

Fall 2025 CS189/289A Final Exam Reference Sheet

PyTorch Optimizers

<u>torch.optim.RMSprop</u> params: (iterable) lr: (float, default=1e-2) alpha: (float, default=0.99) weight_decay: (float, default=0) momentum: (float, default=0)	<u>torch.optim.SGD</u> params: (iterable) lr: (float) momentum: (float, default=0) weight_decay: (float, default=0)
<u>torch.optim.Adam</u> params: (iterable) lr: (float, default=1e-3) betas: (tuple, default=(0.9, 0.999))	<u>torch.optim.AdamW</u> params: (iterable) lr: (float, default=1e-3) betas: (tuple, default=(0.9, 0.999)) weight_decay: (float, default=0.01)
<u>optimizer.zero_grad()</u>	Sets gradients of all parameters to zero.
<u>optimizer.step()</u>	Performs single optimization step.

PyTorch Layers and nn.Module

import torch and import torch.nn as nn. All layers inherit from nn.Module.

<u>nn.Conv1d</u> in_channels: (int) out_channels: (int) kernel_size: (int) stride: (int, default=1) padding: (int, default=0)	<u>nn.Attention</u> embed_dim: (int) num_heads: (int) dropout: (float, default=0.0) bias: True False
<u>nn.Conv2d</u> in_channels: (int) out_channels: (int) kernel_size: (int tuple) stride: (int, default=1) padding: (int, default=0)	<u>nn.MultiheadAttention</u> embed_dim: (int) num_heads: (int) dropout: (float, default=0.0) bias: True False
<u>nn.MaxPool2d</u> kernel_size: (int tuple) stride: (int tuple) padding: (int, default=0)	<u>nn.Dropout</u> p: (float, default=0.5) inplace: True False
<u>nn.AvgPool2d</u> kernel_size: (int tuple) stride: (int tuple) padding: (int, default=0)	<u>nn.LayerNorm</u> normalized_shape: (int tuple)
<u>nn.Linear</u> in_features: (int) out_features: (int) bias: True False	<u>nn.BatchNorm1d</u> num_features: (int)
	<u>nn.BatchNorm2d</u> num_features: (int)
	<u>nn.ModuleList</u> modules: (iterable)
	<u>nn.Sequential</u> *modules: (modules)
<u>module.forward(x)</u>	Defines the forward pass computation.
<u>module(x)</u>	Must be implemented when subclassing nn.Module.
<u>module.parameters()</u>	Returns an iterator over the module's parameters.
<u>module.to(device)</u>	Moves the module to the specified device.
<u>module.train()</u>	Sets the module to training mode.
<u>module.eval()</u>	Sets the module to evaluation mode.

PyTorch Activation Functions

```
import torch.nn as nn
```

<code>nn.ReLU()</code>	Rectified Linear Unit. $f(x) = \max(0, x)$.
<code>nn.Sigmoid()</code>	Sigmoid function. $f(x) = \frac{1}{1+e^{-x}}$.
<code>nn.Tanh()</code>	Hyperbolic Tangent. $f(x) = \tanh(x)$.
<code>nn.Softmax(dim)</code>	Softmax along dimension <code>dim</code> .
<code>nn.LogSoftmax(dim)</code>	LogSoftmax along dimension <code>dim</code> .
<code>nn.GELU()</code>	Gaussian Error Linear Unit.
<code>nn.LeakyReLU(negative_slope=0.01)</code>	Leaky ReLU.

PyTorch Tensor Operations

Let `x`, `y` be tensors. `import torch`.

<code>torch.tensor(data)</code>	Creates a tensor from <code>data</code> .
<code>torch.zeros(size)</code>	Returns tensor filled with zeros.
<code>torch.ones(size)</code>	Returns tensor filled with ones.
<code>torch.randn(size)</code>	Returns tensor with values from normal distribution.
<code>x.shape</code>	Returns tensor shape as <code>torch.Size</code> .
<code>x.view(*shape)</code> <code>x.reshape(*shape)</code>	Reshapes tensor. <code>view</code> requires contiguous memory.
<code>torch.cat(tensors, dim=0)</code>	Concatenates tensors along dimension <code>dim</code> .
<code>torch.stack(tensors, dim=0)</code>	Stacks tensors along new dimension <code>dim</code> .
<code>x.unsqueeze(dim)</code>	Adds dimension of size 1 at position <code>dim</code> .
<code>x.squeeze(dim=None)</code>	Removes dimensions of size 1. If <code>dim</code> specified, only removes that dimension.
<code>x.transpose(dim0, dim1)</code>	Swaps dimensions <code>dim0</code> and <code>dim1</code> .
<code>torch.matmul(x, y)</code> <code>x @ y</code>	Matrix multiplication. Handles broadcasting.
<code>x.backward()</code>	Computes gradients via automatic differentiation. Requires <code>requires_grad=True</code> .
<code>x.grad</code>	Access gradients after <code>backward()</code> .
<code>x.detach()</code>	Returns new tensor detached from computation graph.
<code>torch.no_grad()</code>	Context manager to disable gradient computation.

PyTorch Loss Functions

`import torch.nn as nn`. Loss functions are callable modules.

Loss Function	Description
<code>nn.MSELoss()</code>	Mean Squared Error
<code>nn.CrossEntropyLoss()</code>	Cross-entropy loss for multi-class classification.
<code>nn.L1Loss()</code>	Mean Absolute Error.

General Regularization Techniques

Ridge (L2 Regularization)

Objective: Minimize $E(\mathbf{w}) + \text{L2 penalty}$.

$$\mathbf{w}_{\text{Ridge}}^* = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$$

LASSO (L1 Regularization)

Objective: Minimize $E(\mathbf{w}) + \text{L1 penalty}$.

$$\mathbf{w}_{\text{LASSO}}^* = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

Bias-Variance Decomposition

$$\mathbb{E}[(t - f_{\mathbf{w}}(\mathbf{x}))^2] = \mathbb{E}[(t - h(\mathbf{x}))^2] + (h(\mathbf{x}) - \mathbb{E}[f_{\mathbf{w}}(\mathbf{x})])^2 + \mathbb{E}[(f_{\mathbf{w}}(\mathbf{x}) - \mathbb{E}[f_{\mathbf{w}}(\mathbf{x})])^2]$$

Sigmoid and Softmax

Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid Derivative

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

Softmax: for $z \in \mathbb{R}^K$, $i \in \{1, \dots, K\}$

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}$$

Optimizers

SGD with Momentum

$$\Delta \mathbf{w}^{(\tau)} = \mu \Delta \mathbf{w}^{(\tau-1)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

Weight Decay

$$\Delta \mathbf{w}^{(\tau)} = \Delta \mathbf{w}_{\text{base}}^{(\tau)} - \lambda \mathbf{w}^{(\tau)}$$

Where $\Delta \mathbf{w}_{\text{base}}^{(\tau)}$ is the update from the base optimizer (SGD, Adam, etc.).

Adam

$$\mathbf{m}^{(\tau)} = \beta_1 \mathbf{m}^{(\tau-1)} + (1 - \beta_1) \nabla E(\mathbf{w}^{(\tau)})$$

$$\mathbf{v}^{(\tau)} = \beta_2 \mathbf{v}^{(\tau-1)} + (1 - \beta_2) (\nabla E(\mathbf{w}^{(\tau)}))^2$$

$$\Delta \mathbf{w}^{(\tau)} = \frac{\hat{\mathbf{m}}^{(\tau)}}{\sqrt{\hat{\mathbf{v}}^{(\tau)} + \epsilon}}$$

Neural Networks & Backpropagation

If a variable v affects the loss L through children nodes $c_1, \dots, c_m$, the gradient is:

$$\frac{\partial L}{\partial v} = \sum_{k=1}^m \frac{\partial L}{\partial c_k} \frac{\partial c_k}{\partial v}$$

Normalization Layers & Dropout

Technique	Description
BatchNorm	Normalizes across the batch dimension.
LayerNorm	Normalizes across the feature dimension.
Dropout	Randomly zero out neurons with probability p to prevent overfitting. Training: $x' = \begin{cases} x & \text{with probability } 1 - p \\ 0 & \text{with probability } p \end{cases}$ Inference: rescale x