m=L}^{\ell} (\nabla h_{i,m-1} h_{i,m})^\top \right) \nabla h_{i,L} \mathcal{L} \]

- Backpropagation computes from right to left
- Why not compute it from left to right?
- Why not compute it from the middle out?
The only way?

\[ \nabla h_{i,\ell-1} \mathcal{L} = (\nabla h_{i,\ell-1} h_{i,\ell})^\top \nabla h_{i,\ell} \mathcal{L} \]

Implies

\[ \nabla h_{i,\ell-1} \mathcal{L} = \left( \prod_{m=L}^{\ell} (\nabla h_{i,m-1} h_{i,m})^\top \right) \nabla h_{i,L} \mathcal{L} \]

- Backpropagation computes from right to left
- Why not compute it from left to right?
- Why not compute it from the middle out?
- Optimal order? 53
Video
Power

Suppose

\[ y \sim p(y \mid x) \]

We know optimal form to minimize squared error is

\[ \mathbb{E}_p[y \mid x] \]

Can neural networks achieve this goal?
Power

Suppose

\[ y \sim p(y|x) \]

We know optimal form to minimize squared error is

\[ E_p[y|x] \]

Can neural networks achieve this goal?

For a sufficiently big network and nonlinearity choice, yes! Why?
ReLU

Rectified Linear Units

\[ \alpha(a) = \max(0, a) \]

- Nonlinear, but not differentiable at zero
- Very popular, why?
ReLU

Rectified Linear Units

\[ \alpha(a) = \max(0, a) \]

How does one compute exp on a computer?
ReLU

Rectified Linear Units

\[ \alpha(a) = \max(0, a) \]

How does one compute exp on a computer?

\[ \exp(a) = \sum_{n=0}^{\infty} \frac{x^n}{n!} \]

Can be sped up a lot
Rectified Linear Units

\[ \alpha(a) = \max(0, a) \]

- Ultra fast to compute
- No shrinkage of the gradient
Regularization

Neural networks are very powerful. Can they overfit?
Regularization

Neural networks are very powerful. Can they overfit?

Options?
Regularization

Neural networks are very powerful. Can they overfit?

Options?
- Add l2 regularization to weights
Regularization

Neural networks are very powerful. Can they overfit?

Options?

- Add l2 regularization to weights
- Limit the number of iterations
Regularization

Neural networks are very powerful. Can they overfit?

Options?

- Add l2 regularization to weights
- Limit the number of iterations
- Add noise
Algorithms as Regularization

Consider

- Decision tree algorithm only run for two steps
- Gradient boosting only run for ten steps
- Gradient optimization run for 100 steps
Algorithms as Regularization

Consider

- Decision tree algorithm only run for two steps
- Gradient boosting only run for ten steps
- Gradient optimization run for 100 steps

*All estimation algorithms limit the model capacity*
Bias  Variance

\[ \mathcal{M}^{\text{big}} \]

\[ \hat{f}^{\text{big}}_{1} \quad \ast \quad \hat{f}^{\text{big}}_{2} \]

\[ \mathcal{M}^{\text{small}} \]

\[ \hat{f}^{\text{small}}_{1} \quad \hat{f}^{\text{small}}_{2} \]
What does l2 regularization do?

\[ h_{i,\ell} = \tanh(W_{\ell}^T h_{\ell-1}) \]
What does l2 regularization do?

\[ h_{i,\ell} = \tanh(W^\top_{\ell} h_{\ell-1}) \]

- Lowering l2 shrinks \( w_{k=1,1}^T x_i \)
- Shrinks to 0
- Is this reasonable?
What does l2 regularization do?

\[ h_{i,\ell} = tanh(W_{\ell}^T h_{\ell-1}) \]

- Lowering l2 shrinks \( w_{k=1,1}^T x_i \)
- Shrinks to 0
- Is this reasonable?
- Throws away all “work” of the previous layers
ResNet

Idea: Center everything at its input

\[ h_{i,\ell} = \sigma(W_{\ell}^T h_{\ell-1}) + h_{\ell-1} \]

A residual connection!
ResNet

Idea: Center everything at its input

\[ h_{i,\ell} = \sigma(W_{\ell}^T h_{\ell-1}) + h_{\ell-1} \]

A residual connection!

- Can also improve conditioning
- Gradients can pass back to earlier more simply
The dotted shortcuts increase dimensions. Right:

Figure 3. Example network architectures for ImageNet.

Table 1 shows: the plain network with 34 parameter layers (3.6 billion FLOPs). The 34-layer plain net is in Fig. 4.1. ImageNet Classification.

We evaluate our method on the ImageNet 2012 classification dataset which consists of 1000 classes. The models are trained on the 1.28 million training images, and evaluated on the 50k validation images. We also obtain a final result on the 100k test images, reported by the test server.

We first evaluate 18-layer and 34-layer plain nets. The 18-layer plain net has higher validation error than the shallower 18-layer tailored architectures.

Our implementation for ImageNet follows the practice at multiple scales (images are resized such that the shorter side is in \( \{224, 384, 480\} \times \{224, 384, 480\} \)). When the dimensions increase (dotted line shortcuts in Eqn. (2)), we consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; (B) The projection shortcut in Eqn. (3) is used. We adopt batch normalization (BN) right after each convolution and use a weight decay of 0.0001 and a momentum of 0.9. We use SGD with a mini-batch size of 256. The learning rate is used. We adopt batch

In testing, for comparison studies we adopt the standard 10-crop testing \( \{224, 384, 480\} \times \{224, 384, 480\} \) that consists of 1000 classes. The models are trained for up to 14 iterations. We use momentum of 0.9. We initialize the weights shared in Fig. 3.4. Implementation.

When the shortcuts go across feature maps of two different layers, the projection shortcut introduces additional parameters. For increasing dimensions, we consider two options: (A) The shortcut still performs identity mapping, with extra zero entries padded in Fig. 3.5. Insert shortcut connections (Fig. 3.6).

We adopt batch normalization (BN) right after each convolution and use a weight decay of 0.0001 and a momentum of 0.9. We use SGD with a mini-batch size of 256. The learning rate is used. We adopt batch

Table 2 shows: the residual network with 34 parameter layers (3.6 billion FLOPs). The residual network has lower validation error than the plain network into its counterpart residual version. The identity shortcut connections (Fig. 3.7) which turn the plain network into its counterpart residual version. The identity shortcut connections (Fig. 3.7).
What about limiting the function family?
ConvNet

- Original layers are “fully connected”

\[ h_{i,k,\ell} = \tanh(w_{\ell,k}^T h_{\ell-1}) \]

- Limit connectivity

\[ h_{i,k,\ell} = \tanh(w_{\ell,k}^T h_{\ell-1,1...5}) \]

- \( w_{\ell,k} \) much smaller only five dimensional

- \( w_{\ell,k} \) called a filter

- Create a lot of units by moving around limited connectivity mask

Imposes translation invariance to the features
Bias + Variance

\[ \hat{f}_{\text{big}}^2 \]

\[ \hat{f}_{\text{small}}^1 \]

\[ \hat{f}_{\text{big}}^1 \]

\[ \hat{f}_{\text{small}}^2 \]
ResNet Performance

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 Error</th>
<th>Top-5 Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16 [41]</td>
<td>28.07</td>
<td>9.33</td>
</tr>
<tr>
<td>GoogLeNet [44]</td>
<td>-</td>
<td>9.15</td>
</tr>
</tbody>
</table>

We further note that the degradation problem of plain nets is also witnessed for the bottleneck designs.

ResNet Performance

He 2015
Sequences

Try to make predictions over time

\[ p(x_t | x_{<t}) \]

Examples

- Predict \( t \)-word in document given the previous words
- Predict weather today given the previous weather
- Predict next chord in a song
Recurrent Neural Networks

Maximize the log-likelihood of

\[ \sum_{i=1}^{T} \log p(x_t | x_{<t}) \]

Where \( \log p(x_t | x_{<t}) \)

\[ \log p(x_t | x_{<t}) = \log p(x_t | h_t) \]

\[ h_t = f(h_{t-1}, x_{t-1}; W) \]

- Use neural networks to parameterize \( f \)
- Recurrent because same function gets applied over and over
Recurrent Neural Networks

Maximize the log-likelihood of

$$\sum_{i=1}^{T} \log p(x_t | x_{<t})$$

Where $\log p(x_t | x_{<t})$

$$\log p(x_t | x_{<t}) = \log p(x_t | h_t)$$

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

- Use neural networks to parameterize $f$
- Recurrent because same function gets applied over and over

*Lots of intriguing properties. More on homework!*
Good Performance. Scalable Compute. Answer to supervised ML?
Adversarial Examples

$$x + 0.007 \times \text{sign}(\nabla_x (\theta x )) \quad = \quad x + \text{sign}(\nabla_x (\theta x ))$$

“panda”
57.7% confidence

“nematode”
8.2% confidence

“gibbon”
99.3% confidence

Goodfellow 2014

Why?