



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

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

or minimize

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

Deriving Cost Functions - Regression

Hidden Layers

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

Output Layer

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

Conditional Probability

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

Maximum Likelihood

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

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

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

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

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

Agastya Kalra

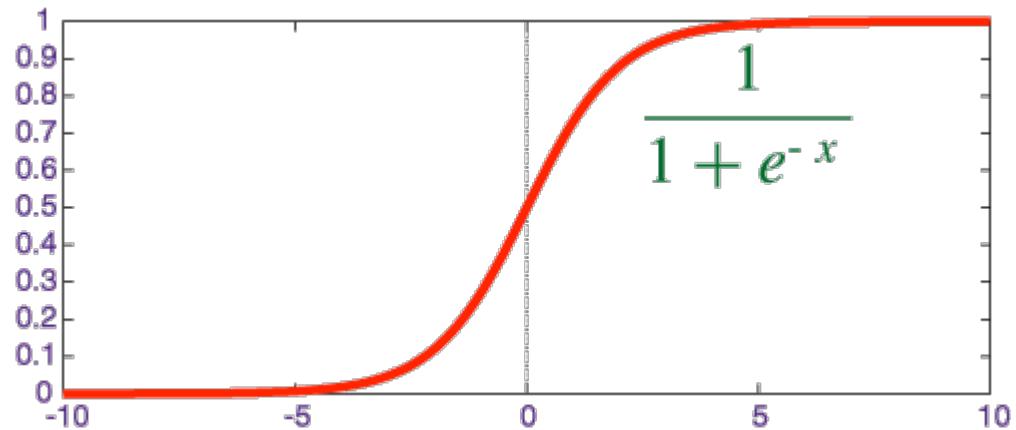
Mean Squared Error Properties

- 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

$y = 0 \text{ or } 1$

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



Binary Crossentropy

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

Derive with Binomial instead of Gaussian

Multiclass Classification

Output Layer

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

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

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

focal
Systems

Investigating Softmax

Log Softmax

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

No gradient saturation

Numerically stable softmax: $\log \sum_j \exp(z_j) \approx \max_j z_j = z_i$

$$\text{softmax}(\mathbf{z}) = \text{softmax}(\mathbf{z} - \max_i z_i).$$

Loss Function Cheat Sheet

Output Types

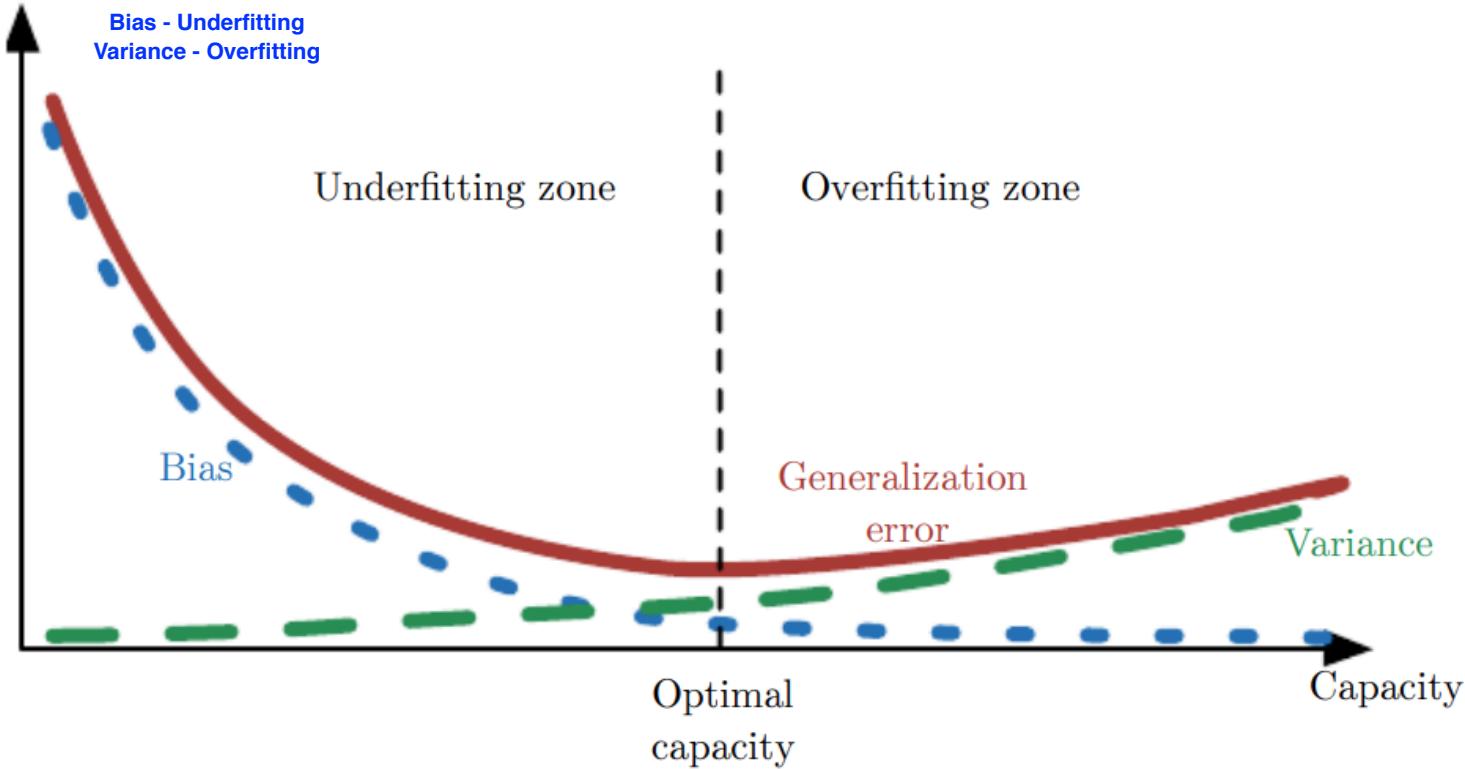
Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous 10/31/17	Arbitrary	See part III: GAN, VAE, FVBN	Various



Outline

- Loss Functions
- Regularization

Bias vs Variance



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

generalization error (also known as the out-of-sample error[1]) is a measure of how accurately an algorithm is able to predict outcome values for previously unseen data

Weight Penalties

Cost function is of the form:

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = J(\theta; \mathbf{X}, \mathbf{y}) + \alpha \Omega(\theta) \quad \text{weights}$$

- $\Omega(\theta)$ is some penalty of the weights constant giving the penalty a weight
- Can also be applied to activations

α

L2 Penalty

- $\Omega(\theta) = \frac{1}{2} \|w\|_2^2$
- It favours mainly 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: $\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} + J(\mathbf{w}; \mathbf{X}, \mathbf{y}),$

Gradient:

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

Update:

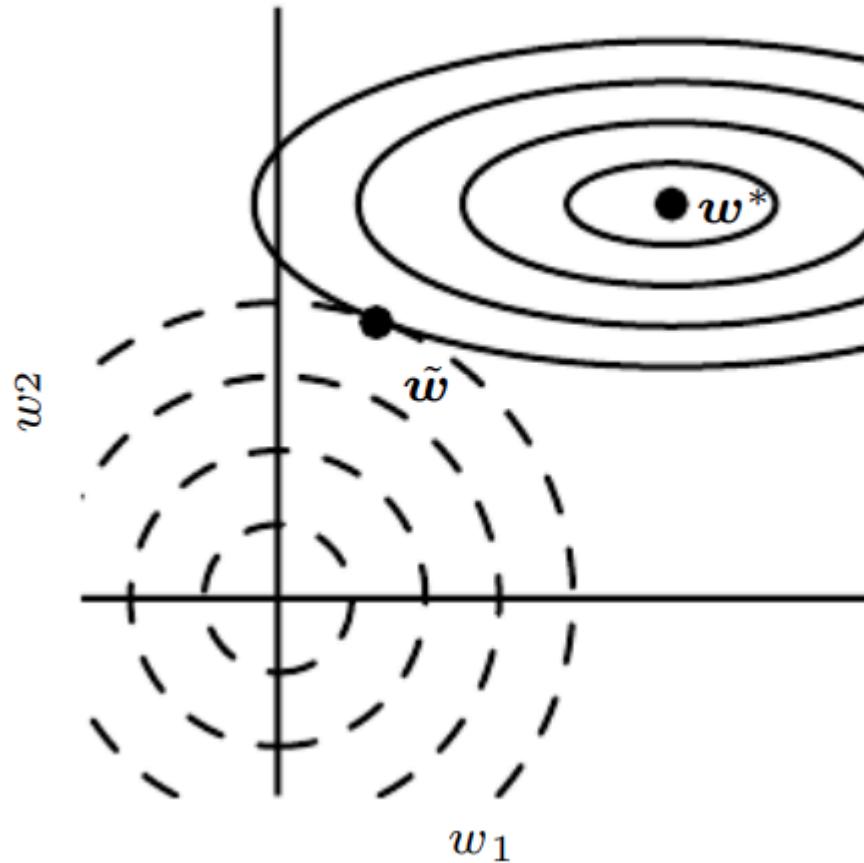
$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon (\alpha \mathbf{w} + \nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y})).$$

Pushing ϵ :

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



Effect of L2 Regularization



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



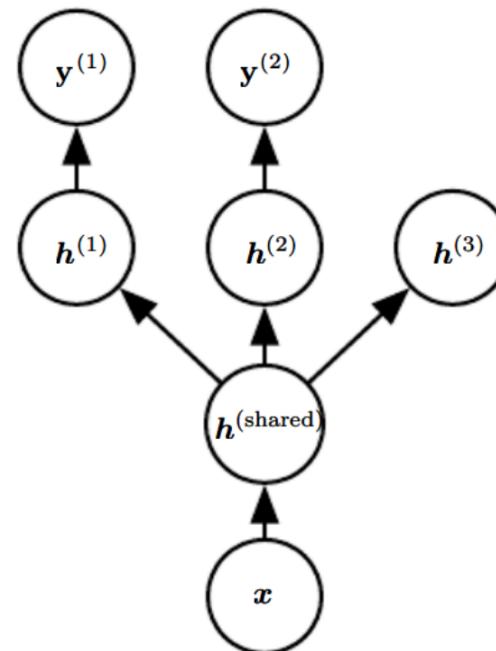
datasets

Injecting Noise in Outputs

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

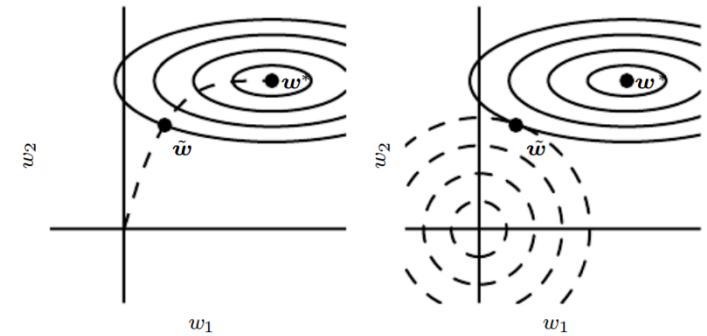
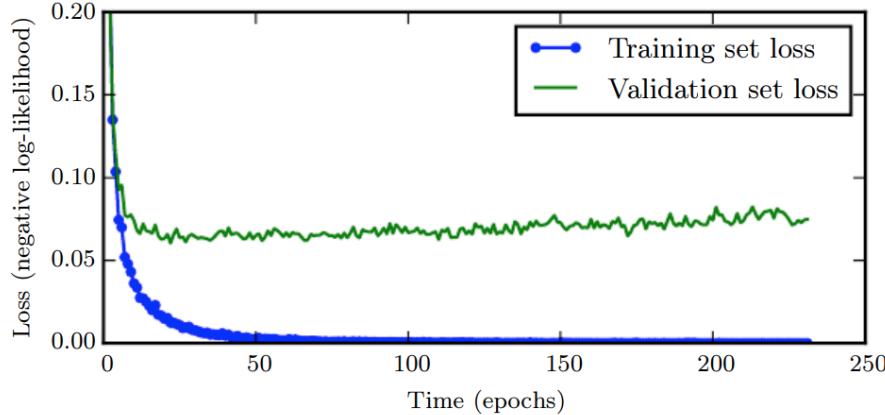
Multi-Task Training

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



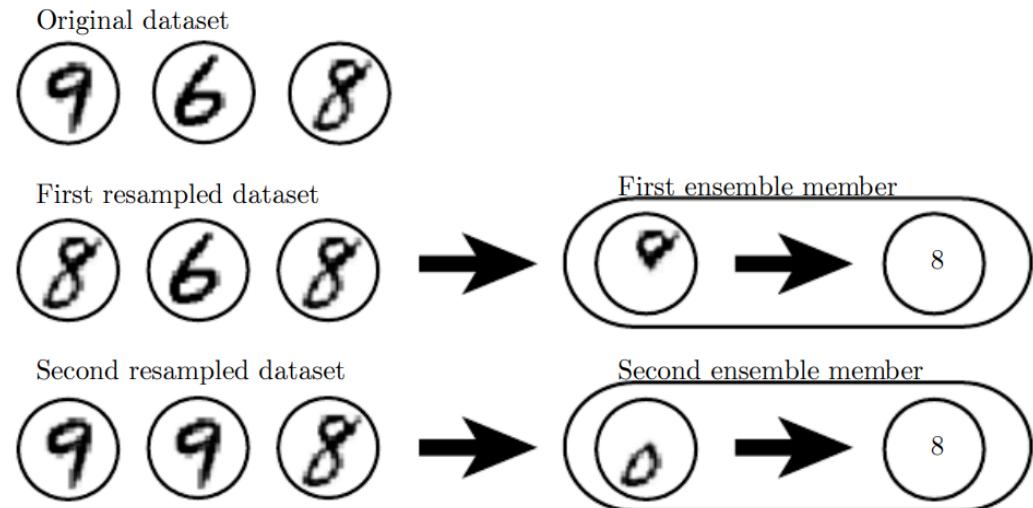
Early Stopping

- 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



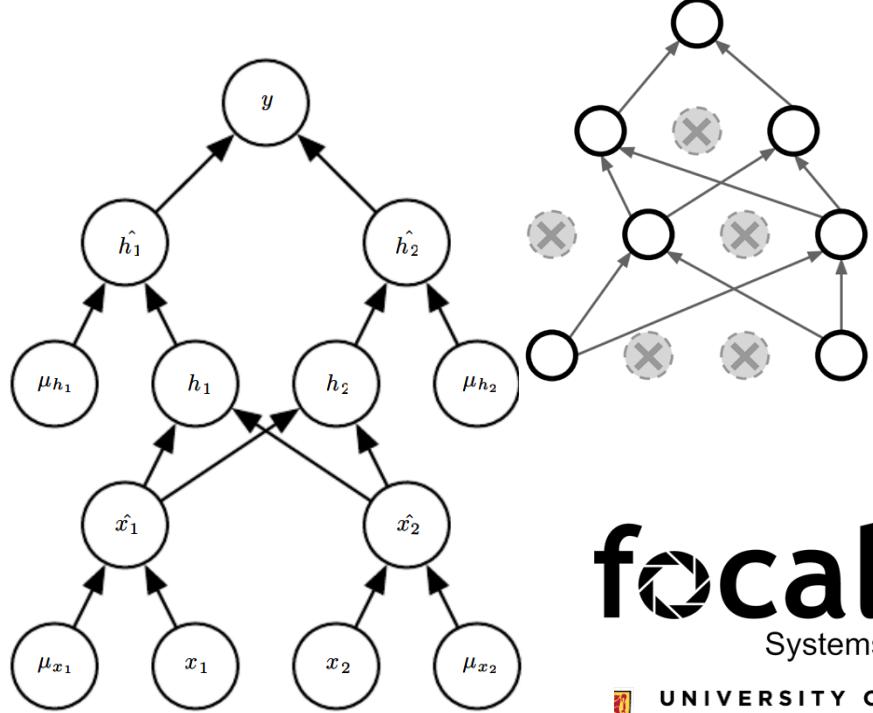
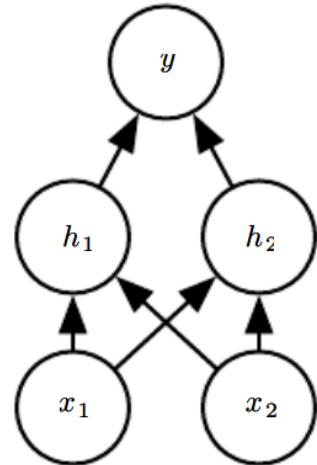
Ensembles

- 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



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



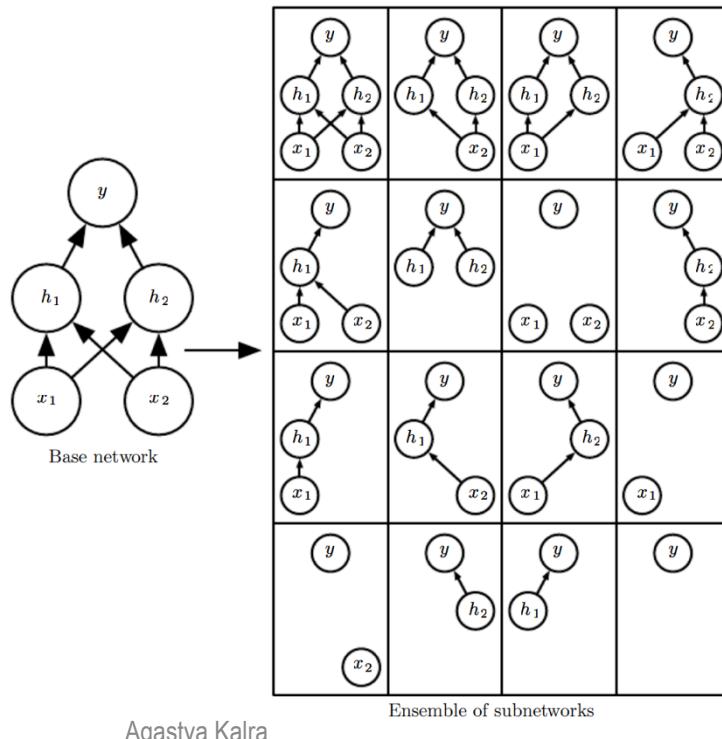
Dropout Test Time

- 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

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

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



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)

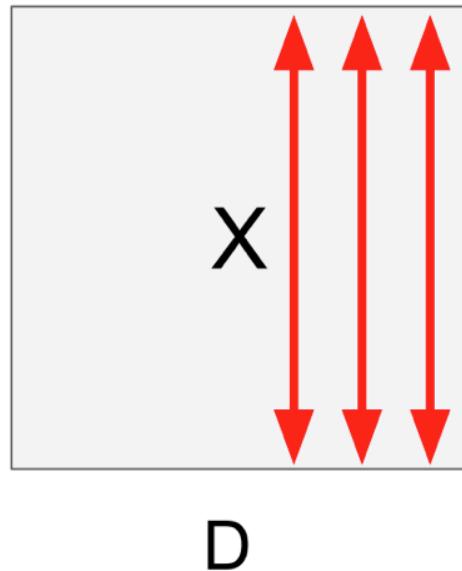
def predict(X):
    # ensembled forward pass
    H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    out = np.dot(W3, H2) + b3
```

test time is unchanged!



Batch Normalization

“you want unit gaussian activations?
just make them so.”



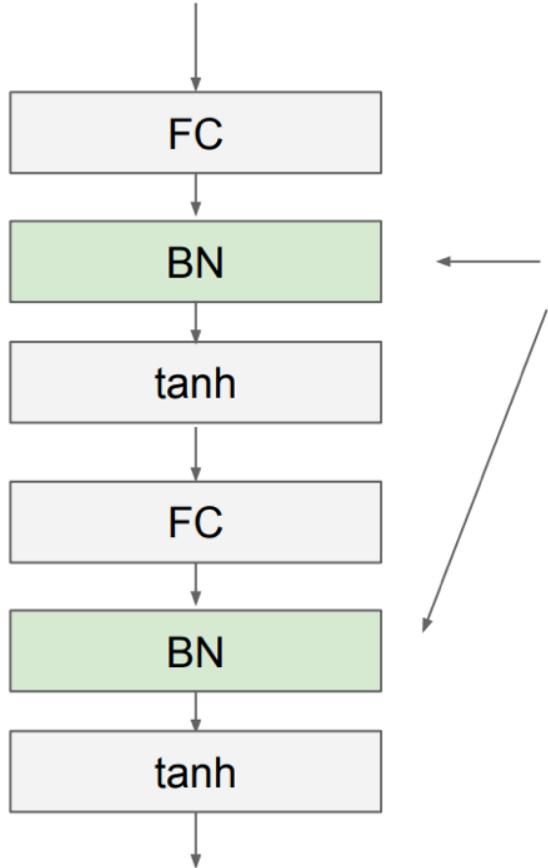
1. compute the empirical mean and variance independently for each dimension.

2. Normalize

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



Batch Normalization



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

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

Batch Normalization

Normalize:

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

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

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

Note, the network can learn:

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

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

to recover the identity mapping.



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

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

- 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



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

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

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

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

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

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