CS489/698: Intro to ML

Lecture 11: Fundamentals of Deep Learning

focal Systems

UNIVERSITY OF WATERLOO

Outline

- Loss Functions
- Regularization

Outline

- Loss Functions
- Regularization

Supervised Learning

maximize $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$

or minimize

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}).$$

Deriving Cost Functions - Regression

Hidden Layers

$$\boldsymbol{h} = f(\boldsymbol{x}; \boldsymbol{\theta}).$$

Output Layer

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}.$$

Conditional Probability

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I}).$$

Maximum Likelihood

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X}; \boldsymbol{\theta}).$$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log P(\boldsymbol{y}^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}).$$

$$\sum_{i=1}^{m} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^{m} \frac{\left\| \hat{y}^{(i)} - y^{(i)} \right\|^2}{2\sigma^2},$$

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2,$$

focal Systems

UNIVERSITY OF WATERLOO

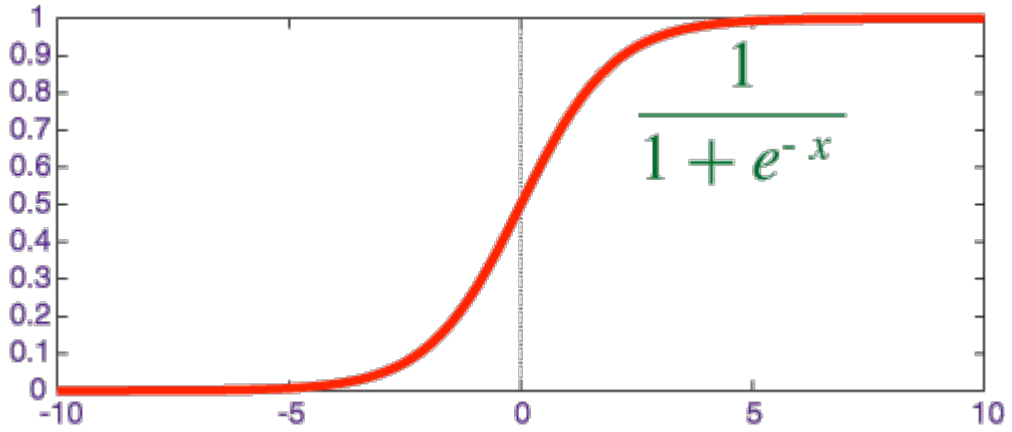
Mean Squared Error Properties

- Can have exploding gradients
- Many shallow valleys, tricky to optimize
- **Trick**: Normalize outputs to be between 0-1 so gradient cannot be greater than 1
- Not desired

Binary Classification

y = 0 or 1

$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right)$$



Binary Crossentropy

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Derive with Binomial instead of Gaussian

focal Systems

UNIVERSITY OF WATERLOO

Multiclass Classification

Output Layer

$$\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}, \text{ on } z_i = \log \tilde{P}(y = i \mid \boldsymbol{x}).$$

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \quad \text{Crossentropy)}$$

$$J(\theta) = -\sum y_i ln(\hat{y}_i)$$

Derived from multinomial distribution

Investigating Softmax

Log Softmax

No gradient saturation

$$\log \operatorname{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j).$$

Numerically stable softr

$$\log \sum_j \exp(z_j) \approx \max_j z_j = z_i$$

$$\operatorname{softmax}(\boldsymbol{z}) = \operatorname{softmax}(\boldsymbol{z} - \max_i z_i).$$

Loss Function Cheat Sheet

Output Types

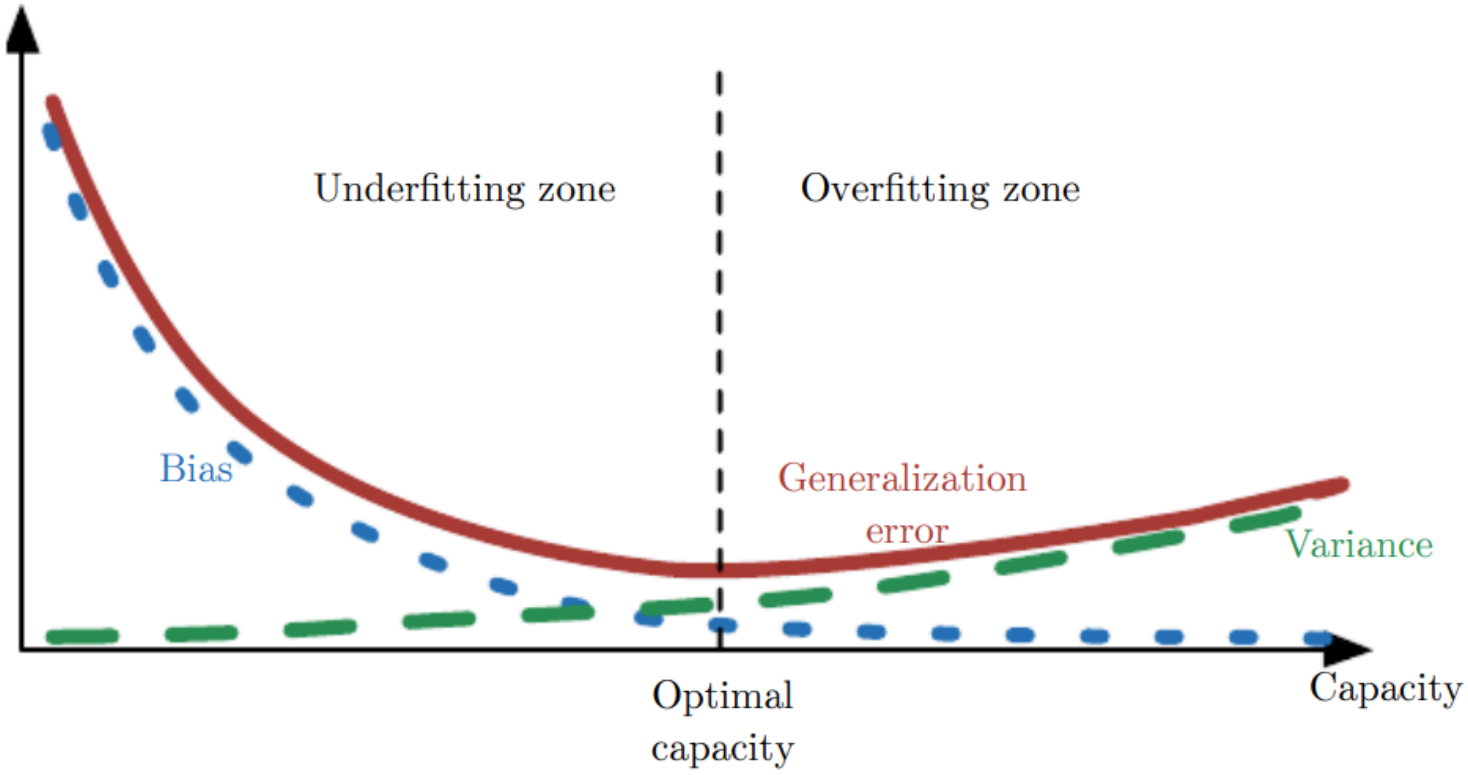
Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	See part III: GAN, VAE, FVBN	Various

Outline

- Loss Functions

- Regularization

Bias vs Variance



focal Systems

UNIVERSITY OF WATERLOO

Regularization

- Any modification made to the learning algorithm that is intended to reduce generalization but not its training error
- Take high capacity model, increase bias in a “good” direction and reduce variance

Weight Penalties

Cost function is of the form:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

veights

- $\Omega(\boldsymbol{\theta})$ is a constant giving the penalty a weight

Can also be applied to activations

α

L2 Penalty

- $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$
- Favours many low weights.
- It wants small changes in input to have minimal effect on output.

if w is a vector of size 4, then it favours [0.25, 0.25, 0.25, 0.25] vs [0,0,0,1]

L2 Gradient

Loss Fn: $$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

Gradient:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

Update:

Pushing u

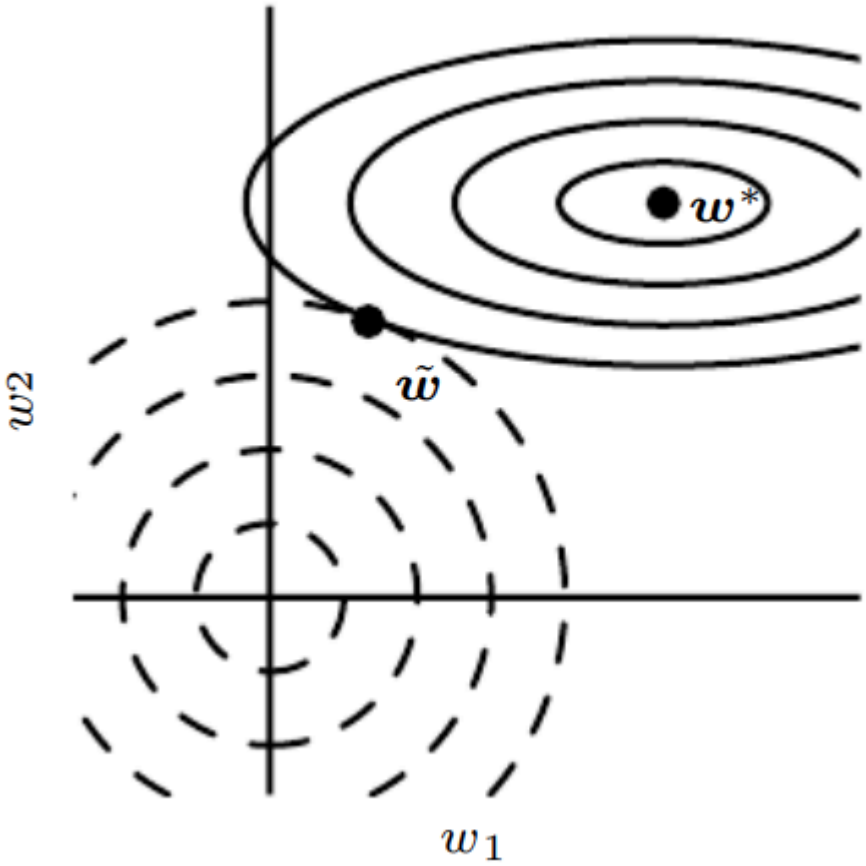
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})\right).$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

focal Systems

UNIVERSITY OF WATERLOO

Effect of L2 Regularization



focal Systems

UNIVERSITY OF WATERLOO

L1 Penalty

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|$$

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

Equal Gradient everywhere!

Encourages sparsity

Data Augmentation

datasets

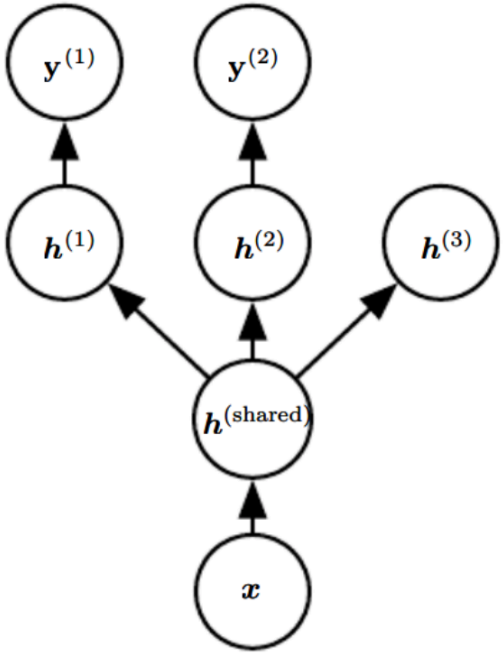
Injecting Noise in Outputs

- Many datasets can have wrong labels
- Maximizing can be harmful when y is wrong
- **Label Smoothing:** if label has prob of being right, set true class to and false class to $1-\epsilon$
- dramatically reduce cost of extremely incor $\frac{\epsilon}{k-1}$ examples $1-\epsilon$

focal Systems UNIVERSITY OF WATERLOO

Multi-Task Training

- Train a model to do multiple tasks from the same input
- Better generalizations

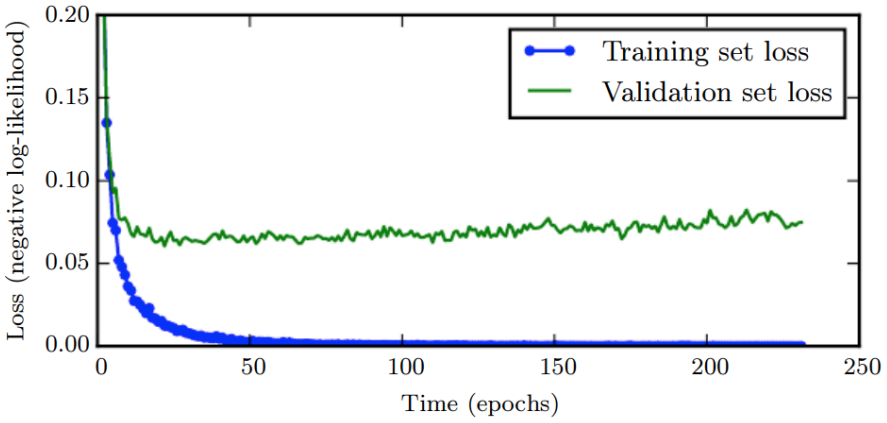


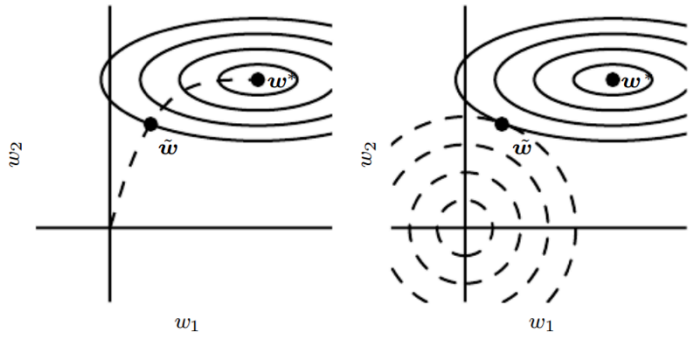
focal Systems

UNIVERSITY OF WATERLOO

Early Stopping

- Run learning for a fixed number of iterations
- Every few iterations, check validation performance
- Save the model with best validation performance
- Implicit hyperparameter search across time

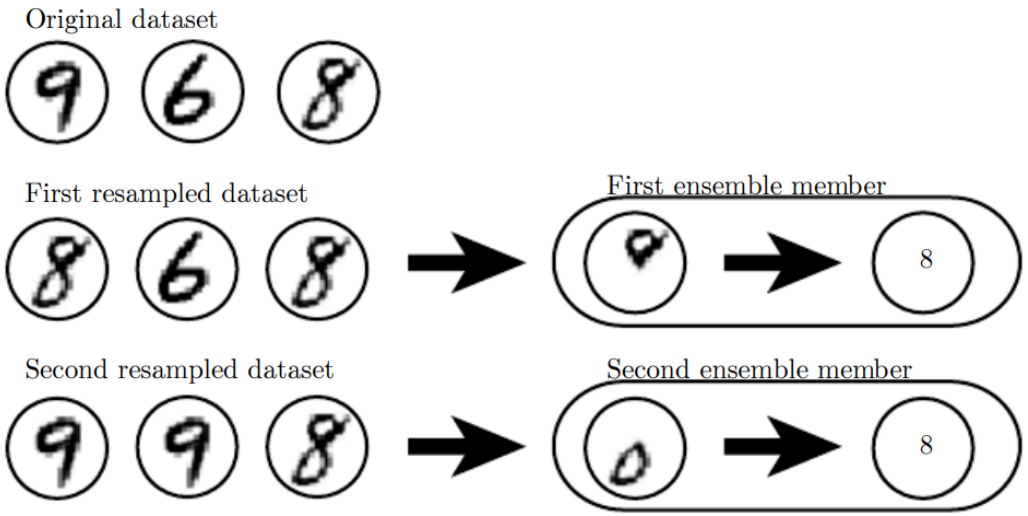




focal Systems UNIVERSITY OF WATERLOO

Ensembles

- Train multiple models and have them weighted vote
- Resample dataset, Re-initialize weights
- Multiple models will have some uncorrelated errors
- Free 2%
- Expensive
- Recycle models by checkpointing and increasing LR

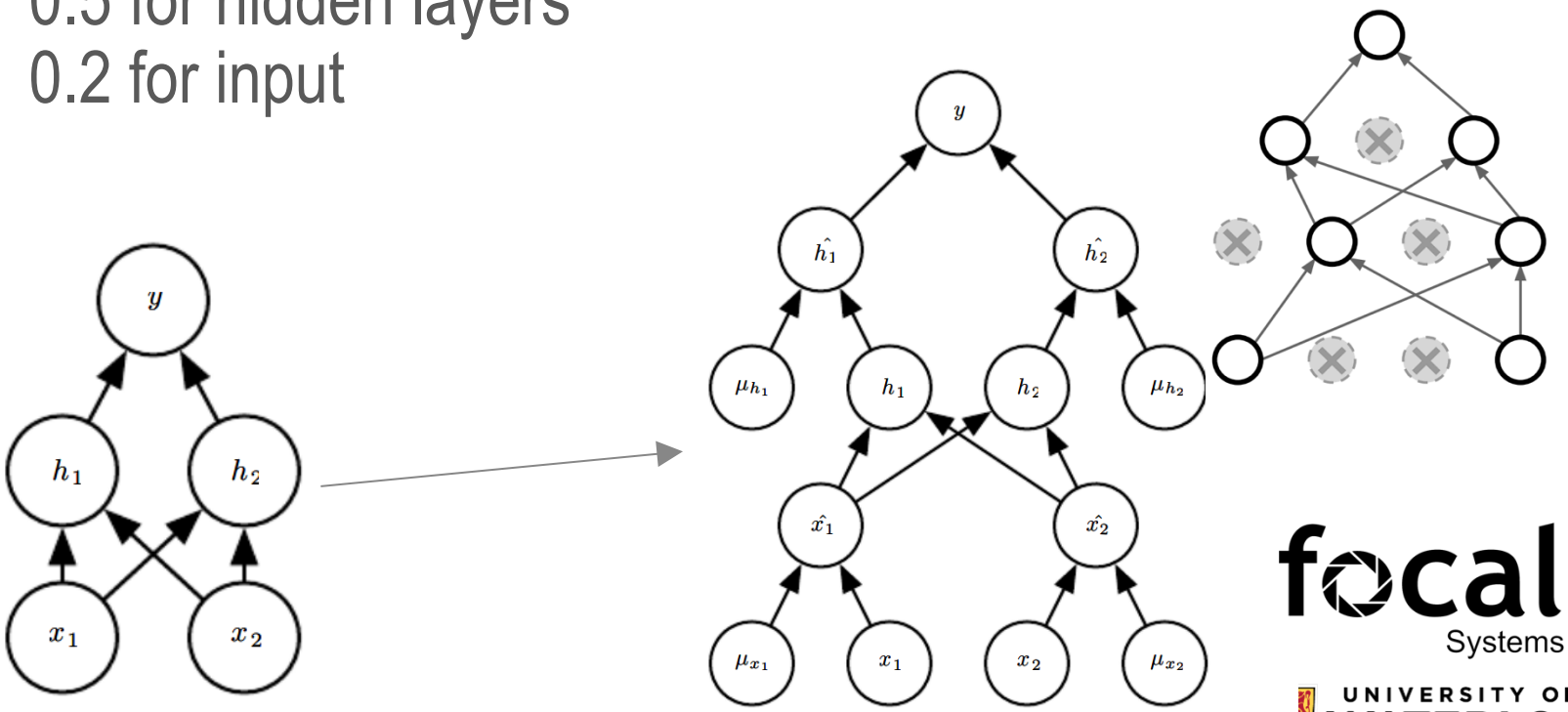


focal Systems

UNIVERSITY OF WATERLOO

Dropout - Train Time

- Multiply activations by binary mask
- **Dropout Probability:** The chance an activation will be turned off.
  - 0.5 for hidden layers
  - 0.2 for input



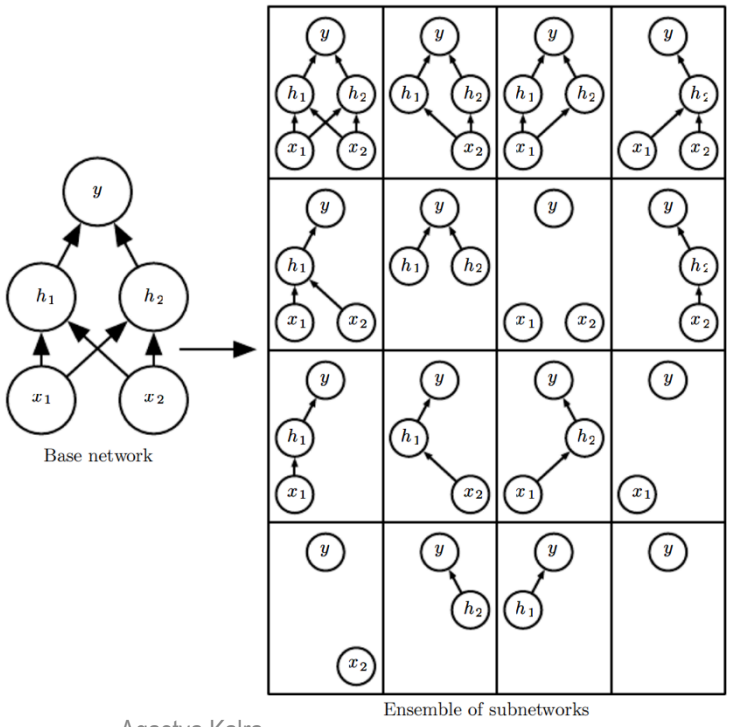
Dropout Test Time

- No Binary Mask
- Multiply all weights by dropout probability
  - Approximately taking the average of all subnetworks
- Inverted dropout divides by dropout probability at train time instead

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$$

Dropout - Why??

- Simultaneously Trains and ensemble of sub-networks
- Forces Redundancy in Model



Dropout - Code

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Dropout Summary

drop in forward pass

scale at test time

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

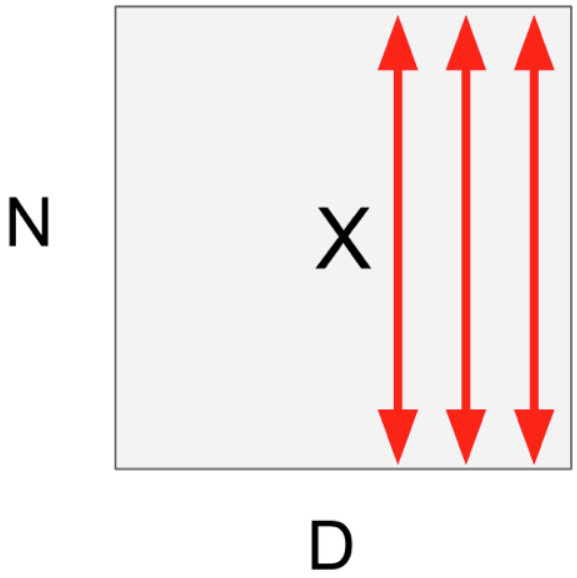
Inverse Dropout - Code

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

test time is unchanged!

Batch Normalization

"you want unit gaussian activations?
just make them so."

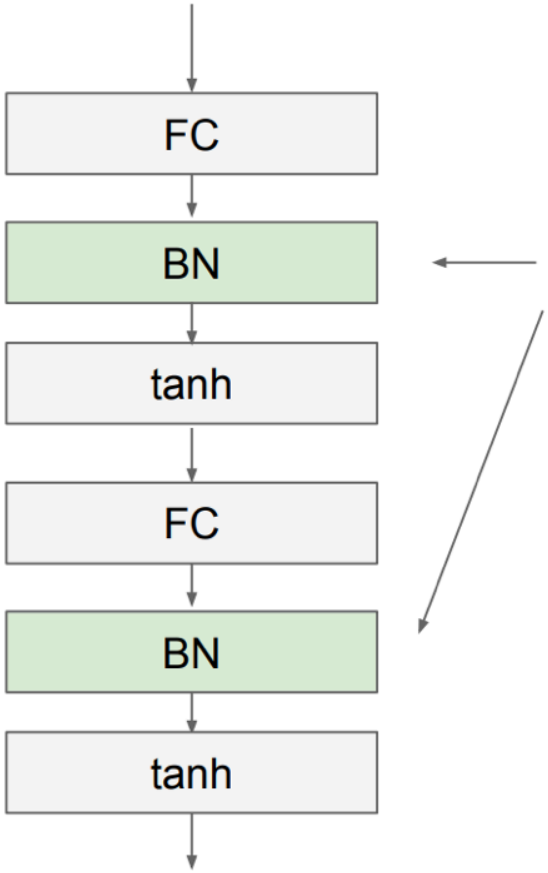


1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Batch Normalization

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

Regularization Summary

- **L2 Weight Decay:** Forces low weights, helps convergence (must do)
- **L1 Weight Decay:** Forces sparsity (optional)
- **Data Augmentation:** Increases dataset size (large gains, must do)
- **Noisy Outputs:** Reduce effect of bad examples (optional)
- **Multi-Task training:** Effective, but hard
- **Early Stopping:** Prevents overfitting (must do)

Regularization Summary cont'd

- **Ensembles:** Free 2%, but expensive
- **Dropout:** Cheap ensemble, (optional)
- **Batch Normalization:** Better gradient flow, (must do)

Deep Learning for Retail