Neural Potential Functions
Reminders

• Homework 2: BP for Syntax Trees
  – Out: Sat, Sep. 28
  – Due: Sat, Oct. 12 at 11:59pm

• Last chance to switch between 10-418 / 10-618 is October 7th (drop deadline)
BACKPROPAGATION AND BELIEF PROPAGATION
Whiteboard:

– Gradient of MRF log-likelihood with respect to log potentials
– Gradient of MRF log-likelihood with respect to potentials
Factor Derivatives

Log-probability:

\[
\log p(y) = \left[ \sum_{\alpha} \log \psi_{\alpha}(y_{\alpha}) \right] - \log \sum_{y' \in Y} \prod_{\alpha} \psi_{\alpha}(y'_{\alpha}) \quad (1)
\]

Derivatives:

\[
\frac{\partial \log p(y)}{\partial \log \psi_{\alpha}(y'_{\alpha})} = 1(y_{\alpha} = y'_{\alpha}) - p(y'_{\alpha}) \quad (2)
\]

\[
\frac{\partial \log p(y)}{\partial \psi_{\alpha}(y'_{\alpha})} = \frac{1(y_{\alpha} = y'_{\alpha}) - p(y'_{\alpha})}{\psi_{\alpha}(y'_{\alpha})} \quad (3)
\]
Outline of Examples

• **Hybrid NN + HMM**
  – Model: neural net for emissions
  – Learning: backprop for end-to-end training
  – Experiments: phoneme recognition (Bengio et al., 1992)

• **Hybrid RNN + HMM**
  – Model: neural net for emissions
  – Experiments: phoneme recognition (Graves et al., 2013)

• **Hybrid CNN + CRF**
  – Model: neural net for factors
  – Experiments: natural language tasks (Collobert & Weston, 2011)
  – Experiments: pose estimation

• **Tricks of the Trade**
HYBRID: NEURAL NETWORK + HMM
Markov Random Field (MRF)

Joint distribution over tags $Y_i$ and words $X_i$
The individual factors aren’t necessarily probabilities.

$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, \text{an}, \text{arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$
Hidden Markov Model

But sometimes we choose to make them probabilities. Constrain each row of a factor to sum to one. Now $Z = 1$.

$$p(n, v, p, d, n, \text{time}, \text{flies}, \text{like}, an, arrow) = \frac{1}{Z} (0.3 \times 0.8 \times 0.2 \times 0.5 \times \ldots)$$
Hybrid: NN + HMM

Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

Continuous HMM emission: $Y_t \in \mathcal{R}^K$

HMM: $p(Y, S) = \prod_{t=1}^{T} p(Y_t|S_t)p(S_t|S_{t-1})$

$$p(Y_t|S_t = i) = b_{i,t} = \sum_{k} \frac{Z_k}{((2\pi)^n |\Sigma_k|)^{1/2}} \exp\left(-\frac{1}{2}(Y_t - \mu_k)^T \Sigma_k^{-1} (Y_t - \mu_k)\right)$$

Gaussian emission:

(Bengio et al., 1992)
Hybrid: NN + HMM

Discrete HMM state: $S_t \in \{/p/, /t/, /k/, /b/, /d/, \ldots, /g/\}$

Continuous HMM emission: $Y_t \in \mathcal{R}^K$

HMM: $p(Y, S) = \prod_{t=1}^{T} p(Y_t|S_t)p(S_t|S_{t-1})$

$p(Y_t|S_t = i) = b_{i,t} = \sum_{k} \frac{Z_k}{((2\pi)^n | \sum_k |)^{1/2}}$

Lots of oddities to this picture:

- **Clashing visual notations** (graphical model vs. neural net)

- HMM generates data **top-down**, NN generates **bottom-up** and they meet in the middle.

- The “observations” of the HMM are not actually observed (i.e. $x$’s appear in NN only)

So what are we missing?
Hybrid: NN + HMM
\[ a_{i,j} = p(S_t = i | S_{t-1} = j) \]
\[ b_{i,t} = p(Y_t | S_t = i) \]

Hybrid: NN + HMM

**Forward-backward algorithm:** a “feed-forward” algorithm for computing alpha-beta probabilities.

\[
\alpha_{i,t} = P(Y_1^t \text{ and } S_t = i \mid \text{model}) = b_{i,t} \sum_j a_{ij} \alpha_{j,t-1}
\]
\[
\beta_{i,t} = P(Y_{t+1}^T \mid S_t = i \text{ and model}) = \sum_j a_{ij} \beta_{j,t+1}
\]
\[
\gamma_{i,t} = P(S_t = i \mid Y_1^t \text{ and model}) = \alpha_{i,t} \beta_{i,t}
\]

**Log-likelihood:** a “feed-forward” objective function.

\[
\log p(S, Y) = \alpha_{\text{END}, T}
\]
A Recipe for Graphical Models

1. Given training data:
   \( \{ \mathbf{x}_i, y_i \}_{i=1}^{N} \)

2. Choose each of these:
   - Decision function
   - Loss function

\[ \hat{y} = f_{\theta}(\mathbf{x}_i) \]

\[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:

4. Train with SGD:
   (take small steps opposite the gradient)

Decision / Loss Function for Hybrid NN + HMM


\[
\begin{align*}
\alpha_{i,t} &= P(Y_{1:t} \text{ and } S_t = i \mid \text{model}) = b_{i,t} \sum_j a_{ji} \alpha_{j,t-1} \\
\beta_{i,t} &= P(Y_{t+1:T} \mid S_t = i \text{ and model}) = \sum_j a_{ij} b_{j,t+1} \beta_{j,t+1} \\
\gamma_{i,t} &= P(S_t = i \mid Y_{1:T} \text{ and model}) = \alpha_{i,t} \beta_{i,t}
\end{align*}
\]

Log-likelihood: a “feed-forward” objective function.

\[ \log p(S, Y) = \alpha_{\text{END},T} \]

How do we compute the gradient?
Backpropagation is just repeated application of the chain rule from Calculus 101.

Recall...

**Backpropagation**

The backpropagation algorithm is a general method for computing the gradient of a neural network. Here we generalize the concept of a neural network to include any arithmetic circuit. Applying the backpropagation algorithm on these circuits amounts to repeated application of the chain rule. This general algorithm goes under many other names: automatic differentiation (AD) in the reverse mode (Griewank and Corliss, 1991), analytic differentiation, module-based AD, autodiff, etc.

**Binary logistic regression can be interpreted as a arithmetic circuit.** To compute the derivative of some loss function (below we use regression) with respect to the model parameters, we can repeatedly apply the chain rule (i.e., backpropagation). That is, the computation $J = \log (1 + e^{x})$, if the inputs and outputs of $\text{logistic}(x)$ is just repeated application of the chain rule. This general algorithm goes under many other names: automatic differentiation (AD) in the reverse mode (Griewank and Corliss, 1991), analytic differentiation, module-based AD, autodiff, etc.

**How to compute these partial derivatives?**

**Chain Rule:**

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$

**Graphical Model and Log-likelihood**

**Neural Network**

Let $y = g(u)$ and $u = h(x)$. How to compute these partial derivatives?
2.2. NEURAL NETWORKS AND BACKPROPAGATION

The material presented here acts as a supplement to later uses of backpropagation such as in Chapter 4 for training of a hybrid graphical model / neural network, and in Chapter 5 and Chapter 6 for approximation-aware training.

2.2.1 Topologies

A feed-forward neural network (Rumelhart et al., 1986) defines a decision function \( y = h(x) \) where \( x \) is termed the input layer and \( y \) the output layer. A feed-forward neural network has a statically defined topology. Figure 2.1 shows a simple 2-layer neural network consisting of an input layer \( x \), a hidden layer \( z \), and an output layer \( y \). In this example, the output layer is of length 1 (i.e. just a single scalar \( y \)). The model parameters of the neural network are a matrix \( \beta \) and a vector \( \alpha \).

The feed-forward computation proceeds as follows: we are given \( x \) as input (Fig. 2.1 (A)). Next, we compute an intermediate vector \( a \), each entry of which is a linear combination of the input (Fig. 2.1 (B)). We then apply the sigmoid function \( a_j = \frac{1}{1+\exp(a_j)} \) element-wise to obtain \( z \) (Fig. 2.1 (C)). The output layer is computed in a similar fashion, first taking a linear combination of the hidden layer to compute \( b \) (Fig. 2.1 (D)) then applying the sigmoid function to obtain the output \( y \) (Fig. 2.1 (E)). Finally we compute the loss \( J \) (Fig. 2.1 (F)) as the squared distance to the true value \( y^{(d)} \) from the training data.

We refer to this topology as an arithmetic circuit. It defines both a function mapping \( x \) to \( y \), and a procedure for numerically computing the function output. The arithmetic circuit is merely a computational graph. More specialized variants of neural networks exist, such as convolutional neural networks for image data or recurrent neural networks for time series data.
2.2. NEURAL NETWORKS AND BACKPROPAGATION

The backward pass computes $dJ/d\beta_j$ and $dJ/d\alpha_{ji}$, but also the partial derivatives with respect to each intermediate quantity $dJ/da_j$, $dJ/dz_j$, $dJ/db$, $dJ/dy$ and the input $dJ/dx_i$.

Recall…
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

**Forward computation**

$$\log p(S, Y) = \alpha_{END,T}$$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

$a_{i,j} = \ldots$ (transitions)

$b_{i,t} = \ldots$ (emissions)

$$Y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

**Forward computation**

$$J = \log p(S, Y) = \alpha_{\text{END}, T}$$

$\alpha_{i, t} = \ldots$ (forward prob)
$\beta_{i, t} = \ldots$ (backward prop)
$\gamma_{i, t} = \ldots$ (marginals)
$a_{i, j} = \ldots$ (transitions)
$b_{i, t} = \ldots$ (emissions)

$$Y_{tk} = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{j, i} x_i$$
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

Forward computation

$J = \log p(S, Y) = \alpha_{\text{END}, T}$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

$a_{i,j} = \ldots$ (transitions)

$b_{i,t} = \ldots$ (emissions)

$Y_{tk} = \frac{1}{1 + \exp(-b)}$

$b = \sum_{j=0}^{D} \beta_j z_j$

$z_j = \frac{1}{1 + \exp(-a_j)}$

$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$

Backward computation

$\frac{dJ}{db_{i,t}} = \frac{\partial \alpha_{\text{model}, t}}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} = (\sum_j \frac{\partial \alpha_{j,t+1}}{\partial \alpha_{i,t}} \frac{\partial L_{\text{model}}}{\partial \alpha_{j,t+1}})(\sum_j a_{ji} \alpha_{j,t-1})$

$= (\sum_j b_{j,t+1} a_{ji} \frac{\partial \alpha_{\text{model}, t}}{\partial \alpha_{j,t+1}})(\sum_j a_{ji} \alpha_{j,t-1}) = \frac{\alpha_{i,t}}{b_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$
Hybrid: NN + HMM

Computing the Gradient: $\nabla \ell(f_\theta(x_i), y_i)$

**Forward computation**

$$J = \log p(S, Y) = \alpha_{END,T}$$

$\alpha_{i,t} = \ldots$ (forward prob)

$\beta_{i,t} = \ldots$ (backward prop)

$\gamma_{i,t} = \ldots$ (marginals)

$a_{i,j} = \ldots$ (transitions)

$b_{i,t} = \ldots$ (emissions)

$Y_{tk} = \frac{1}{1 + \exp(-b)}$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

**Backward computation**

$$\frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$$

$$\frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}}$$

$$\frac{\partial b_{i,t}}{\partial y_{j,t}} = \sum_k \frac{z_k}{((2\pi)^n | \Sigma_k |)^{1/2}} \left( \sum_i \frac{d_{k,ij} \mu_{kj} - Y_{it}}{\Sigma_k} \right) \exp(-\frac{1}{2}(Y_{it} - \mu_k)\Sigma_k^{-1}(Y_{it} - \mu_k)^T)$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db} \frac{dy}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j} \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j} \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j} \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}} \frac{da_j}{d\alpha_{ji}} = x_i$$
**Forward computation**

\[ J = \log p(S, Y) = \alpha_{\text{END}, T} \]

\[ \alpha_{i,t} = \ldots \text{(forward prob)} \]

\[ \beta_{i,t} = \ldots \text{(backward prop)} \]

\[ \gamma_{i,t} = \ldots \text{(marginals)} \]

The derivative of the log-likelihood with respect to the neural network parameters!

\[ a_j = \sum_{i=0}^{M} \alpha_{j,i}x_i \]

**Backward computation**

\[ \frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}} \]

\[ \frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}} \]

\[ \frac{\partial b_{i,t}}{\partial y_{j,t}} = \sum_k \frac{Z_k}{((2\pi)^n |\Sigma_k|)^{1/2}} \left( \sum_i d_{k,i}(\mu_{ki} - Y_{it}) \right) \exp(-\frac{1}{2}(Y_{it} - \mu_k)\Sigma_k^{-1}(Y_{it} - \mu_k)^T) \]

\[ \frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db} \frac{dy}{db} = \frac{\exp(b)}{(\exp(b) + 1)^2} \]

\[ \frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j} \frac{db}{\beta_j} = z_j \]

\[ \frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j} \frac{db}{dz_j} = \beta_j \]

\[ \frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j} \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j) + 1)^2} \]

\[ \frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}} \frac{da_j}{\alpha_{ji}} = x_i \]
Hybrid: NN + HMM

Experimental Setup:
- **Task**: Phoneme Recognition (aka. speaker independent recognition of plosive sounds)
- **Eight output labels**:
  - `/p/`, `/t/`, `/k/`, `/b/`, `/d/`, `/g/`, `/dx/`, `/all other phonemes/`
  - These are the HMM hidden states
- **Metric**: Accuracy
- **3 Models**:
  1. NN only
  2. NN + HMM (trained independently)
  3. NN + HMM (jointly trained)

(Bengio et al., 1992)
HYBRID:
RNN + HMM
Hybrid: RNN + HMM

• Graves et al. (2013) uses a Deep Bidirectional LSTM
• Each hidden unit is an LSTM
• Deep ➔ More than two layers
Hybrid: RNN + HMM

The model, inference, and learning can be analogous to our NN + HMM hybrid

- **Objective:** log-likelihood
- **Model:** HMM/Gaussian emissions
- **Inference:** forward-backward algorithm
- **Learning:** SGD with gradient by backpropagation

(Graves et al., 2013)
Hybrid: RNN + HMM

Experimental Setup:
- **Task:** Phoneme Recognition
- **Dataset:** TIMIT
- **Metric:** Phoneme Error Rate
- **Two classes of models:**
  1. Neural Net only
  2. NN + HMM hybrids

<table>
<thead>
<tr>
<th>TRAINING METHOD</th>
<th>TEST PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC</td>
<td>21.57 ± 0.25</td>
</tr>
<tr>
<td>CTC (NOISE)</td>
<td>18.63 ± 0.16</td>
</tr>
<tr>
<td>TRANSDUCER</td>
<td>18.07 ± 0.24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>DEV PER</th>
<th>TEST PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBRNN</td>
<td>19.91 ± 0.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>21.92 ± 0.35</td>
<td></td>
</tr>
<tr>
<td>DBLSTM</td>
<td>17.44 ± 0.156</td>
<td></td>
</tr>
<tr>
<td></td>
<td>19.34 ± 0.15</td>
<td></td>
</tr>
<tr>
<td>DBLSTM (NOISE)</td>
<td>16.11 ± 0.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>17.99 ± 0.13</strong></td>
<td></td>
</tr>
</tbody>
</table>

1. Neural Net only
2. NN + HMM hybrids
HYBRID: CNN + CRF
Markov Random Field (MRF)

Joint distribution over tags $Y_i$ and words $X_i$

$$p(n, v, p, d, n, \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$
Conditional Random Field (CRF)

Conditional distribution over tags $Y_i$ given words $x_i$. The factors and $Z$ are now specific to the sentence $x$.

$$p(n, v, p, d, n \mid \text{time, flies, like, an, arrow}) = \frac{1}{Z} (4 \times 8 \times 5 \times 3 \times \ldots)$$
Hybrid: Neural Net + CRF

- In a standard CRF, each of the factor cells is a parameter (e.g. transition or emission)
- In the hybrid model, these values are computed by a neural network with its own parameters
Hybrid: Neural Net + CRF

Forward computation
Hybrid: CNN + CRF

- For **computer vision**, Convolutional Neural Networks are in **2-dimensions**
- For **natural language**, the CNN is **1-dimensional**

Figure from (Collobert & Weston, 2011)
Hybrid: CNN + CRF

“NN + SLL”
- Model: Convolutional Neural Network (CNN) with linear-chain CRF
- Training objective: maximize sentence-level likelihood (SLL)

Figure from (Collobert & Weston, 2011)
Hybrid: CNN + CRF

“NN + WLL”

- Model: Convolutional Neural Network (CNN) with logistic regression
- Training objective: maximize word-level likelihood (WLL)
Hybrid: CNN + CRF

Experimental Setup:

- **Tasks:**
  - Part-of-speech tagging (POS),
  - Noun-phrase and Verb-phrase Chunking,
  - Named-entity recognition (NER)
  - Semantic Role Labeling (SRL)

- **Datasets / Metrics:** Standard setups from NLP literature (higher PWA/F1 is better)

- **Models:**
  - Benchmark systems are typical – non-neural network systems
  - NN+WLL: hybrid CNN with logistic regression
  - NN+SLL: hybrid CNN with linear-chain CRF

<table>
<thead>
<tr>
<th>Approach</th>
<th>POS (PWA)</th>
<th>Chunking (F1)</th>
<th>NER (F1)</th>
<th>SRL (F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark Systems</td>
<td>97.24</td>
<td>94.29</td>
<td>89.31</td>
<td>77.92</td>
</tr>
<tr>
<td>NN+WLL</td>
<td>96.31</td>
<td>89.13</td>
<td>79.53</td>
<td>55.40</td>
</tr>
<tr>
<td>NN+SLL</td>
<td>96.37</td>
<td>90.33</td>
<td>81.47</td>
<td>70.99</td>
</tr>
</tbody>
</table>
Hybrid: CNN + MRF

Experimental Setup:
- **Task**: pose estimation
- **Model**: Deep CNN + MRF
TRICKS OF THE TRADE
- Use ReLU non-linearities (tanh and logistic are falling out of favor)
- Use cross-entropy loss for classification
- Use Stochastic Gradient Descent on minibatches
- Shuffle the training samples
- Normalize the input variables (zero mean, unit variance)
- Schedule to decrease the learning rate
- Use a bit of L1 or L2 regularization on the weights (or a combination)
  - But it’s best to turn it on after a couple of epochs
- Use “dropout” for regularization
- Lots more in [LeCun et al. “Efficient Backprop” 1998]
- Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition)
Deep Learning Tricks of the Trade

• Y. Bengio (2012), “Practical Recommendations for Gradient-Based Training of Deep Architectures”
  • Unsupervised pre-training
  • Stochastic gradient descent and setting learning rates
• Main hyper-parameters
  • Learning rate schedule & early stopping
  • Minibatches
  • Parameter initialization
  • Number of hidden units
  • L1 or L2 weight decay
  • Sparsity regularization
• Debugging \( \rightarrow \) use finite difference gradient checks
• How to efficiently search for hyper-parameter configurations
Tricks of the Trade

• **Lots of them:**
  – Pre-training helps (but isn’t always necessary)
  – Train with adaptive gradient variants of SGD (e.g. Adam)
  – Use max-margin loss function (i.e. hinge loss) – though only sub-differentiable it often gives better results
  – …
• A few years back, they were considered “**poorly documented**” and “requiring great expertise”
• Now there are lots of **good tutorials** that describe (very important) specific implementation details
• Many of them **also apply to training graphical models**!
SUMMARY
Summary:
Hybrid Models

Graphical models let you encode domain knowledge

Neural nets are really good at fitting the data discriminatively to make good predictions

Could we define a neural net that incorporates domain knowledge?
Summary: Hybrid Models

**Key idea:** Use a NN to learn features for a GM, then train the entire model by backprop.
MBR DECODING
Minimum Bayes Risk Decoding

• Suppose we given a loss function \( l(y', y) \) and are asked for a single tagging
• How should we choose just one from our probability distribution \( p(y|x) \)?
• A minimum Bayes risk (MBR) decoder \( h(x) \) returns the variable assignment with minimum **expected** loss under the model’s distribution

\[
h_\theta(x) = \arg\min_{\hat{y}} \mathbb{E}_{y \sim p_\theta(\cdot | x)} \left[ l(\hat{y}, y) \right]
= \arg\min_{\hat{y}} \sum_y p_\theta(y | x) l(\hat{y}, y)
\]
Minimum Bayes Risk Decoding

\[ h_\theta(x) = \arg\min_{\hat{y}} \mathbb{E}_{y \sim p_\theta(\cdot|x)}[\ell(\hat{y}, y)] \]

Consider some example loss functions:

The **Hamming loss** corresponds to accuracy and returns the number of incorrect variable assignments:

\[
\ell(\hat{y}, y) = \sum_{i=1}^{V} (1 - \mathbb{I}(\hat{y}_i, y_i))
\]

The MBR decoder is:

\[
\hat{y}_i = h_\theta(x)_i = \arg\max_{\hat{y}_i} p_\theta(\hat{y}_i \mid x)
\]

This decomposes across variables and requires the variable marginals.
Minimum Bayes Risk Decoding

\[ h_\theta(x) = \arg\min_{\hat{y}} \mathbb{E}_{y \sim p_\theta(\cdot|x)}[\ell(\hat{y}, y)] \]

Consider some example loss functions:

The **0-1 loss function** returns 1 only if the two assignments are identical and 0 otherwise:

\[ \ell(\hat{y}, y) = 1 - \mathbb{I}(\hat{y}, y) \]

The MBR decoder is:

\[ h_\theta(x) = \arg\min_{\hat{y}} \sum_y p_\theta(y \mid x)(1 - \mathbb{I}(\hat{y}, y)) \]

\[ = \arg\max_{\hat{y}} p_\theta(\hat{y} \mid x) \]

which is exactly the MAP inference problem!